Gowlland

SELES v10

Spatially **Ex**plicit **L**andscape **E**vent **S**imulator

User Documentation

March 31, 2021

Andrew Fall, Gowlland Technologies Ltd.
andrew@gowlland.ca

Overview

SELES (Spatially Explicit Landscape Event Simulator) is both:

(a) A spatial modelling language to create a discrete spatio-temporal state space, and to define dynamic processes (*landscape events* and *landscape agents*) that navigate through, and modify, this state-space; and

(b) An application (which includes a *discrete-event simulation engine*) to run SELSE models.

SELES models include one or more quasi-independent processes or agents of change, called *landscape events* and *landscape agents*[1]. The SELES modelling language provides a structured, declarative way to specify landscape events, facilitating model development, comparison, modification, and reuse. This high-level language supports the construction of landscape events that are quasi-continuous, periodic, or episodic; with a fixed or variable time step; deterministic, process-oriented, or stochastic; and in which processes operate locally, regionally, or globally and be either spreading or non-spreading.

The purpose of this document is to provide practical user information for the SELES modelling tool (64-bit version designed to run on Windows 10). This document is the primary reference document for builders and users of SELES models, and focuses on aspects common to most model users, with details on some esoteric aspects omitted.

This document has four parts:

- Part 1: SELES conceptual modelling framework
- Part 2: User interface of SELES10 and the landscape event editor (LSEditor)
- Part 3: SELES model structure and model file syntax
- Part 4: Appendixes
    - Appendix 1: SELES expression syntax (functions and expressions used for landscape event and value summary models)

[1] Only landscape events are described in this document. Landscape agents are similar to landscape events, except that they maintain identity of individuals in the process (e.g. individual dispersers in a population), and are defined with some distinct properties and variables.

Table of Contents

PART 1. SELES: Conceptual Modelling Framework

This Part provides an overview of SELES as a tool to build and run spatio-temporal models.

1 The SELES Paradigm

- Landscape change arises as the result of feedback between system state and definable *processes* or *entities.*
- As *agents of landscape change*, processes react to and modify the landscape state in *spatio-temporal contexts.*
- A spatio-temporal context is the set of information (i.e. state variables) available at a particular time and place.
- Contexts provide a general hierarchical framework for describing landscape dynamics.

SELES is:

- a language for:
  - creating a spatio-temporal state-space;
  - defining behaviours to navigate through this state-space; and
  - specifying state change along the way
- and a simulation engine to run models built using that language.

By managing contexts appropriately, models of various forms can be created, including natural disturbance models, habitat supply models, forest estate models, spatial population and meta-population models, and individual-based models.

This is done by using a general process to model *landscape events* (Figure 1), which will be expanded on later in this Part. One of the key strengths of SELES is that it makes a minimum set of assumptions about specific landscape events other than that they recur on the landscape (Return Rate), initiate in one or more cells (or even all cells), that they may or may not "occur" (affect the cell enough to trigger potential spreading), and that they optionally spread to other cells at a specified rate (Spread Rate and Spread Initiation).

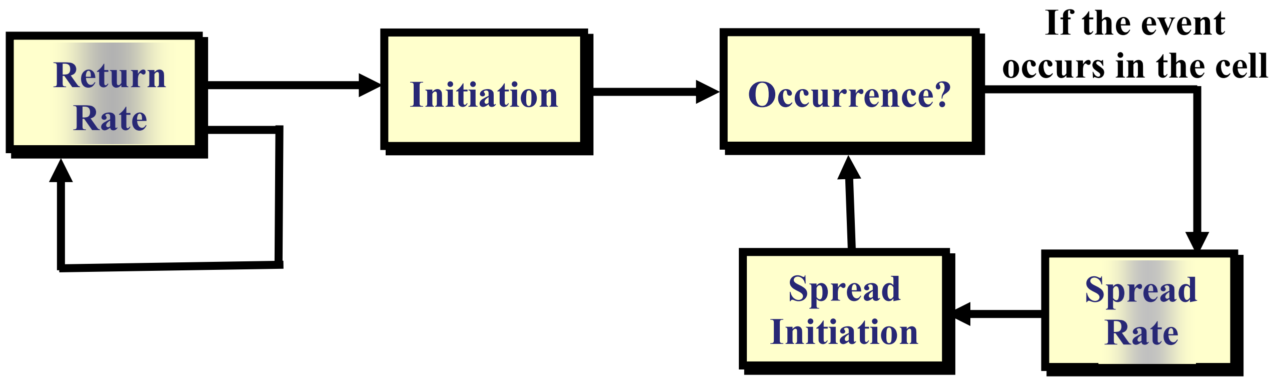


Figure 1. General landscape event process to navigate spatio-temporal contexts. The dark bars represent the passage of simulated time. Arrows represent transitions between process steps.

2 SELES Overview

SELES facilitates the specification and execution of landscape simulation models. SELES is not a landscape model itself, but rather a tool to build models (e.g. the SELES Spatial Timber Supply Model (STSM)). A SELES model consists of a set of raster layers and global variables that define the model state, and a set of landscape events that define model dynamics.

Each dynamic SELES model has two general components:

1. Initial state: This is the set of initial landscape conditions before a simulation begins, and may include spatial and non-spatial (global) variables and constants. Some state variables may be dynamically modified during a simulation, while others may be static. For example, a model of forest change may include forest stand age, which may change over time, and elevation, which remains constant. In an erosion model, however, elevation may be dynamic.

2. Landscape Events: Each process of landscape change (e.g. forest succession, fire, timber harvesting) is specified as a semi-independent sub-model called a *landscape event*. The definition of a landscape event specifies its recurrence frequency and spatial domain (i.e., when and where the event should occur), how it spreads and its effect on model state (Figure 1). All feedback mechanisms between processes are accomplished through state changes to raster layers and global variables, with no direct inter-event communication.

Any number of rasters may be incorporated in a model, but they must represent the same geographic location and resolution. For example, raster layers that represent vegetation cover, elevation, or time-since-disturbance are commonly used as state variables.

SELES provides a declarative modelling language for describing the properties of landscape events, and the simulation engine interprets and executes these specifications, resulting in changes over simulated time to the landscape state. The relationships between the components of a SELES model are shown in Figure 2, which illustrates a simple model with 3 rasters and two landscape events. The landscape events do not interact directly with each other, but rather interact through changes to the state variables.

SELES models are usually stochastic, but can range from completely random to entirely deterministic. Stochastic models may require a Monte Carlo approach, since any single run will represent only one possible sequence of change. SELES models may mix random and deterministic components. For example, a theoretical or empirical model can be prototyped quickly, primarily employing statistical representations of most processes. As greater understanding of the system is developed, this statistical behaviour can be progressively replaced by a more mechanistic approach to describing each process. Earlier models can then be used as benchmarks from which to judge the increase in predictability introduced by the subsequent refinements.

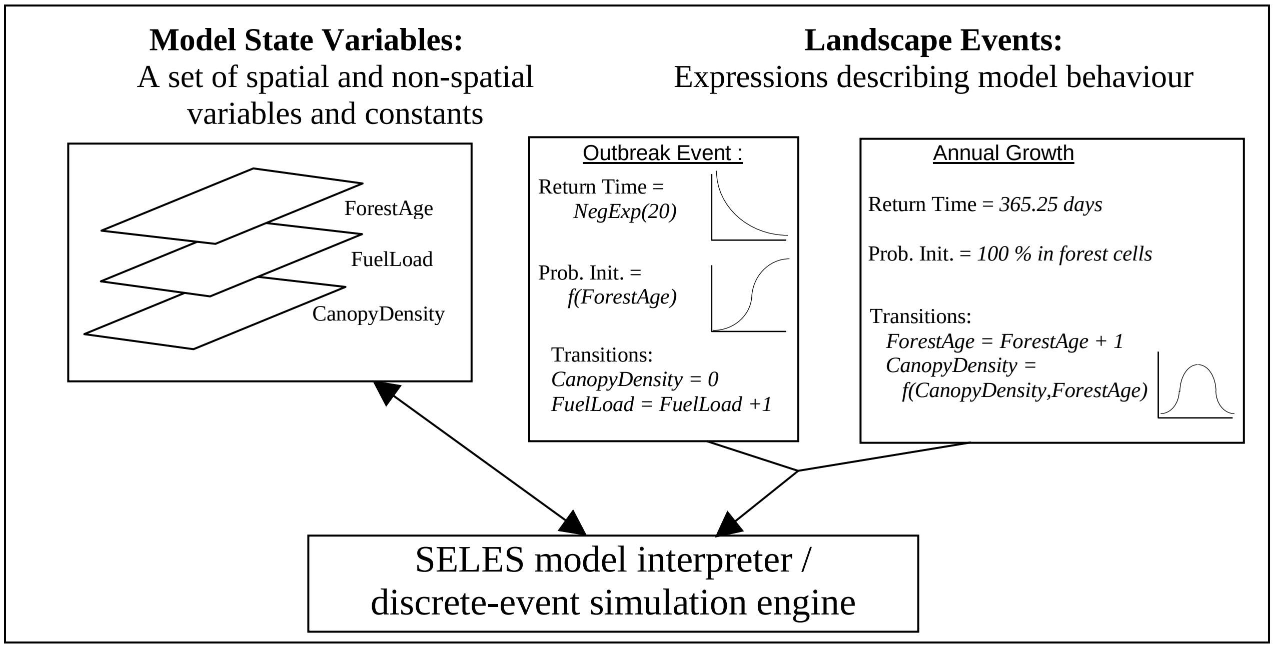

Model State Variables:
A set of spatial and non-spatial variables and constants
Landscape Events:
Expressions describing model behaviour
ForestAge
FuelLoad
CanopyDensity
Outbreak Event :
Return Time = NegExp(20)
Prob. Init. = f(ForestAge)
Transitions:
CanopyDensity = 0
FuelLoad = FuelLoad +1
Annual Growth
Return Time = 365.25 days
Prob. Init. = 100 % in forest cells
Transitions:
ForestAge = ForestAge + 1
CanopyDensity = f(CanopyDensity,ForestAge)
SELES model interpreter / discrete-event simulation engine

Figure 2. Components of a SELES model with three spatial variables and two landscape events, as interpreted and executed by the SELES discrete event simulation engine.

Landscape events create an inherently modular structure, both between events and in their internal specifications, which facilitates description and verification. Table 1 gives an example for a simple *Fire* event. Each landscape event is a semi-independent process model, and thus provides substantial opportunity for re-use and adaptation.

Table 1: Informal specification for a simple Fire event.

Layers required: ForestAge
Global variables: WindSpeed and WindDirection
Cluster variables: FireSize

ReturnTime = 5 years, on average, distributed as by negative exponential
ProbInit = zero in unforested cells, linearly increasing with ForestAge otherwise, with an average value of 0.001 in young forest and 0.01 in very old forest.
NumClusters =1 [only one fire per fire year]
    FireSize = 200 cells, on average, distributed normally with std. dev. = 20
Transitions = if (FireSize > 0) [burn only if this fire still needs to be bigger]
    FireSize = FireSize -1
    ForestAge = 0
SpreadTime = function of WindSpeed.
SpreadLocation = all 8 adjacent cells
SpreadProb = a probability of one in WindDirection, and linearly decreasing to a probability of zero in the opposite direction.

2.1 SELES Discrete Event Simulation Overview

All SELES models are executed by the discrete event simulation engine, which is the heart of the SELES application. The simulation engine uses a priority queue to maintain scheduled events in chronological order. The current time is always the time of the current event (i.e. the one at the head of the queue).

Before a simulation is started, the initial events are loaded on the queue. These may include the first step for process models (landscape events), scheduled output commands, etc.

When the simulation is started (at time 0), the engine starts processing events in order of time. Scheduled events are removed from the queue and executed before moving to the next event on the queue. This event behaviour may alter state variables and schedule one or more events to occur in the future or at the same time step (Figure 2).

Events are added to the queue in the scheduled order of time. If more than one event is scheduled at a given time, these are processed in the order in which they were scheduled.

After all the events at the current time are completed, the time advances to the time of the next event on the queue. In this way, time progresses in discrete steps, although each step can have a different duration and multiple events may be scheduled for the same time.

This process continues until the event queue is empty or, more usually, the time duration of the simulation is reached.

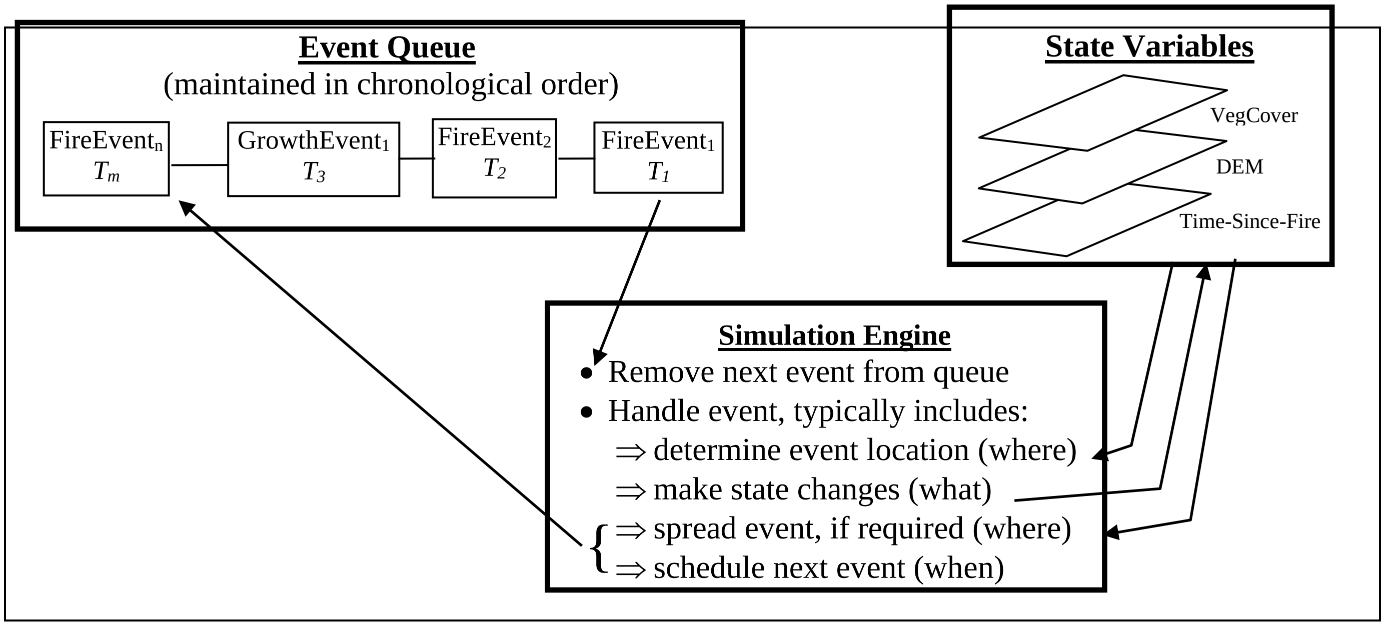

Event Queue
(maintained in chronological order)
FireEvent$_n$ T_m
GrowthEvent$_1$ T_3
FireEvent$_2$ T_2
FireEvent$_1$ T_1
State Variables
VegCover
DEM
Time-Since-Fire
Simulation Engine
• Remove next event from queue
• Handle event, typically includes:
⇒ determine event location (where)
⇒ make state changes (what)
⇒ spread event, if required (where)
⇒ schedule next event (when)

Figure 3. The "event loop" performed by the SELES discrete event simulation engine to process events in chronological order during simulation. The specific behaviour of an event is determined by the event specification.

This process needs to be efficient as there may be millions of events on the queue (e.g. all cells on the active fronts of multiple spreading fires), at many different time points. In SELES10, the event queue was re-implemented using red-black trees to increase efficiency.

2.2 General Inputs and Outputs

The primary inputs and outputs consist for raster grid files and tab-delimited tables. Tabular model output may or may not be dynamic (e.g. mean area burned over a run, or stand age distribution each decade). Output tables can, for example, be loaded into a database or spreadsheet for post-simulation analysis and exploration, or as input to a subsequent SELES model. Spatial outputs also may or may not be dynamic. For example, the model may save a raster grid generated as the final result of a simulation (e.g. a modelled future road network). A model may also schedule periodic output of rasters to produce raster time-series (e.g. Stand Age each decade).

2.3 General Assumptions and Limitations

SELES models can range from simple to complex, but require careful thought to produce a model that can be clearly interpreted. In addition to the caveats that go along with any modelling, the following limitations should be considered when modelling with SELES:

1. Some error checking is provided to ensure that parameters are within bounds, etc. However, there is no checking to ensure that a model is doing anything sensible. Models can be run on a "test suite" of data for which the model outcome can be easily verified. These should include, but not be limited to, testing boundary conditions for the model (e.g. the simplest possible initial conditions for the model), and testing specific behavioural characteristics for each landscape event in the model. It is useful to set up a model to produce output specifically designed to verify model behaviour.

2. SELES does not interpret the values in state raster layers, and care should be taken to apply the correct interpretation and units for these values. This is particularly important for categorical data, where the numerical value in a cell indicates a class not a magnitude. Constants, especially legends associated with raster files, are useful to ensure that the correct index is used for each category.

3. Raster cell values may be "undefined" (e.g. -1) to delineate areas that have no data or are not of interest (e.g. out of the study area). SELES makes no *a priori* assumption as to how to interpret raster values. However, landscape events can easily avoid operations in cells with "undefined" values. Use of a binary study area layer is often convenient.

4. All raster layers used in a single model must have the same grain (cell size) and extent (number of rows, cols). SELES makes no assumptions about the grain of raster cells. It is up to the modeller to determine the appropriate scale and behaviour at that scale for a model. Note: SELES does provide some capability to rescale layers and to align two layers so that their georeferencing information matches.

3 Spatio-Temporal State Space

A *spatio-temporal state space* (or just, state space) is the set of all spatial and non-spatial variables and constants (including the range of values and spatial bounds) combined with a time unit. Global and spatial constants, by definition, do not vary over time, but spatial constants may vary over space (e.g. elevation) and global variables may var over time (e.g. cumulative area burned). Spatial variables may vary over both space and time (e.g. tree stand age).

3.1 Defining a Spatio-Temporal State Space

A SELES dynamic model (defined in a .sel file) defines a spatio-temporal state space as well as the set of landscape events that can change state variables over time in a simulation.

A SELES state space can consist the following types of elements:

- **Global constants**: spatially and temporally invariant (e.g. the maximum stand age that may be represented could be defined using a global constant *MaxStandAge*).

- **Global variables**: may vary over time, but not space (e.g. the rotation for a wildfire model may be defined by a global variable *FireRotation*). These generally define broad-scale state information that may by accessed by any landscape event. For example, a global variable called *WindDirection* might store the prevailing wind direction. A *Wind* event might change the *WindDirection* during the simulation, whereas events called *Fire* and *WindThrow* might use the *WindDirection* to influence their behaviour. In addition, global variables appear on the user interface in the simulation dialog.

- **Spatial constants**: may vary over space, but not time (e.g. elevation may be represented using a spatial constant *Elevation* – assuming that geologic processes that affect elevation are not being modelled).

- **Spatial variables**: may vary over space and time (e.g. tree stand age may be represented using a spatial variable *StandAge*).

Parameters can generally be viewed as any input to set the initial conditions of a model, although most often these are considered to be certain elements of the state-space that are designed to be modified in different scenarios.

Global constants and variables may be single value, one-dimensional arrays (also called vectors) or two-dimensional arrays. Arrays are also called matrices.

3.2 Spatio-Temporal Contexts

At its core, SELES is a tool to define a spatio-temporal state space, and models of processes (*landscape events*) that navigate this state space over time, making changes to the state variables. The simulation engine takes care of carrying out this defined process.

The time and place of landscape change is abstractly called a *spatio-temporal context* (or just context). Examples of possible contexts include:

- The time and place that a stand-replacing fire burns a stand (e.g. resulting in stand age being set to 0, among other changes).
- The time and place that a road is build (e.g. resulting in changes to road and land cover information).
- The time and place that a logging cutblock is initiated (e.g. resulting in changes to target block size).
- The time at which fire rotation changes due to climate change (in this example, there is no "place").
- The time at which spatial or non-spatial info is output to file (i.e. contexts also include practical aspects of model control).

In a spatio-temporal context, the model has access to variable values unique to that time and grid cell location (e.g. elevation and stand age at that place). Variables and constants at that time and place can be used to modify landscape event behaviour, and the landscape event can modify those variables.

In practical terms, a spatio-temporal context is the set of variables available in a given section of a landscape event.

The general process algorithm of landscape events from Figure 1 can be refined to identify in broad terms the contexts associated with each step of the process (Figure 4):

- Simulation start-up (Initial State) is in a *Global Context* (i.e. there is no spatial location and no active process).
- When events recur on the landscape (Return Rate), this is still in a non-spatial context, but there is now an active event instance, which has not yet initiated in any cells. This is called an *Event Instance Context.* The Return Rate specifies the time passed between recurrence of the event.
- Event initiation identifies the number and location of cells in which to initiate the event (between zero and all cells). This moves the process from an *Event Instance Context* to the potential cells in which to initiate (called the *Spatial Contexts*), and then to the cells in which the event actually initiates (called *Active Cell Contexts*).
- The Occurrence step tests whether the event will continue or not in an active cell. If the test succeeds, the context remains the same and the process moves on to spreading. If the test fails, the active cell is terminated.
- Event spread is analogous to event initiation, except that it starts from an active cell. Event spread identifies the number and location of cells in which to spread (often neighbouring

cells). This moves the process from an *Active Cell Context* (the active spreading cell) to the potential cells to which to spread (called the *Recipient Contexts*, which are in cells that are not active but relate to the still-active spreading cell), and then to the cells in which the event actually spreads (called *Active Recipient Contexts*, which are in cells that are now active and can relate to the still-active spreading cell).

- After spreading is finished the spreading cells is deactivated and any *Active Recipients* become simply *Active Cells* and the loop continues with Occurrence testing.
- The process continues until all active cells are terminated either by failing the Occurrence test or by spreading to no new cells.

The reason that these contexts are important is that certain state information is available in each that may be useful for certain types of process models. For example, a landscape event for wildfire that models fire intensity, may track a unique fire intensity for each burning cell (i.e. available in an *Active Cell Context*). When the fire spreads, the intensity of a new burning cell may in part depend on the intensity of the spreading cell, which it can access during the spread step (i.e. in an *Active Recipient Context*) before the spreading cell terminates.

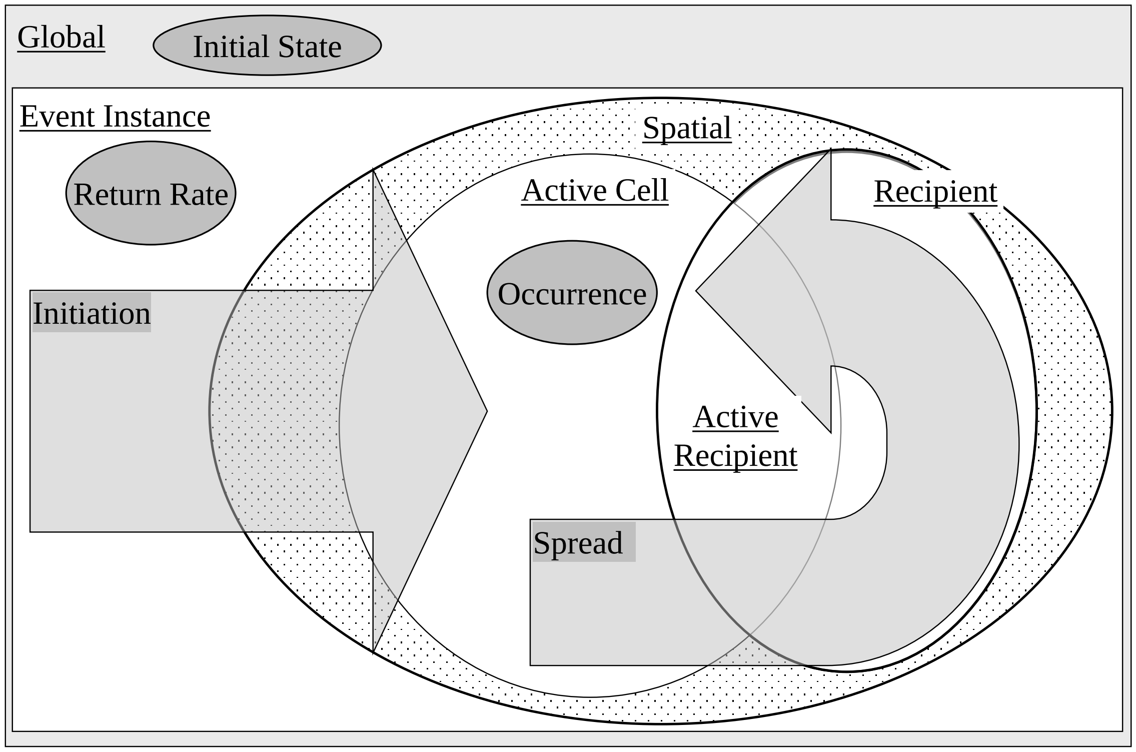


Figure 4. Spatio-temporal contexts associated with steps of the general process algorithm of landscape events shown in Figure 1.

4 Landscape Events

Landscape events are designed as an abstraction of processes that recurrently initiate on a landscape, possibly spread (in contiguous or non-contiguous patches), and that make changes in the spatio-temporal contexts "visited" by the event.

4.1 Landscape Event Properties

Landscape events define when and where state changes should occur along with the nature of those state changes using a set of *properties* (Table 2). Each property is defined by a set of assignments and expressions that are evaluated during the simulation to determine the actions to be taken.

Conceptually, properties determine the time between instances of the event (*ReturnTime* property), the spatial locations in which the event should initiate (*Initiation* properties), and the rate and cells to which the event should spread (*Spread* properties).

Note that properties focus on the behaviour of an event through space and time. State changes are made by associating assignment expressions with these properties to be processed either when/where the property result is evaluated, or when/where the property result takes the process (e.g. a future time, a neighbouring cell, etc.).

At the beginning of a simulation, the number of instances to create is defined by the ***InitialState*** property, which shifts from no instances of the landscape event to the creation of each instance.

After an instance is created, it is scheduled for processing based on the ***ReturnTime*** property, which is the time delay used for the event queue. Processing an event instance consists of event initiation and creating a new event instance to repeat the process. For continuous processes, *ReturnTime* is simply the time step used to discretely model state changes. For example, modelling succession annually corresponds to a *ReturnTime* of one year. For processes that recur sporadically, *ReturnTim*e is the interval between successive instances of the event. For example, a wind storm disturbance agent that recurs three times a year on average would correspond to a *ReturnTime* defined by a probability distribution with a mean of 122 days. *ReturnTime* is also used to help order event sequencing (e.g. first advance stand age, then assess availability to harvest, then attempt to harvest, then report results).

When an event is processed, the ***EventLocation*** property defines a region of possible locations for the event to start, of which some subset will be selected based on the other initiation properties. This allows identification of the set of cells in which an event can *potentially* initiate (e.g. forested cells in the study area).

Table 2 SELES landscape event properties. Each property controls a specific aspect of an event's behaviour.

Property Name	Purpose
Initial State	Defines the initial number of event instances at simulation start-up.
Return Time (when)	Defines the interval of time between successive instances of the event on the landscape. The time-step for an event may be fixed or variable.
Event Location (where possible)	Defines the set of cells in which the event can *potentially* initiate. By default this is the whole map, although this property can be used to restrict an event to a particular spatial region, or a particular type of landscape element.
Number of Clusters (how many)	Defines the number of cells in which the event will initiate. This property works in conjunction with the "Probability of Initiation" to select the cells in which the event will initiate (see Table 3).
Probability of Initiation (where most likely)	Defines the relative or absolute probability that the event will initiate in a particular cell. By default each cell has an equal probability of initiation (i.e., the landscape is homogeneous with respect to the initiation of the event), but this property provides an opportunity for the existing landscape pattern to affect the spatial distribution of the process.
Transitions (what happens)	Defines whether the event actually occurs in a cell, allowing a subtle difference to be made between initiation and establishment of an event in a cell. Although state changes can occur in any property, this is the logical place to determine if the event should cause a state change and the nature of that change. This property also determines whether or not the event will spread from a cell.
Spread Time	Used in spreading events to define the interval of time required for an event to spread from the current cells to its neighbours. It is processed after Transitions, and before spreading actually takes place.
Spread Location	Used in spreading events to define the set of cells to which an event can potentially spread from the current cell. This property is analogous to the "Event Location", except that it only controls the spreading of events. The default is to spread only to a cell's cardinal neighbours.
Number of Spread Recipients	Used in spreading events to define the number of cells to which an event should spread from the current cell. This property is analogous to the "Number of Clusters", except that it only controls the spreading of events. It works in conjunction with the "Probability of Spread".
Probability of Spread	Used in spreading events to define the absolute or relative probability that the event spreads to a particular cell. This property is analogous to the "Probability of Initiation", except that it only controls the spreading of events.
EndCluster	Main expression is presently undefined. Preliminary context is processed when a cluster terminates (when there are no more active cells).
EndEvent	Main expression is presently undefined. Preliminary context is processed when an event instance terminates (when there are no more active clusters).

Probability of Initiation (ProbInit) and ***Number of Clusters (NumClusters)*** together specify the cells in which events will *initiate* (see Table 3). Many types of processes are sensitive to the underlying spatial structure of the landscape. In these cases, the probability that an event will initiate at a specific spatial location is conditional on the properties, or state, of that location. *ProbInit* may specify either an absolute or a relative probability of the event initiating in any model cell, as a function of the cell state. That is, given that the event may initiate somewhere in the *EventLocation*, what is the probability that it will initiate at a given spatial location.

If *NumClusters* is defined then it specifies the number of cells in which to initiate the event. These locations may be drawn purely at random, or may be dependent on the relative probability for each location, as defined by *ProbInit*. If *NumClusters* is not defined, then each cell in *EventLocation* is tested independently for initiation, using *ProbInit* as an absolute probability. Table 3 illustrates the possible combinations of *ProbInit* and *NumClusters*, along with the behaviour invoked when each of these combinations is used.

Table 3 Relationship between *ProbInit* and *NumClusters* properties. These two properties can be used alone or in conjunction to specify where an event will initiate.

