California Individual-based Fish Simulation System

Stream Trout Model

User Guide and Software Documentation

VERSION 2

LITTLE JONES CREEK CUTTHROAT MODEL

Last updated: March, 2001

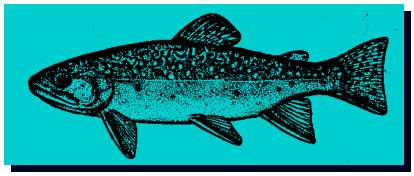
Prepared by:

Steve Railsback
Lang, Railsback & Associates
Arcata CA

Steve Jackson
McKinleyville CA

Prepared for:

US Forest Service, Redwood Sciences Laboratory



CIFSS
http://math.humboldt.edu/~simsys

The Swarm Project
www.swarm.org

Table of Contents

I. INTRODUCTION 1

I.A. DOCUMENT OBJECTIVES 1

I.B. SOFTWARE LICENSE AND CONDITIONS 1

I.C. DOCUMENT REVISION HISTORY 2

II. CIFSS PHILOSOPHY 3

II.A. CIFSS CONVENTIONS 3

II.A.1. Dates 3

III. TROUT MODEL SOFTWARE OVERVIEW 4

III.A. OVERVIEW 4

IV. USER GUIDANCE: OVERVIEW 5

IV.A. SOFTWARE INSTALLATION AND EXECUTION 5

V. FORMULATION AND INPUT TESTING AND REVISION 5

VI. SOURCE CODE REVISION 6

VI.A. CUSTOMIZING THE MAKEFILE 6

VI.B. CHANGING GRAPHICS AND USER INTERFACES 6

VI.B.1. Graphs 6

VI.C. AUTOMATED EXPERIMENTS 6

VII. SETUP, PARAMETER, AND DATA FILES 7

VII.A. SETUP FILES 7

VII.A.1. Experiment Swarm setup file 7

VII.A.2. Model Swarm setup file 9

VII.B. DATA FILES 13

VII.B.1. Hydraulic data (depth and velocity lookup table) 13

VII.B.2. Time series habitat data 13

VII.B.3. Barrier data 14

VIII. SOFTWARE INSTALLATION AND EXECUTION 15

IX. OUTPUT FILES AND OUTPUT PROCESSING 16
IX.A. FISH OUTPUT FILE 16
IX.B. FISH MORTALITY FILE 16
IX.C. REDD OUTPUT FILE 16
IX.D. REDD MORTALITY FILE 16
IX.E. DEPTH AND VELOCITY AVAILABILITY FILES 17
IX.F. DEPTH AND VELOCITY USE FILES 17
IX.G. CELL-BASED FISH INFORMATION 17
X. SOFTWARE TESTING 19
XI. CONDUCTING MODELING EXPERIMENTS 20
XII. SOURCE CODE DESCRIPTION AND DIRECTORY 21
XII.A. CODE DIRECTORY 21
XII.A.1. ExperSwarm 21
XII.A.2. ScenarioIterator 22
XII.A.3. TroutObserverSwarm 22
XII.A.4. TroutModelSwarm 23
XII.A.5. HabitatSpace 26
XII.A.6. Cell 29
XII.A.7. Trout 32
XII.A.8. Species1 36
XII.A.9. SurvivalProb 36
XII.A.10. Redd 37
XII.A.11. Barrier 38
XII.A.12. FileInput 38
XII.A.13. StationObject 38
XII.A.14. TimeWrapper 39
XII.A.15. Logistic 40

XIII. RESEARCH AND DEVELOPMENT PRIORITIES 42

XIV. QUALITY CONTROL DOCUMENTATION 43

XV. REFERENCES..... 45

I. Introduction

I.A. Document Objectives

This report provides user guidance and program documentation for the second stream trout model developed with the California Individual-based Fish Simulation System (CIFSS), a cutthroat trout model developed for Little Jones Creek, California. The formulation of this model is documented by Railsback and Harvey (in prep.).

Software for the first CIFSS trout model was fully documented in a report prepared for the Electric Power Research Institute (EPRI 1999). To avoid redundancy, this report documents only the significant changes in software implemented for the Little Jones Creek model. Therefore, full documentation of this Version 2 of the stream trout model software is provided by both this report and the EPRI (1999) report. This report retains the same major section headings, but contains text only to describe changes since Version 1.

This document accurately reflects versions of the model archived in March, 2001.

I.B. Software License and Conditions

Swarm is distributed under the Library GNU General Public License as published by the Free Software Foundation. The CIFSS software is developed and distributed under the GNU General Public License as published by the Free Software Foundation. A full copy of the license is included with the software. Various versions of the software are copyrighted by their sponsors.

Under the terms of the GNU license, distribution of the software is controlled by its copyrighter; proprietary versions of the software can be produced and sold or otherwise distributed. However, anyone who receives a copy of the executable code has the right to obtain and modify the source code.

The CIFSS codes are distributed in the hope that they will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

I.C. Document Revision History

Date	Author	Changes Made	Basis
5/1998	GR	Original draft.	G. Ropella's documentation for prototype.
8/16/1999	SFR, SJ	Version 1 completed.	Version 1 of model software and formulation.
3/17/2000 4/14/2000	SFR, SJ	Version 2 changes.	Version 2 of software; cutthroat model formulation.
6/7/2000	SFR	Updates for new outputs: habitat availability and use, summarized mortality file.	Allow observation of habitat preferences.
1/18/2001	SFR	New outputs: fish density by cell, summarized redd mortality, additional age class.	Facilitate a variety of model experiments.
3/1/2001	SFR	Minor updates and cleanup.	Preparation for archiving with V. 2 of code.

II. CIFSS Philosophy

II.A. CIFSS Conventions

One change in conventions was made in Version 2, how date values are input and handled.

II.A.1. Dates

Dates are no longer specified with the year and Julian day. Instead, dates are input in a standard "MM/DD/YYYY" format, and converted by the software to an internal C date format (the number of seconds since the start of 1970). Examples of this format are provided for input files (Sect. VII.A.2).

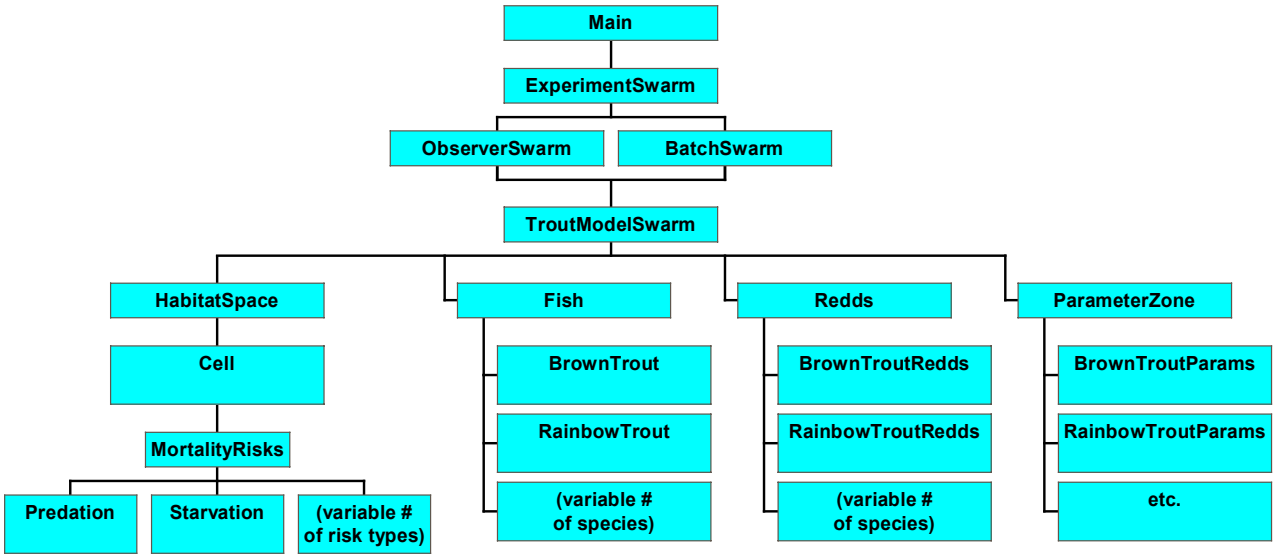
Input specifying a day of the year (e.g., the day on which spawning season starts) are now in the "MM/DD" format, e.g., `5/15` for May 15th.

