



Spark user guide

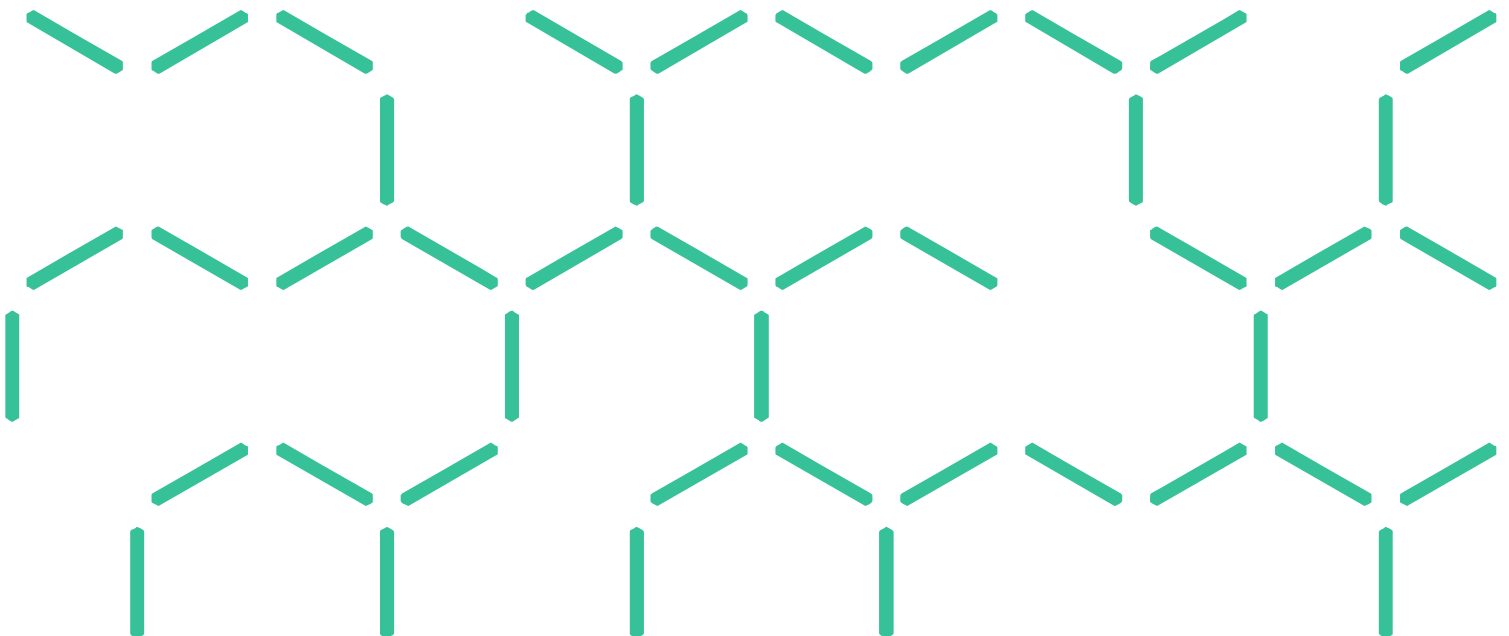
User guide for the Spark software applications

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

EP166699

February 2017

Spark version 0.9.4



Citation

Hilton JE, Swedosh W, Hetherington L, Sullivan A and Prakash M (2016) Spark user guide 0.9.4. CSIRO, Australia.

Copyright

© Commonwealth Scientific and Industrial Research Organisation 2016. To the extent permitted by law, all rights are reserved and no part of this publication covered by copyright may be reproduced or copied in any form or by any means except with the written permission of CSIRO.

Important disclaimer

CSIRO advises that the information contained in this publication comprises general statements based on scientific research. The reader is advised and needs to be aware that such information may be incomplete or unable to be used in any specific situation. No reliance or actions must therefore be made on that information without seeking prior expert professional, scientific and technical advice. To the extent permitted by law, CSIRO (including its employees and consultants) excludes all liability to any person for any consequences, including but not limited to all losses, damages, costs, expenses and any other compensation, arising directly or indirectly from using this publication (in part or in whole) and any information or material contained in it.

CSIRO is committed to providing web accessible content wherever possible. If you are having difficulties with accessing this document please contact enquiries@csiro.au.

Contents

1	Summary.....	11
2	Introduction.....	12
	2.1 Spark and configurability.....	13
3	Spark-gui application.....	14
	3.1 Configuration.....	17
	3.2 Series Input.....	21
	3.3 Gridded input.....	23
	3.4 Initialisation model.....	30
	3.5 Rate-of-spread models.....	32
	3.6 Experimental options.....	35
	3.7 Log.....	37
4	Spark-batch application.....	38
	Appendix	39
	References	40

Figures

Figure 1 - Experimental grassfire burn to provide data for new rate-of-spread models.	12
Figure 2 - Schematic of Spark layers	13
Figure 3 - Spark-gui initial screen.....	14
Figure 4 - Spark gui output.....	15
Figure 5 - Spark gui arrival time output	16
Figure 6 - Spark-gui configuration screen	17
Figure 7 - Output GeoTIFF (shaded by arrival time), and isochrones (black lines).....	20
Figure 8 - Spark-gui series input screen	21
Figure 9 - Spark-gui gridded input screen; meteorological data, file selection.....	24
Figure 10 - Spark-gui gridded input screen; meteorological data, directory selection	25
Figure 11 - Spark-gui gridded input screen; meteorological data, data transform	26
Figure 12 - Spark-gui gridded input screen; terrain data.....	27
Figure 13 - Spark-gui gridded input screen; custom input layers.	29
Figure 14 - Spark-gui initialisation model	30
Figure 15 - Spark-gui rate-of-spread models	32
Figure 16 - Speed definition examples.....	33
Figure 17 - Experimental options.....	35
Figure 18 - Spark-gui simulation log.....	37
Figure 19 - spark-batch test result	38

Tables

Table 1 - Field names for general parameters	18
Table 2 - Field names for start parameters.....	19
Table 3 - Field names for output options.....	20
Table 4 - Field names for series input	21
Table 5 - Pre-defined time series names for spark-gui.....	22
Table 6 - Meteorological input layer projections and time conversion parameters.....	23
Table 7 - Field names for meteorological inputs	23
Table 8 - Field names for meteorological inputs	26
Table 9 - Field names for terrain inputs.....	28
Table 10 - Input parameters to define custom layer names	29
Table 11 - Field names for initialisation model.....	31
Table 12 – Experimental options	35
Table 13 - Field names for disruption behaviour module	36
Table 14 - Internal variables used in Spark models.	39

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.

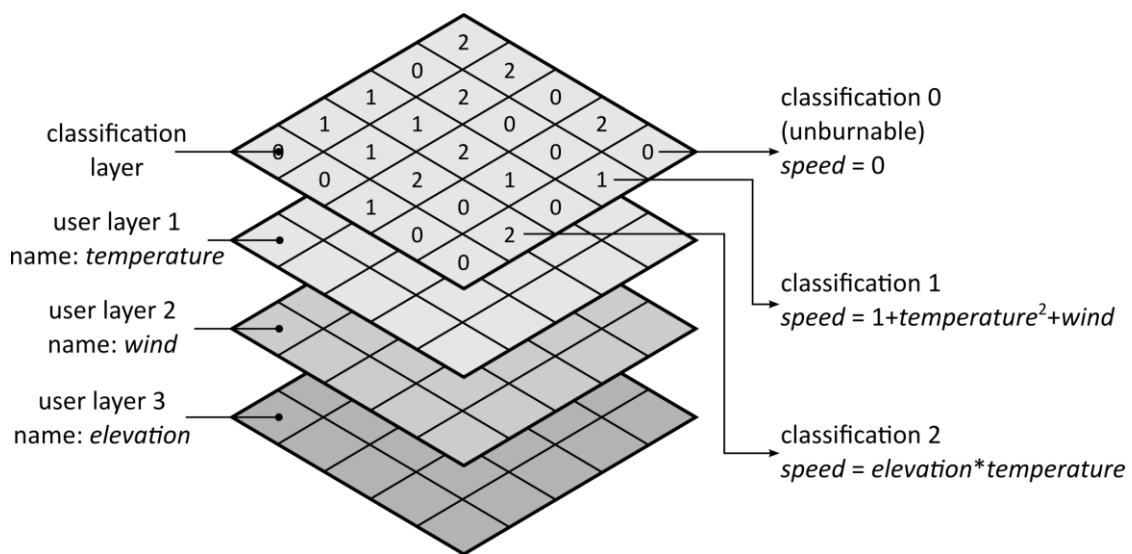


Figure 2 - Schematic of Spark layers

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 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 and a shapefile of isochrones.

