SLAM

Species **L**ife-Cycle **A**nalysis **M**odules

Paul McElhany (project manager, scientist)
Miroslav Kos (design & development)
Anne Mullan (scientist)
Howard Coleman (development)
2005-2012, **NWFSC/NOAA**

Species Life Cycle Analysis Modules (SLAM) was developed as a tool to investigate factors limiting recovery of listed salmonids, but can be applied to any species for modeling population survival over stages. [more]

The most recent version of SLAM, along with examples and more information, can be found at

http://ecologybox.org/user/SLAM/SLAM

Table of Contents

Table of Contents

Table of Contents

Acknowledgements

SLAM makes use of the software libraries listed below. Licenses for these libraries are included in the "licenses" subdirectory of your SLAM installation.

Apache cglib, commons-collections, commons-logging :
SLAM uses the Code Generation library (cglib) (Copyright 2001-2004 The Apache Software Foundation), Apache Commons Collections (Copyright 2001-2008 The Apache Software Foundation) , Apache Commons Logging (Copyright 2003-2007 The Apache Software Foundation),

For more information about the Code Generation Library, see
http://cglib.sourceforge.net/

For more information about the Commons Collections, see
http://commons.apache.org/collections/

For more information about Commons Logging, see
http://commons.apache.org/logging/

Colt Open Source Libraries for High Performance Scientific and Technical Computing in Java :
For evaluation of various mathematical functions, SLAM uses Java libraries created and maintained by the CERN Colt project – Copyright (c) 1999 CERN - European Organization for Nuclear Research.

For more information about the Colt project, see
http://acs.lbl.gov/software/colt/

dom4j library :
SLAM uses the dom4j library, Copyright 2001-2010 (C) MetaStuff, Ltd. All Rights Reserved.

For more information about dom4j, see
http://dom4j.sourceforge.net

Groovy :
For internal, user-defined scripting, SLAM uses Groovy, sustained and led by SpringSource (http://www.springsource.com/) and the Groovy Community.

Groovy is licensed under the Apache 2 license. For more about Groovy, see
http://groovy.codehaus.org/.

Hibernate :
For persistence and retrieval of Java objects, SLAM uses Hibernate, produced by the JBoss Community – Copyright (C) 2004-2012 Red Hat Inc. and the various authors.

For more information about Hibernate, see
http://hibernate.org/hibernate

HyperSQL HSQLDB :
For data storage and retrieval, SLAM uses HSQLDB, produced by the hsql Development Group – Copyright (c) 2001-2010, The HSQL Development Group.

For more information about HSQLDB, see
http://hsqldb.org/

JFreeChart :
For data display, SLAM uses JFreeChart, produced by JFree.org – (C)opyright 2000-2011, by Object Refinery Limited and Contributors.

For more information about JFreeChart, see
http://www.jfree.org/jfreechart/

JGoodies FormLayout :
For user interface layout, SLAM uses the FormLayout library produced by JGoodies – Copyright (c) 2002-2012 JGoodies Karsten Lentzsch

For more information about the FormLayout library, see
http://www.jgoodies.com/freeware/libraries/forms/

Parallel Java Library :
To support parallel processing, SLAM uses the Parallel Java Library, developed by Prof. Alan Kaminsky in the Department of Computer Science, Rochester Institute of Technology. – Copyright © 2005-2012 by Alan Kaminsky.

For more information about the Java Parallel Library, see
http://www.cs.rit.edu/~ark/pj.shtml

Stochastic Simulation in Java (SSJ) :
For generating values of variables drawn from random distributions and other tasks, SLAM makes use of SSJ, a Java library for stochastic simulation developed under the direction of Pierre L'Ecuyer in the Département d'Informatique et de Recherche Opérationelle (DIRO), at the Université de Montréal – Copyright (C) 2008 Pierre L'Ecuyer and Université de Montréal

For more information about SSJ, see
http://www.iro.umontreal.ca/~simardr/ssj/indexe.html

XStream :
For XML serialization support, SLAM uses the XStream library – Copyright (c) 2003-2006, Joe Walnes; Copyright (c) 2006-2009, 2011 XStream Committers

For more information about XStream, see
http://xstream.codehaus.org/index.html

Introduction to SLAM

In SLAM, you can create, modify, and run very complex life cycle models, but the basic steps involved are simple. In this quick introduction, you go through the basic steps involved in running a SLAM model and examining the output:

1. Opening a life cycle model
2. Adding a scenario
3. Running a simulation
4. Examining the results

You can also take a look at SLAM's interface.

Opening a sample life cycle model

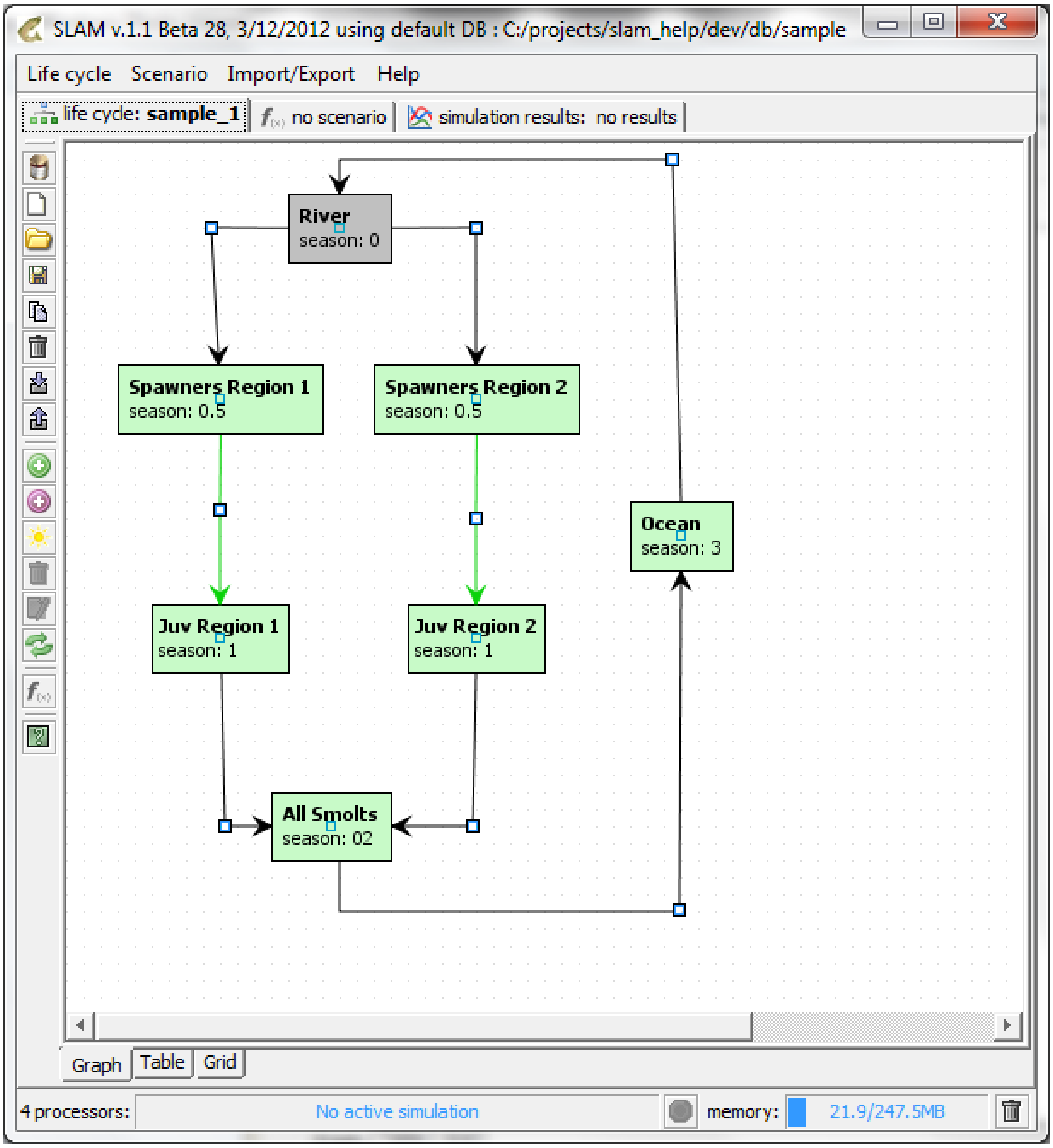
There are three ways to open the sample life cycle model in SLAM:

Load the sample from the database : If the sample database is included in your default SLAM/db directory, switch to it and load the sample_1 life cycle model.

Import the sample from an export file : If you have a copy of the sample_1_l.xml file, import it and save it in your current database as sample_1 . (If you're using the sample database, there's probably already a sample_1 database in it, so you have to change databases or save the life cycle under a different name.)

Create the life cycle : You can create your own sample_1 life cycle model.

When you have the life cycle open, SLAM's graphical display will look like this;



Sample 1 life cycle

Try running the model as-is, without editing a scenario:

1. Create a new scenario by typing Ctrl+Alt+N.
2. Save the scenario without editing it, by typing Ctrl+Alt+Shift+S. When prompted, pick a name for the scenario — though you'll be deleting it shortly.
3. Run a simulation based on the sample 1 life cycle and the default scenario by typing Ctrl+R.
4. Examine the outcome of the simulation in the results viewer by selecting the simulation results tab. You'll see that the stage abundances simply oscillate, reflecting the simple, closed-loop structure of the life cycle model and the fact that no real dynamics have been built into the scenario.

Make things a little more interesting by adding a scenario for the life cycle model. (You can delete the default scenario created above, if you like.)

Adding a scenario to the sample life cycle

There are three ways to add a scenario to the sample 1 life cycle model:

Load the scenario from the database : If you loaded the sample 1 life cycle from the sample database, a scenario named sc_1 should be associated with the life cycle. Load the scenario from the database.

Import the scenario sc_1 from an export file : If you have a copy of the sample_1_l.xml file, import the sc_1 scenario and save it in your current database as sc_1 . (If you're using the sample database, there's probably already a sc_1 scenario associated with the sample 1 life cycle; you can either change databases or save the scenario under a different name.)

Create the scenario : You can create the sc_1 scenario for the sample 1 life cycle model.

With a scenario in place you can test out the model by running a simulation.

Running a simulation

With a life cycle model and a scenario loaded (for example, the sample 1 life cycle and the sc_1 scenario), run a simulation by typing Ctrl+R. When the simulation configuration dialog appears, enter 100 for years and 100 for trajectories , (though other sets of values will work as well).

While the simulation runs, the simulation status bar at the bottom of the SLAM main window will display the message finished 0 of 1 simulation(s) . On reasonably modern computers, simulating sample 1/sc_1 for 100 years and 100 trajectories should take less than a minute. When the simulation finishes, the … 0 of 1 … message will be replaced by No active simulation and SLAM will automatically open the results in the results viewer, where you can examine them.

Browsing results

After the simulation finishes running, the results viewer is automatically opened in the results tab with the simulation-generated data loaded into it. You'll see a list of simulations on the left side and a list of life stages on the right side. Select your simulation results on the right *and* one or more life stages on the left to see results in the graph window.

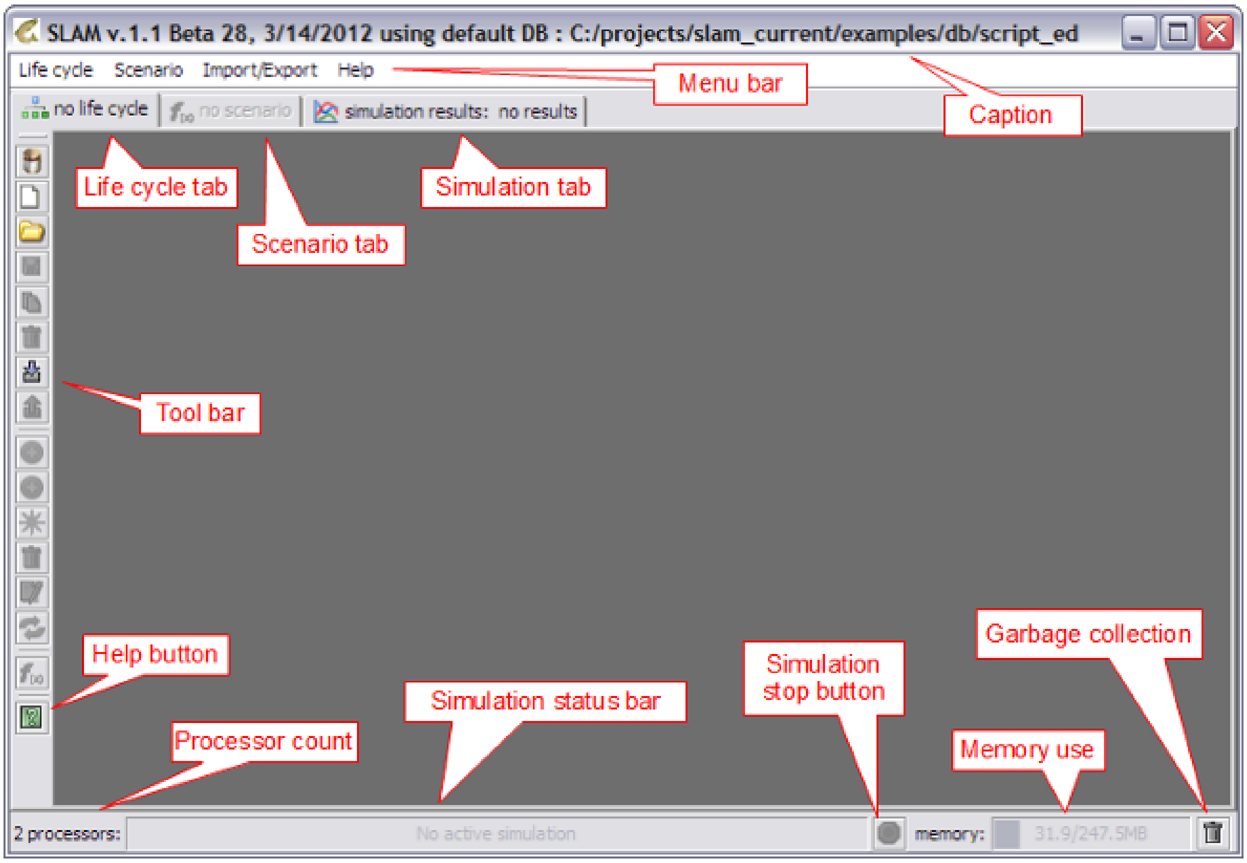
As you select different life stages in the right-hand panel, you'll see the graphical display depict them. By choosing different chart types in the lower right corner of the display, you'll see different ways of looking at the output abundance series.

There's much more to know about the results viewer.

After that quick run-through, it may be helpful to read about the SLAM interface.

The SLAM interface

When you first open SLAM, the interface will look like this. Various parts of the interface are labeled and are explained below.



The SLAM interface

Interface features:

- **Caption:** The application caption contains the identifier of the SLAM version and the location on disk of the current SLAM database. In the example the version is 1.1 Beta 28, 3/14/2012 (the date refers to when SLAM was built — it isn't the day on which you're running SLAM) and the database is stored in these two files:

  C:/projects/slam_current/examples/db/script_ed.script
  C:/projects/slam_current/examples/db/script_ed.properties
- **Menu bar:** There are four top-level menu items:
  - **Life cycle** contains options for creating, loading, saving, and deleting life cycles, as well as for specifying the current SLAM database.
  - **Scenario** contains options for creating, loading, saving, and deleting scenarios, as well as options for configuring and running simulations.
  - **Import/Export** contains options for importing and exporting life cycles and scenarios.
  - **Help** contains options for accessing the help system.
- **Life cycle tab**: This tab displays the graphical editor in which you edit the current life cycle.

- **Scenario tab**: This tab displays the scenario editor in which you can configure the current scenario.
- **Simulation results tab**: This tab contains SLAM's results viewer for examining the outcome of simulations.
- **Tool bar**: Tool bar buttons allow you to perform actions on the object loaded in the selected tab — if the life cycle tab is selected, then clicking the tool bar buttons will perform operations on the life cycle; if the scenario is open, then the tool bar functions apply to the scenario; if the results tab is displayed, the tool bar functions operate on the simulation results viewer. If you hover the mouse over a tool bar button, a tool tip appears to give you a hint about the current function of the button.
- **Help button**: This provides access to SLAM's help documentation. You can also see the help pages by way of the Help menu item.
- **Processor count**: This field displays the number of processors available on your computer. SLAM attempts to distribute simulation computations across multiple threads when there are processors to support them.
- **Simulation status bar**: Messages displayed in this field report the status of simulations in progress.
- **Simulation stop button**: This button allows you to stop the simulation that is currently running.
- **Memory use**: This field shows the memory being used by SLAM. Memory is displayed in the format *nn.n/mm.m*MB, where the *n*s represent the current amount of occupied memory and the *m*s show the maximum memory SLAM can use.
- **Garbage collection**: Clicking this button will cause the Java garbage collector to schedule a cleanup of allocated memory that is no longer required.

Theory

Species Life Cycle Analysis Modules (SLAM) was developed as a tool to investigate factors limiting recovery of Endangered Species Act listed salmonids, but can be applied to any species for modeling populations based on survival rates. SLAM is a population model, built and defined by user input to estimate a stock or population s survivability. The model tracks numbers of individuals, through stages based on rates of survival from one stage to another, and builds in uncertainty and variability. A method to test the sensitivity of the model is also built into the user interface.

Biological perspective

Salmonids (species which migrate from freshwater to saltwater and back in their life cycles) can have complex life histories. This is especially true for Chinook salmon (Onchorhynchus tshawytscha) and steelhead (O. mykiss). Certain runs (populations in a given drainage) of salmon spawn and out-migrate at several times per year (usually denoted by either the season when fry start to emerge from the spawning beds, or when the fish return to spawn, i.e. spring Chinook). Because of their complex life history strategies, salmonid survival is particularly difficult to parameterize (capture in a mathematical representation) in a model.

Life cycle, season, generation

1. Life stages

   Life stages in the model are meant to reflect the life stages we recognize in an actual population. Life cycle diagrams can be built and parameterized (and tracked separately) in the model for wild and hatchery origin fish, as well as different types within a certain species/race (as in lake-type versus stream-type sockeye).
2. Transitions

   Transitions in the model represent a linkage between life stages, and function as a rate of survival from one life stage to the next. If there is a survival rate that the user would like to have captured, or modeled, this should be entered in the model as a transition.
3. Cohorts: pooled group, same value group, initial abundance

Cohorts are the number of fish-years that capture an entire life cycle of a given spawning year, similar to the idea of a generation. For example, we know the Lewis River fall Chinook population spawns, and then the fish that hatch and grow from that spawning event will all out-migrate and return to their natal river to spawn within 5 years. So 5 years is the fish-year cohort for that population; you have to model at least 5 years out to capture the natural range of spawners returning from that initial brood year.

There is a way in the model to group together life stages of similar behavior or survival rates. This is the pooled group function, which sets the survivals the same for each stage in that group, unless the user changes it. If the user changes the rate for a life stage within the pooled group , only that life stage transition will change and the others will remain the same. The advantage to this can be something like pooling together hatchery origin fish, which might have consistently lower survivals than wild origin.

Initial abundances can be set for any life stage. The user can define a number, to initialize or seed the life stage and that phase of the trajectory. The initial abundance can still be chosen from a distribution with associated uncertainty, but the difference is that model does not automatically calculate a number for the fish abundance at that life stage, but instead uses the initialization number. This is recommended to use for each fish-year , since the first year (and the first several cohorts) will take a long time to regulate and populate and will show abundances of zero or very small. If the initialized stage comes after a modeled stage (a non-initialized stage), then the number of fish that was calculated in the modeled stage is carried over, and added to the initialized abundance.

Simulation scenario

interpretation, scenario parameters (initialization of starting life stages, recruitment functions)

1. simulation process: sampling from distributions
2. sensitivity analysis

Biological and mathematical definitions for salmon

- Recruitment functions

  Survival is the probability that an individual will mature to the point where it is capable of reproducing. There are often different rates of survival, for different stages (life stages, mortality factors, obstructions to food or to get to critical habitat). Survival is linked in some species to a certain type of food availability, or temperature, for example. If there is not enough food available, the rate of survival for the individual will decrease. We then extrapolate the probabilities of survival for one individual to the population. Methods of assessing population size and recruitment rates (number of fish that become spawners, per individual; also called productivity) have developed where there is a fishery (a harvested population). These methods were often initially created to assess whether harvest levels are sustainable to the population. The method of assessing a population is encoded into a mathematical function, based on biological and ecological theory. These equations are often referred to as recruitment functions , and graph the number of fish, or abundance of the population (spawning adults), against the number of offspring at a given age. The rate of survival is captured by the transition rate from one life stage to another. Rates of survival are based on a recruitment function, or a specification of how many successful (living, breeding) adults are produced from one given parent.

- Population, life history
- Density-dependence, recruitment

Life cycles

A SLAM *life cycle* represents an organism’s population life stages and the transitions between them. SLAM life cycles can also model non-reproductive relationships between life stages and transitions as well as static or dynamic external influences on transitions.

Life cycles can be arbitrarily complex, but in practice a single life cycle generally represents one species and one locality (*e.g.*, watershed).

In SLAM you can create new life cycles and modify existing ones. Once a scenario has been defined and saved for a life cycle, the life cycle is read-only and can no longer be edited. You can save the life cycle under a different name, however, creating a new, scenario-free copy which can be edited. In this way, you maintain both your new and original versions.

There are two basic, population entities in a life cycle: life stages and transitions. Life stages represent the number of individuals in given developmental phases during given intervals. Transitions represent the survival rates from stock stages into recruit stages.

Life cycles can also contain environment variables and dynamic drivers which serve as *influences* on relationships between stock and recruit stages. Environment variables can be constants, values drawn from probability distributions, or data read from input files. A dynamic driver's value is computed during each trajectory by a script that can read values of other elements in the life cycle model.

- Creating and editing life cycles
- Saving life cycles
- Loading existing life cycles
- Deleting life cycles
- Read-only life cycles

Creating and editing life cycles

You create a new life cycle in SLAM by selecting the ‘Life cycle/new life cycle’ menu item, by typing Ctrl+N (that is, hold the Ctrl key down while typing the letter ‘n’, or by clicking the ‘new life cycle’ button in the toolbar on the left-hand side of SLAM's main window.

When you have created the new life cycle, the ‘life cycle’ tab in SLAM's main window will be displayed, with the tab label reading ‘life cycle: **new life cycle**’.

You populate your life cycle model by creating elements (life stages, environment variables, dynamic drivers, and transitions) in the graphical work space in the life cycle tab. Initially this space is a blank work area with a dotted grid. When you create new elements, you can drag them wherever you like; if you drag an element to the right or bottom edge of the work area, the view will scroll and scroll bars will appear. You can return to your original spot by using the scrollbars or by dragging an element back to the left or top border.

Saving life cycles

You can save a life cycle at any time, by selecting the ‘Life cycle/save life cycle’ menu entry, by typing Ctrl+S, or by selecting the ‘save life cycle’ button in the left-hand tool bar.

When you save a life cycle for the first time, you'll be prompted to name it. You can't rename an existing life cycle, but you can save a copy of the life cycle that is currently open under a new name. To do that, select the ‘life cycle/save life cycle as&#133’ menu item, type Ctrl+Shift+S (that is, hold down the Ctrl and Shift keys while typing the letter S, or select the ‘save life cycle as…’ button in the tool bar on the left. A dialog appears in which you specify the new life cycle name — you'll be informed if the name you choose already exists.

When you save a copy of a life cycle via ‘Save life cycle as…’, SLAM automatically closes the old copy of the life cycle and opens the new one. The new name will now appear in the label of the ‘life cycle’ tab.

The new copy of the life cycle will have no scenarios attached to it and will be editable. Once you have finished editing the new life cycle, you can attach scenarios from the old life cycle to the new one by importing previously exported scenario files.

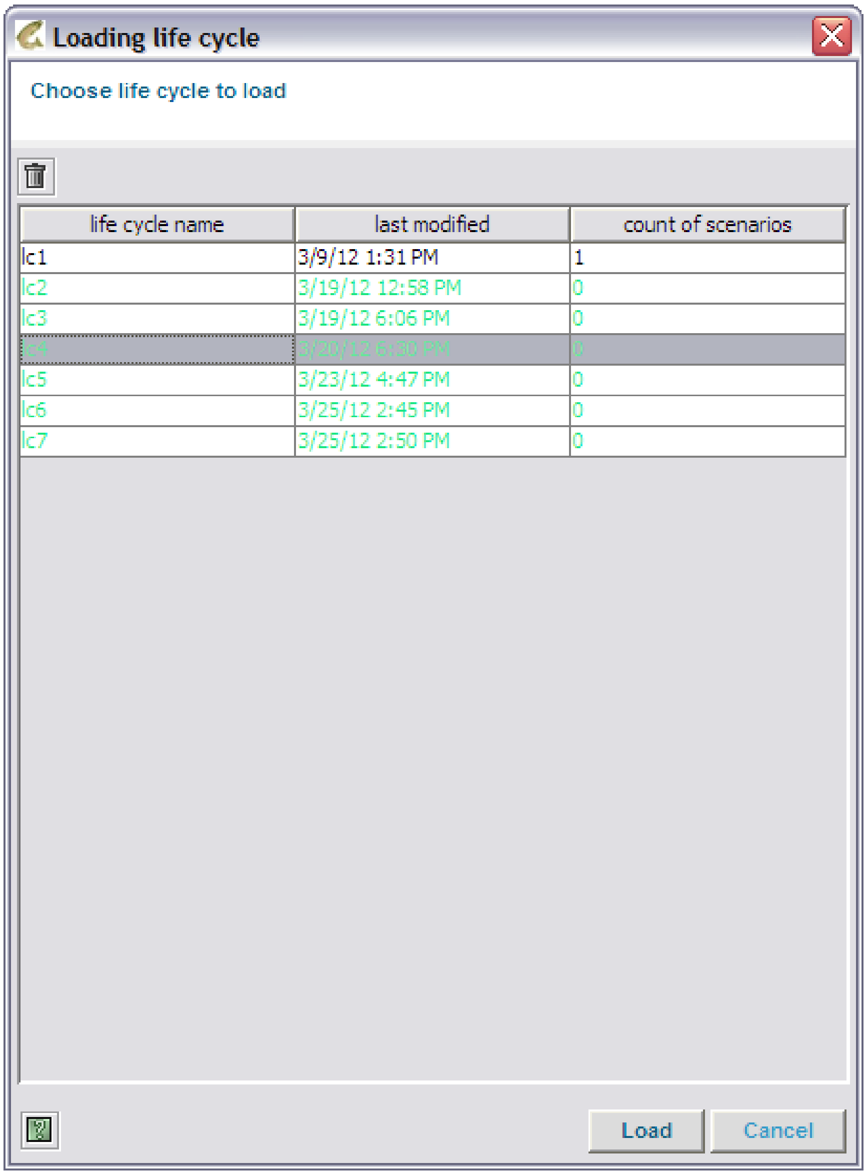
If you have made changes to a life cycle before you close SLAM, you'll be asked whether or not you want to save your work. If your response is “no,” SLAM will discard any pending changes. The same thing happens when you attempt to load or create a new life cycle or create a scenario when you have not saved changes to the current life cycle.

When you save a life cycle, it is stored in SLAM's current database. The database is stored in a pair of disk files. The life cycle will be saved into the current database, but you can change databases while the life cycle is open, then save the life cycle model to the new database.

Loading existing life cycles

You can load a life cycle stored in the database by selecting the ‘life cycle/load life cycle’ menu item, by typing Ctrl+O, or by clicking the ‘load life cycle’ button on the left-hand toolbar.

The ‘Loading life cycle’ dialog will appear, displaying a list of all life cycles in the current database. The list includes the name of each life cycle, the date and time that the life cycle was most recently saved, and the number of scenarios that have been created for the life cycle. Life cycles with no scenarios can still be edited; those with scenarios attached cannot. Scenario-free life cycles are listed in green to indicate that they are editable.



life cycle name	last modified	count of scenarios
lc1	3/9/12 1:31 PM	1
lc2	3/19/12 12:58 PM	0
lc3	3/19/12 6:06 PM	0
lc4	3/20/12 6:30 PM	0
lc5	3/23/12 4:47 PM	0
lc6	3/25/12 2:45 PM	0
lc7	3/25/12 2:50 PM	0

The Loading life cycle dialog

Select a life cycle by clicking it. The ‘Load’ button will be enabled: click it and the selected life cycle will be loaded. (You can also simply double-click the life cycle.) If you change your mind, click the ‘Cancel’ button.

Deleting life cycles

You can delete the life cycle that is currently open by select the ‘life cycle/delete life cycle’ menu item, by typing Ctrl+D, or by clicking the ‘delete life cycle’ button 🗑. If you select ‘Yes’ in the confirmation dialog that appears, the current life cycle and any scenarios attached to it will be deleted from the database. (*Deleted life cycles are **not** recoverable unless you have exported them.)*

You can also delete life cycles that are not currently loaded by opening the ‘Loading life cycle’ dialog (see above). Select the life cycles you want to delete and click the ‘Delete’ button 🗑 above the list of life cycles.

Read-only life cycles

You can make changes to a life cycle as long as the life cycle has no scenarios attached to it. Once a scenario has been created and saved for a life cycle, the life cycle becomes *read-only* and can no longer be edited.

A read-only life cycle is labelled '[READ ONLY]' in the life cycle tab. The stages of a read-only life cycle do not display the small, central squares that you drag to create new transitions. In the 'Load life cycle' dialog, the names of read-only life cycles are black, not green in color and their scenario counts are greater than zero.

You *can* modify the layout of read-only life cycles, as this doesn't affect their structure for simulation purposes.

You can render a life cycle editable again either by deleting all scenarios attached to the life cycle or, less destructively, by saving a copy of the life cycle under a different name using the 'save as' functionality described above in the Saving life cycles section.

Creating and editing life stages

Important! You can only edit a life cycle if it is not **read-only**.

Life stage elements in SLAM represent biological life stages: collections of fish of a certain age and phase of development, possibly divided further into groups. Life stages and the transitions between them are the basis of SLAM's model of the life history of a population. The value of a life stage is the abundance of the group it represents; this abundance is calculated over time through each trajectory in a simulation run.

Stages are connected to other stages by means of transitions, which define the survival rate from stock to recruit stages. More than one recruit stage can derive from a given stock stage; in this manual such a stock stage is called a splitting stage or simply a split .

The functions that define survival ratios of transitions and splits can depend on the abundances of other stages and on variables with values that are fixed, read in from data files, or calculated from mathematical relationships during the simulation run. Inputs of this kind are called influences and can be applied in various ways.

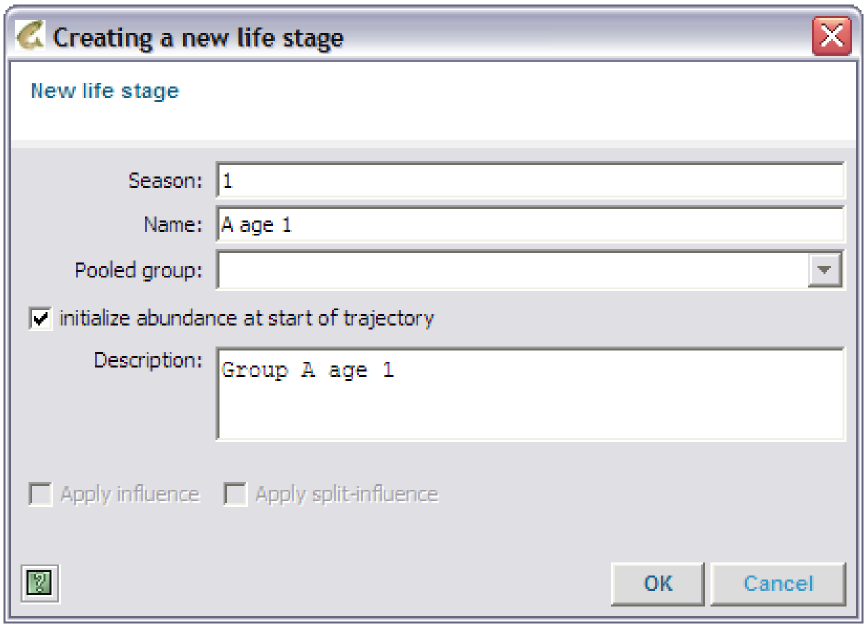
- Defining stages
- Defining transitions between stages
- Defining influences on transitions
- Defining self-influencing stages
- Defining split influences
- Defining self-influencing splits

Defining stages

Important! You can only edit a life cycle if it is not marked **read-only**.

You create a new life stage by clicking the 'new life stage' button on the toolbar to the left of the life cycle diagram. You can edit an existing stage by double-clicking it in the life cycle diagram.

When you click the 'New life cycle stage' button, the life stage editing dialog appears.



Season: 1
Name: A age 1
Pooled group:
✓ initialize abundance at start of trajectory
Description: Group A age 1
Apply influence
Apply split-influence

Stage editing dialog

Configuration details

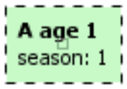
- **Season** *required*
  An integer or decimal fraction corresponding to the year associated with this life stage.
- **Name** *required*
  The name that will identify this life stage in the life cycle graph, simulation results, and everywhere else in SLAM.
- **Pooled group**
  The pooled group to which the stage belongs. You can select an existing pooled group name, if any have been defined, from the drop-down list or type in a new one.
- **Initialize abundance at start of trajectory**
  If this box is checked, the stage will be initialized to a starting value at the beginning of a trajectory. Otherwise, its abundance will initially be zero. If the initialization option is selected, the details of initialization can be configured in the scenario editor.
- **Description**
  Text documenting whatever you'd like documented about the stage.
- **Apply influence**
  If this box is checked, the transition from this stage to its recruit stage will be influenced by a script-defined function. You can configure the influence script in the scenario editor.
  The 'Apply influence' option is available if there is one and only one transition from this stage to another stage. The check box will be grayed-out (disabled) if no transitions originate from this stage or if more than one transition does (*i.e.,* if the stage splits.)

When the stage is first created, no transitions originate from it and the 'Apply influence' option is not available. After this stage has been created and you draw a transition from it to a second stage, you can edit this stage by double-clicking it and the 'Apply influence' option will be available.

- **Apply split-influence**
  If this box is checked, the ratios of recruit abundances derived from this stage can be influenced by script-defined functions. *This option is available only for splitting stages; if fewer than two transitions originate from this stage, the 'Apply split-influence' check box will not be enabled.* When this stage is first created it will have no transitions associated with it, so the 'Apply split-influence' check box will be disabled. Once you have created the stage and defined at least two transitions from it to recruit stages, you can edit the stage by double-clicking it; then, the 'Apply split-influence' option will be enabled.

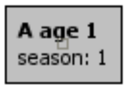
For example, the stage defined in the dialog above is named 'A age 1', it is assigned to season 1, it is not a member of a pooled group, and it will be initialized according to its scenario-dependent configuration at the beginning of each trajectory.

The appearance of the new stage in the life cycle graph will depend on how you define it. Below is the stage that would result from the new stage dialog pictured above: a green rectangle containing the stage name and season.



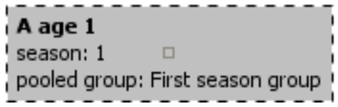
Initialized life cycle stage graph element

If the life cycle stage is not to be initialized, it is represented in the graph by a gray rectangle.



Non-initialized life cycle stage graph element

A life cycle stage defined to belong to a pooled group called 'First season group' looks like this:



Non-initialized, pooled-group member life cycle stage

Newly-created stages will almost certainly not be where you want them to be in the life cycle graph, but you can move them around.

Transitions

Important! You can only edit a life cycle if it is not marked **read-only**.

Transitions in SLAM represent the dynamics of a population model. Changes in abundance across transitions represent either the reproductive rate of the parent stage or the survival rate from a stock stage to its recruit stage(s).

- Reproduction and survivorship transitions
- Creating transitions
- Editing transitions
- Influences on transitions

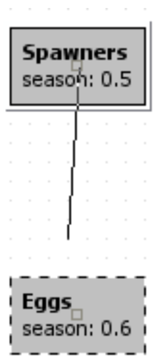
Reproduction and survivor transitions

SLAM allows transitions to be defined as *reproduction* or *survivorship* transitions. Survivorship transitions represent the fractional survival of members of the stock stage to become members of the recruit stage or stages. Natural or artificial attrition may result in the resulting recruit population(s) adding up to fewer individuals than were present in the stock stage. But there cannot be more recruit individuals than were present in the stock stage. For any survivorship calculation automatically limits the total number of recruits to be no more than the abundance of the parent stage.

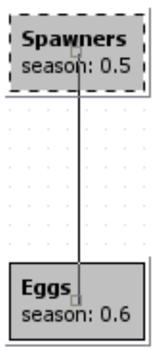
Reproduction transitions permit the creation of *new* individuals. The number of recruits is limited only by the nature of the functional relationship you define for the transition; it can be greater than the abundance of the stock stage.

Creating transitions

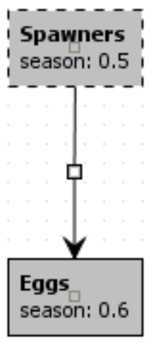
A transition between two life stages can be created in the life cycle diagram view by dragging and dropping a transition arrow from one life stage onto another. Left-click to grab the center of a life stage box (denoted by a small square), then drag to the center of the destination life stage box. While there is no undo operation to remove a transition, you can select the transition by clicking it, then remove it with the Delete key.



Drag transition to target stage



Drop transition on target stage

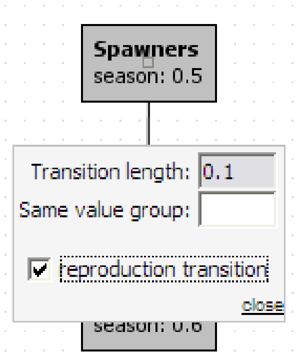


New created transition

If the transition you're trying to define is not allowed (for example, because there is already a transition between the same stock and recruit stages), a dialog will let you know, and the transition will not be created.

Editing transitions

By default, a new transition is designated to be a survivorship transition. To edit the transition, double-click the arrow that represents it in the life cycle graph.

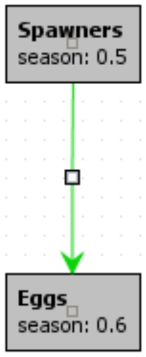


Transition length: 0.1
Same value group:
☑ reproduction transition

Transition editor with reproduction selected

The Transition length field displays the interval in simulation years represented by the transition. The Same value group is not currently in use.

The Reproduction transition check box determines the type of transition relationship. If you select it, the transition becomes a reproduction transition. When you close the editor by clicking close (or clicking anywhere in the life cycle graph outside the editor), the transition arrow will turn green to indicate its state. (In theory, the color change is immediate; in fact, you may have to save the life cycle before the entire arrow turns green.)

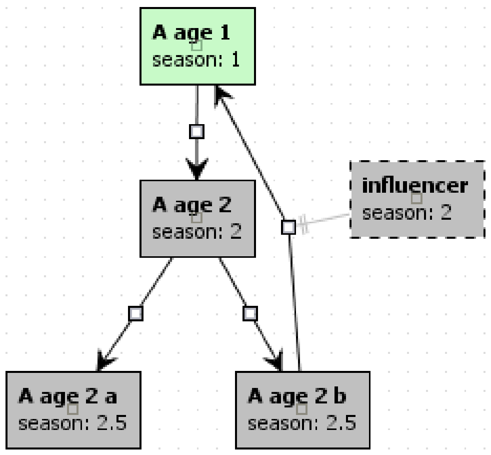


Reproduction transition

Defining influences on transitions

SLAM provides a number of transition relationships that define how the abundance of a stock stage is translated into the abundance(s) of its recruit stage(s). If you need more flexibility in calculating recruit from stock abundances, you can define an influence on a transition. Influenced transitions are defined by scripts which compute the value of the recruit abundance based on the stock abundance and the abundances or values of any life cycle elements defined as influencers for the transition.

In the life cycle graph segment shown below, the influencer stage is an influence on the A age 2 b — A age 1 transition. This means that the abundance of the influencing stage will be available as a parameter to the script that defines the transition relationship. (Scripts, like other transition relationships, are configured in the scenario editor.) Notice that the influence is indicated by a light gray arrow to distinguish it from a stage-to-stage transition in the life cycle graph.



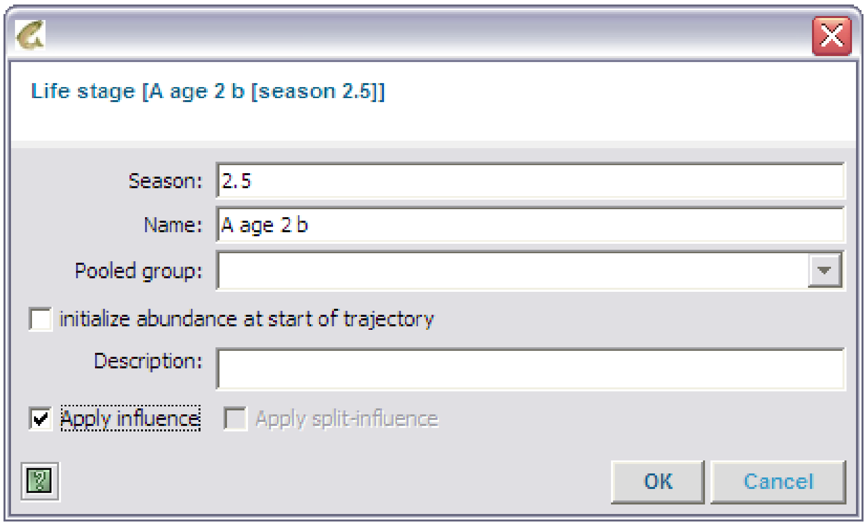
Transition influenced by a stage

Life cycle stages can also be defined as self-influencing.

Self-influencing stages

It isn't necessary to assign external influences to a transition in order to control its behavior with an influence script. If no external life cycle values are needed as parameters by the transition relationship, you can define the transition's stock stage to be self-influencing ; this assigns a script to the transition just as though an influence were defined on the transition. You then configure the script in the scenario editor.

To make a stage self-influencing, double-click its graph element to display the stage configuration dialog, and select the Apply influence check box as shown below.



Life stage [A age 2 b [season 2.5]]

Season: 2.5

Name: A age 2 b

Pooled group:

☐ initialize abundance at start of trajectory

Description:

☑ Apply influence ☐ Apply split-influence

Defining a stage as self-influencing

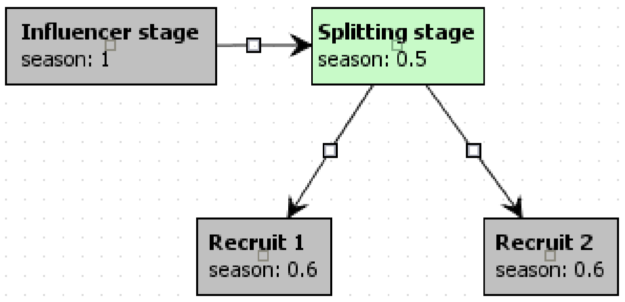
Only stages with exactly one recruit stage can be defined as self-influencing. In other cases, the Apply influence check box will be grayed-out. You can also designate splitting stages as *self-influenced splits*.

There is no visual indication in the life cycle graph that a stage is self-influencing. To check, double-click the stage to open its configuration dialog to see if the Apply influences box is checked.

Split influences

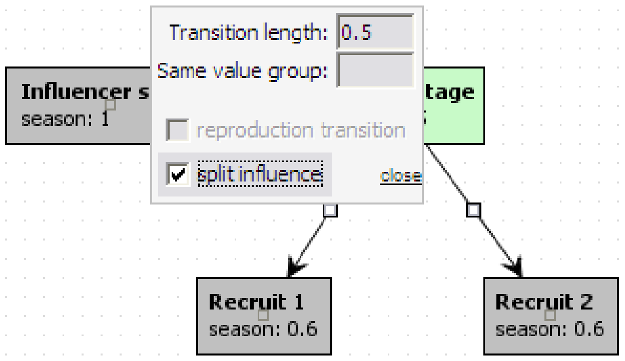
Life cycle stages can influence the abundance ratios of splitting stages — stages with more than a single recruit stage. This means that the abundance of the influencing stage will be available as an input parameter to the scripts that calculate split ratios during the simulation run.

You make a life cycle stage influence a splitting stage in the same way you create a transition from one to the other, by dragging the mouse cursor from the center of the influencing stage to the target stage. By default, the resulting relationship will be a transition. An example graph segment is shown below, with Influencer stage influencing Splitting stage .



Sample stage-to-splitting-stage transition

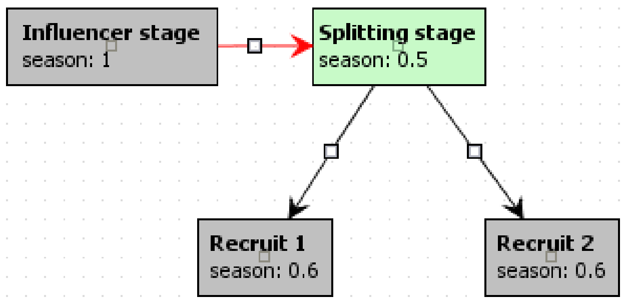
To convert the transition to a split influence, edit the transition by double-clicking it. The transition editing dialog is displayed as shown.


Transition length: 0.5
Same value group:
reproduction transition
[✓ split influence]

Transition editing dialog with split influence selected

The transition editor now displays an additional control, the split influence check box, because the target is a splitting stage. Notice that the influence check box and the reproduction transition check box are mutually exclusive: before you can select one of them, the other must be cleared.

When you select split influence and close the editor, the arrow will change color from black to red to indicate that it now represents a split influence instead of a transition. (As is the case for reproduction transitions, the arrow may not change color completely until you have saved the life cycle.)

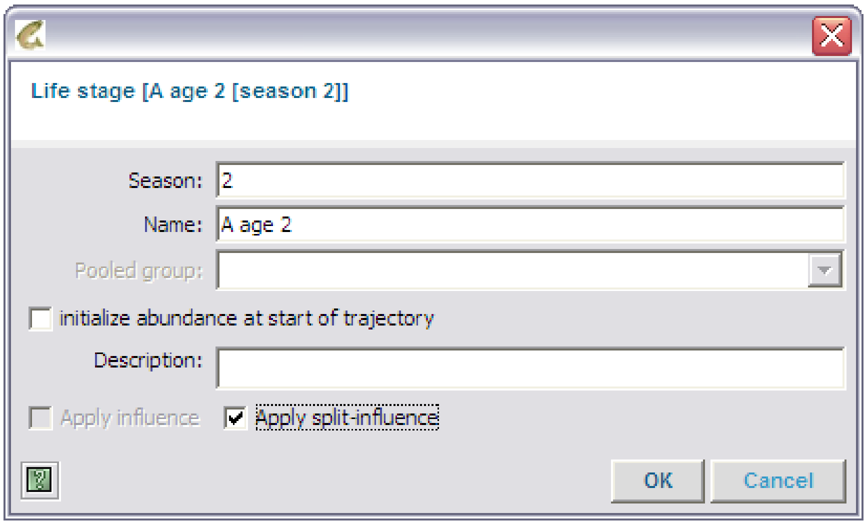


Stage influencing a splitting stage

Now when you edit the splitting stage in the scenario editor, you will have the option to create influence scripts to calculate the split ratios, and these scripts will be able to use the abundance value of Influencing stage as an input parameter.

Self-influencing splits

You can use scripts to compute split transition abundance ratios without defining external split influences. Once you have constructed a split (by creating at least two transitions to recruit stages from the stock stage), double-click the stock stage to display the stage-editing dialog:



Life stage [A age 2 [season 2]]

Season: 2
Name: A age 2
Pooled group:
☐ initialize abundance at start of trajectory
Description:
☐ Apply influence ☑ Apply split-influence

Configuring a self-influenced splitting stage

Select the Apply split-influence check box as shown and click OK. The appearance of the splitting stage element does not change, but when you edit the split in the scenario editor, you'll have the option of applying scripts to the split transitions. Since external influences are not defined, external element values will not be available as inputs to those scripts.

Creating and editing environment variables

Important! You can only edit a life cycle if it is not **read-only**.

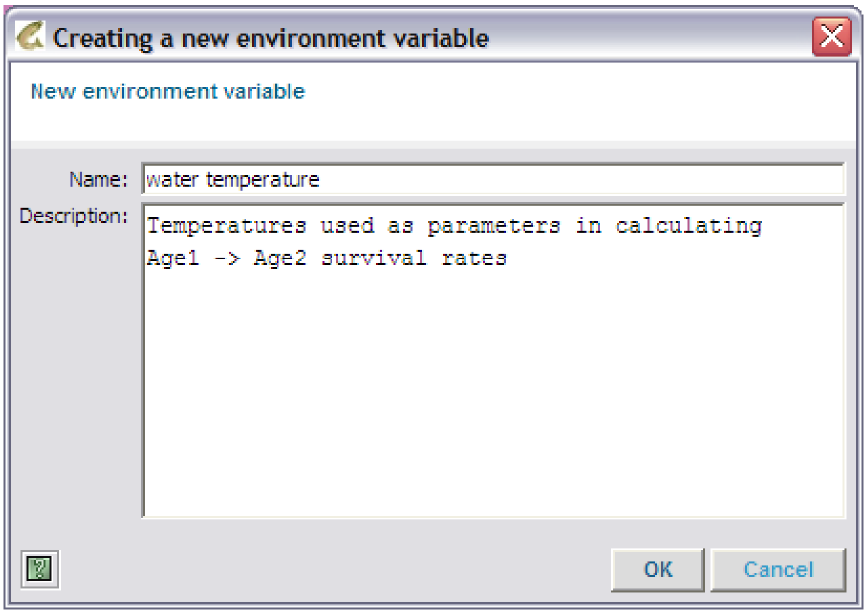
You can use environment variables to influence transitions, splits, or dynamic drivers by way of user-generated scripts. In a life cycle model, such variables typically represent external effects or inputs you would like to incorporate into your model. SLAM environment variables are SLAM parameters, so their values can be constants, draws from probability distributions, or data read from files.

- Creating environment variables
- Creating environment variable influences

Creating environment variables

With a life cycle loaded, create an environment variable by clicking the 'new environment variable' button ☀ in the toolbar on the left side of SLAM's main window.

A dialog appears in which you provide a name for the new environment variable (required) and a description of it (optional but advisable).


Creating a new environment variable

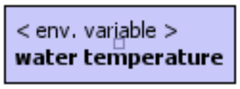
New environment variable

Name: water temperature

Description:
```
Temperatures used as parameters in calculating
Age1 -> Age2 survival rates
```

New environment variable definition dialog

The new environment variable will appear in the life cycle graph as a blue (depending somewhat on your operating system and computer display) rectangle. Left-click and drag it to the preferred location in the life cycle layout.