	ProbInit NOT specified (All cells have equal probability)	**ProbInit** specified (Each cell *i* has individual probability, P_i)
NumClusters NOT specified (No limit on number of clusters for event.)	Event initiates in every cell of EventLocation	ProbInit is the absolute probability of event initiating in each cell, with the expected number of cells to initiate event = $\sum^{C} (P_i)$ (where C = number of cells; and P_i = ProbInit for cell *i*)
NumClusters specified (Event will initiate in exactly *N* cells.)	Exactly *N* locations are chosen at random for event initiation, with an equal probability for every cell.	ProbInit is a relative probability of event initiating in each cell, with the exact number of cells to initiate event = *N*

For each cell that initiates, a new *active cell* and *cluster* are created. The value of ***Transitions*** then determines whether or not a transition *occurs*. This property determines if an event should proceed (i.e. occur) in a cell and, if so, what should happen as a result of that event. The main state transitions for the event are often defined in this property. This property is also used to determine if a spreading event will actually spread. If the event does occur, it will spread, whereas if the event does not occur, the active cell simply terminates without spreading.

Note that we draw a clear distinction between the *initiation* of an event and the *occurrence* of an event. An event is initiated when an event instance is "dropped" into the cell (via a new cluster, or via spread from a neighbouring cell). An event occurs when the *Transitions* property evaluates to true after initiation. For example, a lightning strike may *initiate* a fire in the cell it hits, but whether or not a fire actually *occurs* in that cell might be determined as a function of current fuel and weather conditions.

The **SpreadTime** property specifies if and when an event will spread from the current cell to neighbouring cells. This is analogous to the *ReturnTime* property, except that *SpreadTime* is computed in the current location. If specified, this gives the number of time units in the future to schedule spreading from the current cell. If not specified, then the event will not spread. Another difference from *ReturnTime* is that the consequence of *ReturnTime* is processed before initiation (to allow a set-up context) while the consequence of *SpreadTime* is processed after spreading (to allow a context prior to an active cell terminating).

It is important to understand when and how an active cell spreads. An active cell may spread only if it *occurs* in its own cell, as determined by the *Transitions* property. If the active cell did not occur, then it terminates. If it did occur, the active cell schedules spreading to be processed at some future time as determined by the *SpreadTime* property. When spreading is processed, a number of new active cells are created in neighbouring cells, within the same active cluster and event instance as the original spreading cell.

When a cell attempts to spread, ***SpreadLocation*** defines the set of cells to which it can *potentially* spread, analogous to **EventLocation**. ***Probability of Spread (SpreadProb)*** and ***Number of Spread Recipients (NumRecipients)*** select the set of cells to initiate via spreading, and operate analogous to *ProbInit* and *NumClusters*. The primary difference is that spreading has a source cell whereas event initiation does not. If *NumRecipients* is defined, then it specifies the number of cells to which a cell may spread, and the relative probability of spreading to a cell is given by *SpreadProb*. If *NumRecipients* is not defined, then each cell in *SpreadLocation* is tested independently for initiation using *SpreadProb* as an absolute probability (see Table 3).

Eventually, a spreading cluster (defined as the set of active cells that arrived from a common initiation cell) contains no more active cells, and terminates. When this happens the ***EndCluster*** property is processed, allowing for a finalization context. Likewise, when all clusters that initiated from a given event instance have terminated, the event instance terminates. When this happens, the ***EndEvent*** property is processed. In SELES10, only the preliminary context of these properties is defined.

Note that we have not described how the actual changes to the landscape are to be specified. The properties describe the behaviour of an event as it starts in a landscape, spreads and is finally extinguished.

4.2 Landscape Event Instances

It is important to clarify the difference between the *definition* and an *instance* of a landscape event.

A landscape event *definition* is the set of assignments and expressions defined for the properties (i.e. the syntax in the .lse file). Based on the definition, landscape event *instances* get dynamically created and scheduled on the event queue during a simulation to be processed at some time in the future. When a landscape event *instance* is taken off the queue and executed, it initially has no spatial location. The first step taken when processing an event instance is *initiation*, in which a set of clusters or cells is selected according to the *EventLocation*, *NumClusters* and *ProbInit* properties. After initiation (and often, but not always, before spreading), the landscape event instance schedules a *new* instance to be processed at a later time according to the *ReturnTime* property. A landscape event instance remains *active* as long as it has at least one active clusters. A cluster remains active as long as it has at least one active cell, which may include the initiating cell, or a cell that was reached via spread from this initiating cell. A cell is active if it was scheduled on the event queue to spread, or if it is currently being processed. An active cell is terminated if it fails the *Transitions* test or after it has finished spreading, while spreading may result in the activation of other cells.

An event *instance* is basically a *dynamic activation* of the landscape event. Through initiation of clusters and spread, it acquires specific spatial locations at which it affects the landscape. It initiates clusters and has a duration while active cells spread across the landscape, but it terminates when the last active cell, and hence the last active cluster terminates. The *ReturnTime* is used to schedule a *new* instance of an event.

The distinction between a landscape event definition and its instances is important since more than one instance of a single landscape event may be active at one time (e.g. one instance of a logging process may be created for each management unit).

Note that the identity of an active cell is determined by the source of its initiation, not by its location. That is, it is possible for there to be more than one active cell simultaneously in the same grid cell. For example, this may happen if a fire event spreads to the same cell from two different fire clusters. If such a possibility is undesirable, the landscape event needs to include appropriate state and specify conditions to prevent this from happening (e.g. using a spatial layer that records that an event has already visited this location).

For clarity and brevity in the rest of the document, a landscape event *definition* or *instance* will both simply be referred to as a landscape event.

4.3 Dynamic Context Variables

In additional to the shared state-space (global and spatial variables and constants), a landscape event may declare four types of local *dynamic context variables*:

(a) *Local variables* are the same as global variables (i.e. non-spatial, but may vary over time), but are local to the landscape event (i.e. not defined in the dynamic model file as part of the shared state space, so only available within this landscape event).

(b) *Event variables* define values unique to individual active instances of an event. For example, a *Fire* event might declare an event variable called *NumFires* to store the number of fires for each fire storm event instance. In this example, the *NumFires* variable would be different for each instance of the *Fire* event.

(c) *Cluster variables* define values unique to individual active clusters of an event. For example, a *Fire* event might declare a cluster variable called *FireSize* to store the target size for each individual fire patch during a fire storm. In this example, the *FireSize* variable would be different for each cluster initiated by the *Fire* event.

(d) *Cell variables* (or *active cell variables*) allow values unique to individual cells in which the event instance is currently active. For example, a *Fire* event might declare a cell variable called *Intensity* to store the magnitude of the fire at each cell on the active front of burning fire patches.

When a new event instance, cluster or active cell is created, any event, cluster or cell variables respectively declared will be dynamically created (with no initial value defined).

The value of an event variable will be the same for all clusters created by the event instance, but each cluster will have a unique set of cluster variables. Each cell to which a cluster spreads will refer to the same cluster and event variables, but unique cell variables.

The dynamic state is hierarchically organized as shown in Figure 5, where an active event has a set of active clusters, each of which in turn has a set of active cells.

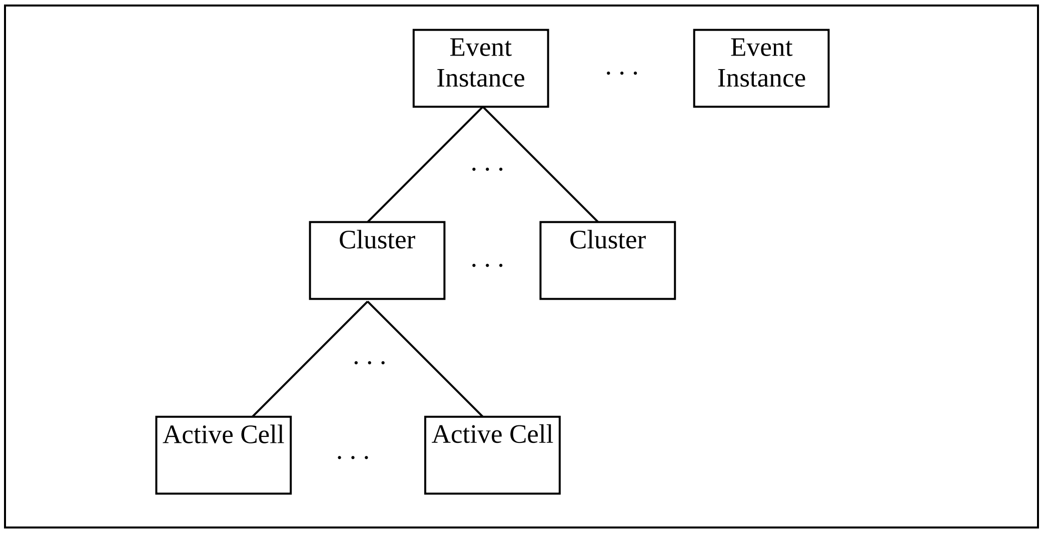


Figure 5. The hierarchical structure of the dynamic state. Each active event has a unique set of event variables, and one or more active clusters. Each active cluster has a unique set of cluster variables, and one or more active cells. Each active cell has a unique set of cell variables.

When an active cell is extinguished, its cell variables are removed from the system. When the last active cell of a cluster is extinguished, its cluster variables are removed. When the last cluster of an event instance is extinguished, its event variables are removed.

SELES handles memory management to ensure that dynamic context variables are allocated when they are activated and destroyed when they are deactivated. Modellers simply need to declare variables of the appropriate type, and use them as needed in the appropriate contexts. Before their first assignment, the value of dynamic context variables is undefined. This hierarchical structure allows lower order entities of the simulation (e.g., cell level) to act semi-independently, but in coordination with higher order entities (e.g., cluster, event, and global levels).

4.4 Spatio-temporal Contexts of Properties

Each property has two associated contexts: the *operating* context and the *consequent* context:

- The ***operating context*** is the context in which property result is evaluated (the *main expression* of the property), as well as associated expressions (called the *preliminary expressions* of the property). The operating context is the result of previous behaviour of the landscape event that led to this context.
- The ***consequent contexts*** arise from evaluating the property, which may result in zero or more new contexts (e.g. as a consequence of specifying that 3 fires will ignite in a model called *Fire*, 3 active spatial contexts will be created, one at each fire ignition point).
- Property contexts are automatically managed by SELES, allowing the modeller to focus on the high-level behaviour of landscape event properties rather than technical details of managing contexts.

Property contexts can be illustrated by expanding the generalized diagram in Figure 4 to show how properties implement the general landscape process algorithm (Figure 6):

- *Initial State*: Evaluated at simulation start up in a *Global Context* to set the number of initial event instances (i.e. there is no spatial location and no active instance). The consequence is to create the first event instances.
- *Return Time*: Evaluated when a new instance is created to set the time interval to schedule the event for processing. The consequence is to cause the event instance to be processed at that future time (i.e. when the scheduled event is taken off the queue) prior to cluster initiation. Both contexts are non-spatial.
- *Event Location*: Evaluated after an event instance returns to define the set of cells (region) to which the event can potentially initiate. The consequence is to identify those potential cells (spatial).
- *Number of Clusters*: Evaluated after an event instance returns to define the number of clusters/cells to initiate. The consequence is to identify those cells and create a cluster and active cell for each (i.e. create *Active Cell Contexts*).
- *Probability of Initiation*: Evaluated in each potential cell to initiate from the *Event Location* to define the probability of selecting each cell for activation. The consequence is to identify those cells and create a cluster and active cell for each (i.e. create *Active Cell Contexts*).
- *Transitions*: Evaluated to test if the event "occurs" in an active cell (reached by initiation or spread). If the test passes, then spread continues. Otherwise the active cell terminates.

- *Spread Time*: Evaluated in an active cell after successful "occurrence" to set the time interval to schedule the active cell for spreading. The consequence is to cause the active cell to spread at that future time. Both contexts are *Active Cell*.
- *Spread Location*: Evaluated in an active cell after successful "occurrence" to define the set of cells (region) to which the active cell can potentially spread. The consequence is to identify those potential cells (*Recipient Contexts* since the potential cells are not active, but the source spreading cell is still active).
- *Number of Recipients*: Evaluated in an active cell after successful "occurrence" to define the number of cells to which to spread. The consequence is to identify those cells and create a new active cell for each (i.e. create *Active Recipient Contexts* since the new cells are active and the source spreading cell is still active).
- *Spread Probability*: Evaluated in each potential cell to which to spread to define the probability of selecting each cell for activation. The consequence is to identify those cells and create an active cell for each (i.e. create *Active Recipient Contexts*).
- After spreading is finished the spreading cell is deactivated and any *Active Recipients* become simply *Active Cells* and the loop continues with Occurrence testing (*Transitions*).
- The process continues until all active cells are terminated either by failing the *Transitions* test or by spreading to no new cells.

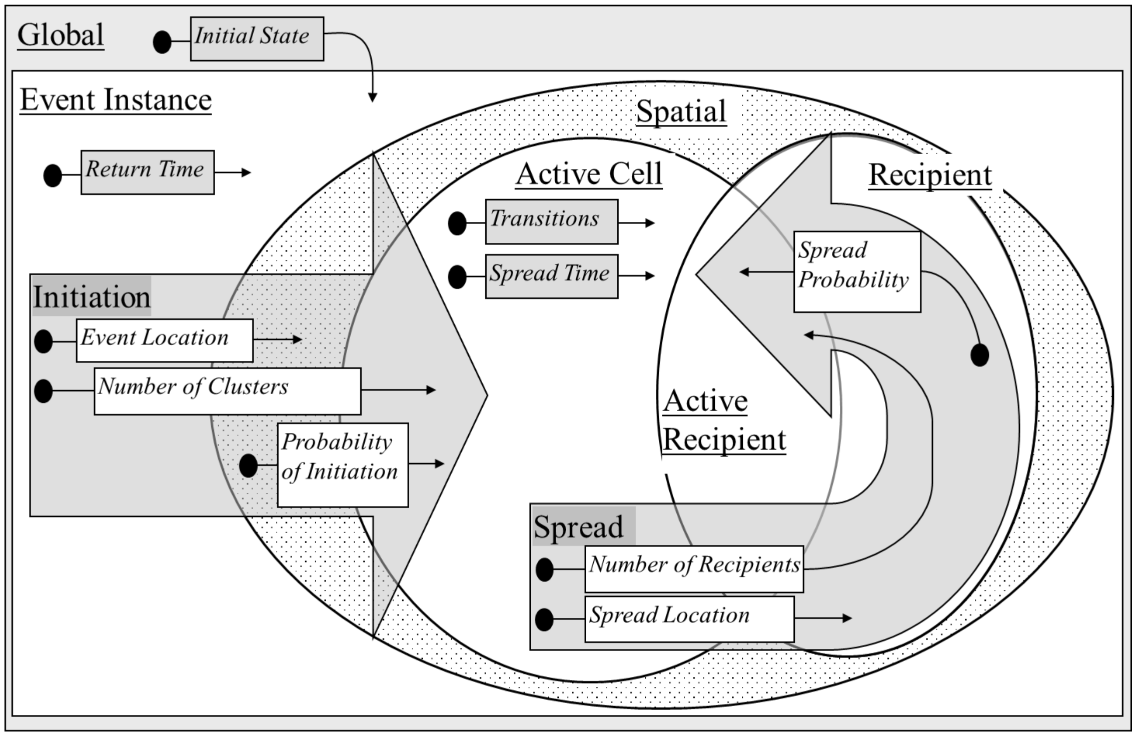


Figure 6. Spatio-temporal contexts associated with the property operating context (dot) and consequent context (arrow), refining the general process algorithm of landscape events shown in Figure 1 and Figure 4.

The types of variables to which expressions can refer in each of these contexts are summarized in Table 4, which also includes the default values for the main expression of properties. Contexts always include global variables and constants. Contexts that arise in a specific grid cell include spatial variables and constants, and may include different local dynamic context variables.

Table 4 Value and default for main expression, and context of landscape event properties.

Property Name	Value of Main Expression	Default	Operating Context	Consequent Context
Initial State	Integer	1	Global	Event Instance
Return Time	Floating point	0 (event is processed once at time 0)	Event Instance	Event Instance (same as Preliminary but at a later time)
Event Location	Spatial region	Whole map	Event Instance	Spatial (potential cells to initiate)
Probability of Initiation	Floating point (for each cell in Event Location)	1.0	Spatial (potential cells to initiate)	Active Cell (cells in which the event initiates)
Number of Clusters	Integer	Undefined (emergent)	Event Instance	Same as above
Transitions	Boolean (for each cell activated by initiation or spread)	TRUE	Active Cell (initiation and spread)	Active Cells (cells in which the event “occurred”)
Spread Time	Floating point	None (event doesn’t spread)	Active cell (cells in which the event “occurred”)	Active cell (same as Preliminary but at a later time)
Spread Location	Spatial region	Cardinal neighbours	Same as above	Recipient (potential cells to which to spread)
Probability of Spread	Floating point (for each cell in Spread Location)	1.0	Recipient (potential cells to which to spread)	Active Recipient (cells to which the event spreads)
Number of Spread Recipients	Integer	Undefined (emergent)	Active cell (cells in which the event “occurred”)	Same as above
EndCluster	Boolean	TRUE	Cluster	n/a
EndEvent	Boolean	TRUE	Event Instance	n./a

Expressions in a spatial context may refer to the value of a spatial variable in a given cell. Some properties have no spatial context (e.g. *ReturnTime*) and so cannot refer to active cell or cluster variables, but can refer to layers in their entirety (e.g., setting *StandAge = 0* in *Transitions* will set the value in the current cell, but this expression in *ReturnTime* will set all cells in the *StandAge* spatial variable).

Put more formally, let $S = \{s_1, s_2, ... s_n\}$ be an n element vector containing the values of all state variables defined in the spatio-temporal context of an expression in a SELES model. Then a SELES expression specifies a function: $y = f(S)$

If an expression has a spatial context that ranges over a region of cells, then each cell in the region will potentially have a unique value. In this case, we can think of the result as a "transient" raster layer, with the value in cell i,j equal to $y_{i,j} = f(S_{i,j})$. Thus, when a spatial expression is evaluated at a particular cell, the function parameters take on the values of the state variables at that cell, and the result may be assigned to a spatial variable at that cell. Readers familiar with GIS will recognize a similarity between such spatial expressions and raster overlays. One difference is that while a GIS overlay generally produces a complete and durable raster layer, the value of a SELES spatial expression may be transient and/or partial -- it is only computed for cells in which it is required, and may be discarded when the operation is complete. There are other important differences. One is that the variables also include the global variables as well as *dynamic context variables* defined for the landscape event. Another is that SELES expressions may include random numbers drawn from a distribution each time the expression is evaluated, allowing stochastic expressions rather than strictly deterministic expressions.

4.5 Landscape Event Property Details

This section gives a more detailed description of the duty or behaviour of each of the landscape event properties listed above. In these descriptions, the following notation is defined for brevity:

- Q represents the simulation engine priority event queue described previously (see Figure 3).
- S represents the set of n variables available in the current context. For spatial contexts, this additionally includes a specific spatial location, where $S_{i,j}$ represents a vector of length n variables that contain the values of the state variables at cell i,j.

Each landscape event property is described by the following characteristics:

Result: Specifies the contexts created or modified by this property.

Default: Specifies the default behaviour if the property is not defined.

Context: Specifies the operating and consequent contexts of the property. This determines the set of state variables, S, that can be referred to in expressions. The preliminary expressions and main expression share one context (the *operating context)* while the consequent expressions generally have a different context (the *consequent context)*. A context can either be *global, event instance, spatial, active cell, recipient or active recipient*. A global context includes global and spatial variables and constants, as well as the *Time* and *EndTime* system variables. A reference to a spatial variable (layer) in a global context applies to *every cell* in the layer, which we call a "whole layer" reference. An event instance context additionally includes event variables as well as the *EventId* system variable. A spatial context additionally includes layers (at a specific location), as well as the *Location* system variable. An *active cell* context is a spatial context with an active cell, and hence also includes cluster and cell variables. A *recipient* context is a spatial context for a potential recipient of spread, and so includes the *source* (spreading) cell, location, cell and layer variables (which are accessed using the prefix SOURCE). An *active recipient* context combines an active cell context with a recipient context.

Description: Specification of the property's function.

InitialState

Defines the number of event instances to create at simulation start-up

Result: A set of event instances.

Default: One instance

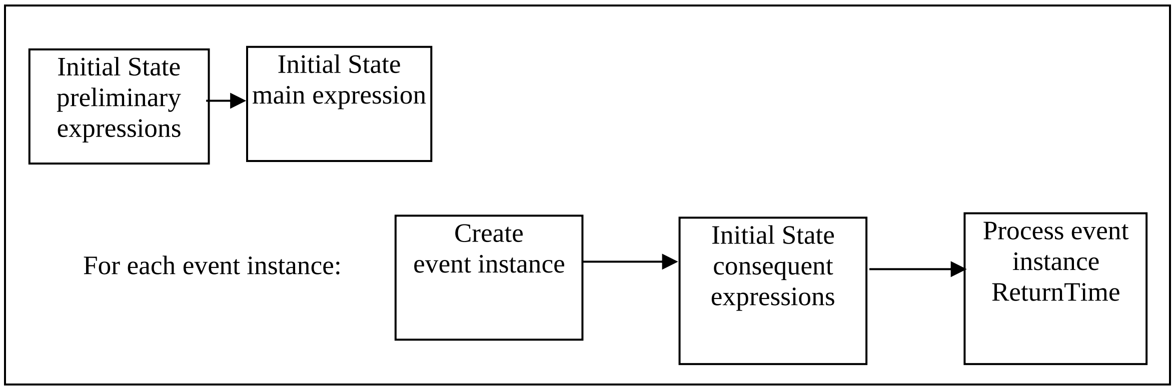
Context: Preliminary expressions: global

Main Expression: global

Consequent expressions: event instance (global)

The main expression is any valid global expression defining the number of event instances to create at the beginning of a simulation.

Description: This property is useful to set up initial conditions local to an event, and to create multiple instances of an event at start-up. The preliminary expressions are evaluated once, while the consequent expressions are evaluated for each event instance created. The initial values for event variables are set to zero. Since there is no instance prior to this property, event variables are not available in the preliminary context. After each event instance is created, *ReturnTime* is evaluated to schedule it for future processing


Initial State preliminary expressions
Initial State main expression
For each event instance:
Create event instance
Initial State consequent expressions
Process event instance ReturnTime

ReturnTime

Defines the interval between creation of new event instances, and when they are processed.

Result: Schedule a new event instance on the event queue Q for future processing.

Default: Schedule the event to be processed exactly once at simulation start-up (*ReturnTime* = 0).

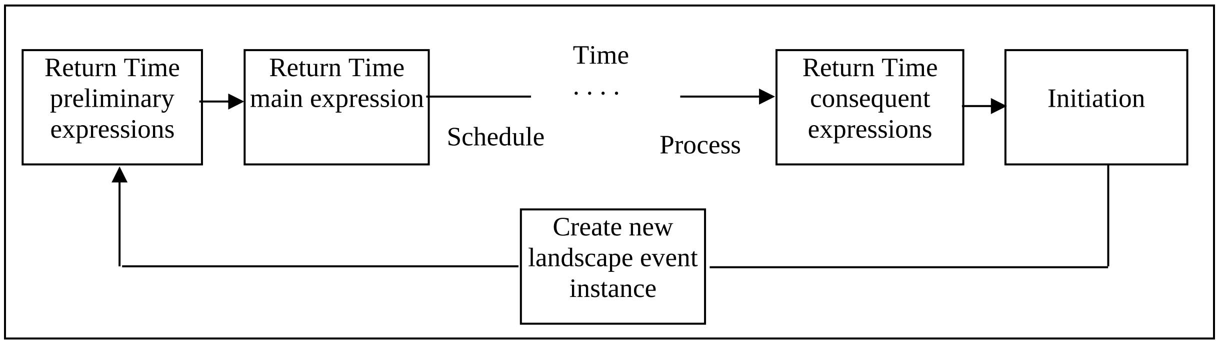
Context: Preliminary expressions: event instance

Main Expression (*rt*): event instance

Consequent expressions: event instance

The main expression is any valid global expression, *rt*(*S*), defining the time elapsed (in time units) between successive instances of the event being created.

Description: When an event instance is created, the preliminary expressions and the main expression, *rt*(*S*), are evaluated. The event instance is scheduled to be processed at *rt*(*S*) time units in the future. When the event instance is taken off the queue, the consequent expressions are evaluated. Then *EventLocation* is invoked to process event initiation. Once initiation is completed, a new event instance is created and the process repeats. The initial values of the new instance are set to the values of the previous instance. Note that a spreading event instance may not be completed prior to creating the next instance.


Return Time preliminary expressions
Return Time main expression
Time
. . . .
Schedule
Process
Return Time consequent expressions
Initiation
Create new landscape event instance

<u>EventLocation</u>

Defines the set of potential cells in which an event can initiate.

<u>Result</u>: A set of spatial locations for potential initiation of an event (*InitRegion*).

<u>Default</u>: Every cell in the landscape.

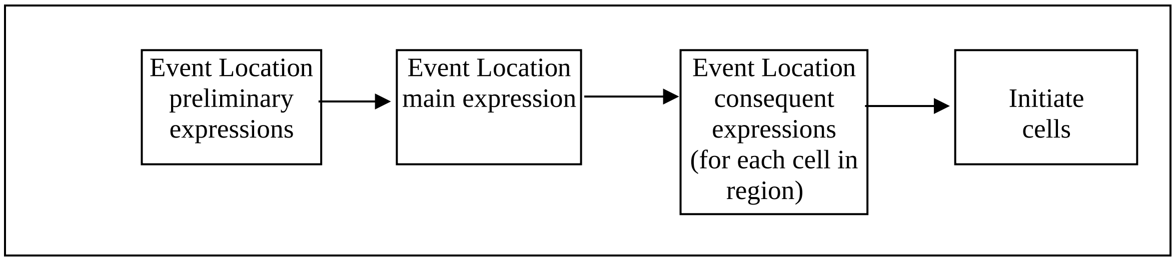
<u>Context</u>: Preliminary expressions: event instance

Main Expression: event instance

Consequent expressions: spatial

The main expression is a region expression that defines the cells in which the event may initiate. Although the consequent expressions are spatial, they can only refer to layer variables and not active cell variables, since at this point, active cells have not yet been created by the event instance.

<u>Description</u>: The set of cells in the *InitRegion* created by this property defines the landscape as seen during the initiation of the event via *ProbInit* and *NumClusters*. Initiation only applies to this set of cells. The preliminary expressions are evaluated once before the main expression, while the consequent expressions are evaluated in each cell in the *InitRegion*.


Event Location preliminary expressions
Event Location main expression
Event Location consequent expressions (for each cell in region)
Initiate cells

ProbInit

Defines the probability (relative or absolute) that an event will initiate in a particular cell.

Result: Transient "probability surface" for cells in the initiation region *InitRegion* (created by *EventLocation*) with floating point values indicating the potential of each cell to initiate with the event (*pInit*).

Default: Every cell in the initiation region has a value of 1.0.

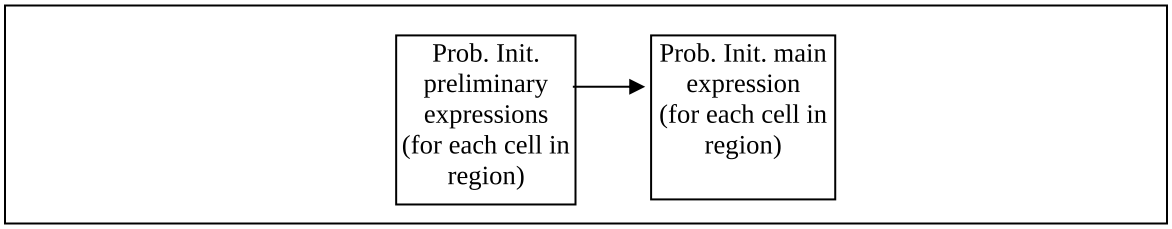
Context: Preliminary expressions: spatial

Main Expression (*pi*): spatial

Consequent expressions: active cell

The main expression is any valid spatial expression, *pi*(*S*), defining the probability, normally between zero and one for absolute probabilities, of an event initiating in a cell. The preliminary expressions and main expression are spatial and so they cannot refer to cluster or active cell variables, since at this point, active cells have not yet been created. The consequent expressions can refer to active cell and cluster variables.

Informal Specification: The preliminary expressions and main expression, *pi(S*), are evaluated for each cell in the initiation region *InitRegion*. The consequent expressions are only evaluated in those cells in which the event actually initiates – see *NumClusters* below.


Prob. Init. preliminary expressions (for each cell in region)
Prob. Init. main expression (for each cell in region)

NumClusters

Defines the number of clusters for an event instance (i.e. number of cells to initiate)

Result: A set of active cells derived from the initiation region *InitRegion* and probability of initiation surface *pInit*.

Default: Every cell *i,j* in *InitRegion* has an absolute probability of initiating equal to $pInit_{i,j}$. Thus, the actual number of clusters, or initiating cells, is emergent ("unbounded").