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

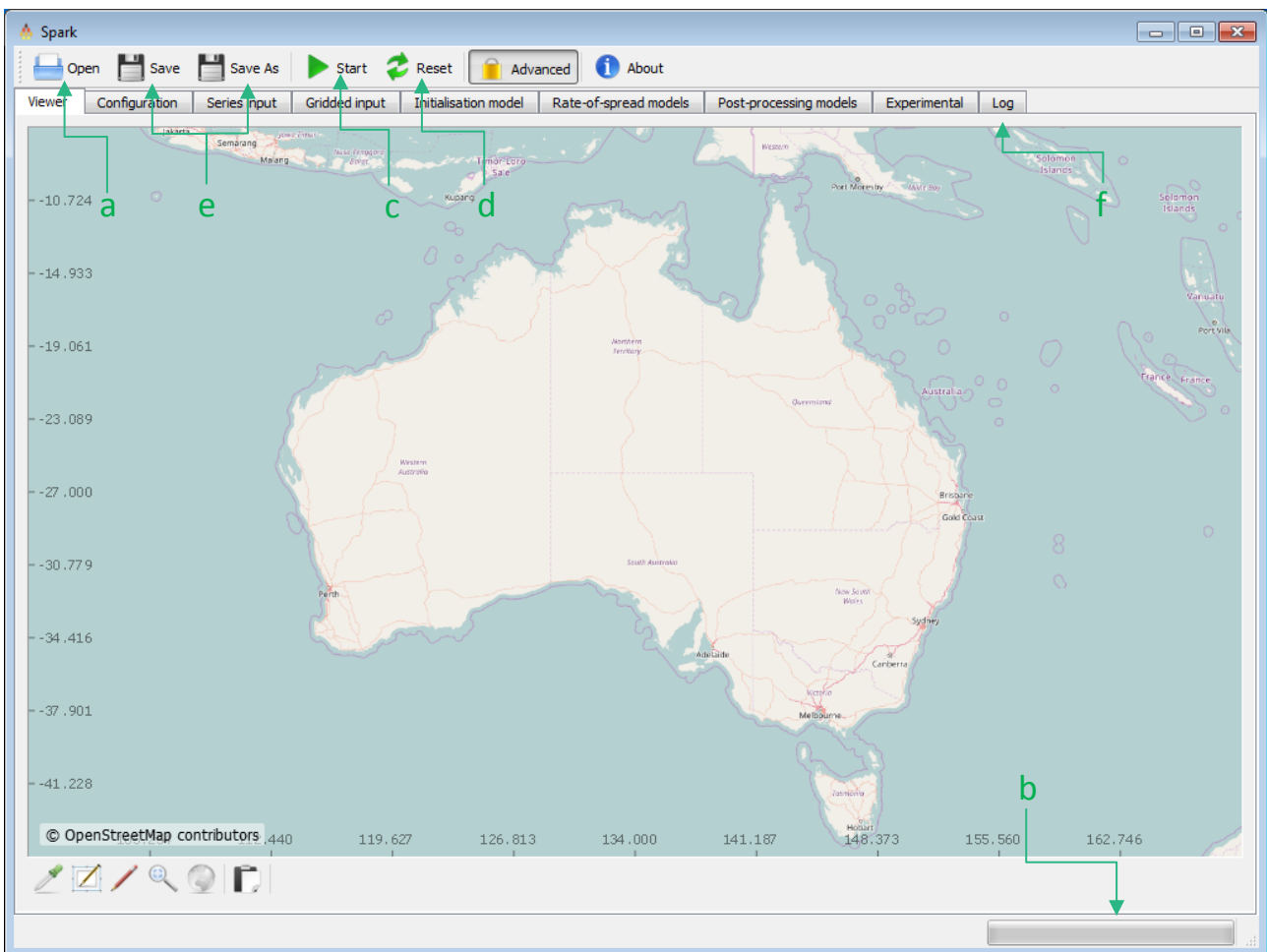


Figure 3 - Spark-gui initial screen.

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.

To load a project, click on *Open* (Figure 3, a) and navigate to `/appdata/spark-gui/proj1`. Select and open the `proj1.xml` file. While the project is loading the status bar (Figure 3, b) will be active. When the project has finished loading the status bar will display 'Ready'.

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 project is executed by pressing the *Start* button (Figure 3, c). Execution can be reset at any time by pressing the *Reset* button (Figure 3, d). Any edits to the project (or a new project) can be saved as a XML project file using the *Save* or *Save As* buttons (Figure 3, e).

When the project is executed the map view will jump to the initial fire location. The simulation should take a few seconds to complete. The default view is a shaded area representing the fire footprint, as shown in Figure 4. Different display layers can be toggled on or off using the named buttons under the map (Figure 4, a). A legend is displayed on the right for the topmost layer.

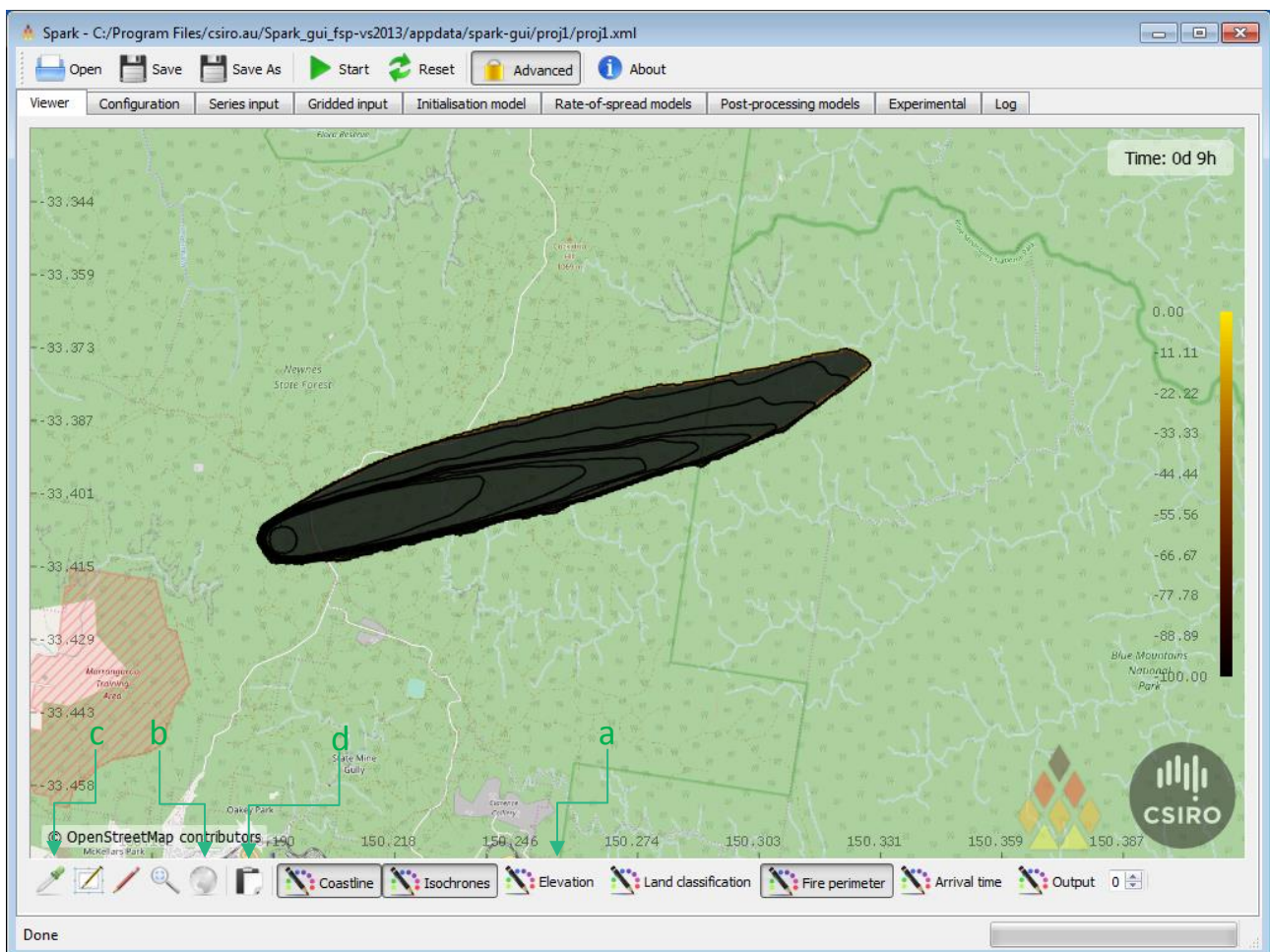


Figure 4 - Spark gui output

To zoom the map to cover the extent of the current layer, use the globe button to the left of the layer controls (Figure 4, b). The current value at the position under the mouse cursor can be displayed for all visible layers by selecting the eyedropper tool (Figure 4, c). 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.

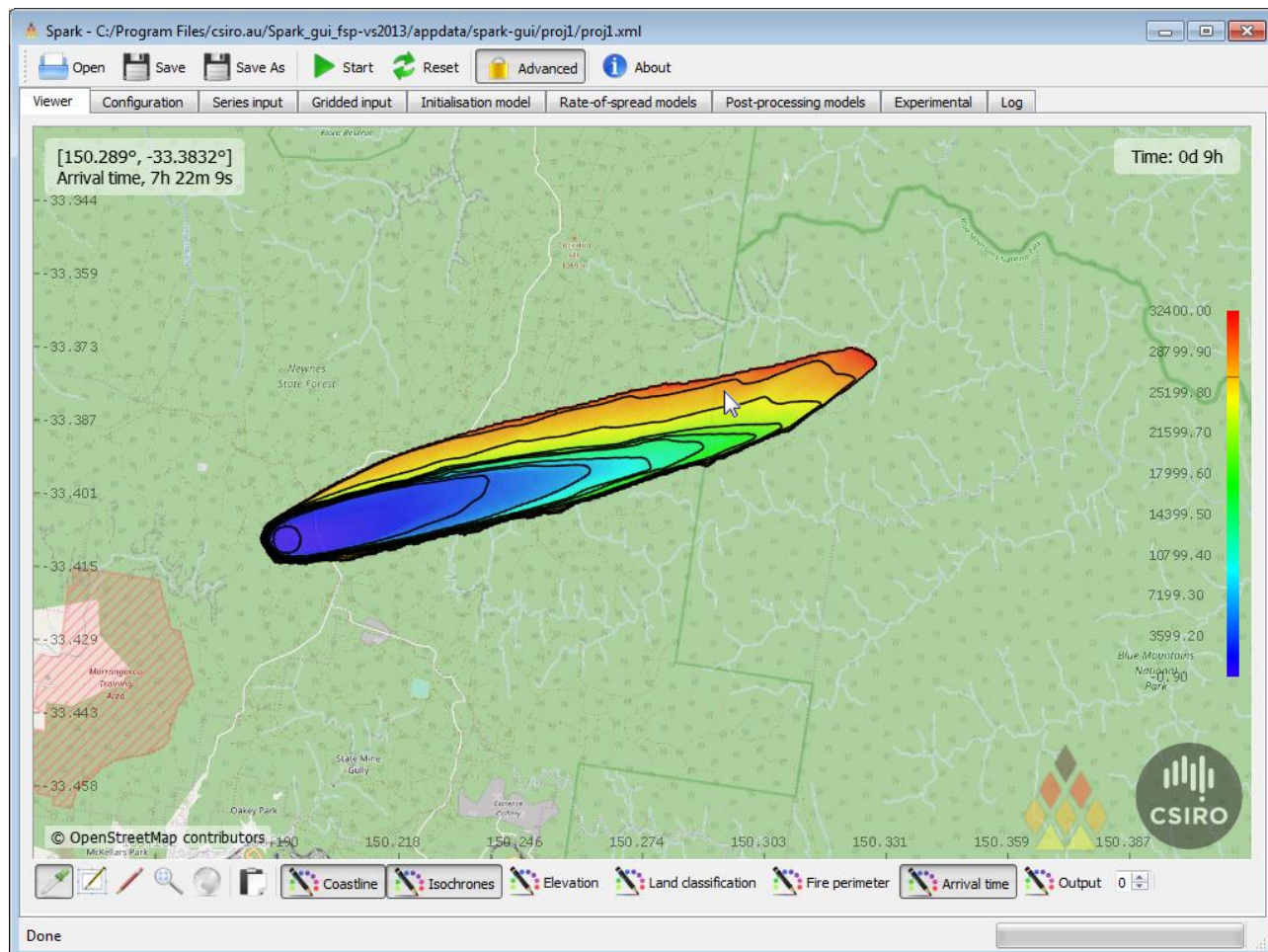


Figure 5 - Spark gui arrival time output

After execution is complete, any output data sets are written to their required locations. A screenshot of the current map view can be captured using the clipboard tool (Figure 4, d).

