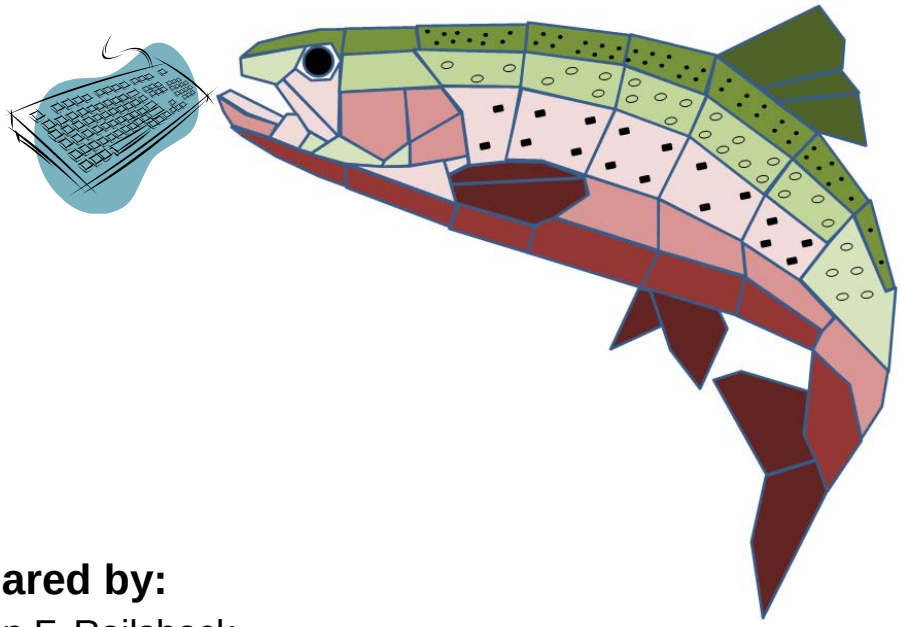
inSTREAM Version 5.0 Software Guide



Prepared by:

Steven F. Railsback
Lang, Railsback & Associates
Arcata, CA
www.LangRailsback.com

Colin Sheppard
Arcata, CA

Last changed: February 3, 2012

Contents

List of Figures ..... **v**

List of Tables ..... **vi**

1. Overview of the *inSTREAM* Software Help Document **1**

2. General Information on Input Files **1**
2.1. Common Characteristics of Input Files 1
2.2. Variable types 2
2.3. Files read by the Swarm object loader 2
2.1. Color variables 3
2.2. Translating files between Unix and Windows 3

3. Setup Files **3**
3.1. Observer Setup 3
3.2. Species Setup 5
3.3. Reach Setup 6
3.4. Model Setup 8
3.5. Experiment Setup 10

4. Parameter Files **11**
4.1. Habitat parameter file 11
4.2. Trout parameter file 11

5. Data Files **15**
5.1. Cell geometry 15
5.2. Cell data 16
5.3. Hydraulic data 16
5.4. Initial population data 18
5.5. Flow, temperature, and turbidity time series data 20

6. Changing Species and Size Classes **21**
6.1. Adding or Removing Species 21
6.2. Changing Age Classes 24

7. Graphical Interfaces **26**
7.1. Main Control Panel 26
7.2. Model Run Controller 27
7.3. Habitat Probe Displays 28
7.4. Graphs 29
7.5. Animation Window 29
7.5.1. Probe displays 30
7.5.2. Movies of the animation window 33

8. File Output **34**
8.1. General Information on Output Files 34
8.2. Output File Descriptions 35

9. Optional Output Files for Testing and Specialized Studies **35**

10. Experiment Manager **39**
10.1. What the Experiment Manager Does 39

10.2. General Procedure for Setting Up Experiments 39
10.2.1. Experiment.Setup format 40
10.2.2. Class and instance names for typical experiments 41
10.2.3. Valid parameter value types 42
10.2.4. Using instance names 43
10.2.5. Controlling where output goes 45
10.2.6. Checking the Experiment Manager 45
10.3. Example Experiment.Setup Files 46
10.3.1. No experiment 46
10.3.2. Replicate simulations 46
10.3.3. Parameter sweep for trout 47
10.3.4. Habitat parameter sweep 48
10.3.5. Multiple parameter sweep 48
10.3.6. Alternative daily input files 49
10.3.7. Multiple simulation years 50
10.3.8. Alternative cell data files 51
10.3.9. Year shuffler scenarios 51
10.3.10. Year shuffling as replication 52
11. Trouble-shooting Guide 53
Index 57

List of Figures

Figure 1. Example Observer.Setup file. 4

Figure 2. Example Species.Setup file. 5

Figure 3. Example Reach.Setup file. 7

Figure 4. Example Model.Setup file. 9

Figure 5. Example habitat parameter file. 11

Figure 6. Example trout parameter file. 14

Figure 7. Example cell geometry file. 15

Figure 8. Example cell data file. 16

Figure 9. Example hydraulic data file. 17

Figure 10. Example population initialization data file. This file defines initial populations for two reaches and three years. 19

Figure 11. Example file for flow, temperature, and turbidity input. 21

Figure 12. Main control panel. 26

Figure 13. Model Run Controller. 27

Figure 14. Habitat probe display. One of these windows is opened for each habitat reach. 28

Figure 15. Mortality output graph. “Time” on the X axis is the number of days simulated. The Y axis is the total number of trout that have died of each of the seven mortality sources. 29

Figure 16. Animation window. One such window is opened for each habitat reach, with the reach name used as the window’s title (top left). 30

Figure 17. Cell probe display. 31

Figure 18. Trout and redd probe displays. 32

Figure 19. Right-clicking on the species name box in a trout probe display opens a (empty) complete probe display (left). Clicking on the green button opens a complete probe display for the trout (right). 33

Figure 20. Example Experiment.Setup file. This setup causes 15 model runs. Three scenarios differ in giving simulated brown trout values of their parameter fishFitnessHorizon values of 60, 90, and 120. Each of these scenarios is run 5 times with different random numbers. 41

List of Tables

Table 1. Contents of Observer.Setup. 4

Table 2. Contents of the Reach.Setup file. 8

Table 3. Contents of the Model.Setup file.....10

Table 4. Optional output files.....36

Table 5. Class and instance names for parameters commonly used in the Experiment Manager. 42

Table 6. Value types for the Experiment.Setup file.....43

Table 7. Trouble-shooting guide.....54

1. Overview of the *inSTREAM* Software Help Document

This document provides user help on how to assemble input for the *inSTREAM* version 5.0 trout population model, how to set up and control model runs, and on the kinds of output the model produces.

This document does *not* describe the graphical user interface (GUI) that can optionally be used to create and modify input files and run the model. Documentation for the GUI is provided in a separate help file. Instead, this document explains the files that the model software itself reads as input and creates as output. The GUI can be used to modify the input files or users can edit them directly.

2. General Information on Input Files

The following three sections provide the detailed format for all the files needed to define an *inSTREAM* run. Setup files provide run-control information, parameter files provide equation coefficients for trout and habitat, and data files provide the numbers needed to define habitat cell characteristics, the initial trout populations, and the time series of flow, temperature, and turbidity that drive the model.

2.1. Common Characteristics of Input Files

All of the setup, parameter, and data input files are in ASCII. They can be maintained and edited using ASCII editors (e.g., Notepad, WordPad, gvim), or by using word processor or spreadsheet software and saving them as ASCII (plain text) or (as noted below) in .CSV format. Spreadsheets are especially useful for maintaining some input files.

Because *inSTREAM* runs in a Unix-like environment (even under Windows), it uses case sensitive file names and variable names. The model needs a file named “Model.Setup” and it will not find and use files named “model.setup” or “Model.setup”. Similarly, a variable named “fishParam” is different from one named “fishparam”. Failing to notice case differences in file and variable names is a common source of frustration.

File names should never include blank spaces.

None of the files require values to be in any particular columns: blanks in the file are ignored (with a few exceptions that are carefully noted). Values on the same line can be separated by one or more space or tab characters.

Many of the input files start with three header lines that are ignored by the software; these can be used to document the file type and where its information came from, and to provide column labels for the remaining lines. These header lines can be up to 200 characters long.

A new feature in version 5.0 of *inSTREAM* is the ability to read most data files (sections 5.2-5.5) in comma-separated variable (CSV) format. This format is widely used for input and output from spreadsheet, statistical, and other data-oriented programs. These files can be generated, for example, by saving Excel spreadsheets in CSV format. CSV files are in ASCII (“plain text”) with commas separating the variables in a file line.

inSTREAM does not completely implement the CSV format standard. According to the CSV standard, it is allowable (but not required) for any inputs, including numbers, to be inclosed in double quotes, but input for *inSTREAM* (except header information) must not be in quotes. Excel generally does not put numbers or text values in quotes when spreadsheets are saved in .CSV format, so usually this issue is unimportant for Excel users. The Calc spreadsheet of OpenOffice and LibreOffice does however enclose text values in quotes, which must then be stripped out using the "search and replace" function of a text editor.

2.2. Variable types

This document refers to variables as belonging to several common types:

- *Integers* are numbers with no decimal places and often describe how many there are of some thing (e.g., how many fish at the start of a simulation).
- *Floats* are numbers with decimal places (what mathematicians call "real numbers"). They can be written in decimal format (e.g., `0.074`) or scientific notation (`7.4E-2`).
- *Boolean* variables are true-false (or "yes-no"). Their value is `0` for false (no) or `1` for true (yes).
- *Text* are variables containing words or strings of alphanumeric characters. Text variables are case-dependent and cannot include blanks. They should not be enclosed in quotes.
- *Dates* are in MM/DD/YYYY format (e.g., `10/1/2012`)

2.3. Files read by the Swarm object loader

The setup files and parameter files are read by Swarm's "object loader" facility. The object loader is a simple tool for reading in variable values for an object. Except as noted below, all the setup and parameter files must be in the following object loader format.

- The first line has only the text `@begin` (unless the first lines are comments; see below).
- There is one line per variable, with each line containing the variable name followed by its value.
- Variables need not be in any particular order.
- The variable must be spelled exactly as it is in the code, including upper/lower case.
- For text variables, the value text is not inclosed in quotation marks.
- Variables containing text must not have trailing blanks after the text input. (The blank will be read as part of the variable's value, which makes the code unable to interpret the value. If the variable is a file name, the code will be unable to open the file because it will look for a file with a blank at the end of its name. This is another potential source of frustration.)

- Integer values should not have a decimal point.
- The last line has only the text `@end`.
- Comment lines (ignored by the computer) can be included in the file; they start with the character #. (In some cases, Swarm has been unable to distinguish comments so they had to be removed.)

Example object loader files are provided in the following sections.

2.1. Color variables

Several setup files specify color variables: the color used to display various objects. Color variables contain a text word that is the color name (e.g., `bluegreen`). Allowable values include all the common colors (red, green, blue, etc.) plus a large number of exotic color names. (Lists of Unix colors can be found on-line; search for "Unix colors".) Invalid color names result in white being used. Do not use color names that contain blanks: e.g., use `DarkViolet` instead of the equivalent `dark violet`.

2.2. Translating files between Unix and Windows

(Users who work only in Windows can ignore this section.) Transferring files between Windows and Unix-based (including Linux) operating systems can result in subtle problems because the two operating systems use different codes for the line ends in ASCII files. Hence, files created in Windows may not work in Linux; and files created in Linux may work in some, but not all, Windows programs. Attempting to run *inSTREAM* in Unix with files created (or edited) in Windows is likely to fail because of this problem (and Unix will not clearly indicate what the problem is).

Unix and Linux operating systems include programs "dos2unix" and "unix2dos" to convert between formats. For example, to convert a directory of files prepared in Windows for use on a Linux computer, in Linux type `dos2unix *` within the directory. Be aware, though, that dos2unix and unix2dos destroy any executable file they attempt to convert, so be careful using them on directories that contain `instream.exe`.

3. Setup Files

Five setup files provide run-time control information about graphical outputs, species, habitat reaches, automated experiments, and the model run itself.

3.1. Observer Setup

This file controls the observer swarm, which provides the graphical interfaces described in Section 7. It must be named "Observer.Setup". An example is at Figure 1.

```
@begin

rasterColorVariable         depth
takeRasterPictures          NO

rasterResolutionX           50
rasterResolutionY           50
maxShadeVelocity            200
maxShadeDepth               150

tagFishColor                 tomato
tagCellColor                 DeepPink1
dryCellColor                 white

@end
```

Figure 1. Example Observer.Setup file.

The variables in Observer.Setup are explained in Table 1.

Table 1. Contents of Observer.Setup.