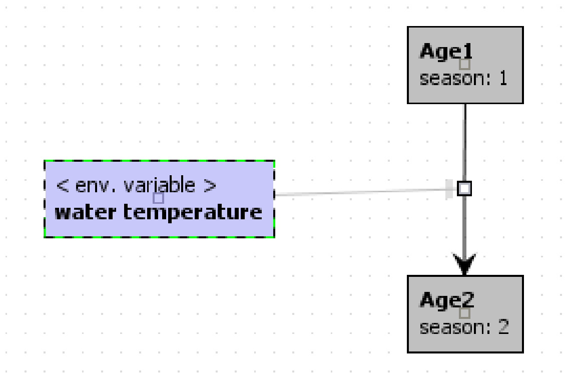
Environment variable element

To delete an existing environment variable, select it by left-clicking and hit the Del key.

Creating environment variable influences

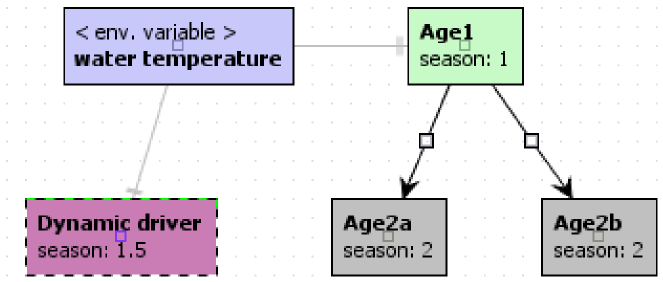
Once you've added an environment variable to the life cycle model, you need to tell the model what the variable influences. Do this by left-clicking the center of the environment variable rectangle (a small square) and dragging the mouse to the transition, splitting stage, or dynamic driver that will be influenced. (A single environment variable can influence as many elements as you like.) A gray arrow will now connect the variable's rectangle to its influence targets.

Below, an environment variable influences a transition.



Environment variable influencing transition

Here, an environment variable influences both a splitting stage and a dynamic driver.



Environment variable influencing split and dynamic driver

The steps described above enable you to create environment variables and define the other life cycle elements they influence. To define how the values of environment variables are determined, create a scenario and use the scenario editor to configure the life cycle's environment variables.

Remember that once you attach a scenario to a life cycle, the life cycle becomes read-only. After that, you can no longer create or delete environment variables or change their influence targets. You can, though, still edit environment variable configurations in the scenario editor.

Creating and editing dynamic drivers

Important! You can only edit a life cycle if it is not **read-only**.

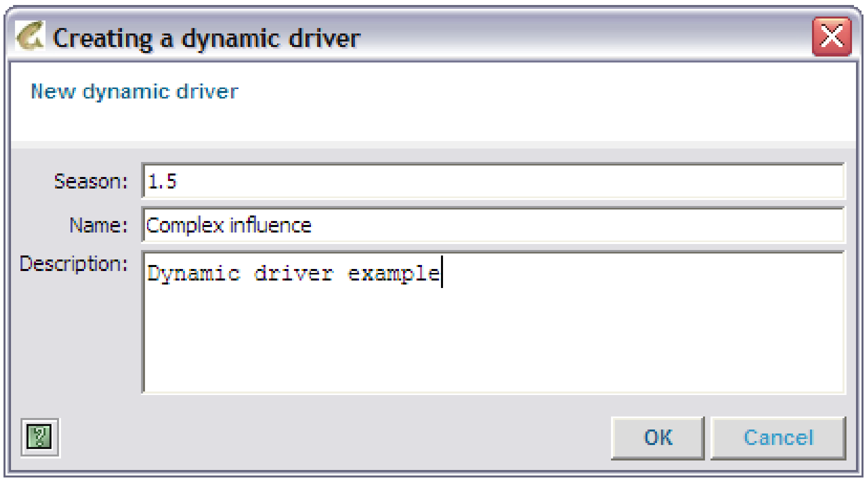
Dynamic drivers, like environment variables, are life cycle elements that you can use to influence transitions, splits, or other dynamic drivers. Dynamic drivers are more flexible than environment variables because their values are calculated by user-generated scripts that can accept as input parameters both the most recent value of the driver itself and the values of influencing stages, environment variables, and other dynamic drivers.

- Creating dynamic drivers
- Connecting dynamic drivers

Creating dynamic drivers

Once you have loaded or created a life cycle, you can create a dynamic driver by clicking the ‘new dynamic driver’ button on the left-hand toolbar.

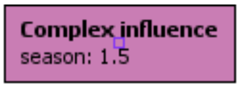
A dialog appears in which you specify the season and the name of the new dynamic driver (both required) and an optional description.



Season: 1.5
Name: Complex influence
Description: Dynamic driver example

New dynamic driver definition dialog

The new dynamic driver is represented in the life cycle graph as a purple rectangle. Left-click and drag it to the preferred location in the life cycle graph.



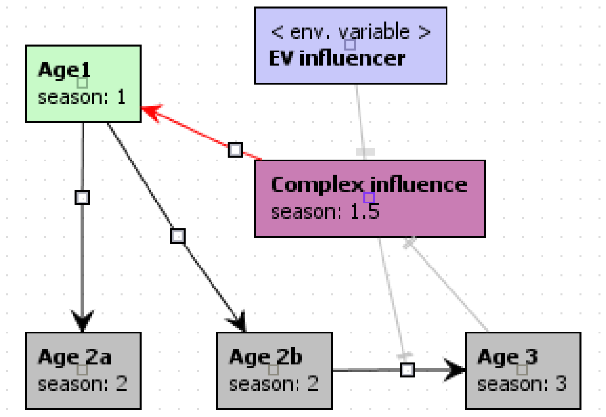
Dynamic driver element

Connecting dynamic drivers

Once you've created a dynamic driver, you'll connect it to the elements it will influence and to the elements it is influenced by. Define influence targets by left-clicking the center of the dynamic driver (a small square) and dragging to the appropriate transition, splitting stage, or other dynamic driver. The value of the dynamic driver will be available as an input to the scripts associated with the influence targets.

Life cycle elements that influence the driver will be available as inputs to the driver's script when you configure it in the scenario editor. Stages, environment variables, and other dynamic drivers can all act as influencers on a driver. Connect influencing elements to the driver by dragging their centers to the driver element in the life cycle graph.

Below, a dynamic driver is influenced by an environment variable and a life cycle stage. The driver influences a transition and a splitting stage. (Note that influence arrows from dynamic drivers to splitting stages, like those from other stages to splits, are colored red in the diagram, while transition influences are gray.)



Dynamic driver influencing a split and a transition

Once you have created the dynamic drivers you need for your model and have connected up their influencers (inputs) and influence targets, you will configure the details of how the drivers work by creating a scenario and configuring the drivers in the scenario editor.

Remember that once you have created a scenario for your model, you will no longer be able to create or delete elements, transitions, or influences in your life cycle model. The configuration details of how the existing elements (including dynamic drivers) function can always be edited, however, in the scenario editor.

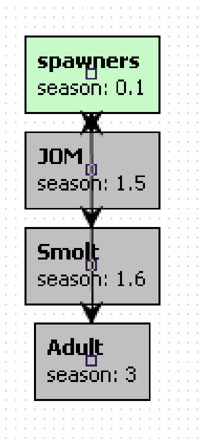
Managing life cycle layout

Important! While you can only edit a life cycle that is not **read-only**, you **can** modify the layout of both read-only and non-read-only life cycles.

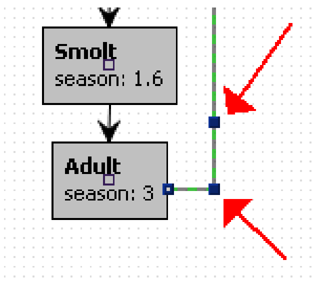
After creating a life cycle, you might wish to change the arrangement of the life cycle diagram view. Layout data is saved within the life cycle.

You can move or change size of life stage rectangles simply by clicking and dragging the mouse; you can also change the position of the transitions and add or delete ‘corners’ in order to make the life cycle graph more readable. For editing transitions, use the following shortcuts:

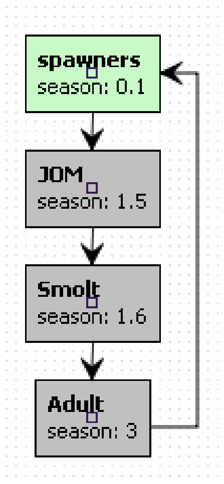
- **adding corners to a transition** : Holding the Shift key down, simultaneously right-click on a transition.
- **removing corners from a transition** : Holding the Shift key down, simultaneously right-click on the corner you want to delete. (When you depress Shift and move the mouse over an existing corner, a plus-shaped cursor will appear to tell you that you're over the corner.)
- **positioning** : Left-click a corner and drag it.



pre–layout



adding/removing corners to transitions



post–layout

SLAM Scenarios

In SLAM, a scenario is a collection of simulation parameters that pertain to a life cycle. The life cycle defines the structure of a life-history model and a scenario for that life cycle specifies a set of initialization constraints and functional relationships that describe the individual elements of the model and how they interact during a simulation run.

For example, the value of a life stage designated as an initialization stage can be initialized at the beginning of a simulation trajectory by a SLAM parameter that is evaluated as a constant, as a succession of data points read from an input file, or as draws from a random distribution. For the same life cycle, simulations run under different scenarios might initialize the same stage in any of these ways. Similarly, the functional relationships determining survival ratios across transitions can be configured in a variety of ways by simulating the life cycle under different scenarios.

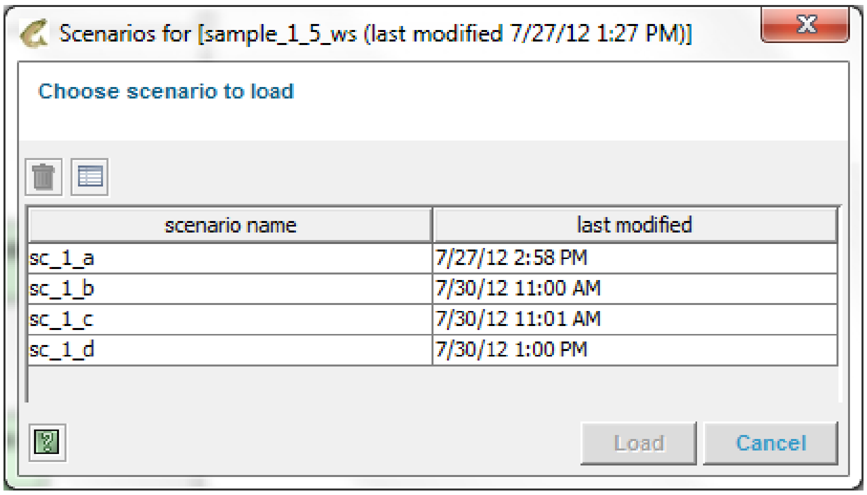
- Creating scenarios
- Loading scenarios
- Saving scenarios
- Deleting scenarios

Creating scenarios

You must have a life cycle open in order to create a scenario for it. Create a new scenario by selecting the Scenario/new scenario menu item, by typing Ctrl+Alt+N, or by selecting the scenario tab and clicking the new scenario button in the tool bar. The new scenario will be loaded into the scenario editor.

Loading scenarios

With a life cycle open, load an existing scenario by selecting the load scenario item in the Scenario menu, by typing Ctrl+Alt+O, or by selecting the scenario tab and then clicking the load scenario button in the tool bar. The 'Choose scenario to load' dialog will be displayed.



Scenarios for [sample_1_5_ws (last modified 7/27/12 1:27 PM)]

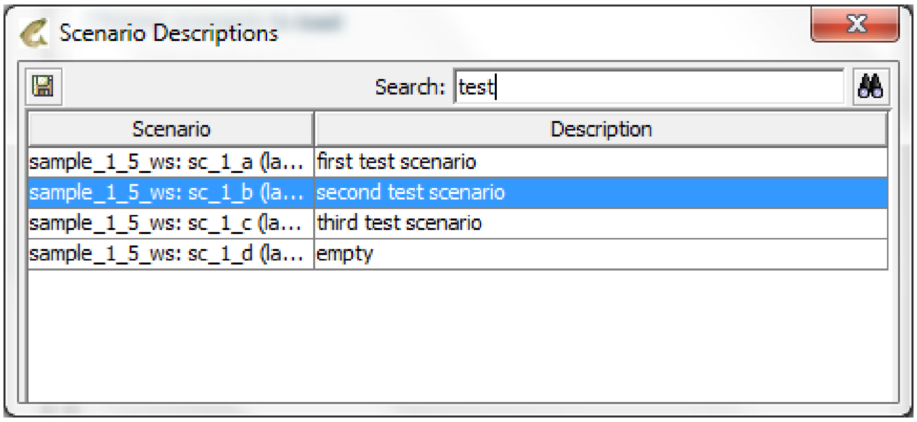
Choose scenario to load

scenario name	last modified
sc_1_a	7/27/12 2:58 PM
sc_1_b	7/30/12 11:00 AM
sc_1_c	7/30/12 11:01 AM
sc_1_d	7/30/12 1:00 PM

Scenario loading dialog

The names and last-modified dates are shown for all scenarios that have been saved to the life cycle. Select one and click the 'Load' button, or double-click the desired scenario. The scenario will be loaded into the scenario editor.

You can also search scenarios by their descriptions: in the scenario-loading dialog, click the 'Description' button above the scenario list. The 'Descriptions' dialog will appear.



Scenario	Description
sample_1_5_ws: sc_1_a (la...	first test scenario
sample_1_5_ws: sc_1_b (la...	second test scenario
sample_1_5_ws: sc_1_c (la...	third test scenario
sample_1_5_ws: sc_1_d (la...	empty

Scenario description-search dialog

This displays a list of scenario names and the descriptions of the corresponding scenarios.

Type a search term into the 'Search' field and click the search button . The first scenario with a description that contains the search term will be highlighted in the list. Continue to click the search button and the highlight will move to successive matching scenarios, finally wrapping back to the first match. If no matches are found, no scenarios will be highlighted.

You can also save a list of scenarios and their descriptions to a text file by clicking the save button above the scenario list.

Saving scenarios

Save the open scenario under its current name by selecting the save scenario item in the Scenario menu, by typing Ctrl+Alt+S, or by selecting the scenario tab and then clicking the save scenario button in the tool bar. Save a scenario under a new name (*i.e.*, create a copy of the scenario with a new name) by selecting the save scenario as... item in the Scenario menu, by typing Ctrl+Alt+Shift+S, or by selecting the scenario tab and then clicking the save scenario as button in the tool bar. You will be prompted for the name of the new scenario.

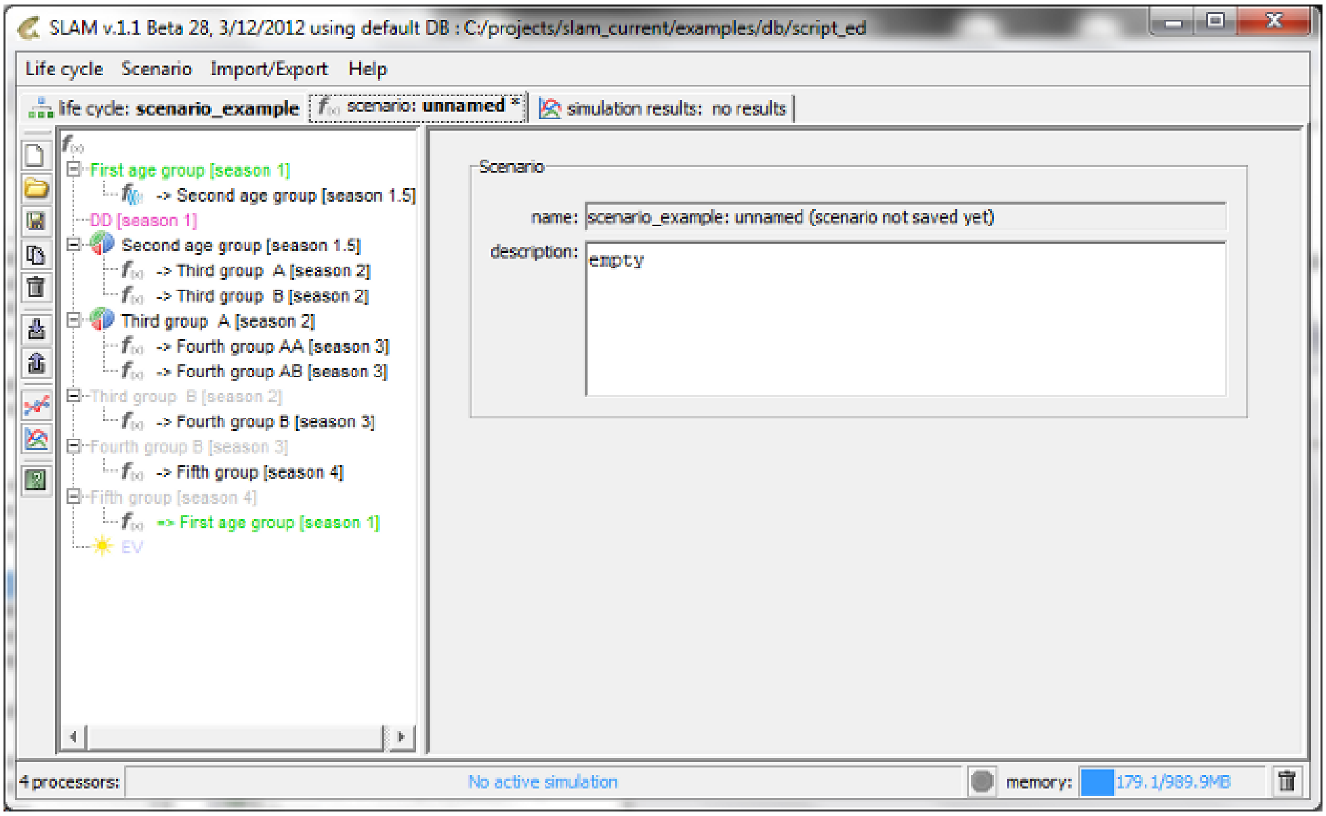
You will be prompted for a name when you save a scenario for the first time or when you save as in order to create a copy. If there is already a scenario saved for the current life cycle with the name you choose, SLAM will prompt you to choose another.

Deleting scenarios

Delete the scenario that is currently open by selecting the delete scenario item in the Scenario menu, by typing Ctrl+Alt+D, or by selecting the scenario tab and then clicking the delete scenario button in the tool bar.

Scenario editor

When you create a new scenario or load an existing one, SLAM opens it in the scenario editor. When a scenario is loaded, you can always display the scenario editor by clicking the scenario tab.


SLAM v.1.1 Beta 28, 3/12/2012 using default DB : C:/projects/slam_current/examples/db/script_ed
Life cycle
Scenario
Import/Export
Help
life cycle: scenario_example
scenario: unnamed *
simulation results: no results
First age group [season 1]
-> Second age group [season 1.5]
DD [season 1]
Second age group [season 1.5]
-> Third group A [season 2]
-> Third group B [season 2]
Third group A [season 2]
-> Fourth group AA [season 3]
-> Fourth group AB [season 3]
Third group B [season 2]
-> Fourth group B [season 3]
Fourth group B [season 3]
-> Fifth group [season 4]
Fifth group [season 4]
=> First age group [season 1]
EV
Scenario
4 processors:
No active simulation
memory:
179.1/989.9MB

name: scenario_example: unnamed (scenario not saved yet)

description: empty

Scenario in the scenario editor

The scenario shown above was created for a life cycle named ‘scenario_example’; the scenario name field reads scenario_example: unnamed (scenario not saved yet) . When you save the scenario for the first time you'll be given the opportunity to name it, and the name will be displayed both in the scenario editor and on the scenario tab. In this example the tab reads scenario: **unnamed*** . The asterisk (*) is there because changes have been made to the scenario which have not been saved.

Below the scenario name field is the description field. If you attach several scenarios to a life cycle, documenting their differences in the description field can save confusion for you or someone else trying to tell them apart. You can also search for a scenario based on words in the description field.

On the left side of the scenario editor is a tree-structured list of the life cycle elements. These are ordered by seasons, with stages and dynamic drivers first, followed by environment variables at the bottom (since environment variables don't have seasons). For large, real-life models this list can be quite long. You can control the display of segments of the list by clicking the +- boxes along the left side. If the list display exceeds the height of the display, scroll bars appear.

The way in which a life cycle element is displayed depends on the element's type.

- Life stages and dynamic drivers are listed as *displayname [season n.nn]*

where displayname is the element's name and n.nn is the season.

- ♦ Initialized life cycle stages (*First age group*) are listed in green.
- ♦ Uninitialized, non-splitting stages (*Third group B*) are displayed in gray.
- ♦ Uninitialized, splitting stages are listed in black with a pie-chart icon (*Second age group*).

- Dynamic drivers (*DD*) are listed in purple.
- Environment variables appear in gray with a sunburst icon (*EV*).

Transitions from life cycle stages are represented by branches below the nodes representing the stages in the scenario editor tree list. Different types of transitions are displayed in different ways:

- Non-reproductive, non-influenced transitions are displayed in black preceded by the function icon: .
  (Example: *Third group B -> Fourth group B*)
- Non-reproductive, influenced transitions are displayed in black preceded by the influence-function icon .
  (Example: *First age group -> Second age group*)
- Reproductive, non-influenced transitions are displayed in green, preceded by the function icon: .
  (Example: *Fifth group -> First age group*)
- Reproductive, influenced transitions are displayed in green preceded by the influence-function icon .
  (*Not shown*)

Configuring elements in the scenario editor

In SLAM, you can configure how life cycle elements (stages, environment variables, and dynamic drivers) behave during a simulation by specifying how their values are calculated at the beginning of each trajectory and how the values change in response to the values of other elements in the course of a trajectory. You do this in the scenario editor, in which you can edit available configuration options based on the type of the element and on element characteristics you specified in the life cycle model.

Element types

- Stages
- Initialized stages
- Splitting stages
- Environment variables
- Dynamic drivers

You can also use the scenario editor to configure transitions.

Configuring life cycle stages in the scenario editor

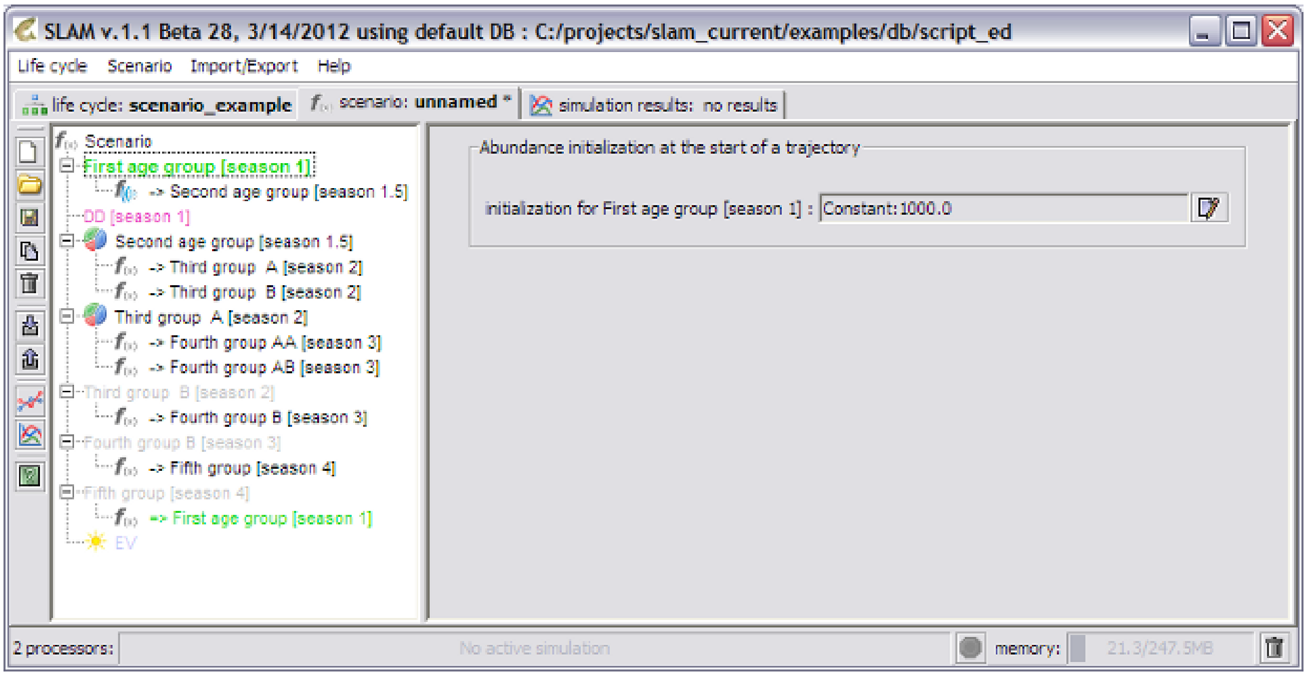
The options available for configuring life cycle stages in a scenario depends on how the stages were defined in the life cycle model. You edit a stage configuration by opening the scenario editor and clicking on the editor tree list item that corresponds to the stage.

A stage that is not initialized, self-influenced, or splitting has no configuration options. As you modify the definition of the stage, configuration options become available as described below. (Recall that you can only change stage definitions in a life cycle that is not read-only.)

- Initialized stages
- Splitting stages

Initialized stages:

If the Initialize abundance at start of trajectory check box is selected for a stage, then you can configure the way in which the stage is initialized at the beginning of each trajectory in a simulation run.


SLAM v.1.1 Beta 28, 3/14/2012 using default DB : C:/projects/slam_current/examples/db/script_ed
Life cycle
Scenario
Import/Export
Help
life cycle: scenario_example
scenario: unnamed *
simulation results: no results
Scenario
First age group [season 1]
-> Second age group [season 1.5]
DD [season 1]
Second age group [season 1.5]
-> Third group A [season 2]
-> Third group B [season 2]
Third group A [season 2]
-> Fourth group AA [season 3]
-> Fourth group AB [season 3]
Third group B [season 2]
-> Fourth group B [season 3]
Fourth group B [season 3]
-> Fifth group [season 4]
Fifth group [season 4]
=> First age group [season 1]
EV
2 processors:
No active simulation
memory:
21.3/247.5MB

Abundance initialization at the start of a trajectory

initialization for First age group [season 1] : Constant: 1000.0

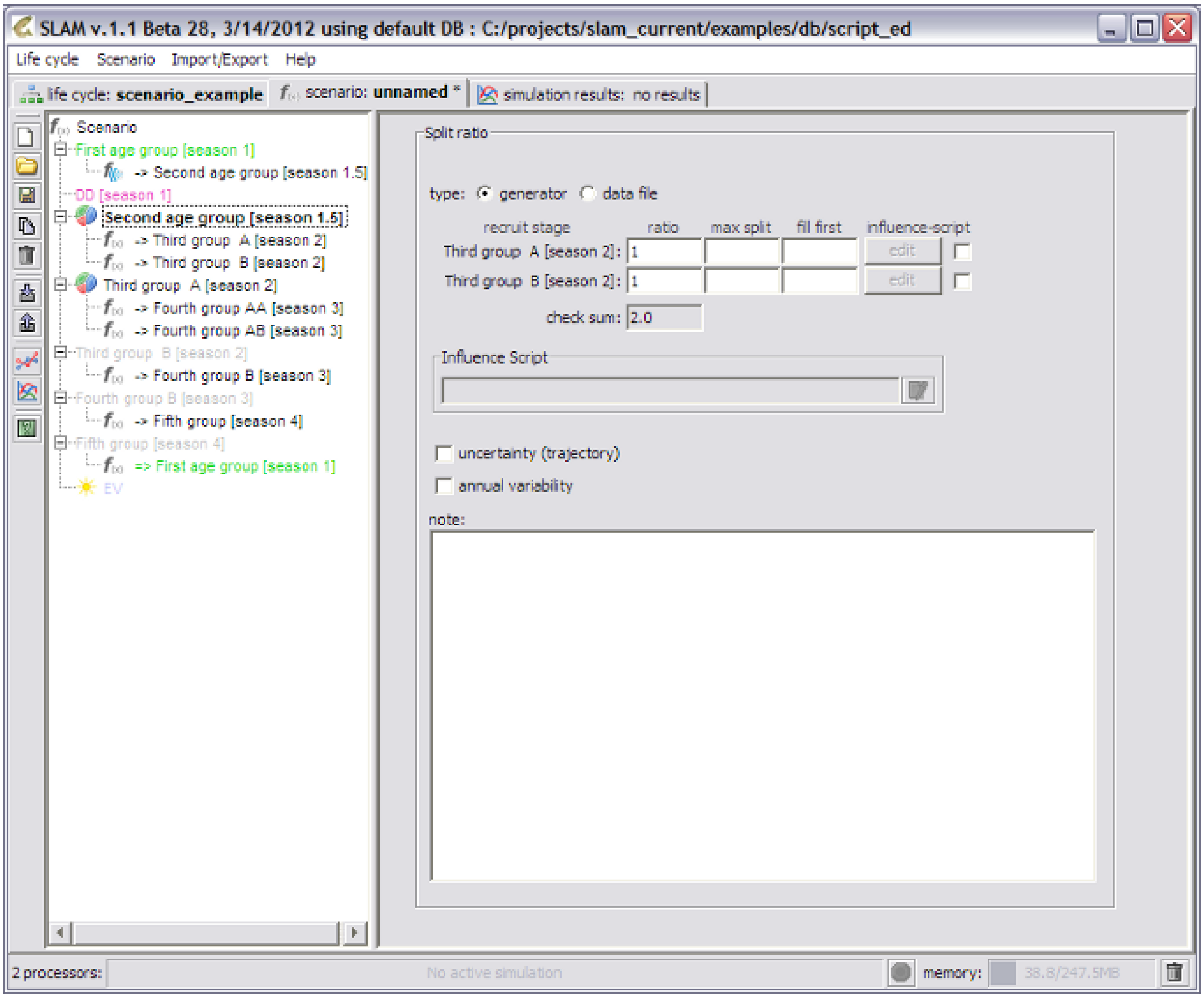
Stage initialization configuration

The initialization value of a stage is a SLAM parameter. Click the edit icon at the right side of the initilization control to open the parameter editor in which you can configure the type and details of the stage's initialization.

Splitting stages:

A stage with more than one recruit stage is a splitting stage. In the scenario editor's split configuration interface, you can configure the way in which SLAM divides the abundance of the stage among its recruits.

The appearance of the split editor depends mostly on whether you want SLAM to calculate split ratios or read them in from data files. By default, split ratios are calculated, and the split editor interface looks like this:


SLAM v.1.1 Beta 28, 3/14/2012 using default DB : C:/projects/slam_current/examples/db/script_ed
Life cycle
Scenario
Import/Export
Help
life cycle: scenario_example
scenario: unnamed *
simulation results: no results
Split ratio
Influence Script
note:
edit
edit
2 processors:
No active simulation
memory:
38.8/247.5MB

Scenario
- First age group [season 1]
  - -> Second age group [season 1.5]
- DD [season 1]
- Second age group [season 1.5]
  - -> Third group A [season 2]
  - -> Third group B [season 2]
- Third group A [season 2]
  - -> Fourth group AA [season 3]
  - -> Fourth group AB [season 3]
- Third group B [season 2]
  - -> Fourth group B [season 3]
- Fourth group B [season 3]
  - -> Fifth group [season 4]
- Fifth group [season 4]
  - => First age group [season 1]
- EV

type: [✓ generator] data file

recruit stage	ratio	max split	fill first	influence-script	
Third group A [season 2]:	1			edit	☐
Third group B [season 2]:	1			edit	☐

check sum: 2.0

☐ uncertainty (trajectory)

☐ annual variability

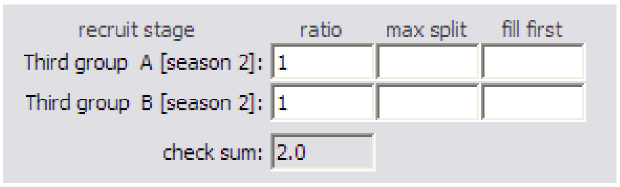
Configuring calculated split ratios

Details

- **type:** Use the type buttons to choose between split ratios that are computed by SLAM during a simulation run and those read in from data files.

influenced and non-influenced splits

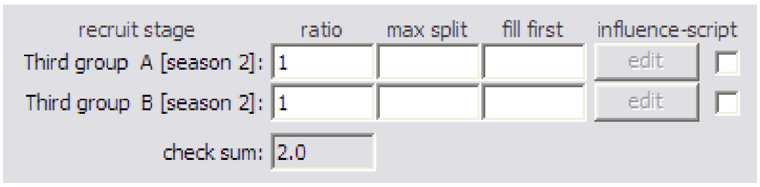
In the life cycle model used as an example, both splitting stages are influenced. If no influences were present, the ratio section of the configuration interface would omit the influence-script column:



recruit stage	ratio	max split	fill first
Third group A [season 2]:	1		
Third group B [season 2]:	1		
check sum:	2.0		

Editing ratios without influences

As the influence-script column is the only difference between the non-influenced and influenced editor interfaces



recruit stage	ratio	max split	fill first	influence-script	
Third group A [season 2]:	1			edit	☐
Third group B [season 2]:	1			edit	☐
check sum:	2.0				

Editing ratios with influences

you can skip that part of the description below if you're only interested in non-influenced splits.

- **ratio configuration:** SLAM's split ratio calculations are configured in the ratio section of the editor.
  - **recruit stage column:** This column contains the name of each recruit stage. Configuration parameters for this branch of the split are contained in the stage's row.
  - **ratio column:** These are values that determine the split ratio (if influence scripts are not enabled for the recruit stage — see below). Only the ratio(s) of the numbers you select are important, as they will be normalized by their total (shown in the check sum field below the bottom row) before being applied to split calculations.

    In the example, a ratio value of 3 for *Third group A* and a value of 2 for *Third group B* will result in 60% of the stock stage's abundance being transferred to the A recruit stage and the remaining 40% going to stage B, subject to subsequent processing. Ratio values of 0.6 and 0.4 would produce the same effect.

    Split ratio values default to one, producing an even distribution of the stock abundance over the recruit stages.
  - **max split column:** These are upper limits you can place on the abundance assigned to the recruit stage. Max split limits only have an effect if the stock stage abundance times the split ratio for the max-limited recruit stage produce a number greater than the limit. This constraint may override the designated split ratios.

    For a stock abundance of 100 and a 60-40 split between recruit stages A and B, 60 fish would go to stage A and 40 to stage B in the absence of limits. If stage A is limited to 70 fish, then the 60-40 recruit count is unchanged. If stage A is limited to 55 fish, then the resulting recruit abundances will be 55 in stage A and 45 in stage B. Note that all of the remaining abundance is allocated to stage B, even though the new stage B total, 45, is greater than the abundance of 40 that would have resulted from a direct application of the split ratios.

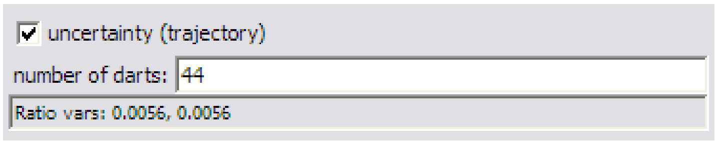
    Do not assign max split values to more than one recruit stage per split.

    - **fill first column:** This value is the abundance that will be allocated to the recruit stage before anything is assigned to other recruit stages in this split. If the stock abundance is less than the recruit stage's first fill value, then the entire stock goes to the recruit stage. This constraint may override the designated split ratios.

      For 60-40 split ratio and a stock abundance of 100, assigning a first fill of 35 to stage B does not change the split outcome. If stage B has a first fill value of 55, though, the split outcomes will be 45 for stage A and 55 for stage B. If stage B has a first fill value of 100 or more, then stage A will end up with an abundance of zero.

      Do not assign first fill values to more than one recruit stage per split.
    - **influence-script column:** This column contains controls that allow you to enable and configure influence scripts if the split is either externally or self-influenced. To enable an influence script for a stage, select (check) the check box for that stage. To edit the stage's script in the script editor, click the edit button.
- **Influence Script:** This field displays the description of the most recently edited influence script for the split. You can open the script editor for this script by clicking the edit icon at the right side of the field. (To avoid confusion, it is advisable to use the edit button corresponding to the recruit stage in order to be certain you are editing the intended script.)
- **Split ratio variability:** You can configure split ratios so that they are randomized about the constant values described above on a trajectory-by-trajectory and/or a year-by-year basis. The random split ratio values are drawn from a Dirichlet random number generator implemented in the Stochastic Simulation in Java (SSJ) package.
    - **uncertainty (trajectory):** If this box is checked, split ratios will be set at the beginning of each trajectory to new, random values which they will maintain throughout the trajectory.

      When the uncertainty checkbox is selected, the uncertainty configuration interface is visible:



☑ uncertainty (trajectory)

number of darts: 44

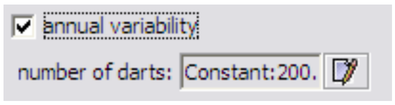
Ratio vars: 0.0056, 0.0056

Trajectory uncertainty configuration

      The number of darts parameter affects the degree to which the random ratios vary about the constant values you designate as described above: the larger the number, the less the variability.

      When you specify the number of darts, SLAM calculates the variances of the random split ratios and displays them below the number of darts field. For two-way splits, the variances of the two ratios will be equal, but for more than two split ratios the variance depends on both how many darts you specified and on the relative sizes of the constant ratio values.
    - **annual variability:** If this box is checked, split ratios will be set to new, random values centered around the constant values described above. Each time (that is, for each season) that split abundances are computed, new annual variabilities are also calculated and applied.

      Selecting the variability checkbox displays the variability configuration interface:



☑ annual variability

number of darts: Constant:200.

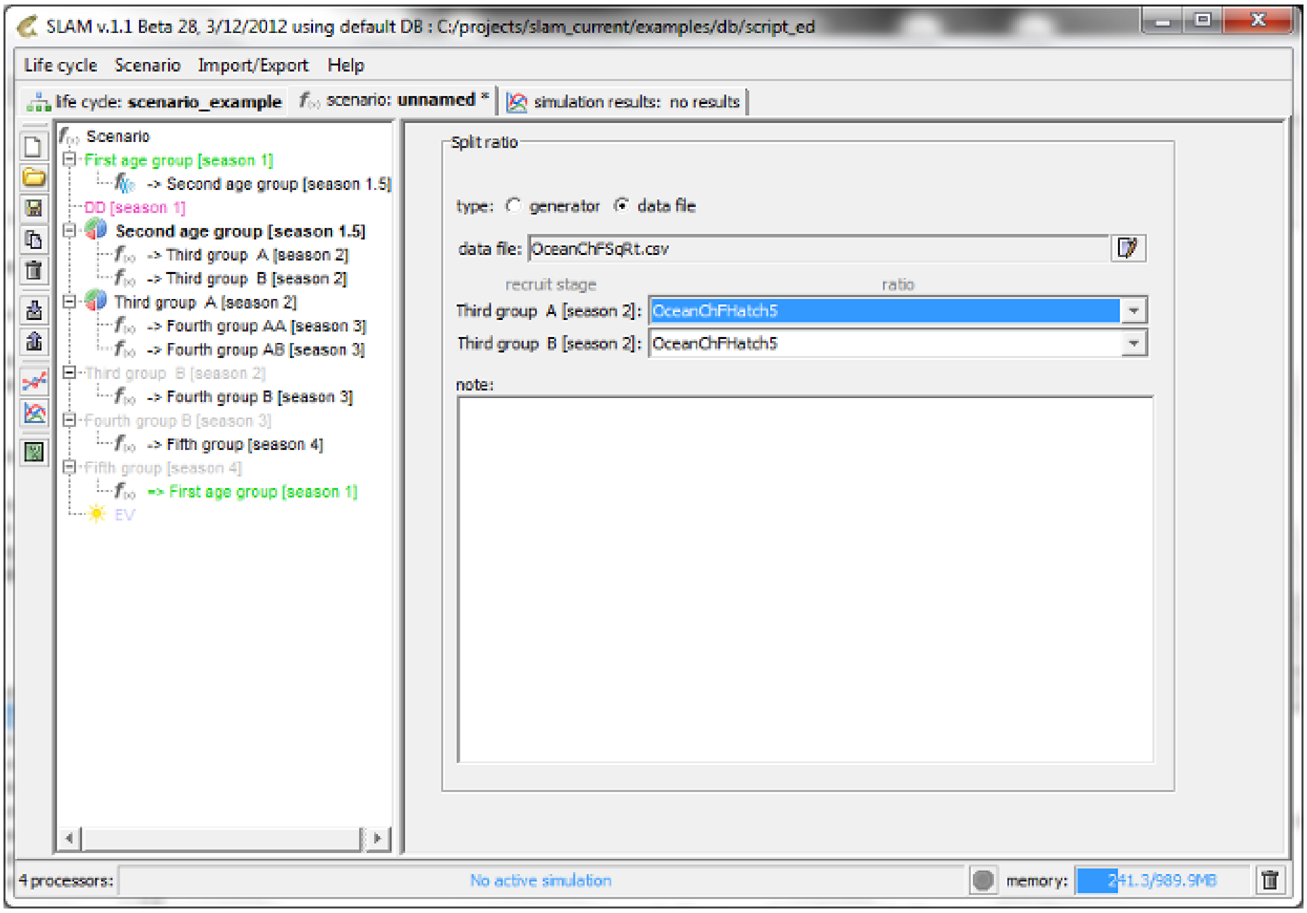
Annual variability configuration

As for trajectory uncertainty, the number of darts parameter determines the variance of the annual split ratios. In this case, the dart-number can be specified as a constant, a draw from a random distribution, or a value read from a data file, like other SLAM parameters. Click the edit icon at the right end of the control to configure the number of darts.

- **note:**This field is for your descriptions or comments about the split configuration.

Ratios from data files

If the 'data files' split ratio type is selected, the interface looks like this:


SLAM v.1.1 Beta 28, 3/12/2012 using default DB : C:/projects/slam_current/examples/db/script_ed
Life cycle
Scenario
Import/Export
Help
life cycle: scenario_example
scenario: unnamed *
simulation results: no results
Scenario
First age group [season 1]
-> Second age group [season 1.5]
DD [season 1]
Second age group [season 1.5]
-> Third group A [season 2]
-> Third group B [season 2]
Third group A [season 2]
-> Fourth group AA [season 3]
-> Fourth group AB [season 3]
Third group B [season 2]
-> Fourth group B [season 3]
Fourth group B [season 3]
-> Fifth group [season 4]
Fifth group [season 4]
=> First age group [season 1]
EV
4 processors:
No active simulation
memory: 241.3/989.9MB

Split ratio

type: ( ) generator (•) data file

data file: OceanChFSqRt.csv

recruit stage	ratio
Third group A [season 2]:	OceanChFHatch5
Third group B [season 2]:	OceanChFHatch5

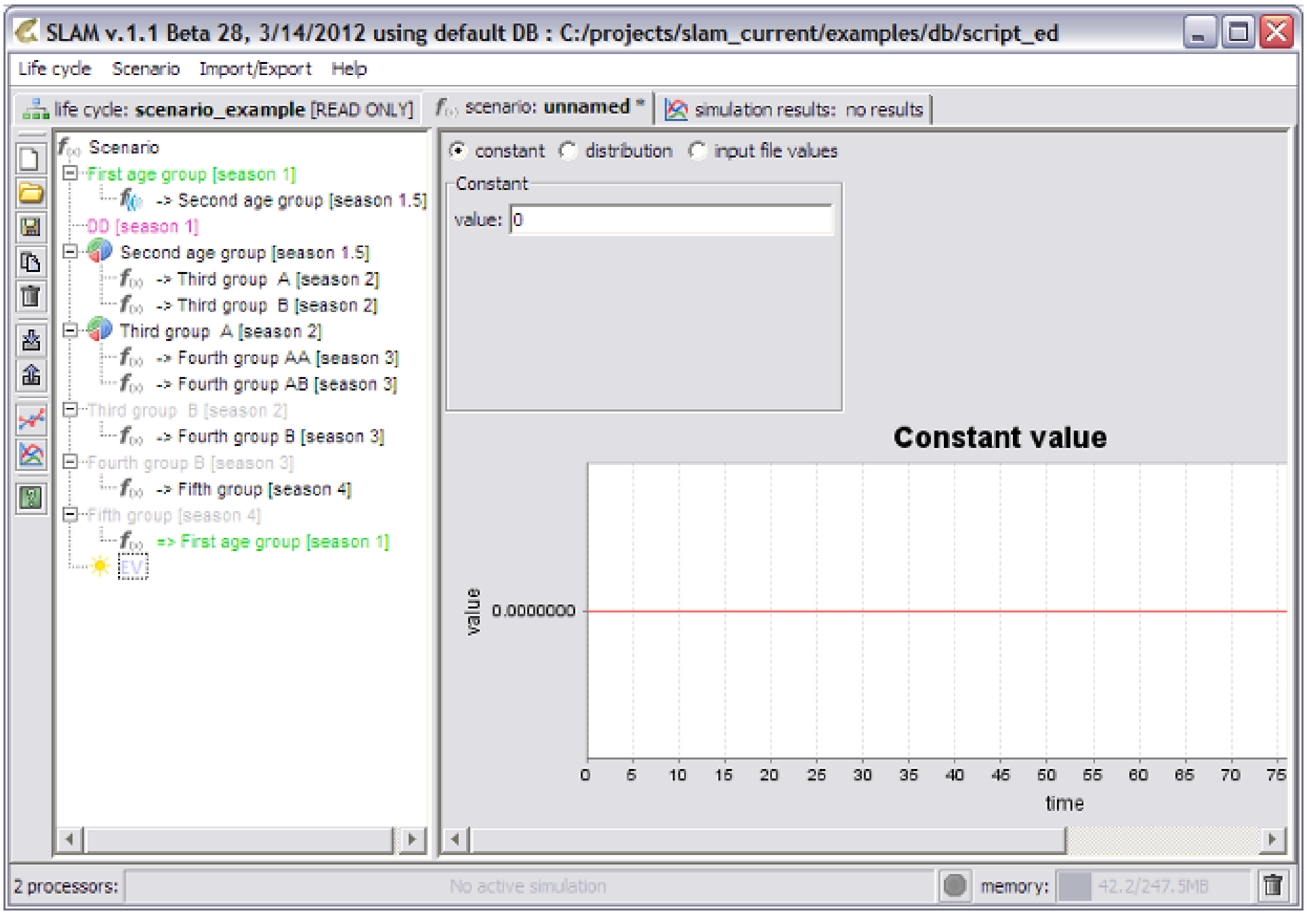
note:

Data file split

Select the file containing ratio values by clicking the browse button . When the file is loaded, you can select a series containing the ratio values for each of the recruit stages in the split by using the drop-down lists in the 'ratio' column.

Configuring environment variables in the scenario editor

Environment variables are implemented as SLAM parameters and can be assigned constant values, values drawn from random distributions, or values read from data files. When you select an environment variable in the scenario editor element tree, the SLAM parameter configuration panel appears:



SLAM v.1.1 Beta 28, 3/14/2012 using default DB : C:/projects/slam_current/examples/db/script_ed

life cycle: **scenario_example** [READ ONLY] scenario: **unnamed** *

Scenario
- First age group [season 1]
  - -> Second age group [season 1.5]
- DD [season 1]
- Second age group [season 1.5]
  - -> Third group A [season 2]
  - -> Third group B [season 2]
- Third group A [season 2]
  - -> Fourth group AA [season 3]
  - -> Fourth group AB [season 3]
- Third group B [season 2]
  - -> Fourth group B [season 3]
- Fourth group B [season 3]
  - -> Fifth group [season 4]
- Fifth group [season 4]
  - => First age group [season 1]
- EV

[✓ constant] distribution input file values

Constant
value: 0

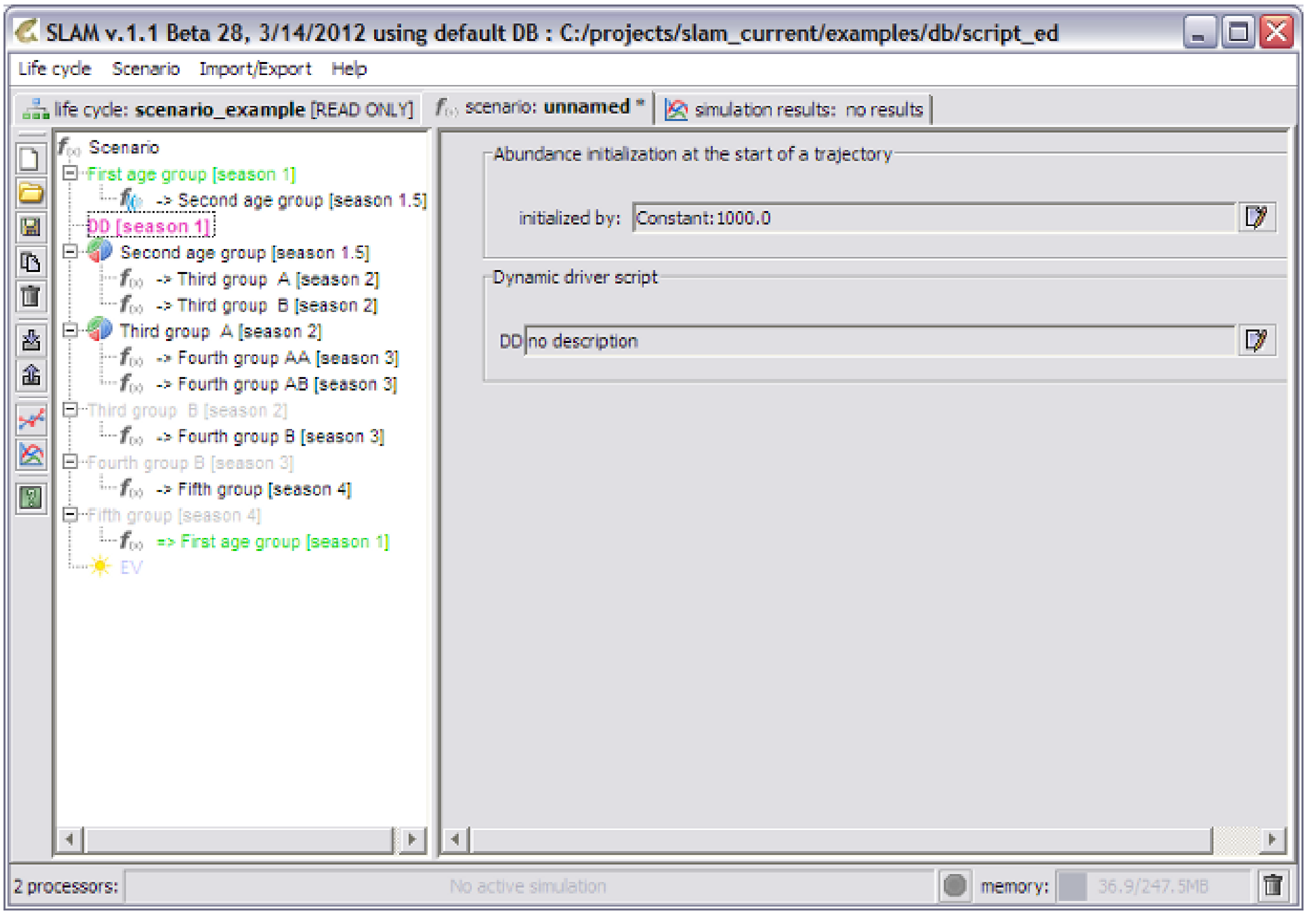
Environment variable configuration

Click the edit icon to open the parameter editor.