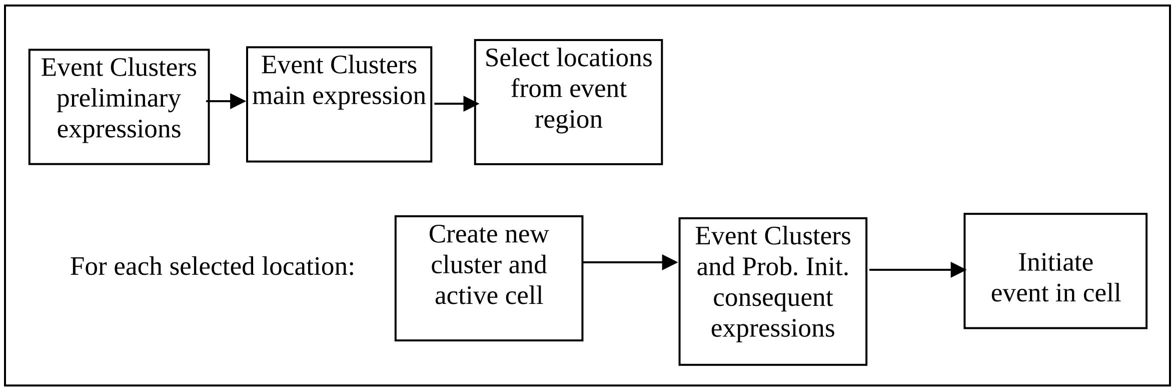
Context: Preliminary expressions: event instance

Main Expression (*nc*): event instance

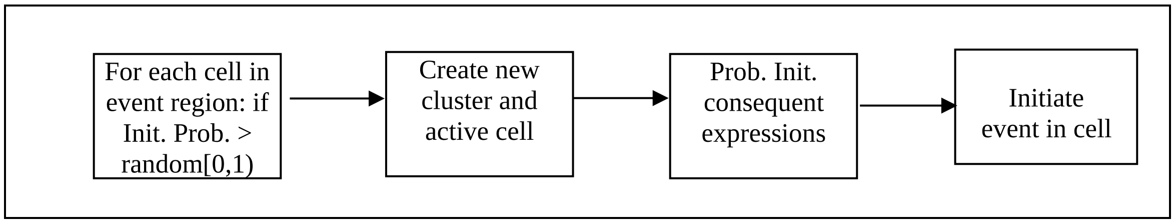
Consequent expressions: active cell

The main expression is any valid global expression, *nc*(*S*), defining the number of cells in the initiation region *InitRegion* in which the event is to initiate.

Description: Once *ProbInit* has produced the "probability surface" *pInit* of potential sites, and their relative probabilities, *NumClusters* is evaluated to determine the actual number of cells in which to initiate the event. The cells in which the event is initiated are selected stochastically from the potential sites based on the probabilities in *pInit* (i.e. the probabilities are *relative)*. For cells that initiate, the consequent expressions for both *NumClusters* and *ProbInit* are evaluated.


Event Clusters preliminary expressions
Event Clusters main expression
Select locations from event region
For each selected location:
Create new cluster and active cell
Event Clusters and Prob. Init. consequent expressions
Initiate event in cell

If *NumClusters* is not defined (or the main expression evaluates to < 0), then cells are selected for initiation *independently* based on *pInit* (i.e. the probabilities are *absolute).*


For each cell in event region: if Init. Prob. > random[0,1)
Create new cluster and active cell
Prob. Init. consequent expressions
Initiate event in cell

Transitions

Defines if the event occurs in an active cell.

Result: Either a continuing active cell or a terminating active cell.

Default: The event occurs (continues).

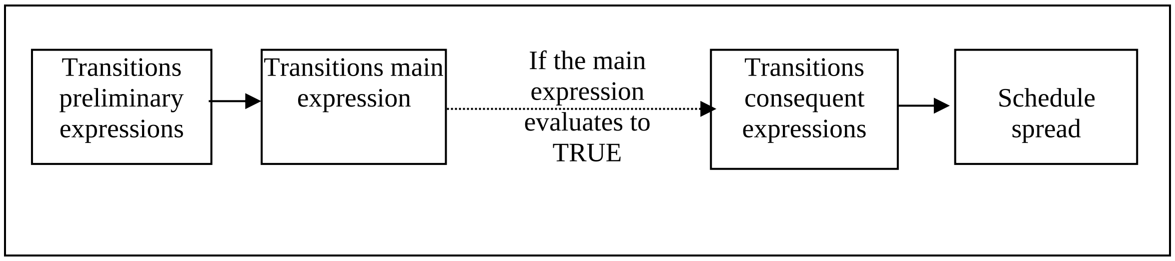
Context: Preliminary expressions: active cell

Main Expression (*tr*): active cell

Consequent expressions: active cell

The main expression is any valid spatial expression *tr*(*S*), defining a Boolean value.

Description: When an active cell has initiated in a cell, either through event initiation or spread, the preliminary expressions and main expression, *tr(S*), are evaluated. If the result of *tr(S*) is TRUE (> 0), then the event *occurs* in the cell. The consequent expressions are only evaluated, and spreading is only attempted, if the event occurs. For spreading events, *SpreadTime* is then invoked to schedule spreading.


Transitions preliminary expressions
Transitions main expression
If the main expression evaluates to TRUE
Transitions consequent expressions
Schedule spread

<u>SpreadTime</u>

Defines the time taken between occurrence of an event in a cell and spreading out of the cell to neighbouring cells, and schedules the active cell to spread at this time in the future.

<u>Result</u>: Schedule an active cell on the event queue Q for future processing of spread.

<u>Default</u>: Event does not spread.

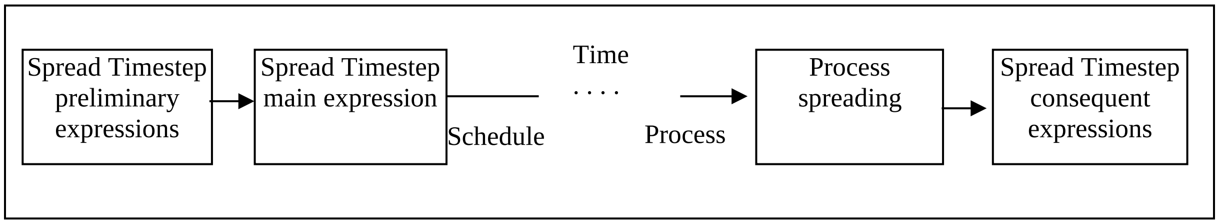
<u>Context</u>: Preliminary expressions: active cell

Main Expression (*st*): active cell

Consequent expressions: active cell

The main expression is any valid spatial expression *st*(*S*), defining the time elapsed (in time units) between when an event occurs in a cell and when it spreads to neighbouring cells. If *st*(S) is negative, then spreading is "immediate" (spreading will be processed without scheduling on the main queue). Once immediate mode starts, all spreading with negative values will be processed using a temporary queue before returning to normal processing. The magnitude of the negative value indicates the ordering within immediate active cells on the temporary queue (e.g. a value of –1 will be processed ahead of –2, and the order for subsequent spreading will be incremental from when immediate spreading started). Note: this facility must be used with caution and only in cases that have well-defined termination. Poorly constructed models that permit immediate spreading repeatedly to the same cell can cause infinite loops.

<u>Description</u>: After an active cell "occurs", the preliminary expressions are evaluated. The main expression is then evaluated to determine the length of time required before the event spreads to neighbouring cells. Spread from this active cell is scheduled on the event queue to be processed at *st*(S) number of time units in the future. *After* the spreading is processed, the consequent expressions are evaluated and the active cell terminates.


Spread Timestep preliminary expressions
Spread Timestep main expression
Schedule
Time
. . . .
Process
Process spreading
Spread Timestep consequent expressions

SpreadLocation

Defines the set of potential cells to which an active cell can spread.

Result: A set of spatial locations for potential spread (*SpreadRegion*)

Default: The four cardinal neighbours (north, south, east and west)

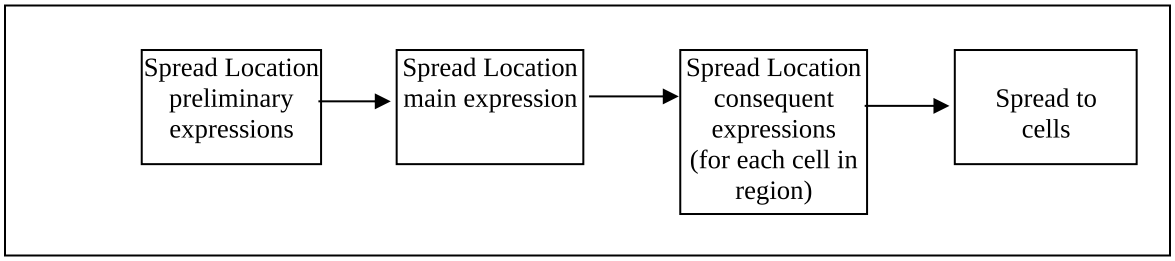
Context: Preliminary expressions: active cell

Main Expression: active cell

Consequent expressions: recipient

The main expression is a spatial region that defines the cells (potential recipients) to which the active cell may spread. Although the consequent expressions are spatial, they are not active and so cannot refer to active cell variables since at this point the potential recipient cells have not yet been activated (but expressions can refer to cluster variables). However, since the source spreading cell is still active, source cell and layer variables can be referenced (and refer to values for the spreading active cell).

Description: The set of cells returned by the region defines the landscape as seen during spread from an active cell via *SpreadProb* and *NumRecipients*. Spreading only applies to this set of cells. The preliminary expressions are evaluated once before the main expression, while the consequent expressions are evaluated in each of the cells in the *SpreadRegion*.


Spread Location preliminary expressions
Spread Location main expression
Spread Location consequent expressions (for each cell in region)
Spread to cells

<u>SpreadProb</u>

Defines the probability (relative or absolute) that an active cell will spread to a particular cell.

<u>Result</u>: Transient "probability surface" for cells in the spread region *SpreadRegion* with floating point values indicating the potential of each cell to initiate via spread (*pSpread*).

<u>Default</u>: Every cell in the spread region has a value of 1.0.

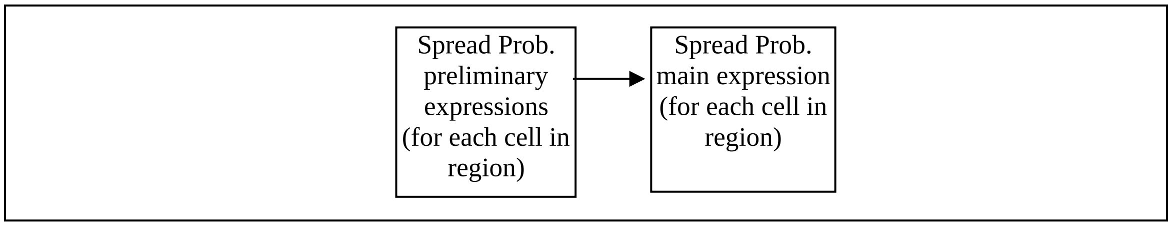
<u>Context</u>: Preliminary expressions: recipient

Main Expression (s*p*): recipient

Consequent expressions: active recipient

The main expression is any valid spread expression, s*p*(*S*), defining the probability, normally between zero and one for absolute probabilities, of an event initiating in a cell via spread from an active cell. Although the preliminary expressions are spatial, they are not active and so cannot refer to active cell variables since at this point the potential recipient cells have not yet been activated (but expressions can refer to cluster variables). However, since the source spreading cell is still active, source cell and layer variables can be referenced (and refer to values for the spreading active cell). The consequent expressions can additionally refer to active cell variables since the new cell is now an active recipient.

<u>Description</u>: After determining the spread location, the preliminary expressions and the main expression, s*p(S*), are evaluated for each cell in the spread region. The consequent expressions are only evaluated in those cells to which the active cell actually spreads – see *NumRecipients* below.


Spread Prob. preliminary expressions (for each cell in region)
Spread Prob. main expression (for each cell in region)

NumRecipients

Defines the number of cells to which to spread.

Result: A set of active cells, derived from the spread region *SpreadRegion* and probability of spread layer *pSpread* from the active cell.

Default: Every cell *i,j* in *SpreadRegion* has a probability of spreading equal to *pSpread*$_{i,j}$. Thus, the actual number of cells spread to is emergent (unbounded).

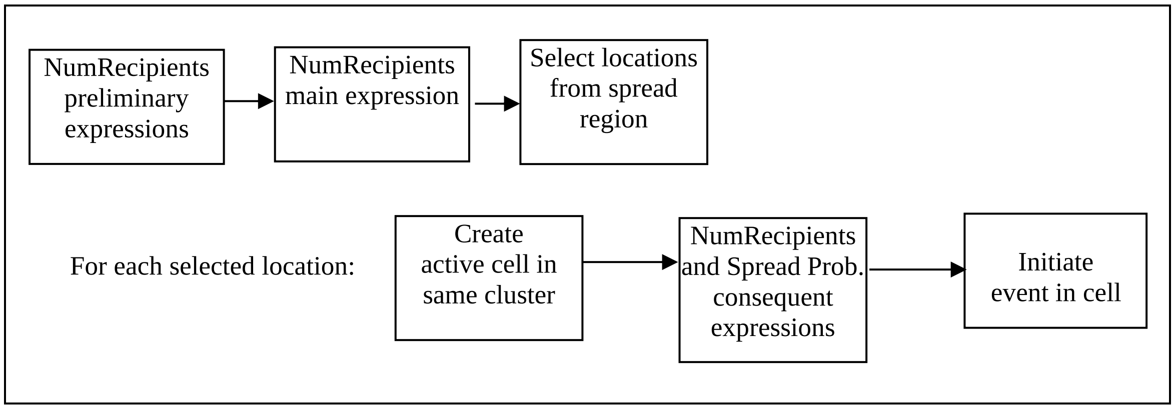
Context: Preliminary expressions: active cell

Main Expression (*nr*): active cell

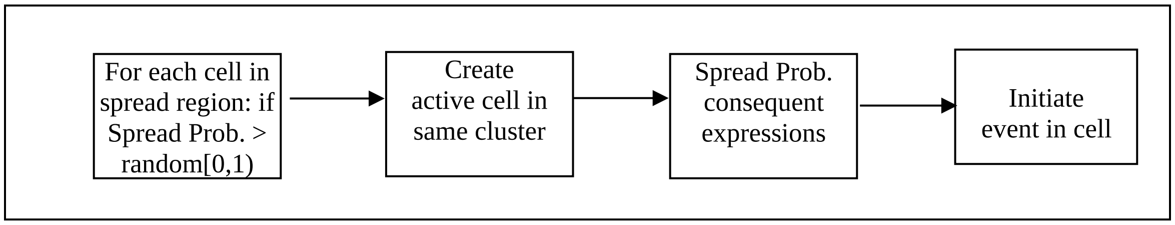
Consequent expressions: active recipient

The main expression is any valid spatial expression, *nr(S)*, defining the number of cells to which to spread.

Description: Evaluated after *SpreadProb* has produced the "spread probability surface" to determine the target number of cells to which the active cell should spread. The actual cells activated via spread are selected stochastically from the potential sites based on the spread probabilities *pSpread*. For cells activated via spread, the consequent expressions for both *NumRecipients* and *SpreadProb* are evaluated as active recipients, and the new active cell is then handled by invoking *Transitions*.


NumRecipients preliminary expressions
NumRecipients main expression
Select locations from spread region
For each selected location:
Create active cell in same cluster
NumRecipients and Spread Prob. consequent expressions
Initiate event in cell

If *NumRecipients* is not defined (or < 0), then cells are selected for spread *independently* based on *pSpread* (i.e. the probabilities are *absolute).*


For each cell in spread region: if Spread Prob. > random[0,1)
Create active cell in same cluster
Spread Prob. consequent expressions
Initiate event in cell

PART 2. **SELES: User Interface**

This part provides an overview of relevant aspects of the user interface of SELES10 and the Landscape Editor. This document assumes that these executables have been installed and can be launched. As with other sections, only features relevant to most users are described.

SELES10 and LSEditor10 are best run on Windows10 or later.

1 SELES Menus

After starting the SELES executable (i.e. seles10.exe), the user will be presented with a blank interface (no raster views or models loaded). A more complete set of menu options will be available after at least one raster is created or loaded. The operations available under each pull down menu are described in the following sections.

SELES can also be launched from a command line, with an optional scenario (.scn) file to load.

1.1 File Menu

The file menu has the following options.

a) **New** (also 1st icon on toolbar)**:** Creates a new view window with default dimensions taken from a current view, and user specified bounds and name. Parameters can be changes to the number of rows/columns, range of values (max/min) and whether the layer is floating point (the default is integer). A pull-down list of current views (“Get Size From Layer”) can be used to select the reference layer to set dimensions and georeferencing.

b) **Open** (also 2nd icon on toolbar): Open an existing file. The result depends on the contents of the file being opened. SELES does not use the file name extension to determine the type of file; rather the type of file is determined from file contents. The two main types of files that can be opened are GIS raster files and SELES scenario files (see next section). These files are each opened by selecting the desired file using the browser.

   The GIS formats supported are GeoTiff (preferred), GRASS (binary, compressed and ASCII), ERDAS (8 and 16 bit) and ARC ASCII. To open a GRASS raster file, select the file in the “cell” directory. The raster contained in the file (in conjunction with information stored in associated files, such as georeferencing, legend and colour) will be displayed in a new raster view of the same name (without suffix).

   SELES Scenario Files (generally with a .scn suffix) are used to execute a sequence of commands, such as loading raster and dynamic model files, changing parameters and running simulations (see next section).

c) **DryRun**: Opens and “dry runs” a scenario file. A dry run performs all tasks in a scenario except for actually running a simulation.

d) **Close** (also 3rd icon on toolbar): Close the current selected raster view. This can also be accomplished by clicking with the mouse on the “Close” icon (the “X”) on the top right corner of the view window.

e) **CloseAll** (also 4th icon on toolbar): Close all views and documents.

f) **Save**: Same as **Save As**.

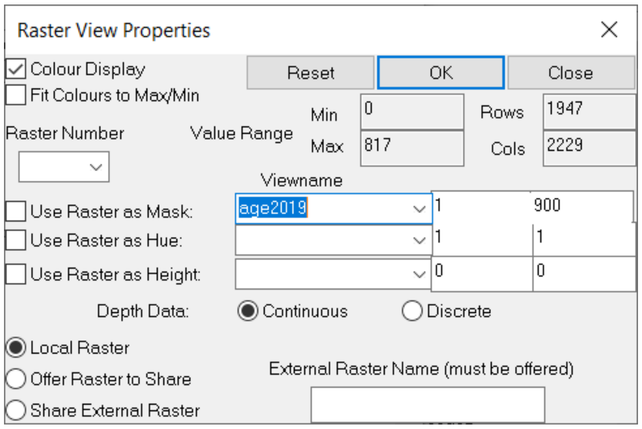
g) **Save As ...** (also 5th icon on toolbar): Save the raster in the currently active view as a raster. This option will bring up a file browser from which the directory and file name can be set. The “Save as type” pull-down menu allows selection of the type of information or GIS format to save (GeoTiff, GRASS, ARC ASCII and Erdas). The directory for a GRASS GIS raster should be the “cell” directory of a GRASS ERDAS). The header and other required associated files will be automatically created in the correct directory of the mapset.

h) **Exit**: Exit the SELES application. This can also be achieved by clicking with the mouse on the “Close” icon (the “X”) on the top right corner of the main application window.

1.2 View Menu

The main options available from this menu are “Raster View Properties”, “Show Legend” and “Histogram”.

a) **Raster View Properties** brings up a dialog box to control raster display.


Raster View Properties

[✓] Colour Display
[ ] Fit Colours to Max/Min

Reset | OK | Close

Raster Number

Value Range: Min 0, Max 817

Rows 1947, Cols 2229

Viewname

[ ] Use Raster as Mask: age2019 | 1 | 900
[ ] Use Raster as Hue: | 1 | 1
[ ] Use Raster as Height: | 0 | 0

Depth Data: [✓ Continuous] Discrete

[✓ Local Raster]
Offer Raster to Share
Share External Raster

External Raster Name (must be offered)

- “Colour Display” check box: sets whether the image is shown in colour or black and white.
- “Fit Colours to Max/Min” check box; rescales the colour lookup table to the range of values in the raster (which may differ from the raster bounds for a dynamic raster).
- “Raster Number” list: only used for multi-layer (interleaved) rasters.
- “Use Image as Mask”: allows selection of another view to use as a visual mask on the current image. If selected, those values in the range [Minimum, Maximum] from the mask image will be drawn over the current layer.
- “Use Image as Hue”: allows a selected image to be used as a hue modifier for the current raster view. The minimum and maximum values from the chosen view should contain values that specify the minimum and maximum hue value in the hue image. Higher values in the hue image will cause the corresponding pixels in the current image to be displayed brighter.

- "Use Image as Height": allows an image to be displayed as a 3-D map, where the selected image provides the height, or elevation, of each pixel map, over which the current raster view is "draped". There are two methods for rendering the height data (i) "Continuous" height data is used when the raster is a continuous variable (such as elevation), in which neighbouring values have similar in values, and are thus interpolated; and (ii) "Discrete" height data is for rasters that represent a categorical variable, and so neighbouring height values are not assumed to be related.
- This dialog box is also used to control where a raster view is shared with a compatible, external application (rarely used).

b) **Show Legend** brings up a dialog that displays legend information associated with the currently selected view. It also shows the colour, value and label under the mouse position.

c) **Histogram** brings up a dialog box with which simple histograms can be displayed of selected layers, optionally masked by other layers. Options include the min/max of the mask layer, histogram class size and min/max values displayed. The colours used for the histogram classes are taken from the colour used for the smallest value in the class.

d) **Histogram2** brings up a more detailed dialog box in which histograms can be specified as in "value summary" models by defining the region over which the histogram is defined, and the function used to define classes (advanced).

1.3 Window Menu

The Window menu has the following options:

a) **New Window**: Creates a new window of the currently active raster view (sometimes uses for display comparisons).
b) **Cascade**: Cascades the raster views.
c) **Tile**: Tiles the raster views.
d) **Arrange Icons**: No effect at the moment.
e) **Minimize All** (also 8th icon on toolbar): Minimize all raster views.
f) **Minimize Initial State** (also 9th icon on toolbar): Minimize all raster views that contain initial state information for the currently loaded dynamic model.
g) **Minimize Static** (also 10th icon on toolbar): Minimize all raster views that contain static information for the currently loaded dynamic model.
h) **Name Document**: Allows user to change the name of the currently active view (rarely used).
i) **List of current windows**: The bottom portion of this menu contains a list of the raster views that currently exist. The active raster view is shown with a check mark beside it. A raster view can be activated by selecting from this list (or by clicking on the window itself).

1.4 Help Menu

Not functional (other than "About seles").

1.5 Static Models Menu

Static models generally produce a single output (raster view or value). The "Align Layer" option on this menu is used to match the georeferencing of one layer based on a second layer.

a) **Neutral Model**: Used to generate random patterns (rarely used for applied models). Not needed to run most models.

b) **Fractal Model**: Used to generate spatial fractal patterns (rarely used for applied models). Not needed to run most models.

c) **Site (Specific) Model**: Used to generate stochastic, static patterns based on relative probabilities using values from other rasters at each site (rarely used for applied models). Not needed to run most models.

d) **(Summarize) Value Model**: *Value models* produce a single value by summarizing a function (specified as a cell expression) that computes a value for each cell in the landscape. The resulting value is displayed in the dialog box when "Run Edits" is pressed.

   Value models are useful for two main purposes:
   - Queries: using queries to calculate the value of one or more rasters (e.g. maximum stand age). This can be done over all cells or over a selected sub-rectangle and/or filtering condition; and or
   - To modify and create rasters: using expressions to assign values in selected cells (e.g. to create new layers or repair minor grid errors). *Note: modified grids will not be saved to disk unless the user presses the Save menu item.*

   To use the value model dialog, set up the function and parameter, as described below, and press "Run Edits". When finished, the dialog can be closed with the Cancel button.

   The following parameters can be used to express a value model:

   i) **Operation**: Chose the desired summary function: mean, min (minimum value), max (maximum value), sum, product, nth root (where n is the number of values computed). These will apply the operation to the value of the main function over cells in the region (e.g. the average stand age in a given management unit). The "single evaluation" function will run the main expression just once instead of over a region.

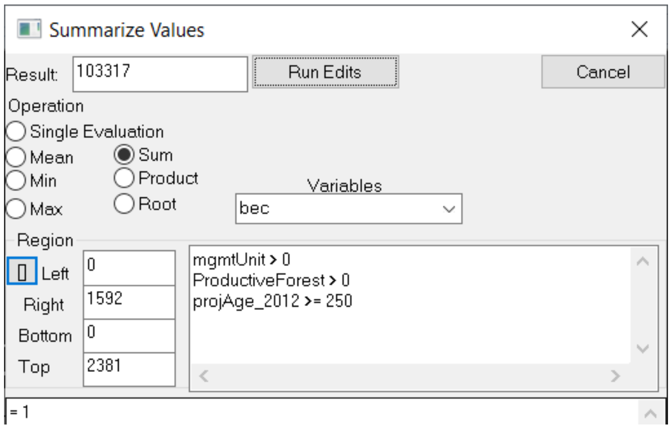
   ii) **Region**: By default, the region is the entire landscape, but the rectangular region can be changed manually (by entering different bounding row and column value) or by using a "selection box" (see next subsection) and pressing the "[]" button to refresh.

   The edit box associated with the region allows users to specify a decision to limit the computation to specific cells (the "region decision"). The default is "TRUE", meaning all cells in the bounding region are included. Expressions in this edit box must evaluate to true or false.

Note: the region decision is implicitly embedded within a logical conjunction (i.e. "AND"). It can consist of one or more expressions, all of which must evaluate to TRUE for the cell to be included.

For example:

```
mgmtUnit EQ 1
ProductiveForest > 0
projAge_2012 >= 250
```



```
mgmtUnit > 0
ProductiveForest > 0
projAge_2012 >= 250
```

```
= 1
```

In this case, only cells with a value for management unit, in productive forest and with stand age 250 or older will be evaluated. All other cells in the region rectangle are filtered out.

iii) **Variables**: is a drop-down list of the names of the raster views that can be used for spatial variables in the expression in both the main edit box and the region decision condition. For example, if the cell expression requires a layer variable called "topography", then this variable will appear in the Variables list (which is implicitly linked to the view of the same name).

iv) **Main Cell Expression Edit box**: This box allows users to enter an arbitrary *cell expression*, and can be used much like a console for querying or modifying the landscape state.

A cell expression has three sub-sections:

- Preliminary expressions: assignments and other expressions that are evaluated prior to the main function. There can be as few or many preliminary expressions as desired.
- Main function: starts with an equal sign, and the expression must evaluate to a number used as the main result for each cell (which are then combined using the operation). There must be exactly 1 main function (even if it is trivial, such as "= 1").
- Consequent expressions: assignments and other expressions that are evaluated after the main function. There can be as few or many consequent expressions as desired.

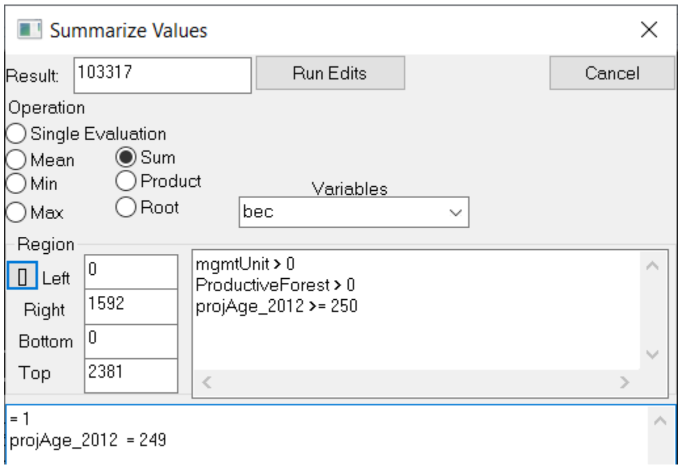
A value model works as follows when pressing "Run Edits". In each cell in the region at which the decision evaluates to TRUE, the main cell expression is evaluated. The result for the cell is the value for the main function. These values are combined using the operation (e.g. mean or max), and the overall summarized value is displayed in the "Result" field.

Since main cell expression is evaluated in each cell in the region, spatial values can be used to both compute the main function, as well as to modify values in each of those cells. Any expression from the SELES landscape event expression language can be used (see Appendix 1). The region can be as small as a single cell, which allows fine or coarse-scale queries and editing.

Note: comments can also be included in the edit boxes, to help clarify complex evaluations (which may be copy-pasted from files).

Consider the following main cell expression using the region in the previous example, and the Sum operation:

```
= 1
projAge_2012 = 249
```



Result: 103317

Operation: [✓ Sum]

Variables: bec

Left	0
Right	1592
Bottom	0
Top	2381

```
mgmtUnit > 0
ProductiveForest > 0
projAge_2012 >= 250
```

```
= 1
projAge_2012 = 249
```

Pressing "Run Edits" will result in two outcomes:

(a) The number of cells that meet the region decision will be counted (i.e. each cell in a management unit, in productive forest and with stand age at least 250 adds 1 to the sum); this result is shown in the result box at the top.

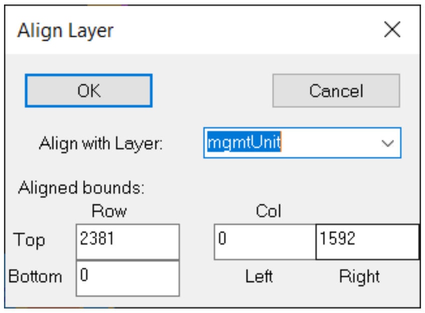
(b) In all cells that meet the region decision, the value of *projAge_2012* is set to 249. Note that if the Value model is run again, the result will be 0 (since no cells will meet the region condition).

e) **Align Layer**: Use to align the currently selected raster view with the layer selected in the "Align with Layer" drop-down list by matching their georeferencing and ensuring the same number of the rows and columns. If the layers are already aligned, no action is performed and a message is given. If the layers are not aligned, then a new view is created with the prefix "*aligned*" added to the raster view name. The contents of the new view will be the original view *clipped* to the new georeferencing. Any cells within the clip region, but not covered by the original layer will have a value of zero.

Rasters in SELES must be aligned to run properly. That is, all raster layers must have the same extent (number of rows and columns) and must represent the same geographic location. This permits simulations to access all relevant information for a cell with a single location index.

Once the layer to align with has been selected, the top-left and bottom-right bounds of the original layer are shown relative to the georeferencing of the "Align with Layer" (called the *aligned bounds*). Any portion of the layer outside the range of the "Align with Layer" is clipped. Once an "Align with Layer" has been selected, pressing the OK button will create a new view with the same name and prefixed with "*aligned*" for the aligned raster.

For example, to align a bec layer with a mgmtUnit layer, select the bec layer, opening the Align Layer dialog and select mgmtUnit from the "Align with Layer" list.