Variable name and type	Definition
rasterColorVariable (text)	Selects the habitat variable used to color-code habitat cells on the animation window. Valid values are `depth` and `velocity`.
takeRasterPictures (text)	Determines whether the raster window is captured in files for post-processing into a movie of the simulation (see Section 7.5.2). Valid values are no and `yes` (also `NO` and `YES`). Raster pictures should not be turned on unnecessarily; they severely reduce execution speed and generate many large output files.
rasterResolutionX (integer) *rasterResolutionY* (integer)	Set the animation window resolution (cm per display pixel) in the X (east-west) and Y (north-south) dimensions. Reducing these variables makes the displays bigger, but requires more memory and more execution time to update the display.The values of *rasterResolutionX* and *rasterResolutionY* can be varied separately to exaggerate one dimension.

maxShadeVelocity (float) *maxShadeDepth* (float)	Control the animation window's shading by depth or velocity. These variables are the maximum velocity (depth) over which cells are shaded; velocities (depths) greater than these values are given the same color. For example, if *maxShadeVelocity* is 200, then cells will be shaded from yellow to red as velocity increases from 0 to 200 cm/s; but all cells with velocity > 200 will have the same, reddest, color. Increasing these variables makes it possible to distinguish among higher velocities (depths) but harder to distinguish among low values.
tagFishColor (text) *tagCellColor* (text)	Set the color for fish and cells that have been tagged via probes. (See Section 2.1 about color variables.)
dryCellColor (text)	Sets the display color of cells with zero depth.

3.2. Species Setup

The species setup file tells *inSTREAM* which species of trout are being simulated and where to find the input files for each. (This file does not, by itself, determine how many species are modeled; see Section 6.1.) The file must be called "Species.Setup". The contents of this file must match the code that defines species in the model, as explained in Section 6.1.

The species setup file does not use the Swarm object loader format. A Species.Setup example for a two-species model is at Figure 2.

```
Species.Setup file, rainbow and brown trout model.
For each species, provide species class name, parameter file name,
population initialization file name, and raster display color.

Rainbow
TurbidCrkRainbowTrout.Params
TurbidCrkRainbowTroutPopInitFile.csv
red

Brown
TurbidCrkBrownTrout.Params
TurbidCrkBrownTroutPopInitFile.csv
olivedrab
```

Figure 2. Example Species.Setup file.

The species setup file starts with a block of three comment lines that are ignored by the software, followed by a blank line. Then come a block of lines for each species in the model. Each such block includes:

- A line with the species name, which must exactly match the name of the source code class for the species. (If this file contains a species “Rainbow”, then the source code files Rainbow.h and Rainbow.m must have been compiled when `instream.exe` was created.)

- A line with the name of the fish parameter file that the species uses. (One parameter file can be used by more than one species.)

- A line with the name of the population initialization data file for the species.

- A line containing the name of the color used for the species in the animation window.

- A blank line, if another species follows.

3.3. Reach Setup

This setup file, which must be named “Reach.Setup”, specifies the number of habitat reaches, how reaches are linked, and what input files should be used for each reach. The file does not use the object loader format. The file contains:

- Three header lines that are ignored by the computer.

- A blank line

- A block of lines for each reach. These blocks start with a line containing only the word `REACHBEGIN` and end with a line containing only the word `REACHEND`. There can be multiple blank lines between these blocks.

```
Reach.Setup file.
Provide one block for each reach.
Example input.

REACHBEGIN
reachName                       WeejakTrib
habParamFile                    LJCHab.Params

habDownstreamJunctionNumber      2
habUpstreamJunctionNumber        3

cellGeomFile                    LJCWeejGeom.Data
cellHabVarsFile                 LJCWeejCell.Data
cellHydraulicFile               LJCWeejHydr.Data
flowFile                        WeejTestFlow.Data
temperatureFile                 LJCLowTemp.Data
turbidityFile                   LJCLowTurbidity.Data

REACHEND

REACHBEGIN
reachName                       LowerMainstem
habParamFile                    LJCHab.Params

habDownstreamJunctionNumber      4
habUpstreamJunctionNumber        2

cellGeomFile                    LowerMainstemGeom.Data
cellHabVarsFile                 LowerMainstemCell.Data
cellHydraulicFile               LowerMainstemHydr.Data
flowFile                        LowerMainstemFlow.Data
temperatureFile                 LJCLowTemp.Data
turbidityFile                   LJCLowTurbidity.Data

REACHEND
(etc. for remaining reaches)
```

Figure 3. Example Reach.Setup file.

The block of lines for each reach contains a separate line for each of the reach's setup variables. The line contains the variable name, one or more spaces, then the variable value. These lines need not be in any particular order within the block. An example Reach.Setup file is at Figure 3, with variables explained in Table 2.

Table 2. Contents of the Reach.Setup file.

Reach variable and type	Definition
reachName (text)	The user-defined name for a reach. (Up to 30 characters.)
habParamFile (text)	The name of the reach's habitat parameter file.
habDownstream-JunctionNumber (integer)	The junction number for the reach's downstream end. (Junction numbers are explained in the model description document.)
habUpstreamJunction-Number (integer)	The junction number for the reach's upstream end.
cellGeomFile (text)	The name of the reach's cell geometry file. File names can be up to 35 characters long.
cellHabVarsFile (text)	The name of the reach's cell habitat variables file.
cellHydraulicFile (text)	The name of the reach's hydraulic input file.
flowFile (text) *temperatureFile* *turbidityFile*	The names of the flow, temperature, and turbidity data files for the reach to use. (Different reaches can use the same files.)

3.4. Model Setup

The model setup file must be called "Model.Setup". It contains basic variables controlling a model run. An example is at Figure 4, and the file contents are explained in Table 3.

This figure and table do not include the optional variables that can be added to Model.Setup to control optional output files; these are explained in Section 9.

```
# Model Setup File for Example inSTREAM 5.0 Application
# Created 12/04/2011

@begin

randGenSeed                          32461

runStartDate                        10/1/1990
runEndDate                          9/30/2001

popInitDate                  10/1/1990

fishOutputFile               LiveFish.csv
fishMortalityFile            DeadFish.csv
reddOutputFile               Redds.out

fileOutputFrequency                     10
appendFiles                   0

siteLatitude                   42

shuffleYears                        0
shuffleYearReplace                  0
shuffleYearSeed                     737899

@end
```

Figure 4. Example Model.Setup file.

Table 3. Contents of the Model.Setup file.

Reach variable and type	Definition
randGenSeed (positive integer)	The seed value for the random number generator used for all stochastic processes except year shuffling. It can be any positive integer.
runStartDate *runEndDate* (date: mm/dd/yyyy)	The dates for which simulations begin and end.
popInitDate (date)	The date of the values in the population initialization input file (Section 5.4) used to create the initial trout population.
fishOutputFile (text)	The name of the output file for statistics on live fish.
fishMortalityFile (text)	The name of the output file for statistics on fish mortality.
reddOutputFile (text)	The name of the output file for statistics on redds.
fileOutputFrequency (integer)	The frequency with which file output is written. If set to 1, output is written for each simulated day; if set (for example) to 10, output is written only each 10th day. (This output is the model's state on the date when output is written, not an average for the period between output dates.)
appendFiles (boolean: 0 or 1 for "no" or "yes")	Whether existing output files should be appended (instead of over-written) at the start of each model run. (See Section 10.2.5 concerning this variable and the Experiment manager.) Valid values are `1` for yes and `0` for no.
siteLatitude (floating point)	The latitude of the study site, in degrees north (used to calculate day lengths).
shuffleYears (boolean: 0 or 1)	Whether the years of input data should be randomly shuffled. (See the model description document concerning year-shuffling.)
shuffleYearReplace (boolean)	Whether year-shuffling should be with (1) or without (0) replacement, if year-shuffling is used.
shuffleYearSeed (positive integer)	The random number generator seed used for year-shuffling.

3.5. Experiment Setup

The experiment setup file "Experiment.Setup" controls *inSTREAM*'s experiment manager. This feature is complex enough that Section 0 is dedicated to it.

Users are reminded to inspect Experiment.Setup any time *inSTREAM* appears to be ignoring a setup variable or parameter (another common source of frustration).

4. Parameter Files

inSTREAM requires two kinds of parameter files, for habitat and trout. These files provide the values for all the parameters defined in the model description document. Each habitat reach, and each trout species, must have a parameter file assigned to it (this assignment is done in the reach and species setup files). More than one reach, or species, can use the same parameter file. Parameter files use the Swarm object loader format (Section 2.3).

4.1. Habitat parameter file

The name of the habitat parameter file for a reach is defined by the user in the reach's setup file (Section 3.3). By convention, the file name includes the reach name, the syllable "Hab", and the extension ".Params"; for example, "SmithCrkMiddleReachHab.Params".

Figure 5 provides an example habitat parameter file. Habitat parameter files must include exactly the same variables as in this example. However, the order in which parameters appear does not matter.

```
# Example habitat parameter file for inSTREAM

@begin
habSearchProd                 7.0E-7
habDriftConc                  1.50E-10
habDriftRegenDist                  500
habPreyEnergyDensity              2500
habMaxSpawnFlow                   4.0
habShearParamA                   0.019
habShearParamB                   0.383
habShelterSpeedFrac              0.3
@end
```

Figure 5. Example habitat parameter file.

4.2. Trout parameter file

The user defines the name of the parameter file for each trout species in the species setup file (Section 3.2). By convention, the file name includes the species name, perhaps the site name, and the extension ".Params"; an example is "SmithCrkCutthroat.Params".

Figure 6 provides an example trout parameter file. All trout parameter files must include exactly the same variables as in this example; the parameters are defined in the model description document. (This figure does not necessarily include the "standard" or most up-to-date parameter values for any site.) The software checks to make sure all the trout parameters are initialized: if any trout parameters are missing from this file, an error statement will be issued and execution will stop.

```
@begin

fishCaptureParam1                  1.6
fishCaptureParam9                  0.5

fishCmaxParamA                   0.628
fishCmaxParamB                    -0.3
fishCmaxTempF1                    0.05
fishCmaxTempF2                    0.05
fishCmaxTempF3                     0.5
fishCmaxTempF4                       1
fishCmaxTempF5                     0.8
fishCmaxTempF6                       0
fishCmaxTempF7                       0
fishCmaxTempT1                       0
fishCmaxTempT2                       2
fishCmaxTempT3                      10
fishCmaxTempT4                      22
fishCmaxTempT5                      23
fishCmaxTempT6                      25
fishCmaxTempT7                     100

fishDetectDistParamA                 4.0
fishDetectDistParamB                 2.0
fishEnergyDensity                 5900

fishFecundParamA                  690
fishFecundParamB                  0.552

fishFitnessHorizon                  90

fishMaxSwimParamA                  2.8
fishMaxSwimParamB                   21
fishMaxSwimParamC               -0.0029
fishMaxSwimParamD                0.084
fishMaxSwimParamE                 0.37
fishMoveDistParamA                 100
fishMoveDistParamB                   2

fishPiscivoryLength               15.0

fishRespParamA                      30
fishRespParamB                   0.784
fishRespParamC                  0.0693
fishRespParamD                    0.03
fishSearchArea                   20000

fishSpawnEggViability               0.8

fishSpawnStartDate                10/1
fishSpawnEndDate                 11/30

fishSpawnDSuitD1                    0.0
fishSpawnDSuitD2                    5.0
fishSpawnDSuitD3                   50.0
fishSpawnDSuitD4                  100.0
fishSpawnDSuitD5                 1000.0
```

fishSpawnDSuitS1	0.0
fishSpawnDSuitS2	0.0
fishSpawnDSuitS3	1.0
fishSpawnDSuitS4	1.0
fishSpawnDSuitS5	0.0
fishSpawnMaxFlowChange	0.2
fishSpawnMaxTemp	14
fishSpawnMinTemp	5
fishSpawnMinAge	1
fishSpawnMinCond	0.985
fishSpawnMinLength	10
fishSpawnProb	0.04
fishSpawnVSuitS1	0.0
fishSpawnVSuitS2	0.0
fishSpawnVSuitS3	1.0
fishSpawnVSuitS4	1.0
fishSpawnVSuitS5	0.0
fishSpawnVSuitS6	0.0
fishSpawnVSuitV1	0.0
fishSpawnVSuitV2	10.0
fishSpawnVSuitV3	20.0
fishSpawnVSuitV4	75.0
fishSpawnVSuitV5	100.0
fishSpawnVSuitV6	1000.0
fishSpawnWtLossFraction	0.2
fishTurbidExp	-0.0711
fishTurbidMin	0.1
fishTurbidThreshold	5.0
fishWeightParamA	0.0124
fishWeightParamB	2.98
mortFishAqPredD1	20
mortFishAqPredD9	10
mortFishAqPredF1	18
mortFishAqPredF9	0
mortFishAqPredL1	4
mortFishAqPredL9	18
mortFishAqPredMin	0.92
mortFishAqPredP1	1.00E-05
mortFishAqPredP9	2.00E-06
mortFishAqPredT1	15
mortFishAqPredT9	8
mortFishAqPredU1	5
mortFishAqPredU9	80
mortFishConditionK1	0.3
mortFishConditionK9	0.6
mortFishHiTT1	28.0
mortFishHiTT9	24.0
mortFishStrandD1	-0.3
mortFishStrandD9	0.3
mortFishTerrPredD1	5
mortFishTerrPredD9	200
mortFishTerrPredF1	18
mortFishTerrPredF9	0
mortFishTerrPredH1	500

```
mortFishTerrPredH9                -100
mortFishTerrPredL1                   6
mortFishTerrPredL9                   3

mortFishTerrPredMin               0.95

mortFishTerrPredT1                  10
mortFishTerrPredT9                  50
mortFishTerrPredV1                  20
mortFishTerrPredV9                 200
mortFishVelocityV1                 1.8
mortFishVelocityV9                 1.4

mortReddDewaterSurv                0.9
mortReddHiTT1                     23.0
mortReddHiTT9                     17.5
mortReddLoTT1                      1.7
mortReddLoTT9                      4.0
mortReddScourDepth                 5.0

reddDevelParamA              -0.000253
reddDevelParamB                0.00134
reddDevelParamC              0.0000321
reddNewLengthMean                  2.8
reddNewLengthStdDev                0.2
reddSize                          1200

@end
```

Figure 6. Example trout parameter file.

