Spark user guide

User guide for the Spark software applications

James Hilton, William Swedosh, Lachlan Hetherton, Andrew Sullivan and Mahesh Prakash

EP166699
Aug 2019

Spark version 1.1.2

Citation

Hilton JE, Swedosh W, Hetherton L, Sullivan A and Prakash M (2019) Spark user guide 1.1.2. CSIRO, Australia.

Contents

1 Summary

Spark is a toolkit for simulating the spread of wildfires over terrain. The toolkit consists of a number of modules specifically designed for wildfire spread. These include readers and writers for geospatial data, a computational model to simulate a propagating front, a range of visualisations and tools for analysing the resulting data.

This document provides an overview and user guide for a graphical user interface for Spark, ‘*spark-gui*’ and the command-line Spark server application ‘*spark-batch*’.

OpenCL

Spark requires OpenCL 1.2 to run. This now comes as standard with all Windows and Mac graphics drivers. Please update your graphics driver before installing Spark to ensure that the latest OpenCL version is installed.

2 Introduction

Wildfires are dangerous and destructive phenomena frequently occurring in periodically dry regions around the world. The occurrence of fires is a natural process that has shaped landscapes and ecosystems over time. However, increasing urbanisation is bringing more population into contact with wildfires along urban boundaries. The risk to human life and infrastructure has led to intensive research into the prediction of wildfire behaviour. Such predictions are used for risk reduction planning, impact assessment or operational emergency management in the event of a wildfire.

Figure 1 - Experimental grassfire burn to provide data for new rate-of-spread models.

The physical process governing fires is very complex, involving interactions over a range of spatial and temporal scales. Despite this complexity, success has been achieved in predicting behaviour using empirical models. These models predict the behaviour of a wildfire using a set of relationships between factors driving the fire (Sullivan 2009b). These factors include weather conditions, such as wind and air temperature, as well as fuel and landscape conditions.

These empirical models can be used to predict rate of spread of a wildfire for a set of given conditions. They are fast to evaluate on a computer making them ideal for providing rapid large-scale predictions for the path of a fire. Alternative computer modelling techniques include fully physical models (Sullivan 2009a), which use a set of interconnected equations governing the underlying dynamics of the fire. These models provide great detail in the physical processes of the fire, but are currently unfeasible to compute at the landscape scale required for operational purposes.

The Spark toolkit is a configurable system for predicting the spread of a fire perimeter over a landscape based on empirical rate-of-spread models. Multiple rate-of-spread models can be employed within the framework representing different fuel types. Different parameters and fuel conditions governing the rate-of-spread can be defined by the user. The system supports standard geospatial data types for fuel layers and meteorological conditions. The predicted results can be written to standard geospatial data types or displayed and viewed within the system.

This user guide covers two particular applications of the Spark toolkit. The first, *spark-gui*, is a fully-featured graphical application allowing the user to read in data layers for fuel and weather, compute a predicted fire perimeter and view the result. The second, *spark-batch*, is a command line tool suitable for running as a server application. This server application could, for example, be used for a predictive ensemble of fire simulations based on different conditions.

2.1 Spark and configurability

A key aspect of Spark is configurability. Spark has been designed to handle multiple rate-of-spread models for different fuel types. Spark has also been designed to be compatible with future fire models and new types of fire behaviour.

Instead of pre-set rate-of-spread models, fire behaviour is *programmed* into Spark using a *C* script. These scripts define the behaviour of the fire in terms of user-defined spatial fuel and meteorological layers. Any valid OpenCL *C* code can be used for these scripts, along with a wide range of additional mathematical operations.

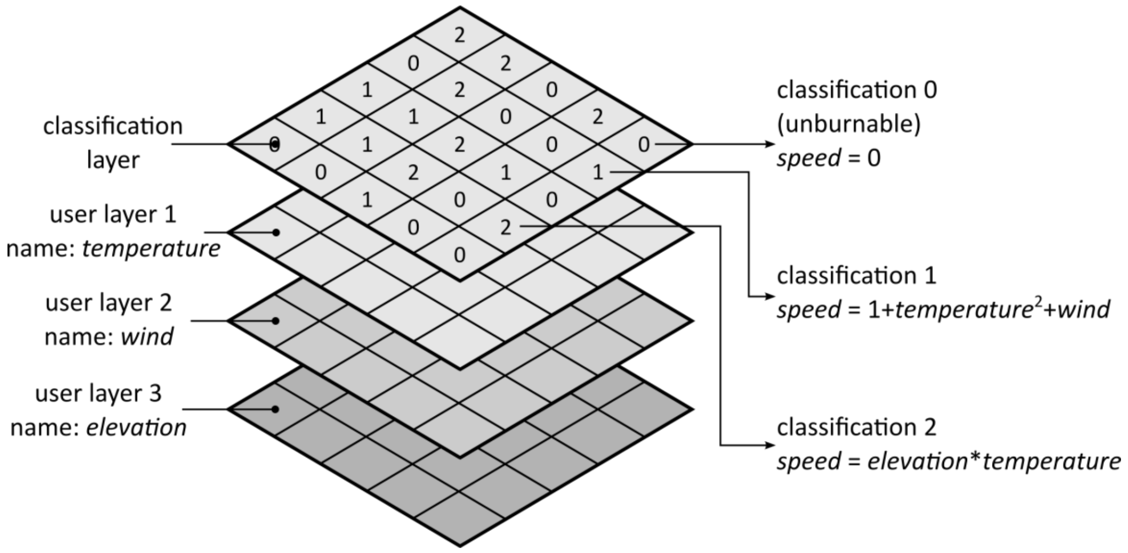

classification layer
user layer 1 name: *temperature*
user layer 2 name: *wind*
user layer 3 name: *elevation*
classification 0 (unburnable) *speed* = 0
classification 1 *speed* = 1+*temperature*^{2}+*wind*
classification 2 *speed* = *elevation***temperature*
2 0 2 1 2 0 1 1 0 2 0 1 2 0 0 0 2 1 1 1 0 0 0 2 0

Figure 2 - Schematic of Spark layers.

Figure 2 shows a very basic example of Spark configuration. The user has four data layers, shown here vertically stacked for illustration. The top data layer is a fuel or land classification, containing a number representing a fuel type. For example, classification *1* may be grassland and classification *2* may be forest. The classification of zero is reserved as un-burnable. The other data layers are the air temperature, the wind data (this is stored a vector but shown as a single layer for illustration) and the land elevation.

The user also requires two fire behaviour models, one for the grassland areas and one for forest areas. The rate of spread in the grassland (classification *1*) is dependent on the temperature and wind (Figure 2, right hand side, middle), whereas the rate of spread for the forest model (classification *2*) is dependent on elevation and temperature (Figure 2, right hand side, bottom).

The chosen rates of spread are entered as formulas in text into Spark. The framework takes care of deciding which cell the fire is in, applying the correct rate of spread and updating the fire perimeter accordingly. Spark also takes care of reading and writing geospatial data layers, alignment and projection of the layers and all spatial and temporal aliasing.

The actual scripts for a particular rate of spread can be very complex. We provide a free source of scripts on our website for the latest fire behaviour models. These can simply be cut and pasted into Spark to provide the desired fire behaviour in different fuel types.

3 Spark-gui application

Spark-gui is an implementation of the Spark toolkit behind a graphical user interface. This general-purpose application allows:

- Up to twelve different types of fuel to be modelled.
- A fire starting condition consisting of a set of points and lines or an ESRI shapefile.
- Either point or gridded input data sets for fuel and weather conditions.
- Output data sets consisting of a raster map of arrival times, a shapefile of isochrones and a map of user-defined variables.

Spark applications are run using an XML project file containing fields for controlling and running the simulation. Three XML sample project files with data are included with the spark-gui application.

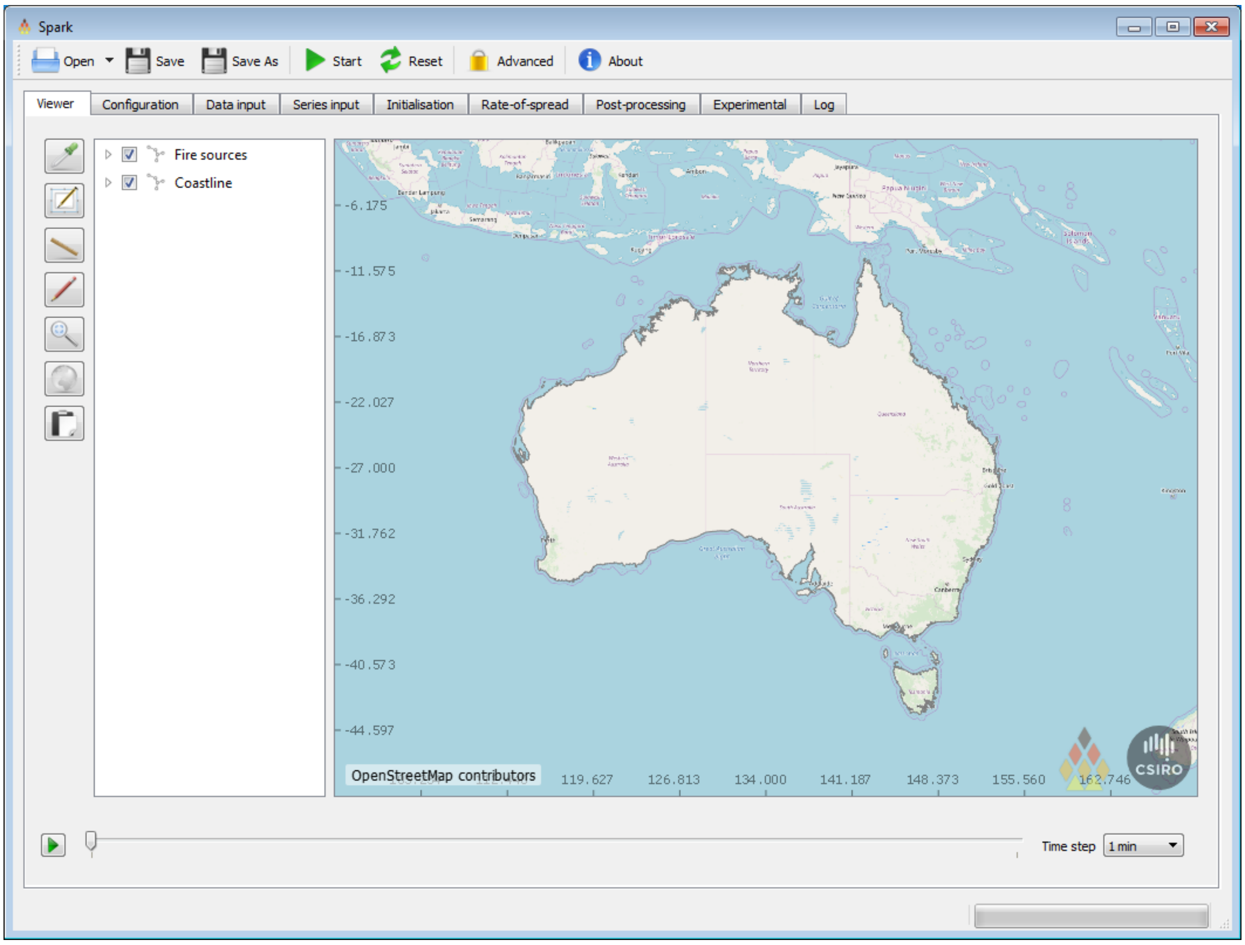


Figure 3 - Spark-gui initial screen.

As an example of the usage of spark-gui we will use the project file *proj1* in the following guide. To open the project, install and run the spark-gui application. The initial screen shows an output map in the viewer window, as shown in Figure 3. The application has a list of geospatial layers in the project on the left-hand side, a preview map on the right-hand side and a set of tabs under the toolbar for various views and configuration options. The toolbar buttons from left to right described in Table 1.

ICON	NAME	DESCRIPTION
🖨	Open	Open project. The arrow to the right of the button is a shortcut for recent projects.
💾	Save	Save the project under the current name in the current location.
💾	Save As	Save the project under a different name or in a different location.
▶	Start	Execute simulation.
🔄	Reset	Reset the project.
🔒	Advanced	Unlock model editing and advanced options.
ℹ	About	Information on the current software version.

Table 1 - Toolbar buttons

To load a project, click on *Open* and navigate to */appdata/spark-gui/proj1*. Select and open the *proj1.xml* file. While the project is loading the status bar will be active. When the project has finished loading the status bar will display 'Ready'.

The project is executed by clicking the Start button. The simulation should take a few seconds to complete. After execution is complete, any output data files are written to their specified locations. The map view will jump the location of these output data sets and display them on the map. The default view displays isochrones of the fire progression, as shown in Figure 4. Different display layers can be toggled on or off using the panel on the left hand side of the map. A legend is displayed on the right for the topmost layer. Execution can be reset at any time by pressing the Reset button. Any edits to the project (or a new project) can be saved as an XML project file using the Save or Save As buttons.

The application working directory

Spark-gui and spark-batch set the directory containing the xml project file as the current directory, so all filenames can be set relative to the xml project file. Alternatively, a full path can be used.

The default directory often has no write access (if installed to Program Files), so the example project directories should be copied somewhere with write access (e.g. Desktop or My Documents on Windows) so these can be run and modified.

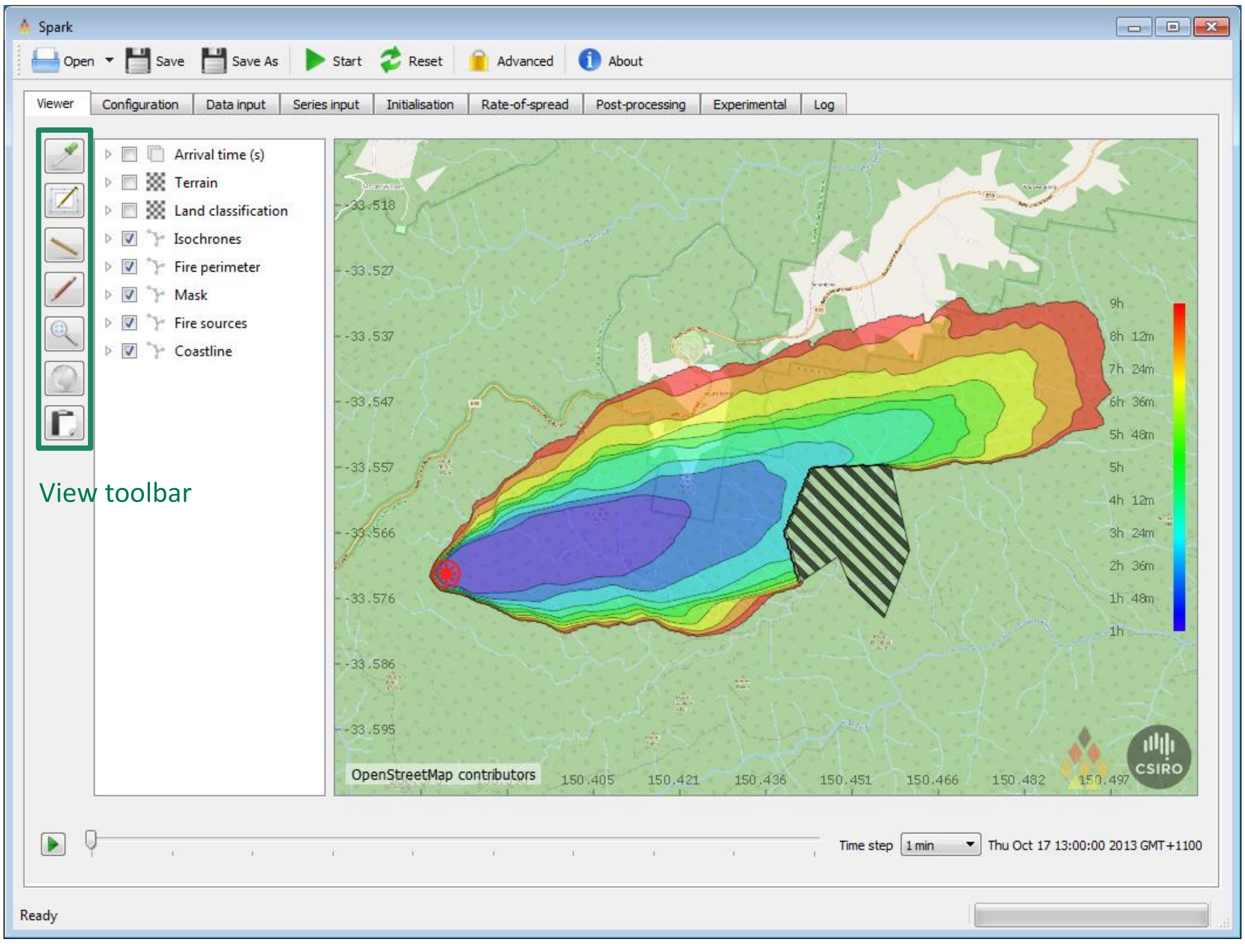


Figure 4 - Spark gui output.

The view toolbar (outlined in Figure 4) allow the map to be interactively queried, moved and captured. The toolbar buttons, from top to bottom, are:

ICON	DESCRIPTION
💉	Inspect topmost layer: values for the point under the layer are displayed in a window in the top left of the map widget.
✎	Select layers: values for selected items are shown under the layer tree.
📏	Measure distance: measures the distance between two points.
✏	Write to layer: used to set fire start points or arrays of points.
🔍	Zoom to selection: zooms to a region selected by clicking and dragging.
●	Zoom to extent: resets view to show all visible layers.
📋	Capture image: captures the current map view and saves to a file.

Table 2 - View toolbar buttons

The latitude and longitude at the current mouse position, as well as the values for all toggled layers is displayed in a table in the upper right corner. These values can be copied to the clipboard by clicking the middle mouse button.

For multi-layer raster data sets, the visible level can be changed using a drop-down menu by clicking on the layer name (marked as 'Currently visible layer' in Figure 5). To use, select the value in the drop-down list and press 'Enter'.