Align with Layer: mgmtUnit

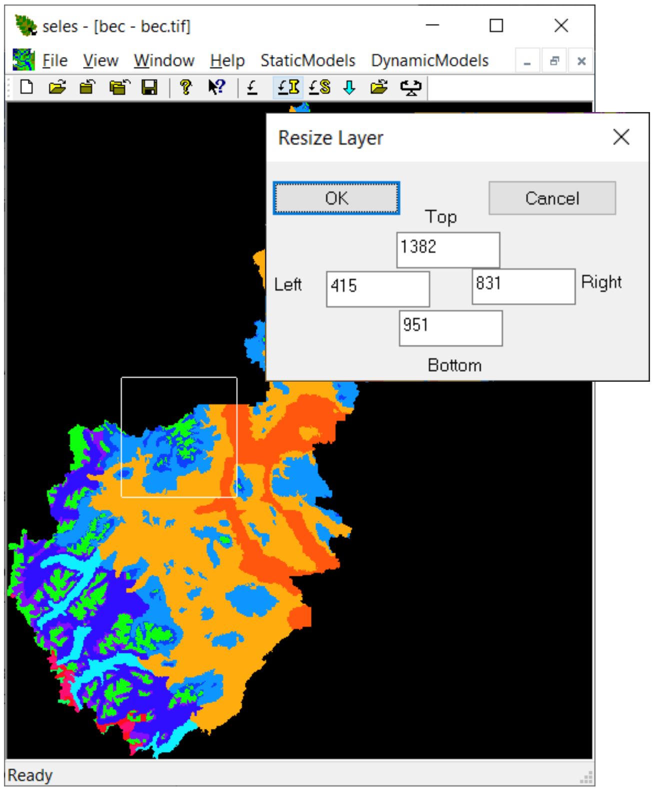
Aligned bounds:

	Row	Col	
Top	2381	0	1592
Bottom	0	Left	Right

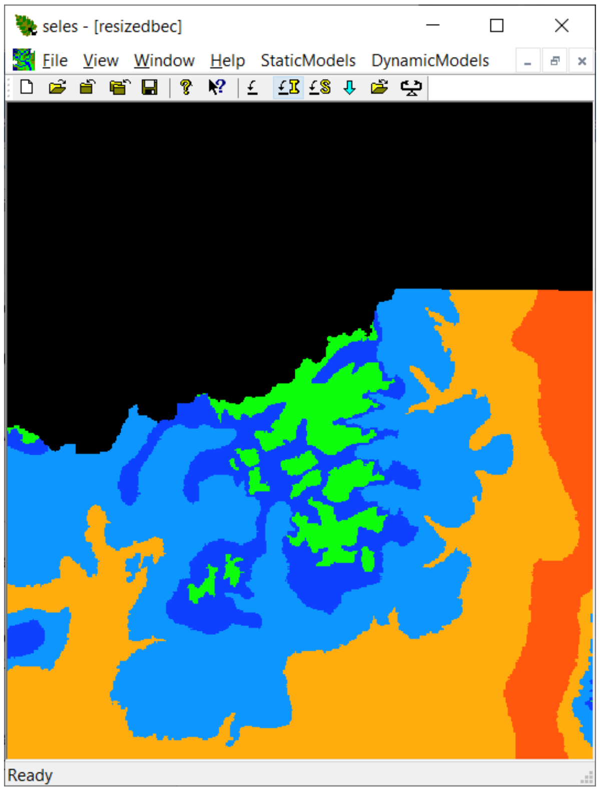
If the layers are already aligned, a warning message will be displayed ("*Error: The two layers are already aligned*"). If the user presses the OK button anyways, the resulting aligned layer will be identical to the original.

Note: to use a Value model in the new aligned layer, the un-aligned layer must be closed (since the Value model requires all layers to match).

f) **Resize Layer**: Use to change the number of rows and columns of the currently selected raster view, without changing its resolution. The dialog allows users to enter bounds (top, left, bottom, right) for the new raster (which are relative to the current raster). The default bounds are those of the "selection box" (see next section). When the OK button is pushed, a new view with the same name and prefixed with "*resized*" is created for the resized raster, as shown in the following example, in which this function is used to create a grid of a selected area in the original layer (shown outlined in white in the left layer; result shown on right).



Top 1382
Left 415
Right 831
Bottom 951



g) **Rescale Layer**: Use to change the resolution (cell size) of the currently selected raster view.

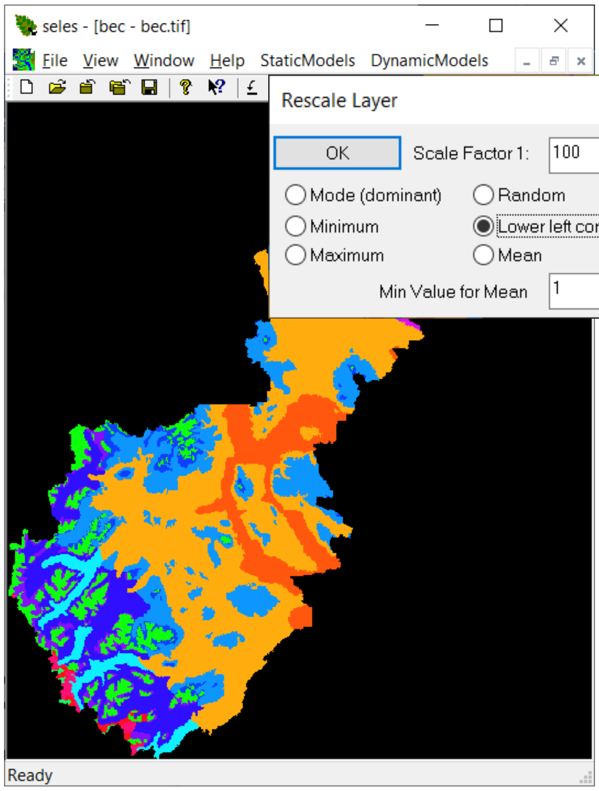
To *increase* cell size: enter a “Scale Factor” that is > 1 (this is the ratio between the new and current linear cell dimensions). For example, a scale factor of 2 will scale cells of size 100m x 100m to cells of size 200m x 200m (a 1:2 linear scale, meaning 1 new cell has the width/length of 2 original cells, resulting in ¼ as many cells in total).

The way that the values of multiple cells are combined into 1 is specified by check boxes:

- Mode: the dominant (most frequent) value of the smaller cells.
- Minimum: minimum value of the smaller cells.
- Maximum: the maximum value of the smaller cells.
- Random: the value of a randomly selected smaller cell.
- Lower left corner: the value of the small cell at the lower left corner of the new larger cell.
- Mean: the average value of the smaller cells (excluding cells with values below the “Min. Value for Mean” (e.g. to exclude no-data values).

When rescaling multiple rasters for a data set, it is often preferrable to simply select the lower left corner option to preserve the consistency of the underlying polygons. Using other options may end up with resulting values from different polygons.

When the OK button is pushed, a new view with the same name and prefixed with “*rescaled*” is created for the rescaled raster, as shown in the following example, which rescales cells of size 100m x 100m (1-ha) by a factor of 100 to cells of size 10km x 10km (100 km^2).

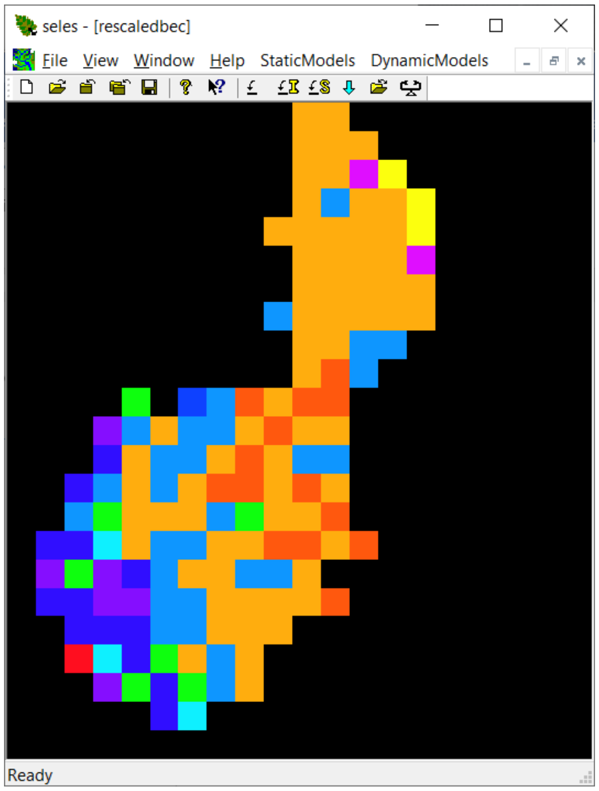


Rescale Layer

OK | Scale Factor 1: 100

Mode (dominant) | Random
Minimum | [✓ Lower left corner]
Maximum | Mean

Min Value for Mean 1



To *decrease* cell size: enter a “Scale Factor” that is < 1 (this is the ratio between the new and current linear cell dimensions). For example, a scale factor of 0.5 will scale cells of size 100m x 100m to cells of size 50m x 50m (a 2:1 linear scale, meaning 2 news cells have the width/length of 1 original cell, resulting in 4 times as many cells in total). When decreasing cell resolution, the rescale functions are not used, as values from the new smaller cells are simply transferred from the previous larger cells. Also, the rescaled layer will look identical to the original, but will have more, finer resolution cells.

In general, rescaling should be done based on whole multiples of cells: When increasing cell size, and integer scale factor is recommended. When decreasing cells size, a rational number is recommended (e.g. ½, ¼, …).

1.6 Dynamic Models Menu

Dynamic models are generally related to temporal dynamics in a simulation.

a) **Dynamic Site Model:** Used to generate stochastic, dynamic pattern sequences based on relative probabilities using values from other rasters at each site (rarely used for applied models). Not needed to run most models.

b) **Dynamic Value Model:** A Dynamic Value Model is a value model that is re-computed periodically during a simulation. Similar to a value model, but with the addition of a scheduling time period to specify how frequently the value model is to be re-run during a simulation. Not needed to run most models.

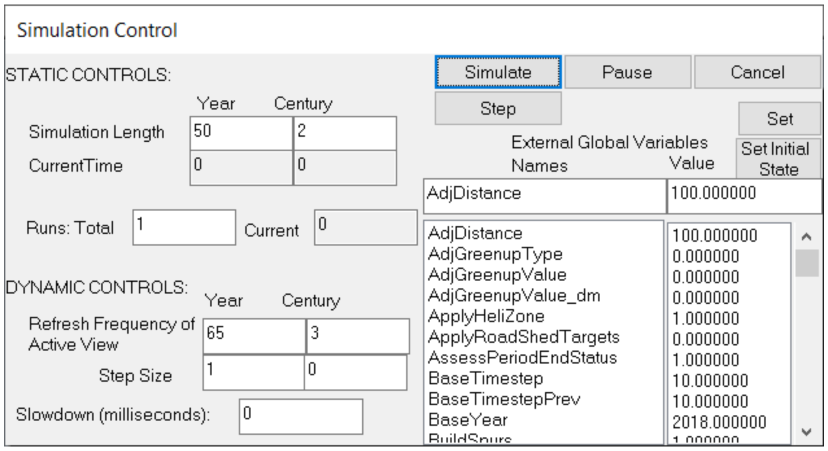
c) **Model Output:** Model Output allows intermediate states of dynamic layers to be saved during a simulation. Such outputs are usually schedule more in a scenario file or a dynamic model file in a more automated manner. Not needed to run most models.

d) **Seles Model:** A SELES Dynamic Model specifies the information required to set up a simulation, including the landscape events, spatial state, global variables etc. This is usually specified in a .sel file in a more automated manner. Not needed to run most models.

e) **Simulate** (also 3rd to last icon on toolbar as a downward pointing arrow)**:** This displays the main simulation control dialog. This dialog is also opened when a simulation is started from a scenario file. Simulations can only be run after the dynamic model and its inputs are loaded.

The dialog allows control of a simulation as follows:

i) **Simulation Length**: Specifies the length of time to run a simulation. The time unit names are usually defined in the model (.sel) file; the default time unit is "day", and meta-unit is "year" (defined as 365.25 days). "Current Time" shows the current time of a running simulation. Note that individual events may use any real-value time increments (which may also be variable or stochastic).



	Year	Century
Simulation Length	50	2
CurrentTime	0	0

Runs: Total 1 Current 0

	Year	Century
Refresh Frequency of Active View	65	3
Step Size	1	0

Slowdown (milliseconds): 0

External Global Variables

Names	Value
AdjDistance	100.000000
AdjDistance	100.000000
AdjGreenupType	0.000000
AdjGreenupValue	0.000000
AdjGreenupValue_dm	0.000000
ApplyHeliZone	1.000000
ApplyRoadShedTargets	0.000000
AssessPeriodEndStatus	1.000000
BaseTimestep	10.000000
BaseTimestepPrev	10.000000
BaseYear	2018.000000
BuildSpurs	1.000000

ii) **Runs**: Total: Specifies the number of Monte Carlo simulation replicated runs. This can only be set before a simulation starts. "Runs: Current" is an output value that shows the current Monte Carlo run number.

iii) **Output Frequency (Refreshed Frequency of Active View)**: Specifies how often the display of the currently selected raster view is updated.

iv) **Step Size**: Specifies how the number of time-steps to be made when "stepping through" a simulation.

v) **Slowdown**: is the number of seconds to slow down a simulation after each time step (which can be useful for debugging).

vi) **Simulate**: Starts a simulation (or a series of simulation replicates). This button will change to a "Stop" while the simulation is running (and back to "Start" when the simulation is completed). Pressing "Stop" will terminate the simulation.

vii) **Pause**: Pauses a simulation. While paused, this button will change to "Continue". Pressing "Continue" will resume the simulation. Note that the simulation will be paused at the next opportunity, but cannot interrupt partially complete functions

viii) **Step**: Allows a simulation to be "stepped through" in discrete time steps. If the simulation has not yet begun, then this button will start it. The Step button can also be pressed at any time during a simulation. After the time period specified by "Step Size" has passed, the simulation will pause and the "Pause" button will change to "Continue". Pressing "Step" again will continue the simulation for "Step Size" time units, and pressing "Continue" will continue the simulation without stepping.

ix) **Cancel**: If a simulation is currently running, Cancel is the same as Stop. If a simulation is not running, then Cancel will close the Simulation Control Dialog.

x) **External Global Variables (viewing and setting)**: This shows the list of global variables defined by the loaded dynamic model (.sel file) and the current value of each. If a simulation is not running, the value is the initial state value. During a simulation, the values of dynamic global variables are periodically updated in the dialog box. Current values can be viewed by scrolling through the list. The currently selected variable (selected by a single mouse click on the variable in the list) is shown in the single-item above the full list..

Values can be changed in the following ways after selecting the global variable so that it appears in the single-item above the full list:

Before a simulation is started: The initial value can be changed by entering a new value in the “Value” edit box and pressing “Set Initial State”. This will change the initial condition used at simulation start-up. Note that pressing “Set” when a simulation is not running will have no effect, since when a simulation is started, the value is always set to the initial state.

While a simulation is running: The current value of a selected global variable can be changed by entering a new value in the “Value” edit box and pressing “Set”. This will change the value at that point in the simulation. The model may make changes to the value. When a new simulation is started, the value will revert to the initial state. Note that pressing “Set Initial State” during a simulation will cause changes when the *next* simulation is start (and have no effect on the currently running simulation).

f) **Reload current dynamic model** (no menu item, but 2nd to last icon on toolbar)**:** This toolbar button will reload the currently loaded dynamic model (.sel file). This is useful during model development if minor changes to the .sel file have bene made. This button should only be pressed when a simulation is not running. Not needed to run most models.

g) **Simulation Probe** (also last icon on toolbar as a stethoscope)**:** Brings up a dialog box that allows probing of internal values in landscape events/agents during simulation. The desired landscape event can be selected from a drop-down menu as well as the desired property context. Values for the agent/cell, cluster/group, event/population and cell variables are shown in different areas of the dialog. If changes happen too fast to see, there is an option to slow down the simulation. Care must be taken when using the probe, since it must modify events to operate. As a consequence, making too many changes on the dialog runs the risk of crashing a simulation due to timing of the user interface operation and the separate simulation engine processes. Not needed to run most models.

h) **Model Report:** Saves a dynamic model report to “SelesModelReport.txt” and “SelesModelReport2.txt” files in the current directory, which include the spatial and global variables and constants in the model state-space by type, as well as the landscape events that include them. The files indicate if a variable or constant appears on the left-hand side (LHS) and/or right-hand side (RHS) of an expression in the named event (i.e. whether or not the event modifies or is influenced by the variable). The only difference between the files is format – the first is easier for browse while the second is more suitable for loading in a spreadsheet. This is useful for model verification, since at a glance it is easy to see how variables are used and to identify problem with state set up. Not needed to run most models.

i) **Simulation Priority:** Allows users to set the priority level for the simulation engine. Not needed for modern multi-processor machines.

2 SELES Mouse and Keyboard Controls

This section assumes that one or more raster views are open.

2.1 Pan and Zoom

On the currently selected raster view, the following mouse and keyboard controls can be used to modify how the raster is viewed:

- Left mouse button down + moving: pan raster view image
- Right mouse button down + moving left/right: zoom out and in, respectively
- Double-right-click: reset display (remove panning, zooming and reset selection box to entire raster)

Note: pan/zoom with the legend open can be used to get a detailed view of grid cell values.

2.2 Selection box

The “selection box” is a rectangular area of the raster view used as the default region for a value summary model and for raster resizing (to focus queries on sub-areas, and to select a sub-area to for resizing).

The default selection box is the entire raster bounds. When set to the default, the selection box is not shown. Double-right click to reset the selection box to the default.

On the currently selected raster view, the following mouse and keyboard controls can be used to change the selection box (shown as white rectangle):

- Left button down, then right button down + moving: set a corner of the selection box, with mouse movement defining the opposite corner.
- Right button down, then left button down + moving: retain the corner last set as above, with mouse movement re-defining the opposite corner.

3 LSEditor

The LSEditor is a text editor executable (LSEditor.exe) designed to work with SELES files, including scenario files (.scn files), dynamic model files (.sel files) and landscape event files (.lse files). The LSEditor uses colours to help separate keywords, comments, variable types and other commands. It also embeds the same language parser as in the SELES10 application to allow checking syntax and consistency of dynamic model and landscape event files.

The colour coding common to all three types of files is:

- Green: comments
- Blue: keywords

If colours do not appear to be correct, colour coding can be refreshed by pressing the colour format button ("paint bucket") twice: turn colour formatting off and back on.

Most menu options are fairly standards for file open, close, saving, copy, paste, etc. The print options are not functional. There are several options specific to dynamic model files and landscape event files, described in the applicable sub-section.

Note: the undo function sometimes loses a character, so check when using undo).

3.1 Scenario script (.scn) files

The colouring helps to avoid keyword typos and separate comments from functional code.

Testing the syntax of a scenario script file essentially requires that it be run, which is the easiest way to test (i.e. by opening in SELES normally or using the "Dry Run" option).

3.2 Dynamic model (.sel) files

The colouring helps to avoid keyword typos and separate comments from functional code.

Dynamic model files basically define the spatio-temporal state space of a model: time units, landscape events to include, spatial variables and constants, global variables and constants, legends, macros, scheduled dynamic raster output, and external script variables used in the file.

The Parse menu (and two parse buttons on the toolbar, identified with a pink "P") can be used to test the syntax of a dynamic model file. Parse (or "Check…") assesses syntax and returns either with a message indicating success, or an error message and highlighting text near the detected error. The second option, "Parse and Save", does the same, except it also saves the file on a successful parse.

For a dynamic model to be parsed, it requires acceptable default values for any external script variables (which may be used to set files names, input file paths, etc.).

3.3 Landscape event (.lse) files

The colouring scheme is enriched for landscape event files to also include colours for the global and local state-space:

- Dark blue: Property names (distinguished from the blue of other keywords)
- Black (italic): Built-in system variables (e.g. *Time*, *Location*)
- Red: Global Variables
- Red-brown: global constants
- Pink: Spatial variables and constants (called "layers" in landscape events)

- Orange: Local variables (global in scope, but local to the landscape event)
- Light blue: Event variables (variables unique to each event instance)
- Medium blue: Cluster variables (variables unique to each cluster, such as a logging or wildfire patch)
- Light green: Cell variables (variables unique to each active cell in an active cluster, such as a cell on an active fire front)
- Yellow: Output file variables (used to output fields to tables)
- Black (non-italic): Temporary variables (global in scope, but only existing in a temporary context)

On occasion the colour scheme can be incorrect after multiple changes. If unsure, press the "colour bucket" (or View: Format) twice: to disable and re-enable colouring.

The parse options described for dynamic model files can be applied to landscape events. In addition, if a dynamic model file is parsed first, the defined state-space is retained in memory and used to assess the consistency of a landscape event with the state-space when parsed.

Other menu and toolbar options are somewhat out of date, but can be useful on occasion:

- File: Save Parse: gives an error message, but saves, then loads how the parse interprets the files. Useful on occasion to check for precedence ordering of complex functions (since the saved parse uses exhaustive parentheses). This option is rarely needed.

- Edit: Expressions (also on the toolbar as "*f(x)*"): provides syntax of most available functions and expressions that can be used in a landscape event (see Appendix 1).

- Edit: Properties (also on the toolbar as "*x* ="): provides syntax of most available landscape event properties.

- Edit: Variables (also on the toolbar as "*X*"): Provides a list of variables, by type, included in the landscape event. This dialog is somewhat outdated. Global constants and variables are not separated, and system variables are incorrectly shown as "multi-layer" variables.

- The + and – signs on the toolbar can be used to increase/decrease font size.

PART 3. SELES: Model Structure and File Syntax

1 General Dynamic SELES Model Structure and File Syntax

A dynamic SELES model consists of a set of interacting files of the following types (Figure 7):

- **Scenario scripts** (.scn files) consist of a sequence of commands, written in the SELES *scenario scripting language*, to load grids and models, run simulations, change parameters and working folders, load sub-scenarios, etc. See Section 2 of this Part.
- **Input (and output) raster grids** can be in a variety of formats, although GeoTiff format is recommended. See Section 3.2 of this Part.
- **Dynamic model files** (.sel files) define model state-space, written in the SELES *model configuration language*, which includes the spatial and aspatial constants and variables, and landscape events. See Section 4 of this Part.
- **Landscape events** (.lse files) define the dynamic processes acting in the landscape, written in the SELES *modelling language*, as a set of properties and associated functional and procedural expressions. See Part 1 and Section 5 of this Part.
- **Legends** define named constants associated with values in grids and that can be used in input tables in place of fixed numbers. These are either integrated with a grid file or in a separate legend file. See Sections 3.2 and 4 of this Part.
- **Input table files** are tab-separated text (.txt) files used to load tables of input parameters. See Section 3.1 of this Part.
- **Macro files** (.ce files) are sets of functional expressions that can be used as parameters or to group functions used in multiple locations in a model. See Section of this 6 Part.

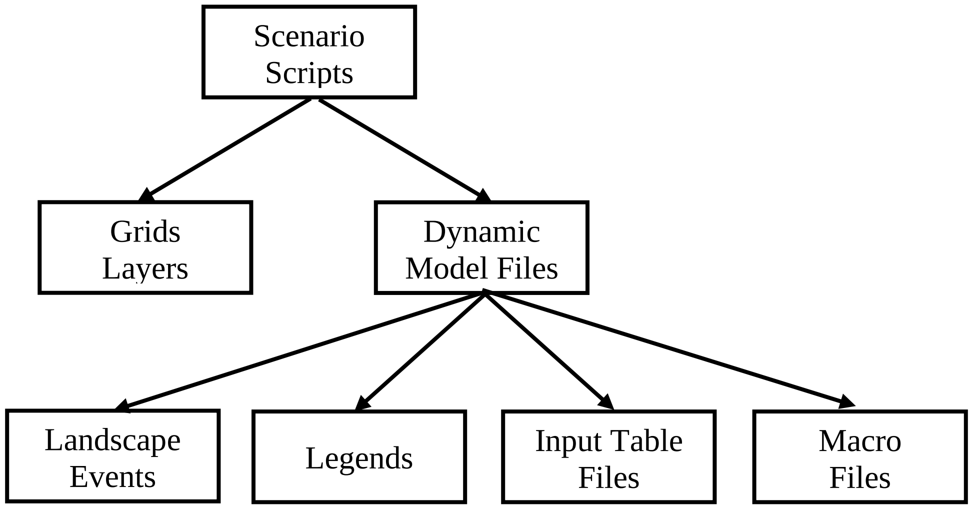


Figure 7. SELES models consist of scenario files that load spatial grids (layers) and dynamic model configuration files, and issue various commands. Dynamic model files in turn load landscape events, legends, input tables and macro files.

PART 3. SELES: Model Structure and File Syntax

1.1 How to Understand Syntax Specifications

The following describes the general rules used to specify syntax in the different types of files.

Syntax specifications may include:

- Punctuation, where shown, must be provided as is.
- Numeric values are prefixed with "#" (e.g. "#Factor")
- Optional elements are surrounded by braces (e.g. "{ variables}")
- An item followed by a superscripted asterisk (*) indicates that the item may be repeated zero or more times.
- An item followed by a superscripted plus sign ($^+$) indicates that the item may be repeated one or more times (i.e. at least once).

1.2 General syntax rules common to all SELES language files (.scn, .sel, .lse, .ce)

Model files generally consist of comments, keywords, commands, expressions and labels.

- **Comments** are for model documentation only, have no effect, and can be placed anywhere in a model file. Comments are surrounded by "/*" and "*/" (e.g. /* This is a comment, and will be ignored by the parser */). If the parser encounters "//", the remainder of the line is treated as a comment. Well-documented models often consist of at least 50% comments.

- White space (extra spaces, tabs, or carriage returns) is ignored and can be used to help format models. Use of consistent indentation (e.g. three spaces for each nested level of sub-expressions) is critical to model readability. Spaces are better to use than tabs since different programs may interpret tab depth differently.

- **Keywords** and other verbatim text are shown in all capitals (e.g. "GLOBAL VARIABLES:"), even if lower case is also acceptable in actual use.

- The parser is not case sensitive, but otherwise keywords must be spelled exactly as shown. Portions of keywords shown in *italics* are optional (e.g., the syntax description "MIN*IMUM*" indicates that either "MIN" or "MINIMUM" is valid).

- **Commands**, except the model type declaration at the top of the file, are generally optional. The default value for commands is given with the syntax description.

- Each **command and expression** must generally be on a separate line (i.e. separated by a carriage return). Some expressions may span multiple lines (e.g. "IF" expressions).

- The last command or expression of a file must end with a carriage return (i.e. the last line of the file should be empty). When the LSEditor cannot locate a syntax error, this should be one of the first items to check.

- **Labels** that are not all upper case (i.e. not keywords) indicate structures defined elsewhere in the section.

- Labels enclosed in angle brackets (e.g. "<*Variable*>") indicate user-defined items for which the modeller must substitute an appropriate value (e.g. variable names, time unit names, input file names or numeric values).

- Labels preceded by a "#" indicate numbers. If not obvious, command and expression descriptions indicate if these are restricted to integers. In most cases, numbers, constants and arithmetic expressions can be used. For example, the following are valid number expressions: 2, 3.5 1/365.25, (5*-3)^4/6, MaxAge+1 (where MaxAge is a defined constant name), and 1/NUMCELLS (where NUMCELLS is a built-in constant name).

- Number Pairs and Number Lists may be enclosed in optional brackets "()", and each number in the list must be separated by a comma (e.g., "(1,2,5,6)" or "1,2,5,6")

- Labels preceded by a "%" indicate a logical value (true or false). Logical values may be any of: '0', 'OFF', or 'FALSE' to indicate false; '1', 'ON', or 'TRUE' to indicate true.

- Text enclosed in double quotes indicates a literal text value (e.g. "This is a literal text string").

- File names must be specified as literal text if they contain any spaces (as a general rule, spaces in filenames should be avoided).

2 Scenario script files (.scn)

A SELES scenario script executes a sequence of commands, in the order specified, to perform tasks such as loading spatial information, opening models and running simulations. Scenario files are specified using a scripting language for setting up and running experiments in SELES. This section describes the syntax and meaning of the commands available for scenarios.

2.1 General Syntax Rules

- Every command must be placed on a separate line (i.e. separated by a carriage return)
- The last command must also have a carriage return (i.e. the last line is blank)

- All **KEYWORDs** are shown in upper case and must be spelled exactly as shown although the scenario language is not case sensitive
- Labels enclosed in "< >" indicate that the modeller must substitute an appropriate value
- Values preceded by a "#" must be integers, those preceded by "#.#" may be any floating point (real) value
- Values preceded by a "%" are logical values (true or false)
- Values enclosed in double quotes (" ") indicate text strings or filenames
- All other values are identifiers (e.g. variable names, file names)

- **White space** (extra spaces, tabs, carriage returns) is ignored

- **Comments** can be placed anywhere in a scenario file and have two forms: Any text enclosed by "/*" and "*/" is treated as a comment (usually used for multi-line comments), while any text to the right of "//" is treated as a comment (usually used for single-line comments or comments following a command). For example:

```
/* This is a comment */
x = 1 // this is a comment, but "x = 1" is not
```

- **File names and folder paths** may optionally be enclosed in double quotes. File names may include a folder path. If spaces or unusual characters are used then use double quotes. Example paths:

```
..\outputs\scn1
"..\my outputs\example scn1"
```

- **Logical** values (also called Boolean values) may be any of the following:

  '0', 'OFF', or 'FALSE' indicate false
  '1', 'ON', or 'TRUE' indicate true

2.2 Scenario Commands

A scenario file must begin with either (the latter is preferred):

```
Scenario Information
```

or

```
Seles Scenario
```

This may be followed by the following types of commands, in any order and number. Commands are processed in the order encountered.

- Load GIS rasters:

```
<FileName>
<ViewName> = <FileName>
```

Raster file names can be optionally preceded by an identifier to name the loaded raster view, otherwise the name is the name of the raster file. For example:

```
Elev100.tif
Elevation = Elev100.tif
```

The first command opens the raster in file "Elev100.tif" and displays it in a window (raster view) of the same name without the suffix. The second command opens the same raster file, but displays it in a window with the name "Elevation". Rasters can be in GeoTiff (preferred) GRASS, ARC GRID ASCII or ERDAS formats.

If a raster format is real-valued, then a real-value raster view is created. Otherwise an integer-value raster view is created. Integer-value rasters are easier to view and work with in general.