For more details, see SLAM parameters.

Configuring dynamic drivers in the scenario editor

The behavior of dynamic drivers is determined by how they are initialized and by the scripts that define how their values change through the course of a simulation run. You define the intialization method and assign scripts in the scenario editor.


SLAM v.1.1 Beta 28, 3/14/2012 using default DB : C:/projects/slam_current/examples/db/script_ed
Life cycle
Scenario
Import/Export
Help
life cycle: scenario_example [READ ONLY]
scenario: unnamed *
simulation results: no results
2 processors:
No active simulation
memory:
36.9/247.5MB

Scenario
First age group [season 1]
-> Second age group [season 1.5]
DD [season 1]
Second age group [season 1.5]
-> Third group A [season 2]
-> Third group B [season 2]
Third group A [season 2]
-> Fourth group AA [season 3]
-> Fourth group AB [season 3]
Third group B [season 2]
-> Fourth group B [season 3]
Fourth group B [season 3]
-> Fifth group [season 4]
Fifth group [season 4]
=> First age group [season 1]
EV

Abundance initialization at the start of a trajectory
initialized by: Constant:1000.0

Dynamic driver script
DD no description

Dynamic driver configuration

Initialization

The initialization value of a dynamic driver is a SLAM parameter. Like all parameters, it can be assign a constant value, a value drawn from a random distribution, or a value read from an input file.

Open the initialization configuration editor by clicking the edit icon at the right side of the initialization control. See SLAM parameters for configuration details.

Dynamic driver script

A dynamic driver must have a script associated with it. The script determines how the driver's value changes during the course of a simulation trajectory in response to the values of the life cycle elements that influence the driver. To create, load, or edit a script, click the edit icon at the right side of the script control. When a script is loaded, the field in the Dynamic driver script box will contain the script's description.

The value calculated by a dynamic driver's script is passed to life cycle elements that are influenced by the driver. The value is also stored and is available to the driver's script as a parameter the next time the script is evaluated.

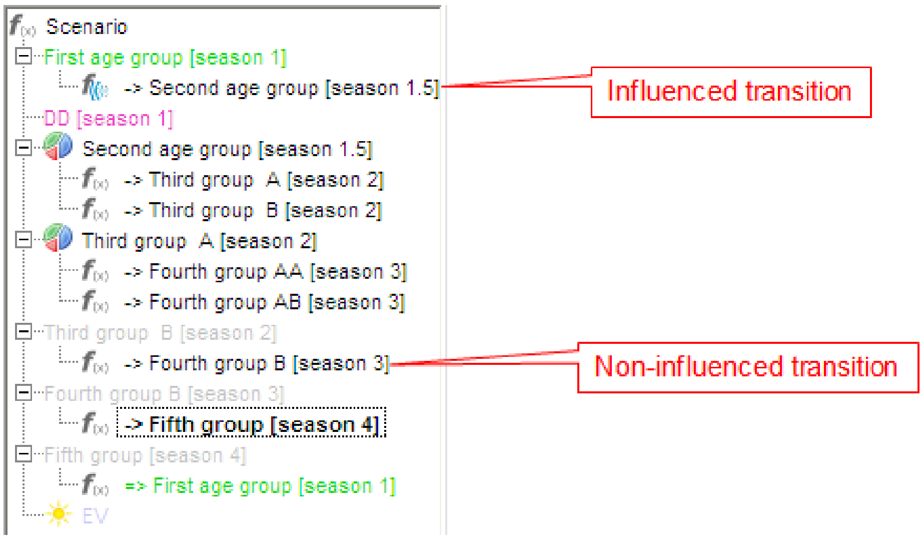
The default script simply returns the current value of the driver.

For more about scripts, see SLAM scripts.

Configuring transitions in the scenario editor

In SLAM, transitions are characterized by functional relationships that determine the abundance of recruit stages based on the parent stage abundance as well as on other model parameters you may have selected. In the scenario editor you can define functional types for transitions and configure the parameters specific to those types. For influenced transitions you can also describe the abundance relationships of transitions by way of scripts.

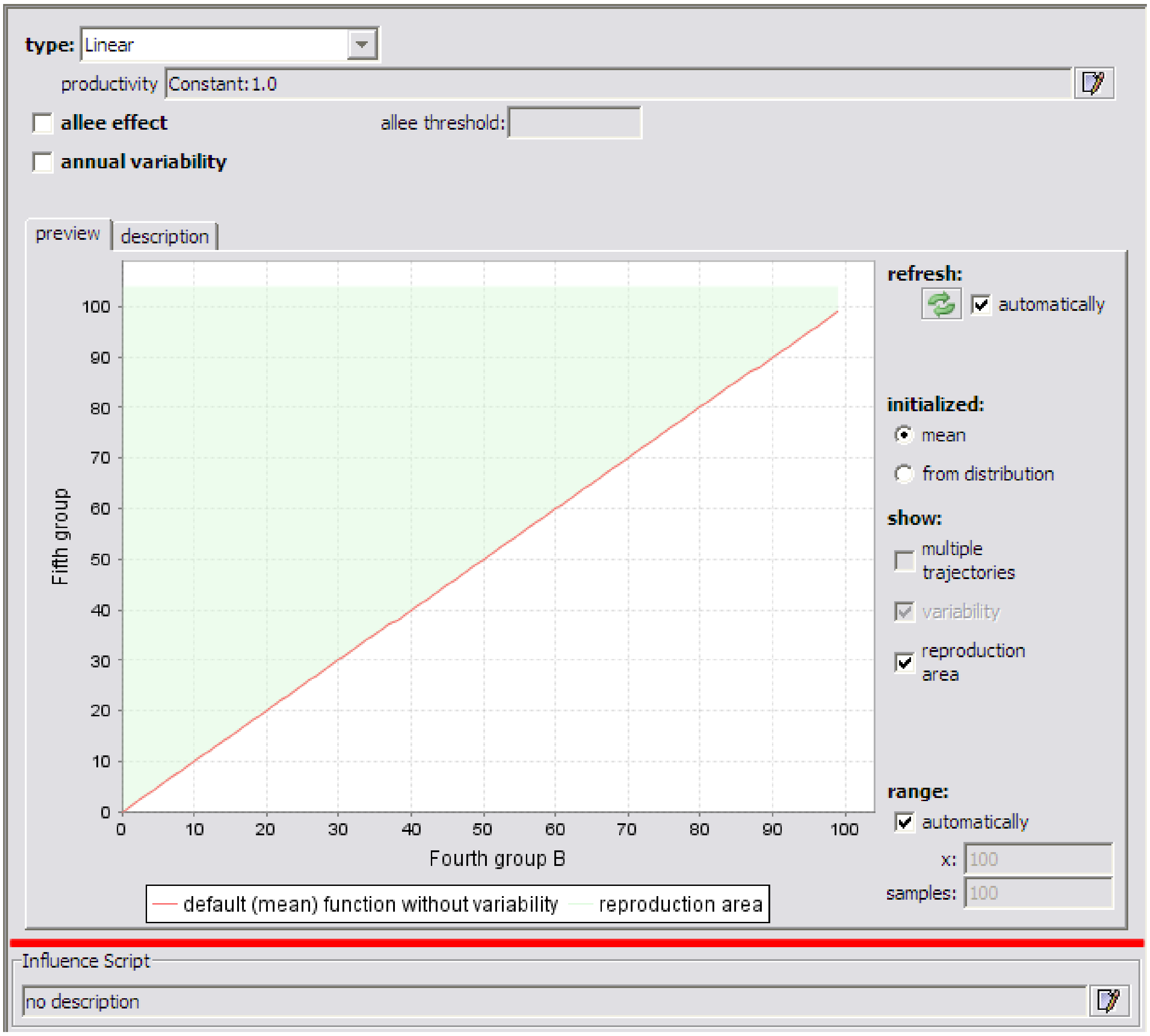
In the scenario editor tree list, transitions from a given stock stage are listed immediately below the stage and are labeled by the names of their recruit stages; if a transition entry isn't visible, expand the list by clicking the + to the left of the transition's stock stage.


Scenario
First age group [season 1]
-> Second age group [season 1.5]
DD [season 1]
Second age group [season 1.5]
-> Third group A [season 2]
-> Third group B [season 2]
Third group A [season 2]
-> Fourth group AA [season 3]
-> Fourth group AB [season 3]
Third group B [season 2]
-> Fourth group B [season 3]
Fourth group B [season 3]
-> Fifth group [season 4]
Fifth group [season 4]
=> First age group [season 1]
EV
Influenced transition
Non-influenced transition

Transitions in the scenario editor tree

In the list, non-influenced transitions are preceded by a f(x) icon and influenced transitions are preceded by the slightly more mysterious icon . Influenced transitions have the same configuration options as non-influenced transitions plus a script editor interface in which you can create, load, and modify influence scripts.

Select a transition in the tree list to open the transition configuration editor.


type: Linear

productivity Constant: 1.0

☐ allee effect   allee threshold:

☐ annual variability

Influence Script

no description

Transition configuration in the scenario editor

Configuration controls that lie above the thick red line (the line is not present in the actual interface) serve to define, preview, and annotate the transition function.

Transition influences

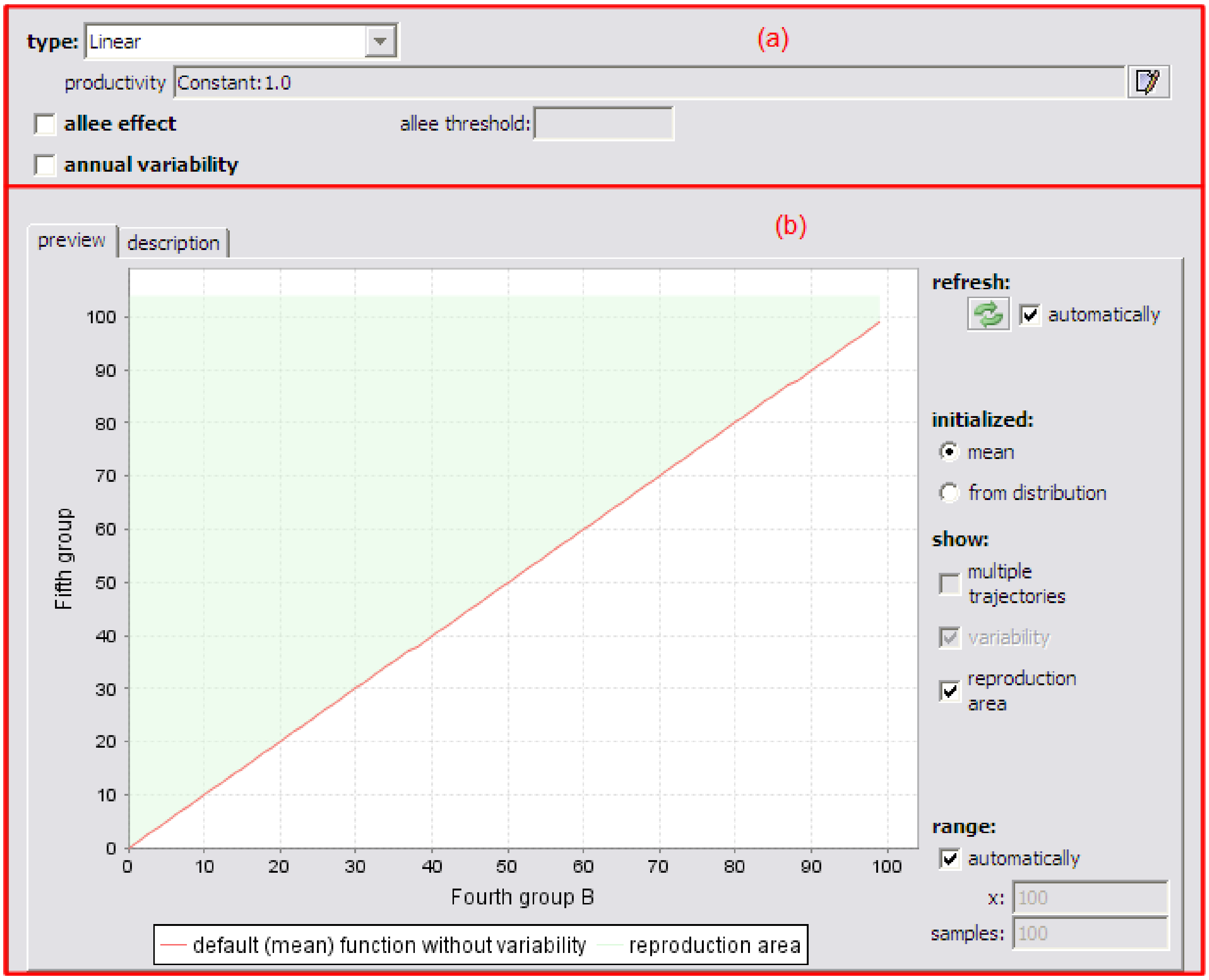
The control shown below the red line provides access to the script editor. *(The script editor control is available only for influenced or self-influenced transitions.)* Instead of using one of SLAM's pre-defined transition functions, you can calculate the abundance of the transition's recruit stage by defining your own relationship in a script.

The text field inside the box labeled Influence script displays the first line of the script's description. If no script has been assigned to the transition, the field contains the text no description as shown. Launch the script editor by clicking the edit icon on the right side of the script control.

Transition functions

Transition functions determine the abundances of recruit stages based on the abundances of their stock stages and on other parameters you have selected. You configure a transition functional relationship in the scenario editor by selecting the transition in the scenario tree editor .

The transition function editor enables you to choose the kind of function that characterizes the transition and to configure its parameters. You can also preview the function to see how it behaves under the parameter values you have assigned to it.



type: Linear

productivity Constant: 1.0

☐ allee effect allee threshold:

☐ annual variability

refresh: ☑ automatically

initialized: [● mean] ○ from distribution

show: ☐ multiple trajectories ☑ variability ☑ reproduction area

range: ☑ automatically

x: 100

samples: 100

Transition function parameter selections

In the upper part of the editor interface (a) you select a function type and configure the parameters that pertain to it.

The lower part of the interface (b) contains two tabs. In one of these, you can preview the transition function, while the other allows you to annotate the transition function configuration.

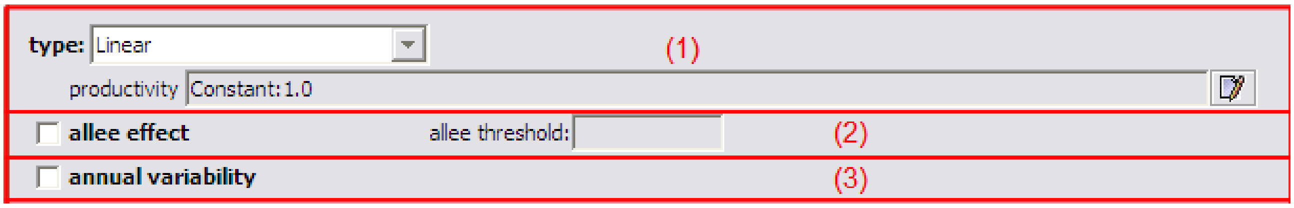
See Previewing transition functions for more information about function previews.

Function description tab: When you select the description tab, the function preview is replaced by a text field (not shown) into which you can type a description, notes, comments, or haiku pertaining to the transition relationship you have defined. This will be stored with the scenario in the SLAM database and will always be available when you edit the scenario.

Configuring transition functions

The upper part of the transition function editor interface contains controls with which you specify what kind of function characterizes the transition and how parameter values for that function should be generated. You can also add Allee effects and variability to transitions.

- Function type configuration
- Allee effect configuration
- Annual variability configuration
  - Bounds and ranges
- Survivorship and reproductive transitions

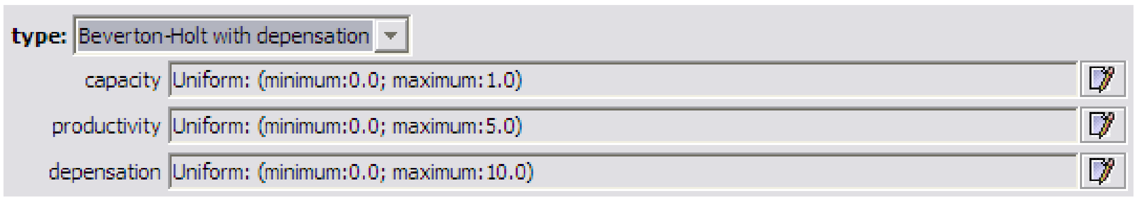

type: Linear (1)
productivity Constant:1.0
allee effect allee threshold: (2)
annual variability (3)

Transition function parameter selections

1. Function type: You can choose one of six function types from the drop-down list:

- Constant
- Linear
- Hockey-stick
- Beverton-Holt
- Beverton-Holt with depensation
- Ricker

Function parameters: Each function has parameters that adjust its behavior. (See types of transition functions for the parameters that apply to each function.) When you select a function type, controls appear beneath the type drop-down list that allow you to configure the parameters. In the picture above, the Linear function type has been selected. Linear functions in SLAM are characterized by a single parameter called productivity , so the productivity control is displayed. The Beverton-Holt with depensation functional type has three parameters: capacity , productivity , and depensation . If you select this type in the drop-down, three fields appear in which you can configure those parameters.


type: Beverton-Holt with depensation
capacity Uniform: (minimum:0.0; maximum:1.0)
productivity Uniform: (minimum:0.0; maximum:5.0)
depensation Uniform: (minimum:0.0; maximum:10.0)

Configuration controls for Beverton-Holt with depensation function type

To edit each parameter, click the edit icon at the right side of the parameter's configuration control. Like other SLAM parameters, functional parameters can be set to constants, to random draws from probability distributions, or to values read from data files.

When you select a function type and modify its parameters, the function preview will change to reflect your choices.

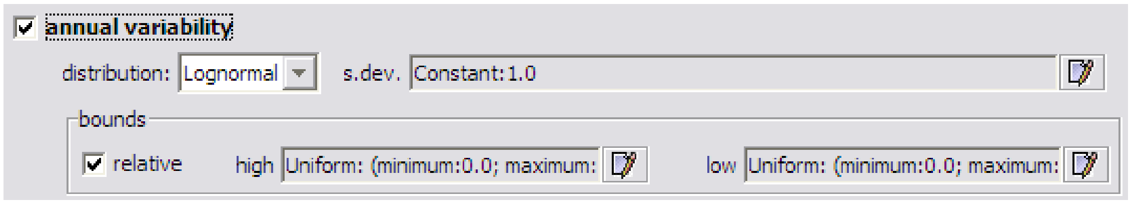
2. Allee effect: If you check the "Allee effect" checkbox, you can enter a population threshold into the "Allee threshold" field. During a simulation, if the parent stage abundance is less than or equal to the threshold, the recruit abundance will be set to zero.

When you enable an Allee threshold the function preview will change to reflect your choice.

3. Annual variability: With this option selected, SLAM introduces random annual variability into its calculations of recruit abundances as follows:

1. The transition function value is calculated as it would be in the absence of variability.
2. Upper and lower bounds on the possible range of values are calculated.
3. A random distribution of a type you specify is parameterized based on the value calculated in step 1 and on the upper and lower bounds calculated in step 2.
4. The recruit abundance that will actually be used is drawn from the distribution defined in step 3.
5. The "variability" (that is, the difference between the unmodified and the modified abundance) is calculated and cached for the current transition and current season. Any future references SLAM makes to this transition in this season (computing recruit abundances in same value groups, for example) will use the cached variability.

When you check the "annual variability" checkbox, the configuration display changes so that you can configure variability parameters.


☑ annual variability
distribution: Lognormal s.dev. Constant:1.0
bounds
☑ relative high Uniform: (minimum:0.0; maximum: low Uniform: (minimum:0.0; maximum:

Annual variability configuration

(Below, "y" represents the value that the transition function produces in the absence of variability, as described in step 1 of the procedure above.)

- **distribution:** You can select the type of random distribution from which the variability will be generated. (See below for an explanation of the ranges over which the distributions are sampled.)
  - **Uniform:** The recruit abundance is drawn from a uniform distribution over the range (r_l,r_u).
  - **Triangular:** The recruit abundance is drawn from a triangular distribution over the range (r_l,r_u) with a peak value of y.
  - **Normal:** The recruit abundance is drawn from a truncated normal distribution over the range (r_l,r_u) with mean value y and standard deviation defined in the "s.dev." field.
  - **Lognormal:** The recruit abundance is drawn from a truncated log-normal distribution over the range (r_l,r_u) with mean value y and standard deviation defined in the "s.dev." field.
- **s.dev.:** This field is visible only when the "Normal" or "Lognormal" distribution type is selected. This parameter is used to calculate the standard deviation of the distribution from which variability is generated. It is a SLAM parameter and can be assigned a constant value, a value drawn from a random distribution, or a value read from a data file by clicking the edit icon 📝 at the right side of

the control.

- **bounds:**
  - ♦ **relative:** When this checkbox is selected, bounds will be calculated as scale factors based on the value y; these are relative bounds. When the checkbox is not selected, the bounds are absolute and are applied as additive increments and decrements to the value y.
  - ♦ The **high** and **low** parameters are the basis for the calculation of upper and lower variability bounds. They are SLAM parameters and can be configured as constants, draws from random distributions, or values read from data files by clicking the edit icon at the right side of their respective controls.

How bounds and ranges are calculated

The parameters that define the distributions from which variability values are drawn depend on the type of distribution, the current unmodified value y of the transition function, the high and low bounds parameters, and the relative bounds flag. The upper and lower ends of the ranges, r_u and r_l are calculated as follows:
For absolute bounds,

$$r_l = y - b_l$$

$$r_u = y + b_u$$

For relative bounds,

$$r_l = (1 - b_l)y$$

$$r_u = (1 + b_u)y$$

where b_u and b_l are values of the *high* and *low* parameters described above.

New range bounds are generated at the beginning of each trajectory during the simulation.

Survivorship and reproductive transitions

The recruit abundance calculated by a non-reproductive transition function can be any value from zero up to the abundance of the parent stage (or a fraction of the parent stage, in the case of pooled groups). If you define a function that would produce more survivors than the stock stage's abundance, SLAM will assign the stock stage abundance to the recruit stage rather than the calculated value.

For example, a linear relationship with a productivity of one will produce one survivor per stock stage individual. If you raise the productivity to ten — asking for ten survivors per stock individual — SLAM will still only assign as many recruit individuals as there are in the stock abundance. You can see this in action in the function preview.

Reproductive transitions are not subject to this limit. From the example above, if you assign a linear function with a productivity of ten to a reproductive transition, SLAM will assign ten individuals to the recruit stage for each individual in the stock stage.

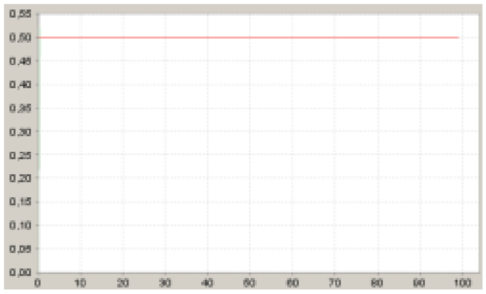
Types of transition functions

There are six functional forms available to use as transition relationships in SLAM. Each has its own set of configuration parameters.

Constant

$$f(x)=c$$

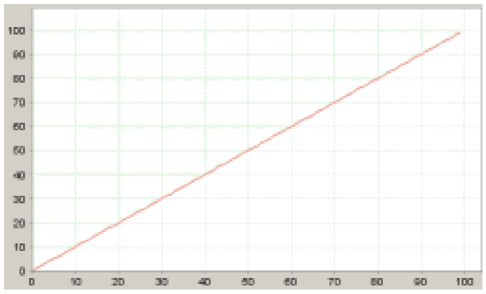
- c : capacity



Linear

$$f(x)=px$$

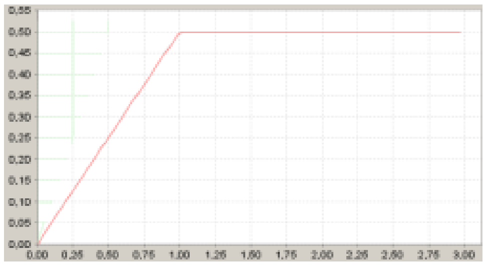
- p : productivity



Hockey-stick

$$f(x)=max(px,c)$$

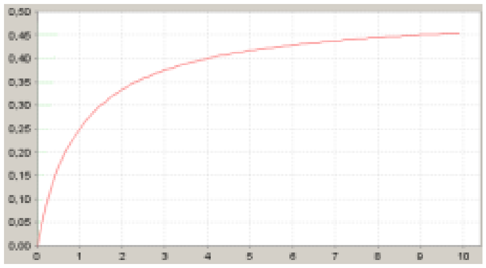
- c : capacity
- p : productivity



Beverton-Holt

$$f(x)=\frac{px}{1+\frac{px}{c}}$$

- c : capacity
- p : productivity

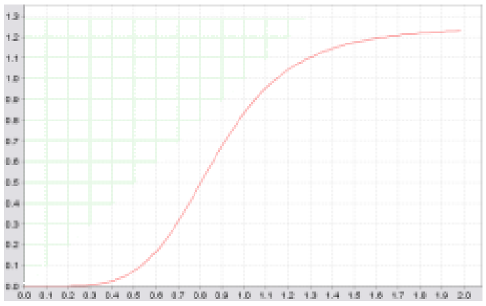


Beverton-Holt with depensation

$$f(x)=\frac{px^d}{1+\frac{x^d}{c}}$$

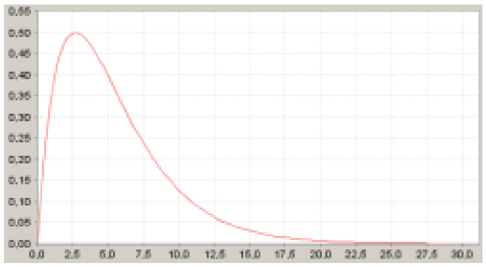
- c : capacity
- p : productivity

- d : depensation



Ricker

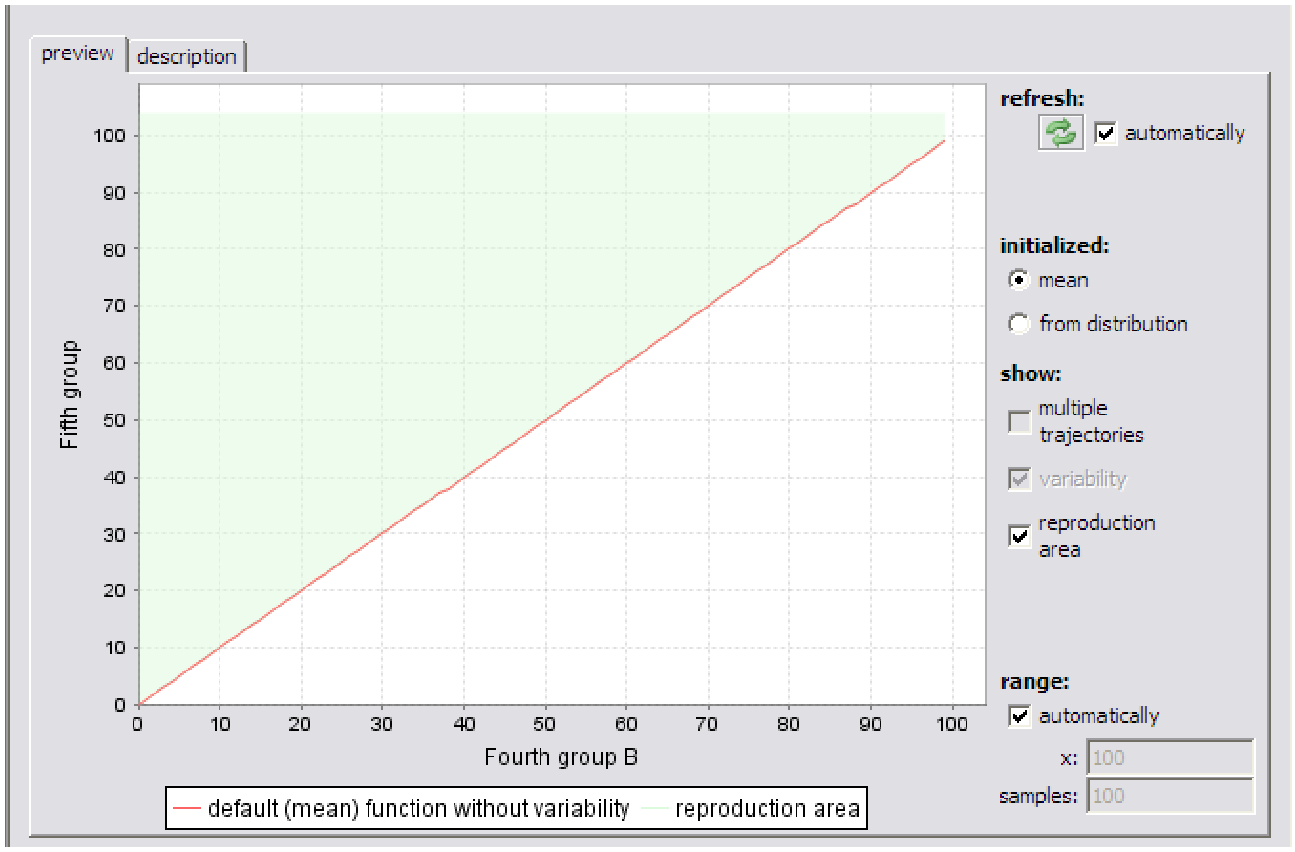
$f(x) = px\, e^{\frac{-px}{ec}}$



- c : capacity
- p : productivity

Previewing transition functions

Configuring a transition function can be a complicated undertaking with a great many knobs to twiddle. The function preview, visible when the preview tab is selected in the transition configuration editor, provides a graphical depiction of the transition function as you define it, allowing you to assess your configuration choices before you run a simulation.



refresh: [✓ automatically]

initialized: [✓ mean] from distribution

show: ☐ multiple trajectories; ✓ variability; ✓ reproduction area

range: [✓ automatically]

x: 100

samples: 100

Transition function preview

The transition function is plotted with the abundance of the stock stage — Fourth group B in the picture above — on the x-axis and the recruit stage abundance — Fifth group — on the y-axis. The recruit-vs-stock abundance relationship is represented by a red line. (A legend describing the graph appears below the x-axis.)

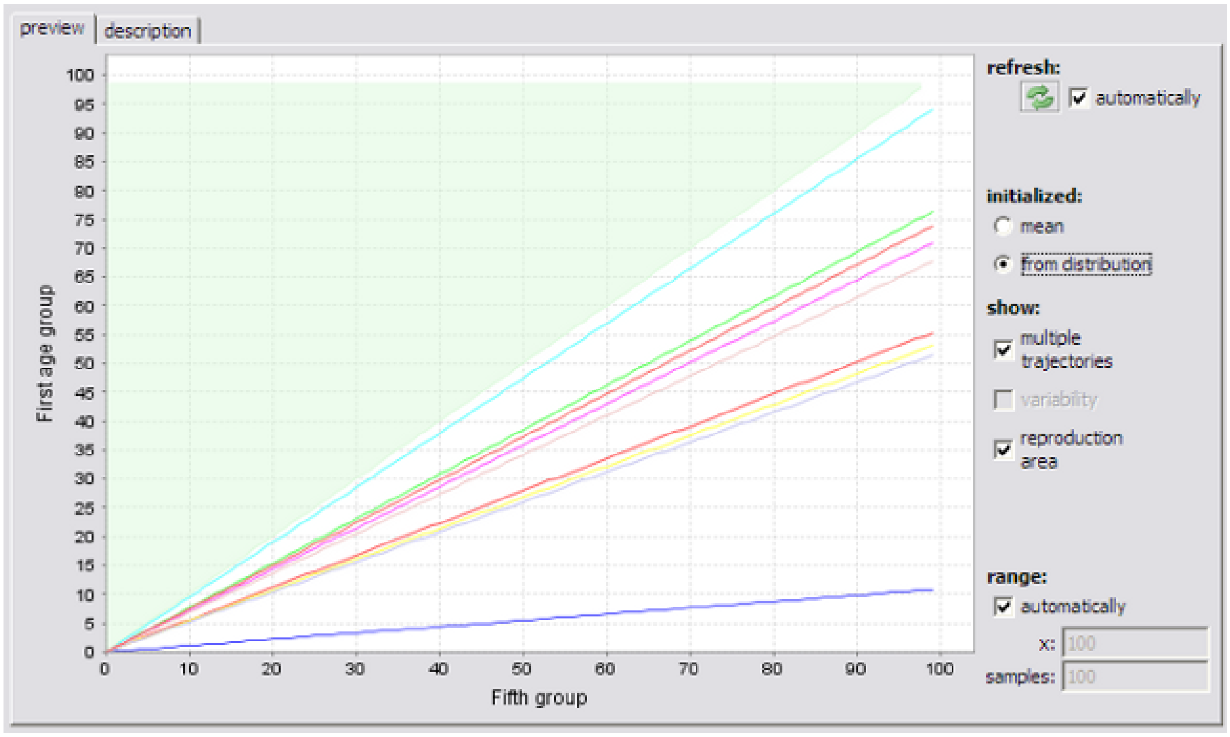
You can customize the preview display in various ways.

- **refresh:** When you change any of the function configuration items described above, the preview must be refreshed in order for you to see the effect on the transition plot. You can refresh the preview manually by clicking the refresh button, or you can select the automatically checkbox and SLAM will update the preview for you whenever you update the function configuration.
- **initialized:** If any of the function parameters are configured as draws from random distributions, the preview plot will be random. There are two ways to look at such a relationship.
  - ♦ **mean:** Each random parameter will be assigned the mean value of the distribution from which it is drawn, and the relationship will be plotted using those parameter values.
  - ♦ **from distribution:** Each random parameter will be assigned a value drawn from its distribution, and the relationship will be plotted using those values. With this option selected,

each time you click the from distribution radio button new draws will be made and the relationship will be re-plotted using the new values.

If none of the function parameters are drawn from distributions, the initialized options have no effect on the preview.

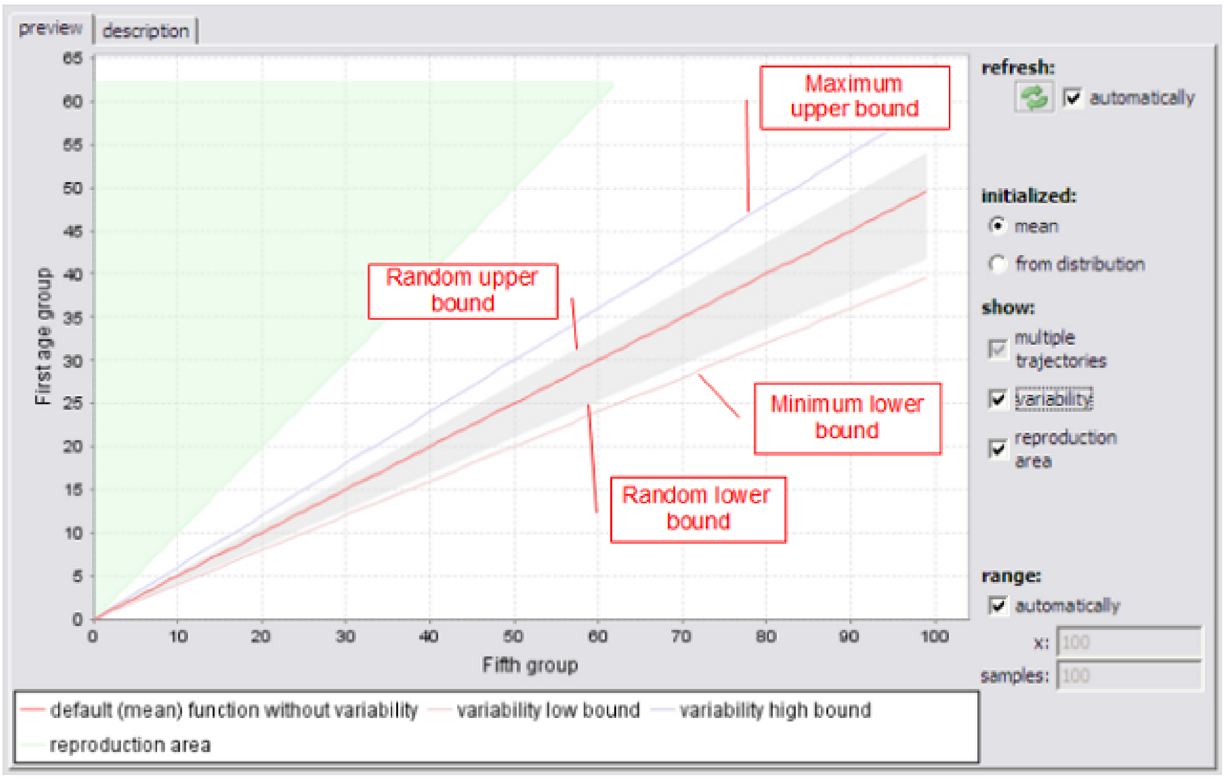
- **show:** These options affect what appears in the preview plot along with the transition function.
  - ♦ **multiple trajectories:** This option is meaningful only for functions with randomly-generated parameters when the from distribution option is selected. In that case, selecting the multiple trajectories checkbox causes several relationships to be displayed on the same set of axes, each plotted using parameters generated by separate draws from the parameter distributions.



Multiple trajectories

Repeated clicking of the from distribution button redraws the plot with a new set of randomly-generated functions, as does clearing and re-selecting the multiple trajectories checkbox.

  - ♦ **variability:** This checkbox is enabled only if you have selected the annual variability option as well as the mean display option. When variability is selected, the preview will display two sets of bounds around the mean function plot.



Variability ranges

The function with no variability added is drawn in red. Surrounding it is a region shaded in gray which represents the range of values which the function with variability added could have taken on. Upper and lower limits of the gray area are derived from random draws, such as those that are computed at the beginning of each trajectory. The upper and lower bounds are labeled Random upper bound and Random lower bound in the figure above. When you refresh the display, the boundaries of the gray region will change as they are re-calculated from new draws.

Single, gray lines drawn above and below the function are the maximum and minimum possible values that the function with variability included can take on. These are derived based on the upper limits of the high and low values defined for the annual variability bounds, including the selection state of the relative checkbox.

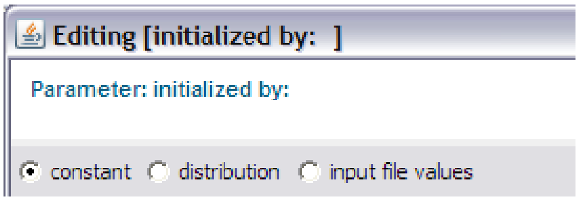
- **reproduction area:** When this checkbox is selected, the region of the plot containing all values that exceed the stock stage abundance is shaded green. As described in Survivorship and reproductive transitions, only functions describing reproductive transitions can take on values in the green area.

SLAM parameters

Many SLAM parameters can be configured as one or another of a standard variety of types. This section describes the ways in which parameters can be configured and how values for the parameters are calculated during a simulation.

'Standard' SLAM parameters are used in several parts of SLAM scenario configurations:
- Stage initialization
- Split ratio variability
- Transition functions
- Environment variable specification
- Dynamic driver initialization

There are three kinds of parameters: constants; generated values drawn from random distributions, and data points read from input files. You set the type for a given parameter by opening the parameter editor and selecting the constant , distribution , or input file values radio button in the editor's upper left-hand corner.



Parameter type selection buttons

The appearance of the parameter editor interface changes according to the type of parameter you select.

Parameter types

- **Constants**, selected by the constant button, are the simplest parameter types. At the beginning of a simulation, constant parameters are set to the value you select and this value remains unchanged throughout every trajectory of the simulation.

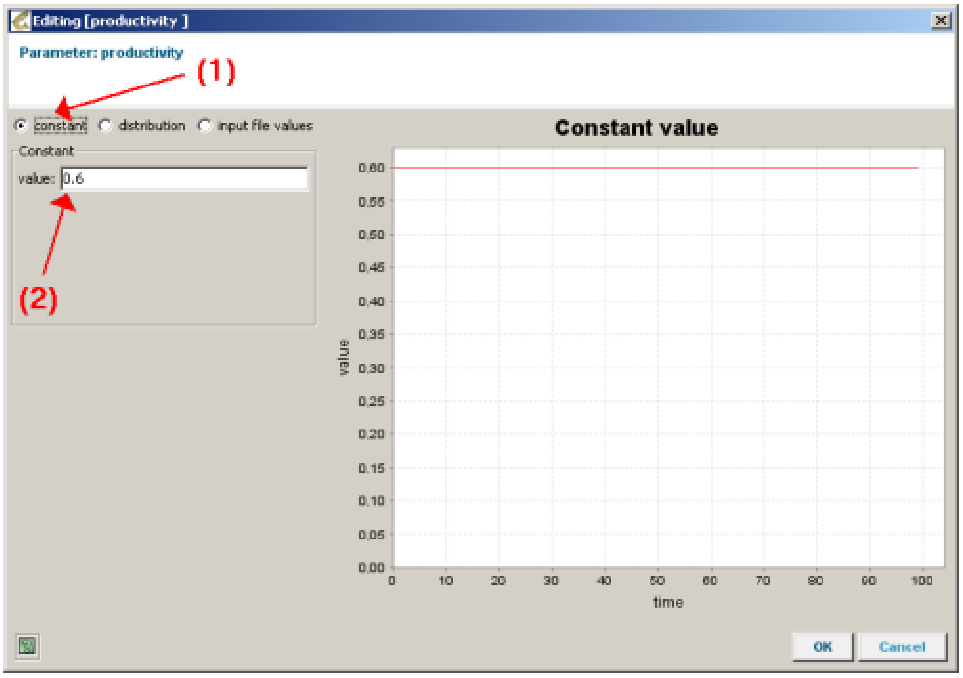
- **Generated values** , selected by the distribution button, are drawn from a probability distribution generator that you specify. Several distributions are available for generating parameter values.

- **Data points**, selected by the input file values button, are read from data files.

Constant

A constant parameter has the same value throughout all trajectories of a simulation.

1. Choose the "Constant" radio button on the top (1)
2. Fill in the value (2)
3. Click the OK button (*Click 'Cancel' to leave unchanged.*)



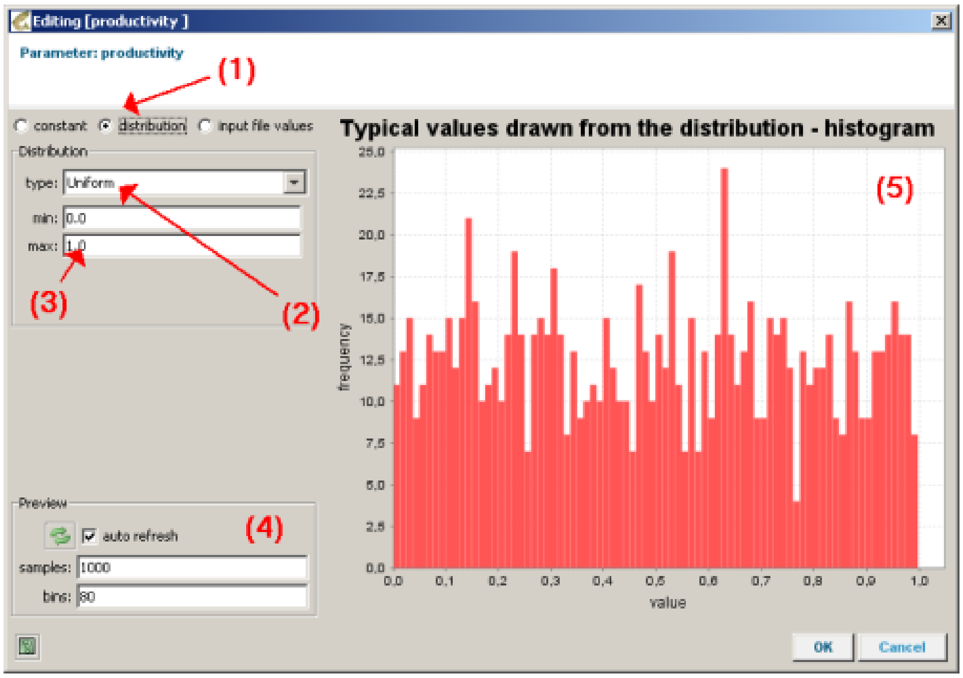
value: 0.6

Generated values

You can have SLAM assign parameter values by making pseudo-random draws from a distribution function. In the configuration panel, specify the distribution by selecting a functional form (Uniform, Normal, *etc.*) from the ‘type’ drop-down list, then choose values for the distribution parameters.

To select and configure a randomly–generated value,

1. Select the distribution radio button. (1)
2. Pick a distribution type from the type list. (2)
3. Assign values to the distribution parameters. (3)
4. A histogram of representative draws from the distribution with the parameters you have selected is displayed on the right. (5)
5. Adjust the appearance of the histogram by using the display controls. (4)
6. Click OK to accept your new parameter definition, or Cancel to forget your changes.



Available distributions

These distribution types are available in SLAM:

- Uniform
- Triangular
- Normal
- Lognormal

For generating random draws from distributions, SLAM uses the **Stochastic Simulation in Java (SSJ)** library developed in the Département d'Informatique et de Recherche Opérationnelle (DIRO), at the Université de Montréal.

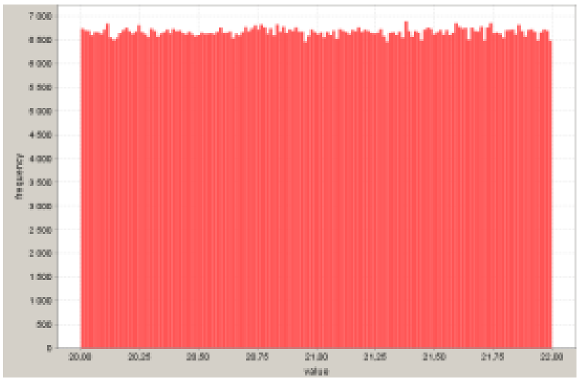
Parameter Distribution Types

These are the distribution types currently available in SLAM, along with the distribution parameters you can specify in the value generator configuration panel :

Uniform distribution

Distribution parameters :

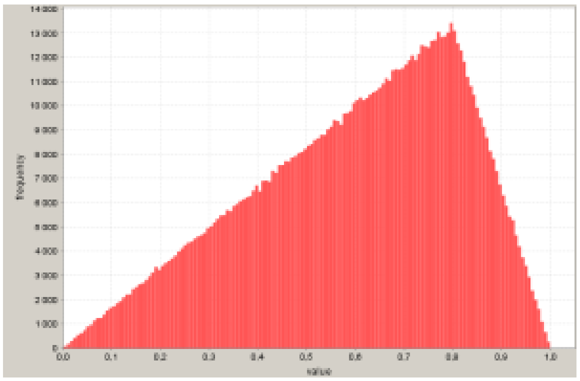
- minimum
- maximum



Triangular distribution

Distribution parameters :

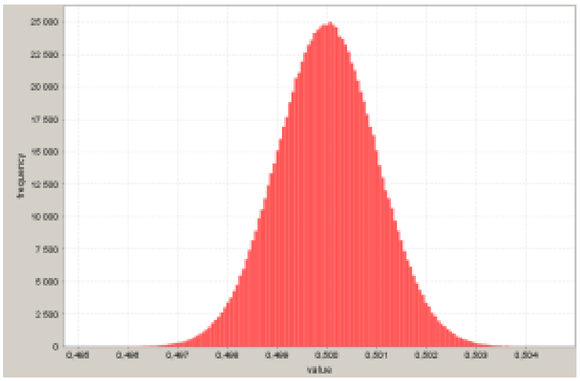
- minimum
- mode
- maximum



Normal distribution (truncated)

Distribution parameters :

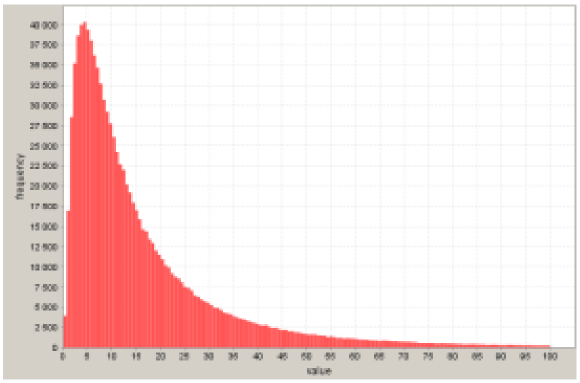
- minimum
- mean
- maximum
- standard deviation



Lognormal distribution(truncated)

Distribution parameters :

- minimum
- mean
- maximum
- standard deviation



Using existing data points

You can assign values to SLAM parameters by reading them in from external data files.

SLAM accepts csv (comma separated variable) files as inputs. They can contain one or more series, in the following format. (*Don't forget to separate series with blank lines.*)

```
<series name>
traj. 1,   242, 10, 62, 194, 307, 237, ... , <data(year n)>
traj. 2,   469, 12, 99, 325, 517, 343, ... , <data(year n)>
...
traj. N,   661,  0,  11, 67, 210, 333, ... , <data(year n)>
blank line
<another series name>
traj. 1,   242, 10, 62, 194, 307, 237, ... , <data(year n)>
traj. 2,   469, 12, 99, 325, 517, 343, ... , <data(year n)>
...
traj. N,   661,  0,  11, 67, 210, 333, ... , <data(year n)>
blank line
...
```

In the example, there are N trajectories, each containing data for n years.

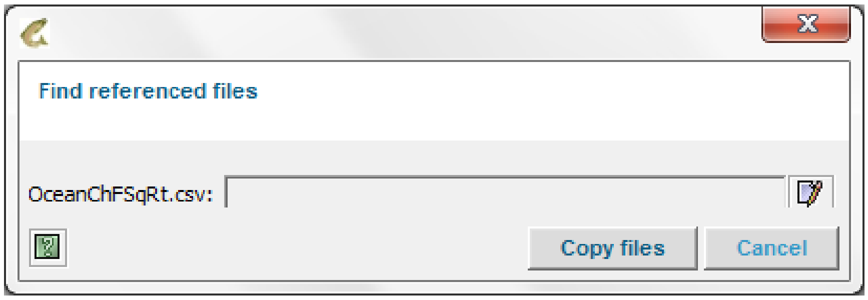
You can use any csv-file generator, such as Microsoft Excel, to produce each trajectory block.

When you load a data file associated with a scenario, SLAM will store it in the datafiles directory. When you export the scenario, you have the choice of whether or not to embed associated datafiles into the export file. If not, remember to keep the data files with the scenario export, as SLAM will ask for them when the scenario is imported again.