5. Data Files

Five kinds of files are used to define habitat and the initial trout population.

5.1. Cell geometry

There is one geometry file per reach, providing the coordinates of the corners of each habitat cell. Cells are polygons that can have any number of sides. Each cell is designated by a cell number that can be arbitrary, but each cell's number must be unique (within its reach; cells in different reaches can have the same number). The geometry can be in any coordinate system, being treated by *inSTREAM* as plain Euclidean coordinates. Different reaches can use different coordinate systems with different origins; coordinate are transformed by *inSTREAM* into an internal spatial representation that does not consider the actual distance among reaches. However, *inSTREAM* follows the conventions of UTM coordinates, with units of meters and a Mercator projection that assumes (for display purposes only) the x direction is east and y is north.

The geometry file format is adopted from the "ungenerate" command of the ArcGIS geographic information system (other GIS packages can also use this format). "Ungenerate" exports the geometry of a polygon system as an ASCII text file. The file format is described in ArcGIS documentation and illustrated by Figure 7. This example depicts three cells, numbers 1, 3, and 4. Each cell is described by a block of data lines, with values separated by spaces (the spacing within a line is not important).

The first line for a cell contains the cell number, followed by x and y coordinates of a cell label that is ignored by *inSTREAM*. Following lines provide the x and y coordinates of each corner of the cell. A line containing just "END" delineates the end of each cell, and two such lines indicate the end of input.

```
         1      2923.3222656      2084.7429199
      2917.0759277      2079.6772461
      2926.8261719      2089.6940918
      2929.4162598      2089.6560059
      2918.4470215      2078.3823242
      2917.0759277      2079.6772461
END
         3      2936.2717285      2087.4851074
      2929.4162598      2089.6560059
      2942.2133789      2087.4470215
      2942.0610352      2084.7048340
      2930.1779785      2087.5612793
      2929.4162598      2089.6560059
END
         4      2937.0334473      2081.5056152
      2942.0610352      2084.7048340
      2941.4516602      2076.0590820
      2934.9006348      2073.7739258
      2930.1779785      2087.5612793
      2942.0610352      2084.7048340
END
END
```

Figure 7. Example cell geometry file.

5.2. Cell data

There is one cell data file for each reach modeled; it provides the static habitat variable values for each cell. The name of a reach's cell data file is specified by the user in the reach setup file. The convention is for these file names to include the reach name and end in "Cell.Data" (e.g., SmithCrkLowerReachCell.Data).

The file still starts with three rows of header information, followed by one row for each cell in the reach. Each of these cell rows contains only four values, separated by spaces, tabs, or commas. The values are:

- The cell number, using the same numbering as in the cell geometry file (Section 5.1). Cells need not appear in any particular order, and cell numbers need not be sequential.
- The fraction of the cell providing velocity shelter for drift-feeding trout.
- A characteristic distance to hiding cover for the cell (in meters, not cm).
- The fraction of the cell providing suitable spawning gravel.
- A code for which end of the reach the cell is at. Valid values are "U" if the cell is in the main channel at the reach's upstream end, "D" if the cell is in the main channel at the downstream end, and "I" for all other cells. These codes are used by fish to calculate how far they are from their reach's ends.

Figure 8 is a (partial) example cell data file. The values in this file can be separated by spaces or tabs, or the file can be in CSV format (Section 2.1; illustrated in Figure 8).

```
Turbid Creek Upstream site cell habitat variables file
Last modified: SFR 4/20/2011
Cell#,FracVelShelter,DistToHidingCover,FracSpawnGravel,ReachEndCode
2,0.1,0.5,0.0,U
4,0.0,2.2,0.0,I
5,0.1,1.5,0.0,U
6,0.1,1.5,0.0,U
7,0.1,1.5,0.5,U
...
```

Figure 8. Example cell data file.

5.3. Hydraulic data

To model how the depth and velocity of each cell varies with flow, *inSTREAM* imports the depth and velocity of each cell at each of many flows (explained in the model description). This information is in one hydraulic data file per reach. The names of these files are provided by the user, for each habitat reach, in the reach setup file (Section 3.3). The naming convention for hydraulic data files is to include the reach name and end with "Hyd.Data", e.g., SmithCrkHyd.Data. However, if the file is in CSV format, its extension should be .csv, e.g., SmithCrkHydData.csv.

The hydraulic data file format was designed assuming that its contents would be prepared using geographic information system (GIS) software and analysis. This design allows output from any

hydraulic model (including one-dimensional models) to be used in *inSTREAM* by importing hydraulic model geometry and results to GIS, processing and checking it as needed, and saving the information in the format used by *inSTREAM*. Another reason for using GIS to process hydraulic data is that it allows the hydraulic model to use a spatial resolution and grid mesh different from the cells used in *inSTREAM*. As the Clear Creek example (Sect. 8.2) shows, results from a highly detailed hydraulic simulation can be processed in GIS to produce average depths and velocities for the larger *inSTREAM* cells. The depths and velocities should be depth-averaged values that represent the entire cell area.

Depths and velocities in this file must be in units of meters (m) and m/s, and flow in m^3/s. This is an exception to *inSTREAM*'s convention of using cm for length units.

Some hydraulic models assign a negative depth to cells that are above water, so *inSTREAM* converts any negative depths to zero when they are read in. However, any negative velocities are treated as an error because input should be the velocity magnitude, which cannot be negative—hence, negative velocity input likely indicates that a velocity component is mistakenly being used instead of the magnitude. If the model encounters a negative velocity it raises an error statement and stops.

The hydraulic data file for each reach (illustrated at Figure 9) includes:

- Two lines of header information ignored by the computer.
- A line that starts with the word "Flows:", and then lists all the flows in the lookup table. The flows must be in ascending order, and in units of m^3/s.
- Another line of header information that should be used to label the following columns by which flow they represent and whether they contain depth or velocity (in the example below, "D@1.42" refers to depth at 1.42 m^3/s, etc.).
- One line for each cell, containing the cell number, then the depth (m) followed by velocity (m/s) for each flow.

This file can also be in CSV format.

```
Hydraulic data input file for inSTREAM 5.0
Clear Creek Reach 3a Data 04 OCT 2010 by D.S. Montoya
Flows: 1.42         2.12          2.83
Cell D@1.42 V@1.42 D@2.12 V@2.12 D@2.83 V@2.83
2       -0.73 0.00   -0.70  0.00   -0.67  0.00
4       -0.70 0.00   -0.67  0.00   -0.64  0.00
5       -0.72 0.00   -0.68  0.00   -0.65  0.00
6       -0.60 0.00   -0.56  0.00   -0.53  0.00
7       0.77  0.07   0.80   0.10   0.83   0.12
8       1.15  0.13   1.18   0.19   1.21   0.24
10      0.96  0.12   0.99   0.17   1.02   0.22
...
```

Figure 9. Example hydraulic data file.

The software makes several checks when reading in the cell hydraulic file:

- The cell numbers must correspond with those in the cell geometry file: each cell defined in the geometry file receives hydraulic input and there are no cells in the hydraulic file but not in the geometry file.
- The flows on line 3 must be in order of increasing magnitude.
- The number of values on each row of hydraulic input (file lines 5 and higher) must equal one (the cell number) plus two times the number of flows defined on line 3.
- There must be no negative velocities.
- Any negative depth values are set to zero (some hydraulic models assign negative depths to locations above the water level).

5.4. Initial population data

This file specifies the initial population of trout that is created at the start of a model run. One file is provided for each species, with the file name provided by the user in the species setup file. When multiple reaches are simulated, initial population data for all reaches are provided in the same file. The file can be in formatted text, or in .CSV format so it can be maintained in a spreadsheet that looks like Figure 10. The file name convention is to include the species and study site names, and end in “InitPops.csv” or “InitPops.Data” (e.g., SmithCrkRainbowInitPops.Data).

The file starts with three comment lines that are ignored by the software; the third line is usually column headings (Figure 10). Next come lines of data that each specify the initial number and size of trout of one age, for one reach.

Initial fish population file for Big Smith Creek					
From field data compiled February 2005					
InitDate	Age	Number	MeanLength (cm)	StdDevLength (cm)	Reach
10/1/2002	0	400.8	10.22	1.4	LowerMainstem
10/1/2002	1	93.6	16.52	1.26	LowerMainstem
10/1/2002	2	19.2	19.04	1.12	LowerMainstem
10/1/2002	3	45.6	27.72	0.98	LowerMainstem
10/1/2003	0	295.2	10.22	0.98	LowerMainstem
10/1/2003	1	55.2	17.22	0.98	LowerMainstem
10/1/2003	2	12	20.02	1.26	LowerMainstem
10/1/2003	3	2.4	28	1.12	LowerMainstem
10/1/2004	0	206.4	11.48	1.12	LowerMainstem
10/1/2004	1	69.6	16.8	0.56	LowerMainstem
10/1/2004	2	52.8	20.72	1.12	LowerMainstem
10/1/2004	3	16.8	28.42	1.54	LowerMainstem
10/1/2002	0	247.2	9.38	0.98	UpperMainstem
10/1/2002	1	40.8	15.54	0.84	UpperMainstem
10/1/2002	2	28.8	20.16	1.4	UpperMainstem
10/1/2002	3	16.8	25.48	2.94	UpperMainstem
10/1/2003	0	172.8	9.66	1.12	UpperMainstem
10/1/2003	1	26.4	16.24	1.12	UpperMainstem
10/1/2003	2	16.8	20.16	1.4	UpperMainstem
10/1/2003	3	0	23.52	1.4	UpperMainstem
10/1/2004	0	112.8	10.5	0.98	UpperMainstem
10/1/2004	1	16.8	17.22	0.42	UpperMainstem
10/1/2004	2	19.2	19.74	1.12	UpperMainstem
10/1/2004	3	2.4	23.66	1.4	UpperMainstem

Figure 10. Example population initialization data file. This file defines initial populations for two reaches and three years.

Each line includes the following values:

- The initialization date. This date is used only as an index allowing the initial population file to contain more than one set of initialization data, which the user selects via the Model.Setup file variable popInitDate. Figure 10 is an example in which the user may want to start the model using data representing October 1 of 2002, 2003, or 2004; any of these initialization data sets can be selected for a particular model run, by setting the value of popInitDate in Model.Setup to 10/1/2002, etc. The initialization date does not need to be the date on which simulations actually begin. Arbitrary values of the initialization date can be used to provide alternative initial population scenarios for simulation experiments.

- The age of fish to initialize.
- The number of fish of the specified age to create.
- The mean length (cm) of the initial fish.
- The standard deviation (cm) of the initial fish. The length of each fish is drawn randomly from a normal distribution defined by the mean and standard deviation specified here.
- The name of the reach for which the fish are initialized. This name must be exactly equal to one of the reach names provided in the reach setup file.

These lines must be in the order illustrated in Figure 10. Lines for one initialization date must be grouped together. Within the block for each initialization date, lines for each reach must be together. Finally, lines for each reach must be sorted in increasing order by age.

There is no limit to how many ages can be initialized, and the initial ages need not correspond to the age classes used to summarize output (Section 6.2).

5.5. Flow, temperature, and turbidity time series data

inSTREAM uses daily input values of flow, temperature, and turbidity, for each reach. Each of these variables is provided in a separate file (each data file provides one time series of one variable). The names of the flow, temperature, and turbidity data files used by each reach is specified by the user in the reach setup file. The file names usually include the reach name, the type of data, and the extension ".csv" or ".Data" (e.g., SmithCrkFlow.csv, SmithCrkTurbid.Data).

These data files are read by the EcoSwarm class TimeSeriesInputManager. The format for each is:

- The first three lines are headers ignored by the software. The third line usually provides column headings.
- Remaining lines provide the value for one day. Each line starts with the date (mm/dd/yyyy), followed by the daily value.