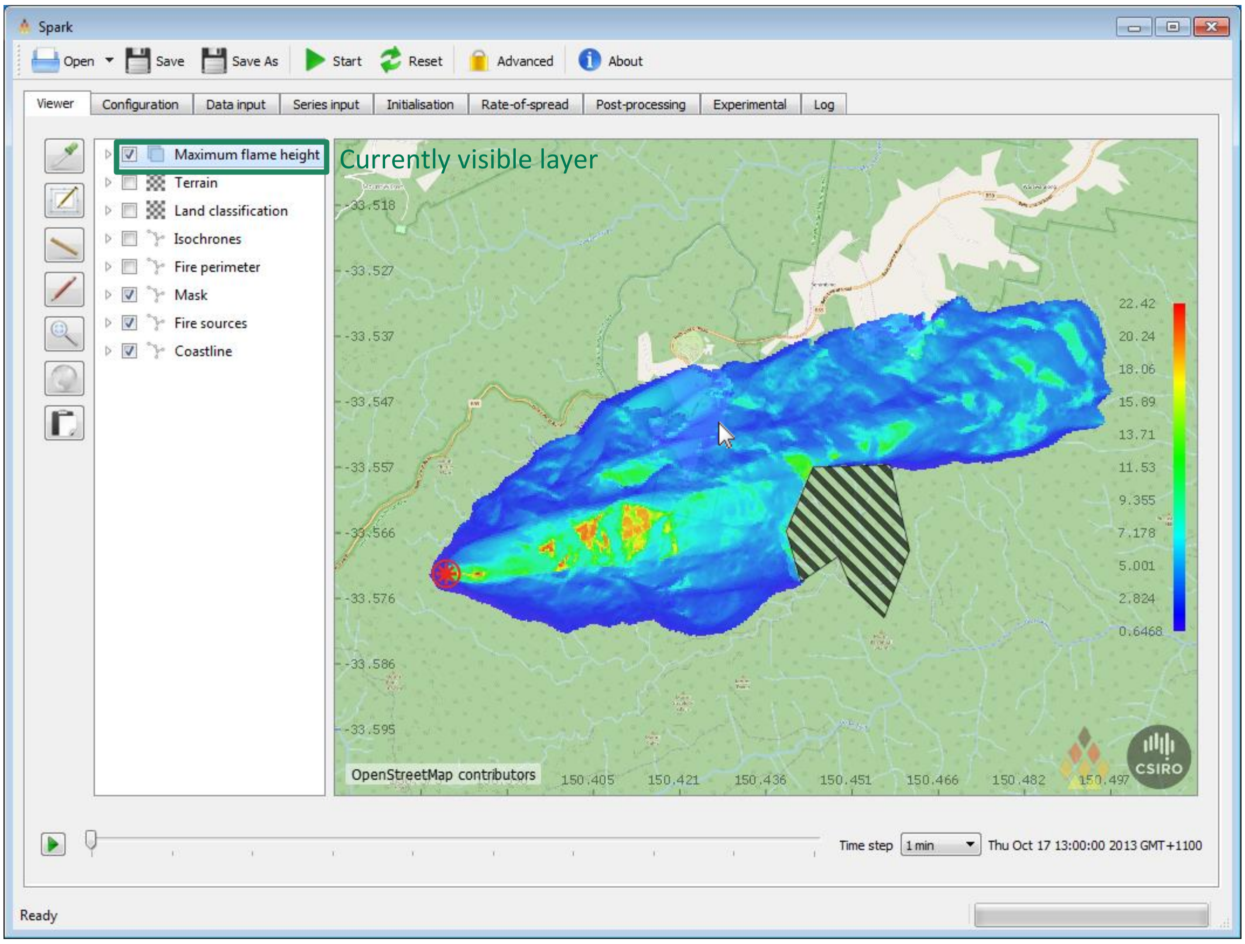


Figure 5 - Spark gui maximum flame height output.

Errors and warnings

If the status bar displays an error, please refer to the log by selecting the 'Log' tab.

If an arrival time output is present, Spark can show the fire perimeter at a specified time. The time is controlled using the time slider bar at the bottom of the gui (outlined in Figure 6). The slider can be manually dragged to a time, or an animation can be toggled using the ‘Play’ button on the left hand side. The drop-down box on the right hand side controls the frequency of the update in simulation time steps. The time stamp on the far right of the slider shows the currently selected time.

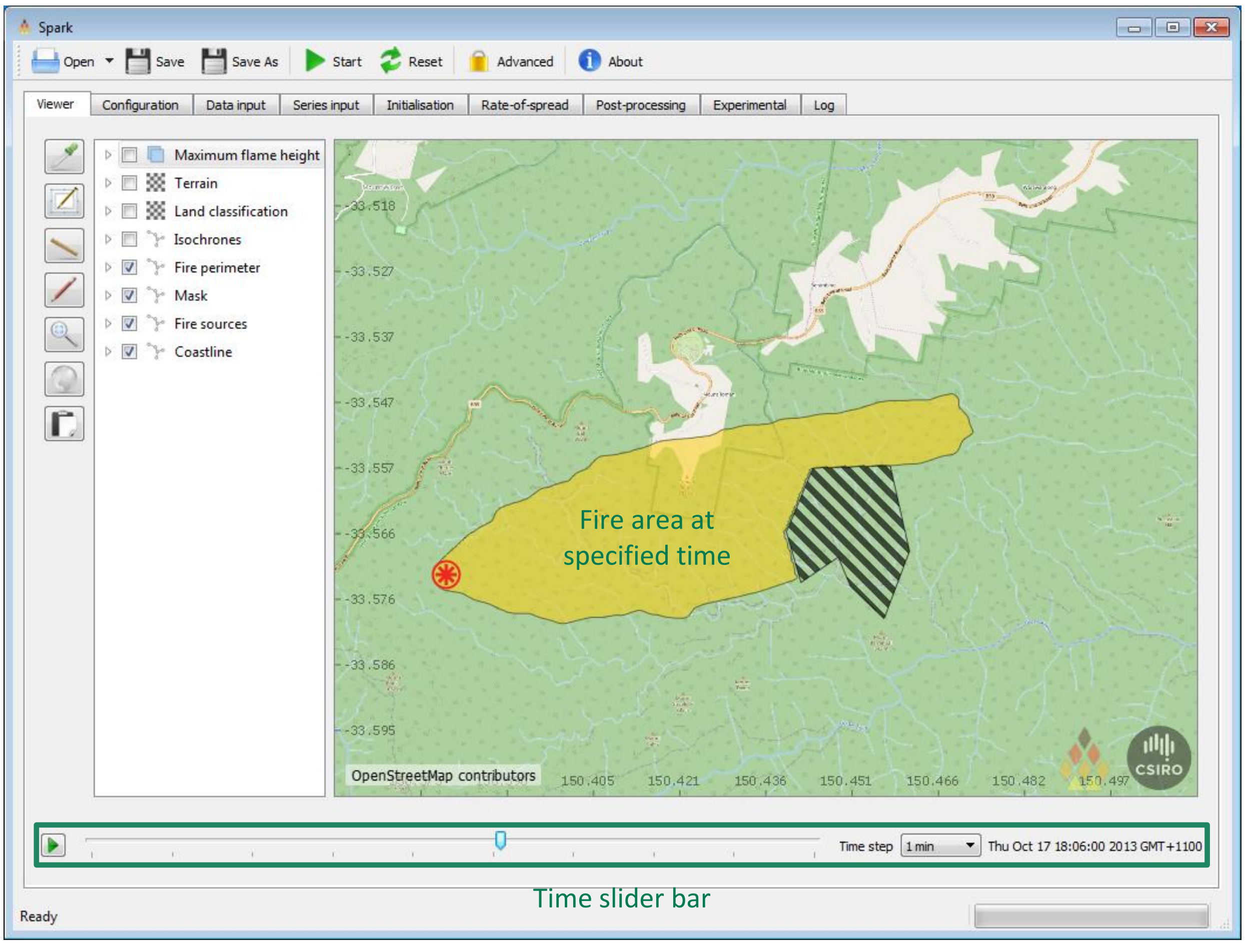


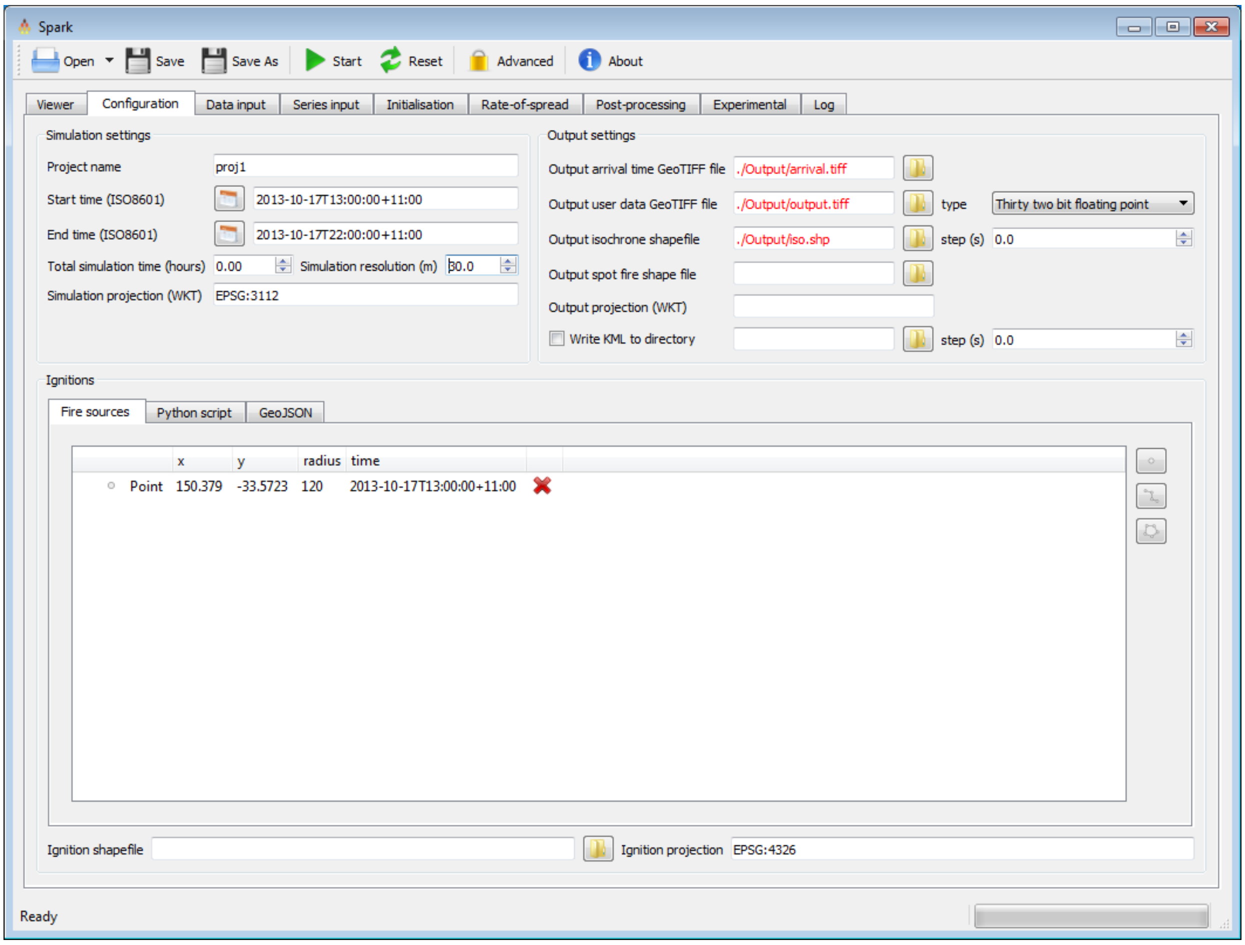
Figure 6 - Spark gui maximum flame height output.

Output comparison

Spark can compare two different fire simulations if a second arrival time raster output is specified in the ‘*Output arrival time comparison file*’ in the ‘*Map display*’ box within the ‘*Data input*’ tab. Colouring and display options for the fire area can also be set within the ‘*Map display*’ box.

3.1 Configuration

After loading the project, the fields within the application will be populated with values defining the simulation. These values include filenames, numbers and scripts. To see the values for *proj1*, select the '*Configuration*' tab. The configuration screen is shown in Figure 7.



Simulation settings

Project name: proj1
Start time (ISO8601): 2013-10-17T13:00:00+11:00
End time (ISO8601): 2013-10-17T22:00:00+11:00
Total simulation time (hours): 0.00
Simulation resolution (m): 30.0
Simulation projection (WKT): EPSG:3112

Output settings

Output arrival time GeoTIFF file: ./Output/arrival.tiff
Output user data GeoTIFF file: ./Output/output.tiff — type: Thirty two bit floating point
Output isochrone shapefile: ./Output/iso.shp — step (s): 0.0
Output spot fire shape file:
Output projection (WKT):
Write KML to directory: — step (s): 0.0

Ignitions

	x	y	radius	time
Point	150.379	-33.5723	120	2013-10-17T13:00:00+11:00

Ignition shapefile:
Ignition projection: EPSG:4326

Ready

Figure 7 - Spark-gui configuration screen

The configuration tab is divided into several groups, covered in the next sections.

Advanced editing

The custom inputs, as well as the model inputs, are locked by default (shown in grey and not editable). To change the custom inputs as well as the initialisation and rate-of-spread-models, click on the 'Advanced' button in the toolbar to unlock editing.

3.1.1 Simulation settings

These fields define the overall parameters for the simulation. All of the parameters must be defined.

NAME	DESCRIPTION	TYPE	XML
Project name	An optional project name. This name is prepended to any output files.	Text	*Project name*
Start time (ISO8601)	The start time of the simulation in the ISO8601 format. Here the start time is set to 1 pm on the 17th October 2013 for the UTC+11 time zone.	Text	*Start time*
End time (ISO8601)	The end time of the simulation in the ISO8601 format. Here the end time is set to 10 pm on the 17th October 2013 for the UTC+11 time zone.	Text	*End time*
Total simulation time (hours)	The length of the simulation in hours. In this example the simulation time is set to zero, meaning the end time string is used.	Number	*Simulation duration hours*
Simulation resolution (m)	The raster resolution of the simulation in metres. All input raster layers are re-sampled to this resolution and all output layers are written at this resolution. Here, the resolution is set to 30 m by 30 m cells.	Number	*Simulation resolution*
Simulation projection (OGC WKT)	The projection used for the simulation in the Open Geospatial Consortium Well-Know-Text standard or EPSG code. All output raster layers and shapefiles are written using this projection. Here the projection is set to the Australian Lambert projection.	Text	*Simulation projection WKT*

Table 3 - Field names for simulation parameters

Start and end time

It is recommended that time zones are specified for time inputs. If these are not specified the simulation will use the local time zone, resulting in different predictions depending on the location the simulation is computed.

Simulation projection

The projection used must be a Cartesian co-ordinate system. Mercator or Lambert projections are recommended. For example, the Lambert projection used in *proj1* is a good general-purpose projection for anywhere in Australia.

3.1.2 Ignitions

These fields define the starting conditions used for the simulation. A starting condition can either be a shapefile representing an initial fire perimeter or a list of points or lines representing starting locations. If both are defined, the shapefile will be used preferentially, and the starting points will not be used.

NAME	DESCRIPTION	TYPE	XML
Fire sources	A table of the current ignition sources, these can be points, lines or polygons	None	*N/A*
Python script	This is a Python script defining the latitude, longitude, radii and start times of each point source. This script must contain the following definitions: • long (point longitude) • lat (point latitude) • radius (point radius) • time (point activation time) These variables are defined as a Python vector and can be generated in any manner within the script. Optionally, the python script can use the *seed* variable (ensemble simulation seed), and the *startDate* variable (start date-time string).	Text	*Start point script*
GeoJSON	A GeoJSON string specifying the ignition conditions. This is created based on (in order of preference) any GeoJSON ignition conditions in the project XML, the Python script and the input shapefile.	Text	*GeoJSON input source*
Ignition shapefile	An ESRI shapefile (.shp) defining an initial fire perimeter. No shapefile is used in this example.	Filename	*Shape file input source*
Start shapefile input projection (OGC WKT)	The shapefile projection in the Open Geospatial Consortium Well-Know-Text standard or EPSG code. No shapefile is used in this example.	Text	*Shape file input projection WKT*

Table 4 - Field names for ignition parameters

Start shapefile

The application currently uses two fields for defining the ESRI shapefile data. These are the shapefile data (.shp) and the projection. The projection is usually stored within a projection (.prj) file, but is not guaranteed to be compatible with the OGC WKT format. To ensure the projection is correctly applied it must be specified in OGC WKT in the shapefile projection field.

3.1.3 Output settings

These fields define the simulation output files.

NAME	DESCRIPTION	TYPE	XML
Output arrival time GeoTIFF file	If this field contains a filename a GeoTIFF containing arrival time within the fire perimeter is written to a file with this name. If the field is empty no file is written.	Filename	*Output raster file*
Output user data GeoTIFF file	If this field contains a filename a GeoTIFF containing the ten user-defined output layers is written to a file with this name. If the field is empty no file is written.	Filename	*Output data raster file*
Output user data GeoTIFF file type	The data type written to the output data GeoTIFF file. The data is converted to this type before being written. Different data types may provide savings in space.	Selection	*Output data raster type*
Output isochrone shapefile	If this field contains a filename an ESRI shapefile of isochrone data is written to a file with this name. The shapefile is written with a projection file containing the simulation projection. If the field is empty no file is written.	Filename	*Output shape file*
Output isochrone shapefile step (s)	The spacing in time between the output isochrones in the shapefile. If this is zero a default value of one hour is used.	Number	*Output isochrone time*
Output spot fire shapefile	If this field contains a filename an ESRI shapefile of spot fire starting locations is written to a file with this name. The shapefile is written with a projection file containing the simulation projection. If the field is empty no file is written.	Filename	*Output spot fire shape file*
Output projection (WKT)	The projection applied to all outputs in the Open Geospatial Consortium Well-Know-Text standard or EPSG code. If this is blank the outputs are written using the simulation projection.	Text	*Output projection WKT*
Write KML to directory	Write Keyhole Markup Language to specified directory for visualisation in Google Earth and Google Maps.	Filename	*KML directory*
Write KML to directory step (s)	The spacing in time between the output frames in the kml directory. If this is zero a default value of one hour is used.	Number	*KML time step*

Table 5 - Field names for output options

The output files are compatible with all common GIS processing tools, such as QGIS (Figure 8).