It no longer makes any difference if input data are in calendar or water years; the true calendar date must be used for all input.

III. Trout Model Software Overview

III.A. Overview

In Version 2, the experiment swarm and batch swarm are fully implemented, so the following model organization is now completely functional.



(Note that while Version 2 of the trout model is parameterized as a single-species cutthroat trout model, the software can still be used for multi-species models.)

IV. User Guidance: Overview

IV.A. Software Installation and Execution

The code is now fully compatible with Version 2.0.1 and Version 2.1.1 of *Swarm*. *Swarm* no longer requires the Makefile to be checked and possibly modified when a code is installed on a computer; the Makefile automatically finds the *Swarm* file necessary to compile the code.

V. Formulation and Input Testing and Revision

(No changes since Version 1.)

VI. Source Code Revision

VI.A. Customizing the Makefile

Swarm no longer requires the user to edit the Makefile to provide the *Swarm* home directory. Therefore, users should normally be able to install and compile the trout model software with no changes to the Makefile or other source code files.

VI.B. Changing Graphics and User Interfaces

VI.B.1.Graphs

The code file `TroutObserverSwarm.m` now contains a method `-drop` that closes graphic objects at the end of each simulation. This method is necessary for using the Experiment Swarm (which generates multiple model runs) in graphics mode. If new graphs are added to the model, new code to drop the models must be added to the `-drop` method.

VI.C. Automated Experiments

The Experiment Swarm for automating model experiments is now implemented. The Experiment Swarm can (a) produce scenario simulations in which the values of parameters or variables are varied, and (b) replicates of each scenario, which vary only by the random number sequences. Currently, experiments are completely controlled by information in a setup file (Sect. VII.A.1). More sophisticated experiments that, for example, calculate new parameter values from the results of previous simulations, could be programmed in the Experiment Swarm.

VII. Setup, Parameter, and Data Files

VII.A. Setup Files

The Experiment Swarm setup file is new.

VII.A.1.Experiment Swarm setup file

This file must be named `Experiment.Setup`. It contains information used by the Experiment Swarm, which executes multiple model runs (Sect. VI.C). The Experiment Swarm setup file controls both (a) one or more *scenarios*, in which parameter or variable values are changed, and (b) one or more *replicates* of each scenario, replicates being separate simulations differing only in the random number sequence used. Setting up scenarios involves specifying which parameter to vary among scenarios, and providing parameter values for each scenario. The same number of replicate model runs are generated for each scenario.

Replicate model runs are generated by multiplying the random number seed (provided in the Model Swarm setup file; Sect. VII.A.2) by the replicate number. For example, if the random number seed is 2001, the first replicate run initializes the random number generator with the seed 2001, the second replicate initializes the generator with seed 4002, and the third replicate uses seed 6003.

This setup file starts with the following blocks of text. Comments can be included as lines starting with the character #.

- Three header lines, not used by the computer. These can be up to 200 characters long.
- A blank line
- Two lines on which (a) the number of scenarios and (b) the number of replicates for each scenario are specified. If the number of scenarios is specified as zero, then the Experiment Swarm makes no changes to parameter values and only conducts the specified number of replicates.
- A blank line
- Two line that provide the variable name and class to which the code sends the current scenario count, during model execution. This should normally not be changed.
- A blank line
- Two line that provide the variable name and class to which the code sends the current replicate count. This should normally not be changed.
- A blank line

Following these initial blocks of text, the file contains one or more additional blocks, which each specify a model parameter to be varied among scenarios and the value it has each scenario. These blocks contain the following lines, and are separated by blank lines.

- The word "ClassName" followed by the name of the class in which the parameter value is defined. "Class" refers to the object-oriented software structure in which each object in the model is an instance of a particular class. The major classes are shown in the figure in Sect. III.A. If necessary, the class for a particular parameter, and the parameter's value type, can be found by searching the code's header (.h) files.

- The word "ParamName" followed by the name of the parameter to be varied.

- The word "ValueType" followed by the kind of value that the parameter contains. The value type must be one of the types defined below.

- The word "Value" followed by the parameter's value for the first scenario. This line is repeated for the second and following scenarios.

The "ValueType" field must contain one of the following words.

- `filename`: the parameter is the name of an input file; the parameter values are names of different input files to be used in different scenarios).

- `date`: the parameter is a date in MM/DD/YYYY format.

- `int`: the parameter is an integer.

- `bool`: the parameter is a boolean variable, with a value of either `YES` or `NO` (these values must be all upper-case, following computer science convention for boolean variables).

- `float`: the parameter is in floating point format.

- `double`: the parameter is in double-precision floating point format.

The following is an example Experiment Swarm setup file.

```
Experiment setup file
created Jan 25 2000
3rd of 3 header lines

numberOfScenarios    2
numberOfReplicates   5

sendScenarioCountToParam: scenario
inClass:                  TroutModelSwarm

sendReplicateCountToParam: replicate
inClass:                   TroutModelSwarm

ClassName    TroutModelSwarm
```

```
ParamName    appendFiles
ValueType    BOOL
Value        YES
Value        YES

ClassName    TroutBatchSwarm
ParamName    modelSetupFile
ValueType    filename
Value        Model1.Setup
Value        Model2.Setup

ClassName    TroutModelSwarm
ParamName    flowFile
ValueType    filename
Value        LJCFlowScen1.Data
Value        LJCFlowScen2.Data

#ClassName    TroutModelSwarm
#ParamName    runStartDate
#ValueType    date
#Value        1/1/2000
#Value        1/1/2000

ClassName   FishParams
ParamName   fishSpawnProb
ValueType   float
Value       0.9
Value       0.5
```

VII.A.2.Model Swarm setup file

The Model Swarm setup file `Model.Setup` has been modified to accommodate the new ability of Version 2 to read multiple hydraulic data files (Sect. VII.B.1) and to provide the name of the new barrier data file (Sect. VII.B.3).

Instead of providing the parameter `hydraulicsFile` with one hydraulic data file name:

```
hydraulicsFile                LJCLowHyd.Data
```

the user now specifies up to five separate hydraulic files. These have parameter names `hydraulicsFile1`, `hydraulicsFile2`, etc. The file `Model.Setup` now typically has lines such as these.

```
hydraulicsFile1               LJCLowHyd1.Data
hydraulicsFile2               LJCLowHyd2.Data
hydraulicsFile3               LJCLowHyd3.Data
hydraulicsFile4               LJCLowHyd4.Data
```

The Model Swarm setup file now must also include the parameter `barrierFile` and its value, which is the name of a barrier data file (Sect. VII.B.3). This file must exist even if there are no barriers in the reach being modeled.

Dates in `Model.Setup` are now specified in MM/DD/YYYY format. For example, instead of separate variables for the model start year and Julian date, there is one variable for model start date.

An example `Model.Setup` file is provided, with explanations below.

```
#
# Currently the file names should be 35 characters or less
#

@begin

randGenSeed                          32461
numberOfSpecies                          1
habParamFile                LJCHab.Params
cellDataFile                 LJCLowCell.Data
flowFile                     SeasFlow100.Data
temperatureFile              SeasTemp100.Data
turbidityFile                SeasTurbid.Data
barrierFile                 LJCLowBarrier.Data

hydraulicsFile1              LJCLowHyd1.Data
hydraulicsFile2              LJCLowHyd2.Data
hydraulicsFile3              LJCLowHyd3.Data
hydraulicsFile4              LJCLowHyd4.Data

runStartDate                       7/1/2000
runEndDate                         12/31/2000
popInitDate                        7/19/1998
fishOutputFile               LJCFishCal.Out
fishMortalityFile           LJCDeadFishCal.Out
reddMortalityFile              LJCReddMort.Out
reddOutputFile                 LJCRedds.Out

tagFishColor                   orange

fileOutputFrequency                    5
appendFiles                  0

depthBinWidth                  10
velocityBinWidth               10
depthMaxBin                    150
velocityMaxBin                 100

depthAvailabilityFileName      depthAvailability.Out
velocityAvailabilityFileName   velocityAvailability.Out

fishDepthUseFileName           DepthUse.Out
fishVelocityUseFileName        VelocityUse.Out

@end
```

Model swarm setup file variables and meeting.