The data lines must be in increasing order by date, and no days (including leap days) may be omitted. Flow values are in m^3/s, temperatures in °C, and turbidity in NTU. Figure 11 is an example file. CSV format is allowed.

```
Temperature data for LJC Lower Site. From 1999-00 measured values
Assembled 1/11/01
Date  Temperature (C)
10/1/1987   11.6
10/2/1987   11.6
10/3/1987   11.5
10/4/1987   11.4
10/5/1987   11.4
10/6/1987   11.3
10/7/1987   11.3
10/8/1987   11.2
10/9/1987   11.1
10/10/1987  11
10/11/1987  10.9
10/12/1987  10.7
...
```

Figure 11. Example file for flow, temperature, and turbidity input.

These files may include dates outside the range being simulated in any particular model run; *inSTREAM* simply finds the values it needs from within the data files. Flow data must include the day after the last simulation day (for modeling redd scouring mortality).

Spreadsheet software is especially convenient for building these time series data files. However, when data are imported to a spreadsheet it can convert the dates to a different format (e.g., dates show up in the spreadsheet as “10/1/87” instead of “10/1/1987”), a common source of frustration. It can therefore be necessary to re-format the date column in the spreadsheet before saving it as the ASCII or CSV file used by *inSTREAM*. Another frequent problem is spreadsheet columns that are slightly too narrow, so some dates (e.g., 10/15/1987 but not 1/5/1987) are saved as “########”. (Using CSV format to save the file for input to *inSTREAM* avoids this problem.) Parsing errors also commonly occur because the number of characters in a date changes. If the date column starts at 1/1/2001, the spreadsheet may decide that the date column needs to be only 8 characters wide. When dates like 10/10/2001 are reached, the last characters end up in the second spreadsheet column, ruining the file. Always inspect spreadsheets carefully before saving the data for input to *inSTREAM*.

6. Changing Species and Size Classes

Minor changes to the software’s source code are needed to change either the species represented by *inSTREAM* or the age classes used to summarize and report output (Section 8). These changes do not require programming skills if the following directions are followed, but it is recommended that users save their original source code separately before making the changes.

6.1. Adding or Removing Species

inSTREAM can use up to 10 species or races of trout. The number and names of species included in a run is specified in the Species.Setup file. However, this setup file can only refer to species that exist in the software as a subclass of the “Trout” class. To exist in the software, a species must have its own interface (.h) file and implementation (.m) file. Therefore, adding a species is simply a matter of creating an interface and implementation file for the species; and telling the software to include those files when it is compiled, by including the new file names in

the Makefile (explained below). Likewise, species can be removed by deleting their files and removing reference to them from the Makefile. And one species can be replaced by another by simply replacing the species name in all the locations discussed here.

It is critical that the name of the new species' interface and implementation files exactly match the species' name in the Species.Setup file. The new .h and .m files must of course be in the same directory as the rest of the source code.

The following steps add a species to *inSTREAM*. As an example, the rare blarney trout is added to a model already containing rainbow and brown trout. If any of the six changes are incomplete, the software may exit with an error or crash without explanation.

Step 1. Create the species interface file. This is easiest done by copying and editing the .h file of an existing species. For example, the file "Rainbow.h" can be copied to a new file "Blarney.h" and edited to:

```
#import "Trout.h"
@interface Blarney : Trout
{

}
+ createBegin: aZone;

@end
```

Step 2. Create the species implementation file. Again, this is easiest done by copying and editing the .m file of an existing species. The new "Blarney.m" file should be edited to:

```
#import "globals.h"
#import "Blarney.h"

@implementation Blarney

+ createBegin: aZone
{
   return [super createBegin: aZone];
}

@end
```

Step 3. Add the new species to the TroutModelSwarm.h file. Near the top of TroutModelSwarm.h is a series of "#import" statements that say which other .h files are used by TroutModelSwarm.h. Simply add a new #import statement for the new species, just like the statement for existing species:

```
#import <stdlib.h>
#import <objectbase/Swarm.h>
#import <analysis/Averager.h>

#import "TroutModelSwarmP.h"
```

```
#import "globals.h"
#import "Rainbow.h"
#import "Brown.h"
#import "Blarney.h"  // New species added here
#import "Redd.h"
#import "HabitatSpace.h"
#import "FishParams.h"
```

Step 4. Modify the Species.Setup file by adding lines for the new species (see Section 3.2).

Step 5. Modify the Model.Setup file to increase the parameter *numberOfSpecies*. The value of this parameter must not be greater than the number of species listed in Species.Setup.

Step 6. Edit the Makefile in three places. In the source code directory is a file called “Makefile”, which provides the compiler with directions for which source code files to include in *inSTREAM*. Users need not try to understand the Makefile, but can just add the new species wherever an existing species is found. First, the OBJECTS statement must have the new species’ class file in it. This statement should look like (your makefile may not look exactly like this):

```
OBJECTS=Trout.o Cell.o Vector.o HabitatSpace.o \
      Redd.o \
      TroutModelSwarm.o \
      TroutObserverSwarm.o \
…

\
        Rainbow.o \
        Brown.o \
        Blarney.o \
\
```

(The “\” characters just mean that the statement continues onto the next line.)

Second, add the new .h file to this statement:

```
TroutModelSwarm.o: TroutModelSwarm.[hm] globals.h  \
      Rainbow.h Brown.h Blarney.h HabitatSpace.h \
      FishParams.h DEBUGFLAGS.h
```

Finally, a line must be added to the statements for each species:

```
Rainbow.o : Rainbow.[hm] DEBUGFLAGS.h
Brown.o : Brown.[hm] DEBUGFLAGS.h
Blarney.o : Blarney.[hm] DEBUGFLAGS.h
…
```

These steps add a new species that has exactly the same formulation as the other species (although the new species can have its own parameter values). Differences in formulation

among species can be implemented by copying the relevant methods from Trout.h and Trout.m to the species' source code files and revising them.

After these changes are made, the software must be re-compiled to create a new version of the executable file instream.exe.

6.2. Changing Age Classes

The age classes used to summarize model results must be defined in the *inSTREAM* source code, but it is easy to modify the code to add or remove age classes. (The number of age classes do not affect simulations at all, only how statistical summaries of results are calculated and reported.)

Most of the changes are made in the file TroutModelSwarm.m. The first such change is in the following lines, which appear in the method "buildObjects":

```
ageSymbolList = [List create: modelZone];

Age0     = [Symbol create: modelZone setName: "Age0"];
[ageSymbolList addLast: Age0];
Age1     = [Symbol create: modelZone setName: "Age1"];
[ageSymbolList addLast: Age1];
Age2     = [Symbol create: modelZone setName: "Age2"];
[ageSymbolList addLast: Age2];
Age3Plus     = [Symbol create: modelZone setName: "Age3Plus"];
[ageSymbolList addLast: Age3Plus];
```

These lines define the age classes, and can be edited to change the number and range of classes. For example, this change breaks the age category of 3 years and higher into two classes (the changes are in bold text):

```
ageSymbolList = [List create: modelZone];

Age0     = [Symbol create: modelZone setName: "Age0"];
[ageSymbolList addLast: Age0];
Age1     = [Symbol create: modelZone setName: "Age1"];
[ageSymbolList addLast: Age1];
Age2     = [Symbol create: modelZone setName: "Age2"];
[ageSymbolList addLast: Age2];
Age3     = [Symbol create: modelZone setName: "Age3"];
[ageSymbolList addLast: Age3];
Age4Plus     = [Symbol create: modelZone setName: "Age4Plus"];
[ageSymbolList addLast: Age4Plus];
```

The second change must be made in the TroutModelSwarm method "getAgeSymbolForLength", which assigns a age class to a fish, using the fish's length. For the default age classes, this method is:

```
- (id <Symbol>) getAgeSymbolForAge: (int) anAge
{
  int fishAge = anAge;

  if(fishAge >= 3)
  {
    fishAge = 3;
  }
  return [ageSymbolList atOffset: fishAge];
}
```

For the above example in which a new age 3 class is added, the method must be changed to:

```
- (id <Symbol>) getAgeSymbolForAge: (int) anAge
{
  int fishAge = anAge;

  if(fishAge >= 4)
  {
    fishAge = 4;
  }
  return [ageSymbolList atOffset: fishAge];
}
```

Finally, the file TroutModelSwarm.h must be edited to declare the new age class symbols. Change this block of code:

```
id <List> ageSymbolList;
id <Symbol> Age0;
id <Symbol> Age1;
id <Symbol> Age2;
id <Symbol> Age3Plus;
```

to this:

```
id <List> ageSymbolList;
id <Symbol> Age0;
id <Symbol> Age1;
id <Symbol> Age2;
id <Symbol> Age3;
id <Symbol> Age4Plus;
```

After these changes are made and the code re-compiled (using `make clean` first), *inSTREAM* will automatically use the new classes to report results that are broken out by age class.

7. Graphical Interfaces

When the *inSTREAM* software is executed in its default graphics mode, a number of graphical interfaces are provided. Most of these are simply graphs that display results, but the animation window is truly an interactive interface to the model, allowing users to probe deeply to observe and even alter the state of individual habitat cells, fish, and redds. Control panels can be used to start, stop, or step through model runs.

Users just interested in starting up their first model runs only need to know that they must hit the "Start" button on the first control panel *twice* to get execution underway.

It is important to understand that the graphical interfaces are updated only at the end of each daily time step, after all other scheduled actions are complete. Mouse clicks and other input to the interfaces are accepted only at the end of the time step: you cannot stop a simulation part way through a day's schedule, and displayed information reflects the model's state at the end of a time step.

When the model is started in batch mode (via the GUI or by using the command `instream.exe -b`), none of these graphics appear. Instead, simulations start immediately and run until completed (or until something goes wrong, or the user kills the job).

7.1. Main Control Panel

As soon as *inSTREAM* is started in graphics mode, the main control panel opens up (Figure 12). This panel essentially operates the Experiment Manager, which then starts the model runs. The Experiment Manager (Section 0) may be set up so only one model run is executed, or so that a multi-run experiment is executed. Users need to know only the following:

Figure 12. Main control panel.

- Model execution is started by hitting the "Start" button once (which creates the Experiment Manager), then a second time (which starts the first model run). After the "Start" button is hit the second time, control of the first model run is passed to the Model Run Controller (described below).

- If the Experiment Manager is set up for a single run, execution can be terminated cleanly by hitting the "Quit" button after the run is finished.

- If the Experiment Manager is set up for multiple runs, the “Start” button must be hit to start each run. After each run finishs, nothing happens until this button is hit again.

- The “Save” button is supposed to tell Swarm to preserve the spatial arrangement of graphical interface windows on the screen and use it next time the code is executed. However, this facility is not reliable and the hidden file it creates can become corrupted and cause frustrating problems. Users are strongly recommended not to use it.

- The “Quit” button stops execution and closes the code.

7.2. Model Run Controller

When a model run has been started from the main control panel, it opens a second control panel labeled “Model Run Controller” (Figure 13). This panel displays the simulation’s current time step (the number of simulation days that have been completed, starting with zero). It also has six buttons that can be used to control execution.

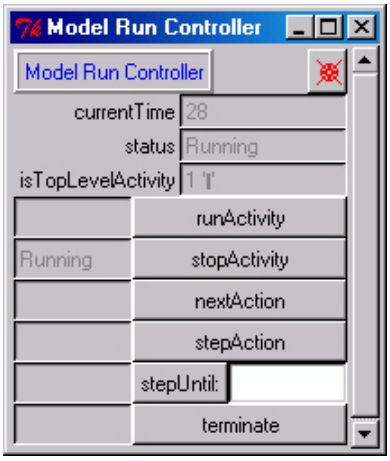


Figure 13. Model Run Controller.

- “runActivity” causes the model to resume execution after it has been paused.

- “stopActivity” causes the model to pause execution (after the current time step has finished).

- “nextAction” causes execution of one more time step, after which execution is again stopped. (Users may have to hit this button a number of times before it executes a full time step; it seems to work well after the “StepUntil” button is used, also.)

- “stepAction” causes execution of one individual action in the model’s schedule (e.g., spawn, move, grow, die).

- “stepUntil” causes execution to continue until it reaches (but does not yet execute) a specified time step. To the right of this button is a window where the time step is input. For example, if the model is stopped at time step 5 and the user wants it to continue for

3 more days, the number 9 should be entered in the stepUntil input window (by typing "9" then hitting the Enter key), and stepUntil pressed.

- "terminate" causes execution to end the current model run, passing control back to the main control panel. The proper way to shut down execution during a model run is to hit "stopActivity", then this button, then "Quit" on the main control panel. (However, causing the model to crash by hitting "terminate" several times does not hurt anything.)

7.3. Habitat Probe Displays

A "probe display" window (probe displays are explained in Section 7.5) is opened for each habitat reach. These are small windows (Figure 14) displaying key habitat variables: the current date, flow, temperature, turbidity, etc. The first variable displayed is the reach's name.

Method probes are in the lower part of the display probes (switchColorRep to unTagAllPolyCells in Figure 14) and have an (initially) blank value window to the left of the method name. A method can be executed by clicking on the button displaying the method's name. When clicked on, these method probes return a value in the window to the left of the button.