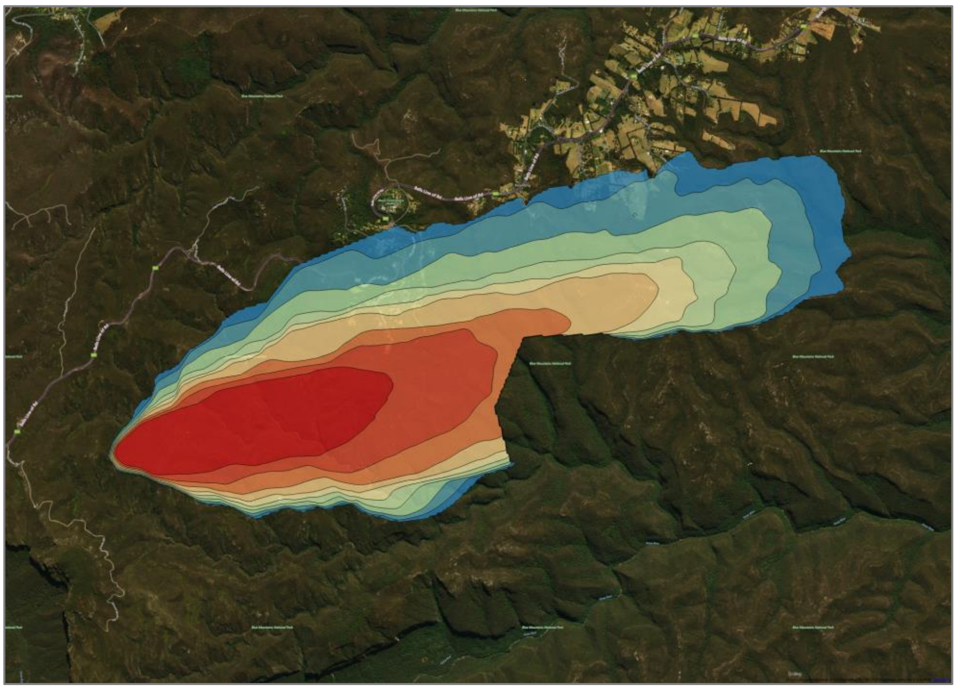


Figure 8 - Output GeoTIFF (shaded by arrival time), and isochrones (black lines).

3.2 Data input

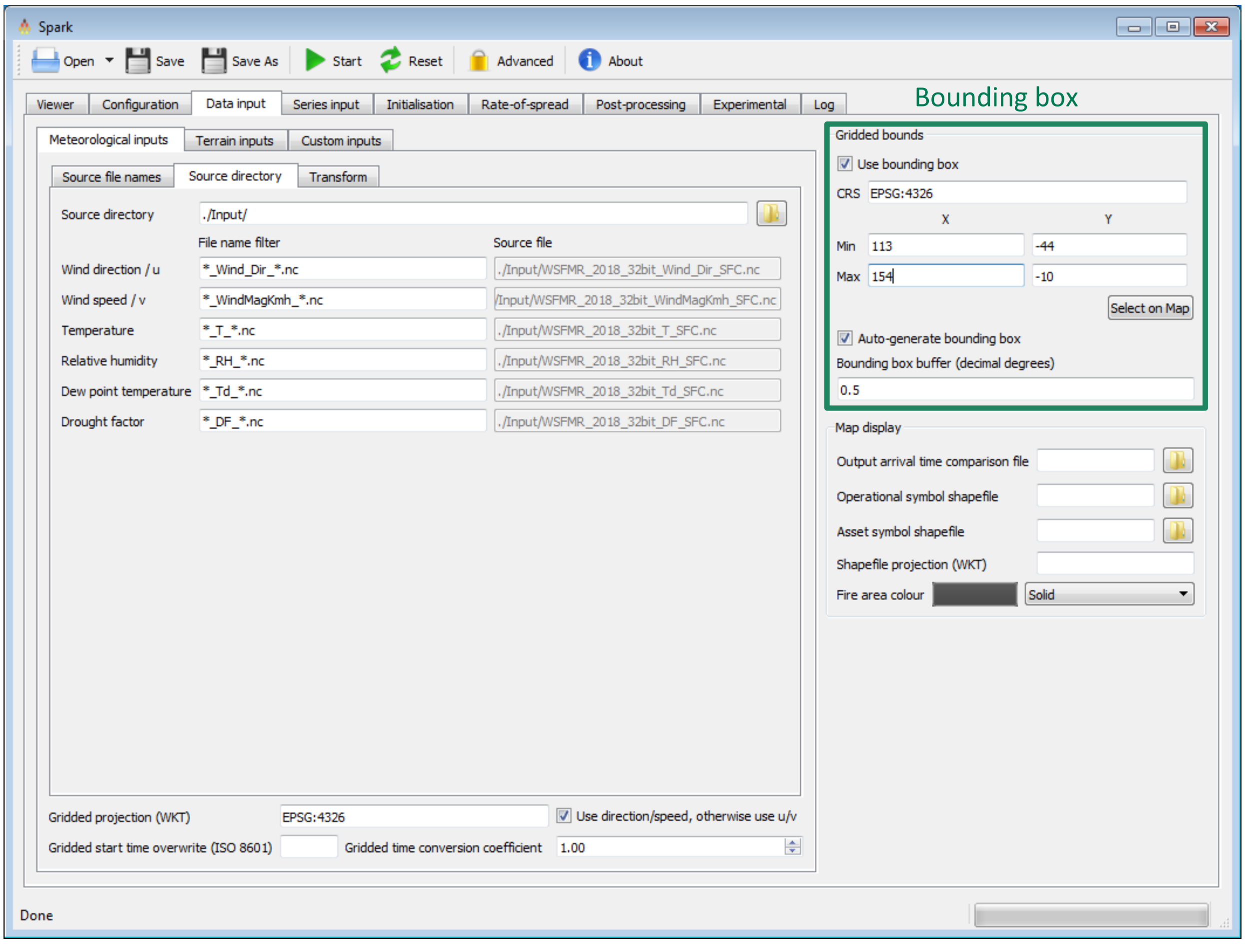
Spark supports two or three-dimensional raster layers (two spatial plus one temporal) and handles all temporal interpolation. Raster data layers are read in using the '*Data input*' tab (Figure 9).

Gridded input can be restricted to a sub-region, if required. This can be applied using the bounding box configuration (Figure 9, outlined), which specifies bounds in longitude and latitude. The bounding box can also be selected on an interactive map using the '*Select on Map*' button. To select a region using this tool, hold *shift* and drag on the map to specify the region.

If required, this sub-region can be automatically generated for a region around the starting points or shapefile. To use this automatic feature, check the '*Use bounding box*' and the '*Auto-generate bounding box*' options. A bounding box is generated in a square region with bounds given by the starting longitude and latitude plus or minus the value in '*Bounding box buffer*'. The value in '*Bounding box buffer*' must be in decimal degrees.

Automatic bounding boxes

Use of automatic bounding boxes can greatly increase the simulation speed when using large geospatial input layers as only small sub-regions of the layers are read in and processed. A reasonable value of the bounding box buffer is 0.4-0.6 decimal degrees for large scale day long simulations. If the automatic option is selected the values in the bounding box input are ignored.



Source directory ./Input/

	File name filter	Source file
Wind direction / u	*_Wind_Dir_*.nc	./Input/WSFMR_2018_32bit_Wind_Dir_SFC.nc
Wind speed / v	*_WindMagKmh_*.nc	/Input/WSFMR_2018_32bit_WindMagKmh_SFC.nc
Temperature	*_T_*.nc	./Input/WSFMR_2018_32bit_T_SFC.nc
Relative humidity	*_RH_*.nc	./Input/WSFMR_2018_32bit_RH_SFC.nc
Dew point temperature	*_Td_*.nc	./Input/WSFMR_2018_32bit_Td_SFC.nc
Drought factor	*_DF_*.nc	./Input/WSFMR_2018_32bit_DF_SFC.nc

Gridded projection (WKT) EPSG:4326
☑ Use direction/speed, otherwise use u/v
Gridded start time overwrite (ISO 8601)
Gridded time conversion coefficient 1.00

Gridded bounds
☑ Use bounding box
CRS EPSG:4326

	X	Y
Min	113	-44
Max	154	-10

☑ Auto-generate bounding box
Bounding box buffer (decimal degrees)
0.5

Map display
Output arrival time comparison file
Operational symbol shapefile
Asset symbol shapefile
Shapefile projection (WKT)
Fire area colour Solid

Figure 9 - Spark-gui gridded input screen; meteorological input section.

3.2.1 Meteorological inputs

Meteorological input layers are configured in the '*Meteorological inputs*' tab. There are six meteorological input layers which can be defined as individual files or as a set of files in a directory. The geospatial projection, along with options to control the time conversion, are specified for all meteorological input layers using the following options:

NAME	DESCRIPTION	TYPE	XML
Gridded projection (OGC WKT)	The raster layer projection for all meteorological raster layers in the Open Geospatial Consortium Well-Know-Text standard or EPSG code.	Text	*Gridded/Layer projection WKT*
Use direction/speed	If set the input wind raster layers are direction and speed. Otherwise the wind raster layers are the *x* and *y* components of the wind vector.	Checkbox	*Gridded/Wind/Use direction speed*
Gridded start time overwrite (ISO 8601)	An optional start date and time in the ISO8601 format to use instead of the values stored in the meteorological raster layers.	Text	*Gridded/Start time overwrite*
Gridded time conversion coefficient	A multiple to apply to the time-step value for the meteorological raster layers.	Number	*Gridded/Time conversion coefficient*

Table 6 - Meteorological input layer projections and time conversion parameters

To set individual files, select the ‘*Source file names*’ tab. The inputs for this tab are:

NAME	DESCRIPTION	TYPE	XML
Wind direction	A raster layer specifying the wind direction in degrees within each cell.	Filename	*Gridded/Wind/Layer direction source file*
Wind speed	A raster layer specifying the wind speed within each cell.	Filename	*Gridded/Wind/Layer magnitude source file*
Temperature	A raster layer specifying the air temperature within each cell.	Filename	*Gridded/Temperature/Layer source file*
Relative humidity	A raster layer specifying the relative humidity (%) within each cell.	Filename	*Gridded/Relative humidity/Layer source file*
Dew point temperature	A raster layer specifying the dew point temperature within each cell.	Filename	*Gridded/Dew point temperature/Layer source file*
Drought factor	A raster layer specifying the drought factor within each cell.	Filename	*Gridded/Drought factor/Layer source file*

Table 7 - Field names for meteorological inputs

Gridded file names can be alternatively generated by selecting a directory and specifying a name filter for each required file. In the example shown in Figure 9 the *Input* directory contains six files containing the text *_Wind_Dir_*, *_Wind_Mag_*, *_T_*, *_RH_*, *_Td_* and *_DF_*. Once the directory and a filter is specified, the *Source file* column will update with the generated file path for verification.

The inputs corresponding to the ‘*Source directory*’ tab are:

NAME	DESCRIPTION	TYPE	XML
Source directory	A directory containing a set of meteorological input files.	Filename	*Gridded/Source directory*
Wind direction	A text filter specifying the wind direction input file.	Filename	*Gridded/Wind/Layer direction source filter*
Wind speed	A text filter specifying the wind speed input file.	Filename	*Gridded/Wind/Layer magnitude source filter*
Temperature	A text filter specifying the air temperature input file.	Filename	*Gridded/Temperature/Layer source filter*
Relative humidity	A text filter specifying the relative humidity (%) input file.	Filename	*Gridded/Relative humidity/Layer source file*
Dew point temperature	A text filter specifying the dew point temperature input file.	Filename	*Gridded/Dew point temperature/Layer source filter*
Drought factor	A text filter specifying the drought factor input file.	Filename	*Gridded/Drought factor/Layer source filter*

Table 8 - Field names for meteorological inputs

Source files

If a directory and file filter is specified, ensure that no name is specified for the file in the ‘Source file names’ tab, otherwise this will take priority.

Once loaded, the gridded meteorological input data can be previewed on the map in the Viewer tab. If the input data has a temporal component, the time used for the input layer preview can be changed using the slider below the map.

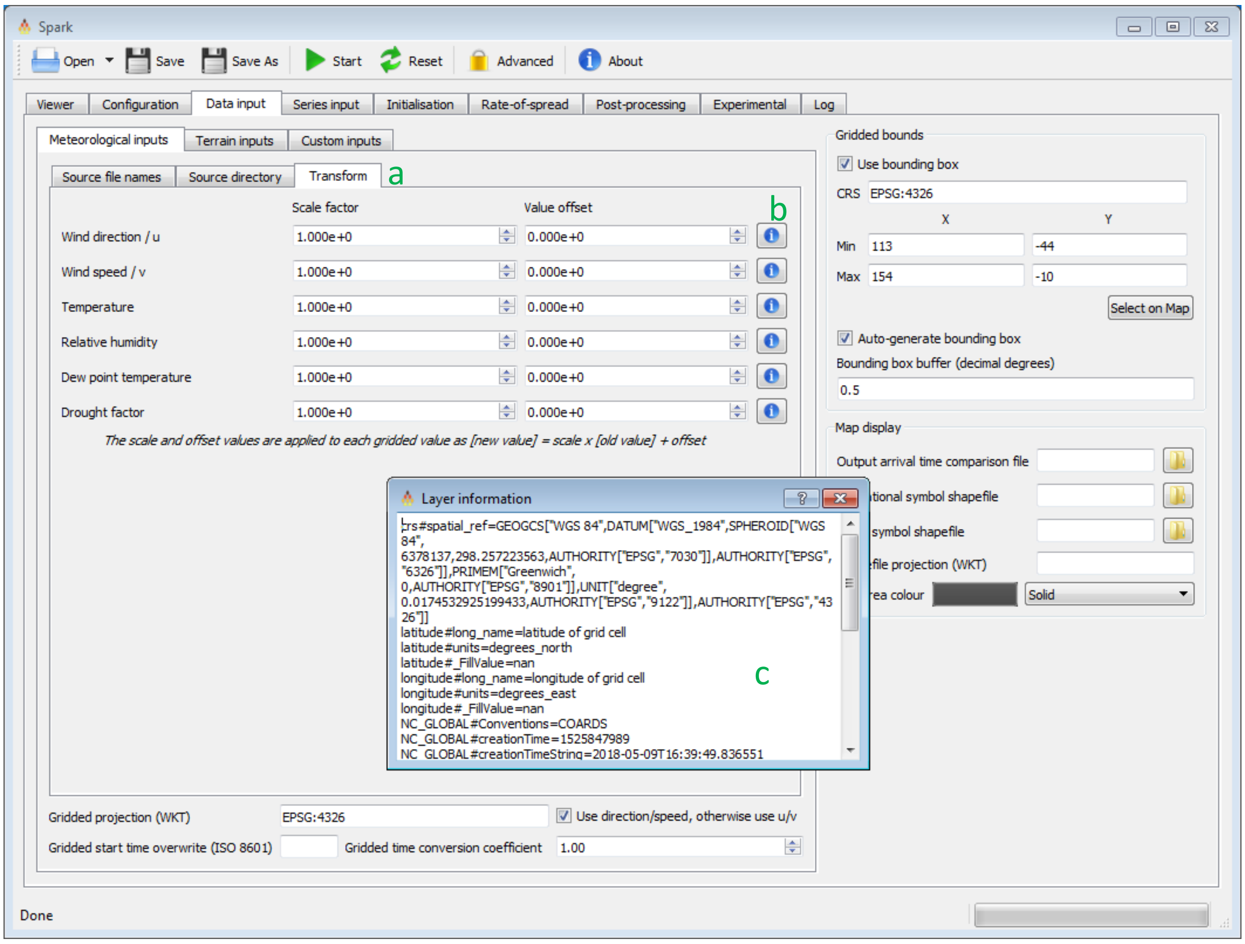
If the input data set is a NetCDF file, the metadata within the file is parsed to generate time information. The parsing process calculates the start time and time step of the input layer. The parsing takes the following steps:

- Search the NetCDF metadata for *time*. If this is not found, stop processing.
- Search the NetCDF metadata for *time#units* and convert to seconds.
- Search for *NETCDF_DIM_time_VALUES*. If this is not found, stop processing.
- Read and convert the *NETCDF_DIM_time_VALUES* array to seconds.
- Check the array for equal time spacing. If the time spacing is unequal, stop processing.
- Calculate the start time of the input layer and the time step.

If the data is not a NetCDF file, or the processing of the metadata fails at any stage, the time step is set to one and the start time is set to zero. In this case, the start time and time step can be manually specified using the *Gridded start time overwrite (ISO 8601)* and *Gridded time conversion coefficient* fields respectively.

Conversion factors can be applied to the gridded data sets using the 'Transform' tab (Figure 10, a). This can be used, for example, to convert data values between different units. A scale and an offset can be applied to the data. The original data is multiplied by the scale factor before the offset value is added.

Information on the data layers can be viewed using the 'information' button at the end of each row (Figure 10, b). This opens a window showing the GDAL layer metadata which usually contains a description of the data units along with other information (Figure 10, c).