Errors and warnings

If the status bar displays an error, please refer to the log by selecting the 'Log' tab (Figure 3, f)

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 (Figure 6, a). The configuration screen is shown in Figure 6.

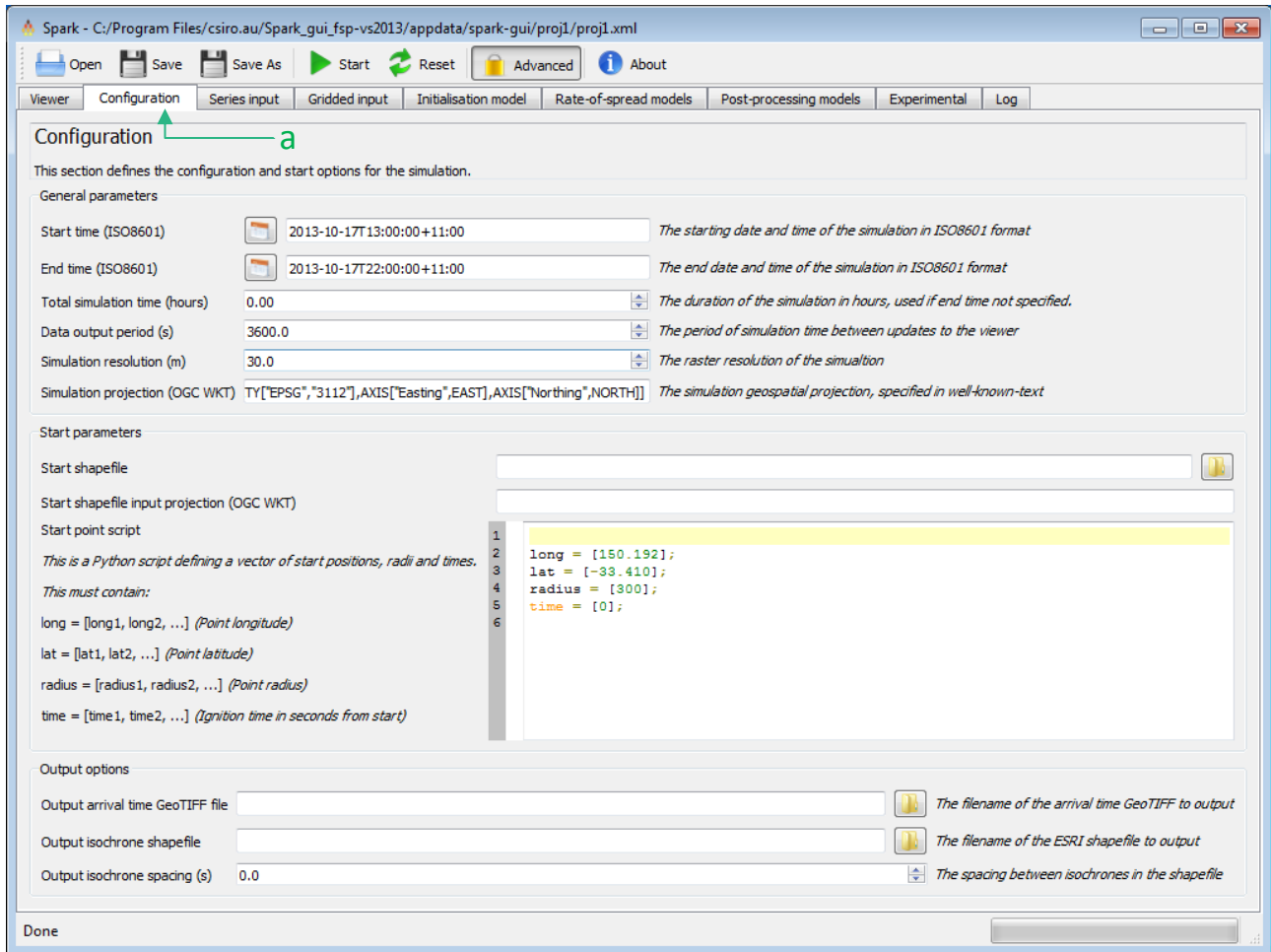


Figure 6 - Spark-gui configuration screen

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 General parameters

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

NAME	DESCRIPTION	TYPE	XML
Start time (ISO8601)	The start time of the simulation in the ISO8601 format . Here the start time is set to 12 pm on the 17 th October 2013 for the UTC+11 time zone.	Text	<i>Start time</i>
End time (ISO8601)	The end time of the simulation in the ISO8601 format . Here the end time is set to 9 pm on the 17 th October 2013 for the UTC+11 time zone.	Text	<i>End time</i>
Total simulation time (hours)	The length of the simulation in seconds. In this example the simulation time is set to zero, meaning the end time string is used.	Number	<i>Simulation duration hours</i>
Data output period (s)	This defines the time period between updates to the map within the Viewer window. This is set to one hour in this example.	Number	<i>Output period</i>
Simulation resolution (m)	The raster resolution of the simulation in meters. 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	<i>Simulation resolution</i>
Simulation projection (OGC WKT)	The projection used for the simulation in the Open Geospatial Consortium Well-Know-Text standard . All output raster layers and shapefiles are written using this projection. Here the projection is set to the Australian Lambert projection.	Text	<i>Simulation projection WKT</i>

Table 1 - Field names for general 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 Start parameters

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 representing point starting locations. If both are defined, the shapefile will be used preferentially and the starting points will not be used.

NAME	DESCRIPTION	TYPE	XML
Start shapefile	An ESRI shapefile (.shp) defining an initial fire perimeter. No shapefile is used in this example.	Filename	<i>Shape file input source</i>
Start shapefile input projection (OGC WKT)	The shapefile projection in the Open Geospatial Consortium Well-Know-Text standard . No shapefile is used in this example.	Text	<i>Shape file input projection WKT</i>
Start point script	<p>This is a Python script defining the latitude, longitude, radii and start times of each point source. This script must contain the following definitions:</p> <ul style="list-style-type: none">• long (point longitude)• lat (point latitude)• radius (point radius)• time (point activation time) <p>These variables are defined as a Python vector and can be generated in any manner within the script. In this example the start location is simply hard-coded into the script.</p>	Text	<i>Start point script</i>

Table 2 - Field names for start 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 options

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	<i>Output raster file</i>
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	<i>Output shape file</i>
Output isochrone spacing (s)	The spacing in time between the output isochrones in the shapefile.	Number	<i>Output isochrone time</i>

Table 3 - Field names for output options

The output files are compatible with all common GIS processing tools, such as QGIS (Figure 7).

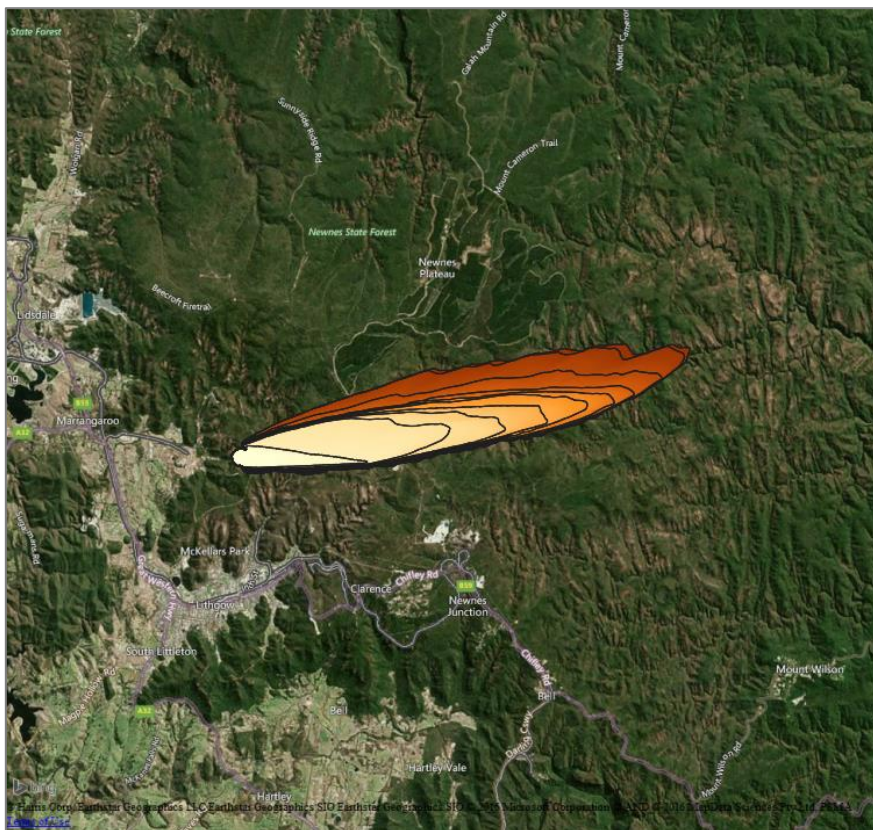


Figure 7 - Output GeoTIFF (shaded by arrival time), and isochrones (black lines)

3.2 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 (Figure 8, a). This window is shown in Figure 8.