Some method probes execute methods that require a parameter (an input to the method). These probes appear as a button with a window on its right (where the parameter value is entered before clicking the button) and with a window on its left, where the value returned from the method is displayed. The tagCellNumber in Figure 14 is an example: the required parameter is the number of the cell that will be "tagged" by changing its color; this number must be entered in the box to the right of the "stepUntil" button, then the button clicked on.

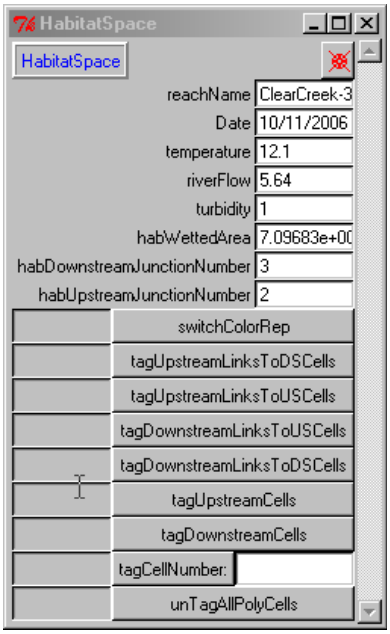


Figure 14. Habitat probe display. One of these windows is opened for each habitat reach.

7.4. Graphs

Graphs can provide summary information on the trout population. These graphs report the status of all trout in the model; they are not broken out by habitat reach, species, age, or any other characteristic. Two graphs are normally displayed.

- A line graph (time series plot) of the cumulative number of trout that have died of each mortality source (Figure 15). The X axis is the number of simulation days. Clicking on any mortality source in the graph's legend highlights the line for that kind of mortality.

- A bar graph showing how many fish are currently alive, by age.

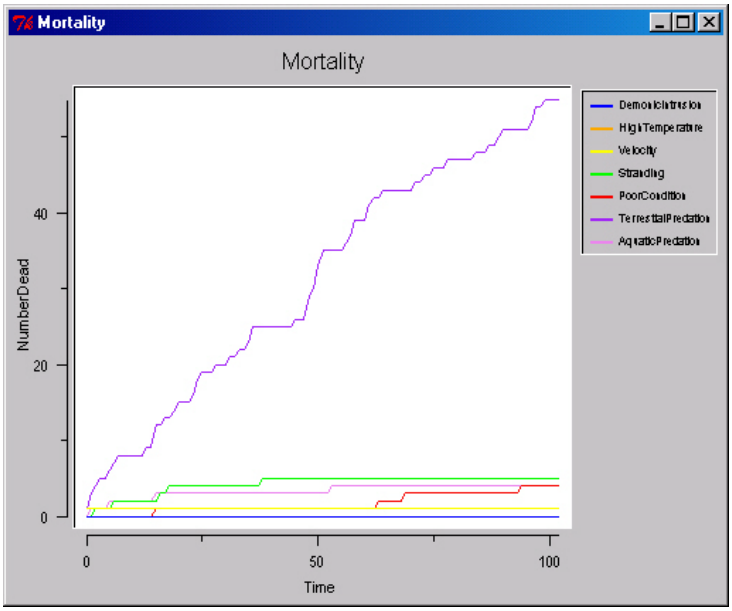


Figure 15. Mortality output graph. "Time" on the X axis is the number of days simulated. The Y axis is the total number of trout that have died of each of the seven mortality sources.

7.5. Animation Window

The primary interactive interface is the animation window, which shows the habitat cells, fish, and redds as the simulation proceeds (Figure 16). An animation window is opened for each habitat reach. The window displays the habitat cells as a plan view (looking from the top down) map. The size of the window can be adjusted in both dimensions using parameters in the observer setup file.

The cells are shaded by either depth (white to dark blue) or velocity (yellow to red), according to the observer setup file parameter *rasterColorVariable*, and whether depth or velocity is used can be changed via the habitat space probe display (Section 7.3). (The shading scheme is defined in the method drawSelfOn in file FishCell.m.)

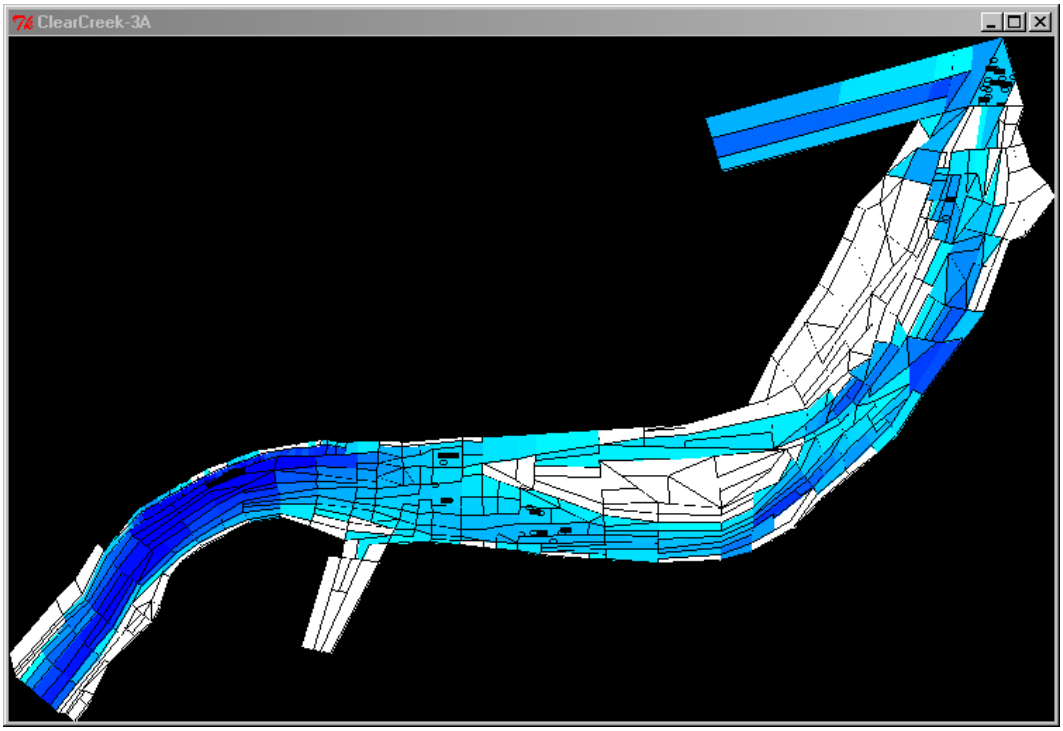


Figure 16. Animation window. One such window is opened for each habitat reach, with the reach name used as the window's title (top left).

Adult fish (age > 0) appear as filled rectangles at a random location in the cell they occupy. The size of their rectangles increases with age. Fish colors depend on species and are set in the species setup file. Juvenile fish (age 0) appear as individual pixels (dots) in their cell. Redds appear as an open oval. Redds are also color-coded by species.

7.5.1. Probe displays

The habitat cells, fish, and redds can be "probed" from the animation window, allowing them to be examined in detail and even altered. To open a probe display to a cell, click on it with the left mouse button. A right mouse button click on a cell opens probe displays for all the fish and redds in the cell. It is usually desirable to pausing execution with the model run controller's "stopActivity" button before attempting to open probe displays, but once open the probe displays can stay open when execution is resumed so the user can see how variables change over time. To close a probe display, click on the red button near its upper right corner.

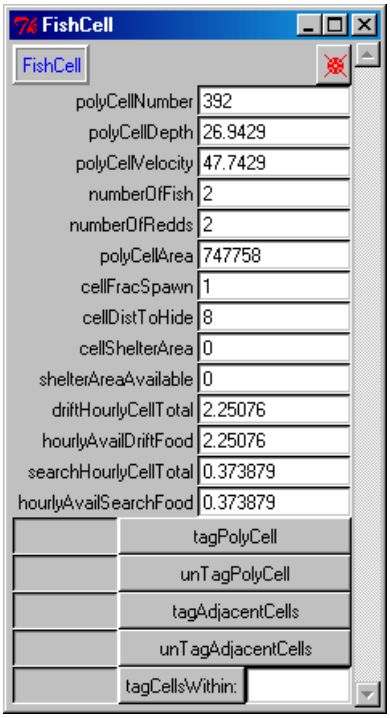


Figure 17. Cell probe display.

Probe displays include two kinds of probes: variable probes allow the user to see and change the value of a particular variable, and method probes allow the user to execute one of the object's methods (a piece of its program that executes some particular function, similar to a subroutine).

Variable probes are in the upper part of the display (polyCellNumber to hourlyAvailSearchFood in Figure 17); variable values show up in little windows to the right of the variable name. The value for a variable can be replaced manually by entering it into the value window over the old value, and then hitting "Enter" (but see the following note about when values are updated).

Note that the animation window is updated after habitat and fish simulations are completed each time step, and before habitat variables (depth, velocity, temperature, flow) are updated for the next time step. Therefore, any time-variable cell variables changed via probes will be re-set and overwritten before the next fish simulations. For example, using the probe to change a cell's depth or velocity will have no effect because these variables are updated with a new daily value as soon as execution is resumed.

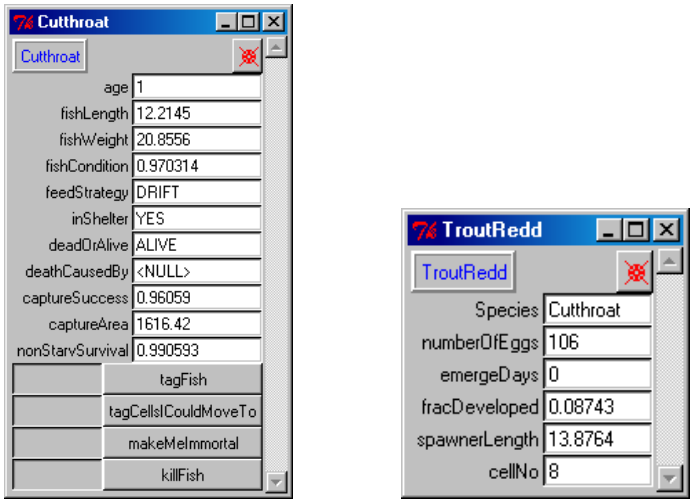


Figure 18. Trout and redd probe displays.

The default probe display for a trout includes four method probes that do the following things:

- "tagFish" turns the probed trout a different color in the animation window, so it can be followed when simulations are resumed. (This color is set in the Model.Setup file.)
- "tagCellsICouldMoveTo" temporarily highlights the cells that are potential movement destinations of the trout (as defined in the model description). The trout's current cell is not highlighted.
- "makeMeImmortal" causes the probed trout to be exempt from mortality for the rest of the simulation. None of the trout's behavior is changed.
- "killFish" causes the trout to die immediately, with the mortality source being demonic intrusion. ("killFish" trumps "makeMeImmortal".)

The probe displays that can be opened from the animation window include only a few selected variables and methods. (Variables and methods can be added to or removed from these displays by editing the code in the method "buildProbesIn" or—for trout—"buildFishProbes", in the code file "TroutObserverSwarm.m".) However, a "complete probe display" that shows all of an object's variables and methods can be opened by right-clicking on the box, at the top left of a probe display window, that states the object's class (either "Cell", "Redd", or the species name). These complete probe displays, like all others, are closed by clicking on the red button at their top right corner.

For a trout, this complete probe display is empty because the methods and variables for a trout are in the Trout superclass, not the subclass for each species. However, all the complete probe display windows include a button, at the top right corner of the window, showing two green boxes and an arrow from the lower box to the upper (Figure 19). This button opens a complete

probe display of the object's superclass. Therefore, to open a complete probe display for a trout, right-click on the blue species name box; then, in the newly opened window, click on the green superclass button.

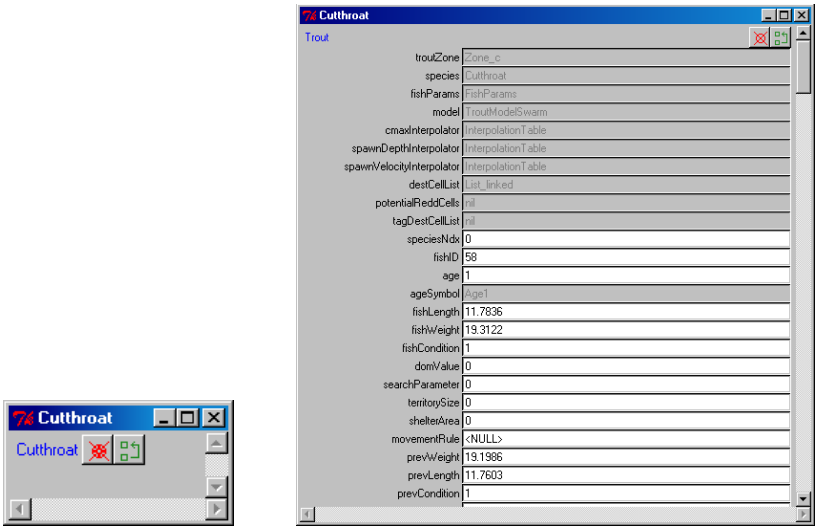


Figure 19. Right-clicking on the species name box in a trout probe display opens a (empty) complete probe display (left). Clicking on the green button opens a complete probe display for the trout (right).