	Scale factor	Value offset
Wind direction / u	1.000e+0	0.000e+0
Wind speed / v	1.000e+0	0.000e+0
Temperature	1.000e+0	0.000e+0
Relative humidity	1.000e+0	0.000e+0
Dew point temperature	1.000e+0	0.000e+0
Drought factor	1.000e+0	0.000e+0

The scale and offset values are applied to each gridded value as [new value] = scale x [old value] + offset

Gridded projection (WKT): EPSG:4326
☑ Use direction/speed, otherwise use u/v
Gridded start time overwrite (ISO 8601):
Gridded time conversion coefficient: 1.00

Gridded bounds
☑ Use bounding box
CRS: EPSG:4326

	X	Y
Min	113	-44
Max	154	-10

☑ Auto-generate bounding box
Bounding box buffer (decimal degrees): 0.5

Map display
Output arrival time comparison file
tional symbol shapefile
symbol shapefile
file projection (WKT)
ea colour: Solid

Layer information

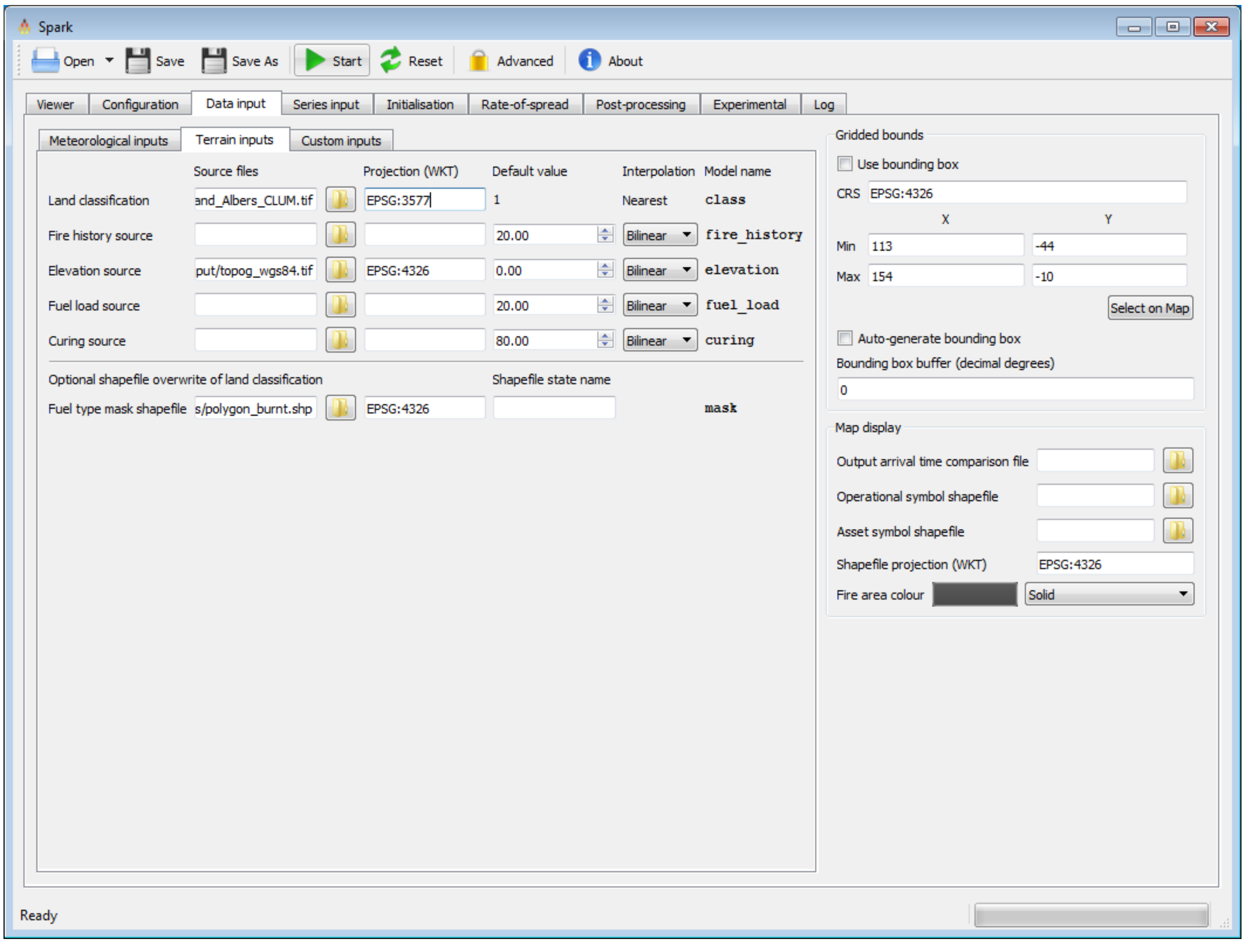
```
crs#spatial_ref=GEOGCS["WGS 84",DATUM["WGS_1984",SPHEROID["WGS
84",
6378137,298.257223563,AUTHORITY["EPSG","7030"]],AUTHORITY["EPSG",
"6326"]],PRIMEM["Greenwich",
0,AUTHORITY["EPSG","8901"]],UNIT["degree",
0.0174532925199433,AUTHORITY["EPSG","9122"]],AUTHORITY["EPSG","43
26"]]
latitude#long_name=latitude of grid cell
latitude#units=degrees_north
latitude#_FillValue=nan
longitude#long_name=longitude of grid cell
longitude#units=degrees_east
longitude#_FillValue=nan
NC_GLOBAL#Conventions=COARDS
NC_GLOBAL#creationTime=1525847989
NC_GLOBAL#creationTimeString=2018-05-09T16:39:49.836551
```

Figure 10 - Spark-gui gridded input screen; meteorological data, data transform.

3.2.2 Terrain inputs

Fuel and topographic input layers are configured in the '*Terrain inputs*' tab (Figure 11). The field values for this tab are given in Table 9.

There are five pre-defined layers within the spark-gui application. These are the land classification, a fire history layer, an elevation layer, a fuel load layer and a curing layer. If any of these are empty, the layer is populated with the value in the '*Default value*' column. The exception to the default value is the fuel classification layer. If this is empty, the classification of the entire simulation is set to one.



	Source files	Projection (WKT)	Default value	Interpolation	Model name
Land classification	and_Albers_CLUM.tif	EPSG:3577	1	Nearest	class
Fire history source			20.00	Bilinear	fire_history
Elevation source	put/topog_wgs84.tif	EPSG:4326	0.00	Bilinear	elevation
Fuel load source			20.00	Bilinear	fuel_load
Curing source			80.00	Bilinear	curing

Optional shapefile overwrite of land classification — Shapefile state name

Fuel type mask shapefile	s/polygon_burnt.shp	EPSG:4326		mask

Gridded bounds

Use bounding box ☐

CRS EPSG:4326

	X	Y
Min	113	-44
Max	154	-10

Auto-generate bounding box ☐

Bounding box buffer (decimal degrees) 0

Map display

Output arrival time comparison file

Operational symbol shapefile

Asset symbol shapefile

Shapefile projection (WKT) EPSG:4326

Fire area colour ■ Solid

Figure 11 - Spark-gui gridded input screen; terrain data.

Each of the layers can be referred to in the initialisation, post-processing and rate-of-spread models by the name given in the '*Model name*' column.

NAME	DESCRIPTION	TYPE	XML
Land classification	A raster layer specifying the land type within each cell.	Filename	*Classification/Layer source file*
Land classification projection	The projection for the fuel classification layer in the OGC WKT standard or EPSG code.	Text	*Classification/Layer projection WKT*
Fuel type mask shapefile	If a shapefile is specified, any areas within the shapefile are set to a value of one within a layer called `mask`. Outside the shapefile the value of `mask` is zero. This can be used to define a firebreak region, or a previously burnt area.	Filename	*Classification/Shape mask source file*
Fuel type mask shapefile projection	The projection for the fuel classification mask in the OGC WKT standard or EPSG code.	Text	*Classification/Shape mask projection WKT*
Fire history source	A raster layer defining a fire history value, such as a date or time since last burn.	Filename	*Fire history/Layer source file*
Fire history projection	The projection for the fire history layer in the OGC WKT standard or EPSG code.	Text	*Fire history/Layer projection WKT*
Fire history interpolation	The interpolation method used for scaling the fire history data. If the data values are integers (such as date concatenations) 'Nearest' should be used.	Selection	*Fire history/Layer interpolation*
Fire history default	The default value for the fire history layer.	Number	*Fire history/Layer default*
Elevation source	A raster layer specifying land elevation with respect to a vertical datum.	Filename	*Elevation/Layer source file*
Elevation projection	The projection for the elevation layer in the OGC WKT standard or EPSG code.	Text	*Elevation/Layer projection WKT*
Elevation default	The default value for the elevation layer.	Number	*Elevation/Layer default*
Elevation interpolation	The interpolation method used for scaling the elevation data.	Selection	*Elevation/Layer interpolation*
Fuel load source	A raster layer specifying fuel load values.	Filename	*Fuel load/Layer source file*
Fuel load projection	The projection for the fuel load layer in the OGC WKT standard or EPSG code.	Text	*Fuel load/Layer projection WKT*
Fuel load default	The default value for the fuel load layer.	Number	*Fuel load/Layer default*
Fuel load interpolation	The interpolation method used for scaling the fuel load data.	Selection	*Fuel load/Layer interpolation*
Curing source	A raster layer specifying curing values.	Filename	*Curing/Layer source file*
Curing projection	The projection for the curing layer in the OGC WKT standard or EPSG code.	Text	*Curing/Layer projection WKT*
Curing default	The default value for the curing layer.	Number	*Curing/Layer default*
Curing interpolation	The interpolation method used for scaling the curing data.	Selection	*Curing/Layer interpolation*

Table 9 - Field names for terrain inputs

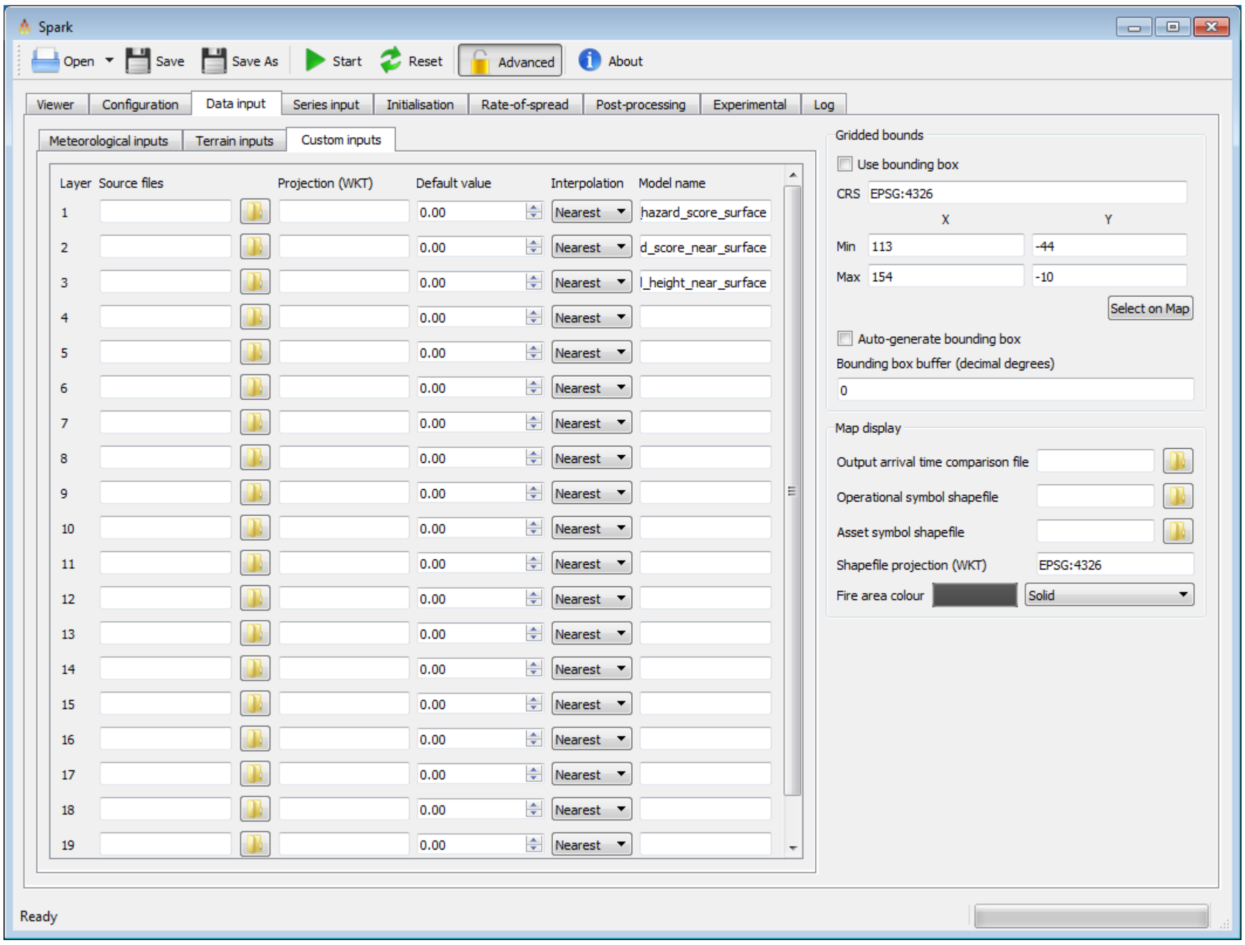
3.2.3 Custom inputs

Rate-of-spread models typically require a range of additional input layers, for example, fuel parameters or wind reduction factors. Up to twenty additional named layers can be specified in the '*Custom inputs*' tab. These layers are initialised to the value given in the '*Default value*' column and are available in all models with the name given in the '*Model name*' column. Optionally, the layers may be read from any GDAL-compatible file. If the source filename is empty, an empty layer initialised to the default value is created.

NAME	DESCRIPTION	TYPE	XML
Source files	The source file for the layer (optional).	Filename	*Custom/Layer file name [1-20]*
Projection (OGC WKT)	The projection for the fuel classification mask in the OGC WKT standard or EPSG code.	Text	*Custom/Layer projection [1-20]*
Model name	The name of a custom layer.	Text	*Custom/Layer name [1-20]*
Default value	The default value for the layer.	Number	*Custom/Layer default [1-20]*
Interpolation	The interpolation method used.	Selection	*Custom/Layer interpolation [1-20]*

Table 10 - Input parameters to define custom layer names

In the *proj1* example, three additional layers are created: *fuel_hazard_score_surface*, *fuel_hazard_score_near_surface* and *fuel_height_near_surface* for the dry eucalypt model.



Layer	Source files	Projection (WKT)	Default value	Interpolation	Model name
1			0.00	Nearest	hazard_score_surface
2			0.00	Nearest	d_score_near_surface
3			0.00	Nearest	l_height_near_surface
4			0.00	Nearest	
5			0.00	Nearest	
6			0.00	Nearest	
7			0.00	Nearest	
8			0.00	Nearest	
9			0.00	Nearest	
10			0.00	Nearest	
11			0.00	Nearest	
12			0.00	Nearest	
13			0.00	Nearest	
14			0.00	Nearest	
15			0.00	Nearest	
16			0.00	Nearest	
17			0.00	Nearest	
18			0.00	Nearest	
19			0.00	Nearest	

Gridded bounds

Use bounding box

CRS EPSG:4326

	X	Y
Min	113	-44
Max	154	-10

Select on Map

Auto-generate bounding box

Bounding box buffer (decimal degrees) 0

Map display

Output arrival time comparison file

Operational symbol shapefile

Asset symbol shapefile

Shapefile projection (WKT) EPSG:4326

Fire area colour ■ Solid

Ready

Figure 12 - Spark-gui gridded input screen; custom input layers.

3.3 Series Input

The applications can use either gridded input data sets (spatial raster maps of parameters) or series inputs (a time series representing the change in a particular parameter). The example project *proj1* uses series input for weather conditions, which can be defined and previewed in the '*Series input*' tab. This window is shown in Figure 13.

The fields in the series input window are:

NAME	DESCRIPTION	TYPE	XML
Script file	A filename which can be used within the Python script. A variable is created within the Python script called '*fileName*' containing the contents of this field.	Filename	*Series/Script*
Time zone	A time zone string which can be used within the Python script. A variable is created within the Python script called '*timeZone*' containing the contents of this field.	Text	*Series/Time zone*
Wind variation speed distribution	If non-zero a random variation is imposed on the wind speed. The wind speed from the series is used as a mean value and a random value is drawn from a normal distribution with the specified width around this mean value.	Number	*Series/Wind speed stdev*
Wind variation direction distribution	If non-zero a random variation is imposed on the wind direction. The wind direction from the series is used as a mean value and a random value is drawn from a normal distribution with the specified width around this mean value. The value is wrapped between 0-360 degrees.	Number	*Series/Wind bearing stdev*
Time series Python script definition	A Python script defining vectors used for time series. This python script also has access to *seed* and *count* variables.	Text	*Series/Source file*

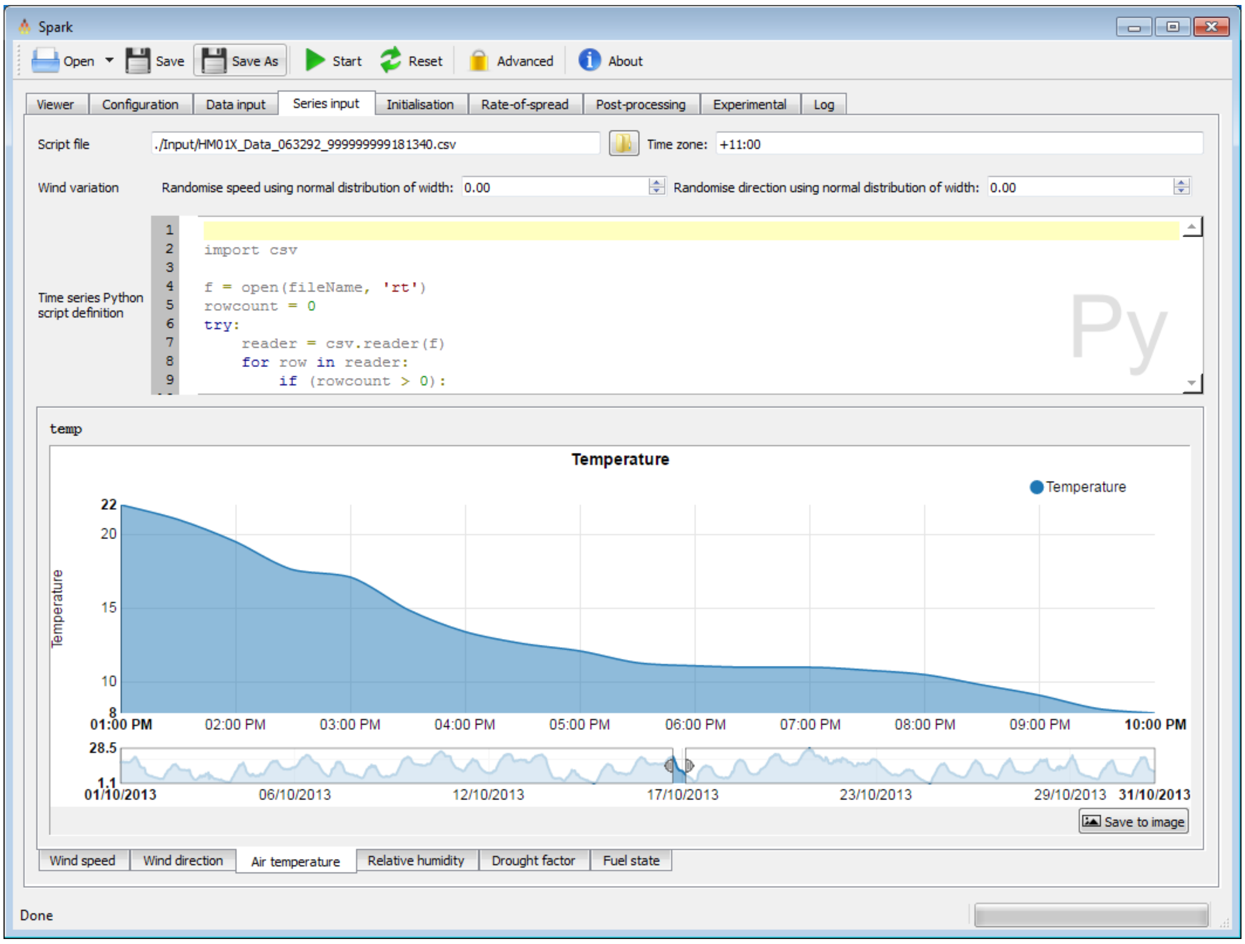
Table 11 - Field names for series input

The window is divided into an input Python script in the upper half and a preview of the generated time series in the lower half. The spark-gui application pre-defines seven time series, given in Table 12. A preview of each of these can be displayed using the appropriate tab at the bottom of the preview. These time series are populated using the Python script field.