Appending an asterisk and numeric expression (e.g. "* 10") to one of the above commands will multiply each cell value read by the amount of the expression. This is useful to scale real-value raster files into fixed point integer raster views. For example:

```
SiteIndex10 = SiteIndex.tif * 10 // Load site index scaled by a factor of 10
```

- Save raster views:

```
SAVE <ViewName> <FileName> <Type>
```

Saves the raster in the specified view to a file. The file type must be one of GeoTiff, GRASS, GRASS COMPRESSED, GRASS ASCII, ARC ASCII, ERDAS8, or ERDAS16. For example:

```
Save StandAge grids\StandAge_year400.tif GeoTiff
```

- <u>Open SELES dynamic model files</u>:

```
<FileName.sel>
```

Load a dynamic model (.sel) file, which defines the spatio-temporal state space (including spatial variables and landscape events). For example:

```
ForestEstateLM.sel // Load Forest Estate landscape model file
```

Prior to loading a dynamic model, all required inputs referenced by the dynamic model file must be loaded, including:

- Raster views used as spatial variables and spatial constants
- External scenario script variables used for input file paths, input file names, initial parameter settings, etc.

Note that at most *one* dynamic model is active at a time. If a subsequent dynamic model is loaded, it supersedes a previously loaded dynamic model.

- <u>Set spatial dimensions</u>:

```
Model Dimensions: <ViewName>
```

This command sets the dimensions (number of rows and columns) of the scenario to the dimensions of a currently loaded raster view. This applies to a dynamic model file when loaded, so dimensions should be specified prior to loading a dynamic model file. For example:

```
Model Dimensions: StudyArea // set dimensions to study area raster view
ForestEstateLM.sel // Load Forest Estate landscape model file
```

Any an input raster that does not have the same dimensions as the dynamic model will be sub- or super-sampled to obtain a set of spatial state layers that all have the same dimensions, and a warning message will be given. In general, matching georeference information for the spatial layers should be done prior to loading a scenario in SELES.

- Simulation control:

```
SimStart #RunLength
SimStart #RunLength #Runs

<Landscape Event FileName> %UseEvent
```

All of these commands must be issued after loading a dynamic model file.

The first command starts a single simulation of the currently loaded dynamic model for #RunLength number of time units (which may be any real value). The meaning of a time unit is normally defined in the dynamic model file (e.g. 1 time unit may be defined to represent a year). The second command iteratively starts #Runs number of runs (replicates), each of which will run for the specified duration. For example:

```
SimStart 400 // run simulation for 400 time units
```

Note that simulations start at time 0, and so scheduled events (by landscape events, by the dynamic model file or by the scenario script) are processed up to, but not including, the #RunLength time.

The last command allows specifying landscape events (loaded in the current dynamic model file) to be activated or deactivated, and can be useful for testing. For example:

```
Logging.lse OFF // disable logging event
```

- Expressions:

```
Expression
```

Expressions are used to assign values to global variables used as parameters, to scenario script variables (see subsequent section), for conditional comments, etc. Expressions evaluate to either a numeric or a logical value.

The following expressions are supported:

```
#Value
<GlobalConstant>            // Must be defined in loaded dynamic model
<GlobalVariable>            // Must be defined in loaded dynamic model
(Expression)                // Parentheses to clarify expression evaluation
Expression + Expression
Expression – Expression
Expression * Expression
Expression / Expression
Expression ^ Expression     // exponentiation: second value is the exponent
```

Expression % *Expression*	// modulo: remainder on division by the second value
| *Expression* |	// absolute value
ROUND(*Expression*)	// nearest integer value
CEIL(*Expression*)	// nearest higher integer value
FLOOR(*Expression*)	// nearest lower integer value
Expression AND *Expression*	// test logical conjunction
Expression OR *Expression*	// test logical disjunction
NOT *Expression*	// test logical negation
Expression EQ *Expression*	// test equality (equivalent to "==")
Expression == *Expression*	
Expression NEQ *Expression*	// test inequality (equivalent to "!=")
Expression != *Expression*	
Expression < *Expression*	
Expression <= *Expression*	
Expression > *Expression*	
Expression >= *Expression*	
Expression << *Expression*	// Shift bits left by specific number of positions
Expression >> *Expression*	// Shift bits right by specific number of positions

Normal precedence rules are applied (e.g. "x + y * x" is evaluated as "x + (y * z)" and "a AND b OR c is evaluated as "(a AND b) OR c"). However, use parentheses to ensure that complex expressions are unambiguous, especially when mixing expression types where precedence rules are not clear (e.g. "x + y > z" could be interpreted as the logical test "(x + y) > z" or as the sum "x + (y > x)").

Examples:

```
TimeStep - 1
(Time > 50) AND (Rate < 100)
```

- <u>Setting global variables used as parameters</u>:

  <*Variable*> = *Expression*
  <*Variable*> = *Expression*, *Expression*, …, *Expressions*
  <*Variable*> = [*Expression*, *Expression*, …, *Expressions*]

When a dynamic model is loaded, it may create global variables and set their initial values (i.e. the value set at the start of a simulation). These commands can be used to change the initial value for a global variable used for a parameter *after loading the dynamic model* (since prior to loading the dynamic model file, the global variable is not yet defined). The latter two forms are for one-dimensional array variables, where an expression can be specified for each entry.

If the global variable being assigned is used as logical value, then ensure that the result evaluates to true or false. For example, if variables x and y both have value TRUE (or, equivalently, 1), then "x + y" evaluates to 2, while "x OR y" valuates to TRUE (or 1).

- <u>Scenario script variables (also called string variables)</u>:

```
$Label$ = <Value>
```

This command sets a script variable named "Label" to the value on the right-hand side. If the script variable does not already exist, it will also be created. The dollar signs ($) must surround the variable name to identify it as a script variable (as opposed to, for example, a raster view name, or a global variable defined in a .sel file). The value can be text, or an expression that evaluates to a numeric or logical value.

In general, after being created, a script variable can be placed wherever a numeric or text value is required, and the script variable name will be replaced by its value. When used in place of a number (e.g. in an expression), the script variable must represent a numeric value. When used in as text, the variable can be concatenated with other text or string variables. For example:

```
$x$ = "meso"
$y$ = 5
MesoView = $x$$y$.tif
```

This will attempt to open a raster file named "meso5.tif" into a view named "MesoView".

If the script variable is assigned the name of a global variable, then by default the text name of the variable is used (not its value). If the value of the global variable is desired, surround its name by "#" instead of dollar signs. In the following example, if "ForestSize" is a global variable defined in the loaded dynamic model with value 200. Then x will have value "ForestSize" and y will have value "200".

```
$x$ = ForestSize
$y$ = #ForestSize#
```

Assigning a value of "." has a special meaning: to assign the complete path of the current working directory. This is useful to be able to return to a directory after doing complex directory movements. For example:

```
$currDir$ = "."
...
cwd $currDir$
```

There are many applications of script variables, including

- To manage folders for model inputs and outputs (e.g. path to spatial inputs)
- To set up simulations to link parameter settings with output folder names (e.g. using a script variable to set a parameter as well as the name of the output folder)

 - To set up iterative simulation runs
 - To allow input file names and paths to be parameters by importing script variables into dynamic model (.sel) files

- General control commands:

```
cwd <Directory>                // "change working directory"
mkdir <Directory>              // "make a directory"

Close <ViewName>
Close All
```

The first command changes the current working directory (which starts in the folder of the scenario file). This is useful for controlling where output files are created for different experiments. The directory name on the right-hand side can be specified using a string variable (e.g. `cwd $outputDir$\$scnDir$`). Folders will be created if they don't exist. The `mkdir` command will explicitly create a directory or folder structure, which can be used to ensure that output folders exist (e.g. the folders used for scheduled raster output).

The first close command closes a specified raster view while the second closes all raster views. This is useful for freeing memory used by rasters after a simulation has run, especially when multiple simulations are processed in batch. If all raster views are closed, any loaded models are eliminated from memory.

- System commands:

```
system "<Command>"
```

This executes any Windows system command available in a command prompt, specified as a literal string. For example, sometimes it is useful to delete and rename output files. For example:

```
system "del /s harvestRate$x$.txt "
system "ren harvestRate.txt harvestRate$x$.txt"
```

If \$x\$ has value "5", then these commands delete the file harvestRate5.txt, and then renames the file harvestRate.txt to harvestRate5.txt.

Note: System commands must be used with care as they are prone to errors. System commands are unable to return errors messages (e.g. if a file delete couldn't be performed because the file is open by another program). The use of system 'commands should be avoided if possible.

- <u>Conditional commands (*if-statements*)</u>:

Conditional commands are used to run a sequence of commands only if a condition is met:

```
if (Condition Expression)
  …
else
  …
end
```

The condition can be any expression, but the result is treated as a Boolean expression. If the result is > 1, the result will be interpreted as true; otherwise it will be interpreted as false.

The commands following the "*if*" will be run only if the condition evaluates to true; otherwise the commands following the optional "*else*" will be run. Also, the "*if*" section may be followed by one or more "*else if (condition)*" sections of similar format before the optional "*else*" section (the "*else*" section, if specified, must be last). If no condition applies, then no commands will be executed.

For example:

```
wildfire.sel
param1 = 1

if (outputVar > 0)
        param1 = param1 + 1

else if (outputVar < 0)
        param1 = param1 - 1
end

SimStart 100 1
```

- <u>Conditional Looping commands (*while-loops*)</u>:

Conditional looping commands are used to run a sequence of commands repetitively (zero or more times), until a condition is no longer met:

```
while (Condition Expression)
  …
end
```

The condition is treated as a Boolean expression the same as for conditional ("*if*") commands. The commands will be repeatedly run until the condition evaluates to false. The following is an example loop:

```
wildfire.sel
cwd age1
param1 = 1
outputVar = -1

while (outputVar < 0)
     param1 = param1 + 1
     SimStart 100
end
```

This will run a series of 100-steps simulations until the global variable *outputVar* is at least 0.

It is important to ensure that the condition will eventually evaluate to false. If the condition is not guaranteed to eventually be invalidated, then this can create an infinite loop (i.e. SELES will never terminate without forcibly ending the seles.exe process).

- <u>Iterative Looping commands (*for-loops*)</u>:

"*For loops*" are used to run a sequence of commands repetitively over a range of values:

```
for ($label$ = #n1 : #n2)
…
end
```

or

```
for ($label$ = #n1 : #n2, #step)
…
end
```

The script variable $label$ successively takes on the values n1, n1+ *step*, … In the first form, the step is implicitly 1. In the second form, the step is explicit (but cannot be 0, which would create an infinite loop). The loop stops when the value for $label$ exceeds n2. Note that loops can be nested.

A *for-loop* is a specific form of a *while-loop*, which can be interpreted as:

```
$label$ = n1
while ($label$ <= n2)
  …
  $label$ = $label$ + step
end
```

The following is an example loop in which script variable x takes on values 1, 2, … 10:

```
wildfire.sel
cwd age1

for ($x$ = 1:10)
        param1 = $x$
        SimStart 100 1
end
```

- <u>File looping/wildcard commands</u>:

Another form a "*for loop*" can be used to run a sequence of commands repetitively over a set of related files. The syntax is:

```
for ($label$ = "Text")
  ...
end
```

where the text should normally have at least one *wildcard*, and the "…" represents any sequence of commands. Note that a *wildcard* is an asterisk (*) used to denote arbitrary text (e.g. "fireParam*.txt"). If there is precisely one wildcard, then the script variable will be assigned the "filler" for this wildcard. If there is more than one wildcard, then the script variable will be assigned the complete text for each match. Loops can be nested. The following is an example loop:

```
wildfire.sel
cwd age1

for ($yr$ = "age*.tif")
        StandAge = age$yr$.tif
        cwd ..\age$yr$
        SimStart 100 1
        Close StandAge
end
```

This sequence opens a dynamic model file and runs a simulation on each file named "age*.tif", where the wildcard may represent the start year. The commands create and change to different directories so that output files for each simulation get placed in separate folders.

- <u>Sub-scenarios:</u>

A scenario file can load commands from another scenario file as follows:

```
scenario: <FileName.scn>
```

The commands in the sub-scenario are loaded and processed as if they were specified at that location. Note that the path to the file name is relative to the folder in which the calling scenario file resides (not the current working directory at this point). This is because sub-scenarios are loaded when reading the scenario file and *before* running any commands (i.e. all commands are loaded in a list before starting execution).

- <u>Scheduling commands</u>:

Commands can be scheduled to be performed periodically during a simulation (e.g. to load a time series of rasters)

```
schedule(#Period)
  …
end
```

The time period is an expression that is recomputed at the start of a simulation and after each time the set of commands is run. The value indicates the time step until the next evaluation. If this is 0, it will not be rescheduled.

Note: commands must be scheduled prior to starting a simulation using *SimStart.*

The following is an example:

```
// Schedule periodic reloading each 25 years
$x$ = 1
schedule(25)
        // Re-open dynamic rasters
        StandAge = $dynamicGisDir$\StandAge_1_$x$.tif
        $x$ = $x$ + 1
end
```

This example schedules commands to be repeated every 25 time units (assumed to be years here). This example assumes that the raster files have suffixes numbered 1, 2, … that represent time periods 25, 50, etc. A script variable "*x*" is used to set up the appropriate suffix. The commands will be periodically scheduled until the simulation terminates.

A current limitation is that scheduled commands only apply to the first replicate of the subsequent simulation (i.e. don't use if multiple runs are specified in the *SimStart* command).

- Raster view display commands:

```
Minimize All
Minimize <ViewName>
Minimize Initial State
Minimize Static
Minimize
Tile
```

The first minimizes all raster views currently open. The second minimizes the specified raster view. The third and fourth respectively minimize all raster views that are part of the initial state of a dynamic model (e.g. starting stand age), or that are static in a dynamic model (e.g. study area). The next minimizes the entire SELES application. The last command tiles the windows that are not minimized.

- Appending output files:

```
Reset Output: %Logical
```

By default, each time a simulation starts (i.e. with the *SimStart* command), output files are cleared. This command can specify that output files are to be appended, rather than cleared. Care must be taken to ensure that old information from previous runs gets cleared (i.e. usually, *Reset Output* is only set to false after the first run). This command is especially useful with looping as shown in the following example:

```
wildfire.sel
for ($x$ = “age*.tif”)
        StandAge = age$x$.tif
        runId = $x$
        SimStart 100
        Close StandAge
        Reset Output: TRUE
end
```

This will place all output in a single set of files, appending values on each run after the first run. Files are cleared the first time through the loop since the *Reset Output* command is not evaluated until after the first run. The variable “*runId*” is assumed to be created by the dynamic model file and can be used to differentiate outputs (e.g. if output in a column in output files).

2.3 How scenario files are processed

- Command blocking:

Opening a scenario file in SELES will execute the commands in the order specified. While a simulation is running, some commands are blocked until the simulation has finished. In particular subsequent dynamic models cannot be loaded, rasters cannot be saved, no further simulation can be started, current working directory cannot be changed and system commands cannot be processed. If such commands follow a *SimStart* command, the command interpreter waits until the simulation has completed before continuing. In such cases, the user interface will be non-responsive (including displayed output from DISPLAY commands in landscape events and agents). As such, it is wise to ensure that a model runs without error before applying in a complex scenario.

- Command ordering relative to a dynamic model file:

Correct ordering of commands relative to a dynamic model file is critical to achieve the desired results.

- All input files (tables, legends, sub-models, macros) loaded in a dynamic model are read at the time the model (.sel) file is loaded in the scenario. This also defines and set all global variables and global constants.

  Prior to loading a dynamic model:
  - Load all rasters used by the .sel file (at least initial conditions, which may be changed subsequently but attributes used by the dynamic model file, such as the maximum value in the raster, are set when the .sel file is loaded).
  - Set all script variables used in the .sel file. Modifying a script variable after loading the dynamic model file will not affect the loading of the .sel file. Script values may be used in a dynamic model file to set input file names, folder paths to input files, etc.

  After loading a dynamic model file:
- Global variables are defined by the dynamic model, and so any changes to their initial values must be done *after* the dynamic model is loaded. In general, it is preferrable to set initial values for global variable parameters in the scenario file than to directly modify the .sel file.
- Care must be taken when changing directories (e.g. when subsequently loading raster files). Folder paths in a scenario file (e.g. to a raster file) are relative to the current directory. Folder paths in dynamic model files (e.g. to input tables) are relative to the folder in which the .sel file resides.

3 Input Data

3.1 Non-spatial Tabular Data

Non-spatial input tables can be loaded into a model in the dynamic model configuration file. This data must be in text (ASCII) format, as a tab-separated set of columns (i.e. .txt files). The first row of the file can optionally have column labels. By default, the first column is assumed to specify the row index (in which case the column indexing starts at 0 from the second column). Files can also be loaded as sequential rows (i.e. with the first row starting at index 0), in which case the column indexing starts at 0 from the first column.

3.2 Spatial Data

SELES10 supports the following raster data formats: GeoTiff[2], GRASS, ARC grid ASCII, and ERDAS (with some support for ARC grid binary). In these formats, a raster is a rectangular grid of fixed size cells, and each cell has a single value. The only limitation to the range of data that can be stored in a raster is the machine word size, which is 64-bits in SELES 10 (eight bytes).

Most rasters are integer-valued, but some formats support floating-point real values (e.g. ARC grid ASCII). Integer-value grids can be used to represent "*fixed-point real values*" (i.e. real values with a fixed precision), with a chosen precision (number of decimal places) or "unit" represented (e.g. integer mat represent metres or centimetres). For example, two decimal places can be represented by using raster units that denote 0.01 of a reference unit (e.g. the number 12.34 would be stored as 1234 in the raster). To use such a grid, cell values can be divided by 100 when retrieved from the layer and multiplied by 100 when stored. In addition, values in a real-valued raster can be automatically multiplied when loading (see the scenario script section).

SELES10 interprets raster header information for dimensions, cell resolution, and georeferencing. For GRASS binary rasters, SELES uses the same relative folders as GRASS, which stores rasters in separate files of the same name in a set of specific directories. Raster cell data is stored in a subdirectory called "cell" while the header information is stored in a subdirectory called "cellhd". When opening GRASS GIS files in SELES, refer to the raster file in the "cell" subdirectory. The other formats supported contain header and raster cell data in a single file.

In addition to the raster cell and header files, GRASS rasters can also have an associated colour lookup table and legend. These optional files are respectively located in subdirectories called "colr" and "cats" in the data set directory (either a subfolder or a sibling folder), and must also have the same name as their associated raster. If there is no colour file, then SELES uses a built-in algorithm for creating colour displays of rasters. If a colour file is found, then SELES will use these colours for displaying a raster.

[2] Preferred format that compresses efficiently, load efficiently and is easily transferred to/from GIS.

To ease manual creation and modification of colour files, we have generalized the GRASS colour file format. A colour lookup table file is basically a list of raster values followed by RBG (red, green, blue) colour triples, each with value between 0 and 255. Thus, pure red is represented as 255:0:0, while dark purple may be represented by an even mix of red and blue as 180:0:180. The following colour specifications are supported by SELES, where *index* is a possible value in a raster and *RGB* is a colon separated colour triple. Punctuation must be provided as shown.

#Index : *RBG*	set the colour of *Index* to *RGB*
#Index1-#Index2: RGB	set the colour of all values in the range [*Index1*, *Index2*] to *RGB*
#Index1: RGB1 #Index2: RGB2	interpolate the colours from *RGB1* to *RGB2* over the range *[Index1, Index2]*

If an index has more than one colour specified then the last colour given will be used. If an index has no colour specified, then the default of black will be used.

3.3 Legend Files

Legends can be used to automatically define named constants for a related set of values, which can be used in a model instead of the numeric values. This improves model readability, reduces errors, and makes models more portable (since a legend label may stay the same, but its underlying numeric value may change with different data inputs).

To be used in SELES, legend labels should be simple, and not contain spaces or punctuation. See Section 4.6 for how to define or load legends in a dynamic model configuration, and the SELES expression syntax for how to use legends for text output in files.

In SELES10, there are two types of files that can be associated with legends (aka categories):

1. *Legend files*: A legend file (typically stored in a *cats* folder) is simply a list of value-label pairs, separated by a colon:

#Number: <Label>	the value *Number* is associated with the given label.

2. *GeoTiff grids with a category table*: GeoTiff files can have legends associated with them in a raster attribute table or a category table, usually stored in an XML file with the same name prefix (but with suffix ".tif.aux.xml").

   One way to "attach" a legend to a GeoTiff in SELES is to (a) create a simple legend file; (b) open the GeoTiff grid; (c) with the raster view active, open the legend file; and (d) re-save the GeoTiff grid. This can be checked by closing and re-opening the GeoTiff file and opening the Legend view.

Either of these types of files can be loaded in a model configuration (.sel) file, which will define each label as a global constant with the specified numeric value.

4 Dynamic model configuration (.sel files)

Dynamic model configuration files (.sel files) are the primary way to create landscape dynamics models in SELES, and are used to define the spatio-temporal state space (i.e. the set of spatial and global variables and constants, combined with a set of landscape events). The state at the start of a simulation is called the initial condition.

4.1 Note on Constants and Variables

Single-value global constants are simply floating-point values that may be used anywhere a numeric value can be used. Global constants by definition are static, and cannot be modified by a model. Since they are specified by name, they improve model readability, and allow changes to be made only in one place, improving model maintenance. Global constants can be defined directly, or loaded from an input table file to create a one-dimensional array (also called a *vector*) or a two-dimensional array.

Global variables are like constants, but may be modified by one or more landscape events, providing dynamic aspatial model state. Global variables can act as model parameters to provide high-level control via the simulation dialog or within scenario scripts, as output statistics, or for communication between events. As parameters, global variables are generally static within the model (i.e. not modified by any event), but are used control as aspect of model behaviour. For example, parameters could be used to control fire frequency, amount of logging, management strategy, etc. Like global constants, global variables can also be loaded from files (but in this case, the file input defines the initial condition, which may be modified during a simulation).

Spatial Constants do not have a separate raster view for initial conditions, since it is assumed that these will not change over time. However, SELES does not prohibit a landscape event from modifying a spatial constant, but such changes will persist until the raster is reloaded. This should generally be avoided.

4.2 Dynamic Model Specification

A SELES dynamic model configures the dynamic process sub-models and the state space, and specifies the initial conditions. It consists of a set of identified *blocks* (or *sections*) that load or define the landscape events, mappings (linkages) from variables in landscape events to rasters in views (windows), legends, constants and initial values for global variables.

This section describes the structure, syntax and semantics (meaning) of commands used to create dynamic model definitions (.sel files), with a focus on recommended syntax for SELES 10 (and does not provide outdated syntax used in earlier versions). Comments can be placed anywhere in a dynamic model file, but will be ignored when describing the structure and syntax.

The first line of a dynamic model configuration file defines the model type: SELES MODEL

This is followed by an optional section to define the time units, following which additional sections can be used to define and configure the model components, which may be in any order or number with the only requirement that later sections can only refer to components defined in earlier section (e.g. it is often convenient to have more than one set of global constants definitions). Upper and lower case are interchangeable for keywords, but variable names are case specific.

The following is the general format of a dynamic model:

SELES MODEL

{ *Model Time Units* }	Define meaning of time units
{ *Landscape Events* }	Identify the landscape events in the model (.lse files), which are initiated in order listed
{ *External Script Variables* }	Used to import script variables from scenario scripts
{ *Legends* }	Define legends, or load legends from file
{ *Submodel* }	Loads a sub-ordinate dynamic model configuration file (.sel file)
{ *Global Constants* }	Define global constants and values (with single values, arrays and/or from input files)
{ *Global Variables* }	Define global variables and initial values (with single values, arrays and/or from input files)
{ *Macros* }	Identify macros (arrays of functions) from input files (.ce files)
{ *Spatial Constants* }	Define spatial (layer) constants and input rasters
{ *Spatial Variables* }	Define spatial (layer) variables and initial condition rasters
{ *Dynamic State Output* }	Specify dynamic output of rasters to file during a simulation.
{ *Refresh Rate commands* }	Specify frequency of display updates

4.3 Model Time Units

Time unit definitions only affect the user interface, but help with interpretation.

TIME UNITS: <*BaseUnit*> <*MetaUnit*> #*Factor* #*DefaultHorizon*	
	Defines the name of a single time unit, and meta-unit (e.g. days and years), as well as the default simulation length. The factor is the number of units per meta-unit.

Example:

```
        Time Units: Year Century 100 250
```

4.4 Landscape Events

Landscape events are specified individually in separate (.lse) files. The dynamic model configuration (.sel) file defines which landscape events to include.

LANDSCAPE EVENTS: { *LandscapeEventFile* }$^+$	Specifies the landscape events to include in the scenario.

A **LandscapeEventFile** takes the form:

<*FileName*.lse> { %*UseType* }	A file name containing a landscape event definition.

The optional "use type" flag is usually not used, indicates whether to load enabled (ON; *default*) or disabled (OFF). This can be changed in a scenario script.

Example:

```
        Landscape Events:
                Wildfire.lse
                Logging.lse
```

4.5 External Script Variables

This block specifies the names of script variables (created in scenario script files) that can be imported into a .sel file, as well as default values to use if they are not defined. Script variables are always surrounded by "$" symbols. They can be placed in the .sel file anywhere, and they will be replaced by their value when the .sel is loaded by a scenario. This is a *literal* replacement (i.e. as if the value was typed in the file), and so script variables can be used in places requiring text or numeric values (and can even be part of a label). If the scenario that loads the .sel file does not define one of these external script variables, then the value defined in this block will be used (and so is the same as if the default value was used instead of the script variable). However, if the scenario defines a script variable *before loading the .sel file*, then this will override the default value.

External script variables have a few advanced applications, such as:

- Making an input file name a parameter by (i) using a script variable for an input file name and (ii) setting/changing the value of the script variable in a scenario script, thereby causing a different file to be loaded when the .sel file is loaded. For example, an external script variable AAC could be defined with a default value of "AAC.txt", and a scenario could set "AAC = AAC1.txt" prior to loading the .sel file to cause the file AAC1.txt to be loaded instead of AAC.txt.
- Making a folder used in the .sel file a parameter. For example, this could be used to set the path to legends or alternate sets of input files.

```
EXTERNAL VARIABLES:
    { ExternalVariableSpec }+
```

A **ExternalVariableSpec** takes the form:

```
$<ScriptVariable>$ = "Text"

$<ScriptVariable>$
```

The value to the right of the equal sign is a default value for the script variable, which will be used if the script variable was not already defined by the loading scenario file. The text does not need to be surrounded by double quotes, but quotes can help in some complicated specifications. In the second form (no default value), an error message will result if the script variable was not previously defined before loading the .sel file.

External script variables can be used elsewhere in the file to set path names, file names, and initial values.

Example:

```
    External Variables:
        $studyArea$ = MoriceTSA
        $legends$ = ..\..\$studyArea$\gisData\grids
        $AACFile$ = AAC_TSR4
```

4.6 Legends

A legend defines a set of related named constants, as well as a lookup vector that can be used for label outputs (i.e. in DISPLAY and OUTPUT expressions in landscape events). Once loaded, legend values are equivalent to single-value global constants. Care must be taken to ensure that all legend values are unique.

LEGENDS: { *LegendSpec* }+	Specifies the legends to include.

A **LegendSpec** can take one of the following forms:

<*LegendName*> = {#:<*Label*>, …, #:<*Label*>}	Specifies a list of number-label pairs that define a legend. Numbers must be integers.
<*LegendName*> = <*FileName*>	Loads legend from either (a) a legend file that has one row for each value-label pair, where each pair has the form "# : Label"; or (b) a GeoTiff grid with embedded legend.

4.7 Global Constants

This block defines the names and values of global constants to include in the model state-space. Single-value global constants are essentially named synonyms for values, which helps improve model readability and adaptability.

GLOBAL CONSTANTS:
{ *GlobalConstantSpec* }+

A **GlobalConstantSpec** can take one of the following forms:

<*ConstantName*> = *Expr*	Defines a simple constant and its value.
<*ConstantName*> = *Expr*, …, *Expr*	Defines a vector of constants and values.
<*ConstantName*> = <*FileName*>	Loads constant values from a file.
<*ConstantName*> = <*FileName*> SEQUENCED	Loads unindexed constant values from a file.

Input files must consist of tab-separated columns (.txt). In the first form, the first column of the file is the row index, with the number of rows based on the maximum value in the first column. Hence a file with two columns defines a one-dimensional array. Files with more than two columns define a two-dimensional array. In the second form, the indices range from 0 … num entries -1, and the number of columns in the array and file are the same.

4.8 Global Variables

This block defines the names and initial values of global variables to include in the model state-space. Single-value global variables specified here will show up in the user interface (unless followed by the “OFF” keyword). Global variables shared between landscape events *must* be specified here. Global variables in landscape events that are not specified here cannot be shared among events, will not show up in the user interface and have an undefined initial value (and should be defined instead as LOCAL variables). Global variables are used for parameters, output values, intermediate state and inter-event communication.

```
GLOBAL VARIABLES:
    { GlobalVariableSpec }+
```

A **GlobalVariableSpec** can take one of the following forms:

<*VariableName*> = *Expr*	Defines a simple global variable and its initial value.
<*VariableName*> = *Expr*, …, *Expr*	Specifies a global variable vector and values.
<*VariableName*>[*Expr*] = <*Value*>	Defines a global variable vector with a common initial value.
<*VariableName*>[*Expr*,*Expr*] = <*Value*>	Defines a global variable 2-D array with a common initial value.
<*VariableName*> = <*FileName*>	Loads values for a global variable from a file (same format as for global constants).
<*VariableName*> = <*FileName*> SEQUENCED	Loads unindexed values for a global variable from a file (same format as for global constants).

The keyword **Accumulate** can be placed to the right of a global variable definition to specify that the value is only initialized at the start of the first run in a multi-replicate simulation, and values accumulate between runs (rather than getting reset to initial state at the start of each subsequent run).