Loading scenarios with data files

Scenarios containing parameters that take on values read from input data files are different from other scenarios in that the contents of the data files, unlike all other parameter definitions in SLAM, are not contained in SLAM's database. Because of this, when SLAM loads a scenario that references a data file, it checks to see that there is a file with the proper name in the SLAM datafiles directory.

If a file cannot be found, you'll be prompted to browse for a copy:



Find input data files

If you don't locate the file, the scenario will not be successfully initialized, and an ugly error dialog will appear that displays the name of the file that couldn't be found along with a Java exception. The scenario will be loaded, so you can edit it, but you can't run a simulation with it.

If you browse to a copy of the input file and select it, SLAM will copy the file to its datafiles directory. The file you select must have the same name as the one referenced in the scenario; this name is displayed in the file-finder dialog as shown above.

If you import a scenario that references input data files and the data files are not imported along with the scenario, you can copy the files to your drive and have SLAM copy them when it loads the scenario. (It will attempt to do this immediately after saving the scenario.) Alternatively, you can copy the input files directly to the SLAM datafiles directory, and SLAM will find them automatically when you load the scenario.

SLAM scripts

SLAM provides a variety of built-in, parameterized functions you can use to define the transition relationship between stock and recruit stage abundances. If your model requires more flexibility, you can modify the results of transition functions or replace them entirely with values calculated by scripts. You can insert pre-existing scripts into your scenario or create new scripts in SLAM's script editor.

You can use SLAM scripts in any of these situations:
– calculating stock-to-recruit abundance relationships
– calculating ratios of splitting stages
– defining dynamic driver behavior *(Dynamic drivers must have scripts associated with them.)*

More about scripts:

- How scripts work: An introduction to SLAM's scripting language.
- Creating and editing scripts: A description of SLAM's script editor.

How SLAM scripts work

SLAM scripts are written in Groovy, a Java-like scripting language. Groovy's syntax follows that of Java (and bears a family resemblance to C, C++, Perl, Python, and several other languages you may be familiar with). Java libraries can be referenced in Groovy scripts, with the result that you can do in Groovy much of what you might have done in a standalone Java program.

(For programming *aficionados*: Groovy code in SLAM scripts is compiled into Java bytecode before a simulation run starts. This means that, aside from the fact that the additional lines of code must be executed, the overhead normally associated with script interpreters doesn't exist.)

Writing scripts

What follows is a very basic introduction to writing Groovy scripts. For a full explanation of Groovy's available features, see the Groovy website.

1. Variables

Contrary to Java practice, variables in Groovy do not need to be declared before they are used - just use them:

```
x = 0
y = -4;
```

You *can* declare them if you like:

```
double x
```

The first time a variable appears in a script, though, it must be initialized (*ie*, assigned a value) if the statement isn't a declaration:

```
x
```

will produce an error if it represents the first occurrence of the variable `x`.

You can end a line with a semicolon `;` or with nothing (*ie*, by typing <return>). You can put multiple statements into the same line by separating them with semicolons

```
x = 0; y = -4;
```

but you must exercise care when you extend a statement across two lines.

```
x
= 0
```

produces an error, but

```
x =
0
```

is okay.

You can assign numbers, strings, built-in Groovy collections, Java objects, etc., to variables. The type of a number is assigned based on what it is assigned to unless you declare it. (The default numeric types appear to be Integers and BigDecimals.)

Don't omit leading or trailing zeros! These statements result in errors:

```
x = .9
y = 9.
```

and should be written

```
x = 0.9
y = 9.0
```

Scientific notation works:

```
x = 6.022e23
y = 3.14159E0
z = 7.297E-2
```

Assign strings to variables as you would have guessed:

```
z = "SLAM"
```

Boolean variables are also available:

```
b = true
nb = false
```

2. Operators:

Arithmetic:
The usual `+-*/` operators are available:

```
x = 4 + 3 * 2
y = 6 - 7 / 2
```

and they *bind* (that is, are evaluated) in the usual order: multiplication and division operations are evaluated before addition and subtraction. For belt-and-suspenders script programmers, parentheses () are available for removing any doubt about the order of evaluation.

```
x = 4 + (3*2)
```

evaluates to 10, while

```
x = (4 + 3)*2
```

evaluates to 14.

Shortcut operators `+=`, `-=`, `*=`, and `/=` are available. The statements

```
w = 1; w += 1;
x = 3; x -= 1;
```

```
y = 1; y *= 2;
z = 4; z /= 2;
```

set all four variables to 2.

Increment and decrement operators `++` and `--` are included in both prefix and postfix forms:

```
x = 1;
y = ++x;
```

leaves x and y both equal to 2, while

```
x = 1;
y = x++;
```

leaves x equal to 2 and y equal to 1. `--` works in an analogous, decrementing fashion.

The modulo operator `%` is available for integers:

```
x = 4 % 3
```

gives `x` a value of 1. Don't attempt to apply the modulo operator to non-integer values as errors will result.

Groovy syntax also includes the exponentiation operator `**` missing from Java.

```
x = 2**3
```

evaluates to 8, and

```
y = 2**-1.5
```

evaluates to 0.35355339059327373.

Logical:
The logical *and*, *or*, and *exclusive-or* operators `&`, `|`, and `^` are available.

```
x = true & true
y = true | false
z = true ^ false
```

are all true, while

```
x = true & false
y = false | false
z = true ^ true
```

all evaluate to false.

Comparison:
Groovy recognizes the comparison operators *greater-than*, *less-than*, and *equals*: >, <, and ==.

```
x = 24 < 42
y = 3 > 2
z = (true == false)
```

are, respectively, true, true, and false.
Note: In Java, the == operator compares values for primitives like *int* but compares identity for objects. In Groovy, == always compares values.

Branching:
Groovy recognizes the usual if-then branch control structure.

```
...
x = 2;
if (y > 3) {
x += 1;
}
else if (y < 2) {
x += 2;
}
else {
x += 3;
}
```

Groovy includes a *switch* structure, which is more general than that implemented in Java.

More information: For much more information about Groovy, including all the features skipped over here, see groovy.codehaus.org.
This link is viewed more satisfactorily when copied and pasted into your browser of choice.

The SLAM script editor

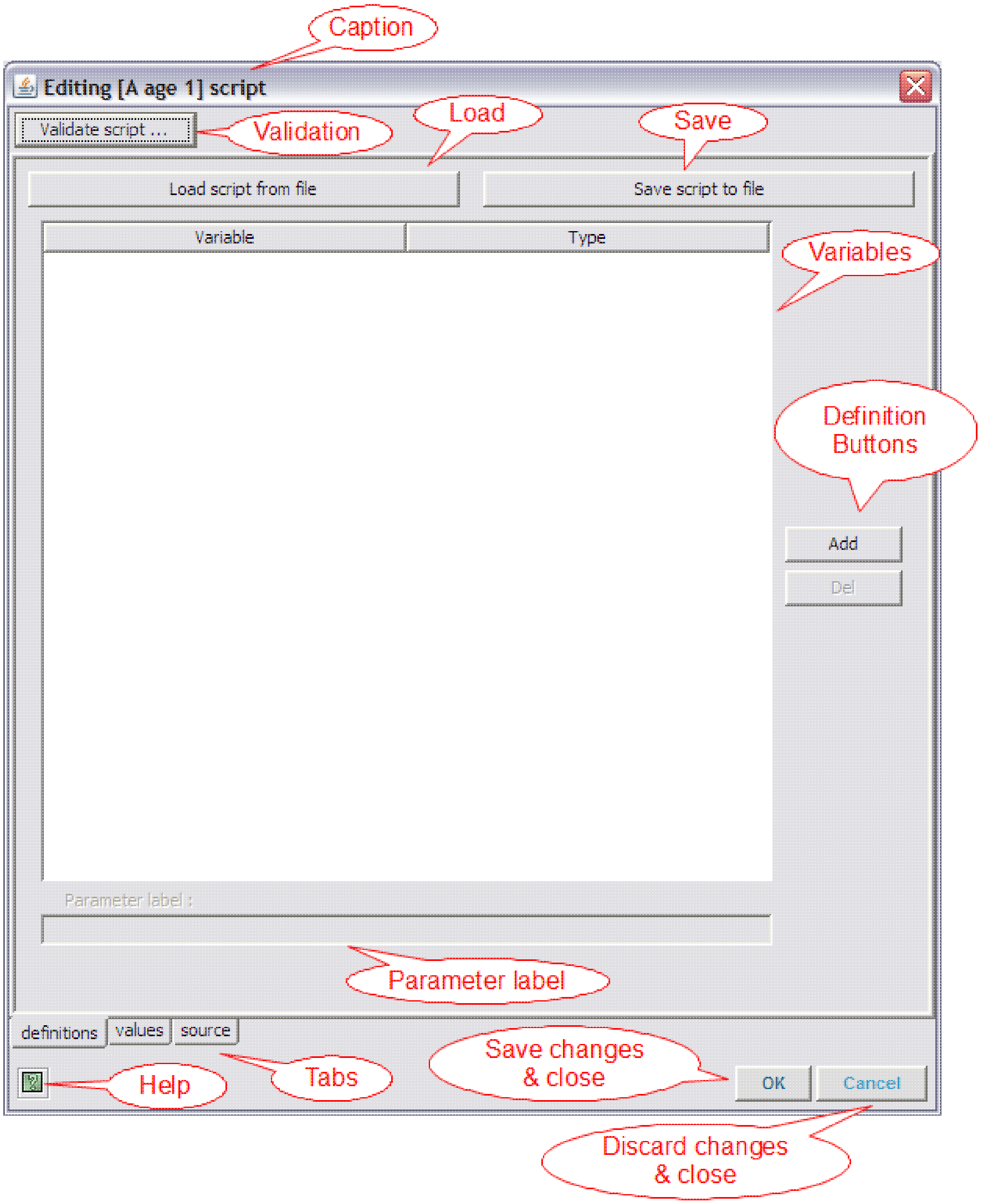
SLAM's script editor enables you to create and modify influence, split-influence, or dynamic driver scripts you've included in your scenarios. You can import and save scripts to and from text files.

The script editor allows you to connect variables in your scripts to the values of influencing elements that the variables represent. It also provides a test facility in which you can perform simple validations on your scripts before running simulations.

Launch the script editor as described in these help pages:

- Configuring transitions in the scenario editor
- Configuring life cycle stages in the scenario editor (splitting stages)
- Configuring dynamic drivers in the scenario editor.

The script editor interface will appear:



Script editor variable definition tab

There are three tabs in the script editor; the variable definitions tab, shown above, is the default. Most of the controls are specific to configuring variable definitions, but some are present in all tab views:

- *Caption* : The caption bar contains the name of the object (transition, split transition, or dynamic driver) being edited.
- *Validation* : Clicking the Validate script... button launches the script validation dialog.

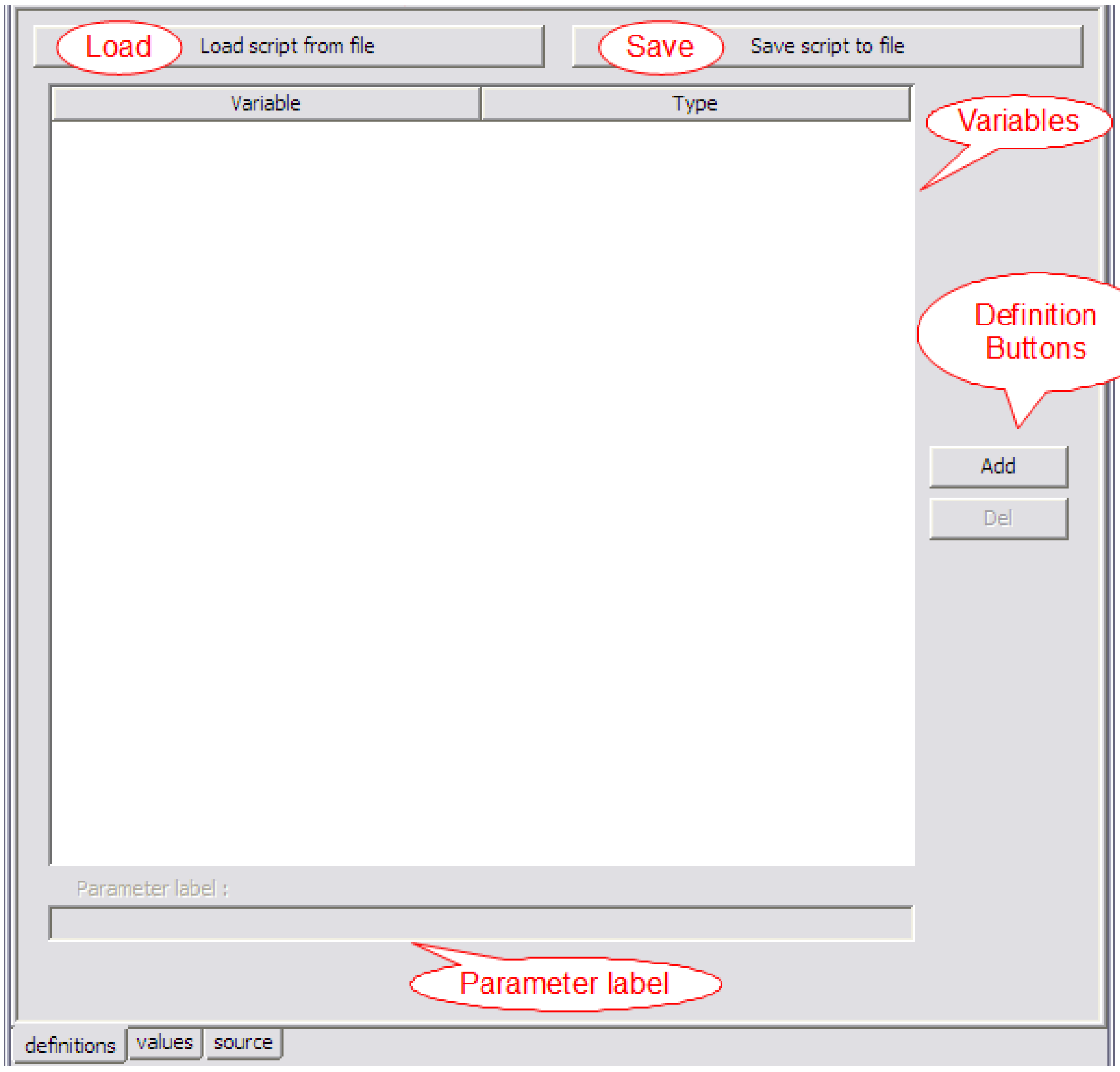
- *Tabs* : The three tab interfaces, definitions, values, and script.
- *Help* : Clicking the help icon launches the SLAM help viewer.
- *Save changes & close* : Clicking the OK button closes the script editor, saving any changes you have made to the script.
- *Discard changes & close* : Clicking the Cancel button closes the script editor, discarding any changes you have made to the script.

The SLAM script editor definitions tab

In the definitions tab, you define or delete variables that represent influencer values, environmental variables, split ratios, or constant coefficients. You then map these variables (or set their values) in the values tab.

Only those script variables need to be defined here that are mapped to life cycle elements or are special constants. Variables that appear in the definitions list will appear in the values tab. Script variables that represent influencer values, split ratios, and environment variables must be defined in this tab in order to be available for mapping to the life cycle elements they represent. Whether you want to define constant coefficients here or simply code their values into the script is up to you.

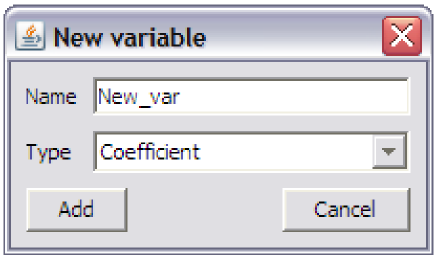
Controls specific to the definitions tab are shown below. (There are also controls common to all tabs.)



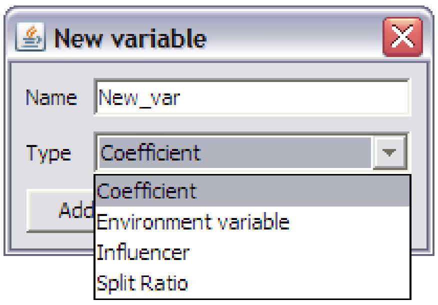
Script editor definition tab

- *Load* : Clicking the Load script from file button displays a file-browsing dialog open to your scripts directory: either influence-scripts, split-influence-scripts, or dynamic-driver-scripts, depending on the kind of script you're editing.

- *Save* : Clicking the Save script to file button displays a file-browsing dialog open to your scripts directory: either influence-scripts, split-influence-scripts, or dynamic-driver-scripts, depending on the kind of script you're editing.
- *Variables* : The variables window contains a list of the names and types of variables defined for this script.
- *Definition buttons* : Clicking the Add button opens the New variable dialog in which you specify the name and type of new script variables.



New variable dialog



New variable types

The variable's name goes in the Name field. Set the variable's type by selecting an option from the Type drop-down list. Variable types specify what the variable represents in the script; the type determines how the variable will be mapped in the values tab.

- ♦ Coefficient : Variable will be assigned a constant value
- ♦ Environment variables : Variable will be assigned the value of an environment variable.
- ♦ Influencer : Variable will be assigned the value of an influencing stage or dynamic driver
- ♦ Split Ratio : Variable will be assigned the value of one of the recruit transition ratios in a splitting stage.

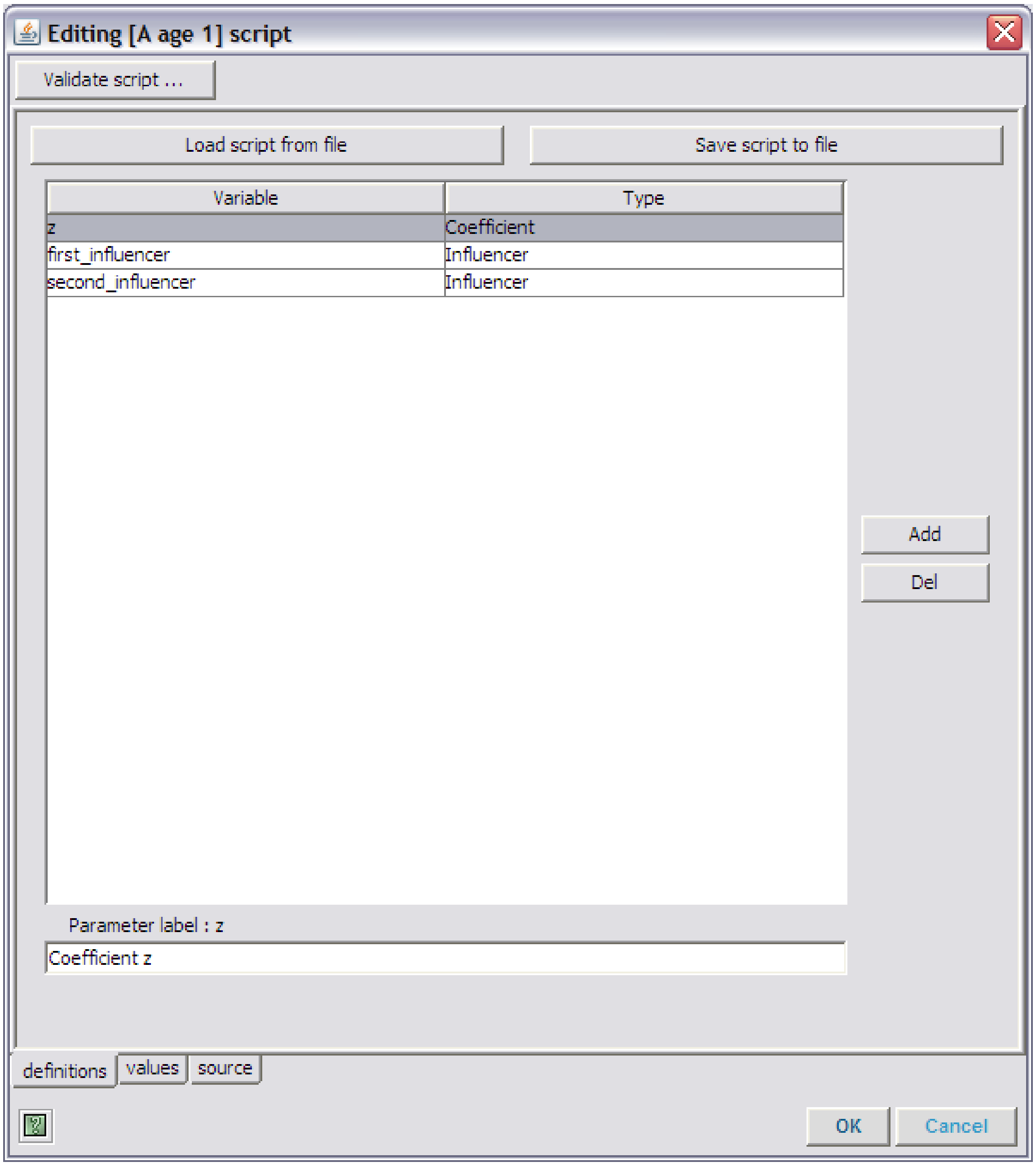
Click Add to create the new variable or Cancel to discard it.

To delete a variable, select it in the list and click the Del button.

- *Parameter label* : This field displays the label corresponding to the selected coefficient-type variable in the list above. If a label is assigned to a coefficient, it will appear in place of the coefficient name in the values tab.

Example

In the editor shown below three variables are defined: *z*, a coefficient, and two influencer variables called *first_influencer* and *second_influencer*. The coefficient variable entry is selected, displaying its assigned parameter label: Coefficient z .



Variable	Type
z	Coefficient
first_influencer	Influencer
second_influencer	Influencer

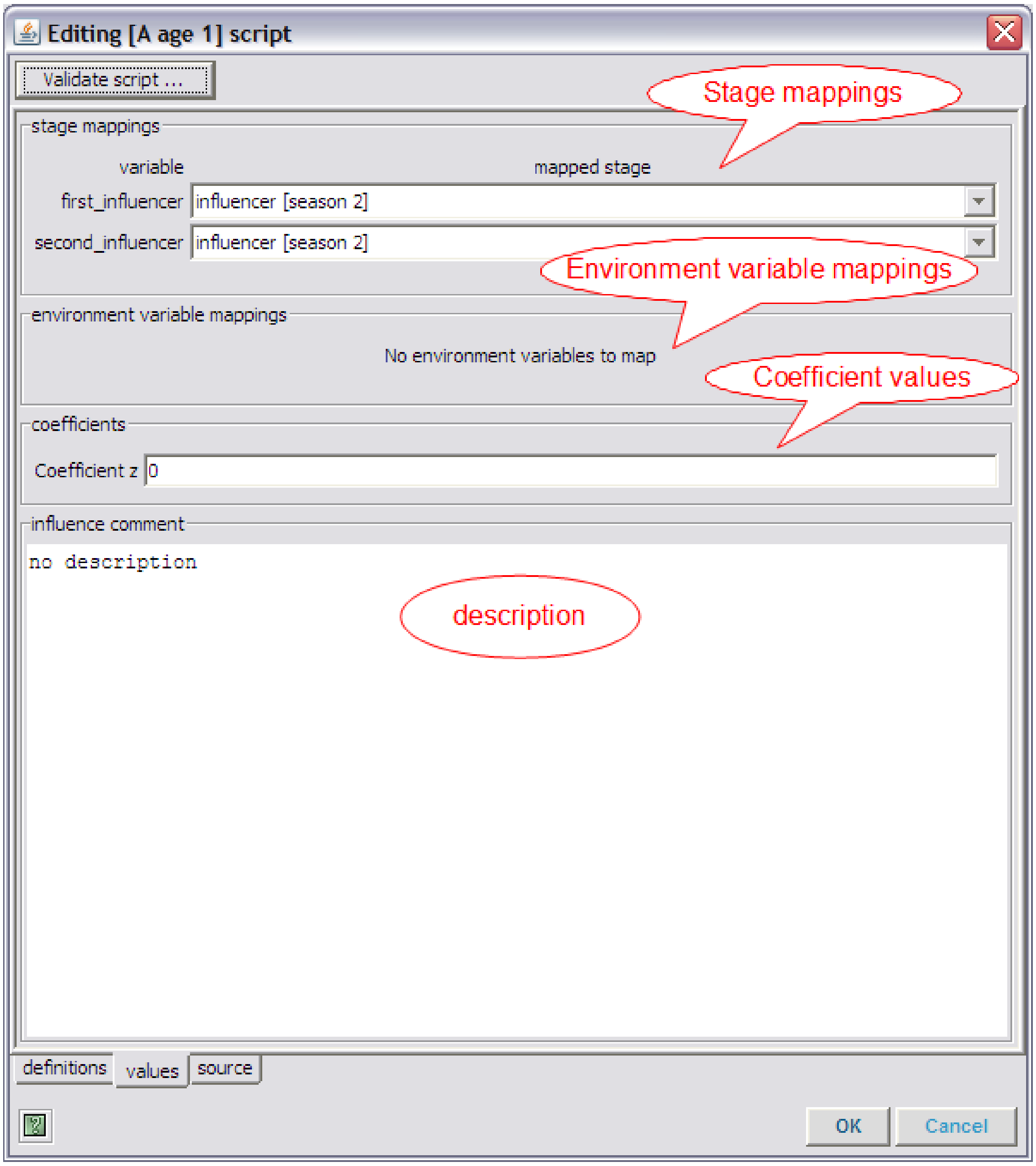
Parameter label : z

Coefficient z

Example variable definitions

The SLAM script editor values tab

In the values tab, you define the associations between variables defined in the definitions tab and the life cycle element values they correspond to. You also specify the value of constant coefficients and provide a text description of the script in this tab.



stage mappings

variable	mapped stage
first_influencer	influencer [season 2]
second_influencer	influencer [season 2]

environment variable mappings

No environment variables to map

coefficients

Coefficient z 0

influence comment

```
no description
```

Script editor value tab

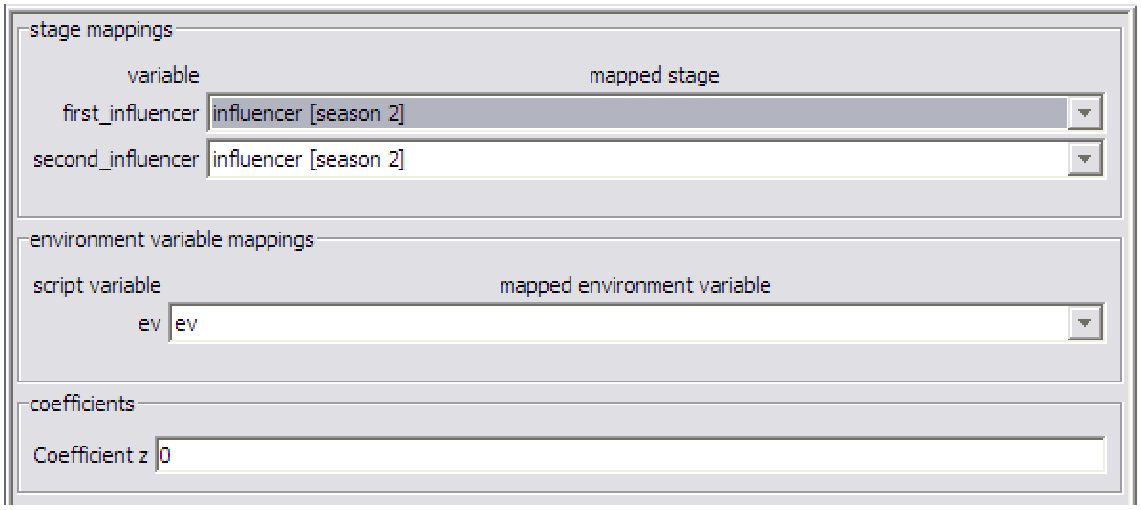
Some controls are common to all tabs. Controls specific to the values tab include the following:

- *Stage mappings* : Influencer variables defined in the definitions tab are mapped to life cycle stages or dynamic drivers that influence the scripted element. Under some circumstances, this control will be replaced by the *lifecycle element mappings* control.

- *Environment variable mappings* : Variables defined as environment-variable types in the definitions tab are mapped to environment variables that influence the scripted element. Available environment variable elements can be chosen from a drop-down list. If no environment variables influence the element *or* if no environment-variable type variables are defined in the definitions tab, this control will be empty. (See *lifecycle element mappings*.)
- *Coefficient values* : Constant coefficients defined in the definitions tab are listed here. Enter the value of each coefficient in the field to the right of the name. If the coefficient was given a parameter label, that will be displayed instead of the variable name.
- *Description* : This field contains any description, notes, or comments you would like to associate with the script.

More examples

For the value tab mapping controls shown below, the influenced transition is influenced by two stages and an environment variable. Two influencer variables and one environment-variable variable have been defined in the definitions tab.



stage mappings

variable	mapped stage
first_influencer	influencer [season 2]
second_influencer	influencer [season 2]

environment variable mappings

script variable	mapped environment variable
ev	ev

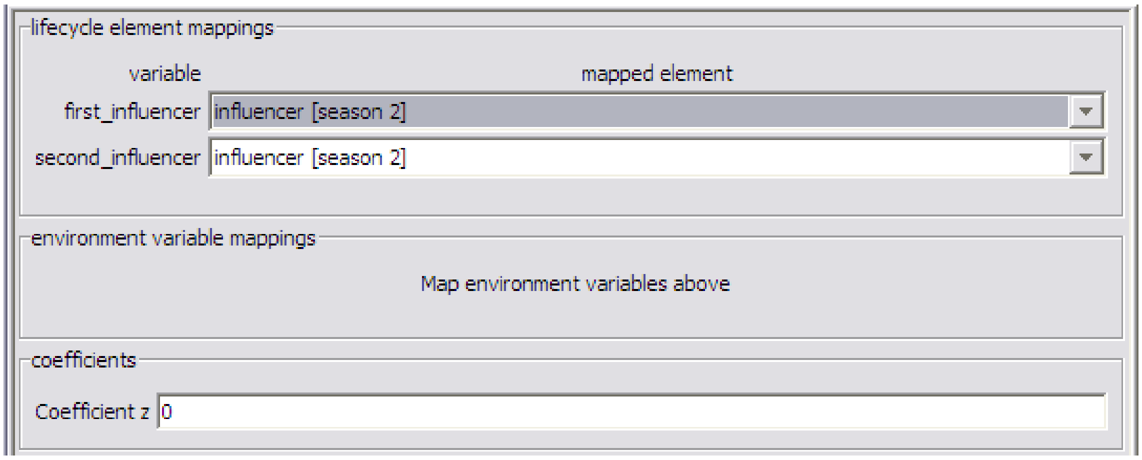
coefficients

Coefficient z: 0

Mapping environment variable influence

The influencers are mapped in the *stage mappings* control, while the environment variable is mapped in the *environment variable mappings* control.

In the mapping controls shown below, no environment-variable variable has been defined on the definitions tab, but an influencing environment variable does exist in the life cycle model.


lifecycle element mappings

variable	mapped element
first_influencer	influencer [season 2]
second_influencer	influencer [season 2]

environment variable mappings

Map environment variables above

coefficients

Coefficient z: 0

No environment variable influencer variable defined

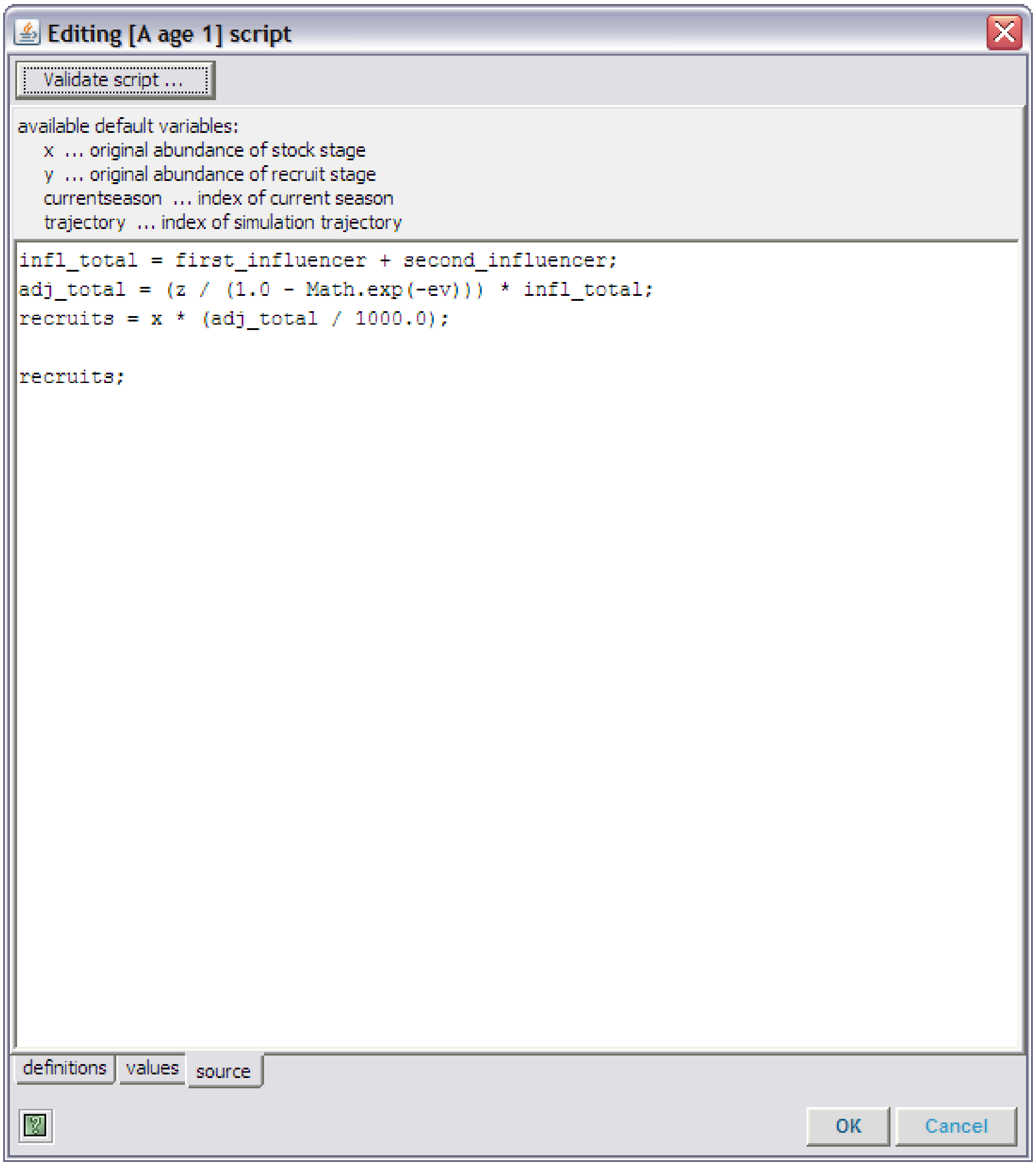
In this case, the *stage mappings* control is replaced by a general *lifecycle element mappings* control, in which not only influencing stages and dynamic drivers, but environment variables are included. Variables available for mapping are still only those defined as influencers in the definitions tab, but they can be mapped to influencing environment variables, if desired.

The *environment variable mappings* control will be empty if either no environment variables influence the scripted element or no environment-variable type variables are defined in the definitions tab.

The SLAM script code editor tab

In the script tab, you create and modify the influence script's code.

An example script in the script tab is shown below.



```
available default variables:
   x  ... original abundance of stock stage
   y  ... original abundance of recruit stage
   currentseason  ... index of current season
   trajectory  ... index of simulation trajectory
```

```
infl_total = first_influencer + second_influencer;
adj_total = (z / (1.0 - Math.exp(-ev))) * infl_total;
recruits = x * (adj_total / 1000.0);

recruits;
```

Script editor script tab

Above the script editing control, there is a list called ‘available default variables’ displaying pre-defined variables that can be used in the script. Which presets are available depends on the nature of the script, so this list changes depending on whether the script implements a transition influence, a split influence, or a dynamic driver.

Notice that there is no difference in how "defined" variables (like first_influencer), preset variables (like x), or variables introduced in the script (like infl_total) are used in the script, except that you don't initialize preset or defined variables: they will be set to their assigned values prior to each script execution during the simulation.

Also bear in mind that mapped and pre-defined variables are "read-only" in the sense that, if you assign values to them, they maintain those values only during the execution of the script and have no effect on abundances or other parameters throughout the rest of the simulation. The only effect that a script has on the simulation is exerted through its return value - all other values calculated during the script's execution are discarded when the script completes.

For more information on writing SLAM scripts, see How scripts work.

The SLAM script validator

The validator tool allows you to perform simple validation testing on your script. This offers the opportunity to check that script syntax and algorithm coding is correct before you attempt to run the script in your simulation.

For a life cycle model of any complexity, it can be difficult to check the values generated by influence or dynamic driver scripts by examining the simulation results. In addition, script errors occurring during a simulation run may produce error messages that are hard to evaluate, if not downright obscure.

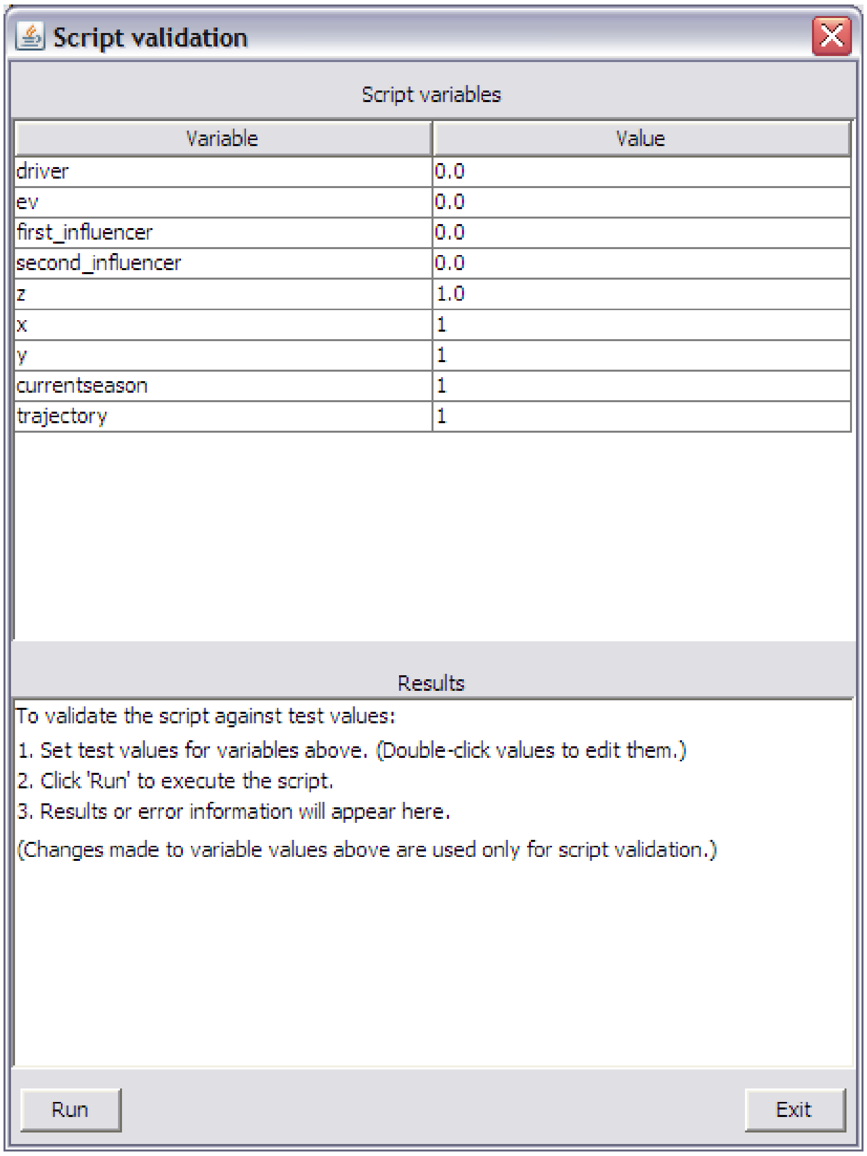
You can run your script in the validator after assigning arbitrary values to all the input variables. In the event that script errors occur (either parsing errors caused by syntax mistakes or runtime errors), the validator attempts to identify the line where the error was detected along with the error description generated by the script engine in order to help you debug the script. (The validator is not always successful in this attempt.)

Because the validator is a stand-alone tool, you must specify the initial values of all script variables. Since there is no simulation running while the validator is in use, the evaluation of interactions between multiple influencers and/or stages is outside the range of its function.

- Description
- Example results

Description

Launch the validator by clicking the Validate script... button above the tabs in the SLAM script editor. The Script validation dialog appears, as shown below.



Script variables

Variable	Value
driver	0.0
ev	0.0
first_influencer	0.0
second_influencer	0.0
z	1.0
x	1
y	1
currentseason	1
trajectory	1

Results

To validate the script against test values:
1. Set test values for variables above. (Double-click values to edit them.)
2. Click 'Run' to execute the script.
3. Results or error information will appear here.

(Changes made to variable values above are used only for script validation.)

Script editor validation window

The validation window is divided into upper and lower sections. The upper part contains a list of defined variables and pre-defined parameters along with the initial values they will be assigned when the validator runs the script. You can edit a variable's initial value by double-clicking the Value column on the row corresponding to the variable.

Pre-defined variables are assigned default values of 1. Defined coefficient-type variables default to the values they are assigned in the values tab. Other defined variables default to zero. As mentioned above, since the validator does not initiate a simulation run, there are no life cycle values available to assign to influencer, environment–variable–, or dynamic–driver–type defined variables.

Similarly, any changes you make to variables in the validation window are limited to the validation process - once the window is closed, those changes are discarded.

The lower section of the validation window contains brief instructions on how to use the validator when the window is first displayed. Once the script runs, results or error messages are displayed here.

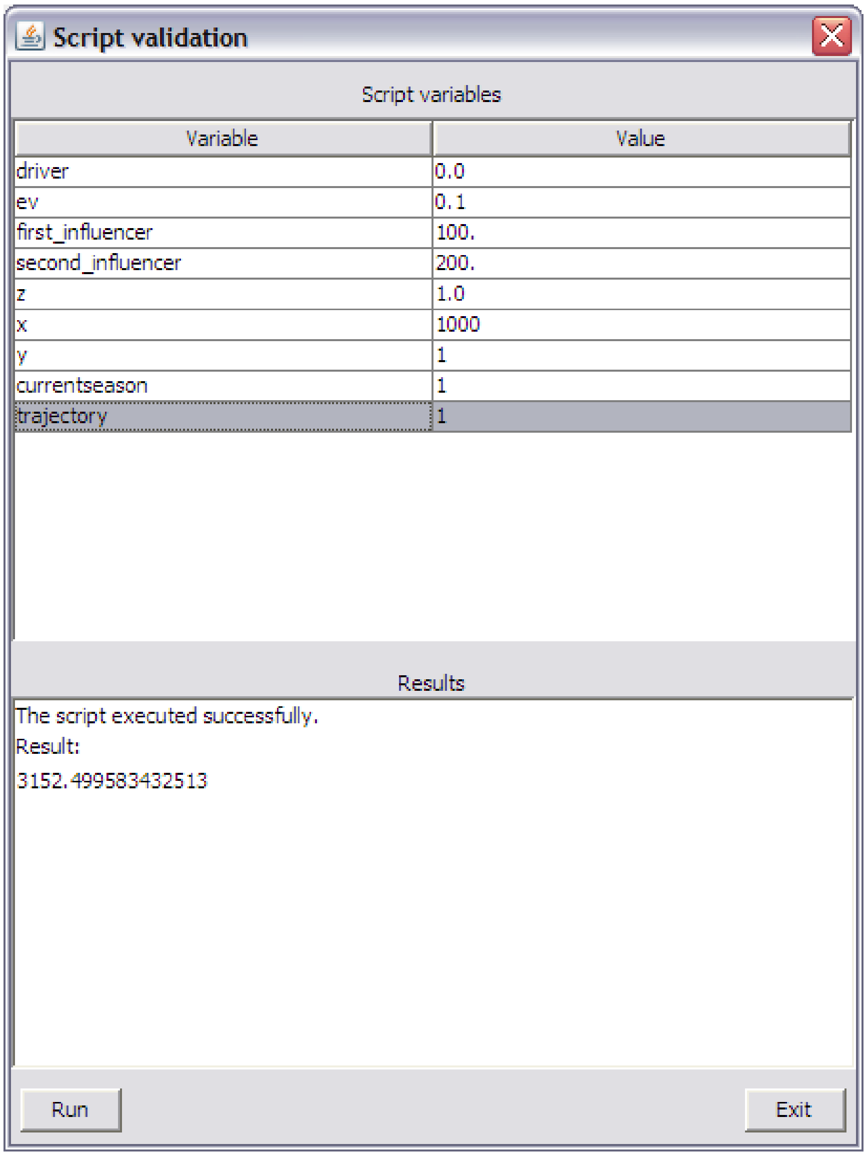
Example results

The script below was used as an example in the script tab description:

```
 infl_total = first_influencer + second_influencer;
adj_total = (z / (1.0 - Math.exp(-ev))) * infl_total;
recruits = x * (adj_total / 1000.0);

recruits;
```

Sample initializations and the result of running the script are shown below.

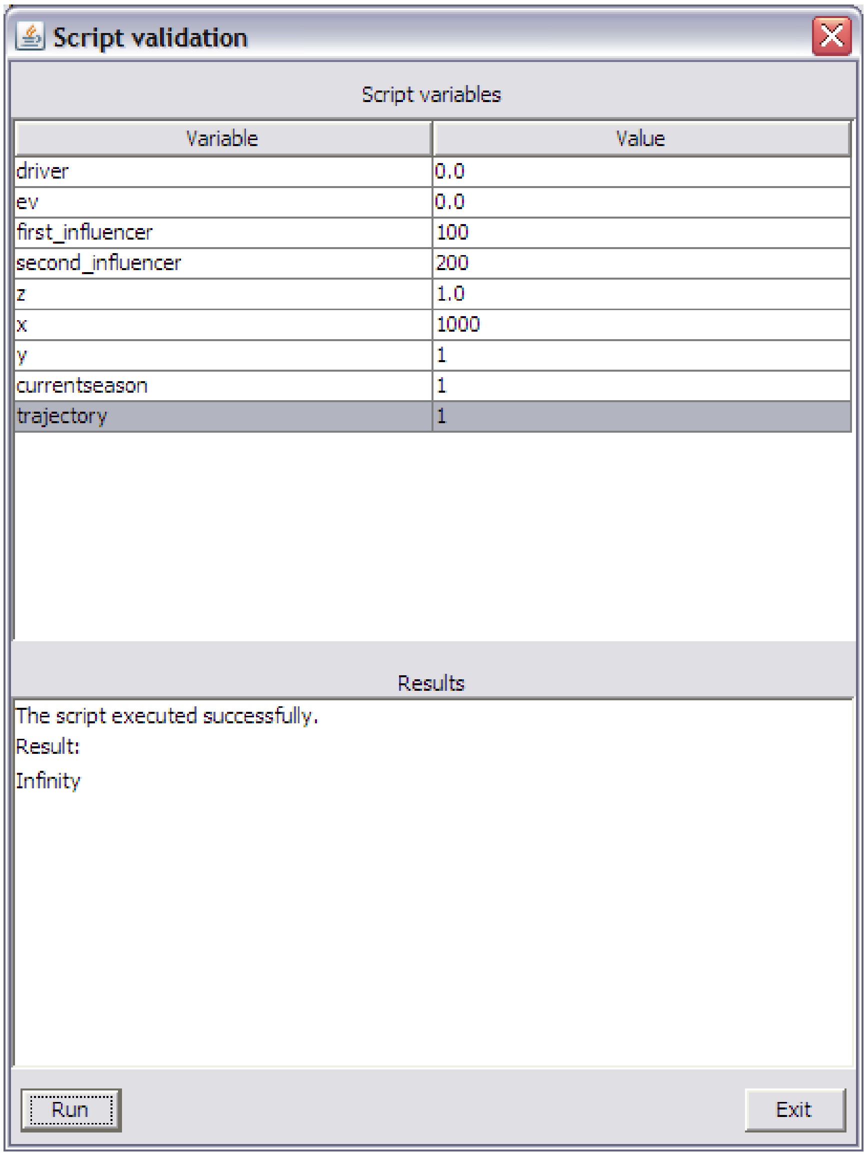


Variable	Value
driver	0.0
ev	0.1
first_influencer	100.
second_influencer	200.
z	1.0
x	1000
y	1
currentseason	1
trajectory	1

```
The script executed successfully.
Result:
3152.499583432513
```

Sample script validation results

As noted in the output window, the script executed successfully and the returned value was 3152.49958343251. Successful script execution generally means that no syntax or runtime errors occurred, not that the calculations were necessarily proper. For example, changing the value of ev to 0 gives rise to a division by zero in the second line of code:


Script validation

Script variables

Variable	Value
driver	0.0
ev	0.0
first_influencer	100
second_influencer	200
z	1.0
x	1000
y	1
currentseason	1
trajectory	1

Results

```
The script executed successfully.
Result:
Infinity
```

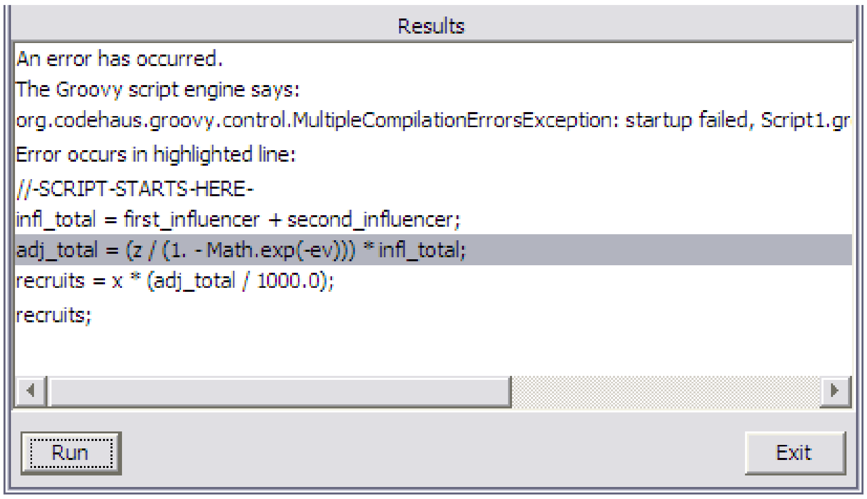
Infinite result validation output

While the script still executed successfully, the result of Infinity might call for further work on the script. (Other valid but not useful numeric values include NaN, which stands for Not-a-Number and which also suggests a problem in the calculation.)