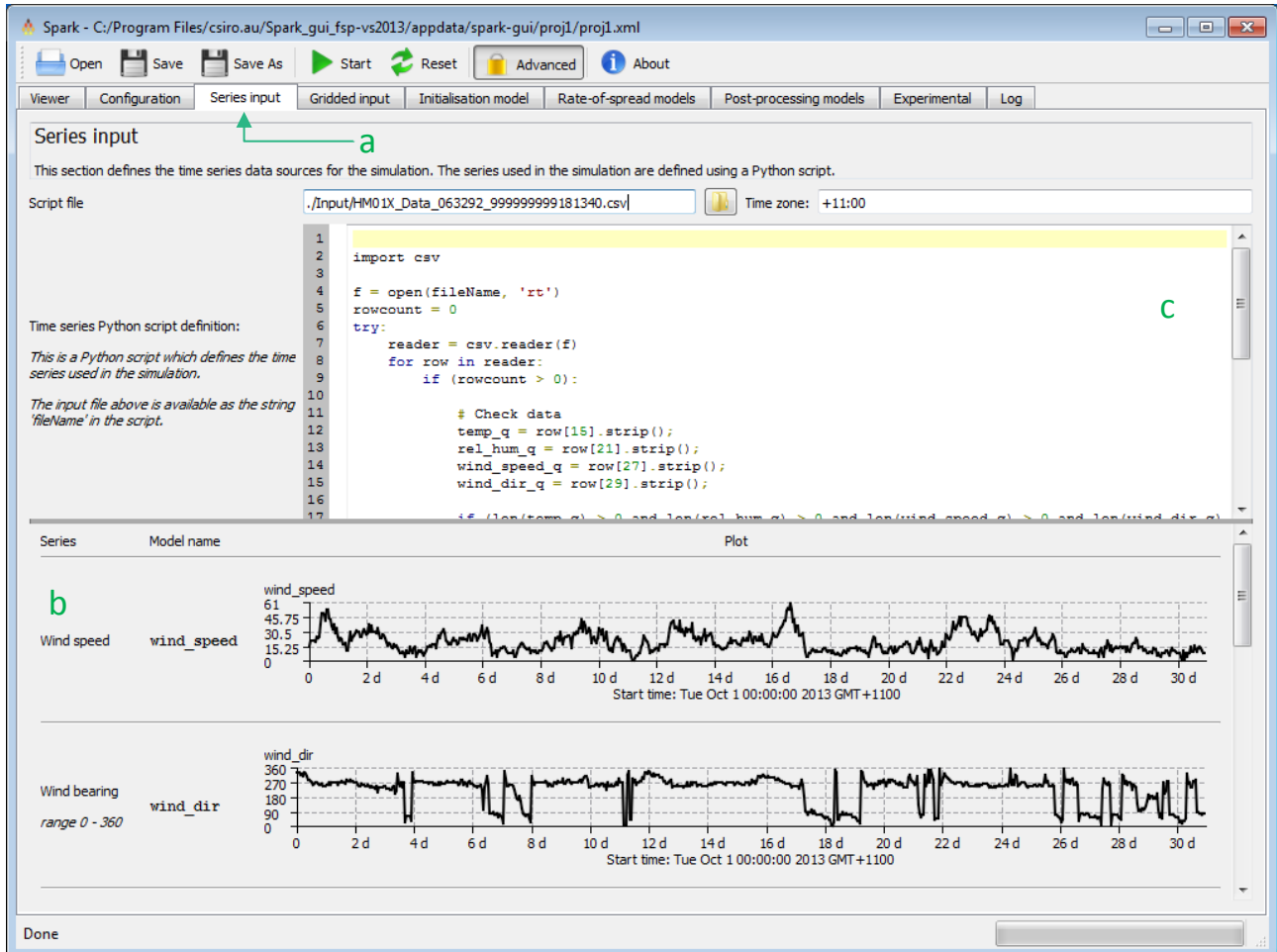


Figure 8 - Spark-gui series input screen

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	<i>Series/Script</i>
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	<i>Series/Time zone</i>
Time series Python script definition	A Python script defining vectors used for time series.	Text	<i>Series/Source file</i>

Table 4 - 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 (Figure 8, b). The spark-gui application pre-defines seven time series, given in Table 5. These time series are populated using the Python script field (Figure 8, c).

NAME	DESCRIPTION
wind_speed	Wind speed.
wind_bearing	Wind bearing in degrees.
temp	Air temperature.
rel_hum	Relative humidity (%).
dew_temp	Dew point temperature.
drought_fac	Drought factor.
curing	Curing parameter.

Table 5 - 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 5. Additionally, it must create a *time* vector which defines the time values in ISO 8601 format. 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.

Time vector

A *'time'* vector must be defined in the Python script. This currently defines the time values for *all* series [as of v0.9.1].

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.

3.3 Gridded input

The application can use gridded input layers. Spark supports two or three-dimensional layers (two spatial plus one temporal) and handles all temporal interpolation. Gridded layers are read in using the 'Gridded input' tab (Figure 9, a).

Gridded input can be restricted to a sub-region, if required. This can be applied using the bounding box configuration (Figure 9, b), 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.

3.3.1 Meteorological inputs

Meteorological input layers are configured in the 'Meteorological inputs' tab (Figure 9, c). 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 (Figure 9, d):

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 .	Text	<i>Gridded/Layer projection WKT</i>
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	<i>Gridded/Start time overwrite</i>
Gridded time conversion coefficient	A multiple to apply to the time-step value for the meteorological raster layers.	Number	<i>Gridded/Time conversion coefficient</i>

Table 6 - Meteorological input layer projections and time conversion parameters

To set individual files, select the 'Source file names' tab (Figure 9, e). 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	<i>Gridded/Wind/Layer direction source file</i>
Wind speed	A raster layer specifying the wind speed within each cell.	Filename	<i>Gridded/Wind/Layer magnitude source file</i>
Temperature	A raster layer specifying the air temperature within each cell.	Filename	<i>Gridded/Temperature/Layer source file</i>
Relative humidity	A raster layer specifying the relative humidity (%) within each cell.	Filename	<i>Gridded/Relative humidity/Layer source file</i>
Dew point temperature	A raster layer specifying the dew point temperature within each cell.	Filename	<i>Gridded/Dew point temperature/Layer source file</i>
Drought factor	A raster layer specifying the drought factor within each cell.	Filename	<i>Gridded/Drought factor/Layer source file</i>

Table 7 - Field names for meteorological inputs

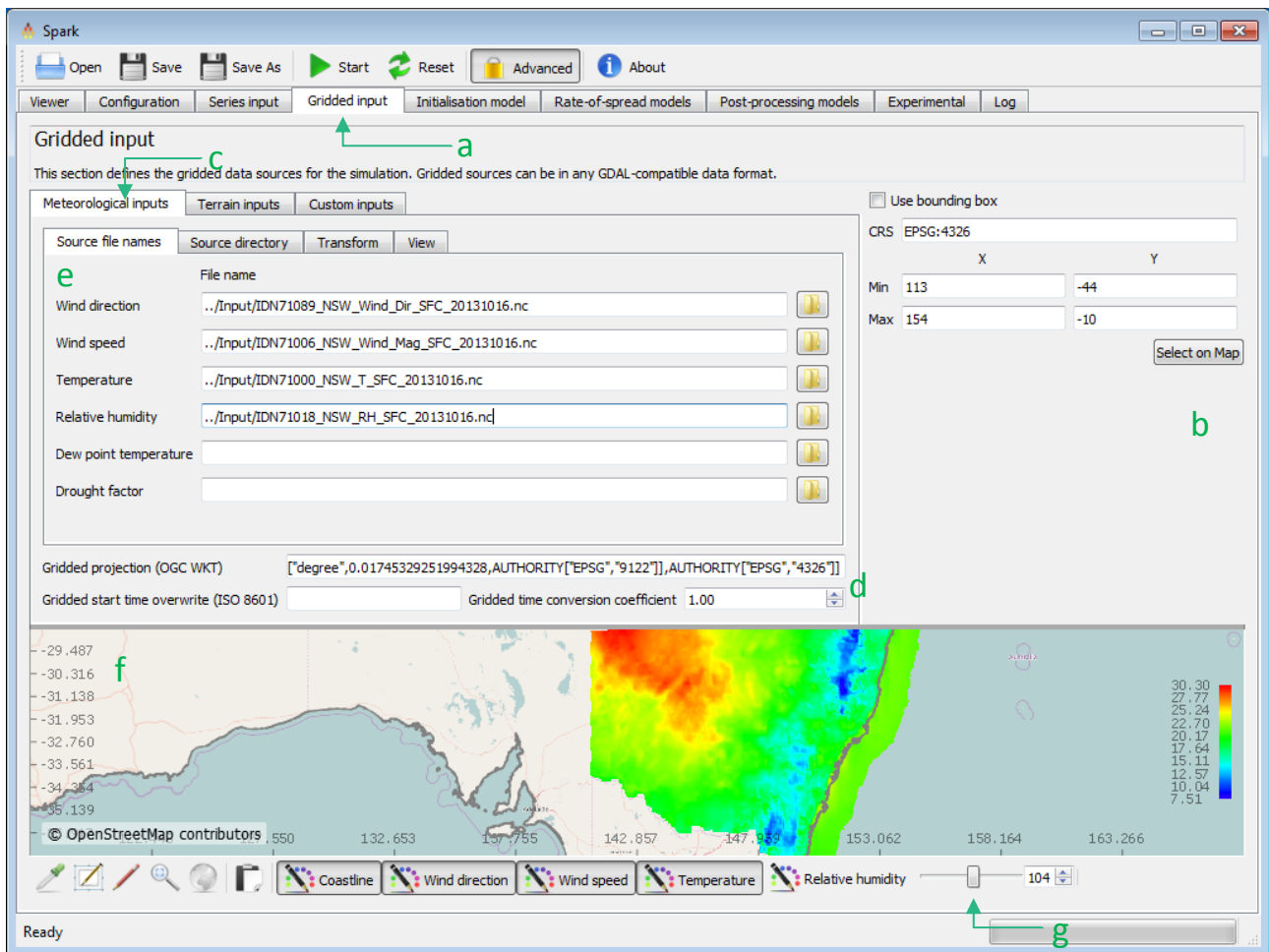


Figure 9 - Spark-gui gridded input screen; meteorological data, file selection

Once loaded, the gridded meteorological input data can be previewed on the map below the tab (Figure 9, f). 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 (Figure 9, g). Individual layers can be turned on or off using the named buttons on this control.

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.