NAME	DESCRIPTION
wind_speed	Wind speed.
wind_dir	Wind bearing in degrees.
temp	Air temperature.
rel_hum	Relative humidity (%).
dew_temp	Dew point temperature.
drought_fac	Drought factor.
fuel_state	Fuel state parameter.

Table 12 - Pre-defined time series names for spark-gui

The Python script, if used, must create vectors containing any or all of the time series names given in Table 12. Additionally, it must create a *time* vector which defines the time values in ISO 8601 format. Optionally, a time series for a particular variable can be specified using the series name with '*_time*' appended. For example, *temp_time* will create a time series for the temperature vector *temp*.



Script file ./Input/HM01X_Data_063292_999999999181340.csv Time zone: +11:00

Wind variation Randomise speed using normal distribution of width: 0.00 Randomise direction using normal distribution of width: 0.00

Time series Python script definition

```

import csv

f = open(fileName, 'rt')
rowcount = 0
try:
    reader = csv.reader(f)
    for row in reader:
        if (rowcount > 0):
```

Figure 13 - Spark-gui series input screen.

In the *proj1* example, the Python script uses the Python CSV parser to read a CSV file and append the values within the CSV file to vectors of the temperature, relative humidity, wind speed and wind direction. The script performs an additional check to make sure an entry in the CSV is not blank, and only adds values to the series if data is present.

Multi-level wind field can be specified using an array of arrays in the Python time series script for the `wind_speed` and `wind_dir` variables. The first array for this is a vector of timestamps, the second array is a vector of wind speeds or directions at the first vertical level, the third array is a vector of wind speeds or directions at the second vertical level, and so on. The wind field for the particle is interpolated from the three-dimensional wind field, so the local wind vector for the particle will smoothly vary between the wind fields specified at various levels.

For example, a multi-level wind field can be specified using the definition:

```
wind_dir = [['2013-10-17T13:00:00', '2013-10-17T14:00:00'], [90, 180] ,[100,
190]]
wind_speed = [['2013-10-17T13:00:00', '2013-10-17T14:00:00'], [25, 30], [30,
35]]
```

The above code would give a wind from a 90° bearing at the ground level and a wind from a bearing of 180° at the first layer at 1pm. This would shift to a wind from a 100° bearing at the ground level and a wind from a bearing of 190° at the first layer at 2pm. Similarly, the wind speed would be 25 km/h at the ground level and 30 km/h at the first layer at 1pm. This would shift to 30 km/h at the ground level and 35 km/h at the first layer at 2pm.

Note the code requires multi-level winds to be enabled, this is currently specified in the firebrand experimental module, see section 3.9.1 for details.

Time vector

A '*time'* vector must be defined in the Python script. This defines the time values for *all* series. To define a time for a particular series this *time* vector can be overwritten using the series name with '*_time*' appended.

Series definitions

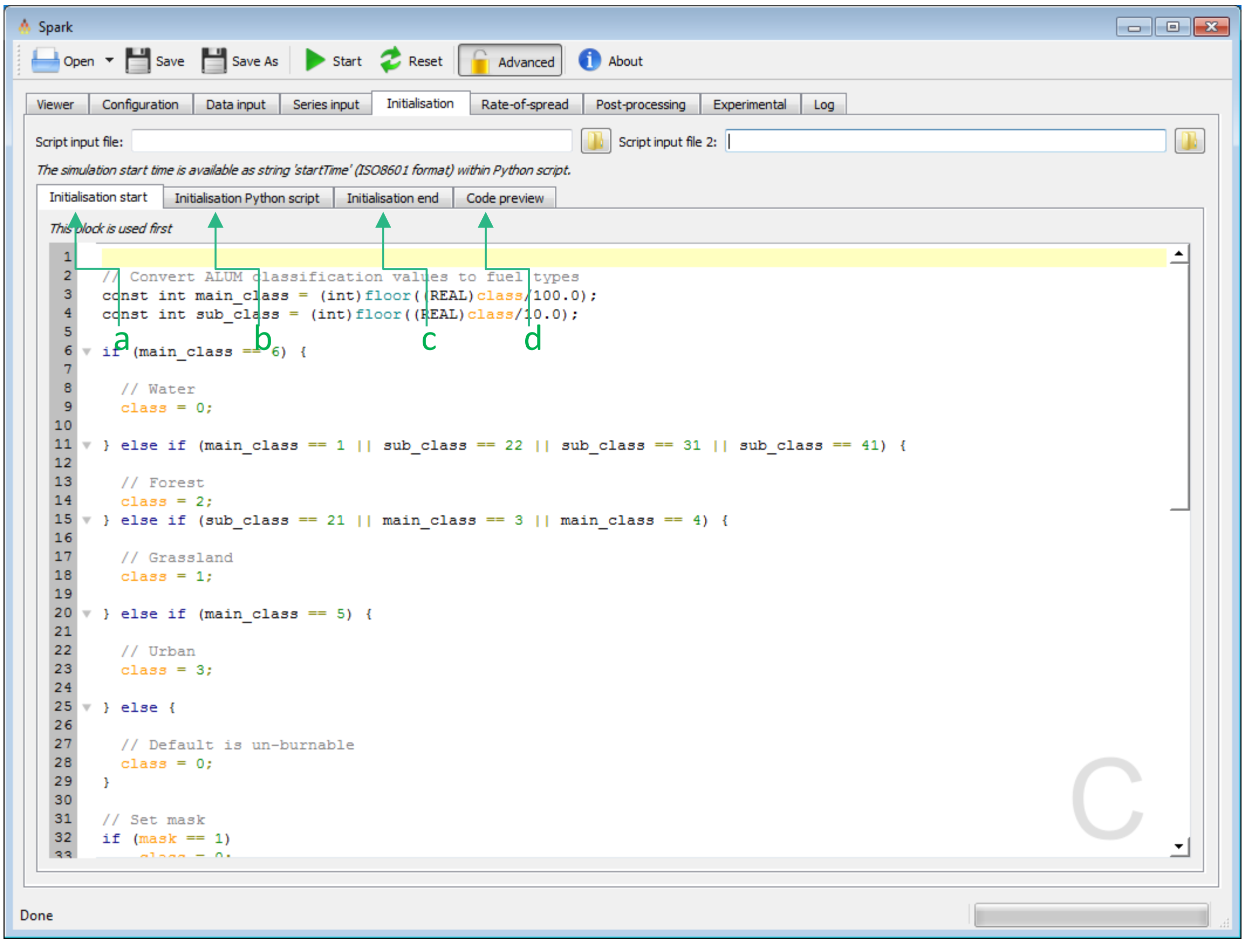
If the Python script does not define a series the series will not be created and will not be available in the rate-of-spread models within Spark.

Units

No units are defined within the framework. The user must take care of unit conversion within rate-of-spread models.

3.4 Initialisation model

The initialisation model is a powerful pre-processing step run over all data layers after they are created and used in the simulation. The use of an initialisation model is entirely optional, but it allows for manipulation and population of the data layers. The model allows values in the data layers to be re-written or populated according to any user-defined function. The final initialisation script must be in OpenCL *C* code.



The simulation start time is available as string 'startTime' (ISO8601 format) within Python script.

This block is used first

```

// Convert ALUM classification values to fuel types
const int main_class = (int)floor((REAL)class/100.0);
const int sub_class = (int)floor((REAL)class/10.0);

if (main_class == 6) {

  // Water
  class = 0;

} else if (main_class == 1 || sub_class == 22 || sub_class == 31 || sub_class == 41) {

  // Forest
  class = 2;
} else if (sub_class == 21 || main_class == 3 || main_class == 4) {

  // Grassland
  class = 1;

} else if (main_class == 5) {

  // Urban
  class = 3;

} else {

  // Default is un-burnable
  class = 0;
}

// Set mask
if (mask == 1)
```

Figure 14 - Spark-gui initialisation model.

The processing in *proj1* shows a typical use of an initialisation model. The raw data layer for the classification uses a three digit Australia land use classification code (ALUM). This three-digit code must be converted into the four classifications used within this example: un-burnable (0), grassland (1), forest (2) and urban (3). For example, the first step converts any ALUM code starting with '6' into an un-burnable region, as codes starting with '6' are water regions.

The second step of the processing carries out a different processing function. In this step, the layers *fuel_hazard_score_surface*, *fuel_hazard_score_near_surface* and *fuel_height_near_surface* for the dry eucalypt model are populated within each cell using an exponential growth curve based on the fuel age layer.

The initialisation model can be used to carry out any such processing of this type. For flexibility the model is split within the Spark applications into three steps. The full model script is built up from three blocks: a starting script (Figure 14, a), a script generated by Python (Figure 14, b) and an ending script (Figure 14, c). For reference the entire initialisation script is shown in a separate tab

(Figure 14, d). This structure allows, for example, a Python script to automatically generate mappings between multiple fuel types and the fuel types used within the application.

The fields in this tab are:

NAME	DESCRIPTION	TYPE	XML
Script input file	A filename which can be used within the Python script. A variable is created within the Python script called 'fileName' containing the contents of this field.	Filename	*Initialisation Python input file*
Script input file 2	A second filename which can be used within the Python script. A variable is created within the Python script called 'fileName2' containing the contents of this field.	Filename	*Initialisation Python input file 2*
Initialisation start	A script specifying the starting block of the initialisation model.	Text	*Initialisation start string*
Initialisation Python script	A Python script defining a generated block of text to append to the initialisation model. The text must be added to a string named 'initString' within the Python script. This python script also has access to the *seed* variable.	Text	*Initialisation Python script*
Initialisation end	A script appended to the end of the start block and the generated Python script.	Text	*Initialisation end string*

Table 13 - Field names for initialisation model

The user-defined output layers can be named in the '*Initialisation Python script*' by setting an outputJSON variable. The JSON much contain an array named '*Layers'* consisting of strings for each of the output layers in order from *ouput0* onwards. An example script naming the first three layer is:

```
JSON = {
    "Layers": [
        {
            "Name": "Arrival time (s)"
        },
        {
            "Name": "Speed (m/s)"
        },
        {
            "Name": "Maximum intensity (kW/m)"
        }
    ]
}
outputJSON = json.dumps(JSON)
```

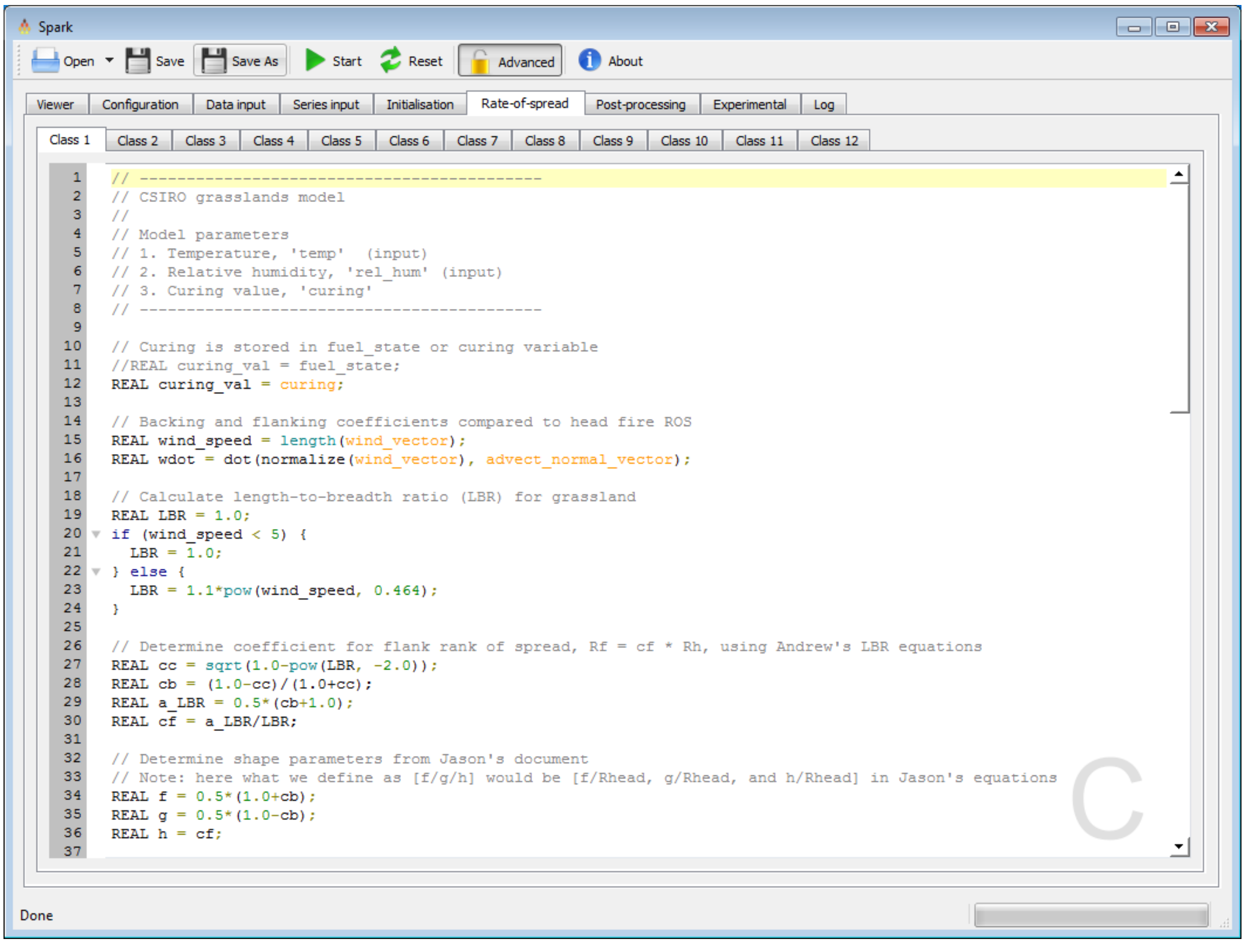
Initialisation and optimisation

For optimisation it is best to put as much calculation as possible within the initialisation model, rather than the rate-of-spread models.

3.5 Rate-of-spread models

The rate-of-spread models are the core of the Spark application. These define the local rate of spread of a fire perimeter based on an empirical relationship. For the example *proj1* three rates-of-spread are defined: the CSIRO grasslands model, the Dry Eucalypt model and a placeholder urban model. The rate-of-spread model scripts must be in OpenCL *C* code.

There are twelve input tabs within this window corresponding to fuel classifications 1-12. The rate-of-spread model in the '*Class 1*' tab is used to calculate the rate of spread within any cells with a fuel classification value of 1, the rate-of-spread model in the '*Class 2*' tab is used to calculate the rate of spread within any cells with a fuel classification value of 2, and so on.



```
// -------------------------------------------
// CSIRO grasslands model
//
// Model parameters
// 1. Temperature, 'temp'  (input)
// 2. Relative humidity, 'rel_hum' (input)
// 3. Curing value, 'curing'
// -------------------------------------------

// Curing is stored in fuel_state or curing variable
//REAL curing_val = fuel_state;
REAL curing_val = curing;

// Backing and flanking coefficients compared to head fire ROS
REAL wind_speed = length(wind_vector);
REAL wdot = dot(normalize(wind_vector), advect_normal_vector);

// Calculate length-to-breadth ratio (LBR) for grassland
REAL LBR = 1.0;
if (wind_speed < 5) {
  LBR = 1.0;
} else {
  LBR = 1.1*pow(wind_speed, 0.464);
}

// Determine coefficient for flank rank of spread, Rf = cf * Rh, using Andrew's LBR equations
REAL cc = sqrt(1.0-pow(LBR, -2.0));
REAL cb = (1.0-cc)/(1.0+cc);
REAL a_LBR = 0.5*(cb+1.0);
REAL cf = a_LBR/LBR;

// Determine shape parameters from Jason's document
// Note: here what we define as [f/g/h] would be [f/Rhead, g/Rhead, and h/Rhead] in Jason's equations
REAL f = 0.5*(1.0+cb);
REAL g = 0.5*(1.0-cb);
REAL h = cf;

```

Figure 15 - Spark-gui rate-of-spread models.