A common error is failure to include a zero on one side of a decimal point when you specify a value in the script code. If the 1.0 in the denominator term of the second line of the script is changed to 1. as shown

```
 ...
adj_total = (z / (1. - Math.exp(-ev))) * infl_total;
...
```

this error results:



```
An error has occurred.
The Groovy script engine says:
org.codehaus.groovy.control.MultipleCompilationErrorsException: startup failed, Script1.gr
Error occurs in highlighted line:
//-SCRIPT-STARTS-HERE-
infl_total = first_influencer + second_influencer;
adj_total = (z / (1. - Math.exp(-ev))) * infl_total;
recruits = x * (adj_total / 1000.0);
recruits;
```

Parse error validation results

The complete error message (which you can see by scrolling or enlarging the validation window) reads

```
 org.codehaus.groovy.control.MultipleCompilationErrorsException:
startup failed, Script1.groovy: 70:
unexpected token: - @ line 70, column 22. 1 error
```

The line number supplied isn't directly helpful, because it counts various initialization blocks not included in the output window, so you can't count down to line 70 to find your error — but the error line is properly highlighted. The column number (22) indicates where the script parser became confused and is a good guess as to where the problem occurred.

For more information on writing SLAM scripts, see How scripts work.

Batch substitution

Batch substitution in SLAM allows you to make copies of scenarios which differ from the originals only in the configuration of particular elements. You do this by selecting a particular scenario as a template and configuring the template so that it defines the changes you desire. Then, after selecting the scenarios that you want to copy the new changes into, run the batch substitution process, creating one new scenario for each of your selected copy targets - or overwriting the original scenarios with the desired changes.

A batch substitution is equivalent to loading each scenario you want to modify, changing the desired scenario parameters, and saving the modified scenario under a new name. If you are changing several scenarios in the same way, or if the parameter changes you are making are at all complex, you'll probably find that using a batch substitution is both quicker and less prone to error.

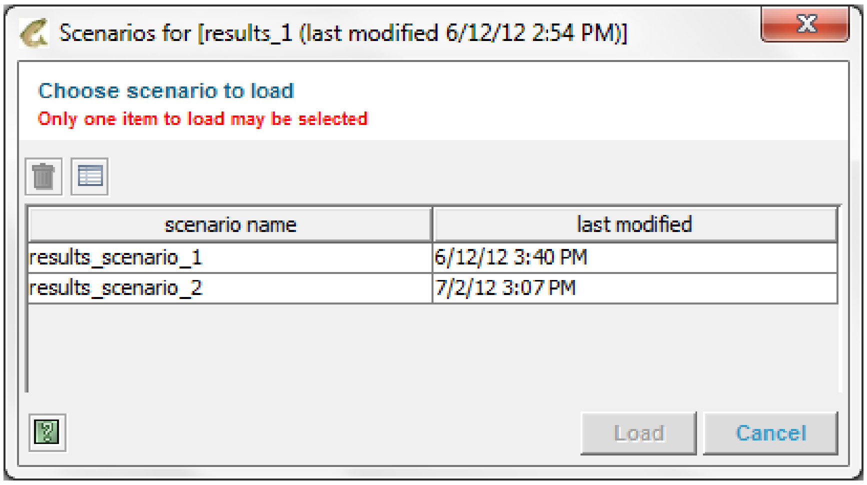
Batch substitution copies scenario A (called a *target*, here) to scenario B, but along the way it replaces specified elements of A with those from scenario C (called the *template*). The result is a scenario B that is a copy of A with a few bits replaced by pieces of C. *And* you can copy more than one target scenario using a common template.

There are two steps to making a batch substitution:

- Create a template
- Select and copy target scenarios

Creating a batch substitution template

With a life cycle model loaded, begin the batch substitution process by selecting the define batch substitution item of the Scenario menu, or type `Ctrl+Alt+U`. The template selection dialog appears:



Scenarios for [results_1 (last modified 6/12/12 2:54 PM)]

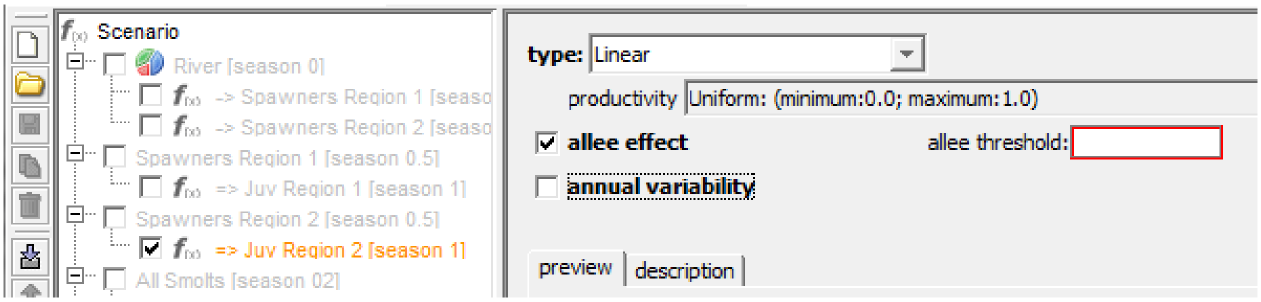
Choose scenario to load
Only one item to load may be selected

scenario name	last modified
results_scenario_1	6/12/12 3:40 PM
results_scenario_2	7/2/12 3:07 PM

Batch template selection dialog

Select a scenario you'd like to use as a template for the substitution process. Any scenario associated with the life cycle model can serve as a template; some may be easier to configure than others. The parameters of selected elements (like transition functions or variabilities) in the template will be inserted into the new scenarios you will create. If an existing scenario already contains some or all of your desired parameter settings, you can use it as a template in order to save effort — and possible mistakes — during parameter configuration.

When you have selected your desired template scenario, click the Load button (or click Cancel to cancel the batch substitution process). The template scenario is loaded into the scenario editor, with a difference: each stage and transition in the element tree on the left is accompanied by a check box. Define the elements you want to replace in your new scenario copies by selecting the check boxes that correspond to them.



Scenario
River [season 0]
-> Spawners Region 1 [seaso
-> Spawners Region 2 [seaso
Spawners Region 1 [season 0.5]
=> Juv Region 1 [season 1]
Spawners Region 2 [season 0.5]
☑ => Juv Region 2 [season 1]
All Smolts [season 02]

type: Linear
productivity Uniform: (minimum:0.0; maximum:1.0)
☑ allee effect   allee threshold:
☐ annual variability

Batch template parameter selection

Shown above, the Spawners Region 2 => Juv Region 2 transition is selected as a substitution parameter and re-configured with a linear transition function influenced by an Allee effect but with no annual variability. Every scenario copy produced by the batch substitution will contain a Spawners Region 2 => Juv Region 2 with exactly these parameters.

You can specify as many substitution parameters as you like by selecting their check boxes and configuring them as you want them to appear in the new copies.

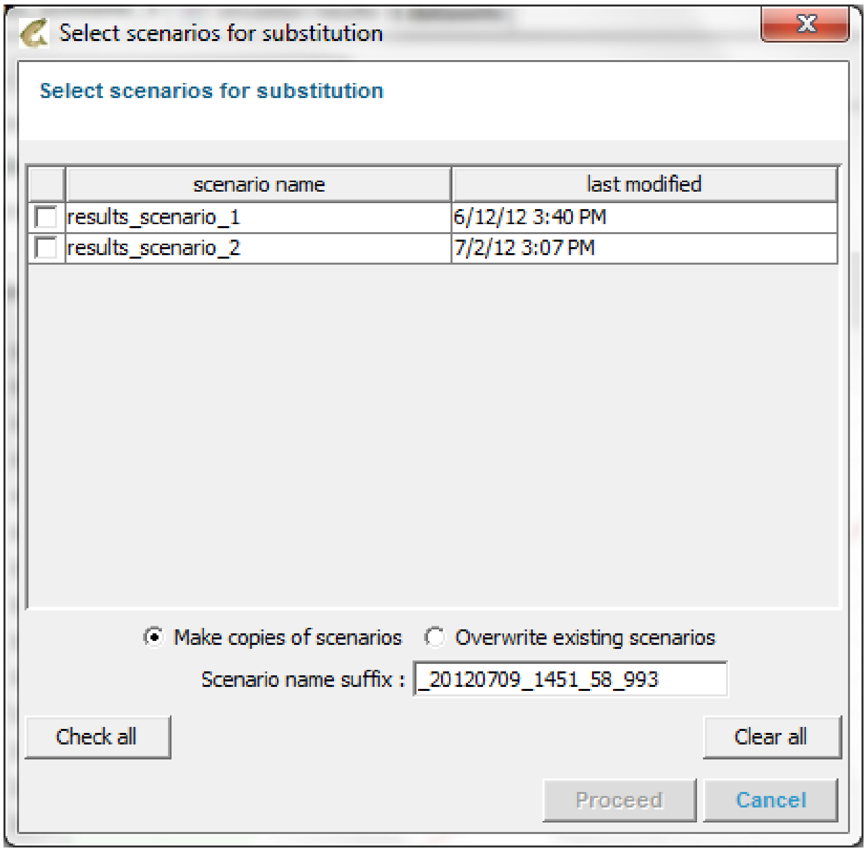
Note : The changes you make to parameter settings in a batch substitution template *do not* affect the configuration of the original scenario you selected as a template.

Life cycle elements that you do not select will not be copied into the new scenarios. They will remain configured as they are in the original copy-target scenarios.

In SLAM's Scenario menu, the define batch substitution item will have been replaced by one that reads perform batch substitution . When you have configured the desired parameters, select that menu item or type `Ctrl+Alt+U` again.

Selecting scenarios to copy

Once you create a batch substitution template, the Scenario menu item perform batch substitution will be avaiable. Select it or type `Ctrl+Alt+U` to display the Select scenarios for substitution dialog.


Select scenarios for substitution

Select scenarios for substitution

	scenario name	last modified
☐	results_scenario_1	6/12/12 3:40 PM
☐	results_scenario_2	7/2/12 3:07 PM

[✓ Make copies of scenarios] Overwrite existing scenarios

Scenario name suffix : _20120709_1451_58_993

Batch substitution target dialog

This dialog displays a list of all scenarios associated with the life cycle model. Check the boxes beside the names of the scenarios you want to copy. When the substitution executes, a copy of each selected scenario will be created. The copy will be identical to the original, except for the parameters defined in the template; these will be configured exactly as the template is configured.

Make copies of scenarios / Overwrite existing scenarios : The default batch substitution operation, as described above, is to copy each of the existing scenarios, with the desired substitution appearing in the copy. If you select the Overwrite existing scenarios option, no copy will be made. Instead, the target scenario will have its element configurations updated to match the selected template configurations.

Scenario name suffix : The names of the new scenario copies will be generated by appending the contents of this field to existing scenario names. The default value is a millisecond resolution timestamp designed more for its utility in avoiding name duplication than for its readability. Feel free to select your own scenario-copy identifier, but try to avoid generating names that are already in use.

Check all / Clear all : Click these buttons to select all or none of the scenarios as substitution targets.

Click Proceed to carry out the batch substitution or Cancel to close the dialog without performing the substitution.

Whether you cancel or execute the substitution, the   perform batch substitution   menu item remains visible until you load or create a new scenario or life cycle model.

When the substitution finishes, the template remains in the scenario editor (and you can re-define and run more substitutions if you like). Newly-created scenarios will appear in the scenario list displayed when you select the   load scenario   menu item. Once you load a scenario or a life cycle model, the   perform batch substitution   menu item will be replaced by   define batch substitution   and you'll have to create a new template before making any further substitutions.

Simulations in SLAM

Running simulations is the main reason SLAM exists. Life cycle model and scenario development tools allow you to create the inputs for simulations, and the dataset-viewing and analysis tools allow you to evaluate the results of simulations.

Simulations are collections of *trajectories*: a trajectory is a single run of the model from beginning to end. The ability to run multiple trajectories allows you to analyze the results of your simulations statistically.

Simulation results are time series of abundances for all the stages in your life cycle model, for each year of the simulation and for each individual trajectory in the simulation. In addition to examining data in the results viewer, you can save entire simulation result sets to files. You can also export specific sections of the data sets from the results viewer, along with images of graphical displays of the data.

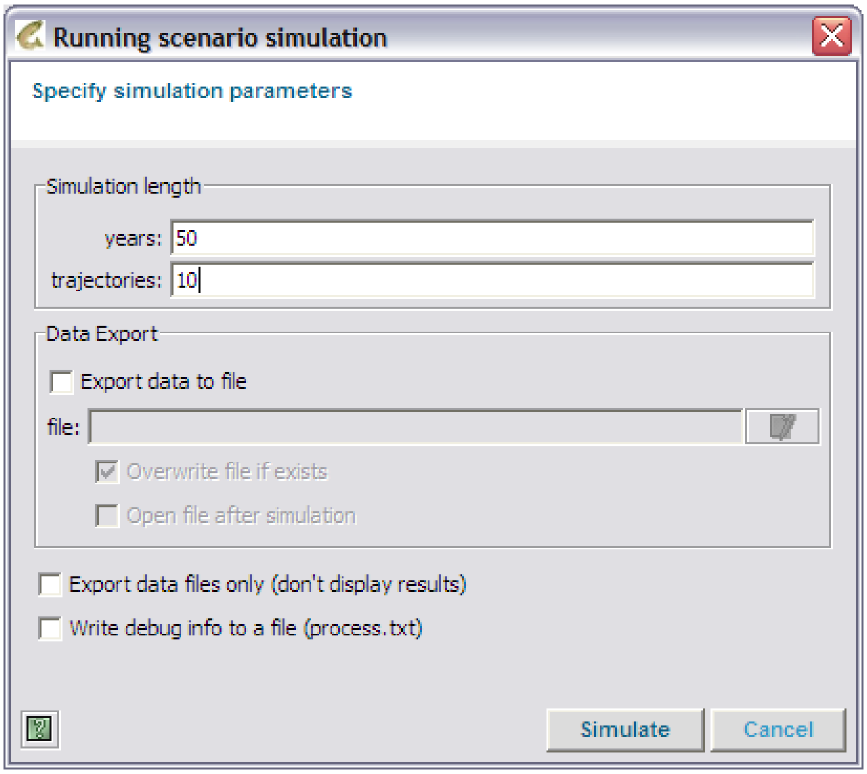
- Basic simulations
- One-at-a-time simulations
- Batch simulations
- Simulation results viewer

Running a simulation in SLAM

Before you run a SLAM simulation, you must

- Create or import a life cycle model: This defines the life stages of your model and describes how those stages succeed one another throughout the biological life cycle you are modeling.
- Create or import a scenario for the life cycle model: The scenario fills in the functional details of transitions among life stages; these are the mathematical models invoked during the course of the simulation run.

Once you've chosen a life cycle model and scenario, you're ready to start a simulation. Select run simulation under the 'Scenario' menu, type Ctrl+R, or, with the 'scenario' tab selected, click the 'run simulation' button on the left-hand toolbar. The Running scenario simulation window appears:



Running scenario simulation

Specify simulation parameters

Simulation length

years: 50

trajectories: 10

Data Export

☐ Export data to file

file:

☑ Overwrite file if exists

☐ Open file after simulation

☐ Export data files only (don't display results)

☐ Write debug info to a file (process.txt)

Simulate | Cancel

Simulation run dialog

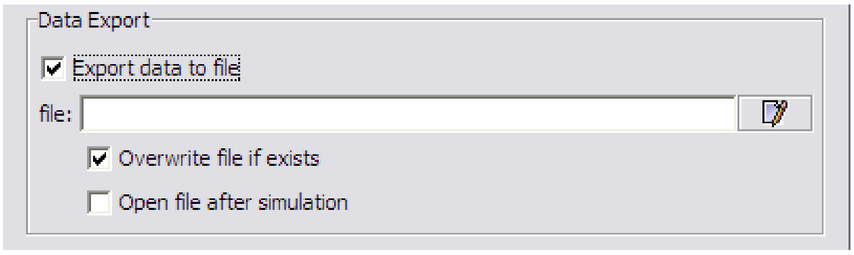
Simulation length

Years: The number of years for which a single pass through the simulation should run.
Trajectories: The number of complete runs through the simulation. The duration of each trajectory is the number of years specified above.

Data Export

Specify how to save simulation results to a file

Export data to file: Check this box in order to save all simulation data to a file. (That there are additional data-saving options available after the simulation completes, in the simulation results viewer.) When the box is checked, the export controls are enabled as shown:


Data Export

[✓] Export data to file

file:

[✓] Overwrite file if exists

[ ] Open file after simulation

Simulation data export controls

file: Click the edit icon to specify the directory and file name for your exported data. The full path to the file will appear in the edit control.

The output file is in a custom SLAM format and is not automatically assigned an extension.

Overwrite file if exists: If this box is checked any existing file with the given name and path will be overwritten during the simulation run. If the box is not checked, attempting to overwrite an existing file will produce an error message, and the simulation will not run.

Open file after simulation: SLAM enables you to examine exported simulation results in the same way that you can examine fresh results held in memory immediately following the simulation. If you check this box, SLAM will open the exported file and read its data in for display rather than using the in-memory results.

You can also display exported data files in the results viewer.

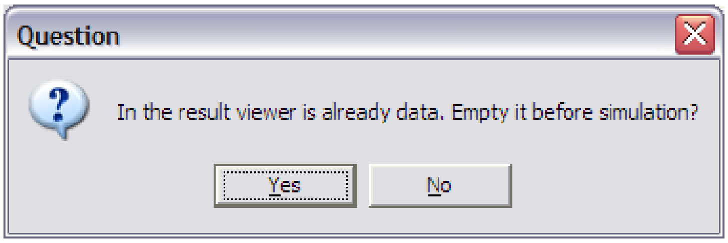
Export data files only (don't display result): If you have specified an export file for the simulation data, checking this box will tell SLAM to run the simulation without opening the results once the simulation is complete.

Write debug info to a file (process.txt): Checking this box results in the output of a file containing details of processing during the simulation run. The file, process.txt is written to the directory containing the SLAM runtime files.
It is suggested that you do not check this box.

When the simulation-run parameters are configured to your liking, click the Simulate button to start the simulation or click Cancel to close the window without running a simulation.

If the simulation data is being exported to a file and would result in overwriting existing data and you have not given SLAM permission to overwrite, an error message will appear and the run will be cancelled.

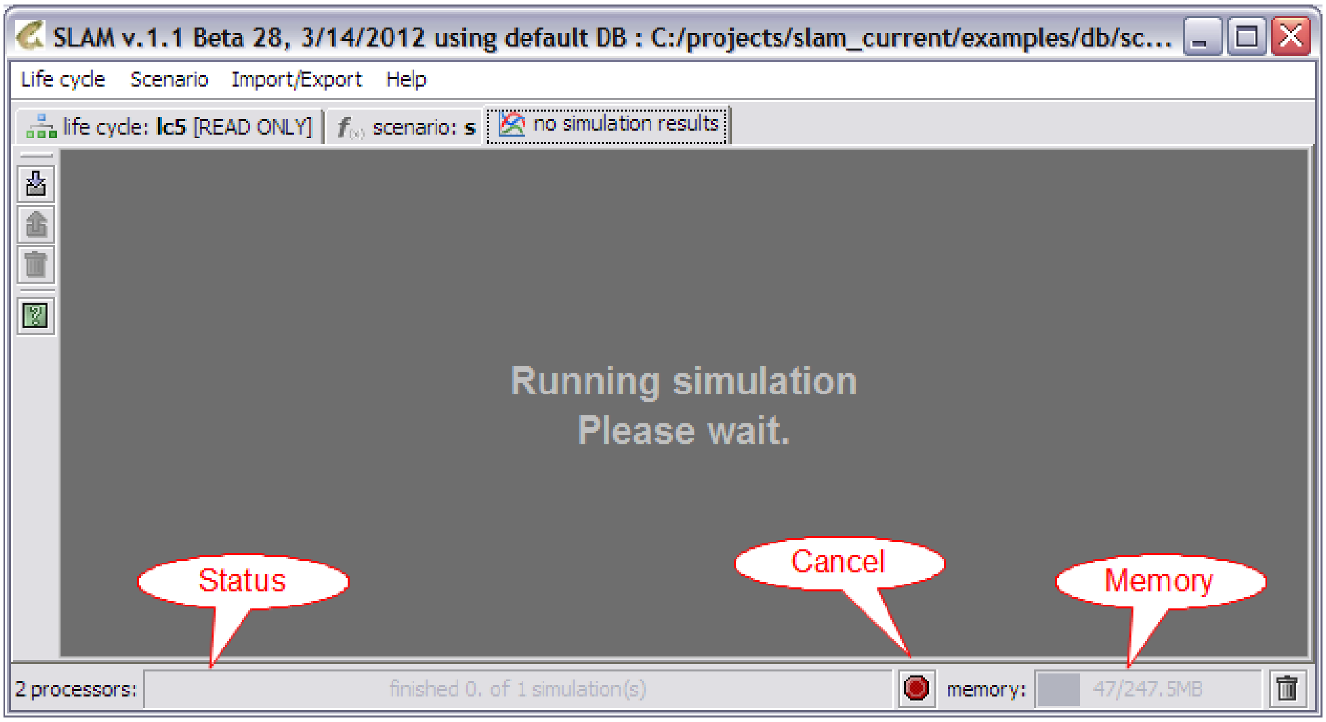
If there is already simulation data in the results display, you will see the following dialog:


Question
In the result viewer is already data. Empty it before simulation?
Yes
No

Clear simulation results query

The simulation results viewer can display multiple result sets; if you want to view the results of the new simulation along with those already in the viewer, click No . If you want to clear any existing results and display only those you're about to generated, click Yes .

Finally, the simulation begins running. The display looks like this:


SLAM v.1.1 Beta 28, 3/14/2012 using default DB : C:/projects/slam_current/examples/db/sc...
Life cycle
Scenario
Import/Export
Help
life cycle: lc5 [READ ONLY]
scenario: s
no simulation results
Running simulation
Please wait.
Status
Cancel
Memory
2 processors:
finished 0. of 1 simulation(s)
memory:
47/247.5MB

Simulation in progress

- **Status**: This message bar tells you how many simulations have been completed. If you are running a single simulation (see also OAT simulations and batch simulations), then the message displayed will be finished 0 of 1 simulations until the simulation is done; then the message reverts to the usual No active simulation .
- **Cancel**: Click the red cancel button in order to halt a running simulation. Clicking it once is enough, though there will be some delay before SLAM halts the simulation in progress.
- **Memory**: This field displays SLAM's memory allocation and available memory. If the left-hand number approaches the right-hand number, SLAM is running out of memory.

When the simulation completes, the simulation results viewer will open automatically unless you are exporting the results and have checked the Export data files only check box.

Running One-At-a-Time (OAT) simulations in SLAM

The One-At-a-Time (OAT) simulation mode helps you to evaluate how sensitive your simulation results are to the sources of variability you have built into your model. It does this by running a group of simulations in which selected variability sources are turned off and on; you can then examine and compare the outcomes of these simulations in the results viewer.

How OAT works

One way to estimate the sensitivity of abundance values in your life cycle model to the value of a particular parameter (*e.g.*, a transition function or split stage ratio) is to vary the parameter randomly over the trajectories in a simulation and then to see how much the resulting abundances vary over the trajectories. If the model abundances are very sensitive to the parameter, then they will vary over a wide range; if they don't depend strongly on the parameter, their range of variation will be smaller.

Usually there are several sources of variability in your model, which can make it difficult to determine why an abundance changes from trajectory to trajectory. To narrow things down, you could design a set of scenarios for the same model in which different parameters were made variable while the others are kept constant (or allowed to vary in some well-understood way). You could then run a simulation for each scenario and compare the results to see how varying parameter values translates into variation in the abundances you're interested in.

Creating a large number of test scenarios, running them separately, and consolidating the results in order to compare them is a lot of work. SLAM's OAT simulation automates this process for you. It allows you to define which parameters you want to isolate, how you want to control the values of those parameters, and how you want to constrain the values of the rest of the parameters in the simulation during the test runs. Once you've created a set of definitions for parameter variation, OAT defines a corresponding set of simulations and runs them so that you can compare the results.

- Configuring OAT
  - OAT observed parameters
  - OAT basic parameters
  - Counting OAT simulations

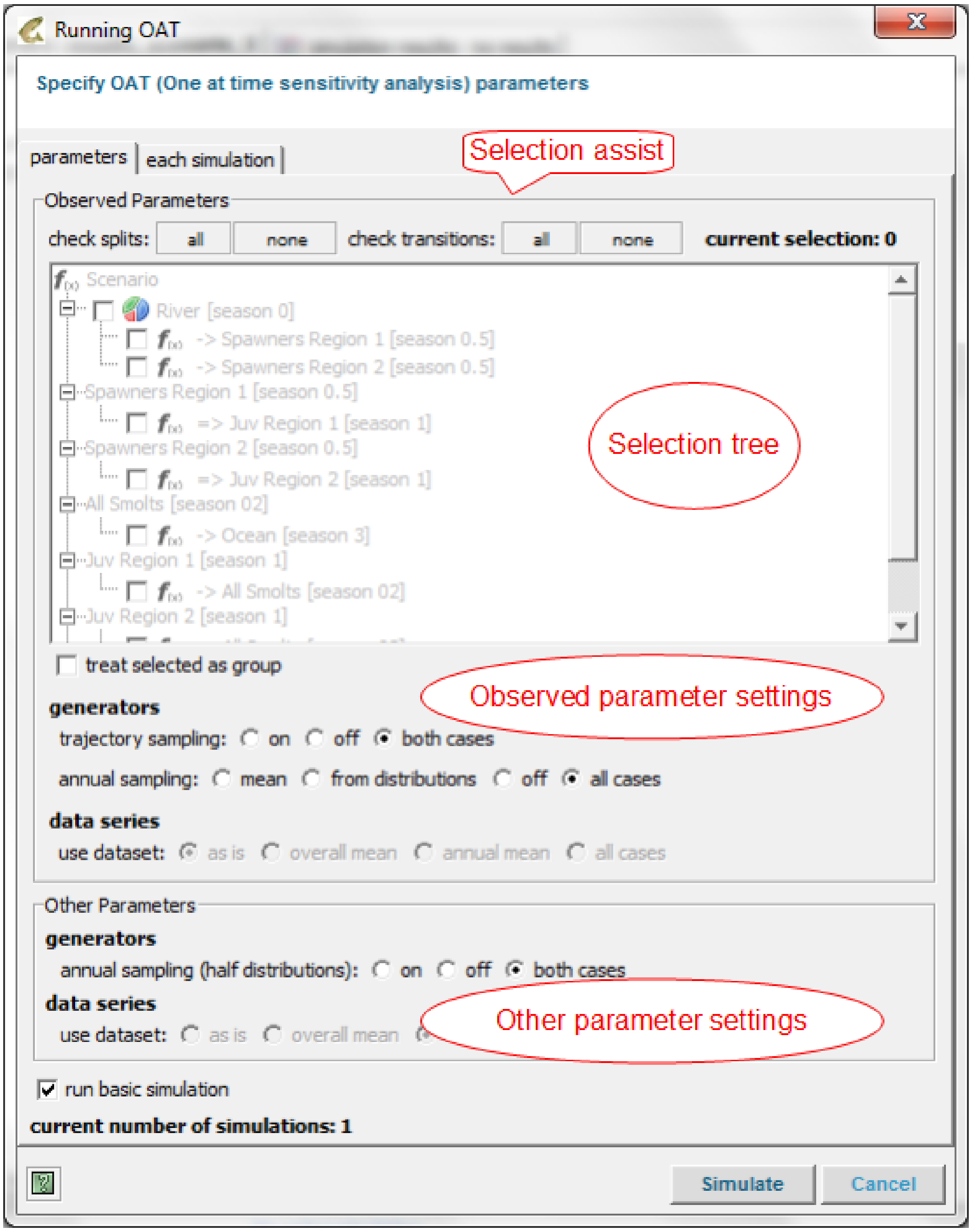
Configuring OAT

Once you've loaded or created a life cycle model and a scenario, begin the configuration of an OAT simulation by selecting the Run OAT option from the Scenario menu, type Ctrl+Alt+R, or click the run OAT button on the left-hand toolbar>. (The toolbar option is available only when the scenario tab is selected.) The Running OAT configuration dialog will appear.

The dialog contains two tabs labeled parameters and each simulation . In the parameters tab you select the parameters you want to observe and specify how you want to treat their variability sources. This determines how many simulations will be run and how those simulations will differ from one another. In the each simulation tab you specify general simulation parameters for the run.

Configuring OAT observed and other parameters

The Parameters Tab


Running OAT

Specify OAT (One at time sensitivity analysis) parameters

parameters | each simulation

Observed Parameters

check splits: all none check transitions: all none **current selection: 0**

- Scenario
  - ☐ River [season 0]
    - ☐ -> Spawners Region 1 [season 0.5]
    - ☐ -> Spawners Region 2 [season 0.5]
  - Spawners Region 1 [season 0.5]
    - ☐ => Juv Region 1 [season 1]
  - Spawners Region 2 [season 0.5]
    - ☐ => Juv Region 2 [season 1]
  - All Smolts [season 02]
    - ☐ -> Ocean [season 3]
  - Juv Region 1 [season 1]
    - ☐ -> All Smolts [season 02]
  - Juv Region 2 [season 1]

☐ treat selected as group

generators

trajectory sampling: ○ on ○ off [✓ both cases]

annual sampling: ○ mean ○ from distributions ○ off [✓ all cases]

data series

use dataset: [✓ as is] ○ overall mean ○ annual mean ○ all cases

Other Parameters

generators

annual sampling (half distributions): ○ on ○ off [✓ both cases]

data series

use dataset: ○ as is ○ overall mean

☑ run basic simulation

current number of simulations: 1

Simulate Cancel

OAT parameter configuration

In the parameters tab you design the simulation set that OAT will run. It is divided into two main sections:
In Observed parameters you identify and configure the particular parameters you are examining.
In Other parameters you define how you want the rest of the model parameters to behave during simulations that isolate a particular observed parameter.

Observed parameters

- **Selection assist** : These controls enable you to make quick group selections in the selection tree. Click the all or none button to the right of check splits in order to select or unselect all split stages in the model. Similarly, use the buttons to the right of check transitions to select and unselect all transitions.
  The current selection count is the number of stages and transitions you have selected in the selection tree.
- **Selection tree** : In this tree control you identify the split stages and transitions you want to treat as observed parameters. Note that as you select new items in the tree, the current selection count above the tree increments; more importantly, the current number of simulations count at the bottom of the window also changes.
- **Observed parameter settings** : These controls define how you want to treat the values of the observed parameters you have selected in the selection tree.

  If the **treat selected as group** check box is not selected, a separate set of simulations is generated in which each item checked in the selection tree is treated as an observed parameter independently, while all other items (checked or not), are treated as other parameters.

  If this box is checked, then a single set of simulations is created in which all the selected items are treated as observed parameters while all unselected items are other parameters.

  Under **generators** there are two groups of variability controls:

  trajectory sampling : These are variability sources that are sampled at the beginning of each trajectory. They include randomly-generated parameters in transition functions and trajectory uncertainty noise added to split ratios.

  There are three options:
  on : Values will be drawn from the distribution(s) specified in the scenario, as usual.
  off : The value assigned to a randomly-generated parameter will be instead the mean value of the distribution assigned to that parameter.
  both cases : Two simulations will be created for each case, one with values drawn as usual from specified distributions (the on case) and one with values equal to distribution means (the off case).

  annual sampling : These settings apply to annual variability sources in transitions and split stages.

  There are four options:
  mean : Annual variability effects will be included, but the value applied will be the mean value of the distribution that characterizes the annual variability.
  from distributions : Annual variability will be randomly generated from specified distributions, as usual.
  off : No variability will be applied.
  all cases : Three simulations will be created for each case, one with variable values set to distribution means, one with values drawn as usual from distributions, and one with variability not applied.

Other Parameters

During each simulation, all transitions and split stages aside from the observed parameters are classified as *other parameters*. You can specify how you want to treat the variability associated with these parameters, as well.

- **annual sampling (half distributions)** : This controls whether variability is applied to all of the ‘other parameters’ or not. Options are the same as for observed-parameter trajectory sampling:

  on : Values will be drawn from the distribution(s) specified in the scenario, as usual.
  off : The value assigned to a randomly-generated parameter will be instead the mean value of the distribution assigned to that parameter.
  both cases : Two simulations will be created for each case, one with values drawn as usual from specified distributions (the on case) and one with values equal to distribution means (the off case).

Including a basic simulation

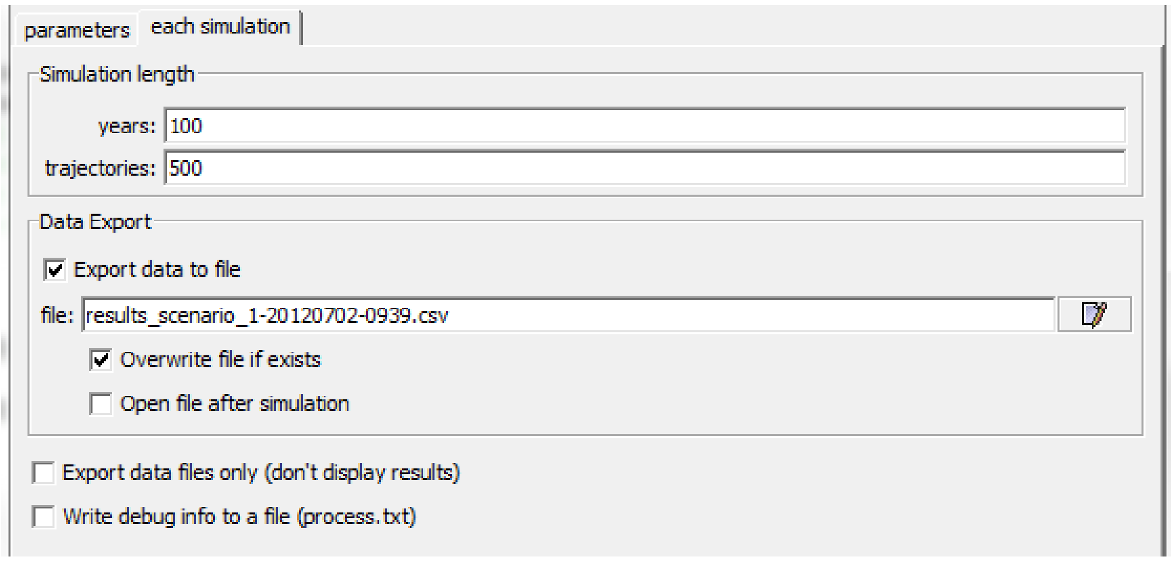
Below the Other parameters controls there is a check box labeled run basic simulation . If you select this box, a standard simulation will be run in addition to the OAT simulations you have defined in the controls described above.

current number of simulations

This field displays the total number of simulations that will be run given the settings you have specified in the configuration controls. It is not directly editable — it updates automatically each time you change your parameter selections.
Remember that this is a total *simulation* count. If you are going to run ten simulations and you specify 500 trajectories for each of them on the each simulation tab, then a total of 5000 trajectories will be run.
For more information on how OAT counts simulations, see counting simulations.

Configuring OAT basic parameters

The Each Simulation Tab



Simulation length

years: 100

trajectories: 500

Data Export

☑ Export data to file

file: results_scenario_1-20120702-0939.csv

☑ Overwrite file if exists

☐ Open file after simulation

☐ Export data files only (don't display results)

☐ Write debug info to a file (process.txt)

Basic simulation configuration

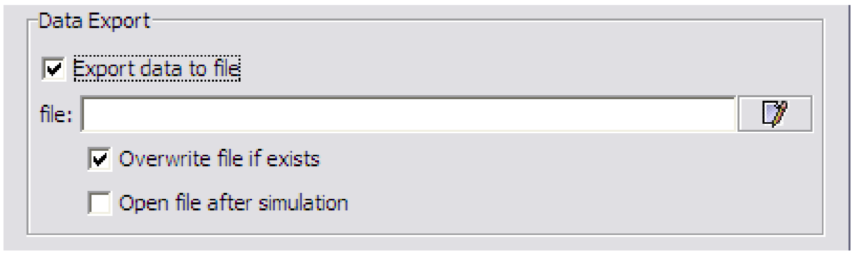
In this tab, you configure parameters of the simulation that aren't specific to OAT runs but apply to all simulations. The controls are similar to those in the Running scenario simulation dialog.

Simulation length

years : The number of years for which a single pass through the simulation (*i.e.*, a trajectory) should run.
trajectories : The number of complete runs through the simulation. The duration of each trajectory is the number of years specified above.

Data Export

Specify requirements for saving the data to a file
Export data to file: Check this box in order to save all simulation data to a file. When the box is checked, the export controls are enabled as shown:



Data Export

☑ Export data to file

file:

☑ Overwrite file if exists

☐ Open file after simulation

Simulation data export controls

file: Click the edit icon to open a file-browsing dialog with which you can select a directory and file name for your exported data. The full path to the file will appear in the edit control.

Overwrite file if exists: If this box is checked any existing file with the given name and path will be overwritten during the simulation run. If the box is not checked, attempting to overwrite an existing file will result in an error message being displayed, and the simulation will not run.

Open file after simulation: SLAM enables you to examine exported simulation results in the same way that you can examine fresh results held in memory immediately following the simulation. If you check this box, SLAM will open the exported file and read its data in for display rather than using its in-memory copy of the results.

Export data files only (don't display result): If you have specified an export file for the simulation data, checking this box will tell SLAM to run the simulation without opening the results once the simulation is complete. Recall that you can open exported simulation data directly from the simulation results window.

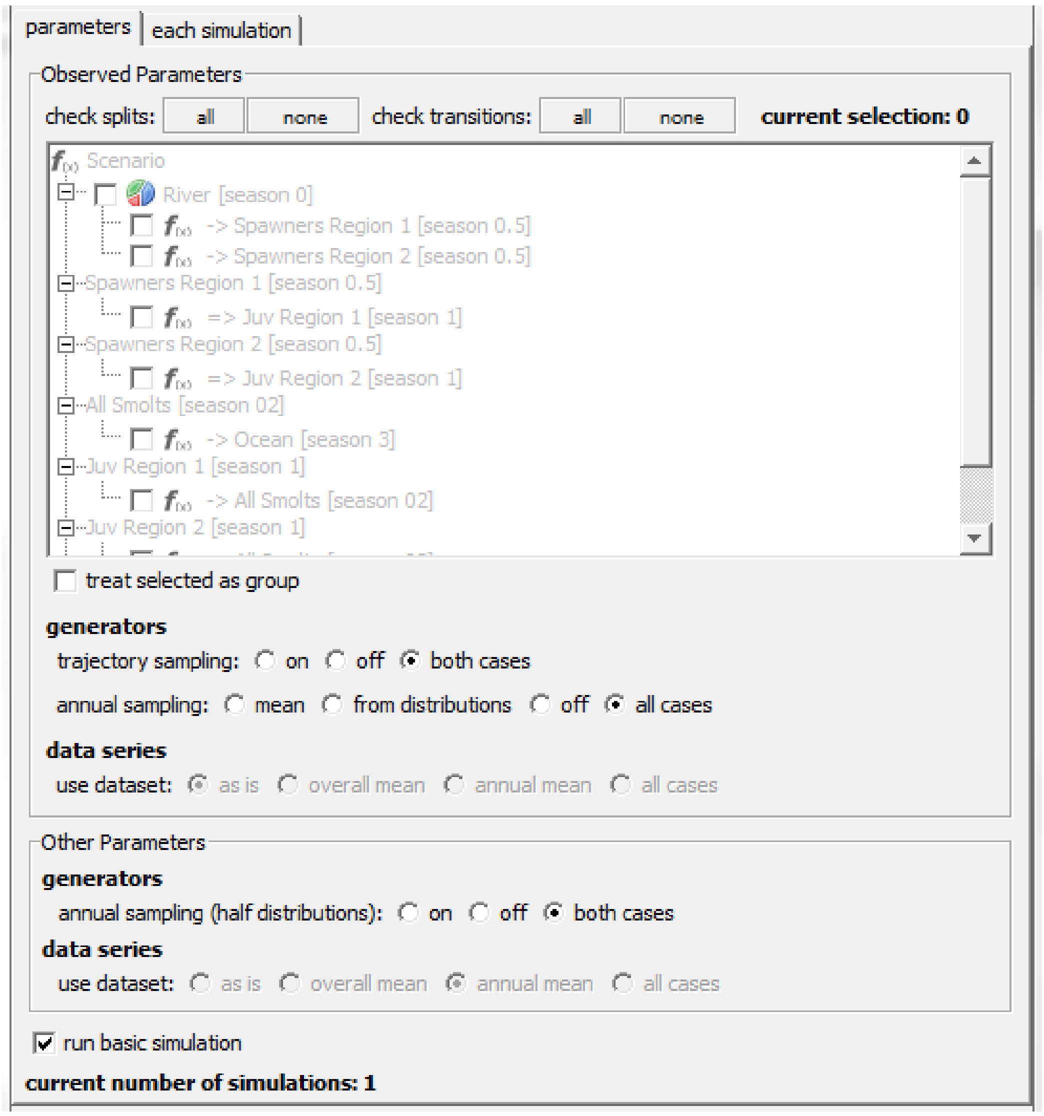
Write debug info to a file (process.txt): Checking this box results in the output of a file containing details of processing during the simulation run. The file, process.txt is written to the directory containing the SLAM runtime files.
It is suggested that you do not check this box.

Counting OAT simulations

The basic SLAM simulation is a single simulation consisting of multiple trajectories, each running over the specifiied number of years. OAT simulations are actually sets of simulations. If your basic model simulation takes a significant length of time to run, it's a good idea to remain aware of how many simulations your OAT configuration will be running.

The current number of simulations display in the OAT parameter configuration interface tells you how many simulations you've told OAT to run. For controlling this number, it may be helpful to understand how OAT generates new simulations based on its configuration.

Adding them up



current selection: 0

Scenario
- [ ] River [season 0]
  - [ ] -> Spawners Region 1 [season 0.5]
  - [ ] -> Spawners Region 2 [season 0.5]
- Spawners Region 1 [season 0.5]
  - [ ] => Juv Region 1 [season 1]
- Spawners Region 2 [season 0.5]
  - [ ] => Juv Region 2 [season 1]
- All Smolts [season 02]
  - [ ] -> Ocean [season 3]
- Juv Region 1 [season 1]
  - [ ] -> All Smolts [season 02]
- Juv Region 2 [season 1]

[ ] treat selected as group

generators

trajectory sampling: on off [✓ both cases]

annual sampling: mean from distributions off [✓ all cases]

data series

use dataset: [✓ as is] overall mean annual mean all cases

Other Parameters

generators

annual sampling (half distributions): on off [✓ both cases]

data series

use dataset: as is overall mean [✓ annual mean] all cases

[✓] run basic simulation

current number of simulations: 1

OAT parameters

If you have selected no stages or transitions in the selection tree and the basic simulation box is not checked, then no simulations will be run.

If you check a single stage or transition in the selection tree, the number of simulations added to the count depends on the Observed parameters and Other parameters options you select. Checking more than one stage or transition multiplies the number of simulations by the number of selected elements.

Each parameter setting includes single and multiple simulation modes. If you select the single option, then every simulation in the set will be configured according to this option. If you select the multiple option, an additional group of simulations will be created for each possible value of the parameter. Counting the simulations is a matter of multiplying out the number of combinations implied by your configuration parameter specifications.

For example, Observed parameters trajectory sampling can be configured as on , off , or both . If you select on or off , every simulation in the set will be run with that configuration. Select both and two simulations will be run, identically configured except that one has trajectory sampling turned on while the other runs with trajectory sampling turned off. Thus the both option doubles the number of simulations run.

The possible multipliers are show below:

Config option	**Value**	**Multiplier**
	Observed parameters	
trajectory sampling	on	1
trajectory sampling	off	1
trajectory sampling	both cases	2
annual sampling	mean	1
annual sampling	from distributions	1
annual sampling	off	1
annual sampling	all cases	3
	Other parameters	
annual sampling	on	1
trajectory sampling	off	1
trajectory sampling	both cases	2

To count the number of OAT simulations in your set, multiply together the values in the third column of the table that correspond to your parameter settings, then multiply by the number of elements you have selected in the selection tree.

If the treat selected as group box is checked, then options are not varied independently for each observed parameter, and you can omit the final multiplication.

Finally, if you check the run basic simulation box, this adds one more simulation to the total.

Example

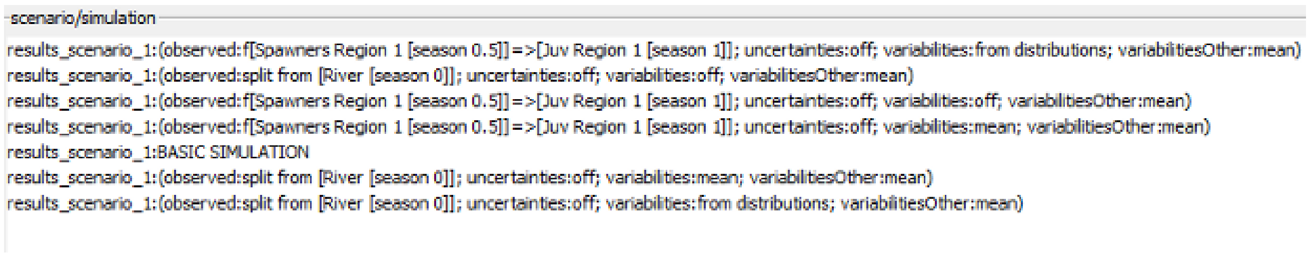
By default, the multiple values are selected for each option, as is the basic simulation option. If you check a single life cycle element, the total default simulation count is 2 x 3 x 2 + 1 or 13 simulations.

OAT Simulations in the Results Viewer

When you click the Simulate button in the OAT configuration dialog, the simulation set begins to run. You can monitor progress in the status bar at the bottom of the SLAM window. This tallies the number of simulations completed versus the total number to be run.

As simulations begin to finish, their results will be displayed in the results viewer before the complete OAT simulation set has been executed. (When all the simulations in the set are done, the status bar message reads Finished .)

Simulation results are listed in the simulation list pane of the results viewer:



```
scenario/simulation
results_scenario_1:(observed:f[Spawners Region 1 [season 0.5]] =>[Juv Region 1 [season 1]]; uncertainties:off; variabilities:from distributions; variabilitiesOther:mean)
results_scenario_1:(observed:split from [River [season 0]]; uncertainties:off; variabilities:off; variabilitiesOther:mean)
results_scenario_1:(observed:f[Spawners Region 1 [season 0.5]] =>[Juv Region 1 [season 1]]; uncertainties:off; variabilities:off; variabilitiesOther:mean)
results_scenario_1:(observed:f[Spawners Region 1 [season 0.5]] =>[Juv Region 1 [season 1]]; uncertainties:off; variabilities:mean; variabilitiesOther:mean)
results_scenario_1:BASIC SIMULATION
results_scenario_1:(observed:split from [River [season 0]]; uncertainties:off; variabilities:mean; variabilitiesOther:mean)
results_scenario_1:(observed:split from [River [season 0]]; uncertainties:off; variabilities:from distributions; variabilitiesOther:mean)
```

Example OAT results in the simulation list

The automatically-generated result set names identify the simulations in terms of the option values applied to them. As an example, the first line of the display pictured above reads:

```
    results_scenario_1:(observed:f[Spawners Region 1[season 0.5]]=>[Juv Region 1[season 1]];
    uncertainties:off, variabilities:from distributions; variabilitiesOther:mean
```

For the scenario called results_scenario_1 , this result set is the outcome of a simulation in which the Spawners Region 1 to Juv Region 1 transiton was configured with Observed parameters trajectory sampling (uncertainties) set to off and annual sampling (variabilities) set to drawn from distributions; and Other parameters annual sampling (variabilitiesOther) set to mean.

Or, in summary:

scenario : results_scenario_1
element : f[Spawners Region 1[season 0.5]]=>[Juv Region 1[season 1]]
Observed parameters trajectory sampling : uncertainties:off
Observed parameters annual sampling : variabilities:from distributions
Other parameters annual sampling : variabilitiesOther:mean

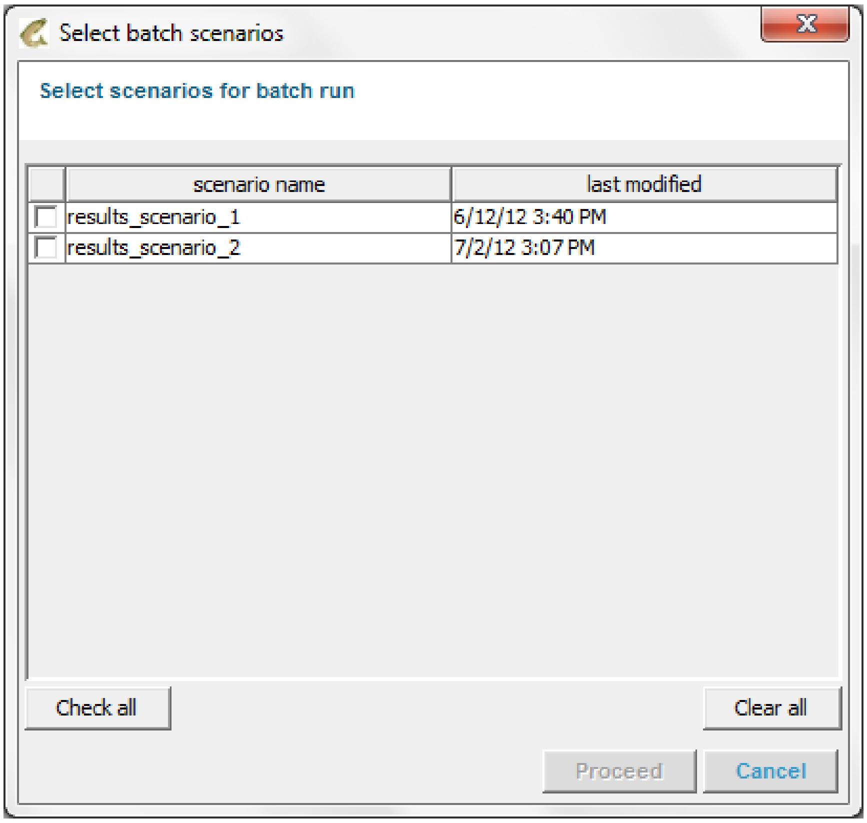
The results of the basic simulation, if it is run, are labeled in the standard way: results_scenario_1:BASIC SIMULATION .

You can display results singly or in combination using the results viewer.

Batch Simulations

If you want to simulate several scenarios in SLAM, it may not be convenient to remain at the computer in order to start each successive run when the previous one finishes. In order to avoid this, SLAM provides a means of queuing up multiple simulations and running them consecutively, without user intervention.