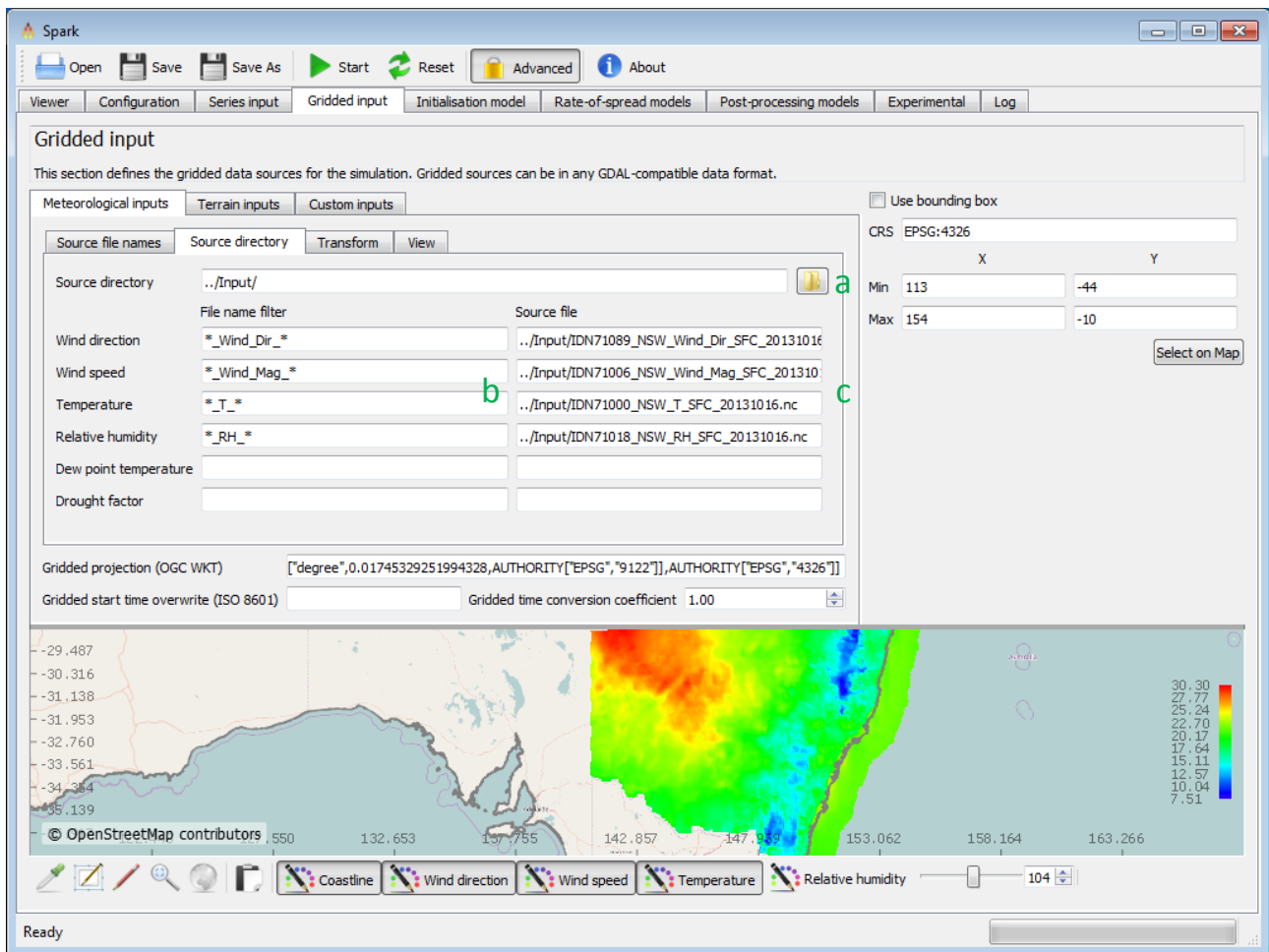


Figure 10 - Spark-gui gridded input screen; meteorological data, directory selection

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 10 the *Input* directory (Figure 10, a) contains four files containing the text *_Wind_Dir_*, *_Wind_Mag_*, *_T_* and *_RH_* (Figure 10, b). Once the directory and a filter is specified, the *Source file* column will update with the generated file path for verification (Figure 10, c).

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.

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	<i>Gridded/Source directory</i>
Wind direction	A text filter specifying the wind direction input file.	Filename	<i>Gridded/Wind/Layer direction source filter</i>
Wind speed	A text filter specifying the wind speed input file.	Filename	<i>Gridded/Wind/Layer magnitude source filter</i>
Temperature	A text filter specifying the air temperature input file.	Filename	<i>Gridded/Temperature/Layer source filter</i>
Relative humidity	A text filter specifying the relative humidity (%) input file.	Filename	<i>Gridded/Relative humidity/Layer source file</i>
Dew point temperature	A text filter specifying the dew point temperature input file.	Filename	<i>Gridded/Dew point temperature/Layer source filter</i>
Drought factor	A text filter specifying the drought factor input file.	Filename	<i>Gridded/Drought factor/Layer source filter</i>

Table 8 - Field names for meteorological inputs

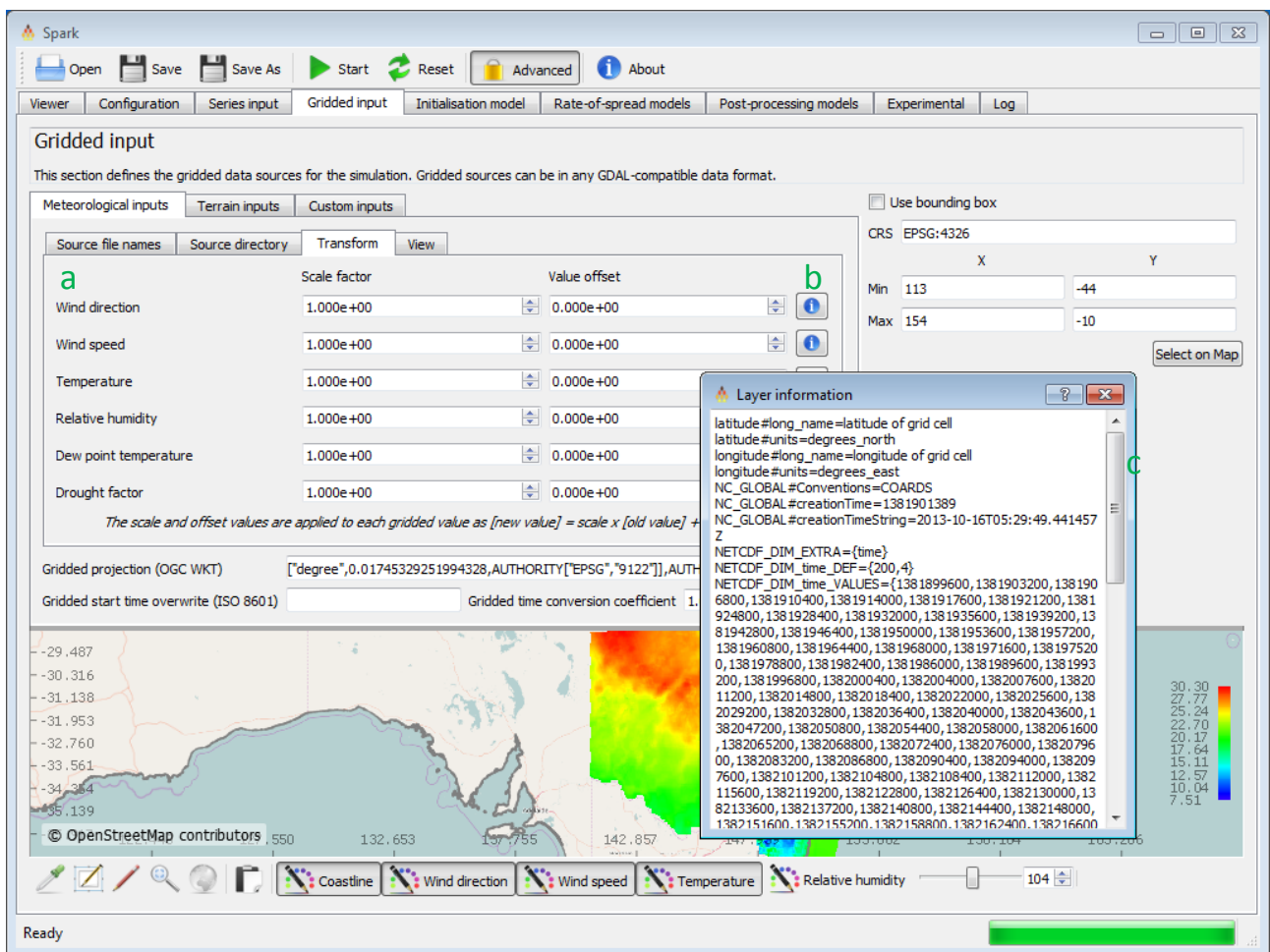


Figure 11 - Spark-gui gridded input screen; meteorological data, data transform

Conversion factors can be applied to the gridded data sets using the 'Transform' tab (Figure 11, 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 11, b). This opens a window showing the GDAL layer metadata which usually contains a description of the data units along with other information (Figure 11, c).

3.3.2 Terrain inputs

Fuel and topographic input layers are configured in the 'Terrain inputs' tab (Figure 12, a). 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.