Variable Name	Meaning
`randGenSeed`	Sets the random number generator seed (integer). Changing the seed creates a new sequence of random numbers. *Not* changing the random seed produces the same sequence of random numbers so the exact same

	model run can be repeated.
numberOfSpecies	Specifies the number of fish species in the model.
habParamFile	Provides the habitat parameter file name. The file name convention is *Hab.Param where "*"defines the site (e.g., TuleHab.Param).
cellDataFile	Specifies the cell geometry and data file name. The convention for this file name is *Cell.Data where "*" defines the site.
flowFile	Specifies the name of the file with daily flow data. The convention for this file name is *Flow.Data where "*" defines the site.
temperatureFile	Specifies the name of the file with daily temperature data. The convention for this file name is *Temp.Data where "*" defines the site.
turbidityFile	Specifies the name of the file with daily turbidity data. The convention for this file name is *Turbid.Data where "*" defines the site.
barrierFile	Specifies the name of the file with locations of barriers to upstream movement. The filename convention is *Barrier.Data.
hydraulicsFile1 hydraulicsFile2 …	Specifies the names of the files with the depth and velocity lookup data. (These files are created by RHABSIM.) The convention for this file name is *Hyd#.Data where "*" defines the site and "#" is the hydraulics file number (1-5).
runStartDate	Date that the simulation starts.
runEndDate	Date that the simulation ends.
popInitDate	The date for the population data used to initialize the model. This date must exist in the population initialization files, but need not be the same as the model start date.
fishOutputFile fishMortalityFile	Provides names for file output. See Sect. IX for file descriptions.

`reddOutputFile`
 	
`reddMortalityFile`	

`fileOutputFrequency`	Sets the number of time steps that execute before file output is written. For example, if the value is 7, then output is written once per week. (Note that the code does *not* average output over the time period between outputs.) This variable is in the Model Swarm instead of the Observer Swarm so it can be used in batch mode.
`appendFiles`	A boolean variable determining whether output files are appended vs. overwritten when each new replicate simulation starts. The value must be `0` or `1` (not `YES` or `NO`, even though `YES` and `NO` are used for this parameter in the Experiment.Setup file). A value of `1` means files are appended instead of overwritten. Normally this parameter should be set to `0` so that a new version of the files is created each time the model is started. The Experiment Swarm setup file should be set so `appendFiles` is `YES` for multiple model runs (Sect. VII.A.1). (This variable can be left out of Model.Setup, which is generally best.)
`tagFishColor`	The color that fish turn when "tagged" via probes in the animation window. The value can be the name of any common color.
`depthBinWidth`	The width (cm) of depth histogram bins used in habitat and fish habitat use file output (see Sect. IX concerning this output.)
`velocityBinWidth`	The width (cm/s) of velocity histogram bins used in file output.
`depthMaxBin`	The maximum depth in the histogram bins used in habitat and fish habitat use file output (see Sect. IX concerning this output.)
`velocityMaxBin`	The maximum velocity in the histogram bins for file output.
`depthAvailabilityFile-Name`	The name of the file for depth habitat availability output (see Sect. IX concerning this output.)

velocityAvailabilityFileName	The name of the file for velocity habitat availability output (see Sect. IX concerning this output.)
fishDepthUseFileName	The name of the file for depth fish habitat use output (see Sect. IX concerning this output.)
fishVelocityUseFileName	The name of the file for velocity fish habitat use output (see Sect. IX concerning this output.)

VII.B. Data files

Changes to data file input include eliminating the need to edit RHABSIM hydraulic data files and use of a new date format.

VII.B.1.Hydraulic data (depth and velocity lookup table)

The hydraulic data input file is generated in the RHABSIM river habitat simulation software. Multiple RHABSIM output files are typically required to defined the depth and velocity lookup data, because RHABSIM typically is calibrated differently for different ranges of flow. Version 1 of the CIFSS trout model required the user to import the RHABSIM files into a spreadsheet to combine them, then save the spreadsheet in a specific format. In this version, multiple RHABSIM files are read by the trout model software.

The software assumes that each RHABSIM file covers a non-overlapping range of stream flows. The first file (`hydraulicsFile1`) covers the lowest flows, and each succeeding file has the next higher range of flows.

The RHABSIM files are used without change. They are generated in RHABSIM using the "Velocity Output File" facility.

VII.B.2.Time series habitat data

Time series data are now input using the MM/DD/YYYY format for dates. Instead of two columns for year and Julian date, there is now one column for the date, followed by another column with the input value (flow, temperature, etc.). The header lines can be up to 200 characters long.

The following is part of a new daily flow input file.

```
Flow (CMS) at Little Jones Lower Site
Estimated from JED Gage  Provided by JW
  Year              Flow
 1/1/1998              0.90
 1/2/1998              1.78
 1/3/1998              2.24
 1/4/1998              4.19
```

```
 1/5/1998              2.62
 1/6/1998              2.06
 1/7/1998              2.23
 1/8/1998              2.15
 1/9/1998              1.81
1/10/1998              1.69
1/11/1998              3.28
1/12/1998              5.29
```

There is also now a daily input file for turbidity as well as flow and temperature.

VII.B.3.Barrier data

Version 2 of the CIFSS trout model includes simulation of barriers to upstream movement. The model formulation assumes fish can move downstream over barriers but cannot move upstream over them. The locations of barriers are depicted only by their distance upstream of the downstream end of the modeled reach, in meters. Multiplying this location by 100 converts it into an X coordinate (cm) in the model's internal coordinate system.

Barriers are input to the model by simply including their location in the barrier input data file. The name of this file is provided in the Model.Setup file (Sect. VII.A.2), and typically is *Barrier.Data, where "*" is the site name. This file must exist even if there are no barriers in the reach being modeled.

The barrier data file includes:

- Three header lines that are ignored by the computer.
- One additional line for each barrier. The line contains only one value, the barrier's location measured as the distance (m) between the barrier and the downstream end of the reach. If there are no barriers, there are none of these lines.

The following example barrier data file includes three barriers to upstream movement.

```
Barrier data for Little Jones Creek, tributary site
File made up by SKJ, 3/7/00
Barrier X, in meters
60.9
84.9
147.5
```

VIII. Software Installation and Execution

Version 2 includes only a few minor changes in how the code is executed.

First, if the Experiment Swarm is used to generate multiple model runs in graphics mode, the first model runs starts as previously: the `Start` button on the main control panel is hit twice, then a subswarm control panel opens to control the actual model run. After each model run finishes, control reverts to the main control panel. The user must hit `Start` again to initiate the next model run.

Second, the software now operates in batch mode if selected. Batch mode conducts the model simulations without the graphical user interfaces, making execution faster and eliminating the need to manually start each simulation from the control panel. This mode is especially useful when conducting multiple long simulations.

Batch mode is selected by starting the model with the argument "-b" (or "--batch"):

```
./Cutthroat.exe -b
```

or:

```
./ Cutthroat.exe --batch
```

The code prints a message notifying the user when the simulation has been completed and output files written.

IX. Output Files and Output Processing

File output has been enhanced in Version 2 to allow analysis of multiple scenarios and replicates and to facilitate analysis of habitat selection by fish. The section describes the seven output files created by all model runs, and an additional file that can be switched on when needed.

Each of these files is updated with a frequency determined by the model setup parameter `fileOutputFrequency` (Sect. VII.A.2). The model setup parameter `appendFiles` determines whether they are overwritten each model run or appended.

All output files provide information on the model status (current scenario number, replicate number, date, and sometimes flow and temperature) at the time each output line is written.

IX.A. Fish Output File

This file provides a summary of the fish population status. It provides the abundance (number of live fish) and mean length of fish. Results are broken out by species and age class.

IX.B. Fish Mortality File

The mortality file provides the cumulative number of fish that have died, by species and age class. On each output date, the file reports the total number of fish that have died, since the start of the model run, of each mortality source.

IX.C. Redd Output File

This new output file provides summary data on redds. It was designed to allow basic redd characteristics to be easily imported into statistical software and analyzed.

The file name is specified via the model setup parameter `reddOutputFile`. The file includes one line of output for each redd created. The line includes the redd's location (transect and cell number), the date the redd was created, the initial number of eggs in the redd, the date the redd became empty, the number of eggs that died due to each mortality source, and the number of fry emerging from the redd.

IX.D. Redd Mortality File

This file provides a separate mortality report for each redd. The report starts with header information reporting the redd location, creation date, and initial number of eggs. For each date of the redd's existence, the file reports the number of eggs dying of each redd mortality source.

This file is created at the end of a model run, and will not be written if the model is stopped before reaching its end date.