To configure a simulation batch, select the run batch item from SLAM's Scenario menu or type Ctrl+Alt+B. The Select batch scenarios dialog appears:

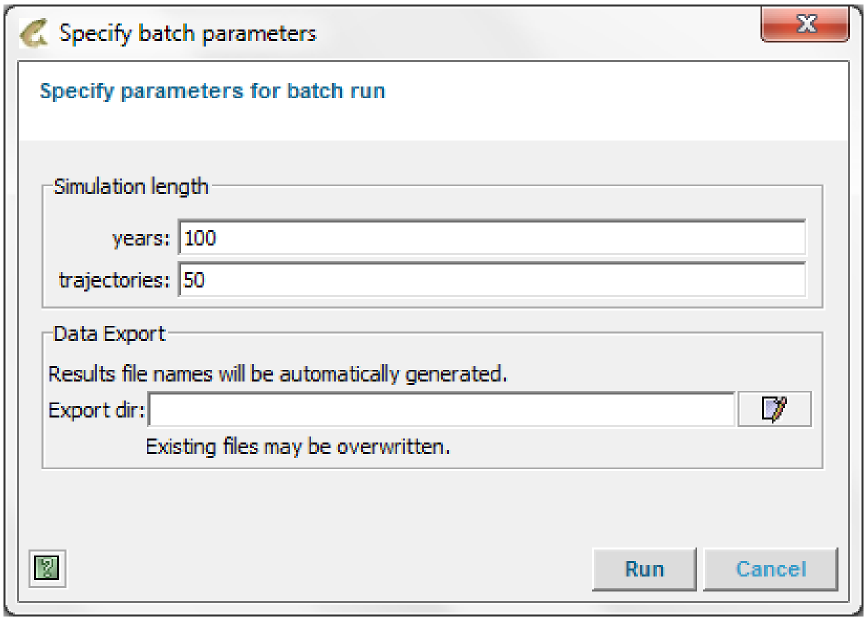


	scenario name	last modified
☐	results_scenario_1	6/12/12 3:40 PM
☐	results_scenario_2	7/2/12 3:07 PM

Selecting scenarios for batch simulations

Select the check boxes next to the names of the scenarios you want to include in the simulation batch. Clicking the Check all button will select all the scenarios in the list, while clicking Clear all will clear all of your selections.

When you have selected the scenarios for batch run, click the Proceed button. The Specify batch parameters dialog appears:


Specify batch parameters

Specify parameters for batch run

Simulation length

years: 100

trajectories: 50

Data Export

Results file names will be automatically generated.

Export dir:

Existing files may be overwritten.

Configuring simulation batch parameters

In this dialog, you specify general parameters that will characterize each simulation in the batch run.

Simulation length : These two parameters determine the overall extent of each simulation in the batch.

- **years**: The value in this field is the number of years in each trajectory.
- **trajectories**: This field holds the number of complete trajectories making up the complete simulation run.

Data export : All batch simulation results are automatically exported. In the Export dir field, select the directory into which you want the results files to be written.

Names of the output result files are automatically generated. They have the form `simname-timestamp.csv`. `simname` is the name of the scenario being simulated. `timestamp` is formatted as `yyyymmdd-hhmm` where `yyyy` is the year, `mm` is the (numeric) month, `dd` is the day of the month, `hh` is the current time hour, and `mm` is the current time minute. Timestamps are included in the generated file name in order to try to avoid overwriting previous results.

By default, batch results are written to the SLAM batch simulations directory. You can select any directory you like, but be sure your user account can write to the directory you choose.

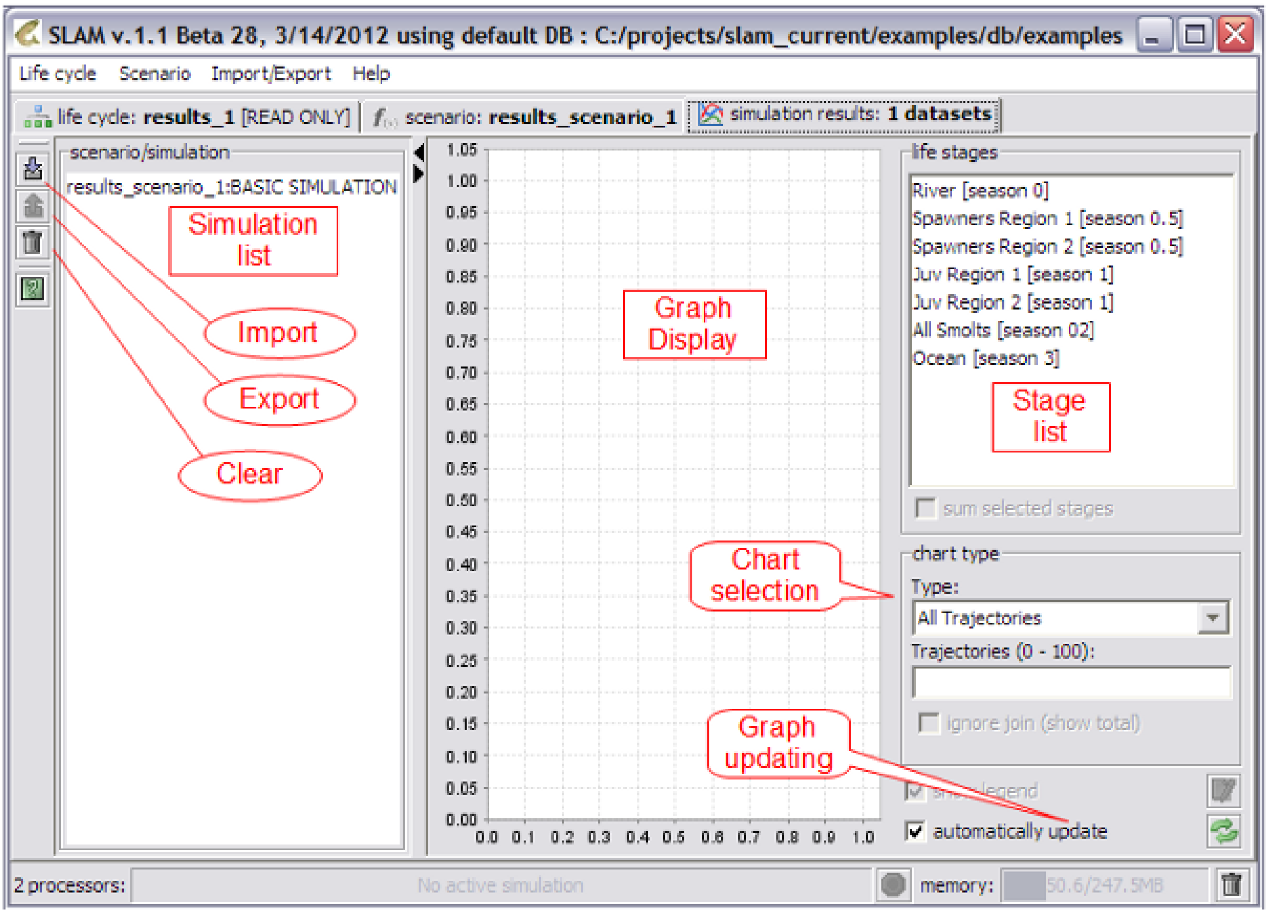
When the simulation parameters and output location are set, click Run to begin execution of the batch (or Cancel to close the dialog without executing the batch). The simulation progress is shown in the simulation status bar at the bottom of the main SLAM window. This display shows you how many simulations have finished versus the total number to be run.

When the simulation completes, results will be displayed in the results viewer. Note that only a single set of results will be loaded automatically; you can examine other result sets by using the results viewer import button.

Simulation results viewer

The simulation results viewer is always accessible in SLAM — you open it by clicking the simulation results tab. The results viewer will also open when a simulation run completes, unless you have configured the run to export the simulation results and not to display results on completion.

When the results viewer opens after the completion of a simulation run, and before you have selected any results for graphing, you'll see a display like this:



Simulation results viewer

Simulation list : The results viewer can display more than one set of results at a time. Each result set, whether generated by a just-finished simulation run or imported from a file, is displayed here. The simulation label is the concatenation of the scenario name and the simulation type.

Select one or more simulation result sets from the list (along with one or more stages) in order to view plots of the data.

Graph display : This is the area where graphs of results will appear when you select simulations and stages.

Stage list : Each stage in the life cycle (and each dynamic driver) is listed here. Select one or more stages along with one or more simulations from the simulation list in order to view data plots.

Import : Click to load previously–exported simulation results into the viewer.

Export : Click to write selected results to a CSV (comma-separated-variable) file. (*You can read CSV files into spreadsheet programs.*)

Clear : Click to clear all data from the viewer. The data will be discarded — if you have not exported the results to a file, you'll have to re-run the simulation in order to rebuild the data.

Chart selection : From the drop-down list, select the type of chart to be displayed for the data you have selected. Parameters for adjusting the plot appearance will be displayed, depending on the chart type.

Automatically update : If the check box is selected, the graph display will change whenever as you make changes either to the chart type or to the data selected. If the check box is cleared, then the graph will only be redrawn when you click the Update button: .

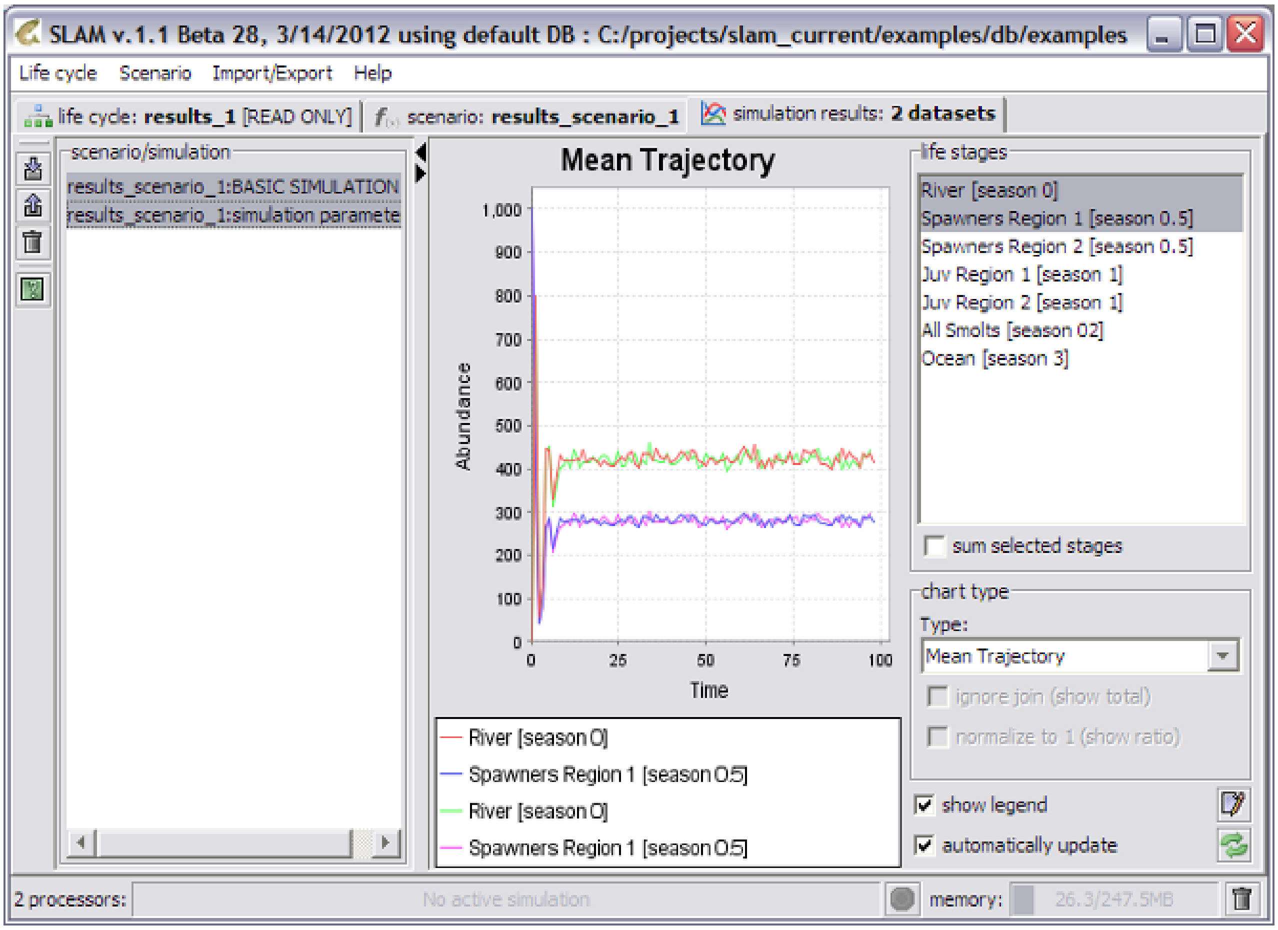
See Examining simulation data for more on using the results viewer to inspect the outcomes of simulations.

Examining simulation data

Display the trace of a single life stage's data by selecting (single-clicking) a simulation in the simulation list and a stage in the stage list.

You can select more than one stage from more than one simulation data set by holding down the Ctrl key as you click the desired simulations or stages. Clicking a selected item with the Ctrl key depressed will de-select the item.

Below, two stages are selected in each of two simulations. Four data sets are plotted in the graph display.



SLAM v.1.1 Beta 28, 3/14/2012 using default DB : C:/projects/slam_current/examples/db/examples
Life cycle
Scenario
Import/Export
Help
life cycle: results_1 [READ ONLY]
scenario: results_scenario_1
simulation results: 2 datasets
scenario/simulation
Mean Trajectory
Abundance
Time
life stages
chart type
Type:
2 processors:
No active simulation
memory:
26.3/247.5MB

results_scenario_1:BASIC SIMULATION
results_scenario_1:simulation paramete

River [season 0]
Spawners Region 1 [season 0.5]
Spawners Region 2 [season 0.5]
Juv Region 1 [season 1]
Juv Region 2 [season 1]
All Smolts [season 02]
Ocean [season 3]

☐ sum selected stages

Type: Mean Trajectory

☐ ignore join (show total)
☐ normalize to 1 (show ratio)
☑ show legend
☑ automatically update

— River [season 0]
— Spawners Region 1 [season 0.5]
— River [season 0]
— Spawners Region 1 [season 0.5]

Multiple data selection

Plots are color-coded. The colors are identified by default in the legend below the graph display. (You can customize or hide the legend.)

If you have selected more than one stage for display, you can plot either the abundances for each selected stage, as shown above, or the sum of the stage abundances; display the sums by checking the sum selected stages check box below the stage list.

Data are grouped by simulation for summing — data from different selected simulations are not combined. If the sum selected stages box is checked, you'll see one line for each simulation, representing the sum of all selected stage data for that simulation.

With data selected for plotting, you can

- Select chart types
- Customize chart legends
- Export selected data to CSV files

Results viewer chart types

In the Chart selection section of the results viewer, you can change the type of chart and the number of trajectories displayed in the graph display. Controls for setting parameters specific to different chart types appear when those types are selected.

These chart types are available:

- All Trajectories
- Mean Trajectory
- Median Trajectory
- Abundance histogram
- Quasi-extinction probability
- Probability of exceeding threshold
- Stage productivity

Chart type : All Trajectories

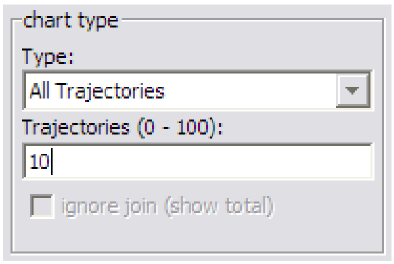


chart type

Type:

All Trajectories

Trajectories (0 - 100):

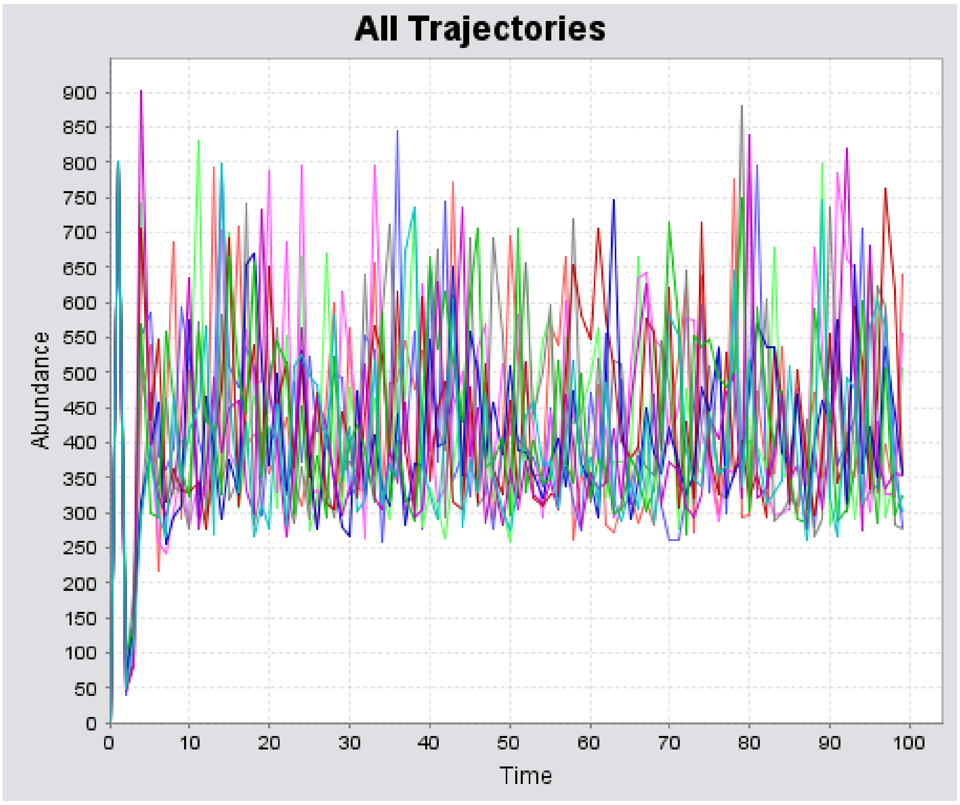
10

All Trajectories

By default, the All Trajectories chart type displays the simulation results corresponding to selected stages for each trajectory of the simulation run. You can restrict the display to the first **N** trajectories by entering **N** into the Trajectories field beneath the drop-down list. Leaving the field blank or entering a **0** will result in all trajectories being displayed. If you enter an invalid value for **N** (less than 0 or greater than the total number of trajectories), the field is highlighted in red to indicate an error.

The chart legend is not displayed when All Trajectories is selected.

All trajectories plot



All trajectories plot, **N**=10

Chart type : Mean Trajectory

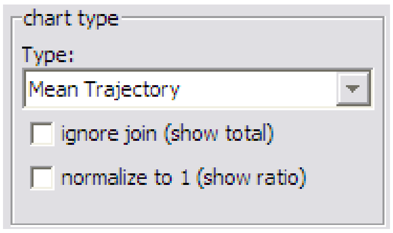


chart type

Type: Mean Trajectory

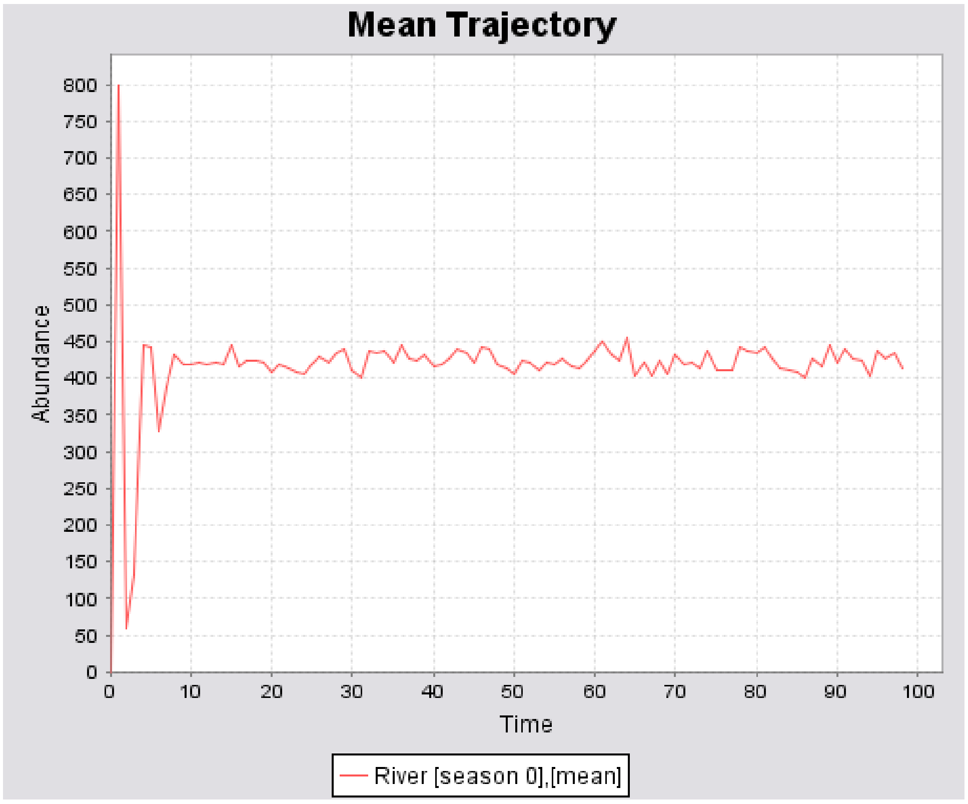
☐ ignore join (show total)

☐ normalize to 1 (show ratio)

Mean Trajectory

This chart type displays the arithmetic average over all trajectories of the data for each selected stage.

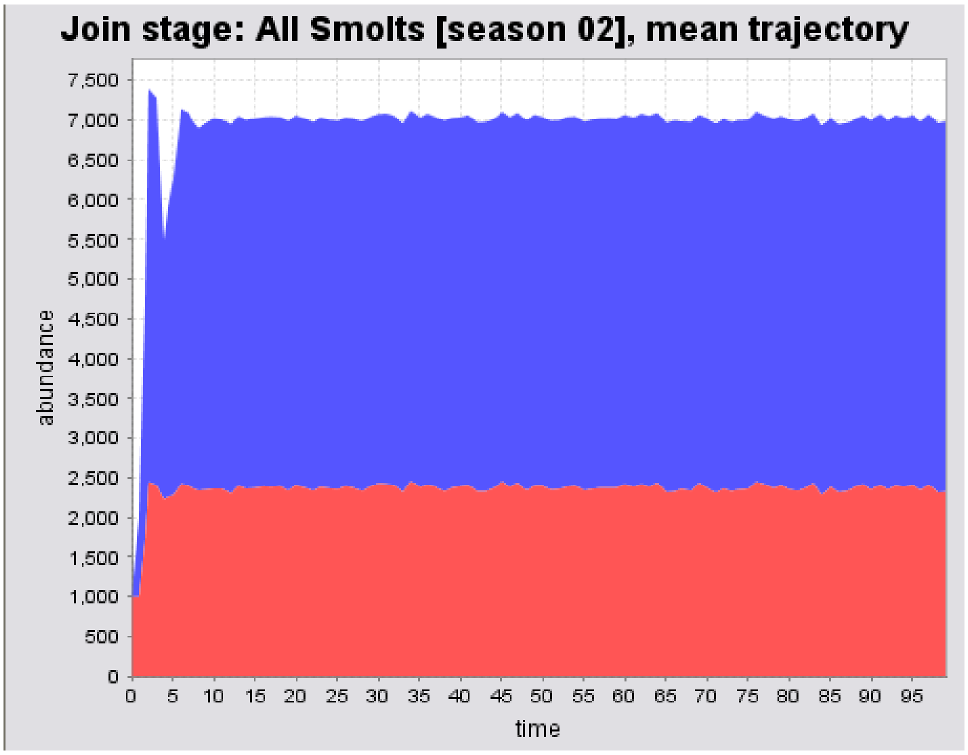
Mean trajectory plot



Mean trajectory plot (non-join stage)

If the only selected stage is a join — that is, a recruit stage with two or more stock stages, you have more options available. By default, the plots of each contributing stage are stacked, as shown:

Stacked mean trajectory plot

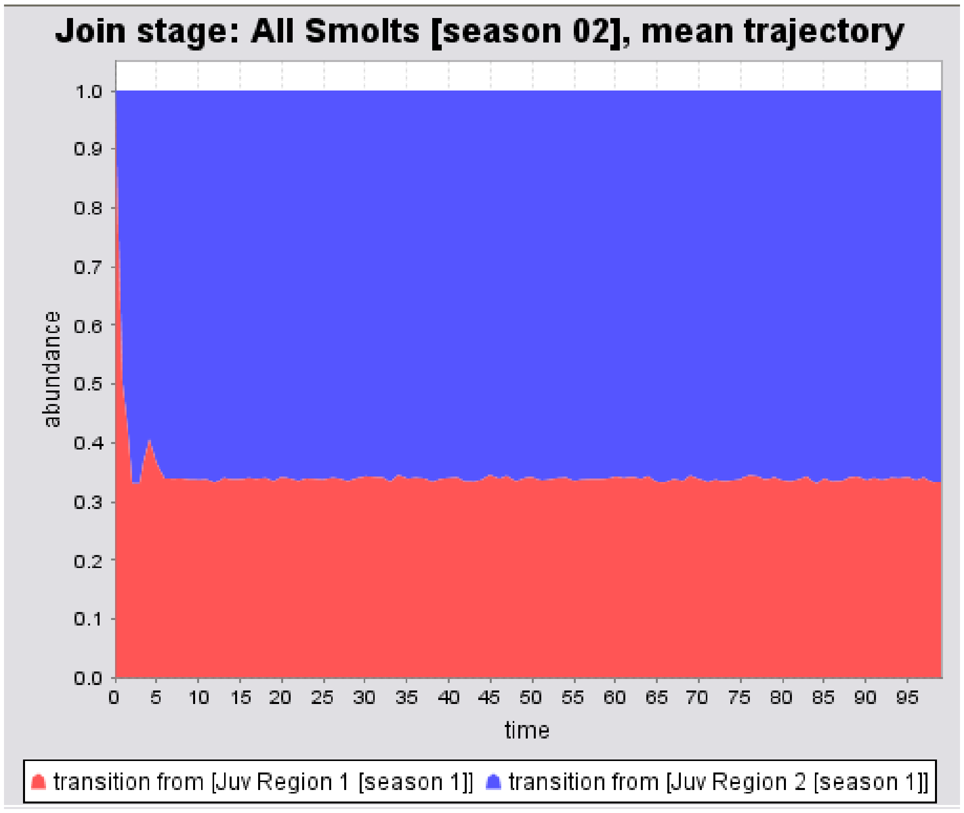


Stacked join stage abundance

The area corresponding to the abundance contribution from each stock stage is shaded in a different color.

To view the ratios of individual stage components to the total at each point in time, rather than the absolute abundance values, check the   normalize to 1 (show ratio)   box.

Stacked join stage ratio plot



Stacked join stage abundance ratios

In order to display only the total for the stage, check the   ignore join (show total)   box. (If more than one stage is selected for viewing, only the totals are plotted for join stages.)

Chart type : Median Trajectory

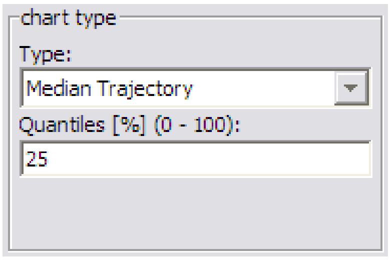


chart type

Type: Median Trajectory

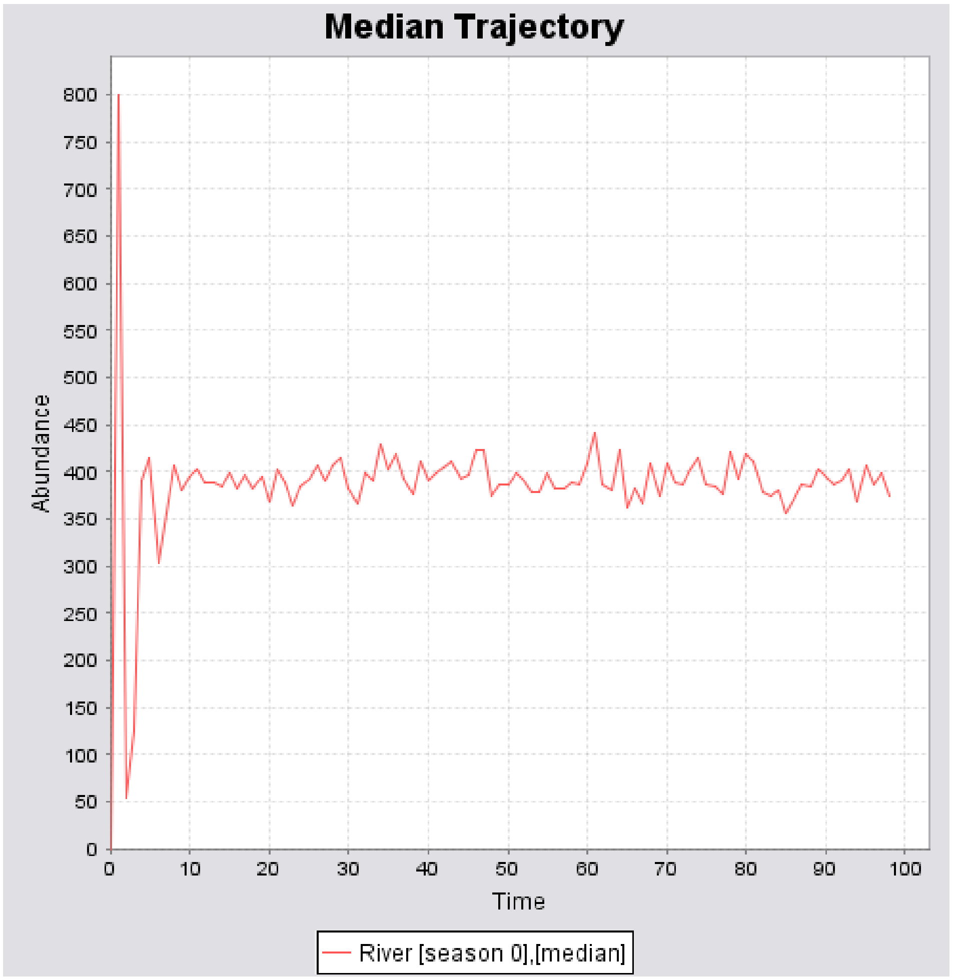
Quantiles [%] (0 - 100): 25

Median Trajectory

This chart displays the median value over all trajectories of the data for each selected stage. In addition, it can display certain other selected data quantiles.

If the Quantiles field is left empty, only the median data points (*i.e.*, the data points at each time step which are larger than half of all points at that step and smaller than the other half) are displayed.

Median trajectory plot



Median trajectory plot

Quantiles : If you enter an integer from two to 100 in the Quantiles field, two additional quantile profiles are displayed. They are calculated as follows:

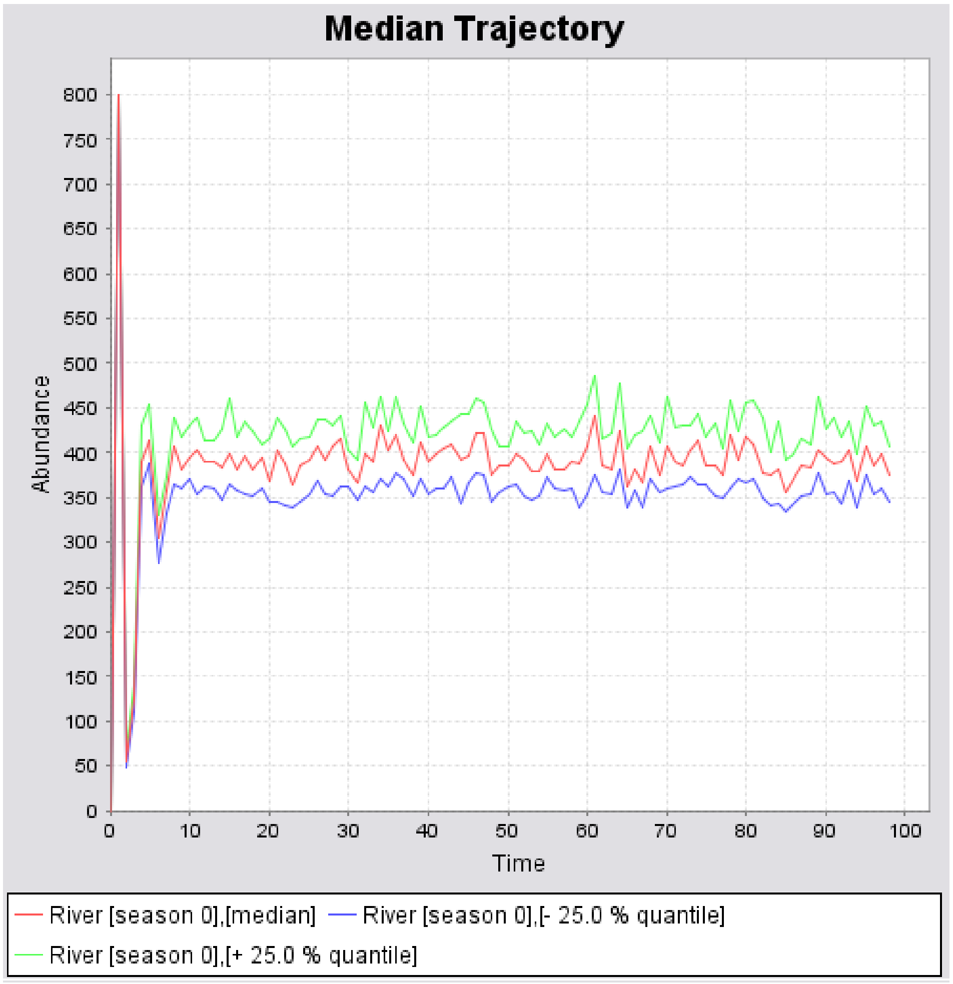
If P is the integer in the Quantiles field, then two percentage levels are calculated:

$$P_u = 50 + P/2$$
$$P_l = 50 - P/2$$

The upper(or lower) profile traces the data points at each time step in the simulation which are greater than P_u(or P_l) percent of the all data points at that time and less than 100-P_u(or 100-P_l) percent of them.

For P=25, the 50% (median), 62.5%, and 37.5% quantiles are plotted:

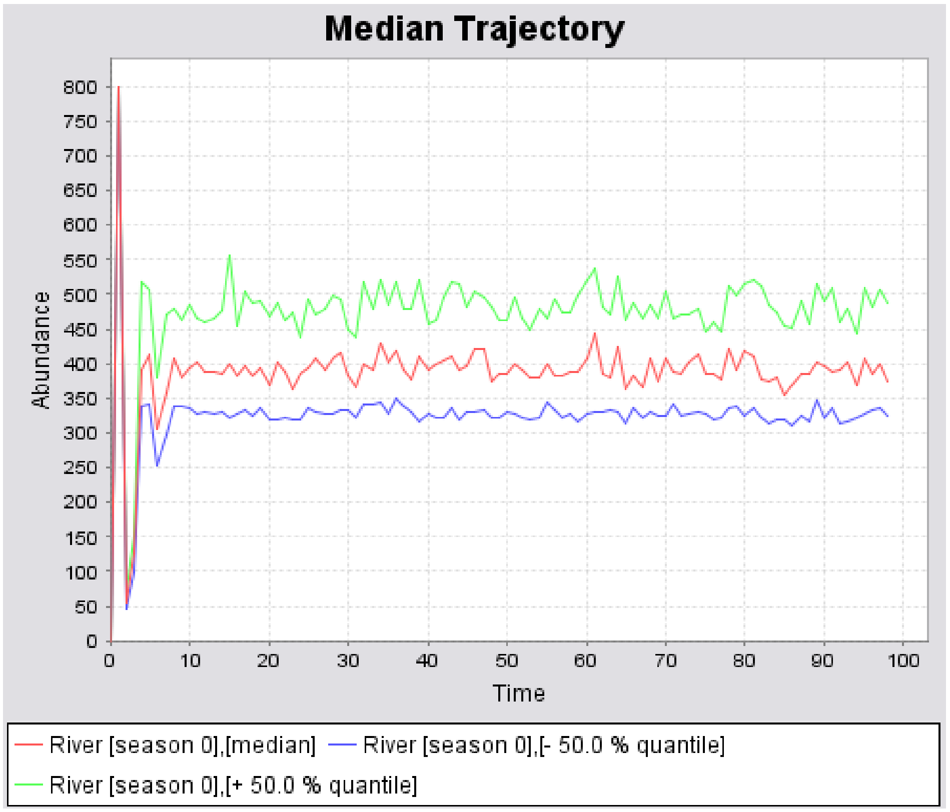
Median trajectory plot, 25% quantiles



Mean trajectory plot & 25% quantiles

To plot true quartiles, set P=50 (resulting in P_u=75 and P_l=25):

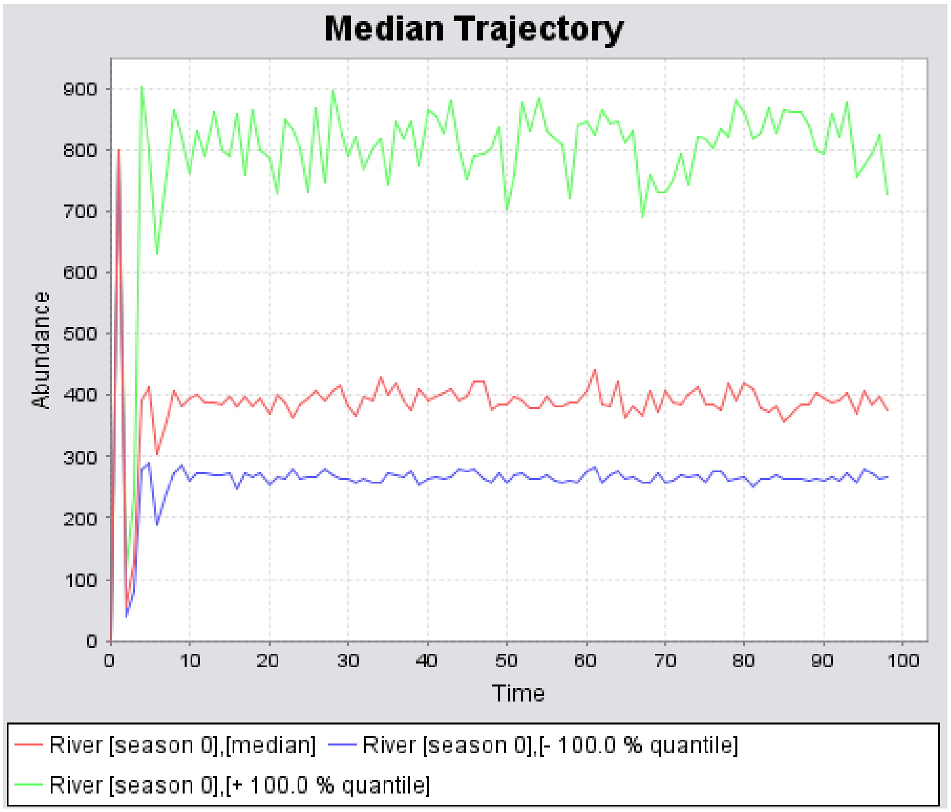
Median trajectory plot, 50% quantiles (quartiles)



Median trajectory & 50% quantiles

To plot the median along with maximum and minimum data points, enter P=100, so that P_u=100 and P_l=0:

Median trajectory plot, 100% quantiles



Median trajectory, with maximum and minimum

Chart type : Abundance Distribution

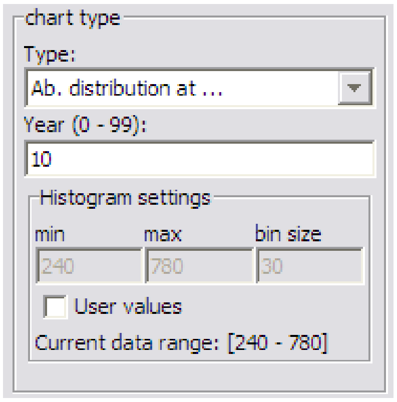


chart type

Type: Ab. distribution at ...

Year (0 - 99): 10

Histogram settings

min	max	bin size
240	780	30

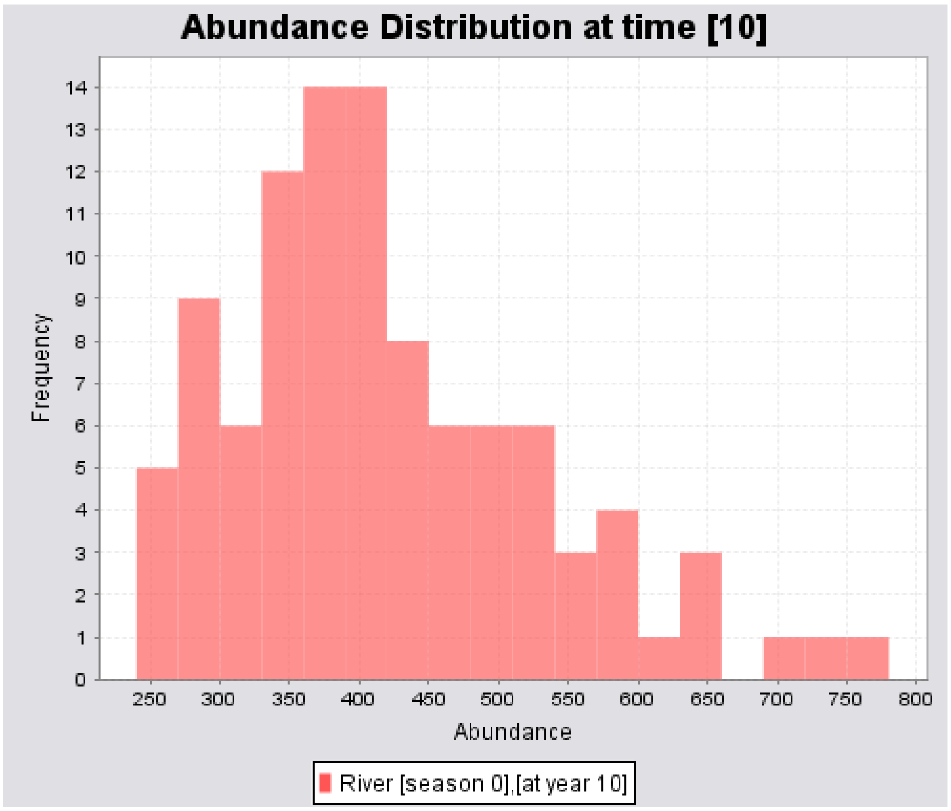
☐ User values

Current data range: [240 - 780]

Abundance histogram

This chart plots the frequency histogram of abundance values for the selected stage(s) for the year you specify.

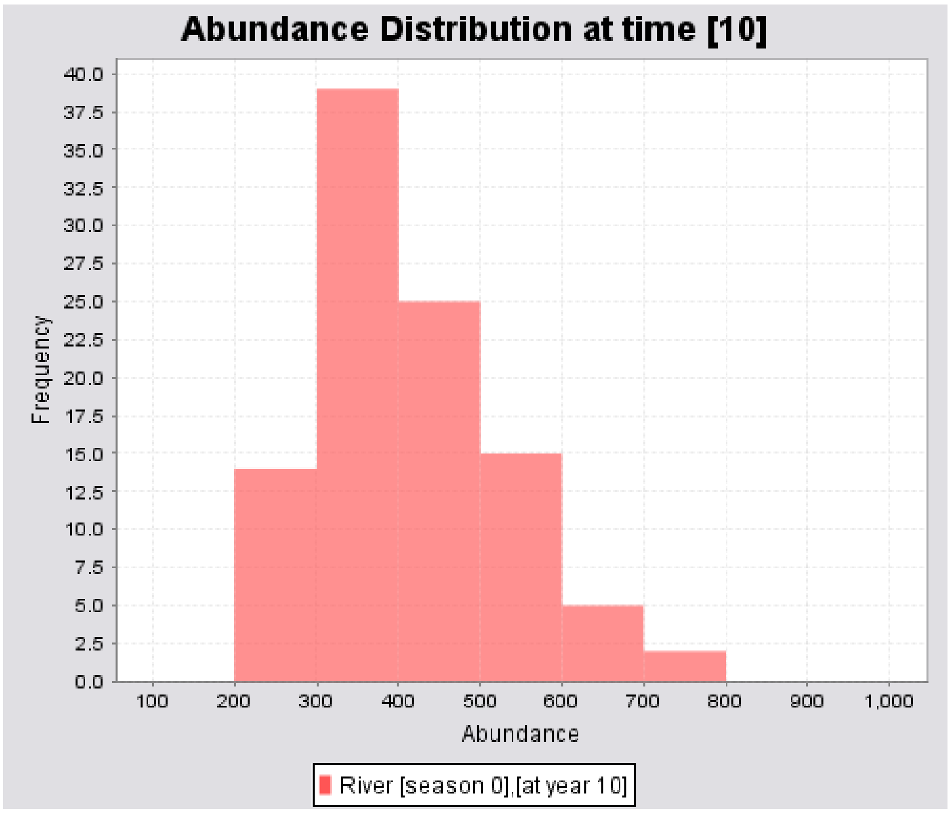
Abundance distribution plot



Abundance histogram

You can adjust the appearance of the histogram by checking the ‘User values’ box and entering your choice of values for the histogram's minimum, maximum, and bin size. For example, with min=100, max=1000, and bin size=100, the histogram looks like this:

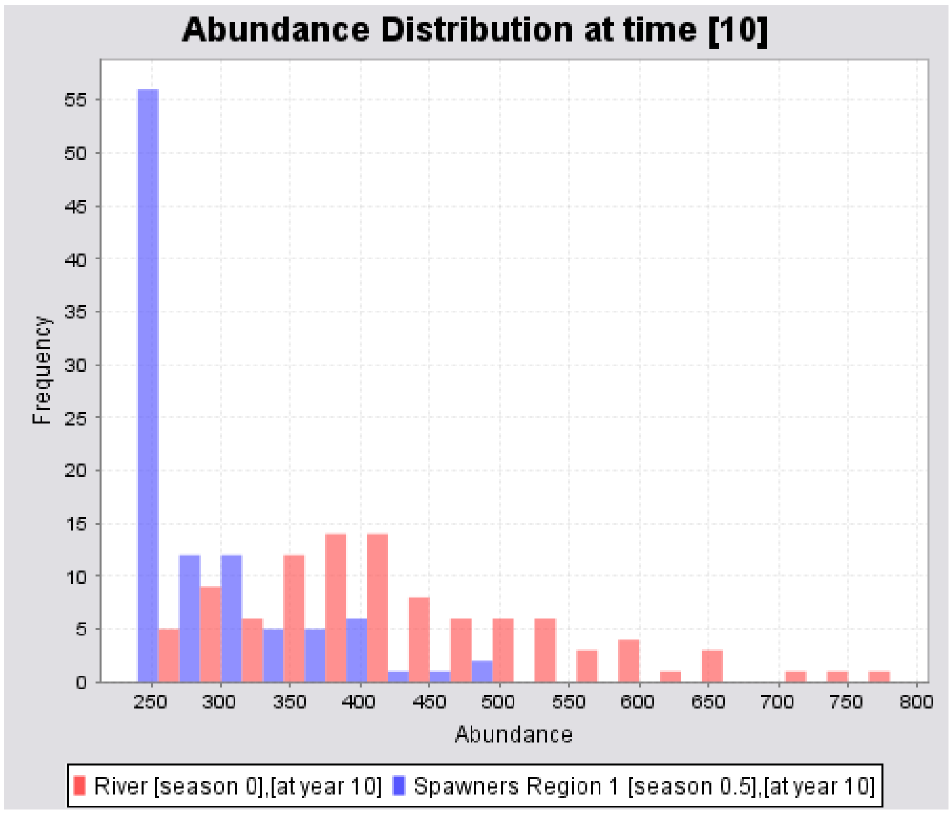
Customized histogram



Histogram with customized plotting parameters

If you select more than one stage, the histograms are interleaved.

Multiple abundance histograms



Abundance histograms of two stages

Chart type : Quasi-extinction probability

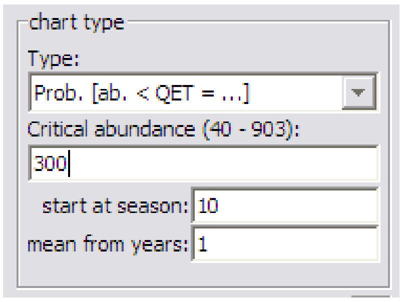


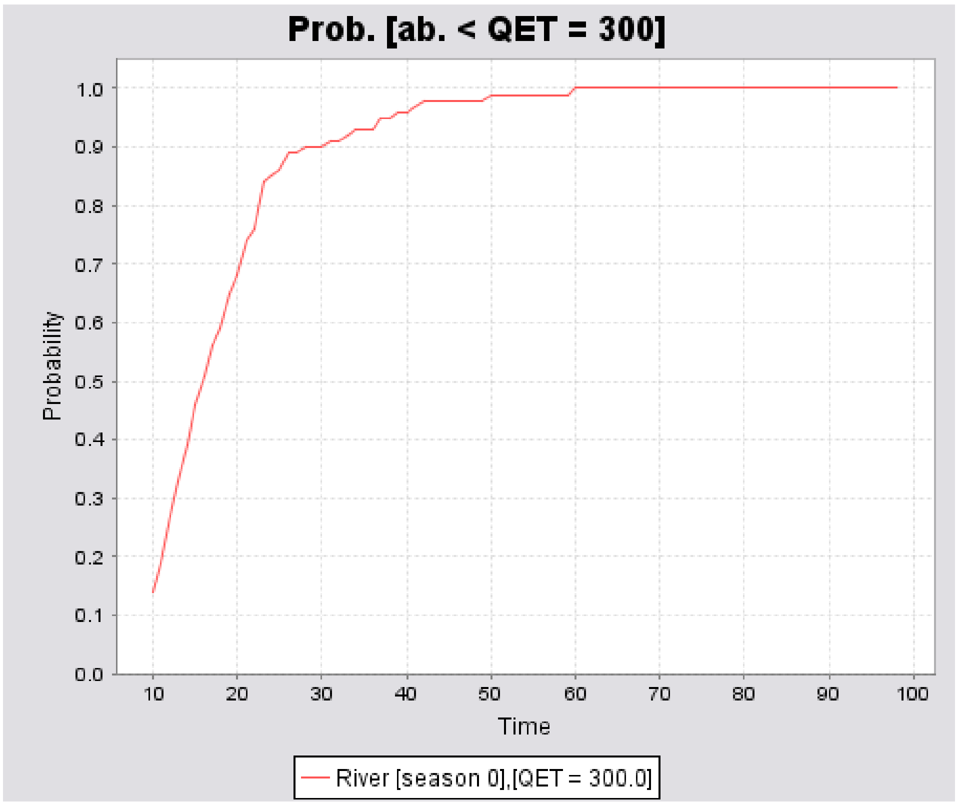
chart type
Type: Prob. [ab. < QET = ...]
Critical abundance (40 - 903): 300
start at season: 10
mean from years: 1

Quasi-extinction probability plot

This chart plots an estimate of the probability that a stage's abundance will drop below a critical level over the course of a simulation run. The critical level is called the quasi-extinction threshold , and the graph can be thought of as representing the likelihood over time that the modeled population will become effectively extinct.