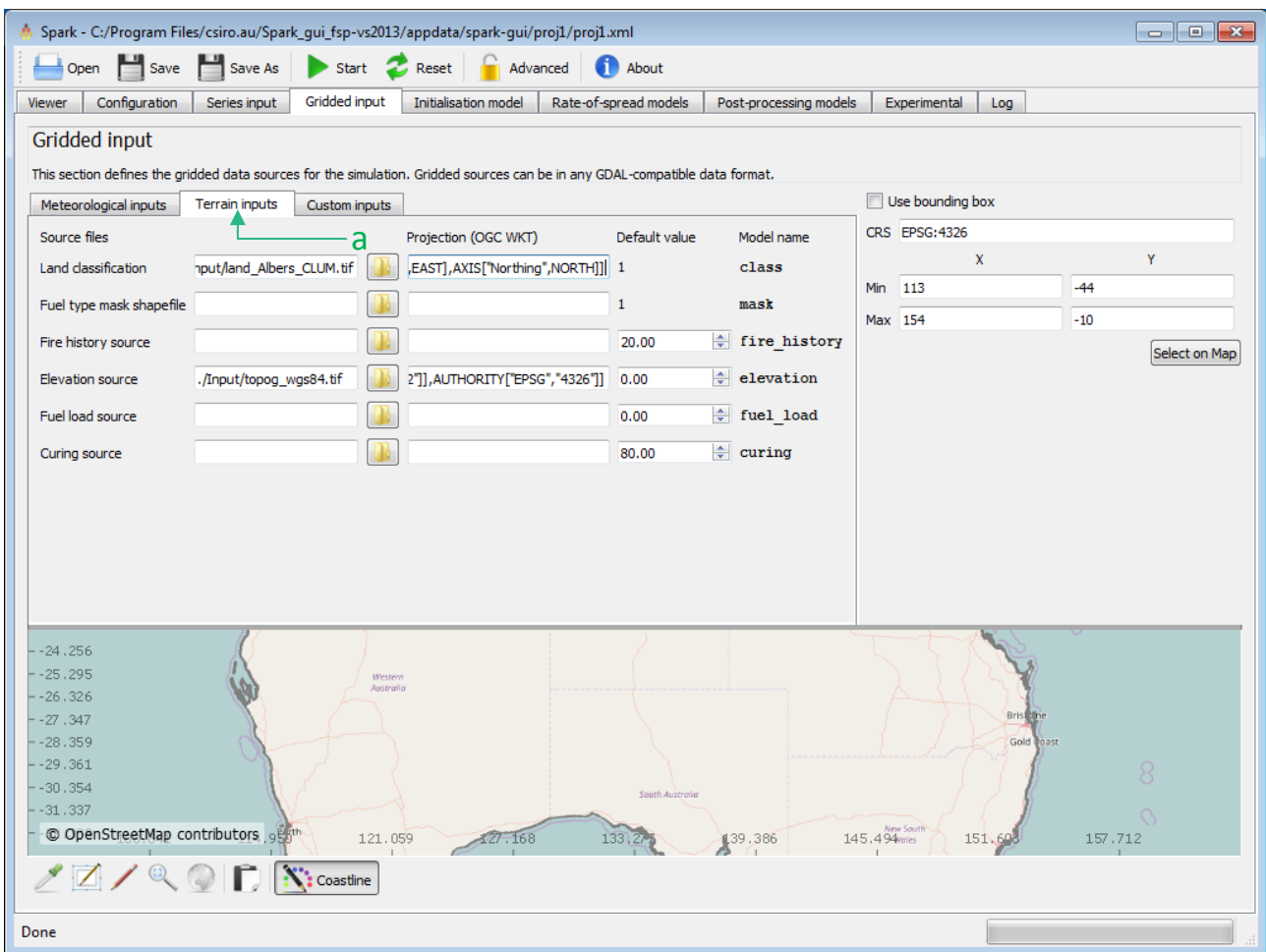


Figure 12 - Spark-gui gridded input screen; terrain data

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
Fuel classification	A raster layer specifying the fuel type within each cell.	Filename	<i>Classification/Layer source file</i>
Fuel classification projection	The projection for the fuel classification layer in the OGC WKT standard .	Text	<i>Classification/Layer projection WKT</i>
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 <code>mask</code> . Outside the shapefile the value of <code>mask</code> is zero. This can be used to define a firebreak region, or a previously burnt area.	Filename	<i>Classification/Shape mask source file</i>
Fuel type mask shapefile projection	The projection for the fuel classification mask in the OGC WKT standard .	Text	<i>Classification/Shape mask projection WKT</i>
Fire history source	A raster layer defining a fire history value, such as a date or time since last burn.	Filename	<i>Fire history/Layer source file</i>
Fire history projection	The projection for the fire history layer in the OGC WKT standard .	Text	<i>Fire history/Layer projection WKT</i>
Fire history default	The default value for the fire history layer.	Number	<i>Fire history/Layer default</i>
Elevation source	A raster layer specifying land elevation with respect to a vertical datum.	Filename	<i>Elevation/Layer source file</i>
Elevation projection	The projection for the elevation layer in the OGC WKT standard .	Text	<i>Elevation/Layer projection WKT</i>
Elevation default	The default value for the elevation layer.	Number	<i>Elevation/Layer default</i>
Fuel load source	A raster layer specifying fuel load values.	Filename	<i>Fuel load/Layer source file</i>
Fuel load projection	The projection for the fuel load layer in the OGC WKT standard .	Text	<i>Fuel load/Layer projection WKT</i>
Fuel load default	The default value for the fuel load layer.	Number	<i>Fuel load/Layer default</i>
Curing source	A raster layer specifying curing values.	Filename	<i>Curing/Layer source file</i>
Curing projection	The projection for the curing layer in the OGC WKT standard .	Text	<i>Curing/Layer projection WKT</i>
Curing default	The default value for the curing layer.	Number	<i>Curing/Layer default</i>

Table 9 - Field names for terrain inputs

Terrain layers

The restriction to five layers is only applicable to the current spark-gui application; the underlying Spark toolkit has no restriction to the number of layers that can be used. New layers will be added in future releases [as of v0.9.1].

3.3.3 Custom inputs

Rate-of-spread models typically require a range of additional input layers, for example, fuel parameters or wind reduction factors. Up to ten 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 'Layer 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 [1-10]
Projection (OGC WKT)	The projection for the fuel classification mask in the OGC WKT standard .	Text	Custom/Layer projection [1-10]
Layer name	The name of a custom layer.	Text	Custom/Layer name [1-10]
Default value	The default value for the layer.	Number	Custom/Layer default [1-10]

Table 10 - Input parameters to define custom layer names

In the *proj1* example, three additional layers are created: *FHSs*, *FHSns* and *Hns* for the dry eucalypt model (Figure 13). These are initialised to zero.

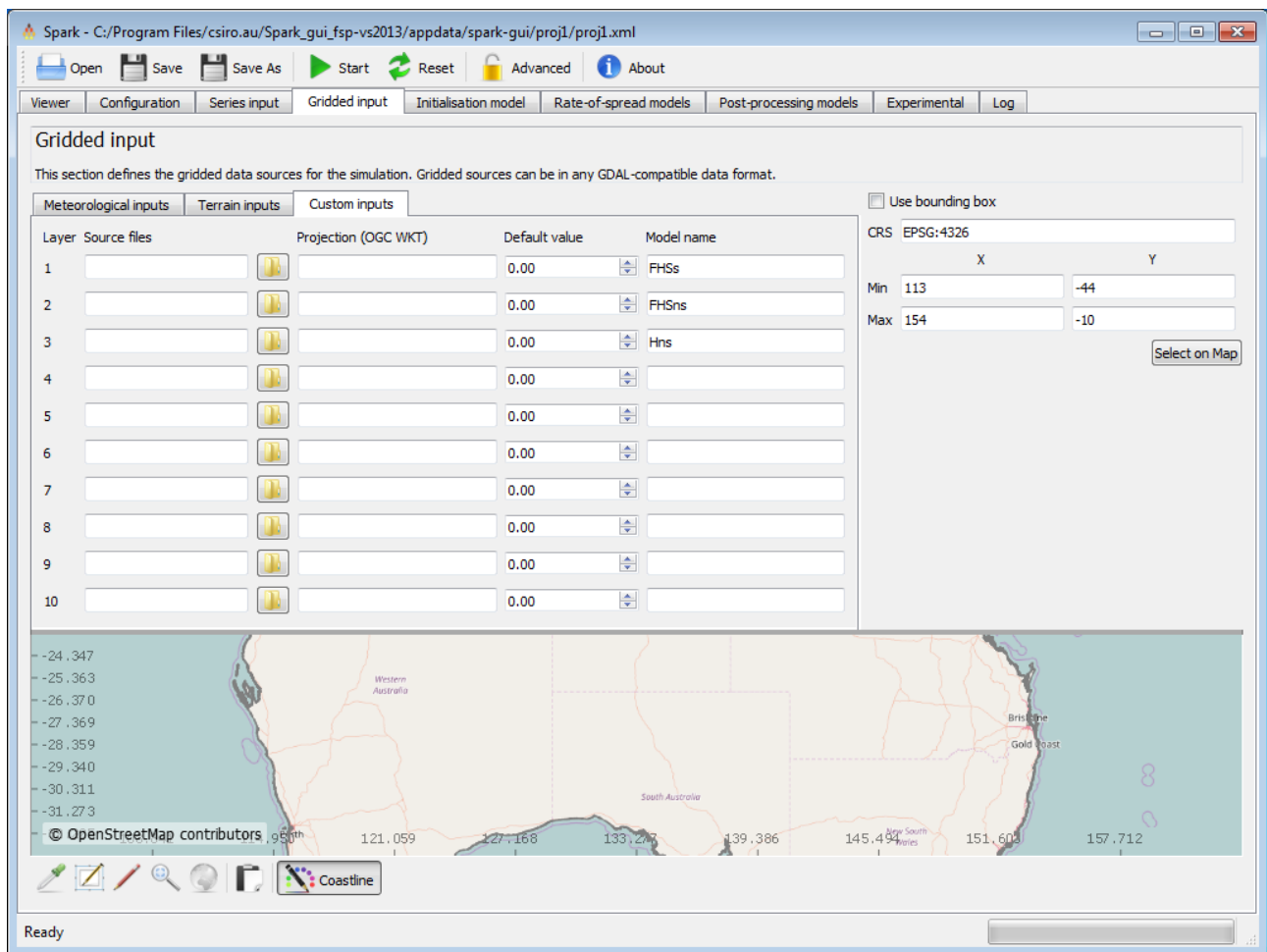
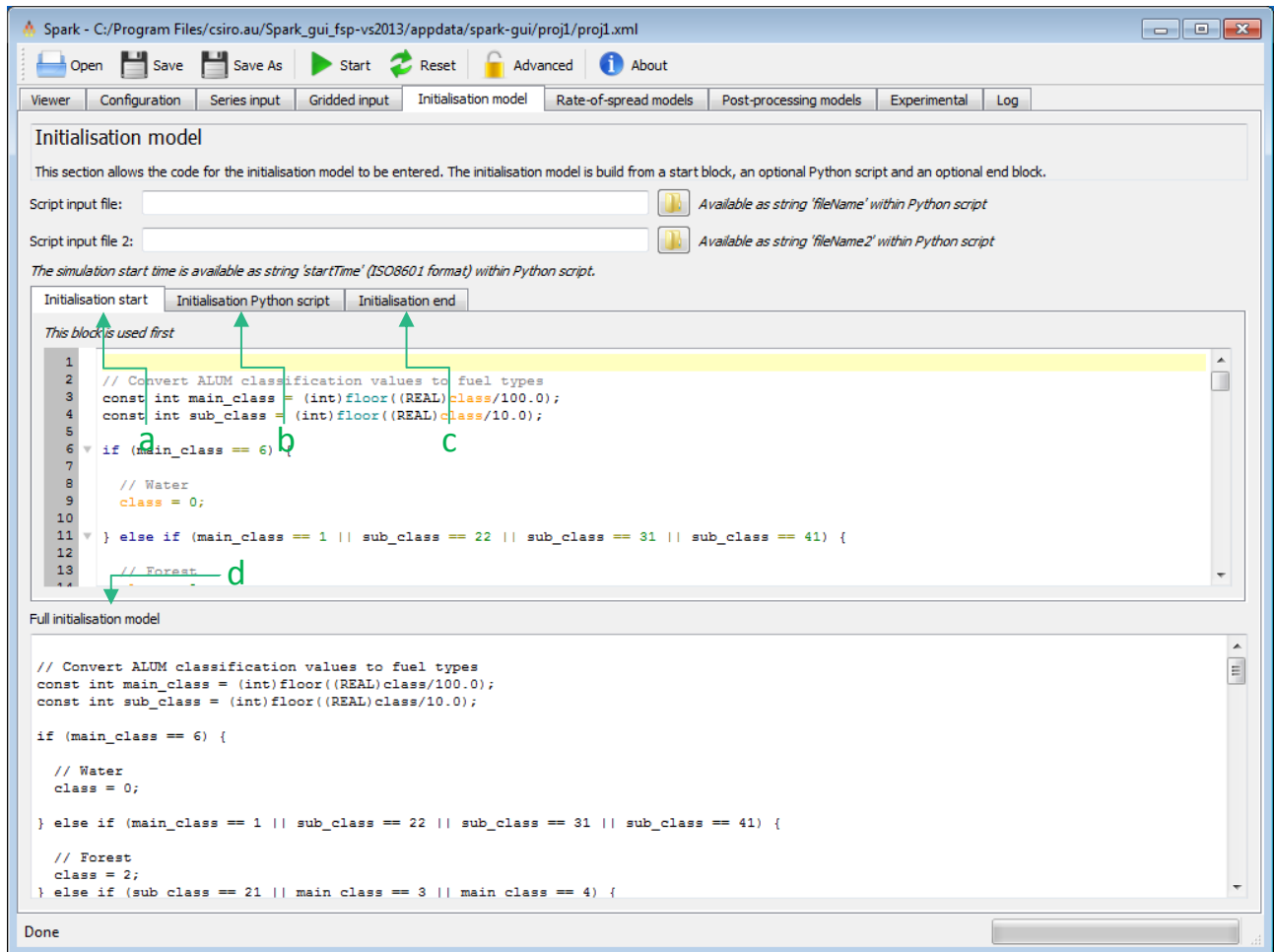