4.9 Macros

This block specifies the names of macros and the (.ce) files containing their definitions, which consist of a set of function expressions. A macro can be used in a landscape event with the same syntax as indexing a one-dimensional array global variable. However, instead of a simple value lookup, indexing a macro results in the execution of the function at that index (which may depend on any other spatial and non-spatial state in the spatio-temporal context in which the macro is evaluated). Macros are an advanced feature, and effectively allow a model to be parameterized with functions or a function to be used in multiple places in a model.

```
MACROS:
    { MacroSpec }+
```

A **MacroSpec** takes the following form:

<*MacroName*> = <*FileName.ce*>	Loads expressions in a macro from a .ce file.

4.10 Spatial Constants

This block defines spatial constants (static layers) as part of the state space. Each definition specifies a *mapping* (or linkage) to the name of the raster view to which the spatial constant refers. Spatial constants used in landscape events (as layers) must be defined in the model configuration file. This block has the following format:

```
SPATIAL CONSTANTS:
    { SpatialConstantSpec}+
```

A **SpatialConstantSpec** takes one of the following forms:

<*VariableName*> = <*ViewName*>	Defines spatial constant *VariableName*, mapped to a raster in the named view, which must exist (e.g. a loaded grid file).
<*VariableName*>	In the second form, the view name is assumed be the same as the variable name.
<*VariableName*> = [*ViewName,…,ViewName*]	Defines a static raster vector (array of rasters).

4.11 Spatial Variables

This block defines spatial variables (dynamic layers) as part of the state space. Each definition specifies a *mapping* (or linkage) to the name of the raster view to which the spatial variable refers for output, and optionally a mapping to the raster view used for initial conditions. Spatial variables used in landscape events (as layers) must be defined in the model configuration file. This block has the following format:

```
SPATIAL VARIABLES:
    { SpatialVariableSpec }+
```

A **SpatialVariableSpec** takes one of the following forms:

<*VariableName*> = <*ViewName*> *Bounds*	Defines spatial variable VariableName with output mapped to the named raster view. Initial state is assumed to be 0. The bounds specify the min and max raster value.
<*VariableName*> = <*ViewName*> *Bounds* <- #*initialValue*	Same as above, but the initial state is specified by the number (all cells set to same value)
<*VariableName*> = <*ViewName*> *Bounds* <- <*ViewName*>	Same as above, but initial state is taken from the second named view.

Raster **Bounds** takes one of the following forms:

[#*MaxValue*]	Maximum value (min. is assumed to be 0)
[#*MinValue*, #*MaxValue*]	Minimum and maximum value

If an output view does not exist, it will be created automatically.

As with spatial constants, the "= <*ViewName*>" can be omitted if the raster view is the same as the spatial variable name.

ViewName can be replaced by a list of view names "[*ViewName*, …, ViewName>]" to define a dynamic raster vector. If input views are given, the number of input and output views must match. However, in this case, no bounds can be given. To set bounds, use in conjunction with single value specifications.

As with global variables, the keyword Accumulate can be placed to the right of a spatial variable definition to specify that the value is only initialized at the start of the first run in a multi-replicate simulation.

4.12 Submodel

This section loads a sub-ordinate dynamic model configuration (.sel) file, including the state-space defined in the submodel file in the main model state-space. Any number of sub-model files can be loaded, each with its own line of the form**:**

SUBMODEL: <*SubmodelFileName.sel*>

A **SubmodelFileName** has the same form as a general dynamic configuration model (.sel) file, except that it should not include the SELES MODEL line or define the time units (i.e. it should simply be a set of one or more optional blocks).

4.13 Dynamic Raster Output

This block allows periodic (spatial time-series) output of the current state of a raster layer during a simulation. The user can either specify a directory or a file name base for a raster. If a directory is specified, the file name base will be the name of the raster. If a file name base is specified, the directory will be the directory of this file name. The actual name of a file created will have the simulation run number and the sequence number of the output appended to the file name. For example, if the file name base is “Species”, then the 3rd output of the raster during the second simulation run will use the file named “Species_2_3”. Setting up raster output is done using the following format:

OUTPUT MODEL FREQUENCY*:*

    { *RasterOutputSpec* }$^+$

A **RasterOutputSpec** takes one of the following forms:

<*RasterName*> FREQ: *Expr* FILE: <*FileNameBase*> TYPE: *OutputType**

<*RasterName*> FREQ: *Expr* DIRECTORY: <DirectoryName> TYPE: *OutputType**

These specify that the raster view named will be output at the specified frequency with a name constructed from the FileNameBase or the name of the raster suffixed with the run number and output sequence number, where **OutputType** is one of:

GEOTIFF	GeoTiff
GRASS	Uncompressed GRASS binary.
GRASS COMPRESSED	Compressed GRASS binary
GRASS ASCII	GRASS ASCII format
ARC ASCII	ARC ASCII format
ERDAS8	8-bit ERDAS format (only use for rasters with bounds less than [0,255])
ERDAS16	16-bit ERDAS format.

NOTE: The frequency expression is limited to numeric values, constants or simple global variable names.

4.14 Refresh Rate Commands

OUTPUT FREQUENCY: <#*days*>	Specifies the default display *refresh* rate for dynamic raster views. This can be changed on the user interface.

4.15 Configuration Expressions

Configuration expressions are a restricted form of expression used to define values based on arithmetic combinations of numbers and previously defined constants or variables.

#*Number*

<*Constant*>

<*Variable*>

(*Expr*)

Expr+ Expr

Expr – Expr

Expr * *Expr*	
Expr / *Expr*	
Expr ^ *Expr*	Exponentiation
Expr % *Expr*	Modulo (remainder on division)
MIN(<*ViewName*>)	Minimum bound for raster.
MAX(<*ViewName*>)	Maximum bound for raster.
CELL SIZE(<*ViewName*>)	length/width (usually in metres) of a cell.
NUM > Number (<*ViewName*>)	Number of values in raster greater than the number specified.

ROWS(<*Global Name*>)
COLS(<*Global Name*>)
	Number of rows or columns, respectively, (indexed from 0) of a global variable, global constant or legend.

TOP(<*ViewName*>)	Top georeferencing for raster (from header)
BOTTOM(<*ViewName*>)	Bottom georeferencing for raster
LEFT(<*ViewName*>)	Left georeferencing for raster
RIGHT(<*ViewName*>)	Right georeferencing for raster

Georeferencing information will depend on the raster projection.

5 SELES Landscape Events (.lse files)

Landscape events are the heart of SELES. Landscape event specifications consist of two parts:

- a set of definitions that declare the applicable portion of the spatio-temporal state space (e.g. layers and variables used and modified); and
- the relevant *properties* that specify the behaviour of the process being modelled.

Available properties are all optional, with well-defined defaults, so a model can be precisely and succinctly defined. All state changes caused by the landscape event are specified as explicit assignments made to the defined variables as an effect of one of its properties.

The general format of a landscape event is as follows:

```
LSEVENT: <event name>
{ Definitions }
{ Property }*
```

5.1 Variable Declarations

This section is used to identify the spatial (layer) and global variables and constants from the shared state-space accessed by the event, and to declare variables in the hierarchical dynamic state to be used by the event. Defined variables can be used in the expressions in the event properties or associated preliminary and consequent expressions. Each variable has a "scope", which determines where it is valid to be used, and what context the variable resides in (i.e. the state variables to which it can refer). For example, a variable with *global scope* may be referred to in any expression and takes on a single value (aspatial), whereas a variable with *spatial scope* can only be used in expressions that refer to individual cells and potentially takes on a different value at each cell.

The declaration section consists of one or more variable definitions:

```
DEFINITIONS
    { Variable Definition }+
ENDDEF
```

A *Variable Definition* has the following general form:

```
variable type: variable name [, variable name]*
```

The following types of variables can be defined in a landscape event:

(a) *Spatial layers* include spatial variables and constants from the shared state-space (and must be defined in the dynamic model file).

(b) *Global variables and constants* include non-spatial variables and constants from the shared state-space (and must be defined in the dynamic model file).

(c) *Macros* from the shared state-space (and must be defined in the dynamic model file).

(d) *Local global variables* specify non-spatial global variables that are internal (local) to the event (i.e. not defined in the dynamic model file). Initial state is undefined, and such variables should be assigned values prior to use. If an array is defined, the bounds must be explicitly defined. Note: *Local global variables* have a global scope, but are unique to the landscape event in which they are defined (i.e. if two events define a local global variable of the same name, they will refer to different variables).

(e) *Output variables* define file variables to create output tables during a simulation using a *record* expression to save a row in a tab-delimited file. Output variable declarations also specify the file name to which output records are to be stored. These are a primary means of obtaining text output from a model.

(f) *Event variables* are dynamic context variables unique to each *instance* of an event (e.g. each fire initiation event may have an *EventSize* variable).

(g) *Cluster variables* are dynamic context variables unique to each *cluster* of an event instance (e.g. each fire opening may have an *OpeningSize* variable).

(h) *Cell variables* are dynamic context variables unique to each *active cell* of an event instance (e.g. an active cell may have an *Intensity* variable).

There are three other types of constants and variables that do not appear in the definitions section:

(i) *System constants* are system-defined constants, including:
   - *NUMROWS: the number of rows in the scenario*
   - *NUMCOLS: the number of columns in the scenario*
   - *NUMCELLS: the number of cells in the scenario (i.e. NUMROWS * NUMCOLS)*

(j) *System variables* are system-defined variables, including:
   - *Time*: the current simulation time
   - *EndTime*: the end time of the simulation
   - *Run*: the current run number
   - *Location*: current spatial location index in a spatial context. The row and column of the location can be obtained using the functions ROW(Location) and COL(Location).
   - *Index*: the current index value in an OVER INDEX expression.
   - *EventId:* a unique identifier assigned to each instance of a given landscape event

(k) *Temporary local variables*: Any other undefined variable is assumed to be a *temporary local* variable. Temporary local variables are created automatically on their first assignment in a context, and have a scope that is generally limited to the immediately subsequent assignments or main expression of a property.

A *VariableDefinition* has the following syntax, where additional variable names can be included on one line, separated by commas (except for output variables and local constants).

To include variables from the shared state-space (must be defined in the .sel file):

LAYER: <*Layer Name*>	include spatial variables/constants
LAYER: <*Layer Name*>[#*NumSubLayers*]	same as above, but for raster vectors
GLOBAL CONSTANT: <*Constant* >	include global constants
GLOBAL CONSTANT: <*Constant*>[]	same as above, but for arrays
GLOBAL VARIABLE: <*Variable* >	include global variables
GLOBAL VARIABLE: < *Variable* >[]	same as above, but for arrays
MACRO: <*Macro*>[]	include macros (arrays of functions)

To define variables internal to the landscape event:

OUTPUT VARIABLE: <*Variable* > = <*FileName*>	define an output variable and file name
CONSTANT: <*Constant* > = #*Value*	define a local constant
LOCAL: <*Variable* >	define local global variables
LOCAL: <*Variable*>[#*Dim*]	same as above, but for arrays, where #Dim is either one number (1-dimension arrays) or a comma-separated pair of numbers (2-dimension arrays)
EVENT VARIABLE: <*Variable*>	define event instance variables
CLUSTER VARIABLE: <*Variable* >	define cluster variables
CELL VARIABLE: <*Variable*>	define active cell variables

Advanced: Sets, lists, trees and graphs are dynamic linked structures. To include dynamic linked structure variables from the shared state-space (must be defined in the .sel file as regular global variables, but must only be used with expressions for the defined type):

```
GLOBAL SET{#n} VARIABLE: <Variable>                // include set variables
GLOBAL LIST{#n} VARIABLE: <Variable>               // include list variables
GLOBAL TREE{#n} VARIABLE: <Variable>               // include tree variables
GLOBAL GRAPH{#n1, #n2} VARIABLE: <Variable>        // include graph variables
```

For these types of variables, an *item* is defined in SELES as a vector of values (i.e. a one-dimensional array) with dimension #n.

A set is an unordered collection of items, where common operations include union, intersection and membership testing. A list is an ordered (sequenced) collection of items, where common operations include adding or removing a new head (front) or tail (end) element and cycling over all elements. A tree is a hierarchical collection of items, where common operations include adding child nodes and traversing the tree (e.g. in depth first order). Graphs are two inter-related sets of items: a set of nodes (with dimension #n1) and a set of links (with dimension #n2), where each link is associated with a start and end node.

See the set, list, tree and graph expressions for how to use these variables. In the dynamic model file, these variables are defined the same as other variables (i.e. as if they are single values rather than collections). In general, SELES manages all the dynamic allocation/deallocation of memory required, and cleans up the variables upon simulation termination.

In addition to single-value global variables, these types of dynamic linked structure variables:

- can be specified as arrays (e.g. to create an array of sets).
- can be applied to layer variables (e.g. a raster in which each cell represents a list).

5.2 Landscape Event Properties

Landscape event behaviour is constructed by defining the applicable properties, each of which has the same general structure. The modeller need only specify those properties relevant to a given landscape event, since properties have well-defined defaults when not specified explicitly. Each landscape event property consists of the following parts:

Preliminary expressions
Main expression
Consequent expressions

Only the *main expression* is required when using a property, and it is a function evaluated in the operating context that defines the *value* of the property (see Table 2 in Section 4.1 of Part 1). The *preliminary* and *consequent expressions* allow state changes to be associated with a property in the operating and consequent contexts, respectively (see Section 4.4 of Part 1). Preliminary expressions are evaluated immediately before the main expression in the operating context. One effect of evaluating the main expression is to modify the context.

The consequent expressions are evaluated after the main expression in the resulting consequent contexts, which may be in at a different time and/or in one or more different locations. Each property has a particular behaviour with respect to if and when consequent expressions get evaluated:

InitialState: for each event instance created at simulation start-up
ReturnTime: when an event initiates
EventLocation: for each cell in the defined region
NumClusters and ProbInit: for each cell in which the event actually initiates
Transitions: if a transition occurs (i.e. if the main expression returns TRUE (1))
SpreadTime: after a cell has finished spreading
SpreadLocation: for each cell in the defined spread region
NumRecipients and SpreadProb: for each cell that is actually spread to
EndCluster: n/a (only preliminary contexts used)
EndEvent: n/a (only preliminary contexts used)

Using the example Fire event from Table 1 (in Section 2 of Part 1), a *FireSize* cluster variable may be declared as a cluster variable to control the number of cells to which a particular Fire cluster will spread. The consequent assignment *FireSize* = *normal*(μ=200,σ=20) in the *Number of Clusters* property will assign a random *FireSize* to each new cluster. In the *Transitions* property, the main expression *FireSize* > 0 will only allow a Fire cluster to spread while *FireSize* is positive. The consequent assignment *FireSize* = *FireSize* – 1 in the *Transitions* property will record the fact that a cell has burned. Thus, Fires will spread until they burn *FireSize* cells and then extinguish. The contexts, value type of the main expression, and default for each property are defined in Table 4 (see Section 4.4 of Part 1).

A *property* has the following general syntax:

```
{ Property Name }
      { Expression }*                             // preliminary expressions
      { Property Name } = { Expression }          // main expression
      { Expression }*                             // consequent expressions
{ PropertyEndTag }
```

If there are no state-change expressions associated with a property, then the enclosing name and end tag are optional. Properties can be specified in any order following the *Definitions* section. A property should not be specified more than once.

The keywords for property names and their end tags are as follows:

INITIALSTATE	ENDIS
RETURNTIME	ENDRT
EVENTLOCATION	ENDEL
NUMCLUSTERS	ENDNC
PROBINIT	ENDPI
TRANSITIONS	ENDTR
SPREADTIME	ENDST
SPREADLOCATION	ENDSL
NUMRECIPIENTS	ENDNR
SPREADPROB	ENDSP
ENDCLUSTER	ENDEC
ENDEVENT	ENDEE

The following are examples for part of the Fire event from Table 1 (in Section 2 of Part 1):

```
RETURNTIME = NEGEXP(5)           // 5 years, on average, distributed as by negative exponential

PROBINIT = IF Forested EQ FALSE THEN 0 ELSE ForestAge/10000        // prob. 0.01 at age 100

NUMCLUSTERS
      NUMCLUSTERS = 1            // one fire per year
      FireSize = NORMAL(200,20)  // distributed normally with mean 200 and std. dev. 20
ENDNC

TRANSITIONS
      TRANSITIONS = FireSize > 0 // Occur (TRUE) only if FireSize variable is > 0
      FireSize  = FireSize – 1   // Decrement FireSize variable
      ForestAge = 0              // Set ForestAge to 0 (i.e. burn cell)
ENDTR
```

Notes on main expressions for properties:

- *InitialState*: Should be an integer ≥ 1 (to specify the number of event instances to create on simulation start-up).

- *ReturnTime*: Should be a real value ≥ 0. A value of 0 at simulation start up means schedule for processing at time 0 (right away). A value of 0 later in a simulation means terminate the event (i.e. a time increment of 0 is not allowed as time would never advance).

- *EventLocation* and *SpreadLocation*: Must be *region* functions, which specify sets of cells. The main expression of all other properties cannot be region functions.

- *NumClusters* and *NumRecipients*: Should be an integer ≥ 0 (to specify the number of clusters to initiate or cells to which to spread, respectively).

- *ProbInit* and *SpreadProb*: Should be a real value ≥ 0. A value of 0 always excludes a cell from initiation or spread. Values > 0 specify the absolute or relative probability of initiation in, or spread to, the cell. These values will be absolute probabilities if *NumClusters* or *NumRecipients* are undefined, respectively, and relative probabilities (weights) otherwise. Absolute probabilities should be between 0 and 1. Relative probabilities can be any number ≥ 0 (as they are interpreted relative to the values in other cells).

- *Transitions*: Should be a logical value. An active cell will continue only if the value is TRUE.

- *SpreadTime*: If the value is ≥ 0, then spreading is processed normally via the event queue. That is, spreading cells will be scheduled on the event queue, and processed at the appropriate time (which may be interspersed with spreading of other clusters in this event instance, or with the behaviour of other landscape events).

  If the value of *SpreadTime* is negative, then spreading will be processed in *immediate mode* in which a cluster spreads entirely without being placed on the event queue. That is, once an event initiates in a cell, it will spread *before the event attempts to initiate another cluster*. This is useful for modeling processing patches sequentially rather than in parallel. In this mode, the absolute value of *SpreadTime* is used to prioritize ordering of cells spreading within the cluster in the same way as the main event queue (using a temporary queue for the cluster, which continues until it is empty). Usually *SpreadTime* is always either ≥ 0 (normal spread) or < 0 (immediate mode), but mixtures are possible (i.e. the decision of whether spreading for an active cell is placed on the normal event queue or processed immediately depends on its unique *SpreadTime*).

Note: The consequent expressions of *SpreadTime* are processed *after* the scheduled active cell has finished spreading (and just prior to termination), not when the active cell is taken off the queue. This is designed to allow the consequent expressions to be used for any processing prior to an active cell terminating (e.g. to test if any spread actually happened). If needed, the preliminary expressions of *SpreadLocation* and *NumRecipients* are processed as soon as the active cell is taken off the queue.

5.3 Specifying State-Change

Changes to the state variables are specified as assignments in the preliminary and consequent expressions of a property.

The general form of a state-change is an assignment of the form:

<Variable> = { Expr }

Expression can also involve more procedural elements (e.g. iteration, conditions, etc.), within which state-changed assignments are expressed.

5.4 Expressions

Property expressions (main, preliminary, consequent) form the core of the SELES modelling language. Expressions can range from simple constants to complex functions, can be deterministic or stochastic, continuous or discrete, computed at a single cell or over a neighbourhood of cells, etc. Most expressions are functions, but some are procedural.

Here we focus on a high-level description of available expressions. See Appendix 1 for syntax details on all available expressions.

The following describes some of the expression types available:

(a) *Constants*: basic constant values including numbers or named constants.

(b) *Variables*: named variables available in the context.

Global variable and constant arrays are indexed to lookup the value at given row (and possibly column) of the array. These indices may be simply or more complex function expressions. Macros are used like one-dimensional arrays, with a single index, but instead of a value lookup, the value is the result of evaluating the macro function (which it itself an expression).

In a spatial context, the value of a spatial layer (variable or constant) is the value at the current location.

Spatial variable and constant names may optionally be preceded by the SOURCE keyword in spatial contexts associated with an obvious different location that led to this context. This includes the *SpreadProb* property, as well as the *NumRecipients* and *SpreadLocation* consequent expressions, for which use of the SOURCE keyword will refer to spatial values at the source spreading cell rather than the current cell (i.e., the value at the spreading cell rather than the recipient cell, which is the current location in those contexts). The SOURCE keyword can also be used in *region* expressions in spatial contexts to refer to values in the cell at which the region expression is evaluated.

(c) *Arithmetic functions*: basic arithmetic (including exponents, logarithms, modulo arithmetic).

(d) *Boolean functions*: expressions that evaluate to true or false (including equality testing, relations such as "<", and logical tests such as AND, OR, NOT).

(e) *Conditional expressions*: There are two forms of *"if expressions"* and one form of "while loops"*:*

- Functional if-expressions: defined as `if (expression1) then (result1) else (result2)`, using an assignment. For example:

  ```
  x = if (Remainder > 0) THEN 1 ELSE 0 // assign x either 1 or 0
  ```

  This form of if-expression can be equivalently specified in a more compact functional form defined as `(expression1 ? result1 : result2)`. For example:

  ```
  x = (Remainder > 0 ? 1 : 0) // assign x either 1 or 0
  ```

- Procedural if-expressions: defined without an assignment, with sections of commands for each alternative. For example:

  ```
  IF Remainder > 0
    x = 0
  ELSE
   x = 1
  END
  ```

  This form may be "chained" with `ELSE IF (expression1)` sections between the `IF` and `ELSE` sections. Also, the `ELSE` section is optional.

- While-loops: defined without an assignment, with sections of commands for each iteration. For example:

  ```
  WHILE Remainder  > 0
    Remainder = Remainder - 1
  END
  ```

(f) *Iteration*: Similar to *"for-loops"* in other languages. These are procedural, with sections of commands for each iteration. For example:

```
x = 0
OVER INDEX(1,10)
 x = x + 1
END
```

(g) *Spatial functions*: Functions based on spatial locations (including distance, direction).

(h) *Output expressions*: Expressions to output records to files or to display on the screen.

(i) *Probability Distributions*: draw values from probability distributions such as uniform, normal, Weibull, negative exponential, etc. For example, an expression that specifies a normal distribution with a mean of 20.0 and a standard deviation of 5.0 will return a value drawn from this distribution each time it is invoked. The value such expression return is not deterministic, and may be different each time it is evaluated. This is one way in which stochasticity can be introduced into landscape events.

(j) *Probability Density Functions*: return the probability for a given value from probability distributions.

(k) *Trigonometric functions*: basic trigonometric functions based on angles in degrees (including SIN, COS, TAN).

(l) *Class and Discrete expressions*: define separate values or sub-functions (*switching*) for different values of a categorical variable (e.g. different values for each tree species).

(m) *Bit-vector functions*: these tree a variable as an array of bits, and allow setting and testing of individual bit positions.

(n) *Control expressions*: these provide some additional simulation control, such as setting the random number seed or pausing on certain conditions.

(o) *Composite functions*: provide a means of combining multiple sub-functions, such as assessing the conjunction (“AND”) of multiple conditions.

(p) *Region functions:* define spatial regions. These functions are unlike other functions in that they don't return a single value, but rather a set of spatial locations (cells). Their application is limited:

- to define the main expression of the *Event Location* and *Spread Location* properties, and
- to use with the “OVER” keyword to apply a set of sub-expressions over the cells in the region.

(q) *Matrix expressions:* define operations on one-dimensional and two-dimensional arrays (arrays).

(r) *Set, list, tree and graph functions:* define operations on dynamic linked structures (advanced).

5.5 Property options to modify initiation/spread algorithms

There are several optional flags that can be used to modify the default algorithms used for initiation (via *ProbInit* and *NumClusters*) and spread (via *SpreadProb* and *NumRecipients*). This is usually done by flag keywords to the right of the property section name. A range of different initiation/spread algorithms can be invoked by the specification and options used for these properties.

- **Initiation/spread with replacement**
  By default, the cells selected for initiation or spread are unique (i.e. selection is *without replacement*). Selection *with replacement* is possible by adding the keywords "WITH REPLACEMENT" following the *NumClusters* or *NumRecipients* property section names. In this case a given cell may be selected more than once for initiation or spread. For example:

```
NUMCLUSTERS WITH REPLACEMENT
    NUMCLUSTERS = 100                   // A cell may have > 1 clusters initiated in it
ENDNC
```

- **Initiation/spread order**
  By default, the order in which cells are attempted for initiation and spread is simply bottom-left to top-right of the cells in the region. This is efficient and sufficient for many landscape events. The following options allow alternative algorithms to be applied:

  - **Randomized initiation/spread order**

  The "RANDOM" flag used with *ProbInit* and *SpreadProb* causes cells to be processed in random order for initiation or spread. That is, the cells in the *EventLocation* are randomly shuffled after evaluating *ProbInit* or *SpreadProb*, but before initiation. For example:

```
PROBINIT RANDOM
    PROBINIT = 0.1
ENDPI
```

  - **Sorted (ordered) initiation/spread order**

  The "ORDERED" flag used with *ProbInit* and *SpreadProb* causes cells to be processed for initiation or spread in order of decreasing values of *ProbInit* or *SpreadProb*. That is, the cells in the *EventLocation* are sorted in decreasing order after evaluating *ProbInit* or *SpreadProb*, but before initiation. For example:

```
PROBINIT ORDERED                        // Process in decreasing order of stand age
    PROBINIT = StandAge
ENDPI
```

In general, the RANDOM and ORDERED options simply change the order in which cells in the *EventLocation* are tested for initiation (the default is row by row from the bottom left to top right). However, in conjunction with other behaviour, these can provide more control over the order in which cells are initiated or spread.

Note that sorting large probability surfaces can take a fair amount of time.

- **Probability re-computation**

  By default, probability surfaces created by *ProbInit* and *SpreadProb* are computed at the start of initiation and spread, and are not affected by changes to the state as initiation/spread unfolds. A model can force these to be re-computed during initiation/spread (e.g. to update probabilities changes made up to that point). This requires two steps:
  - The "RECOMPUTE" flag needs to be set for *ProbInit* or *SpreadProb* to inform SELES that the model may force re-computation of the probability surfaces.
  - The "RECOMPUTE" expression can be included to actually force the re-computation (usually in *Transitions* or consequence of *ProbInit)*.

  The effect depends on whether *NumClusters/NumRecipients* is specified:
  - If *NumClusters/ NumRecipients* is not specified (unbounded): the probabilities will be recomputed for the remaining cells in the *EventLocation/SpreadLocation* that have not yet been attempted for initiation or spread.
  - If *NumClusters/NumRecipients* is specified (bounded): the probabilities will be recomputed for all cells in the *EventLocation/ SpreadLocation*, and will function as though the WITH REPLACEMENT flag is set (or with the ORDERED flag set, will re-start processing of cells in the sort order). If cells that have already initiated prior to the RECOMPUTE expression should not be re-initiated, then ensure appropriate conditions are specified in *ProbInit* or *Transitions*.

  This option is usually used in conjunction with the ORDERED flag to force a resorting of the cells remaining to be attempted for initiation or spread. For example:

```
PROBINIT ORDERED RECOMPUTE
    PROBINIT = FuelLoad
ENDPI

TRANSITIONS
    TRANSITIONS = TRUE

    IF …                                // sufficient change to FuelLoad layer
        RECOMPUTE
    END
ENDTR
```

  Note that frequent re-computation of the probability surfaces can take a lot of time.

Table 5 summarizes the variations of initiation/spread that can be set up with options for *ProbInit* and *SpreadProb*. Number control refers to whether the number of cells is unbounded (*NumClusters* / *NumRecipients* not specified or < 0) or bounded (*NumClusters* / *NumRecipients* ≥ 0).

Table 5 Initiation/spread algorithms applied with different specifications and options

Initiation (*ProbInit* or *SpreadProb*)	**Number Control (*NumClusters* or *NumRecipients*)**	***ProbInit* / *SpreadProb* Options**	**Initiation/Spread Algorithm Description**
Not specified	Unbounded	-	Initiate everywhere (in event location) in default order
Defined	Unbounded	-	Initiate independently in cells (selected using absolute probabilities)
Defined	Unbounded	RANDOM	Initiate independently in cells, processed in random order
Defined	Unbounded	ORDERED	Initiate independently in cells, but process cells in decreasing order of probability
Defined	Unbounded	RECOMPUTE	Initiate independently in cells, but the probability surface for remaining cells may be recomputed dynamically (via a call to the RECOMPUTE expression)
Defined	Unbounded	RANDOM, RECOMPUTE	As for RANDOM and RECOMPUTE combined
Defined	Unbounded	ORDERED, RECOMPUTE	As for ORDERED and RECOMPUTE combined
Not specified	Bounded	-	Initiate in specified number cells (equal likelihood)
Defined	Bounded	-	Initiate in specified number cells (selected using relative probabilities)
Defined	Bounded	ORDERED	Initiate in specified number of cells, chosen in order of *ProbInit*
Defined	Bounded	RECOMPUTE	Initiate in specified number of cells, but the probability surface for all cells may be recomputed dynamically (via a call to the RECOMPUTE expression)
Defined	Bounded	ORDERED, RECOMPUTE	As for ORDERED and RECOMPUTE combined

Note: RANDOM order is not applicable when the number of clusters/spread cell is bounded by *NumClusters* or *NumRecipients*, since in that case the default is already a random selection process (possibly weighted with the relative probabilities of *ProbInit* or *SpreadProb*).

- **Initiation/spread while a condition is met**

  If *NumClusters* or *NumRecipients* are not defined (either because they are not specified or the value of the main expression is < 0), then the number of clusters initiated, or cells to which to spread, is *unbounded* (not explicitly limited, with the expected number equal to the sum of the probabilities).

  By default, if *NumClusters* or *NumRecipients* are defined, then the number of clusters initiated, or cells to which to spread, is explicitly *bounded* (limited to the value of the main expression).

  The "WHILE" keyword can be used in the main expression of *NumClusters* and *NumRecipients* to specify that the number to clusters to initiate, or cells to which to spread, as bounded by a conditional expression rather a fixed number. That is, rather than evaluating to a number of cells to select (e.g. NUMCLUSTERS = 5), the "WHILE" keyword specifies that the expression is a condition, and that cells are to be chosen for initiation of spread *until that condition is false* (or until all cells in *EventLocation* have been initiated). For example:

  ```
  NUMCLUSTERS
      NUMCLUSTERS = WHILE numLeft > 0
  ENDNC
  ```

  This will continue selecting cells for initiation until the variable *numLeft* is ≤ 0 or every cell has been selected.