To plot the probability that a given abundance will *exceed* a specified threshold, see the Exceeding threshold probability chart.

Quasi-extinction probability plot



Probability of abundance < 300 *vs* time

The value you enter in the Critical abundance field will be used as a quasi-extinction threshold (QET); the probability estimation will be carried out for each year in the threshold starting with the one you enter in the start at season field. For each successive year, the number of trajectories in which your specified QET exceeds the selected abundance either for that year or any previous year is divided by the total number of trajectories. The result is an estimate of the probability that the stage abundance will drop below the QET during or before this year in the simulation.

Equivalently, the graph plots the cumulative fraction of trajectories in which the abundance has dropped below the specified QET up to a given time.

If the stage of interest begins the simulation with a value of zero, then the quasi-extinction probability will always be one if you start the estimate with the first year in the simulation. In general, it's a good idea to specify a start at season value large enough for simulation start-up transient behavior to settle down.

mean from years : The probability estimation technique described above uses the abundance values for each year and each trajectory. This may be subject to distortion by brief abundance spikes. To lessen the effects of such spikes, you can average the trajectory abundances over consecutive years and check the averages against the QET — in effect, smoothing the trajectory data. Specify the number of years over which to average in the mean from years field; a value of one indicates that no averaging will take place.

Specifying an averaging window length of N years effectively removes N-1 plotted points from the right (high time) end of the graph.

Chart type : Exceeding threshold probability

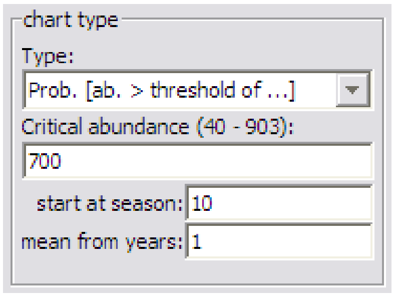
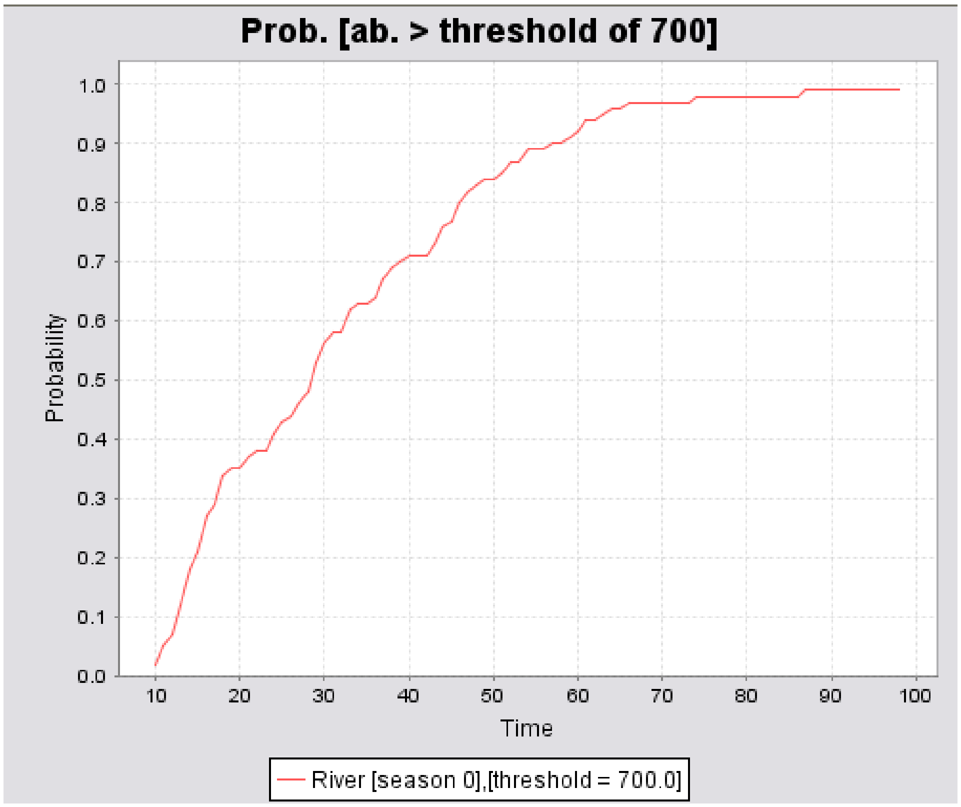


chart type
Type: Prob. [ab. > threshold of ...]
Critical abundance (40 - 903): 700
start at season: 10
mean from years: 1

Exceeding threshold probability plot

This chart plots an estimate of the probability that a stage's abundance will rise above a given level over the course of a simulation run. It is analogous to the Quasi-extinction probability chart, except that the QET chart plots the probability that a stage's abundance will fall below a specified threshold.

Exceeding threshold probability plot



Probability of abundance > 700 *vs* time

Enter the threshold value in the Critical abundance field; the probability estimation will be carried out for each year in the threshold starting with the one you enter in the start at season field. For each successive year, the number of trajectories in which the selected abundance exceeds your specified threshold either for that year or any previous year is divided by the total number of trajectories. The result is the estimate of the probability that the stage abundance will exceed the threshold at or before this year in the simulation.

Equivalently, the graph plots the cumulative fraction of trajectories for which the abundance has exceeded the specified threshold up to a given time.

if start-up transients in the simulation results skew the above-threshold probability estimate, it may be helpful to enter an integer greater than zero in the start at season field.

mean from years : The probability estimation technique described above uses the abundance values for each year and each trajectory. This may be subject to distortion by brief abundance spikes. To lessen the effects of such spikes, you can average the trajectory abundances over consecutive years and check those averages against the threshold — in effect, smoothing the trajectory data. Specify the number of years over which to average in the mean from years field; a value of one indicates that no averaging will take place.

Specifying an averaging window length of N years effectively removes N-1 plotted points from the right (high time) end of the graph.

Chart type : Stage productivity

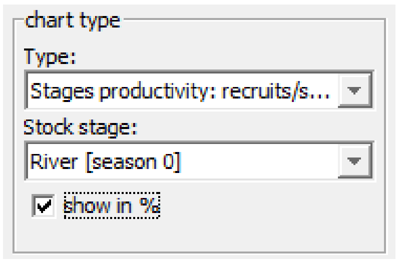


chart type

Type: Stages productivity: recruits/s...

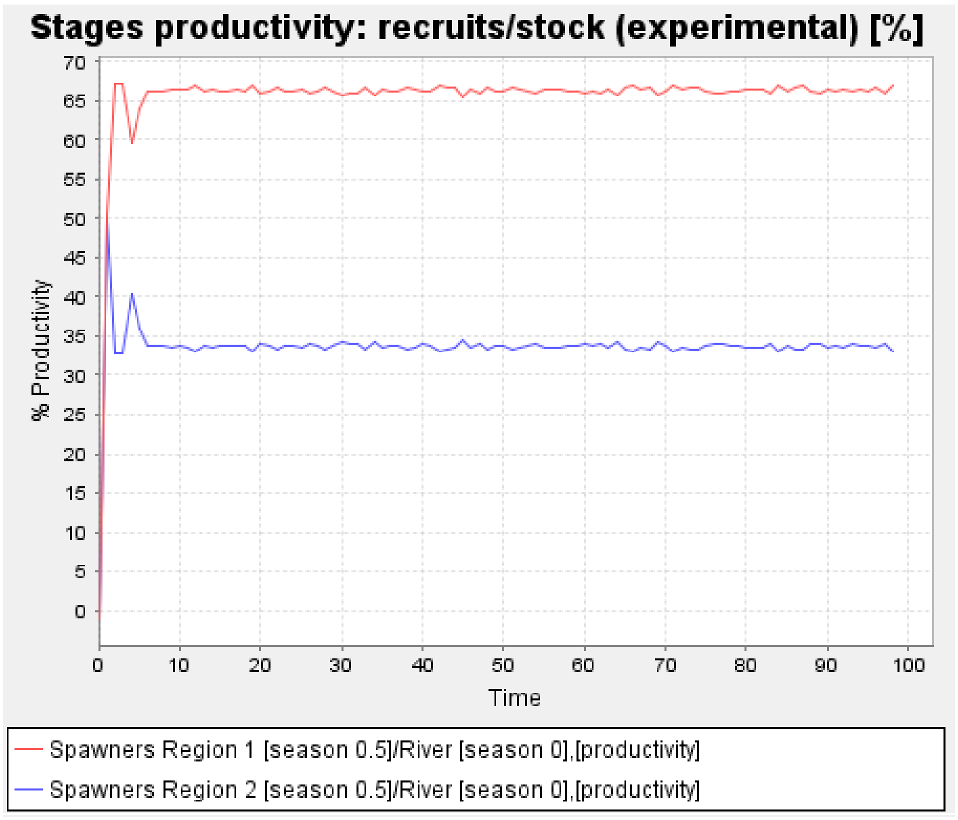
Stock stage: River [season 0]

☑ show in %

Stage productivity plot

This chart type displays ratios of abundances of selected stages to those of a reference stage. You can use it, for example, to examine the productivities of recruit stages referenced to a common stock stage.

Stage productivity plot

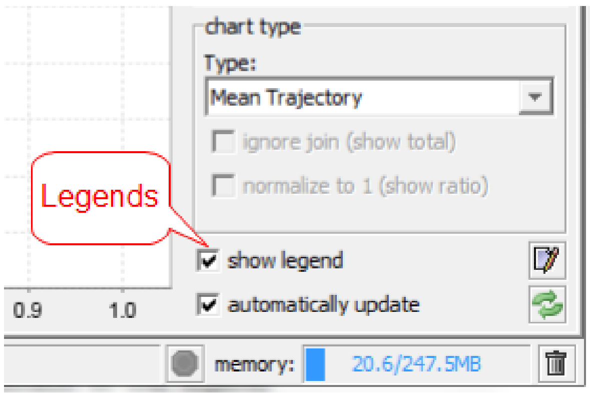


Stage productivity plots

To produce the plot shown above, two stages (Spawners Region 1 and Spawners Region 2 have been selected in the stages list. Their (common) stock stage, River , has been chosen from the Stock stage drop-down list, and the show in % check box has been selected. The resulting plots are the ratios (expressed in percent) of the arithmetic means of the Spawner abundances to the means of the River abundances at corresponding times.

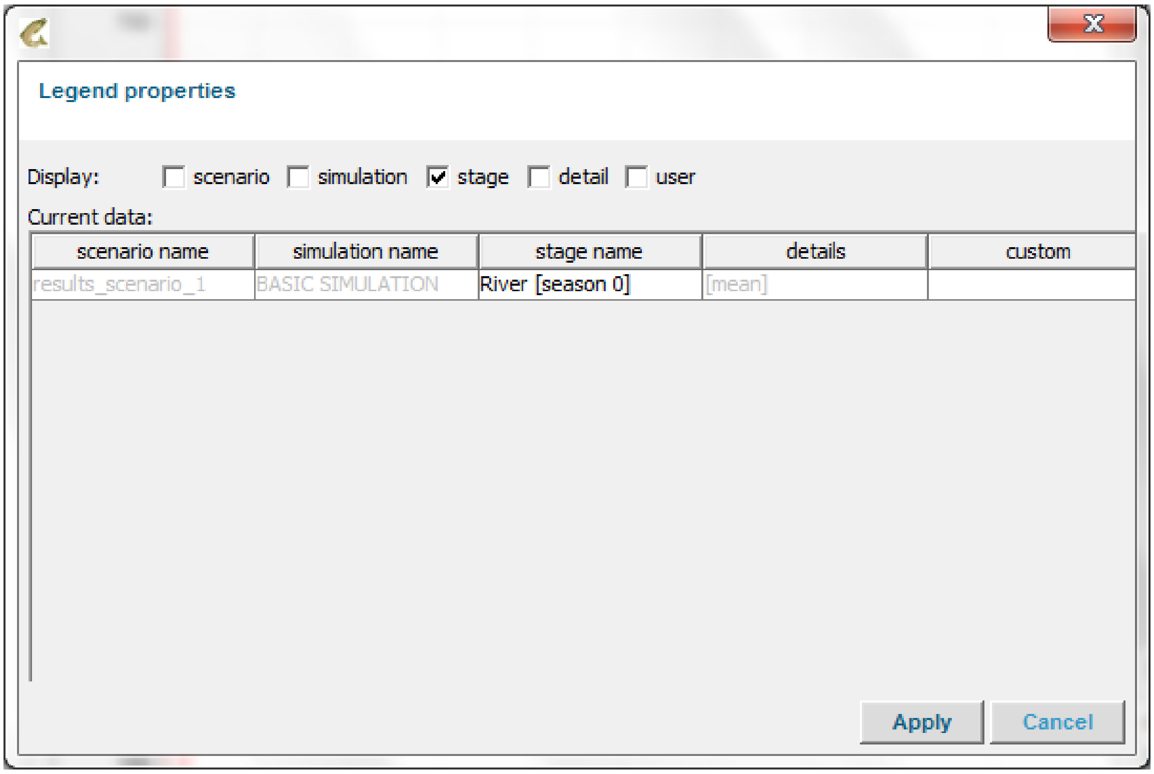
Simulation results chart legends

You can control the appearance of chart legends in the results viewer. You can hide or show legends by clearing or selecting the 'show legend' check box in the lower right corner of the results viewer display.



Show/hide chart legends

Click the edit icon to the right of the 'show legend' box to display the 'Legend properties' dialog box.



Display: ☐ scenario ☐ simulation ☑ stage ☐ detail ☐ user

Current data:

scenario name	simulation name	stage name	details	custom
results_scenario_1	BASIC SIMULATION	River [season 0]	[mean]	

Legend properties dialog

The 'Display' row of check boxes allows you to select segments of the labels that will identify each plot in the graph display. Each check box corresponds to a column in the 'Current data' table, which shows the contents of each legend. If a box is checked, the contents of its column are displayed in black type and that part of the legend will appear below the graph. If the box is cleared, the part of the label corresponding to it is still visible in the properties dialog, but it is displayed in gray, indicating that it will not be visible in the graph display legend.

The check boxes and their corresponding columns represent the following attributes:

scenario (column ‘scenario name’) : The name of the simulated scenario. Since you can load result files from several simulations into the results viewer, you may choose to display results from more than a single scenario at once.
simulation (column ‘simulation name’) : The name of the simulation from which the data is drawn.
Stage (column ‘stage name’) : The name of the stage displayed. By default, only this field appears in the legend.
detail (column ‘details’) : This legend segment identifies the type of chart being displayed. Its contents may include values of parameters specific to the the chart type; for example, the percentage value entered into the quantile field is included in the details for median plots.
user (column ‘custom’) : The final column in the ‘Current data’ table can contain comments you would like to include in the table legend. If there is more than one plot in the graph display, you can provide comments for each by double-clicking the ‘comment’ field for the plot, typing in your comment, and hitting *return*. As is the case for all legend fields, you must select the check box (‘user’ in this case) in order for your comment to appear in the graph legend.

Exporting simulation data

You can save selected data sets as well as an image of the current contents of the graph display, including customized legends. Click the Export button in the results viewer interface to display the Chart & Data Export dialog.

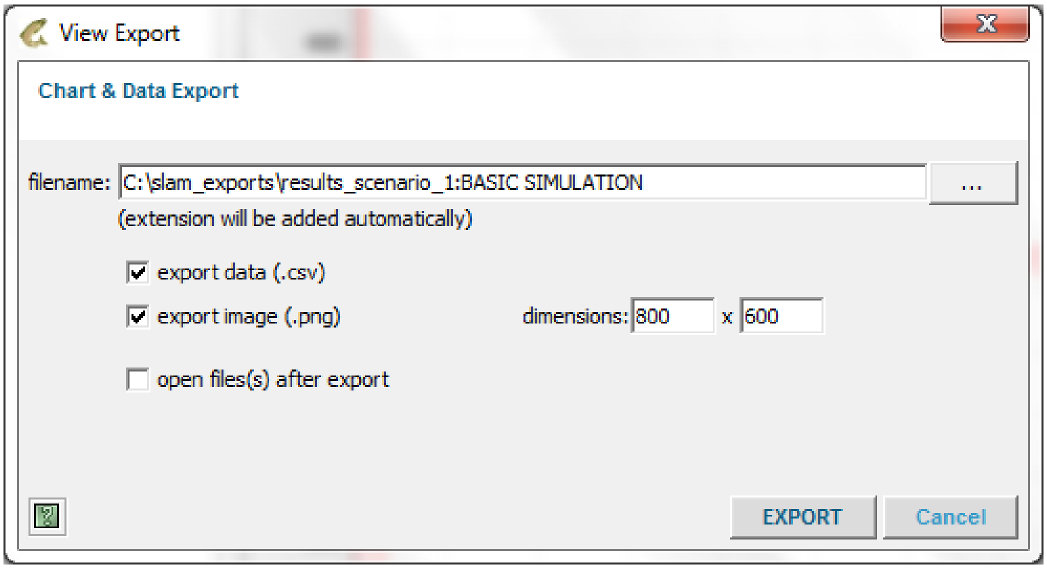

View Export

Chart & Data Export

filename: C:\slam_exports\results_scenario_1:BASIC SIMULATION
(extension will be added automatically)

☑ export data (.csv)

☑ export image (.png)   dimensions: 800 x 600

☐ open files(s) after export

Simulation results export dialog

filename : Enter the path to the file in which you want to store your output data and/or graph image (see below). If you have selected both data and image outputs, SLAM will create two files with the same base name and different extensions.

The filename field will initially contain an automatically-generated file name based on the simulation results name in the simulation list. The default path will point to the SLAM data directory (The slam_exports path shown above is for illustration only.) Feel free to change either, but make sure you have write permissions for the folder in which you want to place your exported files.

export data : Select this check box in order to export the simulation results data you have currently selected in the form of a *comma-separated variable* (CSV) file with the extension csv . CSV files can be imported into most spreadsheet programs.

For each selected data set, the CSV contains a row of simulation year values followed by a row of the abundance values corresponding to the years.

Your spreadsheet program may provide options for column-separation characters, as not all CSVs are actually comma-separated. If this is the case, select the comma-separated option when you import the file into the spreadsheet program.

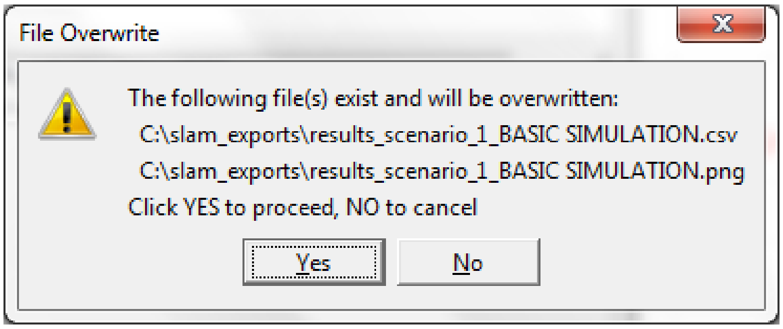
export image : Select this option to export an image of the graph display as it exists when you execute the export operation. SLAM will create a Portable Network Graphics (PNG) file with the extension png containing the graph display image, including any legend customizations you may have defined.

The created image size is by default 800 pixels wide by 600 pixels high. You can change the size by entering the appropriate values into the **dimensions** fields.

open file(s) after export : If you select this check box, SLAM will open the exported files after they are created. The files will be opened by the default programs assigned to csv and png files in your operating system.

You must select at least one export file type in order to enable the Export button. You can click Cancel at any time to close the dialog window.

When you click the Export button, SLAM will examine your export choices to determine if there are files in your selected output directory that would be overwritten by the export operation. If there are, you'll see a warning dialog like this:


File Overwrite

The following file(s) exist and will be overwritten:
C:\slam_exports\results_scenario_1_BASIC SIMULATION.csv
C:\slam_exports\results_scenario_1_BASIC SIMULATION.png
Click YES to proceed, NO to cancel

Export data overwrite warning

To continue with the export, click Yes . To cancel the export, click No .

When the export is complete, SLAM will use your operating system's functionality to open a window in the folder into which the export files were written.

Important note : You cannot re-import the files exported from the results viewer. The Import control loads entire simulation sets into the viewer; if you try to import a csv file created as described above, a failure will result.

SLAM data

SLAM uses two methods for storing models in order to meet two different needs. For everyday storage of life cycle models and scenarios, SLAM uses an internal database. This is the storage mechanism you are using when you load and save life cycles and scenarios. SLAM also provides mechanisms for exporting life cycles and scenarios to specially formatted files and for re-importing models from those files. The export/import capability allows you to make backups and to transfer models from one SLAM to another.

Understanding how exports and imports work will help you to use those functions. The internal database is integrated into SLAM so that its function is as transparent to the user as possible, but there are are aspects of database operation and maintenance, like switching databases and automatic backups, that can make SLAM simpler to use and more resilient against computer glitches.

- SLAM databases
- Exporting and importing models

The SLAM Database

SLAM stores all the details of life cycle models and scenarios in a database. (This excludes the contents of data files you may have associated with scenarios — but the paths to those files *are* kept in the database.) If you never change databases in SLAM and never have occasion to recover data from an automatic backup, you should be able to use SLAM without thinking about the database. Switching databases can help you organize the work you do in SLAM, and SLAM's database backups, like any other backup system, are only uninteresting until you need them.

How SLAM uses its database

SLAM uses an "in-process" database called HSQLDB to store all the information that specifies lifecycles and their associated scenarios. The first time you load or store a lifecycle after SLAM starts up, the embedded HSQLDB engine reads the current database into memory. (This is what's going on when you see the message about initializing the database.) From that time until you close SLAM or change databases), data is read from and written to this memory-resident database.

When you load a lifecycle or scenario, SLAM reads the model details from the database and uses them to build Java objects. When you modify, delete, or create aspects of the model, the information exists in the form of Java objects until you tell SLAM to save the lifecycle or scenario to the database.

Before SLAM loads your database, the information is stored in HSQLDB files on the hard drive. Each database uses two of these files with the same base file name but different extensions. If your database is called MyDB, the files are named MyDB.properties and myDB.script. The .properties file is small and contains general HSQLDB parameters. Your data is stored in the .script file, which describes the SLAM database table structure and tells HSQLDB how to populate the tables when the database is read into memory.

Changing Databases

The first time SLAM stores a lifecycle model after its initial installation, it creates a new database in a default directory.

The path to the database files that SLAM is currently using is displayed in the caption bar at the top of SLAM's main window. If this path is

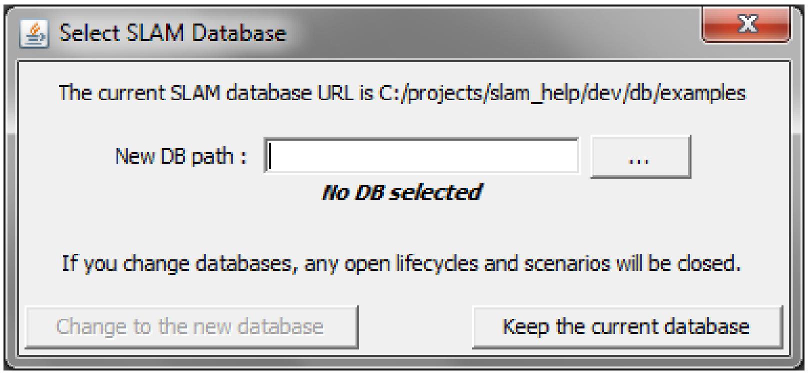
```
/directory1/directory2/slam/db/MyDB
```

then the database contents will be written to and read from these files:

```
/directory1/directory2/slam/db/MyDB.properties
/directory1/directory2/slam/db/MyDB.script
```

If you like, you can store all of your SLAM models in the default database SLAM creates following installation. You may find it useful to organize your SLAM models by storing them in different databases, for example, to group models by project.

To create a new SLAM database or to change your default to a database you previously created, click the Select database item in SLAM's Life cycle menu entry, type `Ctrl+B`, or click the 'select database' button on the left-hand toolbar. The Select SLAM Database dialog will appear.


Select SLAM Database

The current SLAM database URL is C:/projects/slam_help/dev/db/examples

New DB path :

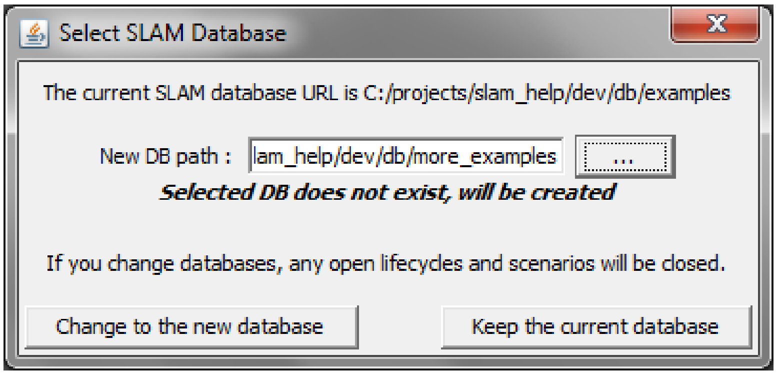
No DB selected

If you change databases, any open lifecycles and scenarios will be closed.

The select database dialog

In it there is an edit field labeled New DB path ; to the right of the field is a browse button labeled ... . Click the browse button to display a file-browser window, and use it to navigate to the directory where you would like to create your database or where the existing database files are located. What you do next depends on whether you are creating a new database or changing to an existing one.

To create a new database, type the database name (omitting the extension) into the box at the bottom of the file-browser window and click the Select DB button. The file-browser will close and you'll see the database path displayed in the edit field of the Select SLAM Database dialog. Beneath it will be the message Selected DB does not exist, will be created . (If you see instead Selected DB exists, will be opened , then you're opening an existing database rather than creating a new one; you'll either need to select a different database name or create your new database in a different directory.)



The current SLAM database URL is C:/projects/slam_help/dev/db/examples

New DB path : lam_help/dev/db/more_examples

Selected DB does not exist, will be created

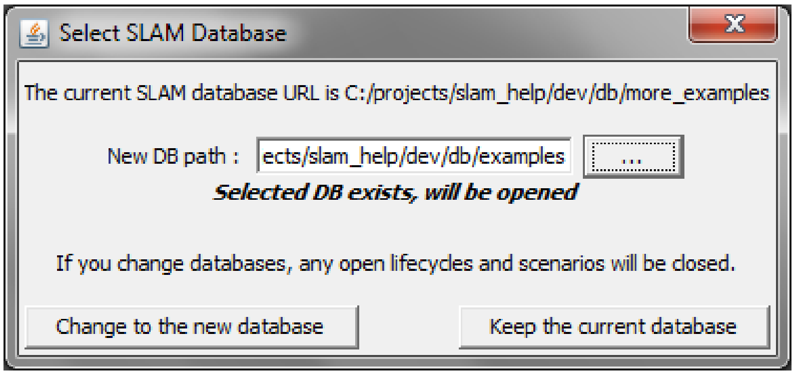
If you change databases, any open lifecycles and scenarios will be closed.

Creating a new database

Click the Change to the new database button and you're done. The default database path in SLAM's caption bar will change to reflect your choice, and the next lifecycle model you save will go into the new database. (Since it is a new database, if you try to load a lifecycle from it, the list of stored lifecycles will be empty.)

If you change your mind about creating the new database, click the Keep the current database button and SLAM will keep using the current database.

To select an existing database in the directory you've chosen, select either the .script or .properties file corresponding to that database and click the Select DB button. The Select SLAM Database dialog now displays the name of the new database (without the extension) above the message Selected DB exists, will be opened . Click Change to the new database to complete the process or Keep the current database to cancel the database change.



The current SLAM database URL is C:/projects/slam_help/dev/db/more_examples

New DB path : ects/slam_help/dev/db/examples

Selected DB exists, will be opened

If you change databases, any open lifecycles and scenarios will be closed.

Selecting an existing database

Automatic Database Backups

Sometimes things go wrong: SLAM crashes, your computer crashes, or the power fails. In SLAM this could result in a corrupted database and a loss of models and the effort that went into creating them. While the aim is to minimize the likelihood that something will fail and corrupt your data, SLAM also includes an automatic backup feature designed to mitigate the damage should such a problem occur.

Each time SLAM opens a database for reading or writing, it first copies the current database files to a backup folder on the hard drive. You can access a SLAM backup in several ways, but it's important to understand first what the backup procedure does.

Since SLAM automatically creates a backup when a database is opened, the following sequence of events will occur with a probability near one: you open your database and see a problem. You shut down SLAM and re-start it, then re-open the database, which is still broken. In the process, you overwrite the preceding backup copy. You might even repeat the sequence a few times, creating even more bad backups.

To make it less likely that you overwrite each and every unbroken backup, SLAM keeps up to ten backups for each database and rotates the current backup location among them.

Suppose the current, just-created database is called MyDB. In the directory that contains the files `MyDB.properties` and `MyDB.script`, SLAM will create an initial backup subdirectory called `MyDB_SLAM_DB_Backup_000` and will copy `MyDB.properties` and `MyDB.script` into it. (On the first go-round, of course, these are copies of an empty database.) The next time the MyDB database is opened, another sub-directory called `MyDB_SLAM_DB_Backup_001` will be created and the current `.properties` and `.script` files will be copied into it. The previous backup files in `MyDB_SLAM_DB_Backup_000` are left unchanged.

Successive sub-directories are created up to `MyDB_SLAM_DB_Backup_009`. After that tenth set of backups is created, SLAM will rotate back to the `MyDB_SLAM_DB_Backup_000` directory and overwrite the `.properties` and `.script` files in it with backups of the current versions. There are always ten sets of backups for the SLAM's current, default database (fewer if the database hasn't been opened ten times yet).

If you create a new database called, say, `widget` in the same directory as `MyDB`, then a new set of backup sub-directories will be created and populated, called `widget_SLAM_DB_Backup_000`, `widget_SLAM_DB_Backup_001`, and so on.

You can roll back to a previous version of your database simply by opening the database files in the appropriate backup sub-directory. Since SLAM doesn't tell you which database backup is the newest one, you'll need to examine the file system (using your operating system's file explorer application, for example) to see which backup corresponds to the last time you opened the database without errors. (Remember, the most recent backup is probably a copy of the broken database.)

Once you have decided which backup you want to open, you have a choice about how you want to use that backup. If you simply change databases to the backup as described above, SLAM will set the backup as its current database and will start making backups of it in the backup sub-directory. (That is, SLAM will treat the backup database exactly the same as any other database, creating backups of it as it is used.) This is the most cautious approach, since it avoids overwriting any existing database or backups, but it complicates the layout of databases in the file system.

Another approach is to close SLAM, choose a backup database, and copy the files from the backup sub-directory back to your main database directory, overwriting your current, buggy database. When you restart SLAM, it will automatically use the overwritten files as its default database and will make backups of those files into the existing backup directories. This is the least cautious approach, it overwrites both the current database and, over time, your existing backups as well.

A reasonable middle course might be to use the first option to open backups until you find one that best suits your needs. Then, when you're satisfied that you've found the most useful backup, you can close SLAM and manually copy the backup's `.properties` and `.script` files to the desired location, renaming them if you like. Then you can re-open SLAM and change your database default to the newly-copied files.

A word of caution: the automatic backup system is intended for use in emergencies. It is an excellent idea for you to back up your own files on a regular basis. To do this, shut SLAM down or change to a different database in order to be certain that your most recent modifications have been written to the drive. Then copy the `.properties` and `.script` files to your desired backup media. If you want to use a backup you previously created, copy the two files to your hard drive and, if necessary, tell SLAM to use them as its current database, as described above. (Be sure you copy the files into a directory where SLAM can both read and write.)

Old Databases in New SLAMS

By design, new versions of SLAM are capable of reading databases created by earlier versions. Every effort is made to see that this in fact happens. Sometimes, though, bugs will appear that prevent SLAM from reading old databases. It is a good idea to export lifecycles and scenarios you want to continue using into XML files using the SLAM version they were created with before you upgrade. When compatibility problems are detected, they will be corrected in SLAM updates, but the patch will probably not be released during the afternoon (or even the week) in which you encounter the difficulty, so maintaining two paths for saving your important models may help avert frustration.

Exporting and Importing Models

SLAM allows you to export life cycle models and scenarios to files stored on your computer's drive and to read those exported files in again. All model details are stored in the internal database and the export-import feature does not replace that storage. The ability to save models to external files allows you to perform model-specific backups and to copy models from one SLAM to another.

Backups : Exported files can provide useful, model-specific backups. While your SLAM databases are automatically backed up, you may want to avoid the possibility of accidentally modifying a life cycle or scenario stored in the database. If you're cautious about computer failure destroying your work, you can copy exported files to external storage media.

Copying models : Exported files can be moved from one SLAM installation to another. (This is subject to version compatibility - files generated by newer SLAM versions may not work properly in SLAMs that are significantly older, though every effort is made to maintain backward compatibility among versions.) You can copy exported files to any storage medium that works on your computer, re-copy them to another computer with SLAM installed, and import the models there. Or, you can email models to colleagues. (While you may be able to do this with database files, copying and distributing them is not a recommended procedure.)

There are four export/import tasks:

- Exporting life cycle models
- Importing life cycle models
- Exporting scenarios
- Importing scenarios

These tasks share common or similar features:

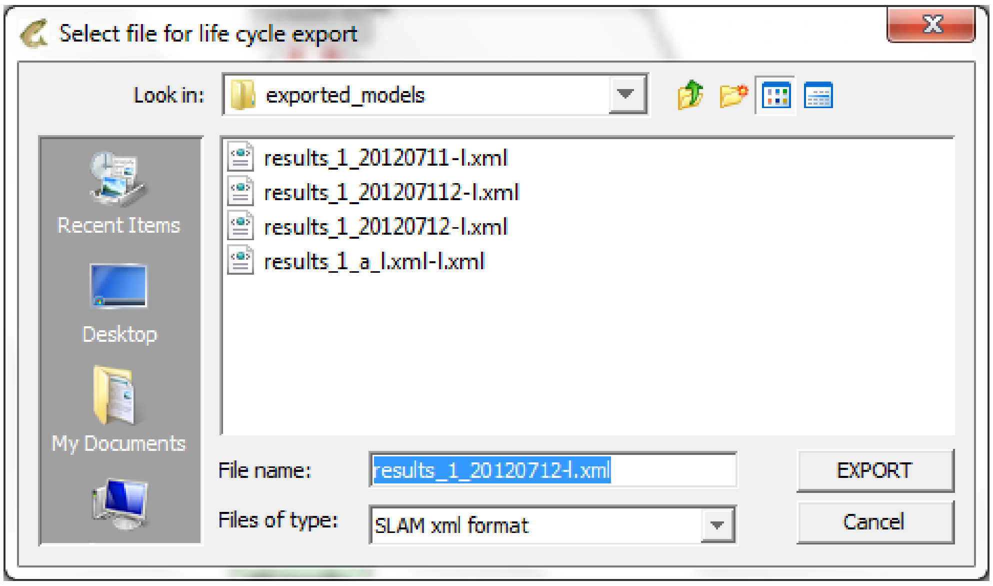
Common export/import features

Exporting Life Cycle Models

You can export life cycle models to files on the computer's drive or other storage media and import them again. This allows you to make life-cycle-specific backups and to share life cycles with other SLAMs.

After you create or load a life cycle, export it by selecting the export life cycle to a file item from the Import/Export menu, typing Ctrl+X, or clicking the export life cycle button in the left-hand toolbar.

A file browser dialog labeled Select file for life cycle export will appear, with a default file name like results_1_20120712-l.xml in the File name field.


Select file for life cycle export

Look in: exported_models

results_1_20120711-l.xml
results_1_201207112-l.xml
results_1_20120712-l.xml
results_1_a_l.xml-l.xml

File name: results_1_20120712-l.xml

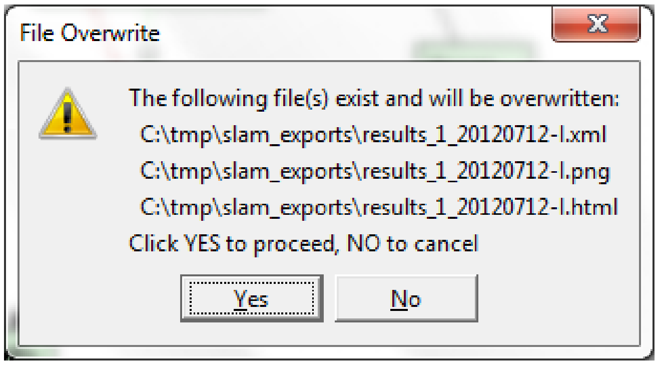
Files of type: SLAM xml format

Life cycle export file selection

To use a different file name, type it into the File name field. While only the xml file name is displayed, three files will be created.

Click the EXPORT button to export your life cycle (or Cancel to cancel the operation).

If any of the export files being created would overwrite existing files in the export directory, you'll be given the opportunity to proceed or cancel the operation by a dialog like this:

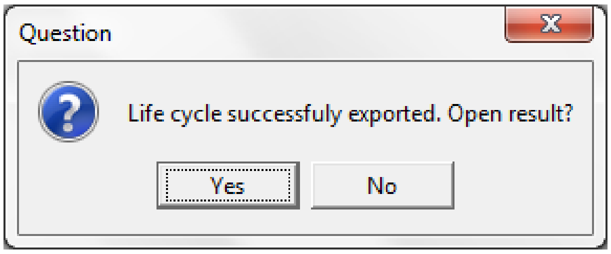

File Overwrite

The following file(s) exist and will be overwritten:
C:\tmp\slam_exports\results_1_20120712-l.xml
C:\tmp\slam_exports\results_1_20120712-l.png
C:\tmp\slam_exports\results_1_20120712-l.html
Click YES to proceed, NO to cancel

Export file overwrite warning

For more information on the files created during an export and the rules for generating default export names, see More about Exports and Imports.

When the file export is complete, you will be asked by this dialog



Life cycle successfuly exported. Open result?

Life cycle export complete

if you want to open the exported files. If you click Yes your browser will open the exported HTML and PNG files that describe the lifecycle. If you click No the dialog simply closes.

At this point, the life cycle export is complete. The life cycle file can now be imported into SLAM.

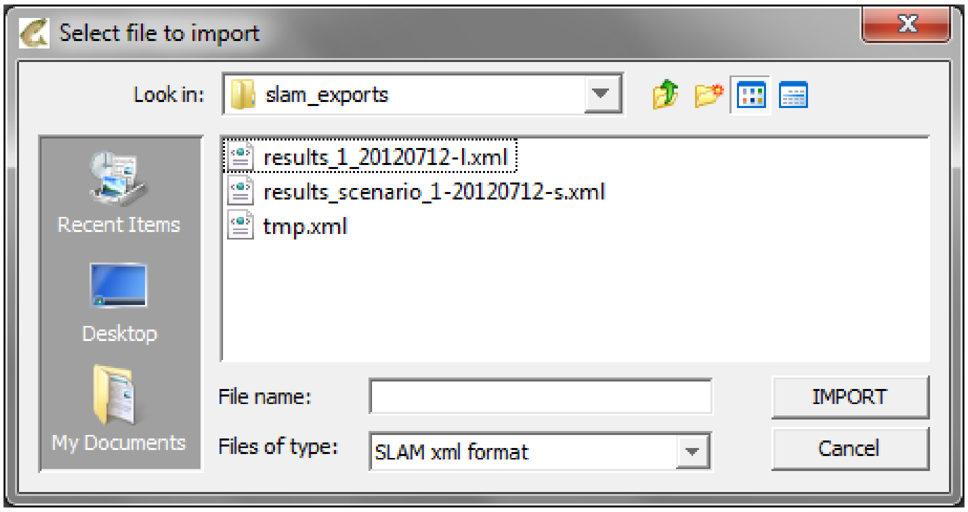
Note : A life cycle export file contains only the life cycle; it does not include information about the scenarios associated with the life cycle. For scenario exports, see Exporting scenarios.

Importing life cycle models

You can import life cycles from previously exported life cycle models.

Import a life cycle model by clicking the Import life cycle from file item in SLAM's Import/Export menu, by typing Ctrl+I, or by clicking the life cycle import button in SLAM's left-hand toolbar.

The Select file to import dialog appears:



Life cycle import file selection

Browse to the directory where the desired life cycle export file is located. Select the file and click Import to import the life cycle, or click Cancel to cancel the operation.

Note : You can import life cycles from either exported life cycle files or from exported scenario files. For more information, see Export file formats.

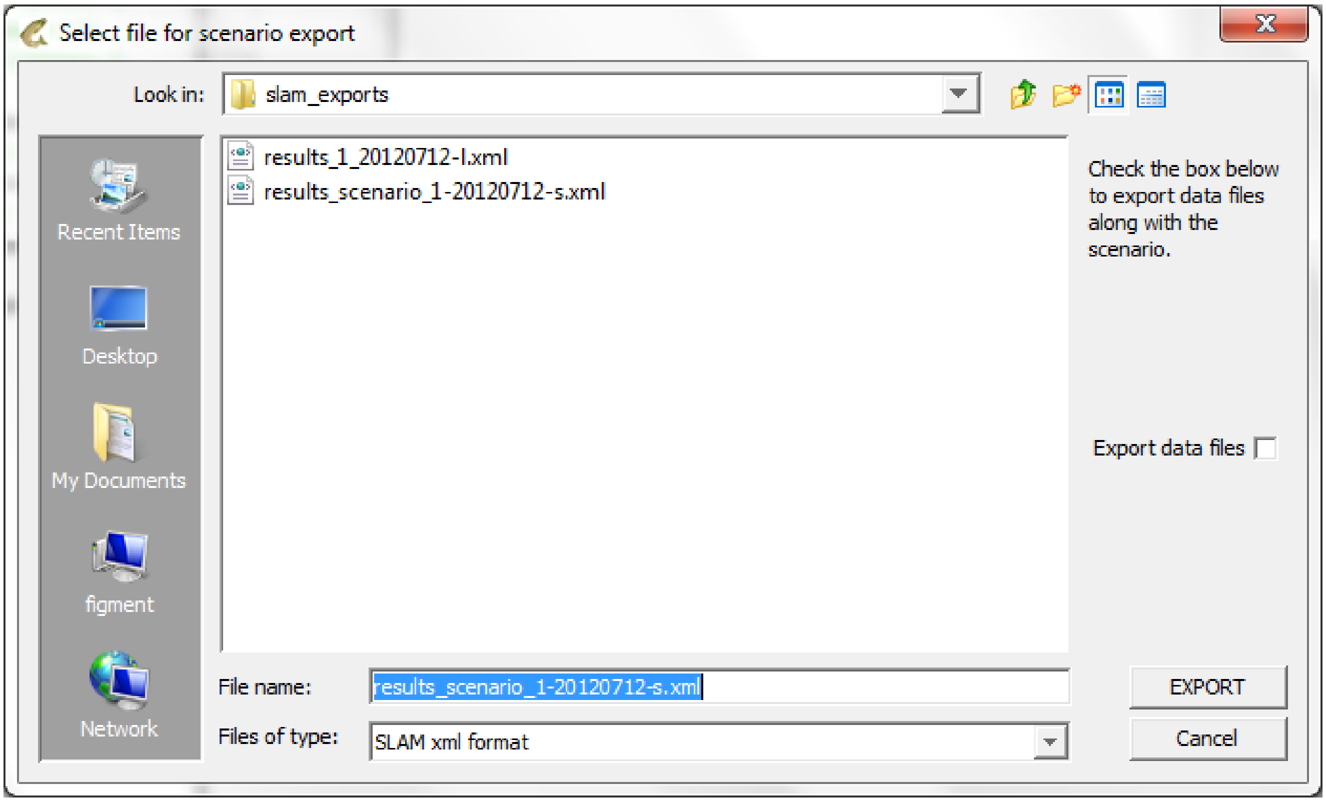
When you click Import , the life cycle will be imported. Note that, before you can create or import a scenario, you must save the imported life cycle to the database.

Exporting Scenarios

You can export scenarios to files on the computer's drive or other storage media and import them again. This allows you to back up specific scenarios and to share scenarios with other SLAMs.

After you create or load a scenario, export it by selecting export scenario to a file from the Import/Export menu, by typing Ctrl+Alt+X, or by clicking the export scenario button in SLAM's tool bar.

A file browser dialog labeled Select file for scenario export will appear, with a default file name like results_scenario_1_20120712-s.xml in the File name field.


Select file for scenario export

Look in: slam_exports

results_1_20120712-l.xml
results_scenario_1-20120712-s.xml

Check the box below to export data files along with the scenario.

Export data files ☐

File name: results_scenario_1-20120712-s.xml

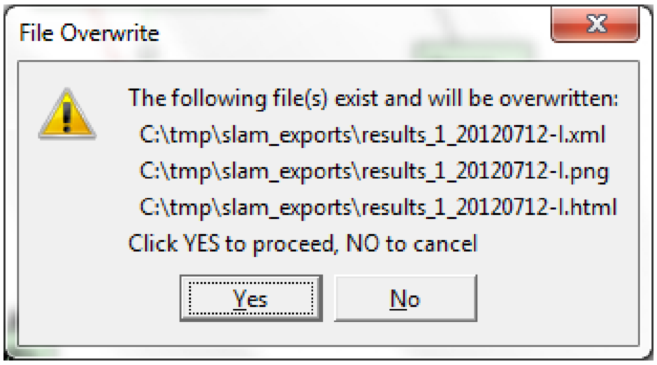
Files of type: SLAM xml format

Life cycle export file selection

If you decide to use a different file name, type it into the File name field. Note that, while only a file name with an xml extension is displayed, a total of three files will be created.

To include the contents of input data files attached to the scenario, check the Export data files box on the right side of the dialog. See Data files and exports for more information;

If any of the export files being created would overwrite existing files in the chosen export directory, you'll be given the opportunity to cancel the operation by a dialog like this:

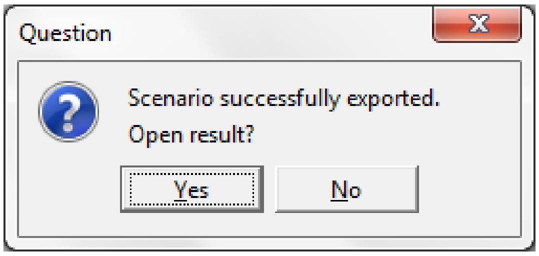

File Overwrite

The following file(s) exist and will be overwritten:
C:\tmp\slam_exports\results_1_20120712-l.xml
C:\tmp\slam_exports\results_1_20120712-l.png
C:\tmp\slam_exports\results_1_20120712-l.html
Click YES to proceed, NO to cancel

Export file overwrite warning

(For more information on the creation of default names, see More about Exports and Imports.)

When the file export is complete, you will be asked by this dialog


Question

Scenario successfully exported.
Open result?

Scenario export complete

if you want to open the exported files. If you click Yes your browser will open the exported HTML and PNG files that describe the lifecycle. If you click No the dialog simply closes.

At this point, the scenario export is complete.

Data Files and Exports

You can define some of the parameters in scenario relationships to take on values read from data files. When you export a scenario that has data files associated with it, you have two options for handling the files.

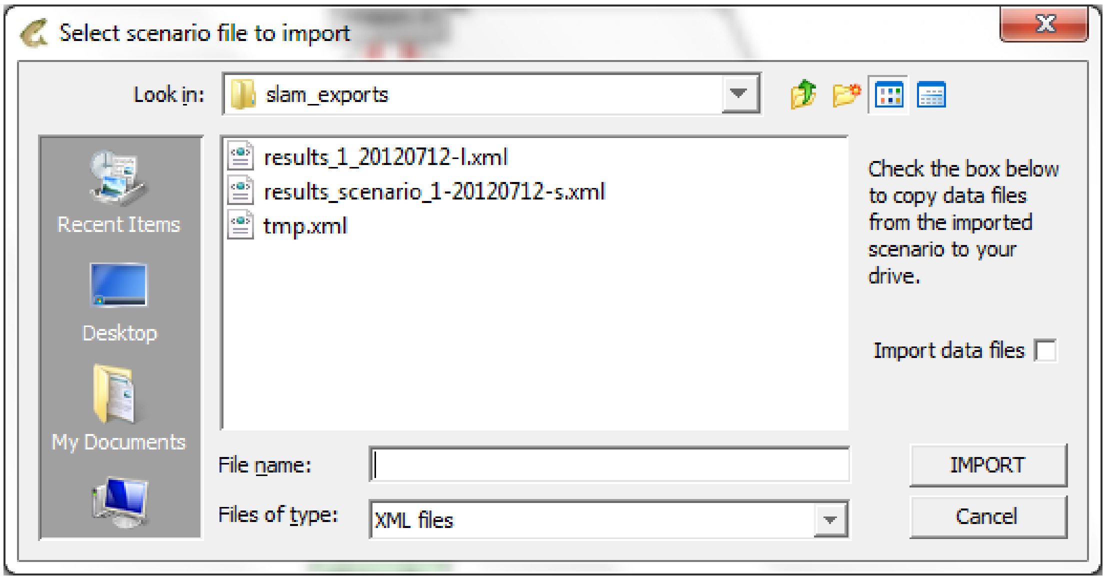
- **Export the scenario only (without data files)** : This is the default mode of scenario export, with the Export data files box not checked in the scenario export dialog. In order to import such a scenario, the data files must be available. When the scenario is loaded, SLAM will prompt for the files if it can't find them in the data-file directory. This means that, in order to use the scenario export for backups or copying, you must also back up or copy the data files. SLAM will display a dialog reminding you of this when the export is completed.
- **Export the scenario with data files embedded** : When the Export data files check box in the scenario export dialog is checked, the associated data files will be read and encoded into the exported scenario. When the scenario is imported, you'll be given the option of having SLAM re-create the data files on disk from the copies embedded in the scenario export file. An advantage of embedding data files is that you only have to keep track of a single file. A disadvantage is that the scenario export file, since it contains the data files, may become significantly larger.

Importing scenarios

You can import previously exported scenarios.

Import a scenario by selecting import scenario from the Import/Export menu, by typing Ctrl+Alt+I, or by clicking the import scenario button on the left-hand toolbar.