Figure 13 - Spark-gui gridded input screen; custom input layers.

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 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 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 *FHSs*, *FHSns* and *Hns* 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 display at the

bottom of the 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	<i>Initialisation Python input file</i>
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	<i>Initialisation Python input file 2</i>
Initialisation start	A script specifying the starting block of the initialisation model.	Text	<i>Initialisation start string</i>
Initialisation 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.	Text	<i>Initialisation Python script</i>
Initialisation end	A script appended to the end of the start block and the generated Python script.	Text	<i>Initialisation end string</i>

Table 11 - Field names for initialisation model

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 six input tabs within this window corresponding to fuel classifications 1-6. 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.

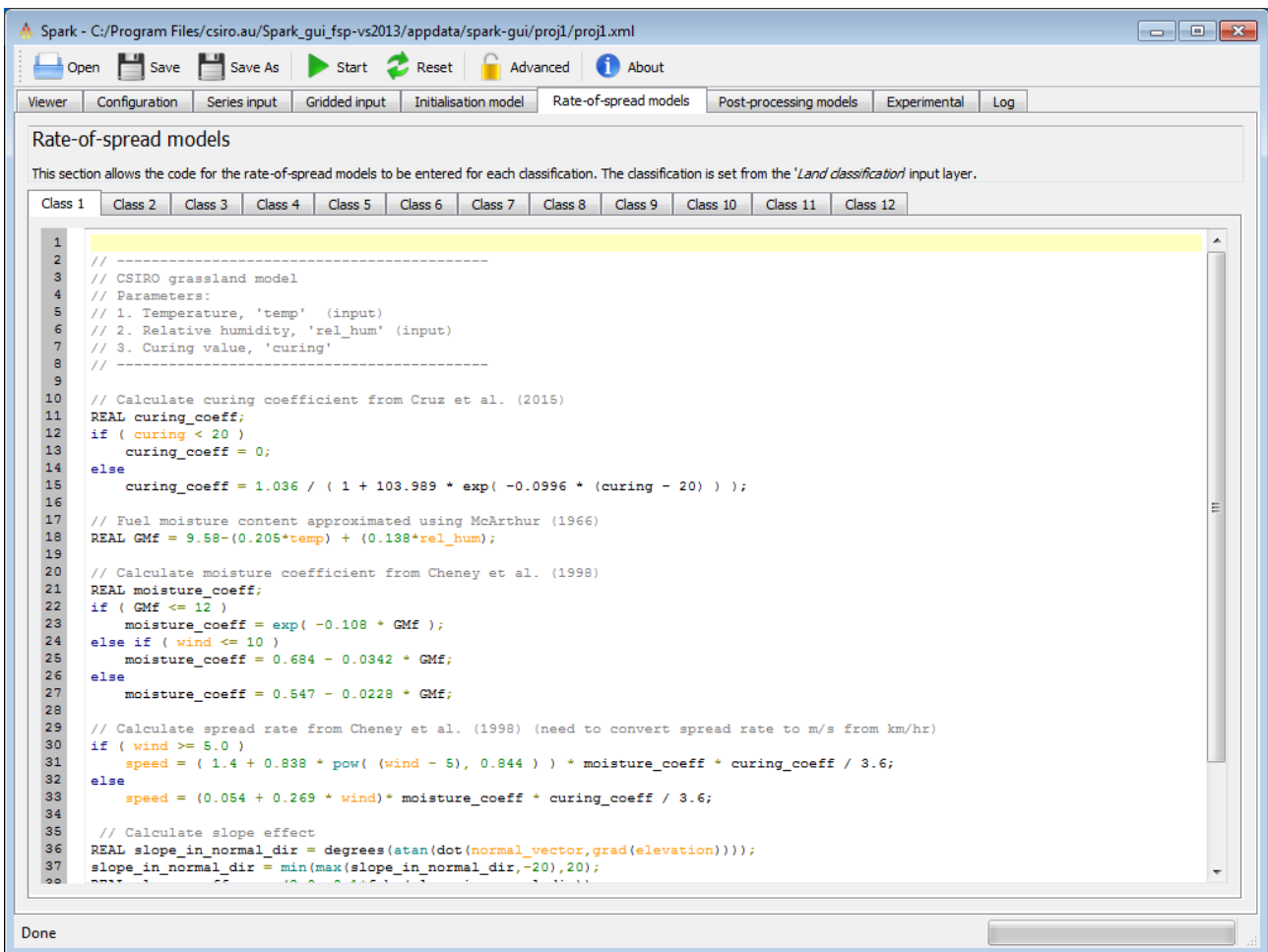


Figure 15 - Spark-gui rate-of-spread models

The grassland and dry eucalypt models are complex models containing a large 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 = (REAL)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 meters, 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 quantity¹ 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 + wind2`, resulting in a pointed fire perimeter.

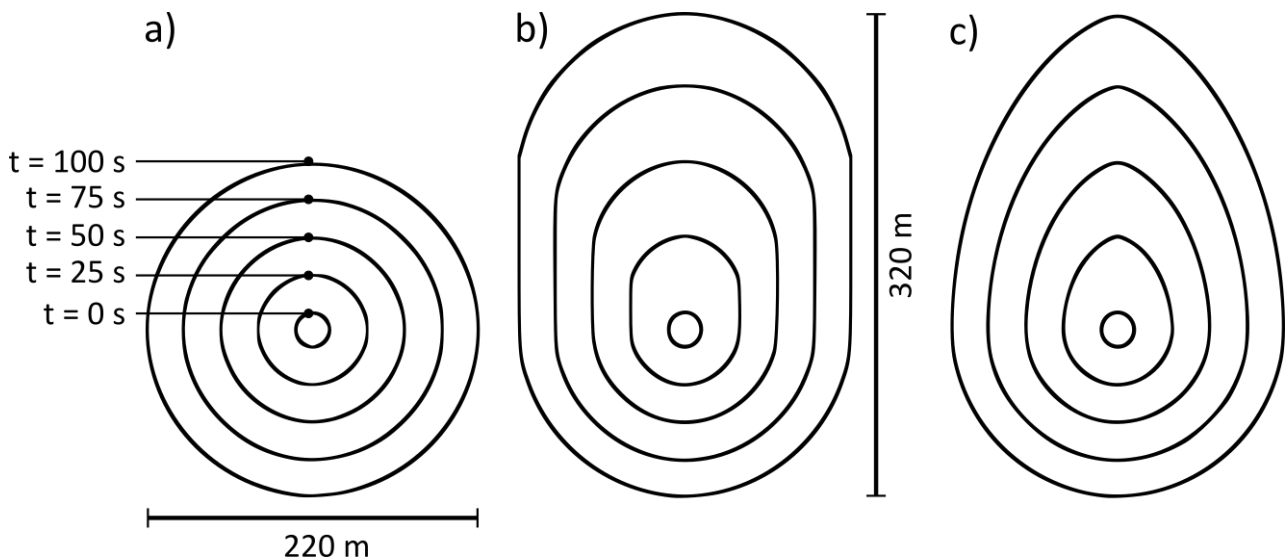


Figure 16 - Speed definition examples

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 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.