Table 6 summarizes the variations of initiation/spread that can be set up using a conditional initiation/spread. The "WHILE" option can also be combined with the "RECOMPUTE" option.

Table 6 Initiation/spread algorithms applied with different specifications and options when using the conditional "WHILE" option with *NumClusters* or *NumRecipients*.

Initiation	**Number Control**	***ProbInit* / *SpreadProb* Options**	**Algorithm Description**
Not specified	Conditional	-	Initiate in randomly selected cells until WHILE expression evaluates to false
Defined	Conditional	-	Initiate in cells selected stochastically based on probabilities until WHILE expression evaluates to false
Defined	Conditional	ORDERED	Initiate in cells according to decreasing value of probabilities until WHILE expression evaluates to false

5.6 Property options to modify *SpreadTime*

- ***SpreadTime* as absolute, not relative**

By default, *SpreadTime* is interpreted as the elapsed time during spread (even conceptually in immediate mode, although no time is actually consumed). That is, the time step is "relative" to the time at which the source cell is processed. Setting the option "ABSOLUTE" for *SpreadTime* specifies that the value of the main expression refers to an absolute time, not an incremental time step. In this case, time is no longer necessarily a forward progression, as events may be scheduled to be processed in the "past". This option is primarily intended for non-temporal models that use the event queue as a priority queue rather than a time series of events.

- **Immediate spread mode**

Normally, *SpreadTime* should be a value $\geq$ 0, representing the incremental time for scheduling spread on the event queue. In this case, when a new cluster is initiated, and passes *Transitions*, spreading from the initiating cell is placed on the event queue while other clusters are initiated. This results in *parallel*, or interleaved, spreading of all the clusters initiated by an event instance (with exact sequencing depending on *SpreadTime*).

Specifying negative values for the main expression of *SpreadTime* causes clusters to be spread in *immediate mode*, whereby spreading of a cluster is processed prior to any further initiations of other clusters, and without consuming simulation time. That is, immediate mode can be used to process clusters *sequentially* rather than simultaneously. Sequential processing of clusters has a range of uses such as patch identification, and placing harvesting blocks one at a time until a target has been reached.

The absolute value of *SpreadTime* in immediate mode has the same interpretation as in regular mode in terms of the ordering of cell processing within the cluster. That is, lower absolute values will be processed before higher values at the specified rate (e.g. if a portion of the cluster has a *SpreadTime* of –2 and another has a *SpreadTime* of –10 will the former will process 5 spreading steps for every one of the latter). This is done using a temporary priority queue.

Spreading in immediate mode will continue until the temporary priority queue is empty (i.e. the cluster terminates), at which point the next cluster, if any, will be initiated.

This option must be used carefully. Since time is not consumed, immediate mode increases the risk of an infinite loop. Further, the user interface must wait until immediate mode completes because the user interface uses the priority queue to interact with the simulation engine. Hence, if an infinite loop is entered in immediate, the simulation cannot be stopped using the interface (unlike with normal parallel spreading). Hence, immediate model should only be used when necessary, and after appropriate testing ensures the model will operate as designed.

6 SELES Macro Files (.ce files)

A SELES macro essentially defines a named array of function expressions. Macros are defined similarly to landscape events, and loaded in a dynamic model configuration (.sel) file. Macros are declared in landscape events as MACRO variables, but are otherwise used similarly to one-dimensional global variables, with the difference that indexing into a macro returns the value from evaluating the function at that index.

Macros can be useful for operations that need to be performed identically in several parts of a model, or for allowing a model to have expressions as parameters.

6.1 Specifying macro files

Macro file specifications consist of two parts:

- a set of definitions that declare the applicable portion of the spatio-temporal state space (e.g. layers and variables used and modified); and
- a list of one or more function expressions.

The general format of a macro definition is as follows:

```
MACRO: <Macro Name>
{ Definitions }
{ Expression }+
```

The *Definitions* part has the same format as in landscape events, and can include the state variables that are used in the context intended for the macro use, which must be available in that context.

Each expression must have the form of an assignment (see Section 6 of this Part):

```
<Variable > = Expr
```

where (i) the variable is not used, and so can have an arbitrary name (but avoid using names of known variables); (ii) the expressions must be a function that returns a value that will be returned when the macro is indexed; and (iii) the expressions are indexed starting at 0.

The following is an example:

```
MACRO: Priorities
DEFINITIONS
    LAYER: MgmtUnit
ENDDEF

p1 = MgmtUnit EQ 1 // identify management unit 1
p2 = MgmtUnit EQ 2 // identify management unit 2
```

6.2 Including macro files in a dynamic model

Macro files are declared in a dynamic model file in a *macros* section. For example, if the preceding macro example was stored in file priorities.ce, then it could be included in a .sel file as:

```
Macros:
        PriorityMacros = priorities.ce
```

The number of expressions in a macro can be identified, and set to a global constant or variable, using the ROWS() built-in function. For example:

```
Global Constants
      NumPriorityMacros = ROWS(PriorityMacros)
```

6.3 Including and using macros in landscape events

Macros can be included in the definitions section of a landscape event in a similar way to one-dimensional global variables:

```
MACRO: macroName[]
```

For example:

```
DEFINITIONS
      :
      MACRO: PriorityMacros[]
      :
ENDDEF
```

Macros are used similarly to looking up a value in a one-dimensional array (e.g. x = macroName[j]). However, indexing into a global variable array simply looks up the current value at that index. Indexing into a macro evaluates the function at that index, in the current spatio-temporal context (e.g. the result may depend on values at the location in which the lookup takes place). For example, a landscape event may use the above macro as follows:

```
PROBINIT
      PROBINIT = PriorityMacros[currMgmtUnit] EQ TRUE
ENDPI
```

In this example, the macro function indexed by the variable currMgmtUnit will be evaluated every time step in each cell in the event location (and will presumably evaluate to true in some cells and false in others).

PART 4. Appendices

Appendix 1: SELES Expression Syntax

Expression specifications are the core of the SELES modeling language used for landscape events, macros and value summary models. They provide a general structure with which simple or complex, deterministic or stochastic expressions can be constructed. Expressions are divided into the following groups (which correspond to the sub-sections of this appendix):

Constants, Variables and Macros
Arithmetic Functions
Boolean Functions
Conditional Expressions
Iteration Expressions
Spatial Functions
Output Expressions
Probability Distributions
Probability Density Functions
Trigonometric Functions
Class and Discrete Expressions
Bit-Vector Functions
Control Expressions
Composite Functions
Region Functions
Matrix Expressions
Set, List, Tree and Graph Expressions

Functional expressions return a value, which are typically assigned to a variable (e.g. *x = Expr*), whereas procedural expressions do not return a value and so cannot be assigned to a variable or used where a return value is expected. Note that an assignment is a special form of procedural expression (i.e. an assignment such as *x = Expr* is a procedure to set a variable value).

Variables referred to in expressions can be any of the variables specified in the *Definitions* section that is within the context of the expression, or a valid local variable (i.e. a temporary local variable within the context of this expression). Dynamic variables (cell, cluster, etc.) should have been previously assigned before being used.

Models often require the use of Boolean (true or false) valued variables. Some expressions return Boolean values (e.g. EQUAL or GREATER OR EQUAL). If a variable is Boolean valued, then the keywords TRUE and FALSE can be used, and are equivalent to 1 and 0, respectively.

The following subsections summarize the syntax and meaning of the available expressions and their synonyms and variants. The label "*Expr*" indicates that a sub-expression is to be substituted (generally a function).

Use parentheses to ensure the desired precedence order is used, especially when mixing types (e.g. relations and arithmetic). Standard precedence rules are applied (e.g. "x + y * z" is interpreted as "x + (y * z)"), but there are no standard precedence rules between types. For example, if "x + y > 0" and "x AND y > 0" are meant as conditions "(x + y) > 0" and "x AND (y > 0)", parentheses can ensure proper interpretation, and avoid interpretation as "x + (y > 0)" and "(x AND y) > 0", which are odd, but syntactically valid.

1.1 Constants, Variables and Macros

ZERO	returns 0.0
ONE	returns 1.0
#*Number*	returns the value of the number specified
<*Constant*>	returns the value of a named constant
<*Constant*>[*Expr*]	returns the value of a named vector constant
<*Constant*>[*Expr*, *Expr*]	returns the value of a named array constant
<*Variable*>	returns the value of an independent variable
<*Variable*>[*Expr*]	returns the value at an index in a one-dimensional array

<*Variable*>[*Expr*, *Expr*]	returns the value at an index in a two-dimensional array
<*Macro*>[*Expr*]	returns the value of expression an index in a macro

1.2 Arithmetic Functions

(*Expr*)	returns the expression value (just adds parentheses)
Expr + *Expr*	returns the sum of two expression values
Expr - *Expr*	returns the first expression value minus the second expression value
Expr * *Expr*	returns the product of two expression values
Expr / *Expr*	returns the first expression value divided by the second expression value
Expr % *Expr*	returns the first expression value modulo the second expression value – remainder on division (assumes that the expression values are integers)
| *Expr* |	returns the absolute value of the expression value
ABS(*Expr*)	same as above
ABSOLUTE(*Expr*)	same as above
Expr ^ *Expr*	returns the first expression value to the power of the second expression value; an nth root can be computed by using 1/n as the power
EXP(*Expr*)	returns the base of the natural logarithm (e) to the power of the expression value (i.e. e^*Expr*)
LOG(*Expr*)	returns the natural logarithm of the expression value. Logarithms in another base *b* can be computed as logb(x) = ln(x) / ln(b)
ROUND(*Expr*)	returns the expression value rounded to the nearest integer

FLOOR(*Expr*)	returns the largest integer smaller than the expression value
CEILING(*Expr*)	returns the smallest integer larger than the expression value
MIN(*Expr*, *Expr*)	returns the smaller of two expression values
MAX(*Expr*, *Expr*)	returns the larger of two expression values
CLAMP(*Expr*, *MinExpr*, *MaxExpr*)	returns the value X of the expression restricted to the range [Min, Max]: if X < Min then returns Min; if X > Max returns Max; otherwise returns X

The following are some short-hand forms of some basic arithmetic assignments:

<*Variable*>++	equivalent to "<*Variable*> = <*Variable*> + 1" (increment)
<*Variable*>--	equivalent to "<*Variable*> = <*Variable*> - 1" (decrement)
<*Variable*> += *Expr*	equivalent to "<*Variable*> = <*Variable*> + Expr" (add)
<*Variable*> -= *Expr*	equivalent to "<*Variable*> = <*Variable*> - Expr" (subtract)
<*Variable*> *= *Expr*	same as "<*Variable*> = <*Variable*> * Expr" (product)

1.3 Boolean Functions

Expr EQ *Expr*	returns TRUE if the expression values are equal; otherwise FALSE
Expr == *Expr*	alternate form of EQ comparison (Note: two equal signs distinguish this from an assignment)
Expr NEQ *Expr*	returns TRUE if the expression values are not equal; otherwise FALSE
Expr != *Expr*	alternate form of NOT EQUAL comparison
Expr < *Expr*	returns TRUE if the first expression value is less than the second expression value; otherwise FALSE

Expr <= *Expr*	returns TRUE if the first expression value is less than or equal to the second expression value; otherwise FALSE
Expr > *Expr*	returns TRUE if the first expression value is greater than the second expression value; otherwise FALSE
Expr >= *Expr*	returns TRUE if the first expression value is greater than or equal to second expression value; otherwise FALSE
!*Expr*	returns the negation of the Boolean expression value (i.e. returns TRUE if the expression value is FALSE; otherwise FALSE)
Expr AND *Expr*	returns TRUE if both expressions are TRUE; otherwise FALSE
Expr OR *Expr*	returns TRUE if at least one expression value is TRUE; otherwise FALSE

Note that relations can be cascaded, with the interpretation that each operator applies independently to its surrounding pair of expressions. Thus, the expression: *Expr* <= *Expr* <= *Expr* (e.g. 5 <= age <= 50) specifies a "between" relation, which is TRUE only if the value of the centre expression falls in the specified range.

1.4 Conditional Expressions

Note the difference between the first two "functional if's" (i.e. a value is returned) and the latter "procedural if's" (i.e. that don't return a value, but control flow).

IF *Expr* THEN *Expr* ELSE *Expr*	if value of the IF expression is TRUE ($\geq$1), then returns the THEN expression value, otherwise returns the ELSE expression value
(*Expr* ? *Expr* : *Expr*)	same as above, but this form can be used as an embedded sub-expression

IF *Expr*
…
END
	procedural IF statement to process sub-expressions denoted by "…" only if the expression evaluates to TRUE

IF *Expr*
…
ELSE
…
END
	procedural IF statement to process the first set of sub-expressions denoted by "…" if the expression evaluates to TRUE and the set of sub-expressions after the ELSE otherwise; note that "ELSE IF expr" forms can be strung together

WHILE *Expr* … END	procedural while loop: process sub-expressions denoted by "…" while the expression evaluates to TRUE

1.5 Iteration Expressions

Iteration is a common aspect of models. Most programming languages offer a "for-loop" of some kind. SELES uses "over index" procedural expressions for iterating. While-loops can also be used.

OVER INDEX(*Expr*, *Expr*)
    DECISION *Expr*
    …
END
	procedural iteration: evaluates a set of expressions over a linear sequence of indices. If the optional decision expression is specified, then the set of expressions will only be evaluated for indices for which the decision expression returns TRUE. The keyword DECISION can be replaced by a "?". The system variable *Index* holds the value of the index at each step in the sequence.

1.6 Spatial Functions

DISTANCE(*Expr*, *Expr*)	returns the distance between two spatial locations (normally obtained using the *Location* system variable). DISTANCE(loc1, loc2) is sqrt(sqr(loc1.x – loc2.x) + sqr(loc1.y – loc2.y))
DIRECTION(*Expr*, *Expr*)	returns the angle in degrees between two locations

AT LOCATION(*Expr*)
  Expr
    :
END
	procedure to evaluate a set of sub-expressions at a specified location

1.7 Output Expressions

```
DISPLAY
        <Label>: Expr
            ....
END
```

procedure to display on the screen a labelled set of values. If no label is given, the expression must be a simple variable, which is used for the label.

```
DISPLAY
DECISION Expr
        <Label>: Expr
            ....
END
```

same as above, but only display if the decision expression evaluates to TRUE. The keyword DECISION can be replaced by a “?”.

```
OUTPUT(<Output Variable>)
        <Label>: Expr
            ....
END
```

procedure to output to the output variable file a labelled set of values. If no label is given, the expression must be a simple variable, which used for the label.

```
OUTPUT(<Output Variable>)
DECISION Expr
        <Label>: Expr
            ....
END
```

same as above, but only output if the decision expression evaluates to TRUE. The keyword DECISION can be replaced by a “?”.

Note 1: a *label : expression* pair can be “expanded” by using the form: *Label#n1:n2 : Expr*, where n1 and n2 are integers or constant names. This will create n2 - n1 + 1 columns rather than a single column. The system variable *Index* can be used in the expression, and will take on values n1, n1+1,…,n2 for each column, respectively.

Note 2: legend labels can be output by using a “$” prefix on an expression (e.g. spp:$Species) in two ways:

(i) If the expression is a layer, then the legend associated with the layer is used.
(ii) If the expression is followed by braces containing a layer with a legend or a legend global constant (e.g. spp:$x {SpeciesLeg}). Legend global constants are defined in the *Legends* section of the .sel file and loaded in a landscape event as a global constant array).

1.8 Probability Distributions

In SELES, probability distributions are treated like functions, although they have the behaviour of selecting a value from the distribution, rather than computing the result of a deterministic function. Each time a distribution is evaluated, the value returned may be different. If a distribution is evaluated many times, and the resulting value frequencies plotted, the histogram should approximate the distribution.

A skewed normal distribution is like two normal distributions, with different standard deviations to the left and right. In a normal distribution, the centre is both the mean and the mode. In a skewed normal distribution, the central hump is only the mode.

NORMAL(*Expr*, *Expr*)	returns a value drawn from a normal distribution, where the first expression specifies the mean and the second specifies the standard deviation
SKEWED NORMAL(*Expr*, *Expr*, *Expr*)	returns a value drawn from a skewed normal distribution, where the first expression specifies the mode and the second and third specify the standard deviations to the left and right of the mode
LOG NORMAL(*Expr*, *Expr*)	returns a value drawn from a log normal distribution, where the first expression specifies the mean and the second specifies the standard deviation of the underlying normal distribution
WEIBULL(*Expr*, *Expr*)	returns a value drawn from a Weibull distribution, where the first expression specifies beta and the second specifies alpha (Note: terminology varies between sources. The version here is equivalent to that in Excel, with reversal of parameters).
NEGEXP(*Expr*)	returns a value drawn from a negative exponential distribution, where the expression specifies the mean
POISSION(*Expr*)	returns a value drawn from a Poisson distribution
UNIFORM(*Expr*, *Expr*)	returns a value drawn from a uniform distribution, where the first expression determines the minimum value and the second determines the maximum

CLASSIFIED_DIST
    #Number: *Expr*
    :
END
	returns a value drawn from a discrete distribution, where each expression determines the relative probability of its associated class being drawn

CLASSIFIED_DIST(*#Number+*)	same as above, but in this form, the classes are assumed to start at zero, and only constant probabilities can be given

CLASSIFIED_DIST *Expr* : END	same as above, but in this form, the classes are assumed to start at zero
CLASSIFIED_DIST[<*Variable*>]	same as above, but in this form, the probabilities are in the variable, which must be a one-dimensional array variable, and so classes start at zero

1.9 Probability Density and Cumulative Density Functions

The following expressions for probability density functions (PDFs) and cumulative density functions (CDFs). The results from a probability density expression should be multiplied by a width factor to obtain a probability of value X (i.e. the estimated area under the PDF curve). Without a width factor, the assumed class size is 1. A temporal PDF makes a PDF conditional, where the variable X is assumed to be a time or other increasing value. Essentially, a PDF will give the probability of value X being selected, while a TEMPORAL PDF will give the probability of X being selected given that if hasn't been selected yet (i.e. given that that actual value of this variable will be $\geq$ X).

NORMAL PDF(*Expr*, *Expr*, *Expr*)	returns a normal probability density, where the first expression is the value, the second is the mean and the third is the standard deviation
NORMAL CDF(*Expr*, *Expr*, *Expr*)	returns a normal cumulative density, where the first expression is the value, the second is the mean and the third is the standard deviation
SKEWED NORMAL PDF(*Expr*, *Expr*, *Expr*, *Expr*)	returns a skewed normal probability density, where the first expression specifies the value, the second is the mode and the third and fourth are the standard deviations to the left and right of the mode
SKEWED NORMAL CDF(*Expr*, *Expr*, *Expr*, *Expr*)	returns a skewed normal cumulative density, where the first expression specifies the value, the second is the mode, and the third and fourth are the left and right standard deviations

LOG NORMAL PDF(*Expr*, *Expr*, *Expr*)	returns a log normal probability density, where the first expression specifies the value, the second is the mean and the third is the standard deviation
LOG NORMAL CDF(*Expr*, *Expr*, *Expr*)	returns a log normal cumulative density, where the first expression specifies the value, the second is the mean and the third is the standard deviation
NORMAL TEMPORAL PDF(*Expr*, *Expr*, *Expr*)	returns a temporal normal probability density
SKEWED NORMAL TEMPORAL PDF(*Expr*, *Expr*, *Expr*, *Expr*)	returns a temporal skewed normal probability density
LOG NORMAL TEMPORAL PDF(*Expr*, *Expr*, *Expr*)	returns a temporal log normal probability density
UNIFORM TEMPORAL PDF(*Expr*, *Expr*, *Expr*)	returns a temporal uniform probability density

1.10 Trigonometric Functions

Angles are assumed to be in degrees (not radians), and in compass directions (i.e. increasing clockwise from north, in contrast to geometry, which increases counter-clockwise from east).

SIN(*Expr*)	returns sin(theta), where the expression value is interpreted as an angle theta in degrees
COS(*Expr*)	returns cos(theta), where the expression value is interpreted as an angle theta in degrees
TAN(*Expr*)	returns tan(theta), where the expression value is interpreted as an angle theta in degrees
ARCSIN(*Expr*)	returns the inverse sine
ARCCOS(*Expr*)	returns the inverse cos
ARCTAN(*Expr*)	returns the inverse tan
ARCTAN(*Expr*, *Expr*)	returns the inverse tan where first expression is the x offset and second is the y offset; providing two expressions gives more information about the quadrant

1.11 Class and Discrete Expressions

Classified (or categorical) functions use the variable as an index into the list of numbers or expressions that follow. The variable must be non-negative. If the form "Number: Expr" is used, then the number refers to the value of the indexing variable. The other forms do not specify the class value, and so the classes are implicitly sequential from 0.

CLASSIFY(<*Variable*>)
    #Number: *Expr*
    :
END
	if the value of the variable is equal to one of the classes listed (left side of colon), then returns the result of the associated expression; otherwise returns 0

CLASSIFY(<*Variable*>)
    Expr
    :
END
	returns the result of the kth expression in the list, where k = <Variable> – minimum value of the variable (this may be non-0 only for layers)

CLASSIFY(<*Variable*>): (#*Number*+)	returns the kth number in the list, where k = <Variable> – minimum value of the variable (this may be non-0 only for layers)

Switch expressions are similar to classified functions except that they use the categorical value to select a set of sub-expressions to apply, rather than return a value. Switch expressions are also like a set of procedural IF expressions (one test for each class), but are more efficient because the expressions for a given class are reached in one indexing step rather than via a series of tests. The CLASS keyword is used to separate the expressions associated with each category.

SWITCH(<*Variable*>)
    CLASS #*Number*:
    ...
    CLASS #*Number*:
    ...
    :
END
	procedure: if the value of the variable is equal to one of the classes listed (identified by CLASS keyword), then evaluate the associated sub-expressions denoted by "..."

Interpolation tables are similar to classified functions except that the variable is assumed to be continuous instead of a discrete class. An interpolated value between classes is returned. Lookup tables are also similar, except that both the variable and classes are assumed to be continuous values. An interpolated value between classes is also returned. Both of these have been rarely used.

INTERPOLATE(<*Variable*>)
    #*Number*: *Expr*
    :
END
	returns an interpolated value where the value of the variable is placed between two classes (left side of colon) or at either end of the list

INTERPOLATE(<*Variable*>): (#*Number*+)	same as above, except that the classes are implicitly 0, …, k-1

LOOKUP(<*Variable*>)
#*Number*: *Expr*
:
END
	returns the interpolated value where the value of the variable is placed between two classes (left side of colon) or at either end of the table, and classes can be any real values

1.12 Bit-Vector Functions

Bit-vector functions treat an integer value as a sequence of bits, each of which can be independently set and accessed. This can allow use of a single layer to hold lots of Boolean information, since each cell has up to 8, 16, 32 or 64 bits. For example, bits 0, 1, 2 and 3 can be used to store four Boolean states (a, b, c and d), and the sequence 0110 represents that states a and d are false and states b and c are true. *PositionList* is a semi-colon separated sequence of integers or ranges (lower-upper) indicating positions in the integer (0-63). For example, the *PositionList* 1;4-6;27 indicates bits at positions 1, 4, 5, 6 and 27.

SETAT(*Expr*, *PositionList*, #*Type*)	returns the value of the first expression, with the positions in *PositionList* set to: FALSE (0) if Type = 0, TRUE (1) if Type = 1, and flip the bits if Type = 2
SETAT(*Expr*, *PositionList*, #*Type*, #*Prob*)	same as above, except each position is set with probability #*Prob*
SELECTAT(*Expr*, *PositionList*)	returns FALSE (0) if none of the bits in *PositionList* are set in the expression value and TRUE (1) otherwise
MAX POSITION(*Expr*)	returns the maximum bit position that is set to 1 in the expression value, or –1 if no bits are set
MIN POSITION(*Expr*)	returns the minimum bit position that is set to 1 in the expression value, or –1 if no bits are set
BITWISE OR(*Expr*, *Expr*)	returns the bitwise OR of two values
BITWISE AND(*Expr*, *Expr*)	returns the bitwise AND of two values
BITWISE XOR(*Expr*, *Expr*)	returns the bitwise exclusive-OR of two values
BITWISE NOT *Expr*	returns the bitwise NOT of a value
SHIFT LEFT(*Expr*, *Expr*)	returns the value of first expression shifted to the left by the number of bits specified by second expression

SHIFT RIGHT(*Expr*, *Expr*)	returns the value of first expression shifted to the right by the number of bits specified by second expression

1.13 Control Expressions

PAUSE	halts the simulation and displays a dialog box to which the user must respond to continue
PAUSE IF *Expr*	if the expression evaluates to TRUE, halts the simulation and displays a dialog box to which the user must respond to continue
PAUSE(#*Value*)	pauses the simulation for the specified number of milliseconds and then continues
DEBUG	update the simulation probe, if it is loaded and the event is selected; used to debug/verify event behaviour
SEED()	returns the random number seed
SEED(*Expr*)	sets the random number seed, and returns the value of the next seed
RECOMPUTE	causes probability surface to be recomputed before next selection; only valid in during initiation or spread when the RECOMPUTE flag has been set for *ProbInit* or *SpreadProb*

1.14 Composite Functions

Compound expressions all have the same form with a type name followed by a list of sub-expressions. Composite expressions returned a combined result of the sub-expressions, and are sometimes used to write expressions more clearly. The general format is as follows:

```
<CompositeExpressionName>
    Expr
      :
END
```

The following composite expressions are supported:

SUM	returns the sum
PRODUCT	returns the product
DIVIDE	returns the successive division
MEAN	returns the average
GEOMETRIC MEAN	returns the geometric mean (nth root of the product for n expressions)
MIN	returns the minimum
MAX	returns the maximum
EQUAL	returns TRUE (1) if the sub-expression values are all the same and FALSE (0) otherwise
NOT EQUAL	returns the FALSE (0) if the sub-expression values are all the same value and TRUE (1) otherwise
OR	returns TRUE (1) if at least one sub-expression evaluates to TRUE and FALSE (0) otherwise
AND	returns TRUE (1) if all sub-expressions evaluate to TRUE and FALSE (0) otherwise
LESS OR EQUAL	returns TRUE (1) if the sub-expression values are ordered according to "<=," and FALSE (0) otherwise
ORDERED	same as LESS OR EQUAL
LESS THAN	returns the TRUE (1) if the sub-expression values are ordered according to "<", and FALSE (0) otherwise
STRICT ORDERED	same as LESS THAN
GREATER OR EQUAL	returns the TRUE (1) if the sub-expression values are ordered according to ">=", and FALSE (0) otherwise
GREATER THAN	returns the TRUE (1) if the sub-expression values are ordered according to ">", and FALSE (0) otherwise

1.15 Region Functions

A region function returns a set of *locations* (indexes to the row and column of individual cells). Their use is limited to the main expression of *EventLocation* and *SpreadLocation* properties and in OVER REGION functions. In all cases:

- the DECISION expression is optional
- the keyword DECISION can be replaced by a "?"
- if there is a decision expression, then only those cells for which this expression returns TRUE will be included

REGION WHOLE MAP	defines the region consisting of the entire landscape

DECISION *Expr*

REGION RECT (*Expr*, *Expr*, *Expr*, *Expr*) DECISION *Expr*	defines a rectangular region
REGION CENTRED (*Expr*, *Expr* [, *DistanceType*][, WRAPPED]) DECISION *Expr*	defines a region centred on the current cell

DistanceType is either CARDINAL or EUCLIDEAN (the default if not specified). Cardinal distance between two cells is the minimum number of cardinal steps (up, down, left, right) to reach one cell from the other. Euclidean distance is the straight-line distance between two points. The WRAPPED flag, if present indicates that the region wraps around the sides (vertically and horizontally) of the landscape (e.g. a location x positions beyond the right-hand side of a raster will be mapped to x positions in from the left-hand side of the raster). If not present, the landscape does not wrap.

REGION LOCATION LIST(#*Number*+) DECISION *Expr*	defines a region consisting of a set of known locations, separated by commas
REGION LOCATION LIST(<*Variable*>, #*n*) DECISION *Expr*	defines a region consisting of a set of n locations stored a variable, which must be a one-dimensional array with a dimension of at least n
REGION LOCATION (*Expr*) DECISION *Expr*	defines a region consisting of a single location
REGION VECTOR (*StartLocation*, *EndLocation*) DECISION *Expr*	defines a region consisting of cells along the approximate line between the two locations defined by the expressions (the start and end locations)

Cost surface and least-cost path regions are advanced features that are best understood with an example model. These two types of regions are often used together. A cost surface region is used to produce the least-cost distance diffusing out from a source location, stopping when either a target location or a maximum cost is reached. The least-cost path region can use spatial outputs from the cost surface to “walk” down the least-cost path cells from the target location back to the source location. The *LeastCostNeighbs* and *AnchorLoc* are layers to record information to assist this process.

The following define a region surrounding the current cell with cumulative costs less than the value of the *MaxCost* expression. Stops growing when *EndLocation* is reached or all costs are greater than *MaxCost*. The second form also records gradient and anchor location layers. The *CostSurface* layer records the cumulative cost from the cost function

```
REGION COST SURFACE(EndLocation, MaxCost, <CostSurface>)
    DECISION Expr
    COST Expr

REGION COST SURFACE(EndLocation, MaxCost, <CostSurface>,<LeastCostNeighbs>,<AnchorLoc>)
    DECISION Expr
    COST Expr
```

The following defines a region with cells that approximate the least-cost path between two cells. The first generates the cost function internally. The latter two use input information from a previously-computed cost surface region. Providing both the gradient and anchor location layers provides a more accurate approximation.

```
REGION LEAST COST PATH(StartLocation, EndLocation)
    DECISION Expr
    COST Expr

REGION LEAST COST PATH(StartLocation, EndLocation, <LeastCostNeighbs>)
    DECISION Expr

REGION LEAST COST PATH(StartLocation, EndLocation, <LeastCostNeighbs>, <AnchorLoc>)
    DECISION Expr
```

Over region expressions are frequently used to visit all cells in a landscape that meet some condition (e.g. to compute the size of the forest at initiation).

```
OVER RegionFunction              procedure to apply a set of sub-expressions at all
  Expression                     spatial locations specified by a region function
          :
END
```

1.16 Matrix Expressions

These functions apply to operations on arrays, which are a form of matrix. Matrix assignment is indicated using “[=]” instead of “=”, and matrix equality testing uses “[==]” instead of “==” (or “EQ”). The dimensions of the array variables must match as required.