The grassland and dry eucalypt models are complex models containing a fair amount of code. A much simpler example illustrating the rate of spread scripts is the urban rate of spread model in the *Class 3* tab. This rate-of-spread model is given by:

```
speed = 0.01*wind;
```

All rate-of-spread models must contain a `speed =` definition. This defines the outward speed of the fire perimeter. Sets of isochrones for three different speed definitions are shown in Figure 16. In each case the initial fire is a circle of diameter 20 metres, shown as the inner isochrone at zero time. Isochrones are shown at 25 second intervals.

- In the first case (Figure 16, a) the speed definition is `speed = 1`. The initial circular perimeter spreads with a radial speed of 1 m/s in all directions, ending as a circle of diameter 220 metres.
- The second case (Figure 16, b) uses the wind vector defined in the simulation. The wind vector in these examples is in the vertical direction with respect to the figure, from the bottom to the top, with magnitude 1 m/s. The speed definition here is `speed = 1 + wind`, where `wind` is specially defined quantity[1] representing the component of the wind in the direction of the fire front. In this case the fire grows outwards at a rate of 1 m/s and moves forward at an additional rate of 1 m/s, resulting in a stretched obround fire perimeter.
- The final example (Figure 16, c) is an example of a non-linear rate of spread. The speed definition here is `speed = 1 + wind`^{2}, resulting in a pointed fire perimeter.

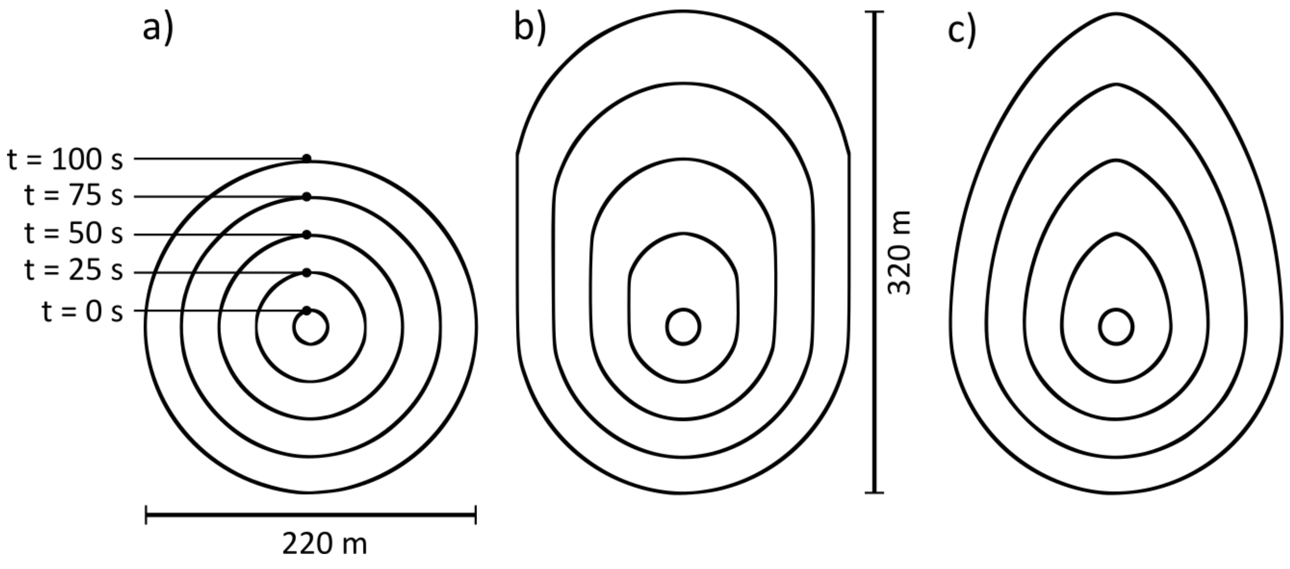


Figure 16 - Speed definition examples.

In the case of the urban model, the fire perimeter is simply moved in the direction of the wind at 1% of the wind speed.

The other two models in the example use a much more complex relationship for the rate of spread which is coupled with factors such as fuel moisture and the topography of the terrain. For example, the CSIRO grasslands model calculates the rate-of-spread in a number of steps.

- Firstly, the two-dimensional shaping of the fire is determined by calculating the length to breadth ratio of the spreading fire, based on the current wind speed. This is then used to figure out the fraction of the head fire rate of spread to be used around the fire perimeter.

[1] Mathematically the variable *wind* is defined as $wind = \max(\hat{\mathbf{n}} \cdot \mathbf{w}, 0)$, where $\mathbf{w}$ is the wind vector and $\hat{\mathbf{n}}$ is the normal vector to the fire perimeter.

- Next, the curing coefficient used in the model is calculated using an empirical formula given by Cruz 2015. The `curing` variable corresponds to the value defined either in the terrain input layer, or the curing time-series. Note that local variables, such as `curing_coeff`, can be defined and used in the script.

```
// Calculate curing coefficient from Cruz et al. (2015)
REAL curing_coeff;
if (curing < 20)
    curing_coeff = 0;
else
    curing_coeff = 1.036/(1+103.989*exp(-0.0996*(curing-20)));
```

- Next, the grassland fuel moisture is calculated using a relationship given by McArthur 1966, based on the local temperature, `temp`, and relative humidity, `rel_hum`. These two layers correspond to

```
// Fuel moisture content approximated using McArthur (1966)
REAL GMf = 9.58-(0.205*temp) + (0.138*rel_hum);
```

- The fuel moisture is used to calculate the grassland moisture coefficient:

```
// Calculate moisture coefficient from Cheney et al. (1998)
REAL moisture_coeff;
if (GMf <= 12)
    moisture_coeff = exp(-0.108*GMf);
else if ( wind_speed <= 10 )
    moisture_coeff = 0.684-0.0342*GMf;
else
    moisture_coeff = 0.547-0.0228*GMf;
```

- Finally, the speed is calculated using the CSIRO grassland model (Cheney 1998). Note the spread rate is converted to m s^{-1} from km hr^{-1}:

```
// Calculate spread rate from Cheney et al. (1998)
if ( wind_speed >= 5.0 )
    head_speed = (1.4+0.838*pow((wind_speed-5),0.844))*moisture_coeff*
    curing_coeff/3.6;
else
    head_speed = (0.054+0.269*wind_speed)*moisture_coeff*curing_coeff/3.6;
```

- The model then applies the Kataburn (Sullivan 2014) slope correction:

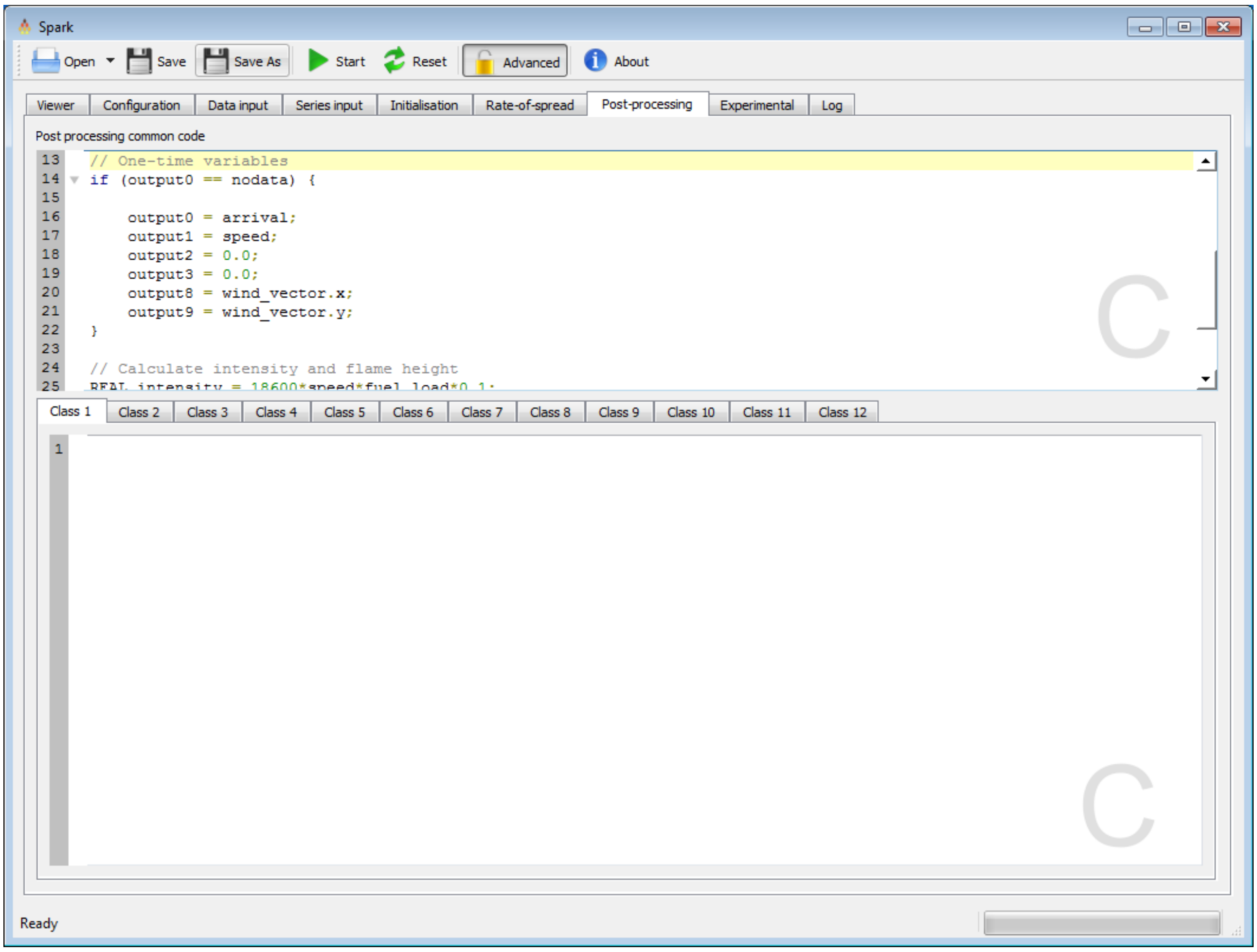
```
// Calculate slope effect
REAL slope_in_normal_dir =
degrees(atan(dot(normal_vector,grad(elevation))));
slope_in_normal_dir = min(max(slope_in_normal_dir,-20),20);
REAL slope_coeff = pow(2.0, 0.1*fabs(slope_in_normal_dir));

if (slope_in_normal_dir >= 0)
   speed *= slope_coeff;
else
   speed *= slope_coeff/(2*slope_coeff-1.0);
```

Similar processing is carried out for the Dry Eucalypt model to calculate the rate of spread.

3.6 Post-processing models

The post-processing models are key to creating user defined output in Spark. Without any post-processing models, Spark will only output the arrival time raster and a shapefile of isochrones.

The post-processing models are used to calculate the output variables (*output0*, *output1*, …, *output9*) based on user defined empirical relationships of propagation/solver variables (such as *speed* and *fuel_load*) and other user defined data layers. After the simulation has been completed, these output variables are saved as layers 0 to 9 respectively in the *Output user data GeoTIFF file* specified in the *Configuration* tab.



```
// One-time variables
if (output0 == nodata) {

    output0 = arrival;
    output1 = speed;
    output2 = 0.0;
    output3 = 0.0;
    output8 = wind_vector.x;
    output9 = wind_vector.y;
}

// Calculate intensity and flame height
REAL intensity = 18600*speed*fuel_load*0.1;
```

Figure 17 - Post-processing.

There are twelve input tabs within this window corresponding to fuel classifications 1-12. The post-processing model in the '*Class 1*' tab is used to calculate the outputs within any cells with a fuel classification value of 1, the rate-of-spread model in the '*Class 2*' tab is used to calculate the outputs within any cells with a fuel classification value of 2, and so on. There is also a *Post processing common code* script which applies the code to all classes before the individual class code is run. The post-processing model scripts must be in OpenCL *C* code. It should also be noted that the post-processing models run at the end of each time step on the entire burned region unless specified otherwise.

For the example *proj1*, all of the post-processing is completed in the *Post processing common code* script as there is no difference in post-processing models for different classes in this case. The first few lines are for calculating ‘one time’ variables, in other words, outputs that you only wish to calculate at the time when the fire enters that cell. In this case, arrival time is being saved to *ouput0* and propagation speed is being saved to *output1*:

```
if (output0 == nodata){
    output0 = arrival;
    output1 = speed;
    output2 = 0;
    output3 = 0;
}
```

The following lines use empirical models to calculate the fireline intensity and flame height. In this case, the maximum intensity and flame height experienced in a cell are being saved to outputs 2 and 3 respectively:

```
// Calculate intensity and flame height
REAL intensity = 18600.0*speed*fuel_load*0.1;
REAL flame_height = 0.0775*pow(intensity, 0.46);

output2 = max(output2, intensity);
output3 = max(output3, flame_height);
```

The various parameters accessible in the post-processing models are given in the Appendix.

3.7 Ensembles

Spark can run multiple simulations and calculate statistics using results from these simulations with the *Output ensemble statistics script*. This script is run at the end of each simulation and allows a running variable *R* to be updated. Variables available values to this script are the output user data grids, *A*, the count of times the cell has burnt over all simulations, *B*, the output layer number, *kpos*, and the current run count, *v*. The output statistic is stored in the variable *R*. This is initialised to *nodata* and holds the same value between simulations allowing, for example, running averages to be calculated. The output from this analysis can be written to a multi-layer GeoTIFF file specified using the *GeoTIFF file* parameter.

An example of an ensemble simulation is the *proj1_ensemble* project. This calculates impact probability (count of number of simulations reaching the cell as a percentage):

```
R = 100.0*B/v;
```

Or the cumulative average arrival time:

```
if (A != nodata)
    R = (A+(B-1)*R)/B;
```

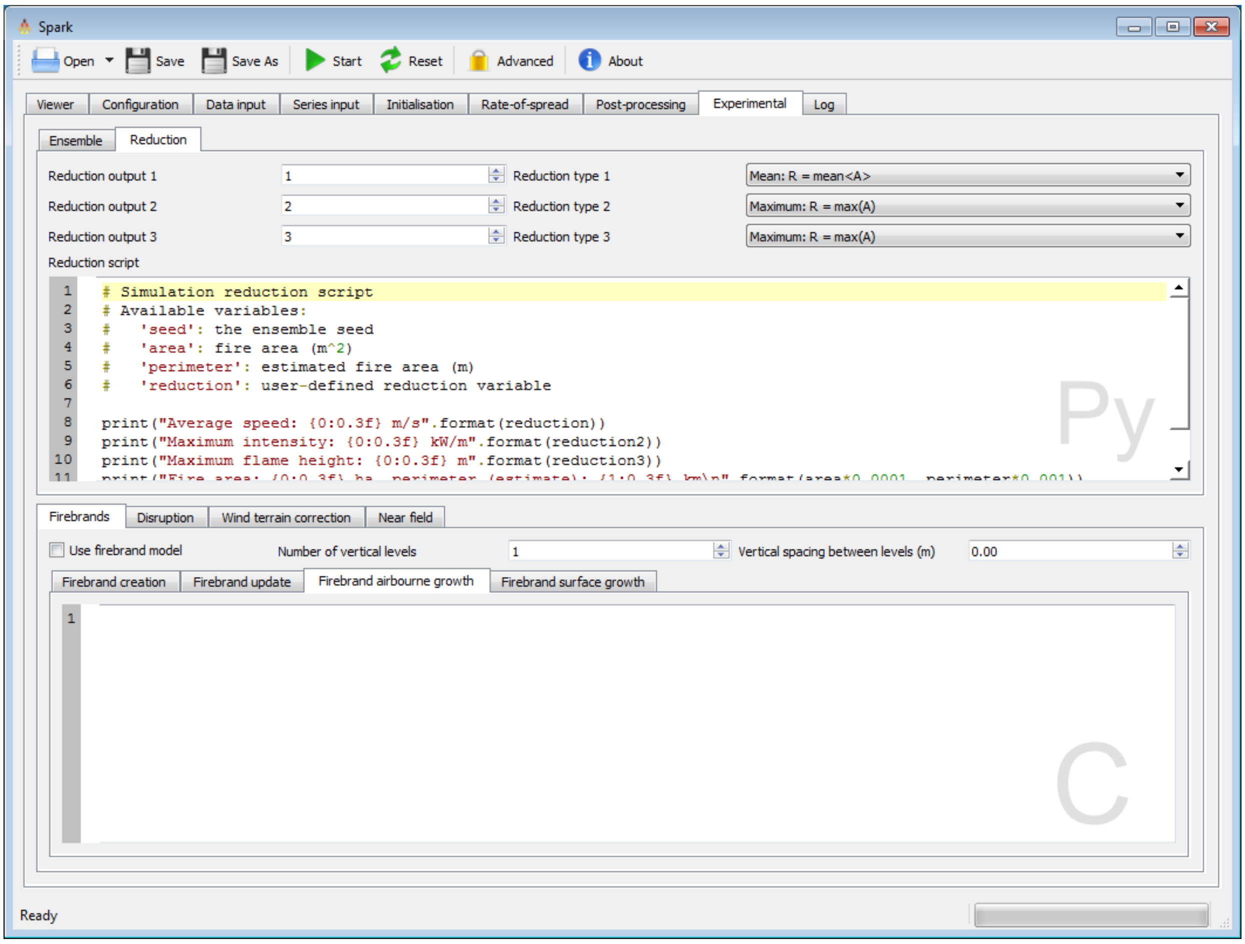
NAME	DESCRIPTION	TYPE	XML
Number of ensemble simulations	This variable sets the number of ensembles to run.	Number	*Number of simulations*
Ensemble layers	The number of output layers (starting from zero) to run the *Output ensemble statistics script* on.	Number	*Output stats layers*
GeoTIFF file	The path and filename of a GeoTIFF file containing the calculated statistic.	Filename	*Output stats raster file*
GeoTIFF type	The data type written to the output data GeoTIFF file. The data is converted to this type before being written. Different data types may provide savings in space.	Selection	*Output stats raster type*
Random seed	The random seed for the OpenCL code within the application. This affects the initialisation, rate of spread, post-processing models and any fire behaviour modules.	Number	*Random seed*
Output ensemble statistics unit type	Sets the displayed unit type for the statistic calculated by the ensemble script.	Selection	*Output stats unit type*
Output ensemble statistics script	An OpenCL script for spatially combining arrival time values from multiple simulations.	Text	*Output stats script*
Simulation control script	This Python script controls the number of simulations, *N*. The *projectName* string can also be set here. The ignition point can also be set by defining *lat*, *long*, *radius* and *time* arrays.	Text	*Simulation number script*

Table 14 - Ensemble options

If the *Number of ensemble simulations* input is larger than 1, then an ensemble number input will appear in the top right corner of the main toolbar. Changing this input allows the user to toggle between outputs of individual simulations in the *Viewer* tab and time series in the *Series input* tab.