Output of this file can be prevented, if desired, by commenting out the statement

```
#define REDD_REPORT
```

that is near the top of the file `TroutModelSwarm.h`, then re-compiling the code.

IX.E. Depth and Velocity Availability Files

These files describe the availability of habitat as defined by cell depth and velocity, in a histogram-like format. The depth and velocity availability and use files (below) are provided so that habitat selection measures can be evaluated. For example, these outputs can be used to develop the equivalent of the Habitat Suitability Criteria used in PHABSIM.

The model setup variables `depthBinWidth`, `velocityBinWidth`, `depthMaxBin`, and `velocityMaxBin` (Sect. VII.A.2) define the histogram bins. The output files provide the river surface area in each depth and velocity bin. Any area with depth (velocity) greater than the value of `depthMaxBin` (`velocityMaxBin`) is provided in a last bin. Output is labeled by bin.

Bins refer to depths up to and including the depth used to label the bin. For example, if the value of `velocityBinWidth` is 10 cm/s and the value of `velocityMaxBin` is 100 cm/s, then the software creates 11 bins. The first is labeled 10 and provides the stream area with velocity between zero and 10 cm/s. The second bin is labeled 20 and provides the stream area with velocity greater than 10 and less than or equal to 20 cm/s. The last bin (labeled ">100") provides the stream area with velocity greater than 100 cm/s.

IX.F. Depth and Velocity Use Files

These files describe the depth and velocity of habitat actually used by fish. They use the same histogram format and bins as the habitat availability output files. Instead of providing the stream area in each bin, these files provide the number of fish using habitat of each depth or velocity range. Separate values are provided for each species and age class of fish.

IX.G. Cell-based Fish Information

A new optional output file provides information on how many fish of each age class use each habitat cell, along with habitat data for the cell. This file was designed for experiments that examine habitat selection.

This file is turned on by including this line in the code file `HabitatSpace.h`:

```
#define CELL_FISH_INFO
```

(Normally, this line is present in `HabitatSpace.h` but de-activated by turning it into a comment:

```
// #define CELL_FISH_INFO
```

.) The code must be re-compiled after making this change.

The cell-based fish output file is always named `CellFishInfo.Out`. It includes one line for each cell, for each output date. Each line includes the date; transect and cell numbers; the cell's

area, depth, velocity, distance to hiding cover, and fraction with velocity shelters (all with distance units of cm; cell area is in cm^2); and the number of fish in the cell, for each age class.

X. Software Testing

One new software testing facility has been added: a set of debugging and testing outputs that can easily be turned on and off. These were designed to demonstrate that variables controlled by the Experiment Swarm (Sect. VI.C) have the correct value during simulations. These debugging outputs are controlled via a code file named `DEBUGFLAGS.h`.

First, the user edits `DEBUGFLAGS.h` to select which debug outputs are desired. This file contains a number of `#define` statements that each turn on a set of debugging print statements. Comments in the file describe what objects and actions each debug output addresses. These `#define` statements are normally commented out (preceded by "//", which tells the compiler to ignore the rest of the line). By removing the "//" comment flag, a `#define` statement is activated.

After editing `DEBUGFLAGS.h` the code must be re-compiled (by typing "Make" in the Swarm terminal window). The model can then be executed and the debug print statements will write to the terminal window. The debug output can be captured to a file by redirecting the model's standard output: instead of starting the model by typing

```
./trout.exe
```

start it by typing

```
./trout.exe > debug.out
```

to capture the debug output in a file called (for example) `debug.out`.

The debug output can be voluminous if more than a few fish are simulated. We recommend starting the model with very few fish if debug output is activated.

Following is the `DEBUGFLAGS.h` file.

```
//
// The following define flags used
// in various places
// This creates a lot of output
// if there are many fish

//
//#define DEBUG_TROUT_FISHPARAMS
//
// The following define flags are used
// in Trout.m
//
//#define DEBUG_SPAWN
//#define DEBUG_MOVE
//#define DEBUG_GROW
//#define DEBUG_FEEDING

//
// The following define flags used
// in Redd.m
```

```
//
//#define DEBUG_REDD
//#define DEBUG_REDD_SCOUR
//#define DEBUG_REDD_DEWATER
//#define DEBUG_REDD_LOTEMP
//#define DEBUG_REDD_HITEMP

//
// The following define the debug flags used
// each of the Survival Probability objects
//
// These flags must be defined when checking
// the values used when the survival probabilities
// create their logistic functions
//

//#define DEBUG_HT_FISHPARAMS
//#define DEBUG_AQUATICPRED_FISHPARAMS
//#define DEBUG_POORCOND_FISHPARAMS
//
// The next one prints each time a fish accesses
// spawning surv prob  see SpawningSP.m
//
//#define DEBUG_SPAWNING_FISHPARAMS
//#define DEBUG_STRANDING_FISHPARAMS
//#define DEBUG_TERRPRED_FISHPARAMS
//#define DEBUG_VELOCITY_FISHPARAMS
```

[Software testing issues and methods discussed in this section are also discussed by Ropella et al. (in prep.).]

XI. Conducting Modeling Experiments

(No changes.)

XII. Source Code Description and Directory

This section provides an updated directory of the trout model's code files and methods.

XII.A. Code Directory

XII.A.1.ExperSwarm

This file contains code for two objects: ParameterManager and ExperSwarm. These objects provide the Experiment Swarm that creates and starts individual model runs with different parameter values. For each model run, a new copy of the Trout Observer and Model swarms (referred to in the code as a subswarm) is created, given values for the parameters being varied, and executed.

Method	Function
`initializeParameters (ParameterManager)`	Reads the Experiment.Setup file and creates a list of parameters to be varied and determines their parameter type. Identifies the number of replicates for each scenario. Creates a ScenarioIterator object and gives it the list of parameters and number of replicates.
`initializeModelFor (ParameterManager)`	Tells a new subswarm what iteration it is, calls the ScenarioIterator's `nextControlSetOnObject` method.
`buildObjects (ExperSwarm)`	Creates and initializes the ParameterManager. Creates the subswarm control panel.
`buildActions`	Contains the Experiment Swarm schedule: set up model, build model, run model, check to stop, drop model.
`setupModel`	Creates a new subswarm (TroutObserverSwarm) and tells the parameter manager to initialize it. Indexes through all the subswarm's objects and finds objects belonging to the class for which variables are to be altered (defined in `Experiment.Setup`); for each such object, calls `initializeModelFor` to give the object its new parameter value.
`buildModel`	Tells the initialized subswarm to execute its buildObjects method.
`runModel`	Executes the subswarm and returns control to the Experiment Swarm when the model run is finished.
`dropModel`	Closes the subswarm and drops it (removes it from memory).

checkToStop	Closes the Experiment Swarm when all model runs are done.

XII.A.2.ScenarioIterator

This class provides an object used by the Experiment Swarm to control model runs.

Method	Function
appendToIterSet-Param	Creates lists of the parameters to be varied for each class of model object, along with the type of the parameters (floating, integer, boolean, file name, etc.) and the parameter values for each scenario. These lists are contained in a *Swarm* map collection.
canWeGoAgain	Keeps track of how many replicates have been completed for each scenario, and how many scenarios have been completed. Determines whether more model runs are needed.
nextControlSet-OnObject	Changes the parameter values of an object for each model run. Reads the map of parameters and values, opens a probe to the object, and gives it the correct parameter value.
sendScenario-CountToParam sendReplicate-CountToParam	Uses a probe to send the current scenario and replicate counts to the parameter and class specified by the user in the Experiment Swarm setup file. These are used to tell the current model swarm which scenario and replicate it is so these values can be included in output files.

XII.A.3.TroutObserverSwarm

The TroutObserverSwarm objects have two main functions. First, they set up all the observeration tools: animation, graphs, and probes. Second, they start and stop the simulations. Methods in the Observer Swarm generally run once when a model run is started, or execute at the end of each time step to display graphical interfaces.

Method	Function
update	Deletes and re-draws the animation raster and graphs each time step. This method needs to be edited when graphs and rasters are added or dropped.
buildActions	Contains the display schedule. Implements the display frequency parameter. Includes code that determines whether to write raster pictures to file.