The Select scenario file to import dialog appears:



results_1_20120712-l.xml
results_scenario_1-20120712-s.xml
tmp.xml

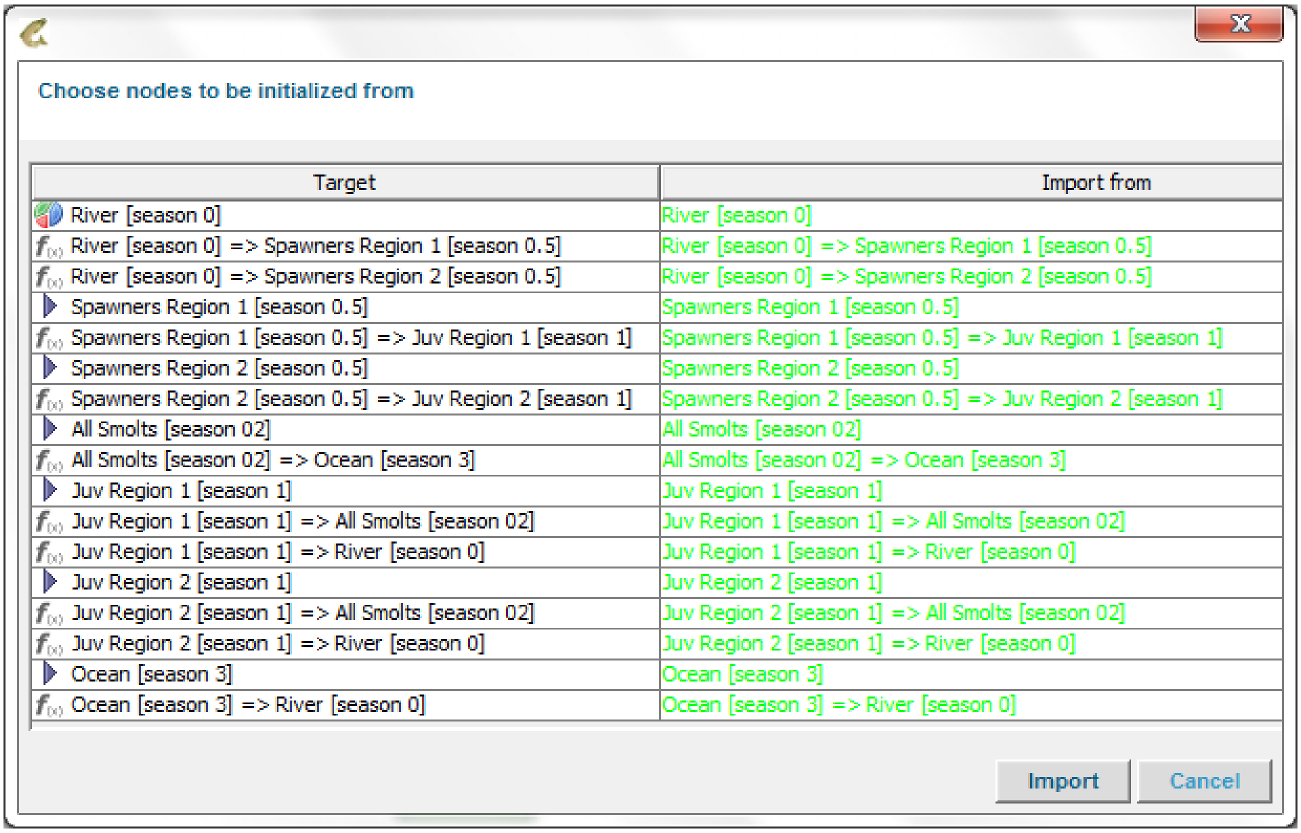
Check the box below to copy data files from the imported scenario to your drive.

Import data files ☐

Select scenario file to import

Select the XML file containing a description of the scenario you want to import and click IMPORT . If there are data files included in the scenario file, check the Import data files box to import them along with the rest of the scenario. If no data files are included in the scenario file, checking the box will have no effect. For more information about input data files and scenario exports, see Data files and exports.

Click IMPORT to import the selected scenario. (Click Cancel to close the dialog without importing a scenario.) The node selection dialog appears:



Choose nodes to be initialized from

Target	Import from
River [season 0]	River [season 0]
River [season 0] => Spawners Region 1 [season 0.5]	River [season 0] => Spawners Region 1 [season 0.5]
River [season 0] => Spawners Region 2 [season 0.5]	River [season 0] => Spawners Region 2 [season 0.5]
Spawners Region 1 [season 0.5]	Spawners Region 1 [season 0.5]
Spawners Region 1 [season 0.5] => Juv Region 1 [season 1]	Spawners Region 1 [season 0.5] => Juv Region 1 [season 1]
Spawners Region 2 [season 0.5]	Spawners Region 2 [season 0.5]
Spawners Region 2 [season 0.5] => Juv Region 2 [season 1]	Spawners Region 2 [season 0.5] => Juv Region 2 [season 1]
All Smolts [season 02]	All Smolts [season 02]
All Smolts [season 02] => Ocean [season 3]	All Smolts [season 02] => Ocean [season 3]
Juv Region 1 [season 1]	Juv Region 1 [season 1]
Juv Region 1 [season 1] => All Smolts [season 02]	Juv Region 1 [season 1] => All Smolts [season 02]
Juv Region 1 [season 1] => River [season 0]	Juv Region 1 [season 1] => River [season 0]
Juv Region 2 [season 1]	Juv Region 2 [season 1]
Juv Region 2 [season 1] => All Smolts [season 02]	Juv Region 2 [season 1] => All Smolts [season 02]
Juv Region 2 [season 1] => River [season 0]	Juv Region 2 [season 1] => River [season 0]
Ocean [season 3]	Ocean [season 3]
Ocean [season 3] => River [season 0]	Ocean [season 3] => River [season 0]

Scenario import node selection

This dialog enables you to map the parameter configuration of life cycle elements and transitions in the scenario file (listed in green in the right-hand column, under Import from) to those in the scenario you're creating (listed in black in the right-hand column under Target). In many cases, you will accept the default mappings.

For more on the mapping process, see Import node mapping.

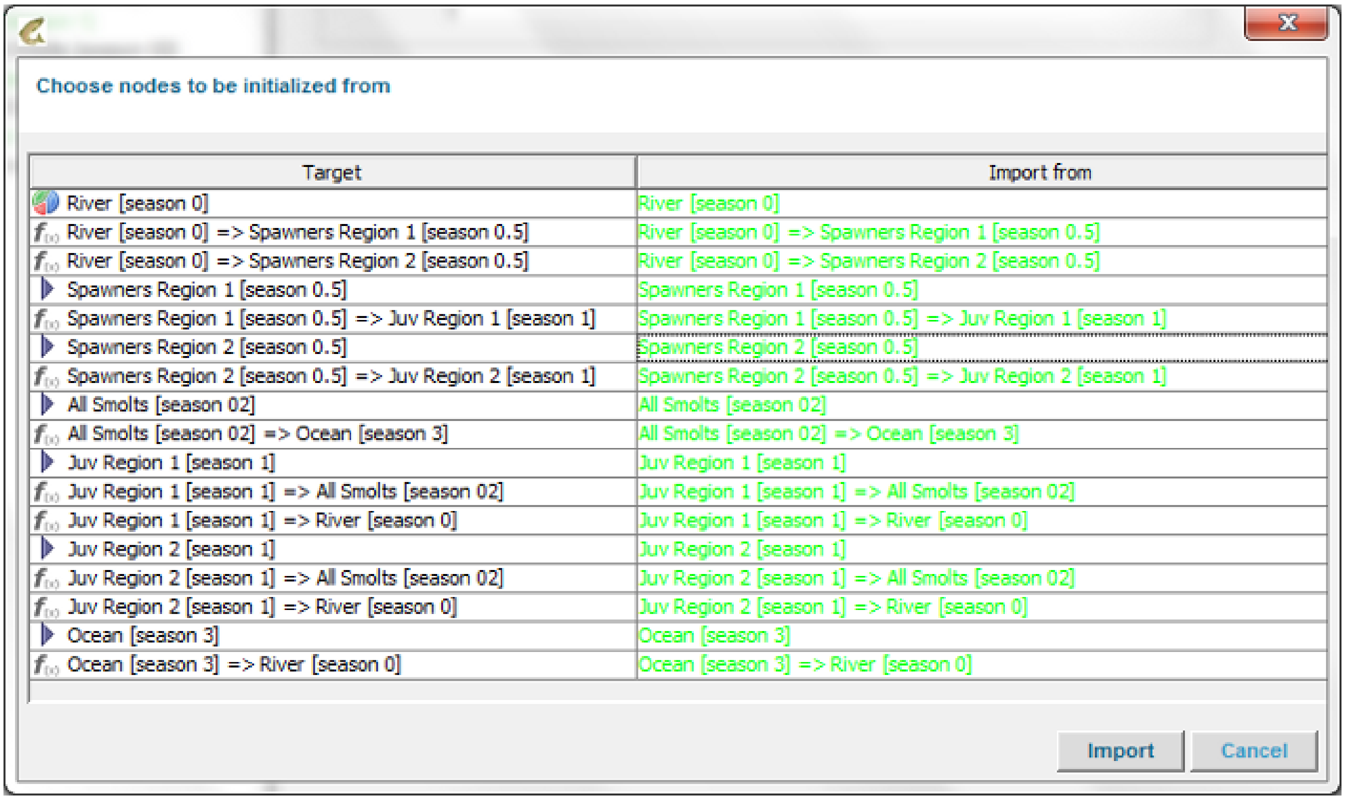
To accept the node selection, click Import . This completes the import process. To cancel the import, click Cancel .

Import node mapping

When you import a scenario, you are given the opportunity to choose which imported life cycle element configurations to apply to elements in the new, target scenario you are creating. Under most circumstances, you'll accept the default: map each imported configuration to the target scenario element with the same name as the imported name.

Sometimes, it may be useful to re-map imported configurations to different elements. When you do this, you are creating a new scenario that is based on, but not the same as the imported version.

Here's an example node-mapping dialog display:



Target	Import from
River [season 0]	River [season 0]
River [season 0] => Spawners Region 1 [season 0.5]	River [season 0] => Spawners Region 1 [season 0.5]
River [season 0] => Spawners Region 2 [season 0.5]	River [season 0] => Spawners Region 2 [season 0.5]
Spawners Region 1 [season 0.5]	Spawners Region 1 [season 0.5]
Spawners Region 1 [season 0.5] => Juv Region 1 [season 1]	Spawners Region 1 [season 0.5] => Juv Region 1 [season 1]
Spawners Region 2 [season 0.5]	Spawners Region 2 [season 0.5]
Spawners Region 2 [season 0.5] => Juv Region 2 [season 1]	Spawners Region 2 [season 0.5] => Juv Region 2 [season 1]
All Smolts [season 02]	All Smolts [season 02]
All Smolts [season 02] => Ocean [season 3]	All Smolts [season 02] => Ocean [season 3]
Juv Region 1 [season 1]	Juv Region 1 [season 1]
Juv Region 1 [season 1] => All Smolts [season 02]	Juv Region 1 [season 1] => All Smolts [season 02]
Juv Region 1 [season 1] => River [season 0]	Juv Region 1 [season 1] => River [season 0]
Juv Region 2 [season 1]	Juv Region 2 [season 1]
Juv Region 2 [season 1] => All Smolts [season 02]	Juv Region 2 [season 1] => All Smolts [season 02]
Juv Region 2 [season 1] => River [season 0]	Juv Region 2 [season 1] => River [season 0]
Ocean [season 3]	Ocean [season 3]
Ocean [season 3] => River [season 0]	Ocean [season 3] => River [season 0]

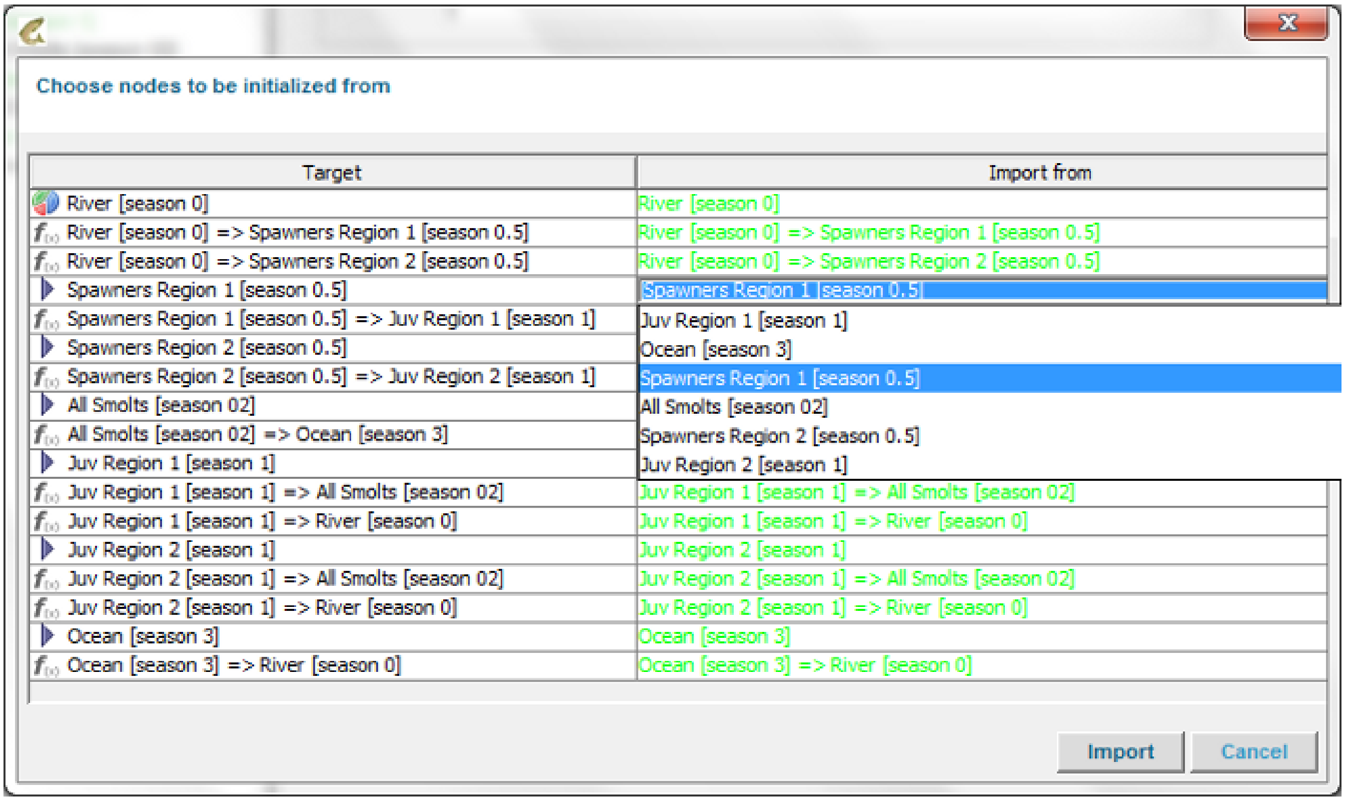
Scenario import node selection

Here the nodes (life cycle elements and transitions) to be created in your new scenario are listed in the left-hand 'Target' column and the imported node configurations are in the right-hand Import from column. Node types are indicated by the icon immediately to the left of the Target name:

Stages are marked with ▶
Splitting stages are marked with ●
Transitions are marked with $f_{(x)}$

In the default mapping shown, each new node is assigned the configuration of the imported node with the same label. To change this assignment, click the Imported from entry corresponding to the new node you want to re-map. A drop-down list appears that contains all import nodes of the same type as the one you are mapping.

For example, in the node mapping dialog shown above, clicking the entry to the left of target Spawners Region 1[season 0.5] results in this dropdown list display:



Target	Import from
River [season 0]	River [season 0]
River [season 0] => Spawners Region 1 [season 0.5]	River [season 0] => Spawners Region 1 [season 0.5]
River [season 0] => Spawners Region 2 [season 0.5]	River [season 0] => Spawners Region 2 [season 0.5]
Spawners Region 1 [season 0.5]	Spawners Region 1 [season 0.5]
Spawners Region 1 [season 0.5] => Juv Region 1 [season 1]	Juv Region 1 [season 1]
Spawners Region 2 [season 0.5]	Ocean [season 3]
Spawners Region 2 [season 0.5] => Juv Region 2 [season 1]	Spawners Region 1 [season 0.5]
All Smolts [season 02]	All Smolts [season 02]
All Smolts [season 02] => Ocean [season 3]	Spawners Region 2 [season 0.5]
Juv Region 1 [season 1]	Juv Region 2 [season 1]
Juv Region 1 [season 1] => All Smolts [season 02]	Juv Region 1 [season 1] => All Smolts [season 02]
Juv Region 1 [season 1] => River [season 0]	Juv Region 1 [season 1] => River [season 0]
Juv Region 2 [season 1]	Juv Region 2 [season 1]
Juv Region 2 [season 1] => All Smolts [season 02]	Juv Region 2 [season 1] => All Smolts [season 02]
Juv Region 2 [season 1] => River [season 0]	Juv Region 2 [season 1] => River [season 0]
Ocean [season 3]	Ocean [season 3]
Ocean [season 3] => River [season 0]	Ocean [season 3] => River [season 0]

Scenario import re-mapping

Since the target node is a life cycle stage, only the imported, non-splitting stages are available for mapping, and any of them can be selected. The Spawners Region 1[season 0.5] stage in your new scenario will be configured in exactly the same way as the import stage you select.

Once you're satisfied with the node mapping selections, click the Import button to complete the scenario import process.

Data Files and Imports

When you import a scenario with associated input data files, one of several things may happen:

- **Data files were embedded in the exported scenario** : If the data files are included in the scenario, you can choose whether to import the file contents along with the scenario by your selection of the Import data files .
  Import data files : If this box is checked, data file contents are read from the scenario's XML file and written to the SLAM datafiles directory. If there is already a file in that directory with the same name as the imported file, the new file name is modified by appending a numeric string; if this happens, SLAM displays a message explaining the changes.
  Don't import data files : If the import data files box is not checked, the scenario is imported but the data file contents are not extracted and written to the disk.
  Note : If data files aren't present when you load the scenario, you will be prompted for the location of a copy.
- **No data files embedded in the exported scenario** : If the scenario file was exported without including data file contents, then only the names of referenced files are present in the scenario file. Once you have imported the scenario and save it, you will be prompted for the location of a copy of any associated input data files.

When SLAM loads a scenario, it always checks for any referenced input data files. If it can't find them, it prompts you for the location of a copy of the files.

More about Exports and Imports

Exported file types

Three files are generated when you export a life cycle model or a scenario. They all have the same base name and are written to the same output directory, differing only by their extensions: xml , html , and png .

- **XML files** : These files contain the complete description of the exported life cycle or scenario. When you import an exported model, the XML file is required. For backups and transfers, the XML file is the one you are interested in.
- **HTML files** : These files contain a description of the exported model that becomes human-readable when rendered in a browser. The HTML file contains a link to the PNG file, so when you open the HTML file in your browser, you'll see the PNG file as well. It is not necessary to maintain the HTML file for backup or model-transfer purposes.
- **PNG files** : These files contain a picture of the graphical representation of the life cycle model, just as it appears in SLAM's life cycle display. The export-generated HTML file links to the PNG file, you can view it either by opening an image-viewing application on your computer or by opening the HTML file in your browser. It is not necessary to maintain the PNG file for backup or model-transfer purposes.

Default export file names

When you export a life cycle or scenario, SLAM generates a default name for the export files, like results_1_20120712-l for a life cycle or results_scenario_1-20120712-s for a scenario. (The three exported file types are named by appending the appropriate extensions to the base name.)

Life cycle model default names are formed by appending a date-stamp (of the form yyyymmdd followed by -l to the name of the lifecycle: for a life cycle named results_1 exported on 12 July, 2012, the base name results_1_20120712-l is generated.

Scenario export default names are formed in the same way as those for life cycle models, except that the name of the scenario is used instead of that of the life cycle, and -s is appended to the name rather than -l . If a scenario named results_scenario_1 is exported on 12 July, 2012, the default name is results_scenario_1_20120712-s.

You can change the default file name to anything you like. If you append an extension other than xml , html , or png , SLAM will treat the extension as part of the file name and append its extensions to it: if you specify foo.bar , your three exported files will be named foo.bar-l.xml, foo.bar-l.html, and foo.bar-l.png.

Export file formats

Both life cycle models and scenarios are exported in three file types: XML, HTML, and PNG. The HTML and PNG files are strictly descriptive and are intended to provide a human-readable profile of the exported entity. They are not used in the import process.

The XML files contain the complete, machine-readable description of the life cycle or scenario and are required for importing the exported entity back into SLAM. They can be examined in standard text editors or specialized XML viewers.

There is no schema defined for the XML files, as they are intended to be produced only by SLAM and imported only into SLAM.

The life cycle export XML files contain descriptions of all life cycle elements and transitions, including graphical information that permits drawing the life cycle graph.

A life cycle export file describes *only* the life cycle. No scenario information is included in the life cycle export file.

The scenario export XML files are more complex. They include complete descriptions of all functional relationships defined by the scenario, including function parameters and the bodies of any influence scripts assigned to the scenario. In addition, an XML description of the life cycle to which the scenario is attached is also included in the scenario's XML; this description is equivalent to the XML contained in the life cycle export file. Thus, the life cycle and the scenario can be imported from the scenario export file.

The scenario export file can also contain the contents of any input data files attached to the scenario. If the contents of these files are included in the scenario XML, they can be used to re-create the files when the scenario is imported.

Notes

This section contains notes on aspects of SLAM not covered in the rest of the manual. While you probably won't need to know much about the topics listed here in order to create models and run simulations, this information might be helpful if you're trying to understand a little more about how SLAM works.

SLAM directories

SLAM maintains several directories on the computer's drive where it keeps files that it writes to and reads from. While reading a file is not generally a problem, some care must be taken to insure that SLAM can write data to files when it needs to.

In order to maintain security and system file integrity, most modern operating systems do not grant write privileges over the entire file system to most user accounts. Applications like SLAM are often installed under accounts with heightened privilege levels but are actually run under accounts with lessened read-write access. This means that when SLAM is running, the operating system may not allow it to write to the directory into which it was installed.

For example, under Windows 7 SLAM typically is installed in a subdirectory of the Program Files (x86) directory. The installation includes database and script subdirectories, which contain files that SLAM must be able to modify as it runs. Unless the user runs SLAM while logged in as a system administrator (a questionable practice), the application will be blocked from writing to anything under Program Files (x86) , and SLAM would crash. Similar problems may arise under other OSes.

In order to be able to write to files, SLAM creates a directory called SLAM in a writable location in the file system into which, by default, it puts all output and read-write files. The location of this SLAM directory varies with the operating system, but it is generally in whatever is identified as the user directory . For example, on a Windows XP platform the SLAM directory is usually created under

C:\Documents and Settings\username\Application Data\

while on Windows 7 it lies under

\Users\username\AppData\Roaming

where username is a placeholder for the name of the logged-in user account.

When SLAM is installed, many of the subdirectories it needs are located under its installation directory. When these are accessed, SLAM copies the contents to new subdirectories under the writable SLAM directory. From then on, these directories are used.

In many cases you have the option of browsing to different input or output directories. When you do so, be sure your current login account has the privileges necessary for accessing those directories.

SLAM directory contents

SLAM
SLAMdefaults.slm - file containing last-used values for various input fields in SLAM.
- batch_simulations : exported batch simulation results
- datafiles : input data files
- db : SLAM databases
- dynamic_driver_scripts : scripts defining dynamic driver behavior
- exported_models : life cycle model and scenario export files
- influence-scripts : scripts defining transition influences
- OAT_simulation_data_export : exported OAT simulation results
- simulation_data_export : exported simulation results

- **split-influence-scripts** : scripts defining split influences

SLAM garbage collection

SLAM is a Java program and runs in a software framework called the Java Virtual Machine, or JVC. The JVC provides many of the services to Java processes that the operating system provides to non-Java programs.

The JVC limits the maximum amount of memory space that a Java program can use. A Java program *allocates* memory when it creates new software objects, which is a frequent occurrence in most programs. Each memory allocation counts against the maximum memory limit maintained by the JVC for the program.

The lifetime of most software objects is short, and when they are no longer needed by the program, their allocated memory can be returned to the JVC; this returned memory no longer counts against the program's memory limit.

Freed program memory is not returned to the JVC's memory store instantly when the object using the memory is no longer needed, as doing so could inflict a serious overhead penalty on the program. Instead, a background process called the *garbage collector* periodically sweeps through the program's memory allocation, identifying blocks of memory that are safe to reclaim.

In some cases, the garbage collector may be slow to reclaim newly-freed memory. In such cases, you can *suggest* that it run a memory sweep.

SLAM displays the current memory allocated to itself, along with the upper limit allowed by the JVC, in the Memory use section of the interface along the bottom of the main window, near the right corner. To the right of this display is the Garbage collection button 🗑. Clicking this tells SLAM to pass along the hint to the JVC that running the garbage collector would be a good idea.

Usually, you'll see the allocated memory drop when you click the garbage collection button. Repeatedly clicking it will cause the allocated memory to drop to near the minimum memory required for SLAM to run in its current state.

Examples

This section contains descriptions of how to construct example life cycle models and scenarios. You might find that working with these samples helps you to understand SLAM a little more than simply reading the manual.

As more example models are produced for SLAM, they will be made available at the SLAM project website:

http://ecologybox.org/user/SLAM/SLAM

Look in the examples folder for import files containing life cycle models and scenarios.

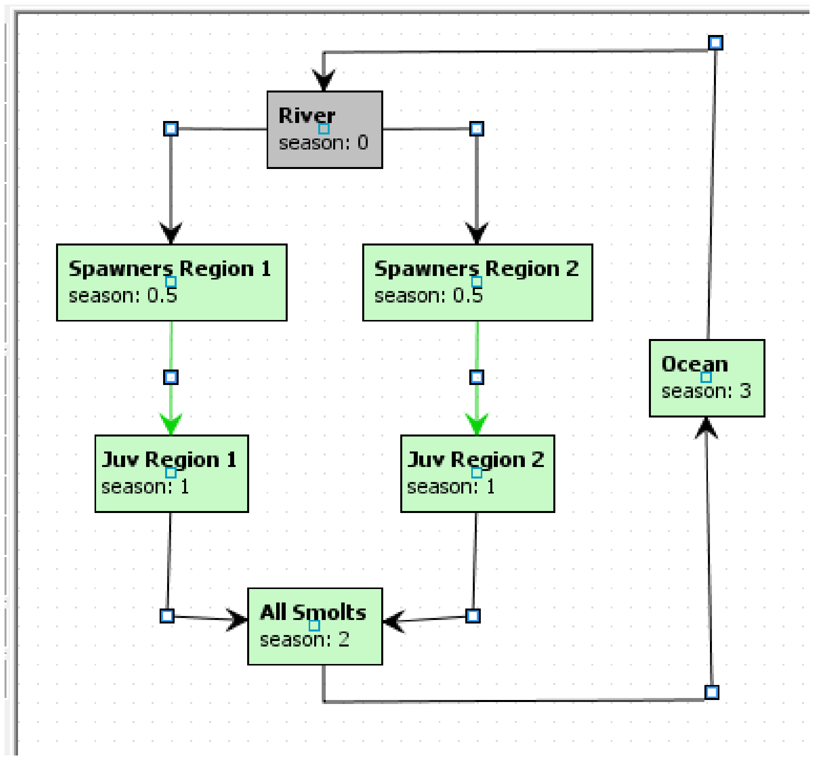
Models:

- Sample 1 life cycle
- sc_1 sample scenario

Creating the sample 1 life cycle model

These steps will take you through the creation of the sample 1 life cycle model[†]. If you're using the SLAM sample database, there will already be a life cycle model named sample_1 , so you'll either need to use a different name for the model you create or change databases and save your model in the new database.

Here's a graph of the completed life cycle model:



Complete sample 1 life cycle

1. River life stage
2. Spawners Region 1 stage
3. Spawners Region 2 stage
4. Remaining stages

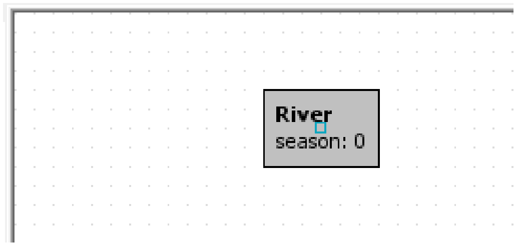
[†]*The sample 1 life cycle model exists solely for the purpose of demonstrating SLAM; any resemblance between it and a model of an actual population would be both unintended and remarkable.*

Creating the River stage

With a new life cycle model open, create a new stage. Assign these parameters:

Season : 0
Name : River
Pooled group : none (leave blank)
Initialize abundance ... : leave unchecked
Description : Anything you like

You'll have a gray rectangle labeled River with a season of 0. Drag it to a point a bit below and to the right of the top-left corner of the life cycle graph display.



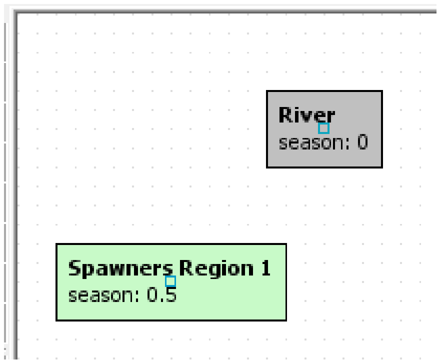
Sample 1 : River stage

Creating Spawners Region 1

Create a new stage. Assign these parameters:

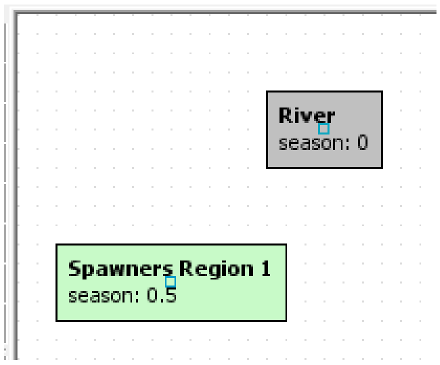
Season : 0.5
Name : Spawners Region 1
Pooled group : none (leave blank)
Initialize abundance ... : check this box
Description : Anything you like

You'll have a green rectangle labeled Spawners Region 1 with a season of 0.5. Drag it to a point below and a little to the left of the River stage.



Sample 1 : Spawners Region 1 stage

Create a transition by dragging the mouse from the center of River to Spawners Region 1 . You can change the layout of the transition from a straight line by dragging the small square near its center to a new location.



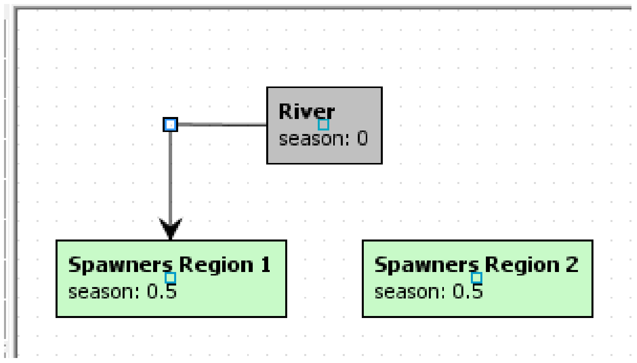
Sample 1 : River Spawners Region 1 transition

Creating Spawners Region 2

Create another new stage as you did for Spawners Region 1. Assign these parameters:

Season : 0.5
Name : Spawners Region 2
Pooled group : none (leave blank)
Initialize abundance ... : check this box
Description : Anything you like

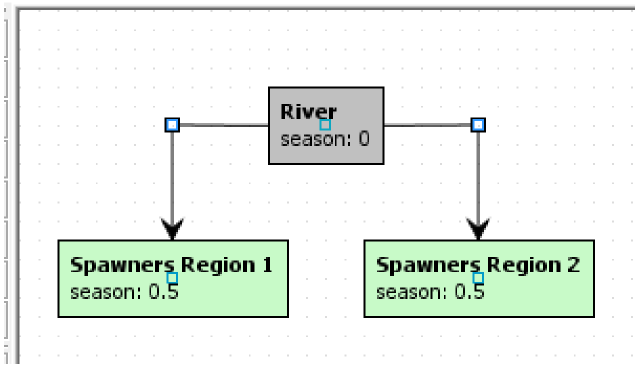
Position the green rectangle for Spawners Region 2 below and a little to the right of the River stage.



Sample 1 : Spawners Region 1 stage

Again as you did for Spawners Region 1, create a transition from River to Spawners Region 2 .

Even though a transition already exists from River to Spawners Region 1 , you can create a new one to Spawners Region 2 . By doing so, you turn River into a splitting stage.



Sample 1 : River Spawners Region 1 transition

Creating remaining stages

Create the remaining stages in the sample 1 model:

Juv Region 1 :
Season : 1
Name : Juv Region 1
Pooled group : none (leave blank)
Initialize abundance ... : check this box
Description : Anything you like

Place below Spawners Region 1 and create a transition from Spawners Region 1 to Juv Region 1 . Designate the transition as reproductive by editing it: double-click it, and check the reproductive transition check box.

Juv Region 2 :
Season : 1
Name : Juv Region 2
Pooled group : none (leave blank)
Initialize abundance ... : check this box
Description : Anything you like

Place below Spawners Region 2 and create a transition from Spawners Region 2 to Juv Region 2 . As above, designate the transition as reproductive by editing it: double-click it, and check the reproductive transition check box.

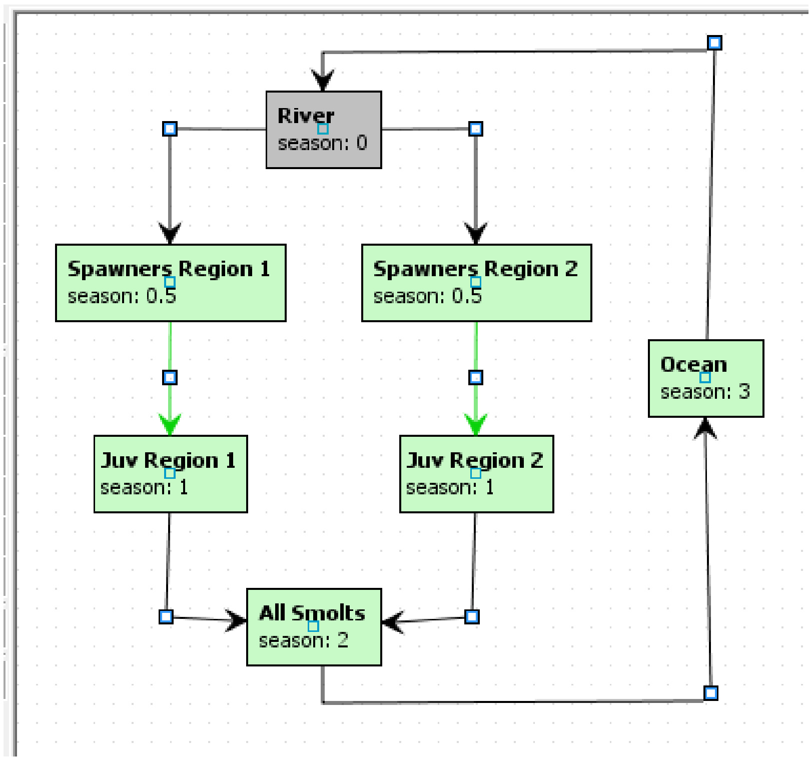
All Smolts :
Season : 2
Name : All Smolts
Pooled group : none (leave blank)
Initialize abundance ... : check this box
Description : Anything you like

Place below and between Juv Region 1 and Juv Region 2 . Create transitions from *both* Juv Region 1 and Juv Region 2 to All Smolts .

Ocean :
Season : 3
Name : Ocean
Pooled group : none (leave blank)
Initialize abundance ... : check this box
Description : Anything you like

Place to the right of the rest of the stages. Create a transition from All Smolts to Ocean and from Ocean to River .

The sample 1 model is complete and should look something like this:



Complete sample 1 life cycle

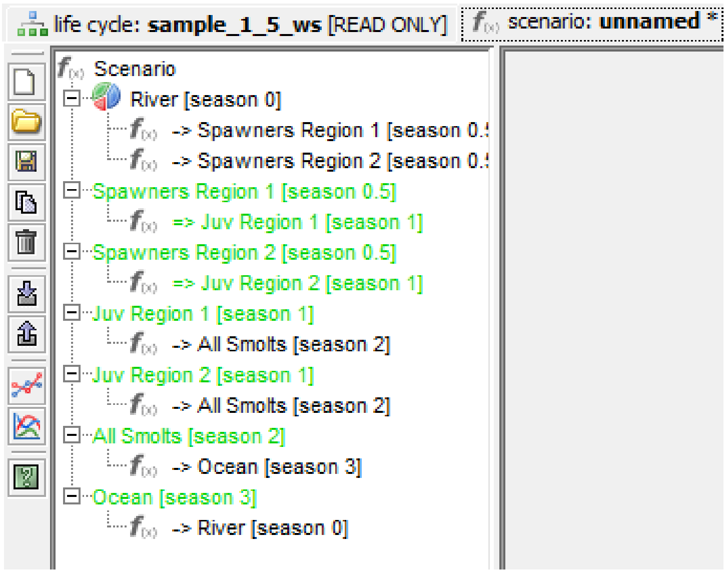
Remember that the layout of the graph, while it may help or hinder understanding of the model, doesn't matter to SLAM when it runs a simulation, as long as all the life cycle elements are in place and connected properly.

Save your new life cycle model to your database.

Creating the sample 1   sc_1   scenario

These steps will take you through the creation of the sc_1 scenario for the sample 1 life cycle model[†]. If you're using the SLAM   sample   database, there will already be an sc_1 scenario attached to the sample_1 life, so you'll either need to use a different name for the scenario you create or change databases and save your life cycle and scenario in the new database.

With the sample_1 life cycle loaded, begin by creating a new scenario. The scenario editor tree control will look like this:



life cycle: **sample_1_5_ws** [READ ONLY]   scenario: **unnamed** *

```
Scenario
  River [season 0]
    -> Spawners Region 1 [season 0.5
    -> Spawners Region 2 [season 0.5
  Spawners Region 1 [season 0.5]
    => Juv Region 1 [season 1]
  Spawners Region 2 [season 0.5]
    => Juv Region 2 [season 1]
  Juv Region 1 [season 1]
    -> All Smolts [season 2]
  Juv Region 2 [season 1]
    -> All Smolts [season 2]
  All Smolts [season 2]
    -> Ocean [season 3]
  Ocean [season 3]
    -> River [season 0]
```

New scenario for sample 1 life cycle

Save the scenario as   sc_1   and the scenario tab label will display **sc_1** instead of **unnamed**.

You can examine the default scenario settings.

- Initialized stages are all set to constant values of 1000.
- Splitting stages are all configured as generator types, with fixed 1:1 split ratios (no influences are configured, even if they exist). No uncertainty or annual variability is defined.
- Transitions are all described by linear functions with a slope of one, so they simply copy the input abundance to the output. Again, no influences are configured.

To configure the new scenario, follow these steps:

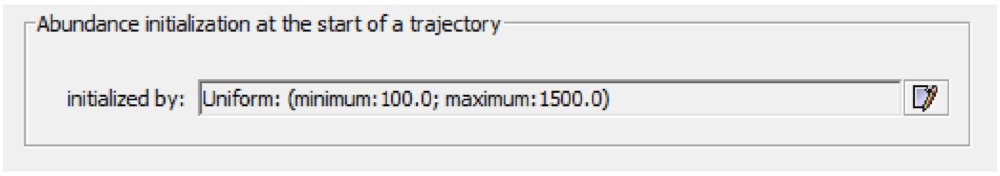
1. Stage initialization
2. Simple transition
3. Complex transitions

[†]*Like the sample 1 life cycle model, the sc_1 sample scenario exists solely for the purpose of demonstrating SLAM; any resemblance between it and a model of an actual population is unlikely.*

Stage initialization

In order to characterize the behavior of the model when stage initializations vary from trajectory to trajectory, you can modify the initial abundance of a stage so that at the beginning of each trajectory it takes on a value drawn from a random distribution.

With your sc_1 scenario loaded into the scenario editor, select the Spawners Region 1 [season 0.5] stage in the selection tree. Edit the stage's initialization, changing it from a constant of 1000 to a draw from a uniform distribution with minimum 500 and maximum 1500. When you're done the Abundance initialization at the start of a trajectory field will look like this:


Abundance initialization at the start of a trajectory

initialized by: Uniform: (minimum:100.0; maximum:1500.0)

Spawners Region 1 [season 0.5] initialization

If you save your scenario and run a simulation, you'll see a random variation in the early time steps for the Spawners Region 1 abundance.

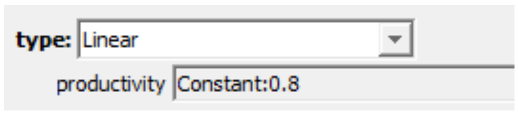
Now that you've changed a stage initialization, try modifying one of the transition function definitions.

Simple transition

The nature of transition function relationships plays a major role in determining the overall behavior of life cycle models. This step in setting up sample scenario sc_1 makes a simple change to one transition in the sample 1 life cycle.

With sc_1 loaded into the scenario editor, select the Ocean River transition in the selection tree and edit the productivity field of the transition function.

The productivity is by default set to a constant value of one. Edit the parameter to change its value to 0.8. When you're finished, the function editing fields for the transition will look like this:


type: Linear
productivity Constant:0.8

Ocean [season 3] -> River [Season 0] transition

If you save the sc_1 scenario and run a simulation, you'll see the effect of this transition function modification on the model's abundances over time.

In real life, your models may involve more complex transition definitions. In the next scenario definition step, you'll see how to define such transitions.

Complex transitions

SLAM enables you to create complex relationships between stock stages and the recruit stages they transition to. In this sc_1 scenario configuration step, two transitions are given functional forms different from the simple linear relationship of the previous step.

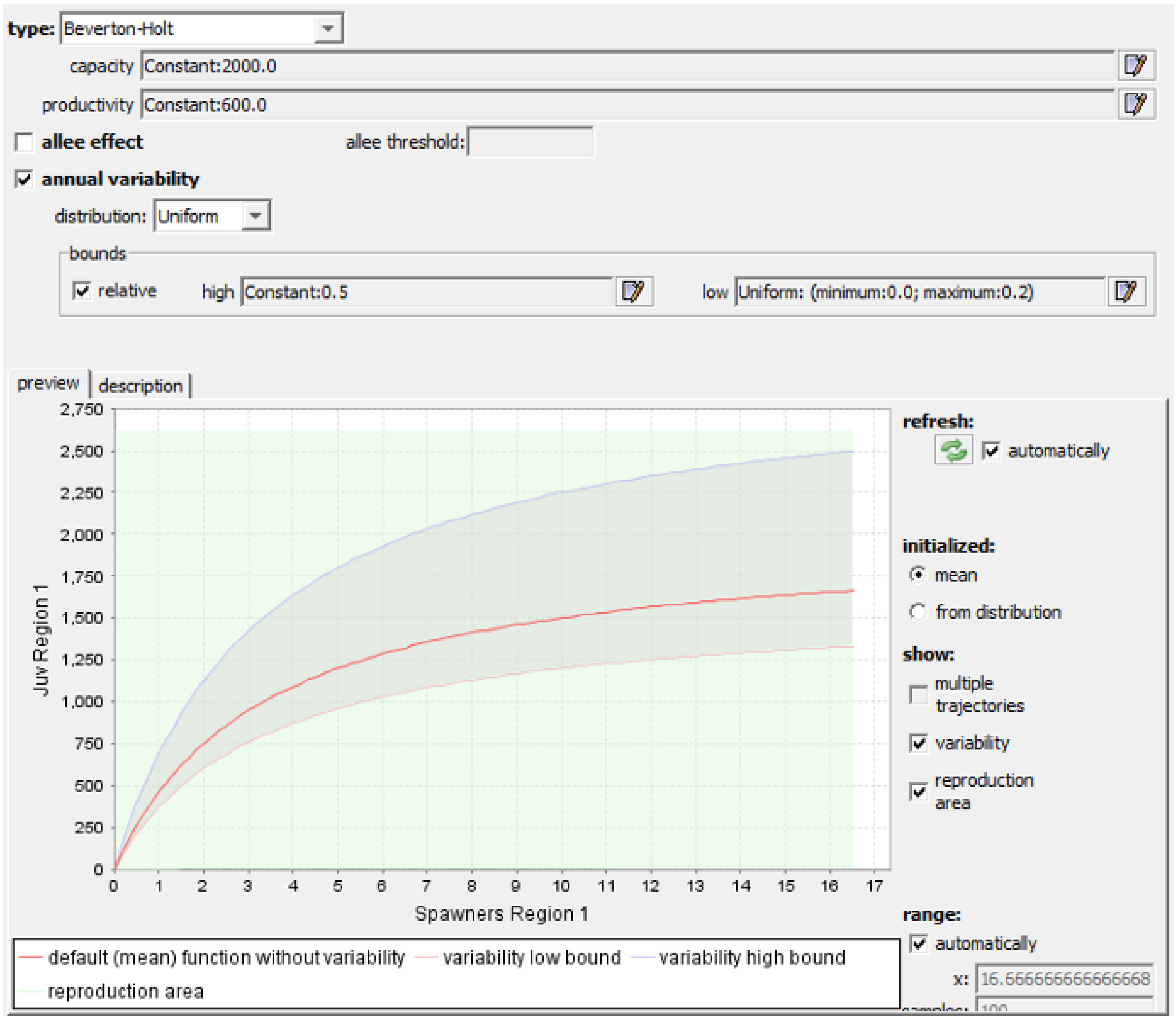
Spawners Region 1 **Juv Region 1** : edit the transition function as follows:

- **type** : Set to Beverton-Holt.
- **capacity** : Set to Constant:2000
- **productivity** : Set to Constant:600
- **annual variability** : Check this box and set the following parameter values:
    - **distribution** : Uniform, s. dev. = 0.3 (constant)
    - **bounds** : relative : checked
        - high : Constant:0.5
        - low : Uniform, minimum = 0, maximum = 0.2

Spawners Region 2 **Juv Region 2** : edit the transition function as follows:

- **type** : Set to Hockey-Stick.
- **capacity** : Set to Uniform, minimum=3000, maximum=5000.
- **productivity** : Set to Uniform, minimum=3000, maximum=5000.
- **annual variability** : Check this box and set the following parameter values:
    - **distribution** : Lognormal, s. dev. = Constant:0.3
    - **bounds** : relative : *not* checked
        - high : Uniform, minimum = 0, maximum = 0.2
        - low : Uniform, minimum = 0, maximum = 0.2

When you have made these changes, the Spawners Region 1 Juv Region 1 transition will look like this in the scenario editor:


type: Beverton-Holt

capacity Constant:2000.0

productivity Constant:600.0

☐ allee effect   allee threshold:

☑ annual variability

distribution: Uniform

bounds

☑ relative   high Constant:0.5   low Uniform: (minimum:0.0; maximum:0.2)

preview | description

refresh: ☑ automatically

initialized: [✓ mean] ○ from distribution

show: ☐ multiple trajectories ☑ variability ☑ reproduction area

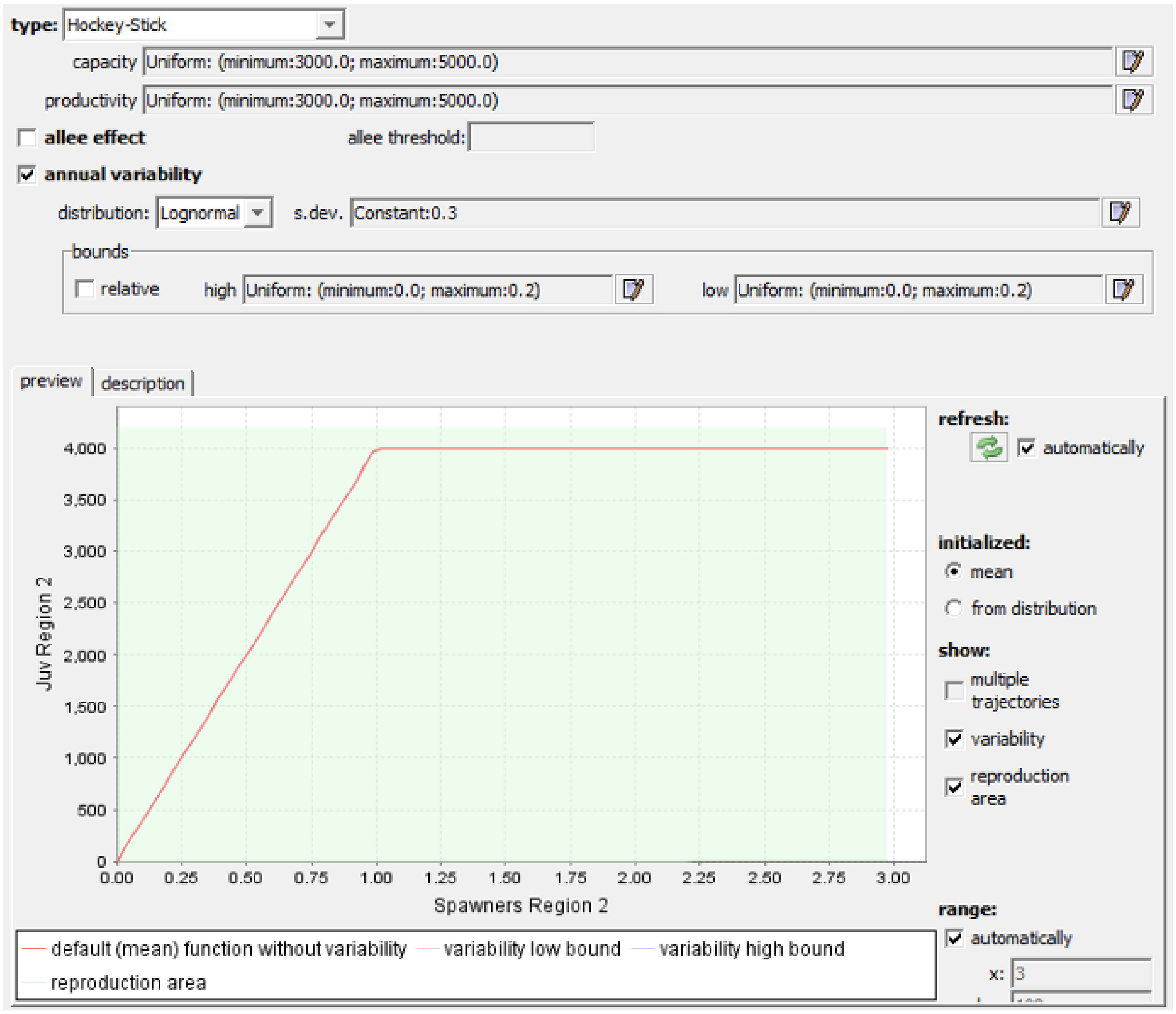
range: ☑ automatically

x: 16.666666666666668

samples: 100

Spawners Region 1   Juv Region 1 transition function

The Spawners Region 2 Juv Region 2 transition will look like this:


type: Hockey-Stick

capacity Uniform: (minimum:3000.0; maximum:5000.0)

productivity Uniform: (minimum:3000.0; maximum:5000.0)

☐ allee effect allee threshold:

☑ annual variability

distribution: Lognormal s.dev. Constant:0.3

bounds

☐ relative high Uniform: (minimum:0.0; maximum:0.2) low Uniform: (minimum:0.0; maximum:0.2)

refresh: ☑ automatically

initialized: ● mean ○ from distribution

show: ☐ multiple trajectories ☑ variability ☑ reproduction area

range: ☑ automatically x: 3

Spawners Region 2 Juv Region 2 transition function

Save the scenario and run a simulation. You will see a significant difference in the behavior of populations from that of the previous scenario.

Terms