<Matrix Variable> [=] *Expr*	assign values to all entries of an array variable; if the expression is a matrix function, values will be assigned by index and dimensions must match

The following general functions can be used in which *one or both* of the expressions are matrix (array) variables. Keep other expressions as simple as possible (e.g. constants, variables).

Expr + *Expr*	if both are array variables: returns matrix sum; otherwise: add a value to all entries
Expr - *Expr*	if both are array variables: returns matrix subtraction; otherwise: subtract a value from all entries
Expr * *Expr*	if both are array variables: returns matrix multiplication (the number of columns of the first must match the number of rows of the second); otherwise: multiply a value with all entries

The following are unique matrix functions:

<Matrix Variable> [==] *<Matrix Variable>*	returns TRUE if the arrays have the same values in all indices (matrix equality) and FALSE otherwise
TRANSPOSE(*<Matrix Variable>*)	returns a transposed array (flips rows and columns)
<Matrix Variable>^-1	returns a matrix inverse (a matrix multiplied by its inverse results in a matrix of all 1's)
SUM(*<Matrix Variable>*)	returns the sum of all entries in an array (single value)
SUM ROWS(*<Matrix Variable>*)	returns the sum over the rows of an array (one-dimensional array with one value per column)
SUM COLS(*<Matrix Variable>*)	returns the sum over the columns of an array (one-dimensional array with one value per row)
SORT(*<Matrix Variable>*, *Expr*)	returns an array with the rows of an input array sorted by *Expr*. The system variable "*Index*" can appear in the expression and will represent row index values.
MPM_MULT(*<Matrix Variable>*, *< Matrix Variable>*)	Performs a "matrix population model" (MPM) integer-

	based multiplication. Assumes that the first matrix is two-dimensional and that the second is a vector. The result is two-dimensional, where entries above the diagonal represent "offspring", entries on the diagonal represent "survivors" and entries below the diagonal represent "succeeders".
CONTAG(<*Matrix Variable*>, < *Matrix Variable*>)	Creates a "temporal contagion" matrix. Given an array of probabilities (first parameter) for a set of states and a two-dimensional contagion array (values –1 to 1 indicating affinity), generates a matrix where each row is a prob. dist. for transitions given the row value, and the overall target dist. will be met.

1.17 General Set, List, Tree and Graph Expressions

Set, List, Tree and Graph variables are dynamical structures based on the concept of an *item*, which is a one-dimensional array of attribute values. In general, these represent linked collections of zero or more items, which are added and removed dynamically by a landscape event. A reference to an item is called a *position* (in effect a pointer to an item). In the syntax, a variable that has a position for a value is labelled a "*Pos Variable*".

Sets are unsorted collections of items, called *elements* (e.g. a set of patches, each with attributes for size and type). Lists are collections of items (called *entries*) that are linked in a linear chain with a start (*head*) and end (*tail*). Trees are collections of items (called *nodes*) that are linked hierarchically, where each node has one *parent node* (except for the top-most node, which is called the *root*) and zero or more *children nodes*. Graphs are collections of items (called *nodes*) that are explicitly joined with *links* (which are in turn a type of item that joins two nodes).

These types of variables are created in a .sel file the same as other global and spatial variables, but they must be specified in landscape events with their type and item dimensions, as documented in Section 5.1 of this Part. Variables of these types should only be used with expressions appropriate for their type.

The following are expressions available for all types of these dynamic linked variables:

REMOVE ALL(<*Linked Variable*>)	clears items in a variable; no return value
SIZE(<*Linked Variable*>)	returns the number of items (elements, entries or nodes)
IS EMPTY(<*Linked Variable*>)	returns TRUE if and only if the variable has 0 items

FIRST(<*Linked Variable*>)	returns the position of first item (element, entry or node); 0 if the variable is empty
NEXT(<*Linked Variable*>, <*Pos Variable*>)	returns the position of the item following the position provided; 0 if the position provided was the last item
PREV(<*Linked Variable*>, <*Pos Variable*>)	returns the position of the item preceding the position provided; 0 if the position provided was the first item
REMOVE(<*Linked Variable*>, <*Pos Variable*>)	removes and deletes the item at the position; no return value
GET(<*Linked Variable*>, <*Pos Variable*>)	returns the item at the position; since items are one-dimensional arrays, this function is used with a matrix assignment ("[=]")
GET(<*Linked Variable*>, <*Pos Variable*>, #*Index*)	returns the single value at a specified index for the item at the position
SET(<*Linked Variable*>, <*Pos Variable*>, <*Item Variable*>)	set the item attributes at the position; no return value
SET(<*Linked Variable*>, <*Pos Variable*>, #*Index*, <*Value*>)	set the single value at a specified index for the item at the position; no return value
FIND(<*Linked Variable*>, <*Item Variable*>, *Condition Expr*)	returns the first position of an item that satisfies the condition, or 0 if there are no matches; the item variable is temporary
FIND NEXT(<*Linked Variable*>, <*Pos Variable*>, <Item *Variable*>, *Condition Expr*)	returns the next position of an item that satisfies the condition (following the provided position), or 0 if there are no matches; the item variable is temporary
SORT(<*Linked Variable*>, <*Item Variable*>, <Item *Variable*>, *Condition Expr*)	sorts items according to the condition expression; no return value; the item variables are temporary

1.18 Set Functions

Sets are unsorted collections of items called *elements*. In addition to the general functions (used to remove elements, set and obtain element attributes, etc.), the following are functions unique to sets. Since sets are unsorted, element "uniqueness" is not defined by its position; two elements are considered to be the same if they have the same attribute values.

CONTAINS(<*Set Variable*>, <*Item Variable*>)	returns TRUE if and only if the set variable "contains" the item variable provided, where two elements (items) are identical if they have identical item values
INSERT(<*Set Variable*>, <*Item Variable*>)	adds an element (item) to a set. Only changes the set if it doesn't already contain the element; no return value
UNION(<*Set Variable*>, <*Set Variable*>)	returns the union of two set variables
INTERSECTION(<*Set Variable*>, <*Set Variable*>)	returns the intersection of two set variables
SUBTRACT(<*Set Variable*>, <*Set Variable*>)	subtracts elements in the second set from the first

1.19 List Functions

Lists are collections of items (called *entries*) that are linked in a linear chain with an explicit start (*head*) and end (*tail*). Lists can be useful when the number of entries is not known *a priori* (which poses challenges to using arrays, since appropriate dimensions are not known at model setup), such as creating a list of identified patches. In addition to the general functions (used to remove entries, traverse through the entries, set and obtain entry attributes, etc.), the following are functions unique to lists.

HEAD(<*List Variable*>)	returns the position of the first entry in a list
TAIL(<*List Variable*>)	returns the position of the last entry in a list
INSERT HEAD(<*List Variable*>, <*Item Variable*>)	adds an entry to the front of a list; no return value
INSERT TAIL(<*List Variable*>, <*Item Variable*>)	adds an entry to the end of a list; no return value
INSERT BEFORE(<*List Variable*>, #*Index*, <*Item Variable*>)	adds an entry before a given index; no return value

INSERT AFTER(<*List Variable*>, #*Index*, <*Item Variable*>)	adds an entry after a given index; no return value
INSERT AT(<*List Variable*>, #*Index*, <*Item Variable*>)	adds an entry at a given index; no return value
REMOVE HEAD(<*List Variable*>)	removes and deletes the first entry; no return value
REMOVE TAIL (<*List Variable*>)	removes and deletes the last entry; no return value
REMOVE AT INDEX(<*List Variable*>, #*Index*)	removes and deletes the entry at a given index; no return value
GET HEAD(<*List Variable*>)	returns the first entry; use with a matrix assignment
GET TAIL(<*List Variable*>)	returns the last entry; use with a matrix assignment
GET AT INDEX(<*List Variable*>, #*Index*)	returns the entry at a given index; use with a matrix assignment
POS AT INDEX(<*List Variable*>, #*Index*)	returns the position at a given index; returns 0 if there is not entry at the index.

1.20 Tree Functions

Trees are collections of items called *nodes* that are linked hierarchically. The top-most node is called the *root*. Every other node has one *parent node* and zero or more *children nodes*, which are organized like a list from *left* to *right*. In addition to the general functions (used to remove nodes, traverse through the nodes, set and obtain node attributes, etc.), the following are functions unique to trees.

ROOT(<*Tree Variable*>)	returns the position for the root node of a tree
ADD ROOT(<*Tree Variable*>, <*Item Variable*>)	set the root node of a tree; no return value
INSERT LEFT CHILD(<*Tree Variable*>, <*Pos Variable*>, <*Item Variable*>)	adds a child to a node (identified by the position) on the left of its current children; no return value
INSERT RIGHT CHILD(<*Tree Variable*>, <*Pos Variable*>, <*Item Variable*>)	adds a child to a node (identified by the position) on the right of its current children; no return value

INSERT CHILD(<*Tree Variable*>, <*Pos Variable*>, <*Item Variable*>, #*Index*)
adds a child to a node (identified by the position) at a given index (from the left); no return value

CHILDREN(<*Tree Variable*>, <*Pos Variable*>)
returns the number of children at a given node position

PARENT(<*Tree Variable*>, <*Pos Variable*>)
returns the position of the parent of a node; returns 0 for the root

LEFT CHILD(<*Tree Variable*>, <*Pos Variable*>)
returns the position of the leftmost child of a node; returns 0 if the node has no children

RIGHT CHILD(<*Tree Variable*>, <*Pos Variable*>)
returns the position of the rightmost child of a node; returns 0 if the node has no children

CHILD(<*Tree Variable*>, <*Pos Variable*>, #*Index*)
returns the position of the child at an index (from left) of a node; returns 0 if the node has < *#Index children*

NEXT SIBLING(<*Tree Variable*>, <*Pos Variable*>)
returns the position of the next sibling to right of a node; returns 0 if there is no sibling to the right

PREV SIBLING(<*Tree Variable*>, <*Pos Variable*>)
returns the position of the previous sibling to right of a node; returns 0 if there is no sibling to the left

NEXT DFS(<*Tree Variable*>, <*Pos Variable*>) returns the position of next node in a depth-first pre-order traversal of a tree; returns 0 if the position is the last in the traversal. If the position variable is NULL (0), then the root is returned.

NEXT POSTORDER DFS(<*Tree Variable*>, <*Pos Variable*>)
returns the position of next node in a depth-first post-order traversal of a tree; returns 0 if the position is the last in the traversal. If the position variable is NULL (0), then the root is returned.

GET LEFT CHILD(<*Tree Variable*>, <*Pos Variable*>)
returns entry of the leftmost child for a given node; use with a matrix assignment

GET RIGHT CHILD(<*Tree Variable*>, <*Pos Variable*>)
returns entry of the rightmost child for a given node; use with a matrix assignment

GET CHILD(<*Tree Variable*>, <*Pos Variable*>, #*Index*)
returns entry of the child with given index (from left) for a given node; use with a matrix assignment

1.21 Graph Functions

Graphs are collections of items called *nodes* that may be explicitly joined with items called *links*. The nodes and links are two sets, each with individual item attributes, and with each link additionally joining two nodes. The general functions for linked variables refer to the node set. In addition to the general functions (used to remove nodes, traverse through the nodes, set and obtain node attributes, etc.), the following are functions unique to graphs.

REMOVE ALL LINKS(<*Graph Variable*>)	clears the links of a graph variable; no return value
SIZE LINKS(<*Graph Variable*>)	returns the number of links in graph
IS EMPTY LINKS(<*Graph Variable*>)	returns TRUE if and only if the graph has 0 links
FIRST LINK(<*Graph Variable*>)	returns the position of the first link
NEXT LINK (<*Graph Variable*>, <*Pos Variable*>)	returns the position of the link following the position provided
PREV LINK (<*Graph Variable*>, <*Pos Variable*>)	returns the position of the link preceding the position provided

INSERT NODE(<*Graph Variable*>, <*Node Item Variable*>)
adds a node to a graph; no return value

REMOVE NODE(<*Graph Variable*>, <*Node Pos Variable*>) remove and deletes node item at position (equivalent to REMOVE()); no return value

CONTAINS LINK(<*Graph Variable*>, <*Link Item Variable*>)
returns TRUE if and only if the graph variable “contains” the link item provided, where two links items are considered identical if they have identical item values

INSERT LINK(<*Graph Variable*>, <*Node Pos Variable*>, <*Node Pos Variable*>, <*Link Item Variable*>)
adds a link to a graph, joining the two nodes (identified with position variables); no return value. Only changes the graph if it doesn’t already contain the link

REMOVE LINK(<*Graph Variable*>, <*Link Pos Variable*>) removes and deletes the link item at the position; no return value

GET LINK(<*Graph Variable*>, <*Pos Variable*>)
returns the link item at the position; use with a matrix assignment

SET LINK(<*Graph Variable*>, <*Pos Variable*>, <*Link Item Variable*>)
set the link attributes at the link position; no return value

SET LINK(<*Graph Variable*>, <*Pos Variable*>, #*Index*, #*Value*)
set the single value at the specified index of the link item at the link position; no return value

FIND LINK(<*Graph Variable*>, <*Link Item Variable*>, *Condition Expr*)
returns the first position of a link that satisfies the condition expression, or 0 if there are no matches; the item variable is temporary

FIND NEXT LINK(<*Graph Variable*>, <*Pos Variable*>, <*Link Item Variable*>, *Condition Expr*)
returns the next position of a link that satisfies the condition (following the provided position), or 0 if there are no matches; the item variable is temporary

SORT LINKS(<*Graph Variable*>, <*Link Item Variable*>, <*Link Item Variable*>, *Condition Expr*)
sorts the links according to the condition expression; no return value; the item variables are temporary

LINKED(<*Graph Variable*>, <*Node Pos Variable*>, <*Node Pos Variable*>, #*LinkType*)
returns TRUE if the nodes are linked. Set *LinkType* to 0 for direct links and 1 for indirect paths

Appendix 2: Common Error Messages

Only the most common error messages are listed here. There are some variations in error messages, so specific text of the message may vary.

Notes:

- %s refers to a text label (e.g. file name)
- %d refers to an integer
- %d.d refers to a floating point number

1 Errors during SELES start-up or window creation

InitializeOpenGl: HDC Invalid

InitialsizeOpenGl: RC invalid

These messages indicate a problem with OpenGL (software used for graphics display in SELES). If these problems persist, contact Gowlland.

Error reading config file

On start-up, SELES reads the seles.cfg configuration file. This message may arise if this file has become corrupted or locked by another application.

Error reading command line scenario: %s

SELES can be started on a command line, optionally with the name of a scenario file to load. This error message will arise if there was a problem loading the scenario file from a command line start-up.

2 Opening (reading) and saving (writing) raster files

2.1 Opening raster files

Could not open raster file %s for reading. Check that file exists and that you have read access.

Raster file (GeoTiff, Grass, ARC ASCII, ARC binary, Erdas) could not be opened. Ensure that the file exists at the location and name specified and is available for read access.

Could not open header file %s for writing. Check that you have write access.

Grass raster header file could not be opening. Ensure that the file exists at the location and name specified and is available for read access.

Could not open GeoTiff file %s for reading legend.

GeoTiff raster file could not be opened to read legend information. Ensure that the file exists at the location and name specified and is available for read access. If a GeoTiff file does not have legend information, the legend labels will be set to their corresponding values.

Cannot read Raster file: unrecognized format

Raster file does not have a supported format (or format is corrupted).

2.2 Saving raster files

Could not open raster file %s for writing. Check that you may have write access.

Raster file (GeoTiff, Grass, ARC ASCII, ARC binary, Erdas) could not be opened for write access. Ensure that the file is not locked by another application.

Could not open GeoTiff file %s for writing legend.

GeoTiff raster file could not be opened to write legend information. Ensure that the file is not locked by another application.

Cannot save 64-bit rasters that need more than 52 bits as GeoTiffs (%s) - not supported by GDAL

GeoTiff raster files are currently limited to 52 bits per cell value. Larger ranges not supported by GDAL.

3 Loading SELES model files

3.1 Loading and re-loading a SELES state-space (.sel) file

Error %s in model file at or near [%s][%s] in file %s

Syntax error: The message attempts to indicate the general text near the error (first bracketed text) and the specific text when the error was detected (second bracketed text). In some cases, the actual error may not be near the location of when this was identified. Resolving syntax errors is usually easier in the LSEditor.

Cannot open file: %s. Check that file exists at path specified and has read access. Paths to sub-files are relative to the loading model file

Error loading an input .txt file in a SELES model (.sel) file. File was not found.

File contains an error (possibly near end)

Sometimes the end of the .sel file is reached before expected. This indicates an error, but not the location of the error. Usually this is caused by a form of expression that was “opened” but not “closed” (e.g. if … end expressions).

Shouldn't treat a global variable (%s) as a global constant (in lsevent: %s).

A state variable defined as a global variable in the .sel file is incorrectly loaded as a global constant in the landscape event file.

Cannot convert a global constant (%s) to a global variable (in lsevent: %s).

A state variable defined as a global constant in the .sel file is incorrectly loaded as a global variable in the landscape event file.

Invalid macro array: %s

The macro state variable has loading errors.

Invalid landscape event (%s): layer %s not defined in .sel file

The layer variable in the landscape event is not defined in the state space.

Setting up initial state for variable %s: Cannot locate initial state viewname: %s

The initial state layer for a spatial variable was not found.

Scheduling output for variable %s: Cannot locate output viewname: %s

Scheduled time-series output in the .sel could not locate the specified raster view.

Warning: bounds [%ld,%ld] for dynamic layer %s must be at least as big as bounds [%ld,%ld] for its initial state layer: %s. Re-adjusting.

The minimum bound for a spatial variable must be $\leq$ the minimum for the initial state layer, and the maximum bound must be $\geq$ the maximum value for the initial state layer.

Warning: Sub or super-sampling raster for variable: %s. this may lose georeferencing or make duplicate copies.

The raster does not have the same dimensions at the model (which are usually defined from one of the input layers). The model will attempt to run, but it is preferrable to ensure that all spatial inputs are aligned with the model dimensions.

Error setting up SELES model. Cannot setup model as specified

Errors in loaded/re-loaded .sel file.

script variable not found: %s

An external script variable referred to in the .sel file was not created by the loading scenario file.

3.2 Loading landscape events

Parsing error in LSevent or expr (%s) at or near [%s][%s] (likely cause: keyword or punctuation expected)

Syntax error: The message attempts to indicate the general text near the error (first bracketed text) and the specific text when the error was detected (second bracketed text). In some cases, the actual error may not be near the location of when this was identified. Resolving syntax errors is usually easier in the LSEditor.

Cannot mix random and ordered with absolute initiation

The landscape event includes an invalid combination for ProbInit (random and ordered).

Cannot mix random and reordered with absolute initiation

The landscape event includes an invalid combination for ProbInit (random and re-ordered).

Landscape event: %s : Invalid Property: <property name>

The landscape event has an error in the specified property.

3.3 Errors in expressions (landscape events, macros, value summary model)

Invalid Expression Variable: %s. Layer or location variables not permitted here.

A spatial layer or built-in location variable is being used in an expression that is not in a spatial context.

Invalid Expression Variable: %s. SOURCE Layer or SOURCE location variables not permitted here.

Invalid Expression Variable: %s. SOURCE Cell variables not permitted here.

The SOURCE keyword is being used in an expression that is not in a recipient context.

Invalid Expression Variable: %s. Event variables not permitted here.

An event variable is being used in an expression that is not in an active event context.

Invalid Expression Variable: %s. Cluster variables not permitted here.

A cluster variable is being used in an expression that is not in an active cluster context.

Invalid Expression Variable: %s. Cell variables not permitted here.

An active cell variable is being used in an expression that is not in an active cell context.

Invalid Expression Variable: %s. Index variable not permitted here.

The Index built-in variable is only available in specific expressions, including Over Index Sequence expressions, indexed output records and array sorting.

Invalid Expression Variable: %s. UNWRAPPED Location variable not permitted here.

The UNWRAPPED keyword is for use only in specific region functions.

Invalid Expression Variable: %s. Temporary System variable (sysTmp) not permitted here."

The sysTmp built-in variable is only available in cost surface regions.

Invalid Expression Variable: %s. Structured variables only permitted in set, list and graph functions.

Attempt to use a set, list, graph or tree variable in a normal expression (which would corrupt the variable).

4 Running scenario script files – general

Errors other than those encountered when loading raster files, model state-space files, landscape events or macros.

Cannot open scenario script file: %s. Check that file exists at path specified and has read access. Paths to sub-scenario are relative to the loading scenario, and cannot use script variables.

The scenario script file may be locked by another application.

Error %s in file: %s at or near [%s][%s]

Syntax error: The message attempts to indicate the general text near the error (first bracketed text) and the specific text when the error was detected (second bracketed text). In some cases, the actual error may not be near the location of when this was identified. Editing scenario files in the LSEditor can help find errors, but they only way to check for correct syntax is to try to run the scenario. One option on the File menu is to open as scenario as a “dry run”, which runs all scenario comments, except for a any simulate commands.

File %s contains an invalid command near text: %s

Invalid command. This message usually also implies a syntax error.

At least one incomplete loop/while/if statement in scenario

While-loops, if-statements and other expressions that have an “opening” and “closing” require the closing (usually the “end” keyword). If not closed, the end of file is reached before expected.

invalid specification - Script variable must be numeric: %s (%s)

The script variable is used in a place that requires a numeric value, but does not contain a numeric value.

need to access vector/array global variable/constant with index: %s

An array is used without an index in a place that requires an index (e.g. for a single number in the array).

out of bound index to vector/array global variable/constant: %s[%d].

The index specified for an array is too big.

cannot access global output variable as a number: %s.

Output record variables can only be using an OUTPUT expression, and not anywhere else.

<fn> not supported for strings

There are a number of scenario function expressions in which script variables (aka string variables) are not permitted to be used as parameters.

WARNING re-assigning global constants has no effect (use external script variables as input to the .sel file): %s.

Attempt to assign a value to a global constant in a scenario. Global constants are constant.

out of bound index for global variable/constant: %s. Valid range: %ld. Index: %d

Index to an array is too large.

invalid specification for script variable value: %s = %s

Error in an assignment to a script variable.

there is no simulation currently loaded

Attempt to run a simulation when no model is loaded.

cannot change to specified directory: %s

Problem changing working directory.

invalid specification for loop: %s = %s

Error in specification of scenario loop expression.

no wildcard character in loop expression: %s

Scenario loop expressions over file names require at least one wildcard ("*") in the file name descriptor.

Cannot find raster named %s to save.

Attempt to save a non-existent raster.

invalid command or file name: %s = %s

The scenario parser was unable to resolve the meaning of the expression (e.g. whether this was a variable assignment, file opening, etc.).

invalid specification for variable or raster: %s

Attempt to assign a variable or raster, but there is a problem with the variable name.

5 Errors During Simulation

The previous simulation has not yet terminated…wait a bit.

Sometimes an action must wait until a simulation has properly ended (which can take some time for large models).

Warning: Cannot close document during a simulation

During a simulation, SELES does not allow rasters used as part of the model state-space.

Cannot reload a dynamic model while a simulation is running

A .sel file cannot be loaded/re-loaded while a simulation is running because it would change or reset the state space.

Could not open scenario record file %s for write access. Check if file is open in another application

When a simulation starts, output record files are opened. If any are locked by another application, the simulation stops.

Setting up output record for variable %s: Cannot locate input <variable type>: %s

When an output record is written to a file during a simulation, the variables referenced must be part of the state-space. Missing state variables should normally be caught when loading the model.

```
Could not open record file %s for write access.

Could not open record file %s for append access.
```

When a simulation starts, output record files are opened. If any are locked by another application, the simulation stops. The output file will be opened for append (not cleared at simulation start-up) if specified in the scenario file.

```
"Output Record length too long!.  Fatal Error."
```

This indicates that the record specification has exceed the space allocated for an individual record. To resolve requires reducing the size of the record output in the landscape event.

5.1 Scheduled raster output

```
Directory specified for output file not found

Directory specified for output file is not a directory

Specified output file is a directory
```

These indicate an incorrect specification of the path and/or name spatial time-series raster output in the .sel file.

5.2 Simulation probe

These errors may arise when using the simulation probe during a simulation.

```
Cannot probe selected property: no preliminary assignments to probe

Cannot probe selected property: no consequent assignments to probe
```

The probe can only select the preliminary or consequent expressions of a property if there is at lease one such expression.

5.3 Warnings that do not stop a simulation

Messages that do not cause a simulation to stop are put in the seles.log file.

Error: attempt to divide by zero"

Division by 0 in SELES is treated as 0 (not infinity). A message is put in the seles.log file, but the simulation is not stopped. Checking seles.log for such errors may help identify problems in a model (because in principle, one should never divide by zero).

6 Other User interface operations

6.1 New Layer (File menu)

"Invalid dimensions"

The number of rows and columns in the New Layer dialog must be > 0.

6.2 Raster View Dialog (View menu)

Cannot use selected image as mask/hue/ height: layers must have the same dimensions.

When overlaying rasters (mask) or using one raster as the hue or height for another, the rasters must be the same dimensions.

6.3 Changing a view name (Name Document on Window menu)

Document name must have at least one character: %s

Raster view names cannot be blank.

Cannot have two windows with the same name: %s

Cannot rename a raster view to the same name as another raster view.

6.4 Value Summary models (StaticModels menu)

The expressions in a Value Model are similar to a macro or a single property of a landscape event, and the same types of error messages may arise.

6.5 Aligning a raster (StaticModels menu)

WARNING: Layers to align may not have the same projection: %d vs %d

SELES does not always recognize different projections, but aligning rasters is only possible if the two rasters have the same projection. Re-projecting grids should be done in a GIS.

Warning: Layers to align should have the same cell size (resolution): %f vs %f

Aligning rasters is only possible if the two rasters have the same cell size. A grid can first be re-scaled to match the grid cell size.

The two layers are already aligned

An aligned grid will not be created if the grids are already aligned.

6.6 Rescaling a raster (StaticModels menu)

Scale factor must be positive

Warning: Scale factor must be greater than zero! Cannot resize.

Rescaling the cell size of raster must be a real value > 0, which represents the ratio of cell sizes before and after the scaling.

6.7 Model report (DynamicModels menu)

Cannot open report file: selesModelReport.txt. Check if file is open in another application.

This may arise if the selesModelReport.txt file is locked by another application.

Appendix 3: Fixing Common Errors

1 Problems with scenario scripts

Errors in scenario scripts tend relate to input files (raster grids, .sel files), script variable assignment and use, global variable (parameter) assignment and folder paths. Syntax issues tend to be less common because there are relatively few keywords and syntactic structures, and some commands are very generic in form. For example, consider the command:

```
y = x
```

The interpretation will depend first on whether `y` is previously defined in the state-space of a loaded .sel file. Here are the cases:

- If `y` is a global variable, then `x` must represent a global variable or constant that is assigned to `y`.
- If `y` is a global constant, then this will result in an error message (since global constant values cannot be changed, by definition).
- If `y` is a spatial variable or spatial constant, then `x` must represent the name of a raster file, which used to re-load the grid for `y`.
- If `y` is not defined, then this will result in an error message.
- If `y` is surrounded by dollar signs (i.e. `$y$ = x`), then this will create/assign or re-assign a script variable to the text "`x`".

If a scenario script doesn't run, but the problematic command(s) are not obvious, one simple way to identify the problematic command(s) is to use multi-line comments (i.e. "/*" and "*/") to comment out sections of the script (e.g. the bottom half). Running the script and reducing/expanding the comments commented out can help isolate the problematic commands. For this reason, it is recommended that normal commenting is done using single-line comments (i.e. "//") .

2 Problems with input tables

A common problem when loading a state-space (.sel) file in a scenario script is errors in the input tables. Here are ways to address some common problems:

- Check that the number of columns is correct
- Ensure that there are no gaps (SELES does not allow blanks in input tables). These can be filled with 0's in Excel using a select/go-to-special/blanks, and pasting in 0's.
- Comments and notes can be included in columns to the right of the main column, as long as there is nothing on the first row. SELES uses the first two to determine the number of columns for the array.

- The first column is by default the row id and does not need to be in sequence or complete. The number of rows is determined by the largest id found. Missing rows will be filled with 0's. If there are duplicate row id's, the latest one will be used.
- Global constants (including legend labels) are often used in input tables. These help with readability and reduce errors. However, these are often the cause of two types of error:
    - Duplicate names: Care must be taken to ensure that all global constants, including legends loaded in the .sel file, must be unique. If SELES detects two definitions of a state variable, it will issue an error message. Names that are close can be differentiated (e.g. if Seymour is the name of a landscape unit and a watershed, the former can be changed to luSeymour and the latter to wsSeymour).
    - Undefined constants: If SELES encounters an undefined global constant name in a loaded text file, it will issue an error message that attempts to identify the file name, row and column of the problem, and the undefined text. To correct, ensure that either the global constant is included in the state-space or changed to a value that is part of the state-space.

3 Other problems with state-space (.sel) files

State-space (.sel) files can become quite complex. Any problems are best addressed using the LSEditor. In most cases, parsing the file will identify the location of the first syntax error encountered. Make sure that a .sel file parses correctly before attempting to load in SELES.

If the location of the error is not obvious, using the technique described for scenario scripts can be helpful to isolate the problematic input file, variable specification or other problem.

4 Problems with landscape events

Any problems with landscape events are best addressed using the LSEditor. In most cases, parsing the file will identify the location of the first syntax error encountered. Make sure that a .lse file parses correctly before attempting to load in SELES.

To confirm compatibility with the state-space file, the following steps can be done:

- Open both in the LSEditor
- Ensure that the .lse file parses without errors on its own
- Then parse the .sel file – this will load the state-space definitions
- Parse the .lse file again, which may uncover incompatibilities (e.g. variables defined with different types)

If the location of the error is not obvious, using the technique described for scenario scripts can be helpful to isolate the problematic expressions or property. It can be helpful to comment out entire properties to narrow down the search.

Index