`buildFishProbes`	Sets up the probe map for trout (fish) objects. Edit this to change the variables and methods in fish probe displays.
`buildObjects`	Builds the color map for habitat cells. Creates the animation raster window and sets its parameters. Sets up the mouse buttons to probe cells and fish. Builds the graphs and histograms that are displayed during execution. Edit this method to add new graphs or modify existing ones.
`buildProbesIn`	Sets up the probe maps that open when a model run starts for such classes as: TroutModelSwarm, HabitatSpace, Cell, and TroutRedd. Edit these to change the variables and methods contained in probe displays.
`checkToStop`	Calls the Model Swarm to find out if the model end date has been attained; if so, it stops the Observer activity schedule.
`createEnd`	Loads Observer Swarm parameter values from "Observer.Setup".
`drop`	Drops execution of all graphical interfaces and terminates Observer activity. Needs to be edited when new graphs etc. are added.
`objectSetup`	Creates the TroutModelSwarm, passes it needed Observer-related parameters (raster resolution, etc.), and loads Model Swarm parameters from the file Model.Setup. Executes the Model Swarm's `instantiateObjects` method so its objects exist for the Experiment Swarm to manipulate before the model run starts.
`writeFrame`	Writes raster window to file, to capture a movie of raster animation. Edit this to change the file names or change which window (or the entire screen) to capture.

XII.A.4.TroutModelSwarm

The TroutModelSwarm objects execute the actual simulation events. They create the habitat cells, fish, and redds, and execute their actions each time step.

The major change with Version 2 is adding the method `instantiateObjects`, which is necessary to make the Experiment Swarm work.

Method	Function
`_buildDataZone_`	Builds the data structures used to store parameters for each fish species. Loads fish parameters from the files named in

	Species.Setup.
`_buildFishClass_`	At run time, creates an Objective-C class for each species in the model.
`_buildInitialFish_`	Using data from the population initialization files, builds the starting population lists. Creates the "fish population map" data structure for fish.
`_buildReddDataStructure_`	Builds the "reddMap" data structures that hold redds. (The ability to simulate redds that exist at the start of simulations has not been implemented.)
`_readPopFiles_`	Reads the fish population initialization input files.
`addAFish`	Puts a fish into the "fish population map" data structure.
`buildActions`	Creates the model's action schedules. Determines the order in which update, fish, redd, and overhead actions execute.
`buildObjects`	Calls the methods that create all the model objects, including observer graphics, parameter storage structures, habitat cells, fish classes, and initial fish. Creates the symbols used in the model. Selects the *Swarm* random number generator to be used. Initializes variables that count the number of fish mortalities by cause.
`buildTotalFishPopList`	Creates a list of live fish from the species-specific lists, sorted by dominance. The model executes trout actions on this list.
`createAgeMaps`	Creates the maps on which fish are listed by age class.
`createANewFishFrom`	Called by redds to create new fish. Gives the new fish its length, species, and sex.
`createNewFishWithSpecies Index: Species: Age: Length`	Creates a new trout with the specified species, age, and length. The fish's weight is calculated so that its condition factor equals 1.0.
`getDeadTroutList`	Creates list of total dead fish from species lists.
`initialDayAction`	Can be used to call any methods that should execute only on the first day of a run. Currently used only to remove the "oneAction" method from the action schedule.

instantiateObjects	Calls the methods readSpeciesSetup, _buildDataZone_; creates the habitat space and loads its parameters. This method creates the model objects that can then be manipulated by the Experiment Swarm before the model objects (habitat cells and fish) are created. This allows the Experiment Swarm to control the parameters used to create cells and initial fish.
printFishMortalityFile	Prints out the number of fish dead, by their mortality source, to fishMortalityFile. This method is called from the "printSchedule" in the main model schedule, as defined in the buildActions method.
printFishPopSummaryFile	Prints out the number of fish alive in each age class, to fishOutputFile. This method is called from the "printSchedule" in the main model schedule, as defined in the buildActions method.
printFPMReport	If activated in the TroutModelSwarm.h file, prints out a list of all the fish that were initialized, to test the model initialization methods.
printReddReport	If activated in the TroutModelSwarm.h file, prints out mortality report (reddMortalityFile) for each redd created in a model run.
printReddSurvReport	If activated in the TroutModelSwarm.h file, prints out survival report (reddSurvFile) for each redd created in a model run. This report is for testing survival functions.
processAgeClassLists	Re-builds the age class-specific lists of fish each day. All fish on the total population list are added to their proper age class list (fish remain on both lists).
processEmptyReddList	Moves the redds that are emptied each day (each species' "empty redd list") to the permanent list of empty redds.
processKillLists	Moves the fish that died each day (which are temporarily placed on each species' "killed list") to the permanent list of dead fish.
readSpeciesSetup	Reads the file Species.Setup, which tells the model the name of each species and which files contain its parameters and initialization data. This file also sets the raster color for each species.

setFishColorMap	Links colors provided as input to fish species. Defines the color of tagged fish.
updateCauseofDeath	Uses the killed list each day to increment the counters for how many fish died of each mortality source.
updateFish	Called by the update action schedule; increments the fishs' age on January 1.
whenToStop	Called by the Observer Swarm to determine whether it is time to stop a simulation. Compares current date to the model end date provided as input. Calls any methods that need to execute at the end of a model run; these include some of the report-writing methods.

XII.A.5.HabitatSpace

The HabitatSpace object for a simulation contains and manages all the habitat cells. In general, the HabitatSpace manages variables that depict habitat conditions that are independent of fish populations. These variables typically include the depth, velocity, spawning gravel, velocity shelter, and food availability in each cell. Habitat variables that do not vary among cells (e.g., temperature, flow rate) are generally contained in HabitatSpace, not in the individual cells.

Method	Function
_getCellContainingX: Y:	For any point, finds the cell containing it.
buildBarriersFrom:	Reads the barrier input file and creates barrier objects as specified by the file.
buildCells	Executes the methods for building cells and reading in habitat data. Serves as the schedule for setting up space.
buildFlow_DepthTables	Builds the flow vs. depth lookup table for each cell. Arrays of data are passed to it from the method readVelocityAndDepth. Builds tables for all the cells on a transect at a time. Converts input depth in m to cm, converts table values to logarithms.
buildFlow_VelocityTables	Builds the flow vs. velocity lookup table for each cell. Arrays of data are passed to it from the method readVelocityAndDepth. Builds tables for all the cells on a transect at a time. Converts input velocity in m/s to cm/s,

	converts table values to logarithms.
`calcDayLength`	Calculates the day length (h) from the Julian date.
`calcNumAge2PlusFishPerCM`	Calculates the density of predatory trout, evaluated as the number of live trout of age 2 or older divided by model reach length in cm.
`calcSpaceParameters`	Calculates the boundaries and midpoints of cells.
`createMortalitySymbols`	Builds a symbol for each fish mortality source.
`getAdjacentCells:`	Finds the four cells that are on each side of the specified cell. This list does NOT include the starting cell itself.

`getLogFlow:`
 	
`getTomorrowsLogFlow`
	Gets the proper value from the flow record for the date passed to the method. Tomorrow's log flow is used in the scouring mortality method for redds.

`getNeighborsWithin: of:`	Finds cells that are within a specified distance of a cell. (Distances are measured between the midpoint of cells.) This list does NOT include the starting cell itself.
`isThereABarrierTo:`	Given two cells, determines whether there is a barrier to upstream movement between them. Returns values of 0 if the second cell is upstream of the first and there is a barrier between them. Returns 1 if the second cell is downstream of the first and there is a barrier between them. Returns -1 if there is no barrier between the cells.
`nextTemperature`	Reads the daily temperature from the temperature record.
`printCellDepthReport`	If activated in the `HabitatSpace.h` file, creates a report of the daily flow and depth in each cell. This report is for testing the habitat methods.
`printCellVelocityReport`	If activated in the `HabitatSpace.h` file, creates a report of the daily flow and velocity in each cell. This report is for testing the habitat methods.
`printHabitatReport`	If activated in the `HabitatSpace.h` file, creates a testing report of the daily flow and temperature.
`probeCellAtX: Y:`	Determines which cell was clicked on and opens probe display to it. (Does not control contents of probe display.)

`probeFishAtX: Y:`	Determines which cell was clicked on and opens probe display to all the fish and redds in it. (Does not control contents of probe display.)

`readFlowRecord`
 	
`readTempRecord`
 	
`readTurbidityRecord`
	Opens and reads the files of daily flow, temperature, and turbidity values. The flow records are converted to logarithms and stored in an array. Temperature and turbidity are stored without conversion.