3.8 Reductions

Spark can perform a *reduction* on up to three particular layers in the user-defined outputs. For example, the layer representing maximum flame height can be reduced to a single number for the maximum flame height found anywhere in the simulation. A Python script to carry out this processing is shown in Figure 18.



Reduction output 1	1	Reduction type 1	Mean: R = mean<A>
Reduction output 2	2	Reduction type 2	Maximum: R = max(A)
Reduction output 3	3	Reduction type 3	Maximum: R = max(A)

Reduction script

```
# Simulation reduction script
# Available variables:
#   'seed': the ensemble seed
#   'area': fire area (m^2)
#   'perimeter': estimated fire area (m)
#   'reduction': user-defined reduction variable

print("Average speed: {0:0.3f} m/s".format(reduction))
print("Maximum intensity: {0:0.3f} kW/m".format(reduction2))
print("Maximum flame height: {0:0.3f} m".format(reduction3))
print("Fire area: {0:0.3f} ha, perimeter (estimate): {1:0.3f} km\n".format(area*0.0001, perimeter*0.001))
```

☐ Use firebrand model	Number of vertical levels	1	Vertical spacing between levels (m)	0.00

Figure 18 - Reduction script.

The layers to perform the reduction on is specified in the '*Reduction output*'. A choice of reduction operations is available from the '*Reduction type*' drop-down list.

The available variables in the reduction script are:

NAME	DESCRIPTION
seed	The random seed number of the current ensemble.
area	The final fire area in m^2.
perimeter	The final fire perimeter in m.
reduction	The result from the reduction operation on output 1.
reduction2	The result from the reduction operation on output 2.
reduction3	The result from the reduction operation on output 3.

Table 15 - Reduction script inputs

3.9 Experimental fire behaviour models

Currently, four experimental fire behaviour modules are currently available:

1. Firebrand transport for long range spotting.
2. Disruption layers for roads and rivers.
3. Wind-terrain correction for wind flow modelling over terrain.
4. Near field modelling for fire line attraction and coalescence effects.

These models are experimental as they rely either in part or wholly on parameters which have not been calibrated to wildfire simulations. **As such these models and any examples provided of these models should be used with caution for operational and risk predictions, and should not be used without careful consideration and calibration of necessary parameters**.

Details on the terrain correction and near field model will be supplied in future version of the software as these are currently in research and development stage.

Experimental examples

Any examples supplied using the experimental options are for demonstration only, these models and parameters should not be applied to wildfire scenarios without careful consideration and understanding of the models, parameters and limitations of the models used.

Experimental models

The experimental processing and fire behaviour modules in the 'Experimental' tab are liable to change or move to different locations in future versions.

3.9.1 Firebrand transport

The firebrand model allows the life cycle of a firebrand to be modelled. This is made up of three components:

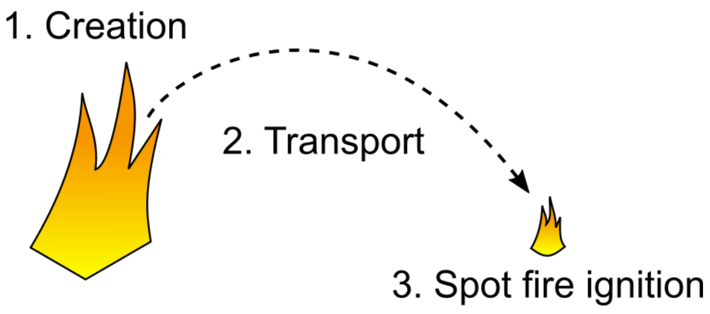


Figure 19 - Stages in the firebrand lifecycle.

1. **Creation**. The creation of firebrands is controlled using the *Firebrand creation* script. Creation is limited to one firebrand per time step per grid cell. To create a firebrand the specified radius must be greater than a minimum radius of 0.5 mm and the specified lifetime must be greater than zero. Firebrands exist until either the radius or lifetime of the firebrand is zero (or lower), or when they have landed and been mapped into the fire solver as a new spot fire.
2. **Transport**. The transport and updates to the firebrand while airbourne are controlled using the *Firebrand update* and *Firebrand airbourne growth* scripts. The *Firebrand update* is used to apply acceleration to the firebrand while airbourne. Typically this will be a wind drag model with gravity, but can be modified to apply any local acceleration. To apply wind drag and gravity a script such as the following can be used:

```
// Drag coefficient
REAL cd = 0.42;

// Ratio of air to particle densities
REAL rho_diff = 1.2/250.0;

// Relative velocity, wind vector must be in m/s
REALVEC3 rel_velocity = wind_vector-velocity;

// Set firebrand acceleration:
//   a = F = 0.5*rho_gas*cd*area*u^2
//       -   -----------------------
//       m      rho_particle*volume

acceleration = (0.375*rho_diff*cd/radius)*length(rel_velocity)*rel_velocity;

// Apply gravity
acceleration.z -= 9.8;
```

The above script calculates the acceleration on the particle from the relative wind velocity, then subtracts the vertical gravitational component. The *Firebrand airbourne growth* script can modify firebrand parameters, such as the radius or local wind vector, before the position of the particle is updated. Although possible it is not recommended that the position or velocity are updated in this script.

3. **Spot fire ignition**. The firebrand is checked for intersection with the ground plane. If it reaches the ground the firebrand processing switches to the ground growth model, controlled by the *Firebrand surface growth* script. This is a basic model allowing only increase in the firebrand radius until it is above the size in which it can be modelled as a new spot fire in the fire spread solver. The *Firebrand surface growth* script has limited access to the data for the fire spread model, including the state, class, subclass and output layers from the main solver. Note that the lifetime of the firebrand still applies on the ground, and if the lifetime of a firebrand on the ground reaches zero it is extinguished.

NAME	TYPE	DESCRIPTION	AVAILABILITY*
position	3D vector	The 3D position of the firebrand (m).	CR, UP, AG, GG
velocity	3D vector	The 3D velocity of the firebrand (ms^{-1})	CR, UP, AG
acceleration	3D vector	The 3D acceleration applied to the firebrand (ms^{-2})	UP
wind_vector	3D vector	The wind vector (ms^{-1}).	CR, UP, AG
cell	3D vector	The centroid of the current grid cell (m).	CR
radius	scalar	The radius of the firebrand (m), this is limited to a minimum of 0.5 mm. If lower than the minimum radius the firebrand is extinguished.	CR, UP, AG, GG
lifetime	scalar	The lifetime of the firebrand.	CR
time	scalar	The current lifetime of the firebrand (s). This is reduced from the initial value of **lifetime** to zero. When zero the firebrand is extinguished.	UP, AG, GG
q	scalar	A general-purpose user defined variable	CR, UP, AG, GG
state	scalar	Whether the cell is currently un-burnable (value 0) or burnable (value 1).	CR, GG
class	scalar	The fuel classification value.	CR, GG
subclass	scalar	The fuel sub-classification, can be 0-255.	CR, GG
distance	scalar	The distance to the perimeter from the point on the ground below the firebrand (m).	CR, UP, AG, GG
speed	scalar	The normal speed of the fire at the perimeter (ms^{-1}).	CR
speed	scalar	The speed of growth of the firebrand (ms^{-1}), this is used to update the radius.	UP, AG, GG
elevation	scalar	The land elevation at the point on the ground below the firebrand (m).	UP, AG, GG
height	scalar	The vertical height of the firebrand above ground (m).	UP, AG, GG
output*[0-9]*	scalar	Internal user-defined data layer, written to 'output grid' layer [0-9].	CR, AG, GG

Table 16 - Firebrand script inputs
*** CR: Creation script, UP: Update script, AG: Airborne growth script, GG: Growth script**

The airbourne transport component is not required to implement firebrands. If required firebrands landing locations can be directly specified. The example project *proj1_firebrand*, shown in Figure 20, uses Ellis' maximum spotting distance model to create new spot fires at a given distance from the fire front. Note this only uses a creation script and a ground growth script.

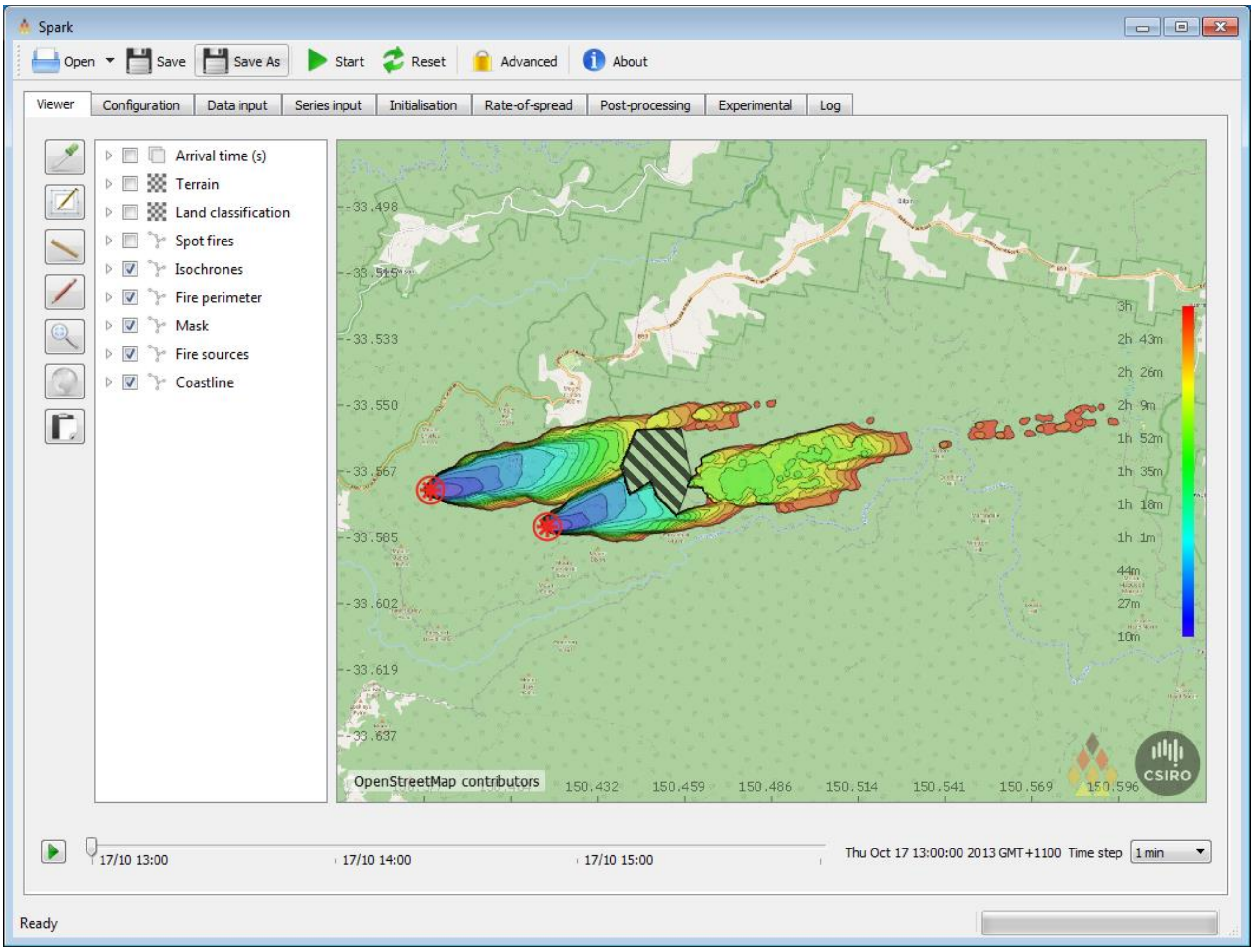


Figure 20 - Maximum spotting distance model implemented using the firebrand processor.

The firebrand model is three-dimensional, and can use a three-dimensional wind field if available. A three-dimensional wind field can be applied as multiple vertical levels using a different time series in each. The number of vertical levels and the spacing between each of the levels is set using the *Number of vertical levels* and *Vertical spacing between levels* input fields on the *Firebrands* processor tab.

An example of a simulation with multi-level winds is shown in Figure 21. Firebrands are created and lofted into a cross-stream causing them to fall to the south of the front. The model assumes that the firebrand is buoyantly lofted for 1 minute after creation, allowing the firebrands to rise and enter the cross-stream. It should be emphasised that many of the characteristics of firebrand transport is still under research and parameters such as lofting times are unknown. Implementation of the firebrand model should consider such factors before use and rely on researched parameters or careful calibration to existing fire data.

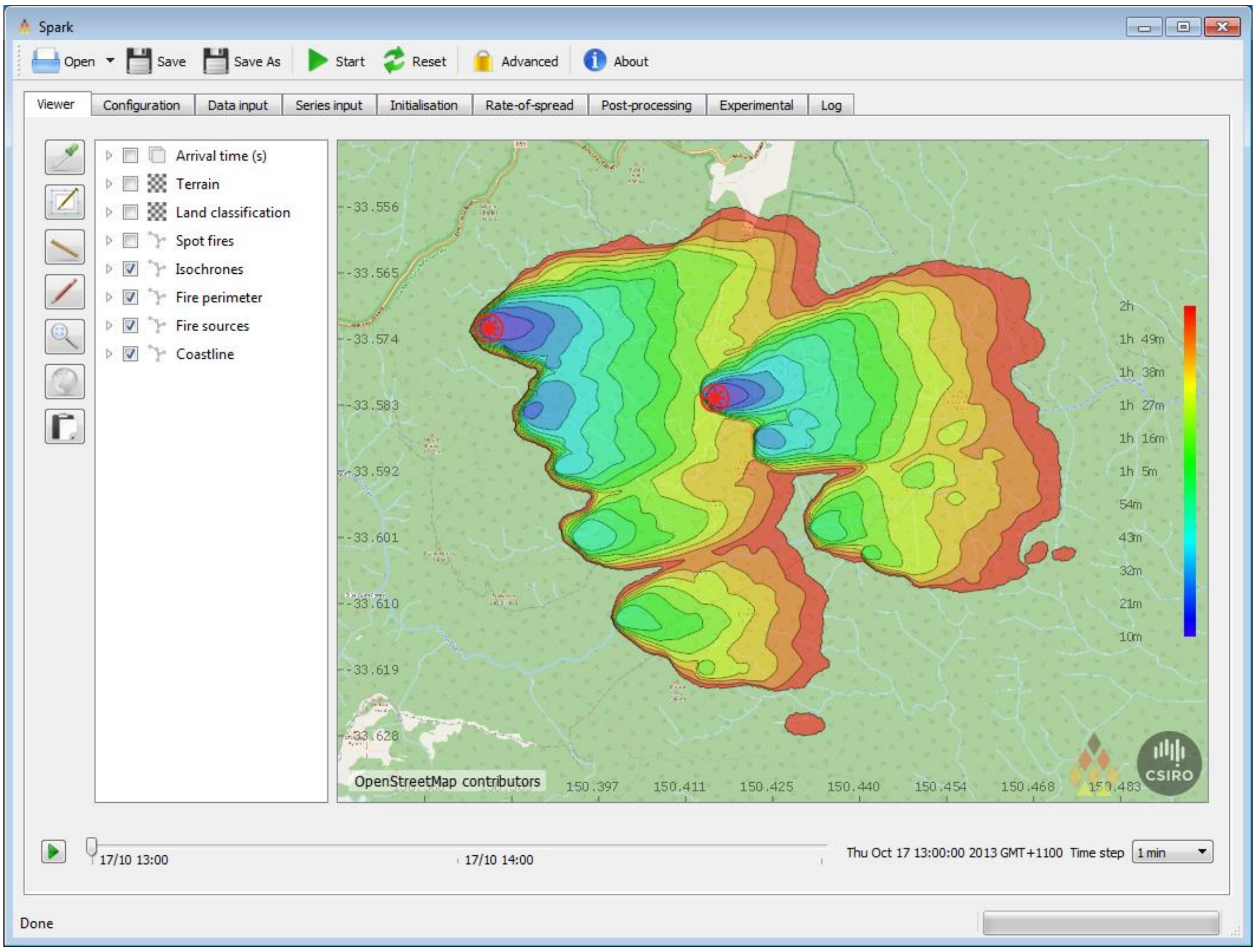


Figure 21 - Spot fires created parallel to the fire due to firebrand transport in a high level cross-wind. Note this is an example of the model and not representative of realistic fire conditions.