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


```
// 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 <= 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 >= 5.0 )
    speed = (1.4+0.838*pow((wind-5),0.844))*moisture_coeff*curing_coeff/3.6;
else
    speed = (0.054+0.269*wind)*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 Experimental options

Spark contains experimental processing and fire behaviour modules in the ‘Experimental’ tab. These options are liable to change or move to different locations in future versions, but are stable and can be used in fire predictions.

The experimental options contain the following inputs:

NAME	DESCRIPTION	TYPE	XML
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	<i>Random seed</i>

Table 12 – Experimental options

Currently, only one fire behaviour module is available: disruption modelling. Spark can read a vector network from a shapefile and use it to impose un-burnable boundaries 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.

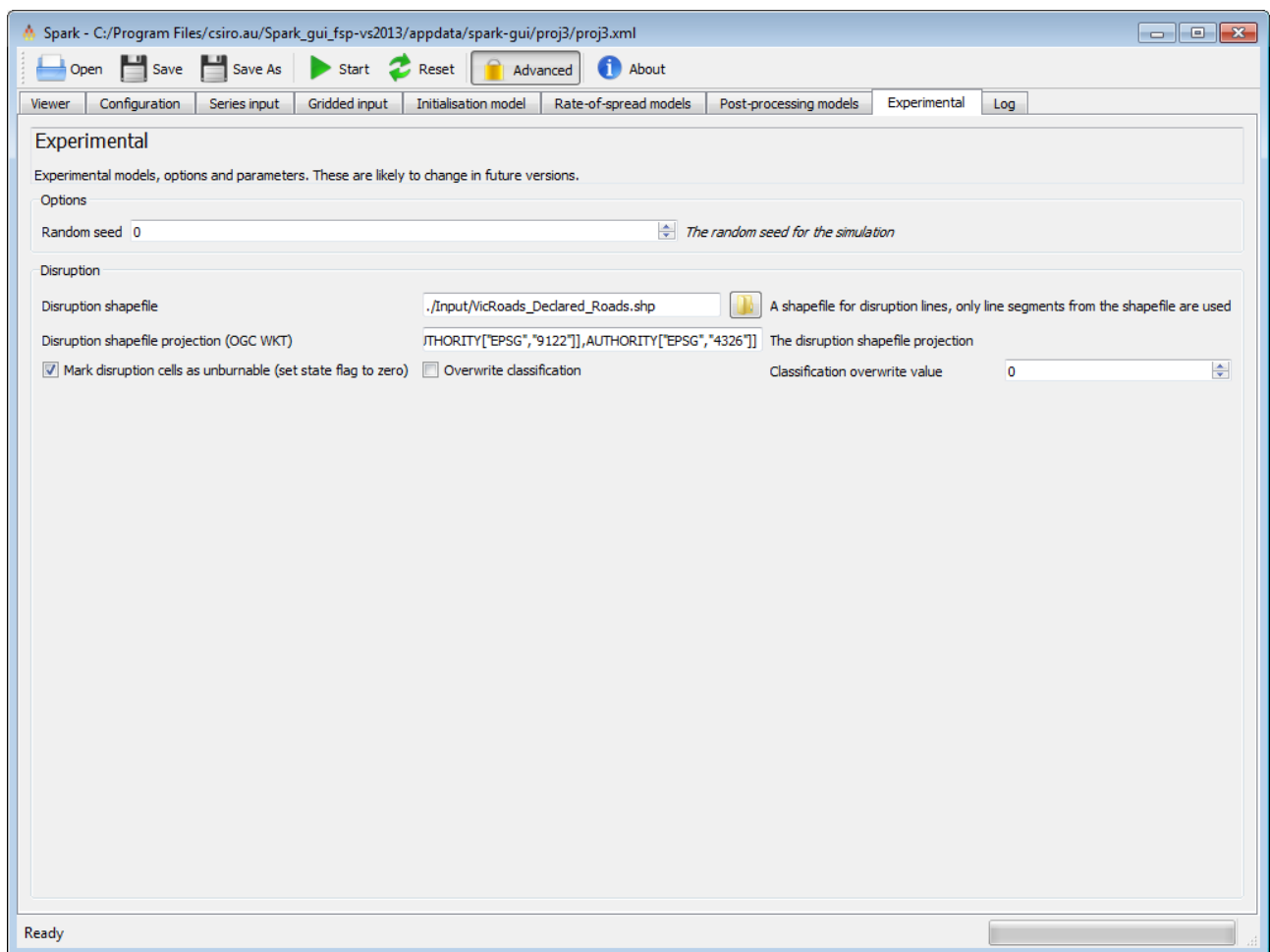


Figure 17 - Experimental options

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 options for the disruption model are:

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	<i>Disruption shape source file</i>
Disruption shapefile projection (OGC WKT)	The projection for the shapefile in the Open Geospatial Consortium Well-Know-Text standard .	Text	<i>Disruption shape projection WKT</i>
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 unburnable.	Checkbox	<i>Disruption clear state</i>
Overwrite classification	Option to overwrite land classification values under the disruption vectors with a new value.	Checkbox	<i>Disruption overwrite classification</i>
Classification overwrite value	The new value for the fuel classification under the disruption vectors.	Number	<i>Disruption overwrite classification value</i>

Table 13 - Field names for disruption behaviour module

3.7 Log

The log provides output and information on the simulation. On start-up similar text to that shown in Figure 18 should appear.

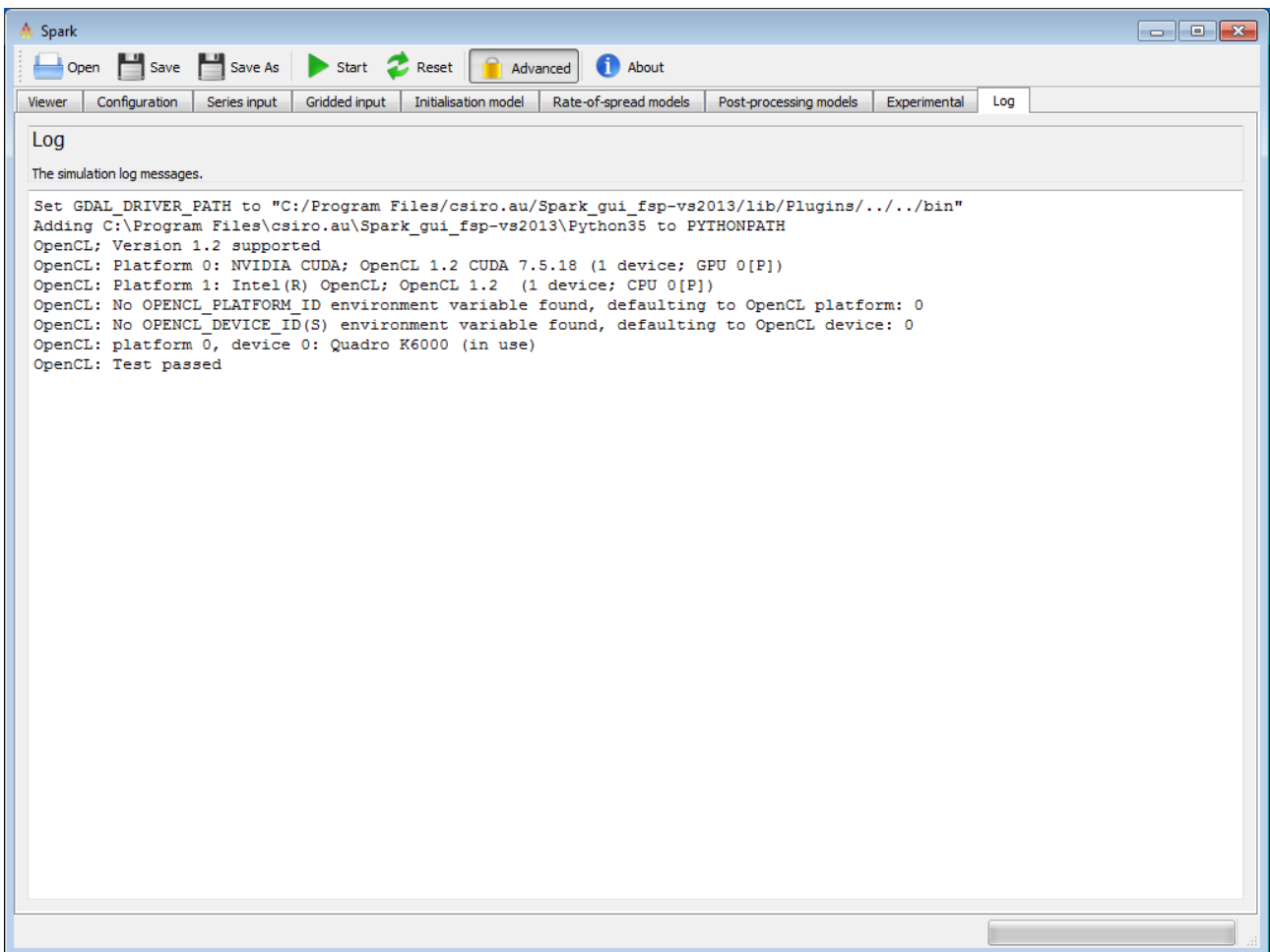


Figure 18 - Spark-gui simulation log

Any 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 19.

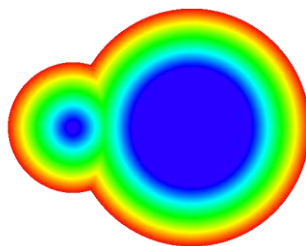


Figure 19 - spark-batch test result

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 '`<`'. 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 14, with the availability of the variable shown in the last column.

VARIABLE	TYPE	DESCRIPTION	AVAILABILITY*
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.	R
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
output0	scalar	Internal user-defined data layer, written to 'output grid' layer 0.	IRP
output1	scalar	Internal user-defined data layer, written to 'output grid' layer 1.	IRP
output2	scalar	Internal user-defined data layer, written to 'output grid' layer 2.	IRP
arrival	scalar	The ignition (arrival) time of the perimeter at the cell (s). No-data values indicate no recorded arrival time.	P
hour	scalar	The current hour in simulation time.	R
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 <i>layername</i> .	IRP
dx(layername)	scalar	The x-spatial derivative of the user-defined layer named <i>layername</i> .	IR
dy(layername)	scalar	The y-spatial derivative of the user-defined layer named <i>layername</i> .	IR
grad(layername)	vector	The gradient of the user-defined layer named <i>layername</i> .	IR

Table 14 - Internal variables used in Spark models.

*** I: Initialization models, R: Rate-of spread models, P: Post-processing models**

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

AT CSIRO WE SHAPE THE FUTURE

We do this by using science and technology to solve real issues. Our research makes a difference to industry, people and the planet.

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