`readGeometry`	Reads the cell data input file. Creates the cells from the coordinates in the input file. Assigns cell habitat variables from the file- these are habitat characteristics that vary among cells but do not vary over time.
`readHydraulicInputFile`	Calls the object "FileInput" (below) that reads the depth and velocity lookup table input file.
`setCellHydraulics`	For each cell, creates a "stationObject" (see StationObject, below) that is a lookup table of depth and velocity vs. flow.
`setSpaceDimensions`	Opens cell data file, calculates the number of transects and maximum number of cells per transect.
`updateCellHydraulics`	Tells each cell to update its depth and velocity for the new daily flow rate. Determines the interpolation parameters the cells use to get the depth and velocity in their lookup tables, and passes these parameters to the cell method `calcDepthAndVelocityAtStationOffset`.
`updateFlowChange`	Each day, calculates the absolute value of the difference between today's and yesterday's flow rate (variable `flowChange`).
`updateHabitat`	Is passed a date. Calls the methods that get the flow and temperature and day length. Sends messages updating each of the cells' survival probabilities and food availability variables, and resets the cells' food and shelter consumption variables.

XII.A.6.Cell

Cell objects represent the individual rectangular pieces of habitat. In addition to knowing its habitat variables (temperature, depth, velocity, food concentrations, etc.), cells also keep lists of the fish that are in them.

In some models, only part of each cell has some specific habitat quality (e.g., spawning gravel or velocity shelter). Cell objects keep track of the total area of such habitat and how much of it has been occupied by fish or redds within the cell.

Cells contain a number of methods that merely pass requests for information up to the HabitatSpace object. For example, a fish may ask its cell for a list of neighboring cells; the cell merely calls the HabitatSpace method `getNeighborsWithin:` and passes the result back to the fish. These "pass-through" methods are not listed here.

Method	Function
`addFish`	Adds a fish to the list of fish in the cell, and removes the fish from its previous cell.
`addRedd`	Adds a new redd to list of those in the cell.
`calcCellShelterArea`	Calculates the cell's area with velocity shelter, once the cell's area has been established and variable cellFracShelter has been set. (Does NOT return the shelter area.)
`calcDepthAndVelocityAtStation-Offset`	To update the cell's depth and velocity, this method is passed two values by HabitatSpace. These values are used with the cell's lookup table to determine the daily depth and velocity.
`calcDriftHourlyTotal`	Calculates the total amount (g) of drift food available each hour, from `habDriftConc`, `habDriftRegenDist`, and cell size and velocity.
`calcSearchHourlyTotal`	Calculates the total amount (g) of search (benthic) food available each hour, from `habSearchProd` and cell size.

`containsX`
 	
`containsY`
	Returns "yes" or "no" if the cell's extent in the X (or Y) dimension includes the specified X (Y) value.

`createBegin`	Starts creation of a new cell by creating a list of pointers to cell variables and initializing the variables.

createEnd	Completes creation by setting the random number generator, creating the lists of fish and redds in the cell, and executing initializeSurvProbs.
drawSelfOn	Draws cell on raster, color coded by selected variable, and draws the cell's fish and redds. Includes code that determines how cell colors vary with the colorVariable.
eatHere:	Used to move a fish into the cell during the fish's "Move" method. Decreases the cell's available food and velocity shelter by the amount used by the fish, and calls addFish.
foodAvailAndConInCell	If activated in the Cell.h file, writes a report to file "FoodAvailability.rpt", for testing food production and availability calculations.
getArea getBenthicProd getCellNumber getCellVelocity getCellFracSpawn getCellDepth getDriftConc getDriftRegenDist getTransectNumber getMidPoint getXExtent getYExtent	Returns the value of cell-specific habitat variables. (XExtent is a cell's length in the X dimension; Yextent is a cell's width.)
getFishYouContain	Returns a list of fish in the cell.
getNeighborsWithin: aRange	Returns a list of all the cells within the specified range. "Range" refers to linear distance between cell midpoints. This method simply passes the range and the cell to the HabitatSpace method "getNeighborsWithin".
getNumberOfFish	Returns the number of fish in the cell, obtained from the cell's list of fish contained.

`getNumberOfRedds`	Returns the number of redds in the cell, obtained from the cell's list of fish contained..
`getReddsYouContain`	Returns a list of redds in the cell.
`getSpawnQualityForSpecies`	Calculates the spawning quality index for the cell and the species passed to it. Gets the suitability factors for depth and velocity from HabitatSpace and uses cell's area and spawning gravel fraction.
`initializeSurvProb`	Builds and initializes the various survival probability objects (representing the kinds of mortality risks). Edit this method and "getSurvivalProbFor" to add or delete survival probability functions.
`removeFish`	Deletes the specified fish from list of those in cell.
`removeRedd`	Removes a redd from list of those in the cell. (Does NOT remove redd from the redd population map.)
`resetAvailHourlyTotal`	Sets the available food variables to the cell totals (so food consumption by fish is zero) before the start of daily fish movement and feeding calculations.
`setBoundary`	Initializes the vector holding the cell's boundaries (the two X coordinates and two Y coordinates that define the cell's edges). Called when the cell is created. Sets the cell's area.
`setCellFlowVelocity`	Stores the flow-velocity lookup table in an array called cellFlowAndVelocity.
`setCellFlowAndDepth`	Stores the flow-depth lookup table in an array called cellFlowAndDepth.
`updateDSCellHourlyTotal`	Calls the methods `calcDriftHourlyTotal` and `calcSearchHourlyTotal` to update the cell's food supply variables each day.
`updateSurvivalProb`	Updates all the survival probabilities, by executing their method "update". Does not need editing if the number or type of survival probability functions changes.

XII.A.7.Trout

The Trout objects contain much of the individual-based model code. The trout methods include spawning, movement, feeding and growth, and mortality of each fish.

The Trout class is the superclass of all fish species in the model (see the "Species1" class below). When the Model Swarm executes fish methods, it looks for the methods in the species class. Following the object-oriented programming inheritance principal, if a method does not occur in the species class, the method is taken from the superclass, Trout. Therefore, the Trout class provides "default" methods for all fish. These methods are overwritten by any method with the same name in the species class code.

Note that the software uses species-specific parameter values for all fish methods, whether the method is coded in the Trout class or a species subclass.

Our design objective for the detailed fish methods was to put each of the formulation's important equations in its own method. This allows the major fish actions to appear in simple code that is easy to follow (see, for example, the methods `move` and `grow`) and that closely follow the formulation document. This also allows the important equations to be modified without editing more than one method, minimizing the potential for a change in one equation to affect another.

The following are important trout methods.

Method	Function
`_createAReddInCell_`	Creates a new redd when spawning occurs. Determines number of eggs and records species, spawner length.
`calcActivityRespirationAt: withSwimSpeed:`	Determines the daily activity (swimming) respiration energy, for the specified cell and fish swimming speed.
`calcCaptureArea`	Calculates the cross-sectional area in which the fish can capture drifting prey, from the reactive distance and cell depth.
`calcCmax`	Calculates Cmax, the maximum daily food consumption.
`calcCmaxTempFunction`	Calculates the temperature-dependence function for Cmax. This method does linear interpolation over the piecewise-linear function provided via input parameters.
`calcDailyDriftFoodIntake`	Determines the daily food intake for drift feeding at a cell. Makes sure daily food intake is not greater than Cmax or the drift food available in the cell.
`calcDailyDriftNetEnergy`	Determines the daily net energy intake for drift feeding at a cell, from the daily food intake. Conducts the energy balance

	between food intake and respiration.
`calcDailySearchFoodIntake`	Determines the daily food intake for active search (benthic) feeding at a cell. Makes sure daily food intake is not greater than Cmax or the benthic food available in the cell.
`calcDailySearchNetEnergy`	Determines the daily net energy intake for active search (benthic) feeding at a cell, from the food intake. Conducts the energy balance between food intake and respiration.
`calcDriftIntake`	Determines potential food intake using the stationary drift foraging strategy, from the capture area and cell velocity.
`calcFeedTime`	Calculates the feeding time (hours of feeding per day), from the cell's day length and temperature.
`calcMaxSwimSpeed`	Calculates the fish's maximimum swimming speed.
`calcNetEnergyForCell`	Provides the net energy intake for the best feeding strategy for the fish at the cell. Compares drift and active search intakes. Records the fish's chosen feeding strategy.
`calcReactDistance`	Calculates the drift-feeding reactive distance from the cell's temperature and velocity.
`calcReactDistance:`	Returns the reactive distance over which a drift-feeding trout can capture prey in the specified cell.
`calcSearchIntake`	Calculates potential food intake using the active searching (benthic) strategy.
`calcStandardRespiration:`	Determines the daily standard respiration energy consumption at the specified cell.
`calcTotalRespirationAt: withSwimSpeed`	Calculates the total daily respiration energy use for a fish, obtaining input from calcStandardRespiration and calcActivityRespirationAt.
`compare`	Compares current fish to another by dominance. This method is used by the QSORT procedure that sorts the fish list by dominance at the end of each time step. This method MUST BE PRESENT for the model to work even though it is not called by the CIFSS code.
`createEnd`	Assigns the random number distributions for spawning and mortality, and initializes food consumption variables.