7.5.2. Movies of the animation window

It is relatively easy to make a digital "movie" of the animation window (or full screen) during a model run; these movies can be used on web sites, in presentations, or in digital appendices to publications. Movies are made by having *inSTREAM* write one graphics output file for each time step, then assembling these files into an animation file.

By default, *inSTREAM* produces the graphics output by capturing the animation window for the first reach in the Reach.Setup file (Section 3.3). Alternatively, the entire screen can be captured, as described below. The steps are:

- In the observer setup file, set the parameter *takeRasterPictures* to YES. This will cause *inSTREAM* to write, at the end of each daily time step, a new graphics file that contains an image of the animation window. The files are in Portable Network Graphics (.png) format, and have sequential file names ("Model001_Frame001.png", "Model001_Frame002.png", etc.).

- Run the model in graphics mode. To protect the animation output, the animation window now remains on top of any other windows.

- Usually, the model should be stopped after 50-100 time steps to keep from producing too many frames to use in a reasonably sized movie.

- Animation software is used to assemble the frames into a movie. GIF Construction Set Professional, for example, is inexpensive shareware that allows the .png files to rapidly be assembled into movies in either .gif of .avi format.

A simple code change causes *inSTREAM* to write out pictures of the entire screen, not just the animation window. In the method writeFrame of source file TroutObserverSwarm.m, simply comment out the statement "[pixID setWidget: raster];" and re-compile the model.

8. File Output

This section describes the main output files that are written for all model runs. These output files provide the kind of detailed summary statistics that are usually most useful for understanding what happened during a simulation. (Section 9 describes optional output files that provide more detailed results for specific parts of the model.)

8.1. General Information on Output Files

This section briefly describes characteristics common to the main output files. All these files are written in CSV format so they are easily opened in spreadsheet software. (See Section 2.1 concerning CSV files.)

The two fish output files are generated using the EcoSwarm BreakoutReporter, a generic output file-writing tool. The BreakoutReporter makes it easy to modify the software to get more, less, or different detail in the output files. By changing a few simple statements in the code, users can (1) output additional variables, (2) output different summary statistics (minimum, mean, maximum, count, sum, standard deviation, variance) for a variable, and (3) change the fish characteristics (e.g., species, age class, habitat reach) by which results are broken out or the order in which results are broken out (e.g., species first, then age; vs. age first, then species). The code for these output files is in the method createBreakoutReporters in the class TroutModelSwarm. (For example, code to add output of the minimum and maximum length and weight of fish is currently written, but commented out, in createBreakoutReporters.) Documentation for the BreakoutReporter is at www.humboldt.edu/ecomodel/software.htm.

Starting with version 5.0 of *inSTREAM*, the BreakoutReporter produces output designed for use with Excel's "PivotTable" feature and statistical software that can summarize data identified by categorical variables. Instead of producing separate columns of output for each category of results (each combination of species, site, size class, etc.), the output files now report all results in one column, with separate columns indicating the category (species, site, etc.) corresponding to the output line.

Output files report the scenario and replicate numbers generated by the Experiment Manager (Section 0). These numbers are needed to separate results from multiple model runs generated by the Experiment Manager.

If a model run stops before it is finished (because an error occurs or because it is killed by the user), the output files will persist and provide results up to when the simulation stopped.

Users should pay attention to the *appendFiles* parameter and how it is set in Model.Setup. This parameter controls whether output files are overwritten vs. appended each time a model run

starts. However, model runs started by the Experiment Manager always (after the first run of a batch) append results, so results from all runs in an experiment batch are appended in one file.

The parameter *fileOutputFrequency* in Model.Setup controls how often output is written to the two fish output files. Using a higher value of *fileOutputFrequency* can reduce execution time (because generating the fish outputs using BreakoutReport requires quite a few computations) and reduce the volume of output to be stored and analyzed.

8.2. Output File Descriptions

The main output files are described here. New users can most easily view these files by opening them in spreadsheet software.

Live fish output file. This file reports a time series of summary statistics on the live trout population. The file name is provided by the user, via the parameter *fishOutputFile* in the model setup file (Section 3.4). By default, this file provides the abundance, mean weight, and mean length of the trout population, broken out by habitat reach, trout species, and age class.

Fish mortality output file. This file reports mortality statistics: the number of trout that have died via each mortality source. Results are broken out by habitat reach, species, and age class. On each output date, lines written to this file report the number of trout that have died of each type of mortality since the previous output date.

Redd output file. This file provides one line of results for each redd, written on the day when the redd becomes empty (when all its eggs have died or turned into new trout). The file name is specified by the user, via the parameter *reddOutputFile* in the model setup file. Results reported for each redd include the spawner's length, weight, and age; the dates the redd was created and emptied; the redd's location (reach and cell); the initial number of eggs; and the number of eggs dying of each redd mortality source and the number emerging as new trout. The output also includes a "ReddID", which is simply a unique alphanumeric code for each redd.

Habitat output files. These files output the values of the time-series habitat variables. One file is written for each habitat reach; its name is the reach's name with "Habitat.out" appended to it (e.g., MainstemReachHabitat.out). Each line reports the date and the daily flow, temperature, and turbidity values (which were input via the files described in Section 5.5).

These habitat output files are actually optional (Section 9) and can be suppressed by commenting out the statement `#define HABITAT_REPORT_ON` in the code file HabitatManager.h.

9. Optional Output Files for Testing and Specialized Studies

The *inSTREAM* software includes a number of optional output files that are normally not written, but can be turned on when more detailed output is desired. These "reports" are often useful for testing changes in the software or parameters, and for supporting more detailed analysis of trout behavior. However, many of these optional output files can be extremely large and writing them can make *inSTREAM* much slower to execute.

The optional output files are turned on adding optional lines to the Model.Setup file (Section 3.4). These optional lines contain the name of a boolean variable controlling the optional output,

followed by the numeral “1”. These lines can be added anywhere between the `@begin` and `@end` lines. For example, the optional redd mortality file is switched on by adding this line to Model.Setup:

```
writeReddMortReport      1
```

The optional output files can be switched back off by setting their control variable to “0” instead of “1”. The redd mortality report will not be written if this is in Model.Setup:

```
writeReddMortReport      0
```

Table 4 provides a complete list of the optional output files. The first column provides the control variable to add to Model.Setup to activate the file. The second column provides the name of the output file. Some of the optional reports produce a separate output file for each habitat reach, in which case the file name starts with the reach name. The third column describes the information provided in the file.

With two exceptions, these optional output files are not controlled by the Model.Setup parameter *appendFiles*; instead, they are always overwritten at the start of each model run. Even for multiple model runs generated by the Experiment Manager (Section 0), these files will report results only from the last model run.

The exceptions are the CellFishInfo.rpt and IndividualFishReport.csv outputs (the last two lines of Table 4). These outputs are controlled by *appendFiles*, and also designed so that results from multiple runs generated by the Experiment Manager are automatically appended.

Table 4. Optional output files.

Control variable for Model.Setup (boolean)	Report file name	Description of report
writeFoodAvailabilityReport	FoodAvailability.rpt	Produces one line each time a fish moves into a cell. Reports the cell’s hourly food production rate, the hourly rate of food availability (unused by fish already in the cell), and the amount consumed by the fish. Results are separate for drift and search food types.
writeCellCentroidReport	(reachname) _Cell_Centroids_Out.csv	At the start of the model run, writes out the coordinates of each cell’s centroid, in the coordinate system of the cell geometry file (Section 5.1).
writeDepthReport	(reachname) _Cell_Flow_Depth_Test.rpt	Produces one file for each habitat reach. One line is written at the start of each day, reporting the flow, and depth in each cell.

Control variable for Model.Setup (boolean)	Report file name	Description of report
writeVelocityReport	(reachname)_ Cell_Flow_Velocity_Test.rpt	Like the depth report, but cell velocity is reported.
writeDepthVelocityReport	(reachname)_ CellDepthAreaVelocity.rpt	Produces one file for each reach. On each day, one line is written for each habitat cell with depth above zero; cell area, depth, and velocity are output.
writeMoveReport	MoveTest.rpt	Writes one line when each trout selects the habitat cell it will occupy, each day. Reports the data (about the selected cell and the trout) used to select this best cell. Also reports intermediate calculations in the trout's movement decision (e.g., its respiration rate, net energy intake). Note: The output is first produced when the fish are initialized on the first simulated day, when fish are not yet sorted in size order.
writeReadyToSpawnReport	Ready_To_Spawn.rpt	Produces one line per day for each female trout. Reports whether the female decided it was ready to spawn and the variables used to make the decision.
writeSpawnCellReport	Spawn_Cell.rpt	Produces one line of output each time a female trout spawns and builds a redd. For each potential spawning cell, reports the cell depth, velocity, and fraction spawning gravel; and the calculated variables used to rate spawning cells: depth and velocity suitability, overall spawning quality.
writeReddMortReport	Redd_Mortality.rpt	Provides a daily egg mortality report for each redd. When each redd is empty, it writes its report to the end of this file. The report includes the ReddID code, the initial number of eggs, and (on separate lines) the number of eggs dying of each mortality source on each day of the redd's existence.

Control variable for Model.Setup (boolean)	Report file name	Description of report
writeReddSurvReport	ReddSurvivalTest.rpt	Produces a report on egg survival for a redd on the day when the redd has no more live eggs (due to mortality and emergence). Reports the redd's species, location (reach and cell numbers), and initial number of eggs. Then, for each day of the redd's existence, a line reports the temperature, flow, depth, and number of eggs dying of each mortality source.
writeCellFishReport	(reachname) _CellFishInfo.rpt	Produces one file for each reach. On each day, writes one line for each habitat cell. Reports cell area, depth, velocity, distance to hiding cover, fraction with velocity shelter, and statistics on the fish in the cell*.
writeIndividualFishReport	IndividualFishReport.csv	Reports the state (location, age, length, weight, condition) of each live fish. One line is produced per fish.

*This file is produced using the EcoSwarm BreakoutReporter, and is very similar to the live fish output file (Section 8.1). The fish statistics provided in this file normally include the number of fish in the cell, broken out by species and age class. However, these statistics are easily modified as described in Section 8.1, by modifying the BreakoutReporter code in method buildCellFishInfoReporter in file HabitatSpace.m.

10. Experiment Manager

All users of *inSTREAM* need to at least be aware of the Experiment Manager because it has powerful features that are often extremely useful, but it can also act like a nightmarish bug if ignored. These features overwrite values provided in setup and parameter files, causing the model's behavior to be inexplicable if one forgets to pay attention to the Experiment Manager. But any serious user of *inSTREAM* will quickly learn to depend on the Experiment Manager to conduct simulation experiments quickly, easily, and reliably.

10.1. What the Experiment Manager Does

The purpose of the Experiment Manager is to allow users to set up and execute simulation experiments that use multiple model runs with different inputs, without having to modify either the software or the parameter files. With a few simple statements in its setup file, a complex experiment can be set up to run automatically. The experiments can include both scenarios and replicates. A scenario is a single set of inputs and parameter values; different scenarios are defined by specifying the inputs or parameters that differ among them. Replicates are repeated runs of the same scenario, with only the pseudorandom numbers (which primarily affect mortality) differing among replicates.

The Experiment Manager has two functions. The first is to set up and execute the number of simulations specified in its setup file. For example, the user may set up the Experiment Manager to define 10 scenarios (perhaps five different values for some parameter, for each of two flow input files) and three replicates. The Experiment Manager would determine that 30 model runs are needed and start one run after another. The second function is to modify the parameters for each of the model runs to implement the scenarios. This function occurs at the start of a model run: the Experiment Manager stops the model after parameter values have been read in, then overwrites the value of any parameters that are specified in its setup file. For replicates, the Experiment Manager simply changes the random number seed (the seed value provided in Model.Setup is multiplied by the replicate number). The Experiment Manager then re-starts the model and has no more effect until the next run starts. (To be precise: the TroutModelSwarm method instantiateObjects is executed; then the Experiment Manager is executed and modifies parameter values; then the TroutModelSwarm method buildObjects is executed and the simulation proceeds.)

The Experiment Manager's scenarios almost always involve modifying the value of variables that are in setup or parameter files. The Experiment Manager is generally not useful for directly modifying variables that are in input files (flows, temperatures, cell characteristics, etc.); however, it works very nicely to define such scenarios by creating different input files and using the Experiment Manager to control which input file is used.

The following sections describe the setup file that controls the Experiment Manager, and provide many examples that can be followed to set up experiments.

10.2. General Procedure for Setting Up Experiments

The general procedure for using the Experiment Manager is to (1) specify the scenarios and replicates in the Experiment.Setup file, (2) run *inSTREAM* in batch (non-graphics) mode, and (3) examine and analyze the results. Results are usually written to one file, with output labeled by

scenario and replicate number. It is always good to archive a copy of the Experiment.Setup file with the output files from an experiment, to document exactly what scenarios were executed.

