

Notes on Implementing StupidModel in Objective-C Swarm

Steve Railsback and Steve Jackson

15th February 2006

1 Introduction

This document provides notes on implementing StupidModel in Objective-C Swarm. StupidModel is a series of template models designed to illustrate common modeling techniques and features. StupidModel (documented separately) includes 16 versions, each adding or modifying parts of the model.

Because StupidModel was implemented in Java Swarm first, this document only describes key differences between the Java and Objective-C implementations. Readers should see *Notes on Implementing StupidModel in Java Swarm* for a detailed description of using Swarm for this model.

Our implementations of StupidModel in Objective-C and Java Swarm are available separately. We used Swarm 2.2. These codes *should not be considered examples of good Swarm programming*; instead, they illustrate simple (not necessarily robust or efficient) ways to implement basic functions. The example codes available at www.swarm.org are better examples of good Swarm programming practices.

2 Key Differences Between Objective-C and Java Swarm Implementations

2.1 Separate interface and implementation files

In Objective-C, the statements declaring variables and methods, and importing other classes, are kept in an “interface” (.h) file, while the methods are coded in a separate “implementation” (.m) file. Hence, each class in the model has two files.

2.2 Import statements

Java requires a separate import statement for each library class your code uses. Objective-C requires only one import statement for each of Swarm’s libraries that you use, so there are far fewer import statements.

2.3 Message syntax

Objective-C's messaging syntax allows statements calling methods to be shorter and more English-like. For example, the Java code to schedule an action is:

```
modelSchedule.at$createAction (0, updateActions);
```

whereas the corresponding Objective-C statement is:

```
[modelSchedule at: 0 createAction: updateActions];
```

2.4 Selectors

In Objective-C, selectors are a built-in type for variables containing the name of a method. In contrast, Java Swarm requires creating each selector used as a new object. Hence, methods that use the names of other methods (especially, for building schedules and creating graphs) are simpler in Objective-C Swarm. For example, to tell the histogram what data to plot, the Java observer swarm uses this statement:

```
try {
    bugSizeHistogramC.setProbedSelector (new Selector
        (Class.forName ("stupidModel_8.StupidBug"), "getMySize", false));
} catch (NonUniqueMethodSignatureException e) {
    e.printStackTrace();
} catch (SignatureNotFoundException e) {
    e.printStackTrace();
} catch (ClassNotFoundException e) {
    e.printStackTrace();
}
```

The corresponding Objective-C statement is:

```
[bugSizeHistogram setProbedSelector: M(getMySize)];
```

2.5 Create and use phases of Swarm classes

Some Swarm classes—for example, EZBin and EZGraph used in StupidModel's observer swarm—are created in two phases: createBegin (a “factory method”) and createEnd (an instance method). “Create phase” methods are used to set up a new instance of the class before its createEnd method is called. Java Swarm requires some rather clumsy work-arounds for this two-phase approach; see the discussion of version 6 in *Notes on Implementing StupidModel in Java Swarm*. Hence, creating Swarm tools such as EZBin and EZGraph are simpler in Objective-C Swarm (see versions 6 and 13).

2.6 Subclassing

(AGENTS ARE SUBCLASSES OF SWARMOBJECT SO THEY CAN DO STUFF LIKE DROP, PROBES...SWARM CONTAINS MUCH MORE OPERATING SYSTEM STUFF)

2.7 Dropping objects

Objective-C does not do “garbage collection”, so users must be careful to drop objects when they are no longer used. (Otherwise, your computer’s memory will rapidly be used up.) Objects created from Swarm classes can be dropped by sending them a “drop” message. We make the StupidBug and HabitatCell classes as subclasses of SwarmObject, the Swarm class that provides the drop method; therefore, we can drop bugs and cells by also sending them a “drop” method. Swarm collections have a method “deleteAll”, which drops all objects in the collection. Examples of dropping objects in StupidModel include:

- In StupidModelSwarm method “addAndRemoveBugs” (versions 12 and later), the statement `[deadBugList deleteAll];` is used to drop all the bugs that have died.
- Whenever we use a “ListIndex” object to loop through a list (e.g., the list of neighbor cells in the “move” method of StupidBugs, in version 11 and later), that ListIndex object is dropped when we are done with it.
- ADD - DROP & SUPER DROP IN BUGS...NEED TO KEEP TRACK OF WHETHER BUG IS ALREADY DEAD (NEGLECTED IN JAVA)