`die`	Determines whether a fish dies each day, calling the survival probability objects for the fish's cell.
`drawSelfOn`	Includes code for how fish is drawn on the animation raster.
`expectedMaturityAt:`	Returns the "Expected Maturity" variable for a cell, used to rate the desirability of movement destinations. Calls other methods to obtain the fish's net energy intake and mortality risks at the cell.
`findCellForNewRedd`	Creates a list of neighboring cells and finds the best one for spawning. Calls `getSpawnQuality` to evaluate cells.
`getConditionForWeight: andLength:`	Returns the condition factor for a fish with the specified length and weight.

`getDailyDriftCellTotalAt:`
 	
`getDailySearchCellTotalAt:`
	These methods return the total daily food available from the cell; they multiply the available hourly food production in the cell by the fish's feeding time.

`getFishDominanceForLength:`	Returns the dominance a fish would have at the specified length.
`getFishShelterArea`	Calculates and returns the area of velocity shelter a fish uses up.
`getFracMatureForLength:`	Returns the fraction of sexually mature length a fish is.
`getIsShelterAvailable:`	Calls the specified cell to determine whether there is velocity shelter available for the fish.
`getLengthForNewWeight:`	Returns the new length of a fish after its weight has increased.
`getNonStarvSP:`	Calls the survival probability objects at the specified cell to get the total survival probability of all risks except starvation; used by `expectedMaturityAt:`.

`getSpawnDepthSuitFor:`
 	
`getSpawnVelSuitFor:`
	Returns the depth (velocity) suitability factor for spawning; includes interpolation code to get factor from the suitability parameters.

`getSpawnQuality`	Determines the quality of a cell for spawning, by calling the spawning suitability methods.
`getSwimSpeedAt:`	Returns the swim speed a fish has at the specified cell and

forStrategy:	feeding strategy (drift vs. search feeding). Calls getIsShelterAvailable to determine whether drift-feeding fish have velocity shelter available.
getWeightWithIntake:	Determines the fish's new weight from the specified intake in joules of energy.
grow	Determines the fish's new weight, length, and condition, using the food intake determined by the move method.
killFish	Used via the trout probe to manually remove a fish from the model.
move	Executes movement methods and moves fish. Calls the selected "moveToMaximize…" methods to select the destination cell. Calls methods to determine list of destinations, pick best destination, and move.
moveToMaximizeExpected-Maturity moveToMaximizeNetEnergy-Intake moveToMaximizeSurvival	The movement rules that determine where a fish moves are coded here in these alternative methods. (moveTo-MaximizeExpectedMaturity is currently hardwired as the movement method, but code is in place to let the method be selected at run time via the Model.Setup file.) Other internal methods are used to provide information on potential growth and risks at movement destinations. Moving also includes determining the feeding strategy and net energy intake at the destination cell.
moveToBestDest	Moves the fish to the cell selected by the moveTo… method. This method also updates a number of instance variables that depend on a fish's new cell and food intake there.
moveReport	If activated in Trout.h, reports the habitat, food availability, and intake conditions at the destination cell for each fish's daily movement. For use in testing movement code. Writes file MoveTest.rpt.
printFishParams	Prints out a list of all the fish's parameter names and values.
printSpawnReport	If activated in Trout.h, reports the habitat conditions and spawning quality variables at cells evaluated as spawning destinations. For use in testing spawning code. Writes file SpawnTest.rpt.
printSurvivalReport	If activated in Trout.h, reports the habitat and fish conditions and survival probabilities at the current cell for

	each fish, each day. For use in testing survival code. Writes file SurvivalTest.rpt.
`readyToSpawn`	Includes all code to determine whether a fish should spawn on a given day, according to the criteria for day of spawning.
`setFishDominance`	Updates the dominance instance variable with the fish's current length.
`setFishWeightFromLength: andCondition:`	Sets the weight instance variable from the specified length and condition factor, using the fish weight parameters.
`setInitialSwimSpeed`	Initializes the fish's maximum swim speed instance variable, for use in movement calculations on first day.
`spawn`	Calls internal methods to determine if a fish is ready to spawn, find cell for new redd, moves fish to cell, and create a redd. Records the date of spawning. Serves as schedule of spawning methods.
`tagFish`	Changes the fish's color to the tagged fish color; for use from the probe display.

XII.A.8.Species1

A CIFSS model includes a class for each fish species being modeled; these species classes are subclasses of the Trout class. At a minimum, this can be an empty class with no methods, in which case all fish methods are taken from the Trout class. Any methods that are in the species classes overwrite those in the Trout class. (Methods could also occur in the species class instead of in the Trout class.)

The Species classes should be used when one species uses different methods (equations) than other species. (Remember that species-specific parameters are used for all methods, whether they occur in the Trout or Species classes.)

The name of the species classes MUST exactly match the species name provided in the file Species.Setup, or else the code will crash when started (Sect. **Error! Reference source not found.**).

XII.A.9.SurvivalProb

The SurvivalProb class is the superclass for all the fish and redd survival functions (referred to as "survival probability functions" for trout and "survival functions" for redds, because redd functions return the fraction of eggs that survive). One such function is created for each habitat

cell, for each of the survival functions included in the code. (Each such function must have its own `.h` and `.m` files, which are not all described here.) These survival functions have three important methods.

- `createEnd` executes when the survival function is created. It creates the logistic functions used to calculate survival and initializes their parameters.

- `update` is executed once per day and updates any of the logistic functions that depend only on cell conditions and not on fish variables. This update reduces the number of computations required each time a fish asks the cell to provide survival probabilities. Redd survival functions do not include an update method because these functions are called far fewer times than are fish survival probability functions.

- `getSPFor:` returns the survival probability for the specified fish. (For redd survival functions, this method is called `getSFFor:` and returns the fraction of eggs that survive.)

XII.A.10.Redd

Fish redds are defined by the class `Redd`, which provides methods used by all redds. (We have not implemented species-specific redd methods.)

The following redd methods are included in our trout instream flow model.

Method	Function
`createEnd`	Initializes redd variables (age, formation date). Initializes counters for the number of eggs lost to each mortality source.
`drawSelfOn`	Provides the code used to draw redds on the animation raster window.
`survive`	Implements redd risks. Calculates the number of eggs lost each day to the various risks (low temperature, high temperature, scouring, dewatering, superimposition) and the number remaining in the redd.
`develop`	Updates the fractional development of the redd's eggs; includes code for how development is controlled by temperature.
`emerge`	Determines the number of new fish created from the redd each day. For each such new fish, the number of remaining eggs is decremented and a call is made to `turnMySelfIntoAFish`.
`removeWhenEmpty`	Drops the redd when all eggs are dead or emerged.
`turnMySelfIntoAFish`	Calculates the new fish's length from the spawner length (which is remembered by the redd object). Calls the Model Swarm method

	`createNewFishWithSpeciesIndex:` to create the new fish.
`printReport`	Prints the `reddMortalityFile` output.
`printReddSurvReport`	If activated in `Redd.h`, prints a report with survival function results and habitat conditions. For testing redd survival code.

XII.A.11.Barrier

Barrier objects are new in Version 2. The barriers themselves do nothing in the model; they affect simulations by affecting trout movement. The barrier objects therefore have only a few methods to record their location (an X-dimension value) and to draw themselves on the animation window.

XII.A.12.FileInput

FileInput is an object to read the hydraulic (depth and velocity lookup table) input files. These input files are created by the RHABSIM software package.

Method	Function
`create`	Builds arrays that temporarily store lookup table data.
`readFile`	Contains the code the reads the RHABSIM files. Creates Station objects (see `StationObject`, below) for each station in the RHABSIM file, which correspond to cells in the trout model. Each station object has a depth and velocity vs. flow lookup table (array), which is filled in by this method. This method is called once for each RHABSIM input file; there can be up to five such files (VII.B.1.