10.2.1. Experiment.Setup format

The Experiment.Setup file controls the Experiment Manager. This file is always read by *inSTREAM*, so it must be configured correctly even if the model is to be run with no automated experiments. This format of this setup file is described here and illustrated via several examples in Section 10.3 (see also Figure 20). Experiment.Setup contains:

- Comments, which can be included anywhere as lines that start 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. These values must each be at least one. There are no built-in limits to how many scenarios or replicates can be used.
- A blank line.
- Two lines that provide the variable name and class to which the code sends the current scenario count, during model execution. These lines should not be changed.
- A blank line.
- Two lines that provide the variable name and class to which the code sends the current replicate count. These lines should also not be changed.
- A blank line.

Following these initial blocks of text, the file contains zero or more additional blocks, which each specify a model parameter to be varied among scenarios and the value it has for each scenario. (At least one such block must be specified if the number of scenarios is greater than one.) There is no limit to how many of these blocks can be specified, or how many parameters can be controlled by the Experiment Manager. These blocks contain the following lines; blocks are separated by a blank line.

- 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. (Class, instance, and variable names are further explained in Section 10.2.2.)
- The word `InstanceName` followed by the name of the instance of the class for which the parameter is to be varied. If the word `NONE` is provided for InstanceName, then the parameter is varied for all instances of the class.

- 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 in Section 10.2.3.
    - The word `Value` followed by the parameter's value for the first scenario. This line is repeated for each scenario: there must be one value provided for each scenario, even if the value is the same for some scenarios.

```
Experiment setup file -
Created Feb 21 2005
For demo example

numberOfScenarios    3
numberOfReplicates   5

sendScenarioCountToParam: scenario
inClass:                  TroutModelSwarm

sendReplicateCountToParam: replicate
inClass:                  TroutModelSwarm

ClassName      FishParams
InstanceName   Brown
ParamName      fishFitnessHorizon
ValueType      double
Value          60.0
Value          90.0
Value          120.0
```

Figure 20. Example Experiment.Setup file. This setup causes 15 model runs. Three scenarios differ in giving simulated brown trout values of their parameter fishFitnessHorizon values of 60, 90, and 120. Each of these scenarios is run 5 times with different random numbers.

10.2.2. Class and instance names for typical experiments

The Experiment Manager is highly flexible, allowing users access to any instance variable of any class in *inSTREAM*. However, using it successfully for unusual experiments requires detailed knowledge of the software; some inputs cannot be usefully manipulated because they have already been used before the Experiment Manager executes (e.g., the hydraulics data files) or because their values are overwritten after the Experiment Manager executes (e.g., reach flow; cell depth and velocity). Table 5 describes the usual applications of the Experiment Manager that can be made with confidence.

Table 5. Class and instance names for parameters commonly used in the Experiment Manager.

Parameters	ClassName	InstanceName
Model setup parameters (any parameters in Model.Setup)	TroutModelSwarm	NONE
File names for cell data, flow, temperature, and turbidity input (parameters *flowFile*, *temperatureFile*, *turbidityFile* in the reach setup file)	HabitatSpace	The reach name specified in Reach.Setup (or NONE if only one reach is simulated)
Habitat parameters (any parameter in a habitat parameter file)	HabitatSpace	The reach name specified in Reach.Setup (or NONE if the Experiment Manager is to alter the parameter for all habitat reaches)
Trout and redd parameters (any parameter in a trout parameter file)*	FishParams	The species name specified in Species.Setup (or NONE if the Experiment Manager is to alter the parameter for all species)

*Trout parameter values are not stored in the model trout themselves, but in a separate object of class "FishParams" for each species.

Input that cannot be manipulated by the Experiment Manager include parameters in the Observer.Setup and Species.Setup files and the hydraulics data file names in Reach.Setup.

10.2.3. Valid parameter value types

The Experiment.Setup file must provide a "ValueType" for each parameter to be manipulated. Valid value types are defined in Table 6. The value type depends on the parameter itself. If users are not sure what type a parameter is, they should find the parameter in the header (.h) file for the parameter's class. For example, the parameter controlled by the setup file in Figure 20 is *fishFitnessHorizon*, the number of days over which a fish evaluates its habitat selection decision. This parameter could potentially be either an integer or floating point (real) number. Searching the file FishParams.h for *fishFitnessHorizon* finds the declaration "double fishFitnessHorizon", indicating that this parameter is a double-precision floating point number, so ValueType should be "double".

Table 6. Value types for the Experiment.Setup file.

Value type*	Definition
`BOOL`	A boolean variable, with a value of either `YES` or `NO` (all upper case)**
`date`	A date variable in MM/DD/YYYY format
`day`	A day-of-the-year variable in MM/DD format
`double`	A double-precision floating point variable (any number that is not an integer)
`filename`	The name of an input file (a character string)
`int`	An integer variable

*The "ValueType" field in an Experiment.Setup file must exactly match one of the values in this column, including upper/lower case.

**Boolean variables can alternatively have values of `0` or `1`, as in other setup files.

10.2.4. Using instance names

The InstanceName field in Experiment.Setup allows the user to manipulate parameters for one instance of a class: for one of several trout species, or for one of several habitat reaches (see Table 5). InstanceName is set to `NONE` if there is only one instance per class, or if the same parameter change is to be made for all instances.

Separate parameter blocks can be used in the Experiment.Setup file to change the same parameter different ways for different species or habitat reaches. The following examples illustrate how the Experiment Manager uses instance names. The examples assume a version of *inSTREAM* with three trout species named Rainbow, Brown, and Cutthroat; and that only one scenario is generated.

Example 1 changes the value of parameter *fishEnergyDensity* to 4555, but only for rainbow trout; other species retain the value in their parameter files:

```
ClassName      FishParams
InstanceName  Rainbow
ParamName    fishEnergyDensity
ValueType      double
Value              4555
```

Example 2 changes the value of *fishEnergyDensity* for all species:

```
ClassName      FishParams
InstanceName  NONE
ParamName    fishEnergyDensity
ValueType      double
Value              4555
```

Example 3 changes the value of *fishEnergyDensity* to 4555 for rainbow trout and to 5000 for cutthroat; brown trout are unaffected:

```
ClassName      FishParams
InstanceName   Rainbow
ParamName      fishEnergyDensity
ValueType      double
Value          4555

ClassName      FishParams
InstanceName   Cutthroat
ParamName      fishEnergyDensity
ValueType      double
Value          5000
```

Example 4 changes *fishEnergyDensity* to 4555 for rainbow and to 5000 for all other species. The order in which these two blocks appear does not matter—changing parameter values for a specific instance always overrides a general change (via `InstanceName NONE`) to all instances.

```
ClassName      FishParams
InstanceName  Rainbow
ParamName    fishEnergyDensity
ValueType      double
Value              4555

ClassName      FishParams
InstanceName  NONE
ParamName    fishEnergyDensity
ValueType      double
Value              5000
```

Example 5 changes *fishEnergyDensity* to 5000. It does not cause an error even though *fishEnergyDensity* is included twice—if the same parameter is included several times in Experiment.Setup, in the same way, the last value is used.

```
ClassName      FishParams
InstanceName  Rainbow
ParamName    fishEnergyDensity
ValueType      double
Value              4555

ClassName      FishParams
InstanceName  Rainbow
ParamName    fishEnergyDensity
ValueType      double
Value              5000
```

10.2.5. Controlling where output goes

Normally the Experiment Manager is set up so output from all scenarios and replicates are sent to the same output files, which are automatically appended for each model run (even if the parameter *appendFiles* is set to `0` in Model.Setup). The standard output files include the scenario and replicate number for all output.

An alternative is to include unique names for output files for each scenario in the Experiment.Setup file. This would write results from each scenario to a different output file (but multiple replicates would still be in the same file). (This capability has not been tested and no examples are provided.)

10.2.6. Checking the Experiment Manager

The Experiment Manager includes several kinds of error checking; execution stops with an error statement if:

- The number of scenarios or replicates is set to zero;
- A ValueType field has an invalid value, or its value does not match that of the parameter;
- A ClassName or InstanceName field has an invalid value;
- The named parameter does not exist in the specified class; or
- The number of values provided for any parameter is not exactly equal to the number of scenarios.

There are several ways to verify that the Experiment Manager produced the intended parameter values. Manipulations of input data (e.g., the flow or temperature input file) can be checked by examining the habitat output files (Section 8.2).

Habitat parameters can be checked by running *inSTREAM* in graphics mode and using the HabitatSpace probe display (Section 7.5.1). (Remember that multiple scenarios can be run in graphics mode by clicking on “Start” on the main control panel after each run finishes; and that all habitat variables can be viewed by right-clicking on the box labeled HabitatSpace in the top left corner of the probe display; Figure 14.)

Trout parameters can be checked by turning on an optional output that prints out trout parameter values at the start of each model run (after they have been manipulated by the Experiment Manager). A separate output file is created for each species; these are named SpeciesXParamCheck.out, where X is the name of the trout species. There are two ways to turn this optional output on; one is to add this line to Model.Setup:

```
printFishParams   1
```

The second way is by controlling this parameter via the Experiment.Setup file, using a block such as this:

```
ClassName     TroutModelSwarm
InstanceName  NONE
ParamName     printFishParams
ValueType     BOOL
Value         NO
Value         YES
```

This example prints out fish parameters only at the start of the second scenario. Note that this output is overwritten each model run, so it only reflects the last scenario started.

10.3. Example Experiment.Setup Files

The Experiment Manager is fairly complicated, so a number of examples are provided here. Most users should be able to design the experiments they need by modifying these Experiment.Setup files.

10.3.1. No experiment

When users want to run a single model run, with no parameters altered by the Experiment Manager, the following Experiment.Setup file can be used.

```
Experiment setup file -
Created Feb 21 2005
Example for de-activated Experiment Manager

numberOfScenarios    1
numberOfReplicates   1

sendScenarioCountToParam: scenario
inClass:                  TroutModelSwarm

sendReplicateCountToParam: replicate
inClass:                   TroutModelSwarm
```

10.3.2. Replicate simulations

Users often simply want to run one scenario several times as replicates. Replication is useful just to understand how stochastic model results are. This Experiment.Setup will run the selected

number of replicates (five, in this example) and append results from each model run to the output files.

```
Experiment setup file -
Created Feb 21 2005
Example for replication of one scenario

numberOfScenarios    1
numberOfReplicates   5

sendScenarioCountToParam: scenario
inClass:                  TroutModelSwarm

sendReplicateCountToParam: replicate
inClass:                  TroutModelSwarm
```

10.3.3. Parameter sweep for trout

One of the most common kinds of experiment is a “parameter sweep”: an experiment in which one parameter is varied over a wide range. The example uses the trout parameter *mortFishTerrPredMin*, and applies the sweep to all trout species in the model. Therefore, this example could be used to evaluate how trout populations respond to increasing levels of terrestrial predation risk (decreasing survival probability).

```
Experiment setup file -
Created Feb 21 2005
Example trout parameter sweep

numberOfScenarios    6
numberOfReplicates   1

sendScenarioCountToParam: scenario
inClass:                  TroutModelSwarm

sendReplicateCountToParam: replicate
inClass:                  TroutModelSwarm

ClassName      FishParams
InstanceName   NONE
ParamName      mortFishTerrPredMin
ValueType      double
Value          0.950
Value          0.961
Value          0.972
Value          0.983
Value          0.994
Value          1.0
```

10.3.4. Habitat parameter sweep

This parameter sweep varies a habitat variable: the drift food concentration in one habitat reach ("MainstemUpperReach"). This example illustrates an experiment to see how trout populations respond to food availability in one of several reaches. This example also illustrates that floating point (type "double") parameter values can be in scientific notation.

```
Experiment setup file -
Created Feb 21 2005
Example habitat parameter sweep

numberOfScenarios    5
numberOfReplicates   1

sendScenarioCountToParam: scenario
inClass:                  TroutModelSwarm

sendReplicateCountToParam: replicate
inClass:                  TroutModelSwarm

ClassName      HabitatSpace
InstanceName   UpperMainstem
ParamName      habDriftConc
ValueType      double
Value          5.0E-11
Value          7.0E-11
Value          9.0E-11
Value          1.10E-10
Value          1.30E-10
```

10.3.5. Multiple parameter sweep

This experiment explores interactions among variables: what happens if two parameters are varied such that all combinations are simulated? This example varies two parameters (for fish vs. terrestrial predation risk), with three values of each; but the same approach can be used with more parameters and more values.

```
Experiment setup file -
Created Feb 21 2005
Example multiple parameter sweep

numberOfScenarios    9
numberOfReplicates   1

sendScenarioCountToParam: scenario
inClass:                  TroutModelSwarm

sendReplicateCountToParam: replicate
inClass:                  TroutModelSwarm

ClassName      FishParams
InstanceName   NONE
ParamName      mortFishAqPredMin
ValueType      double
```

```
Value          0.95
Value          0.95
Value          0.95
Value          0.97
Value          0.97
Value          0.97
Value          0.99
Value          0.99
Value          0.99

ClassName      FishParams
InstanceName   NONE
ParamName      mortFishTerrPredMin
ValueType      double
Value          0.95
Value          0.97
Value          0.99
Value          0.95
Value          0.97
Value          0.99
Value          0.95
Value          0.97
Value          0.99
```

Note that the user must manually insert the correct values for each parameter for each scenario; the Experiment Manager does not automatically create all combinations of several different parameters. (A spreadsheet is useful for creating more complex Experiment.Setup files such as this.)