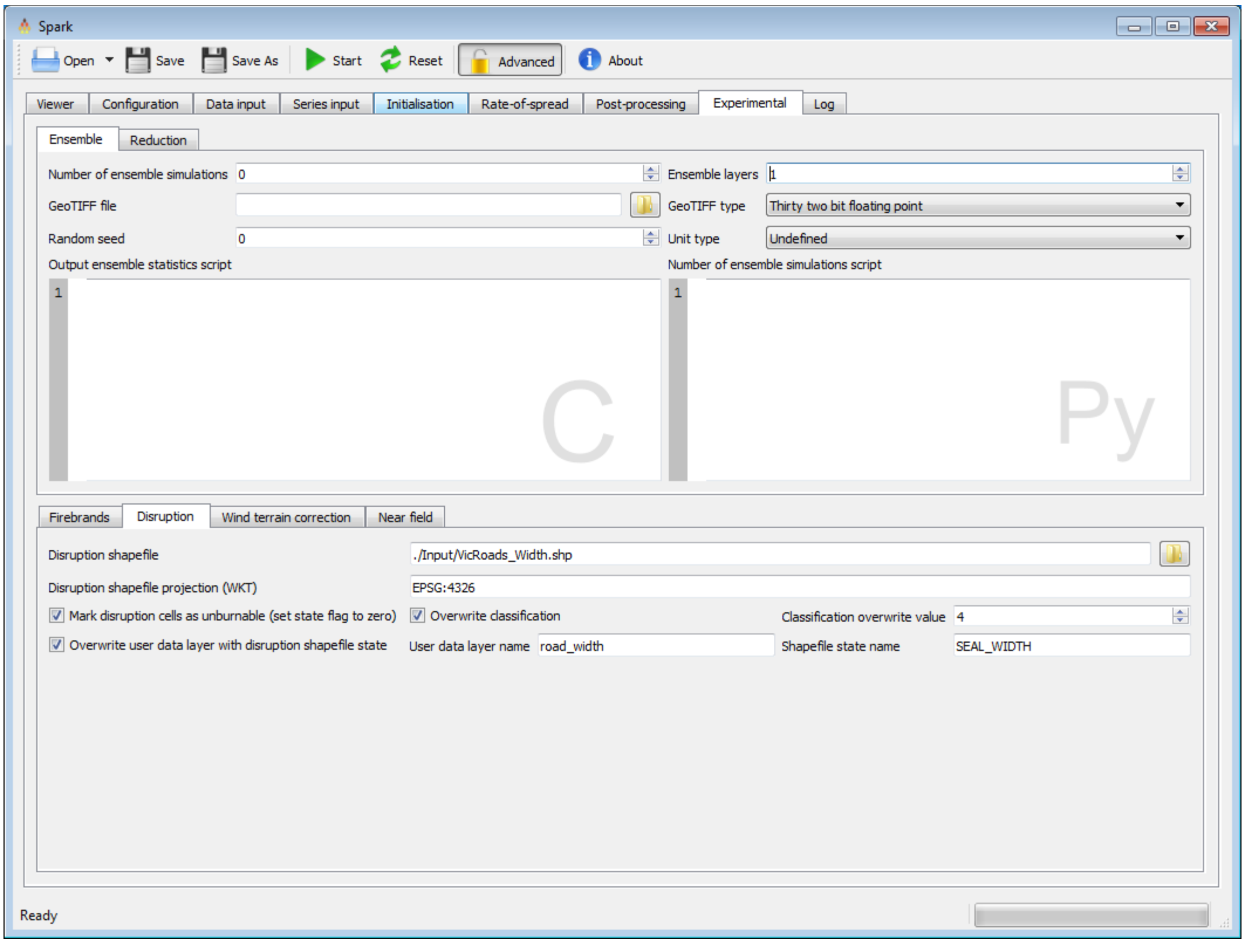
Firebrand parameters

The behaviour of lofted firebrands is still an active scientific research area. The models supplied with Spark are intended to show utility of the modelling system and are not designed to be used for fire predictions without substantial consideration of the underlying model.

Many of the parameters used in these models, including the firebrand creation probability, lofting time and aerodynamic characteristics of the firebrands, should be carefully researched before implementation and use in the system.

3.9.2 Disruption modelling

Spark can read a vector line network from a shapefile and use it to impose a barrier to fire spread or change the classification of the fuel type under the vector. All vectors are imposed as a one cell wide line within the fuel classification layer.



Number of ensemble simulations: 0
GeoTIFF file:
Random seed: 0
Ensemble layers: 1
GeoTIFF type: Thirty two bit floating point
Unit type: Undefined
Output ensemble statistics script
Number of ensemble simulations script

Disruption shapefile: ./Input/VicRoads_Width.shp
Disruption shapefile projection (WKT): EPSG:4326
☑ Mark disruption cells as unburnable (set state flag to zero)
☑ Overwrite classification
Classification overwrite value: 4
☑ Overwrite user data layer with disruption shapefile state
User data layer name: road_width
Shapefile state name: SEAL_WIDTH

Figure 22 - Disruption options.

Uses for the disruption layer include implementing the effect of roads, waterways or fire breaks in the simulation. Disruptions can be made to permanently affect the spread of the fire (for example, by setting the land classification under the disruption vectors to zero), temporarily halting the spread of the fire (for example, by clearing the state flag and making the cell burnable at random in a rate of spread model), or changing the fuel type under the vector layer.

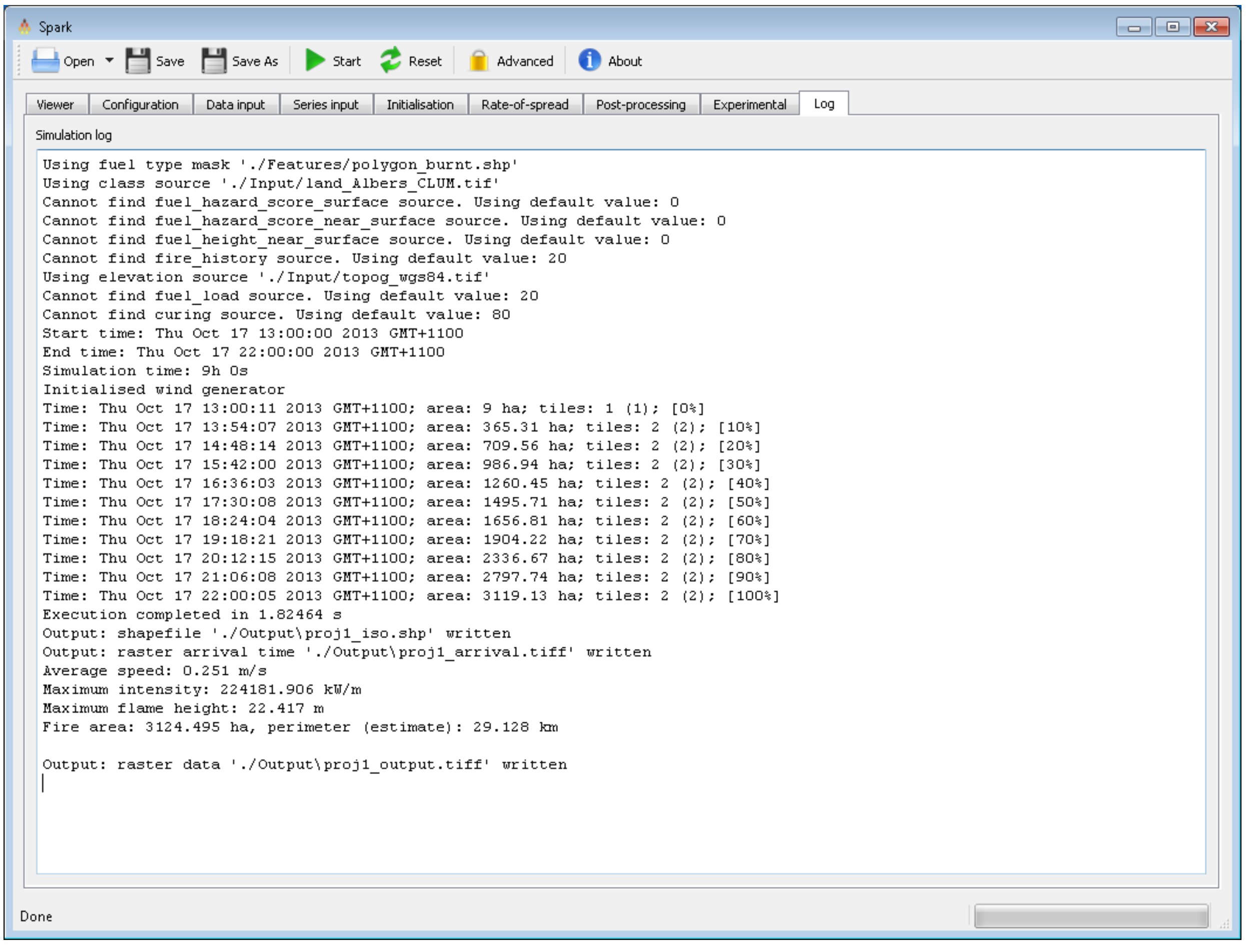
The inputs for the disruption model are shown in Table 17 while example inputs for the disruption model are shown in Figure 22. The example inputs shown are from *proj3* which is supplied with the software. In this example the *VicRoads_Width.shp* shapefile is read in as the *disruption shapefile.* The land classification is overwritten to class 4 in this case so that the disruption can be handled specifically in that rate-of-spread model. The state has also been set to zero, meaning that the fire will not be able to burn that cell unless the state is switched back to one. The inputs also specify that the *Shapefile state name* or field of *SEAL_WIDTH* should be read from the shapefile and its value applied to the user data layer *road_width* so that it can be accessed in both the rate-of-spread models and post-processing models. In this project, an example disruption model is implemented in class 4 where the failure probability of the disruption decreases with a larger road width, and the cell becomes burnable (`state = 1`) if a random probability is exceeded.

NAME	DESCRIPTION	TYPE	XML
Disruption shapefile	An ESRI shapefile for the vector representing the disruption network (e.g. roads, waterways or fire breaks). Only polylines are used from the shapefile.	Filename	*Disruption shape source file*
Disruption shapefile projection (OGC WKT)	The projection for the shapefile in the Open Geospatial Consortium Well-Know-Text standard.	Text	*Disruption shape projection WKT*
Mark disruption cells as unburnable (set state flag to zero)	Sets the state flag of any cells under the disruption vectors to zero, making them currently un-burnable.	Checkbox	*Disruption clear state*
Overwrite classification	Option to overwrite land classification values under the disruption vectors with a new value.	Checkbox	*Disruption overwrite classification*
Classification overwrite value	The new value for the fuel classification under the disruption vectors.	Number	*Disruption overwrite classification value*
Overwrite user data layer with disruption shapefile state	Option to write any numeric field from the shapefile to a user-defined layer in Spark.	Checkbox	*Disruption overwrite user layer*
User data layer name	The names of the user-defined layer in Spark.	Text	*Disruption overwrite user layer name*
Shapefile state name	The name of the field in the shapefile to use.	Text	*Disruption overwrite network state name*

Table 17 - Field names for disruption behaviour module

3.10 Log

The log provides output and information on the simulation. On simulation completion similar text to that shown in Figure 23 should appear.



```
Using fuel type mask './Features/polygon_burnt.shp'
Using class source './Input/land_Albers_CLUM.tif'
Cannot find fuel_hazard_score_surface source. Using default value: 0
Cannot find fuel_hazard_score_near_surface source. Using default value: 0
Cannot find fuel_height_near_surface source. Using default value: 0
Cannot find fire_history source. Using default value: 20
Using elevation source './Input/topog_wgs84.tif'
Cannot find fuel_load source. Using default value: 20
Cannot find curing source. Using default value: 80
Start time: Thu Oct 17 13:00:00 2013 GMT+1100
End time: Thu Oct 17 22:00:00 2013 GMT+1100
Simulation time: 9h 0s
Initialised wind generator
Time: Thu Oct 17 13:00:11 2013 GMT+1100; area: 9 ha; tiles: 1 (1); [0%]
Time: Thu Oct 17 13:54:07 2013 GMT+1100; area: 365.31 ha; tiles: 2 (2); [10%]
Time: Thu Oct 17 14:48:14 2013 GMT+1100; area: 709.56 ha; tiles: 2 (2); [20%]
Time: Thu Oct 17 15:42:00 2013 GMT+1100; area: 986.94 ha; tiles: 2 (2); [30%]
Time: Thu Oct 17 16:36:03 2013 GMT+1100; area: 1260.45 ha; tiles: 2 (2); [40%]
Time: Thu Oct 17 17:30:08 2013 GMT+1100; area: 1495.71 ha; tiles: 2 (2); [50%]
Time: Thu Oct 17 18:24:04 2013 GMT+1100; area: 1656.81 ha; tiles: 2 (2); [60%]
Time: Thu Oct 17 19:18:21 2013 GMT+1100; area: 1904.22 ha; tiles: 2 (2); [70%]
Time: Thu Oct 17 20:12:15 2013 GMT+1100; area: 2336.67 ha; tiles: 2 (2); [80%]
Time: Thu Oct 17 21:06:08 2013 GMT+1100; area: 2797.74 ha; tiles: 2 (2); [90%]
Time: Thu Oct 17 22:00:05 2013 GMT+1100; area: 3119.13 ha; tiles: 2 (2); [100%]
Execution completed in 1.82464 s
Output: shapefile './Output\proj1_iso.shp' written
Output: raster arrival time './Output\proj1_arrival.tiff' written
Average speed: 0.251 m/s
Maximum intensity: 224181.906 kW/m
Maximum flame height: 22.417 m
Fire area: 3124.495 ha, perimeter (estimate): 29.128 km

Output: raster data './Output\proj1_output.tiff' written
```

Figure 23 - Spark-gui simulation log.

Any warnings or errors are highlighted in red with a description of the error encountered.

4 Spark-batch application

The *spark-batch* package is a stand-alone command-line application. The application is configured to be deployed to multi-CPU or GPU servers. The input to the application is an XML project file for the configuration of the solver and a series of input data sets. XML project files created using the *spark-gui* application are compatible with *spark-batch*.

The application must be run in a project directory containing the XML project file. This XML file is supplied to the application as a command line argument, for example, if the input XML file is *proj1.xml* the application is executed using the command:

```
spark-batch ./proj1.xml
```

Outputs will be written to the files defined within the XML file. The batch application also contains a test to ensure the application is working correctly. To carry out the test, execute the command:

```
spark-batch --test
```

If all is working correctly, an image called *spark-batch_test_result.png* will be written to the current directory. The image is shown in Figure 24.

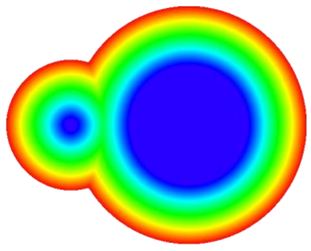


Figure 24 - spark-batch test result

The XML project file can either be generated using the spark-gui application or automatically created. The XML must be contained by an `<operation>` tag. The fields within the file have the following form:

```
<input globalname="Start time">2013-10-17T11:59:00+11:00</input>
```

The `globalname` tag for each of the inputs is given under the ‘XML’ column in each of the tables in the spark-gui section.

XML composition

XML contains escape characters, for example the less than sign ‘<’ is encoded using the string ‘&lt;’. Care must be taken to use these escape characters when manually writing code in XML.

Appendix

Spark contains a number of internally defined variables which provide access to information required for fire modelling. These are given in Table 18, with the availability of the variable shown in the last column.

VARIABLE	TYPE	DESCRIPTION	AVAILABILITY*
area	scalar	The current total fire size in m^2	IRP
easting	scalar	The cell easting value (m).	I
northing	scalar	The cell northing value (m).	I
speed	scalar	Required: sets the normal speed at the perimeter (m/s).	RP
distance	scalar	The distance to the perimeter.	R
wind	scalar	The dot product of the wind and front normal, limited by zero.	R
wind_vector	vector	The wind vector.	RP
normal_vector	vector	The normal vector of the perimeter.	R
class	scalar	The fuel classification value.	IRP
subclass	scalar	The fuel sub-classification, can be 0-255.	IRP
mask	scalar	The mask region, with a value of 1 within the mask and 0 outside.	IRP
state	scalar	Whether the cell is currently un-burnable (value 0) or burnable (value 1).	IRP
random	scalar	A random number from a uniform distribution between 0-1.	IRP
output*[0-9]*	scalar	Internal user-defined data layer, written to 'output grid' layer [0-9].	IRP
arrival	scalar	The ignition (arrival) time of the perimeter at the cell (s). No-data values indicate no recorded arrival time.	P
year	scalar	The current year in simulation time.	IRP
month	scalar	The current month in simulation time.	IRP
day	scalar	The current day in simulation time.	IRP
hour	scalar	The current hour in simulation time.	IRP
time	scalar	The current solver time (s).	RP
layername	scalar	The interpolated value within the cell from the user-defined layer or time series named *layername*.	IRP
dx(*layername*)	scalar	The x-spatial derivative of the user-defined layer named *layername*.	IR
dy(*layername*)	scalar	The y-spatial derivative of the user-defined layer named *layername*.	IR
grad(*layername*)	vector	The gradient of the user-defined layer named *layername*.	IR
nodata	scalar	The no-data value.	IRP

Table 18 - Internal variables used in Spark models.
*** I: Initialization script, R: Rate-of spread scripts, P: Post-processing scripts**

References

Cheney NP, 1998, Prediction of fire spread in grasslands. International Journal of Wildland Fire 8, 1-15.

Cruz MG, 2015a, Effects of curing on grassfires: II. Effect of grass senescence on the rate of fire spread. International Journal of Wildland Fire 24, 838-848

McArthur AG, 1966, Weather and grassland fire behaviour. Commonwealth Department of National Development. Forestry and Timber Bureau, Leaflet 100, Canberra, ACT. 23 pp.

Sullivan AL, 2009a, Wildland surface fire spread modelling, 1990–2007. 1: Physical and quasi-physical models, International Journal of Wildland Fire, 18, 349–368

Sullivan AL, 2009b, Wildland surface fire spread modelling, 1990–2007. 2: Empirical and quasi-empirical models, International Journal of Wildland Fire, 18, 369–386

Sullivan AL, 2014, A downslope fire spread correction factor based on landscape-scale fire behaviour. Environmental Modelling and Software 62, 153-163.

CONTACT US

t 1300 363 400
+61 3 9545 2176
e csiroenquiries@csiro.au
w www.data61.csiro.au

FOR FURTHER INFORMATION

James Hilton
Senior Research Scientist
t +61 3 9518 5974
e james.hilton@csiro.au

Mahesh Prakash
Group Leader
t +61 3 9545 8010
e mahesh.prakash@csiro.au

Andrew Sullivan
Team Leader
t +61 2 6246 4051
e andrew.sullivan@csiro.au