XII.A.13.StationObject

This class provides lookup table arrays of depth and velocity vs. flow for each cell. In addition to the following methods, there are a number of "set" and "get" methods that keep track of the station's transect, cell number, bottom elevation, etc.

Method	Function
`createBegin`	Builds the arrays that store lookup table data.
`getVelocityAtOffset`	Returns the velocity (depth) table value at the specified offset. (Offset

getDepthAtOffset	is the position in the array; offset = 0 is the first value in the table.)
addAFlow	When building the lookup table, adds a flow value at the next empty position in the array. Converts the flow to base-10 logarithm upon storing it. Sets logFlow to -4 if it is zero.
addADepth	When building the lookup table, adds a depth value at the next empty position in the depth column of the array. Converts depth from meters to cm. Converts the depth to base-10 logarithm upon storing it. Sets logDepth to -1 if it is zero.
addAVelocity	When building the lookup table, adds a velocity value at the next empty position in the velocity column of the array. Converts velocity from m/s to cm/s and converts it to base-10 logarithm upon storing it. Sets logVelocity to -1 if it is zero.
checkMaxOffsets	Makes sure there are the same number of flow, depth, and velocity values in the lookup table array.

XII.A.14.TimeWrapper

The TimeWrapper class provides a generic date-handling object. It uses the C time_t time and date format (number of seconds since 1/1/1970). The methods provide useful date manipulations used throughout the trout model.

Method	Function
getTimeTForDate	Converts a MM/DD/YYYY character string to a time_t formatted date.
getYearForTimeT	Returns the year (four-digit integer) for a time_t formatted date.
getJulianDayForTimeT	Returns the Julian date (day of the year, 1-366) for a time_t formatted date.
getJulianDayForDay	Returns the Julian date for a date in MM/DD character format.
getYearForTimeT	Returns the year (four-digit integer) for a time_t formatted date.
getDateForTimeT	Returns a MM/DD/YYYY string for a time_t formatted date.
getDateForTimeT: modifyingMyString:	Replaces the string passed to the method with a MM/DD/YYYY string for a time_t formatted date. Modifying an existing character string

	avoids having to create a new one.
`adjustTimeT`	Corrects a time_t formatted date so it is for 12:00 noon. Useful for overcoming small errors in the date value due to daylight savings time, roundoff, etc.
`getNumberOfDays-Between: aTime and: aLaterTime`	Returns the number of days (integer) between the two specified dates. Value is negative if the second date is before the first.
`isThisTime: onThisDay:`	Accepts a time_t date and a MM/DD formatted day (character string) and returns YES if the date falls on the specified month and date.
`isTimeT: betweenMMDD: andMMDD:`	Accepts a time_t date and two MM/DD formatted days. Returns YES if the date is on a day falling between the two day values.

XII.A.15.Logistic

Logistic functions (sigmoid increases from 0 to 1, or decreases from 1 to 0) are typically used in a number of CIFSS methods. They are especially useful in modeling mortality risks. The Logistic object initializes logistic functions by converting the input used to define a logistic curve into the two parameters used to calculate the logistic function.

As an example, we model the effects of high temperature on daily survival probability as a logistic function of daily water temperature. We define this survival function with four numbers: for a rainbow trout, the survival probability due to high temperature (S_T) is 90% at a temperature of 22° and the survival probability is 10% at a temperature of 26°. The logistic equation is:

$$S_T = \frac{e^{(aT+b)}}{1+e^{(aT+b)}}.$$

The Logistic object calculates the two parameters *a* and *b* from the four input numbers, so this function can be evaluated efficiently as the model executes.

Evaluating the logistic equation causes a computer error when the exponent ($aT+b$ in the above example) is large, because the value of $e^{(aT+b)}$ is larger than the largest value a floating point number can be. (This happens when the exponent is above around 700.) However, the value of the logistic function approaches one to many decimal places as the exponent increases. To avoid the potential computer error we set the value of the logistic function to 1.0 if the exponent is greater than 80.

Method	Function
`create: withName: usingIndep: dep: indep: dep:`	This method is called to create a logistic object. It calls other internal methods to give the object a name and set its parameters *a* and *b* input values. Two pairs of independent (X) and dependent (Y) coordinates are provided to the method; the dependent variables must be between zero and one. In the trout model, the fish parameters `LOWER_LOGISTIC_DEPENDENT` and `UPPER_LOGISTIC_DEPENDENT` are used as the first and second dependent variables; these parameters have values of 0.1 and 0.9.
`evaluateFor:`	Returns the value of the logistic function for the provided independent variable.
`initializeWithIndep:`	Internal method that calculates the logistic parameters.

XIII. Research and Development Priorities

Version 2 of the CIFSS trout model software implements several of the development priorities identified in the previous version: batch mode and the Experiment Swarm. The other priorities identified in the previous User Guide remain to be addressed.

XIV. Quality Control Documentation

This section logs the tests we conducted to verify the computational accuracy of the trout model code. These tests used methods described in Sect. IX.G of the original User Guide. Extensive tests of the software against an independent implementation of key model components were conducted for the Version 2 software. Full documentation of systematic code tests is provided in the spreadsheet CODETESTV2.XLS. (This section does not include testing or calibration of the model's formulation, which is documented in the formulation report.)

Date	Tester	Test
6/6/99	SFR	Test hydraulic habitat simulations. Use cell probes to spot-check depth and velocity interpolation from flow. Units conversion error found and corrected.
7/29/99	SFR, SJ	Final review of source code for Redd.m, Cell.m, HabitatSpace.m, ObserverSwarm.m, Logistic.m
7/30/99	SFR, SJ	Final review of source code for Trout.m, Rainbow.m, Brown.m
8/2/99	SFR, SJ	Final review of source code for all fish and redd survival functions.
8/7/99	SFR, SJ	Final review of source code for TroutModelSwarm.m.
8/5/99	SJ	Run model with 6 species, one species. Successful test of Species.Setup and changeable number of species.
8/8/99	SFR	Test population intialization: see CodeTest.xls, page "PopInit". Verify that initial fish populations have correct number, mean and variance of length.
8/8/99	SFR	Test cell hydraulics: depth and velocity calculations: see CodeTest.xls, pages "Flow-Vel" and "Flow-Depth". Compare cell-specific graphs of velocity (depth) vs. flow are same in model output ("Cell_Flow_Velocity.rpt", "Cell_Depth_Velocity.rpt") as in hydraulic input file.
8/8/99	SFR	Test feeding and energetics methods: see CodeTest.xls, page "FeedTest". Compare intermediate and final results to those from code independently implemented in spreadsheet.
8/15/99	SFR	Test food availability accounting in cells: see CodeTest.xls, page "FoodAvail". Compare food availability, as it changes as more fish enter a cell, vs. independent spreadsheet implementation.
8/13/99	SFR	Test fish survival probability functions: CodeTest.xls, page "Survival". Compare survival probabilities vs. independent spreadsheet implementation.

8/15/99	SFR	Test redd survival functions: see CodeTest.xls, page “ReddSurvival”. Compare fraction of eggs surviving vs. independent spreadsheet implementation, over wide range of temperatures and hydraulics.
8/16/99	SFR	Test fish spawning criteria: see CodeTest.xls, page “SpawnCrit”. Compare intermediate results used to evaluate spawning criteria vs. independent spreadsheet implementation.
3/17/00	SFR, SJ	Complete review of new code. Revisions implemented 3/17. User guide updated.
4/3/00	SJ	Test of Experiment Swarm to ensure that parameter values provided in the Experiment setup file are correctly used within the model. Tests documented and documentation archived.
4/10 - 4/14/00	SFR, SJ	Extensive code tests vs. spreadsheet implementation; see file CodeTestV2.xls. Several bugs in code and parameter values found and corrected.
6/6 - 6/9/00	SFR, SJ	New output files tested to verify habitat area, number of fish vs. area.

XV. References

EPRI (1999). *Tools for individual-based stream fish models*. Electric Power Research Institute report TR-114006, Palo Alto, CA.

Railsback, S. F. and B. C. Harvey (in prep.). *Individual-based Model Formulation for Cutthroat Trout, Little Jones Creek, California*. U.S. Forest Service, Redwood Sciences Laboratory, Arcata CA.

Ropella, G. E. P., S. F. Railsback and S. K. Jackson (in prep.). Software engineering considerations for individual-based models. Manuscript submitted to *Natural Resource Modeling*.