10.3.6. Alternative daily input files

A primary application of *inSTREAM* is to compare alternative *inSTREAM* flow and temperature scenarios. This example illustrates how three different streamflow regimes can be contrasted, by creating three alternative flow input files and using the Experiment Manager to generate replicate simulations of each. (The flow input files could, for example, be generated by a reservoir model simulating alternative reservoir operating rules.) This example applies the same flow input to all habitat reaches, including when only one reach is simulated.

Note that while the input files names for flow, temperature, and turbidity are provided in the Reach.Setup file (Section 3.3), they are passed to the habitat reach objects (class HabitatSpace) before the Experiment Manager is activated. Therefore, the Experiment Manager accesses these file names in HabitatSpace. The parameter names for these input files are the same in HabitatSpace as they are in Reach.Setup (*flowFile*, *temperatureFile*, *turbidityFile*).

```
Experiment setup file - Comparison of alternative flow scenarios
Created Feb 21 2005
Three flow scenarios, five replicates of each

numberOfScenarios    3
numberOfReplicates   5

sendScenarioCountToParam: scenario
```

```
inClass:                  TroutModelSwarm

sendReplicateCountToParam: replicate
inClass:                  TroutModelSwarm

ClassName      HabitatSpace
InstanceName   NONE
ParamName      flowFile
ValueType      filename
Value          FlowScenario1.Data
Value          FlowScenario2.Data
Value          FlowScenario3.Data
```

10.3.7. Multiple simulation years

This example shows how to run the model one year at a time, but with separate runs for several different years. The Model.Setup file includes the parameters controlling when the model starts and stops. In this example, five replicates of three year scenarios are run.

This example is also easily modified to manipulate other parameters in Model.Setup.

```
Experiment setup file - for alternative years
Created Feb 21 2005
Example for changing model setup parameters

numberOfScenarios    3
numberOfReplicates   5

sendScenarioCountToParam: scenario
inClass:                  TroutModelSwarm

sendReplicateCountToParam: replicate
inClass:                  TroutModelSwarm

ClassName    TroutModelSwarm
InstanceName   NONE
ParamName    runStartDate
ValueType    date
Value        10/1/1999
Value        10/1/2000
Value        10/1/2001

ClassName    TroutModelSwarm
InstanceName   NONE
ParamName    runEndDate
ValueType    date
Value        9/30/2000
Value        9/30/2001
Value        9/30/2002
```

10.3.8. Alternative cell data files

Different cell data files might be used to simulate different availabilities of hiding or feeding cover or spawning gravel. The cell data file is a variable of the HabitatSpace class, and the instance name is the name of the habitat reach as defined in Reach.Setup.

```
Experiment setup file - for alternative cell data files
Created Apr 20, 2005
Example

numberOfScenarios    2
numberOfReplicates   5

sendScenarioCountToParam: scenario
inClass:                  TroutModelSwarm

sendReplicateCountToParam: replicate
inClass:                  TroutModelSwarm

ClassName      HabitatSpace
InstanceName   MiddleReachBearCreek
ParamName      cellDataFile
ValueType      filename
Value          HiCoverCell.Data
Value         LoCoverCell.Data
```

10.3.9. Year shuffler scenarios

The year shuffler facility in *inSTREAM* (described in the model description document) can generate scenarios that differ by having years of input data shuffled. A separate random number generator is used to shuffle years, allowing year randomization to be controlled separately from other stochastic processes. The following example shows how to generate five scenarios that differ only in the sequence in which years of input occur. For this experiment to work, year shuffling must be turned on (`shuffleYears` set to 1) in the model setup file (Section 3.4).

```
Experiment setup file -
For year shuffling scenarios
SFR 5/3/05

numberOfScenarios    5
numberOfReplicates   1

sendScenarioCountToParam: scenario
inClass:                  TroutModelSwarm

sendReplicateCountToParam: replicate
inClass:                  TroutModelSwarm

ClassName    TroutModelSwarm
InstanceName      NONE
ParamName    shuffleYearSeed
ValueType    int
```

```
Value        7377
Value        7378
Value        7379
Value        7370
Value        7371
```

10.3.10. Year shuffling as replication

The year shuffler can be used as an alternative way to replicate scenarios: a scenario can be run multiple times with different random re-ordering of the input data to see how robust its results are to the sequence of input years. This example shows how to generate five such year shuffler replicates of three scenarios. The three scenarios are alternative daily flow files that each represent a minimum flow requirement at a diversion dam. As in the previous example, year shuffling must be turned on in the model setup file.

The year shuffler replicates must be treated by the Experiment Manager as separate scenarios because they change the year shuffler random number seed, not the seed used for all other stochastic processes.

```
Experiment setup file -
For instream flow experiments
SFR 5/3/05

numberOfScenarios    15
numberOfReplicates   1

sendScenarioCountToParam: scenario
inClass:                  TroutModelSwarm

sendReplicateCountToParam: replicate
inClass:                  TroutModelSwarm

ClassName    HabitatSpace
InstanceName      NONE
ParamName    flowFile
ValueType    filename
Value        Divers01Flow.Data
Value        Divers01Flow.Data
Value        Divers01Flow.Data
Value        Divers01Flow.Data
Value        Divers01Flow.Data
Value        Divers02Flow.Data
Value        Divers02Flow.Data
Value        Divers02Flow.Data
Value        Divers02Flow.Data
Value        Divers02Flow.Data
Value        Divers03Flow.Data
Value        Divers03Flow.Data
Value        Divers03Flow.Data
Value        Divers03Flow.Data
Value        Divers03Flow.Data
```

```
ClassName    TroutModelSwarm
InstanceName      NONE
ParamName    shuffleYearSeed
ValueType    int
Value        7377
Value        7378
Value        7379
Value        7370
Value        7371
Value        7377
Value        7378
Value        7379
Value        7370
Value        7371
Value        7377
Value        7378
Value        7379
Value        7370
Value        7371
```

11. Trouble-shooting Guide

Table 7 lists the symptoms and solutions for common problems encountered in trying to install and run *inSTREAM*. Additional help for Swarm models in general is available from the Frequently Asked Questions resources on the Swarm web site, www.swarm.org.

Table 7. Trouble-shooting guide.

Symptom	**Potential Cause and Solutions**

When I try to compile the model, the compiler says something like “Makefile: ... Makefile.appl: No such file or directory. make: *** No rule to make target ...”	You might just not be in the right directory, where the code files and Makefile (explained below) are.
	
	The environment variable “SWARMHOME” might not set correctly, most likely because you (1) forgot to set this environment variable, or (2) set it to the wrong directory, perhaps because Swarm was installed to a non-standard directory. (If you are working in Windows with the MinGW release of Swarm, you should have a Windows environment variable set to `/c/swarm`.
	
	If you cannot figure out how to set SWARMHOME correctly, do this: (1) Open the makefile in an editor. In the directory of source code for *inSTREAM* will be a file named “Makefile”, which contains the directions the MinGW compiler uses to compile the model. This is a plain text (ASCII) file that you can edit using Notepad or Wordpad. At the top of the makefile you should see several lines similar to:
	
	`ifeq ($(SWARMHOME),)`
	`SWARMHOME=/usr`
	`endif`
	
	(2) Comment those lines out by putting a “#” character in front of them, and add a new line saying exactly where SWARMHOME is. The result should be:
	
	`#ifeq ($(SWARMHOME),)`
	`#SWARMHOME=/usr`
	`#endif`
	`SWARMHOME=/c/swarm`
	
	where the last line points to `C:\swarm`, where the Swarm libraries must be located. Make sure you use forward slashes “/”. See www.swarm.org/index.php/Swarm_and_MinGW for more information on installing Swarm. See the following entry for a problem likely at this point.

Symptom	Potential Cause and Solutions
I edited the Makefile (or a code file or input file) using a Windows editor (e.g., Notepad, Wordpad, Word), and saved the change; but *inSTREAM* (or MinGW) ignores the change.	Check whether the editor saved the changes in a new file called "Makefile.txt" instead of in the original file "Makefile". These editors sometimes insist that all plain-text files should end in ".txt". You can overcome this insistence by putting the file name in quotation marks when telling the editor where to save the file.
When I open output files in Windows Notepad, the lines are all run together.	You opened a file created in Unix format. (This should not happen to Windows-only users.) Use the Linux "unix2dos" conversion facility to convert individual files to Windows format (in Linux, or MinGW, type "unix2dos *filename*"); or open the files using Wordpad, Word, or Excel, which can handle Unix format.
inSTREAM says it cannot find an input file that really is there.	Make sure the file name is completely correct, including upper vs. lower case. File names cannot include blanks. If you are in Linux, was the file created in Windows? If so, you must use the "dos2unix" utility to convert it to Unix file format; otherwise, *inSTREAM* will not be able to read it. (But if you accidentally use dos2unix on the executable file `instream.exe`, it will be ruined and you will have to recompile it.) See the following symptom if the file name is provided to *inSTREAM* in a setup file.
The model will not read a text parameter such as a file name.	See the previous symptom concerning file names. Make sure there are no blanks after the parameter value. Make sure the setup file containing the parameter is in the proper Unix or DOS file format.
When I try to start the model by typing `instream.exe` I only get the error "bash: instream.exe: command not found".	To start the model you must type `./instream.exe` —not `instream.exe`.

Symptom	Potential Cause and Solutions
I changed a parameter value in the trout (or habitat) parameter file, but the change has no effect.	Check the Experiment.Setup file to see if the parameter value is being controlled by the Experiment Manager.
I edited an input file in Excel (or another spreadsheet program) to make a small change, and now the model crashes (with or without giving a useful error statement). (Similar problems can result from software other than Excel.)	Open the file you edited with a **text editor** such as WordPad (**not** Excel) and inspect it carefully for problems that Excel caused. Are there double quotes are text or other values? If so, remove them. Did small numbers (e.g., 0.0001) get rounded off (e.g., to 0.0)? Did any dates or large numbers get saved as "######"? Did dates get changed in format from the required MM/DD/YYYY (e.g., from `10/01/2009` to `10/01/09`)?
After I edit an input file in a spreadsheet, the model runs for a while then stops; the error statement says that something is wrong with dates.	See the end of Section 5.5.
When I start *inSTREAM* up, it stops while initializing a model run with an error statement saying that a date was improperly formatted. But all the dates in my input files seem to be correct.	There have been mysterious problems with *inSTREAM*'s date/time management software, on some operating systems some of the time. If using Windows, first make sure all input files are in DOS file format. Contact the *inSTREAM* developers if the problem persists.
When I run a large *inSTREAM* experiment, the model runs successfully for a long time and then suddenly crashes with an error saying something about "xmalloc". Re-running it produces the same error in the same place.	Unfortunately, this is due to the limited amount of random-access memory that Windows can allocate to any one program and the way that RAM gets fragmented as the model runs. The only solutions are to run smaller jobs, or run the model in a 64-bit Linux version of the model.

Index

A

appendFiles 10, 34, 36, 45

B

BreakoutReporter 34, 38

C

cellGeomFile 8
cellHabVarsFile 8
cellHydraulicFile 8
`ClassName (Experiment.Setup)` ..40, 42, 45
color variables 3
complete probe display 32
CSV format 1, 16, 21, 34

D

dryCellColor 5

E

Excel 1, 34

F

file names 1
fileOutputFrequency 10, 35
fishMortalityFile 10
fishOutputFile 10, 35
flowFile 8, 42, 49

H

habDownstreamJunctionNumber 8
habParamFile 8
habUpstreamJunctionNumber 8
header lines 1

I

`InstanceName (Experiment.Setup)` 40, 42, 43

M

Makefile 22, 23, 54, 55
maxShadeDepth 5
maxShadeVelocity 5

N

numberOfSpecies 23

O

object loader 2

P

`ParamName (Experiment.Setup)` 41
popInitDate 10
printFishParams 46

R

randGenSeed 10
rasterColorVariable 4, 29
rasterResolution 4
reachName 8
reddOutputFile 10, 35
replicates 39, 40, 46
runEndDate 10
runStartDate 10

S

scenarios 39, 40
scientific notation 48
shuffleYearReplace 10
shuffleYears 10
shuffleYearSeed 10, 51, 53
siteLatitude 10
species name 6, 22

T

tagCellColor 5
tagFishColor 5
takeRasterPictures 4, 33
temperatureFile 8, 42, 49
TimeSeriesInputManager 20
turbidityFile 8, 42, 49

V

`Value (Experiment.Setup)` 41
`ValueType (Experiment.Setup)` ..41, 42, 45