Documentation Set for Swarm 2.2

Swarm Development Group

Published 17 December 2004
Release 2.1.1

The Swarm home page *(http://www.swarm.org)* provides a good introduction to Swarm (what it does and what it aims to be). On the other hand, the The Overview to Swarm aims to give you a more detailed overview of the sorts of things a user needs to do in order to get a simulation up and running in Swarm. The combination of these two documents should help you decide whether Swarm would be a useful tool in the context of your research.

Please direct all questions, bug reports, or suggestions for changes to the Swarm developers (see the Swarm home page *(http://www.swarm.org)* for current contact details).

Revision History

2000-03-28 *mgd*

Swarmdocs 2.1.1 frozen.

2000-03-01 *alex*

Swarmdocs 2.1 frozen.

2000-03-01 Makefile.am *alex*

(noinst_DATA): Remove entirely.

2000-02-15 setmeta.sgml *alex*

Remove swarm@santafe.edu e-mail address, instead refer reader to SDG homepage.

1999-04-06 set.sgml *alex*

Change PUBLIC identifier for DTD to "-//OASIS//DTD DocBook V3.1//EN". Remove JPEG notation class: now part of the 3.1 DTD.

1999-01-13 setmeta.sgml *alex*

Remove old LEGALNOTICE text. Refer to the newly-defined {doc,swarm}-legalnotice entities from global.ent.

1999-01-11 setmeta.sgml *alex*

Make email ULINK an EMAIL tag. Tidy text.

1999-01-04 setmeta.sgml *alex*

Remove <TITLE> element in ABSTRACT.

1998-10-14 set.tex.in *mgd*

Use top_dossrcdir instead of top_srcdir.

1998-10-09 set.tex.in *mgd*

Include tex/macros.tex.

1998-06-17 set.sgml *mgd*

Use refbook.ent instead of src.ent.

1998-06-15 Makefile.am *mgd*

Include $(top_srcdir)/Makefile.common.

1998-06-11 setmeta.sgml *alex*

Removed BIBLIOSET element - moved TITLE, CORPAUTHOR, bibliodata up to under the SETINFO element.

1998-06-11 Makefile.am *alex*

(GENERATED_SGML): Added for revhistory. (noinst_DATA): Added. (SGML, SGML_FILES): Separated out generated/non-gen files.

1998-06-11 setmeta.sgml *alex*

Replaced REVHISTORY with setrevhistory entity.

1998-06-11 set.sgml *alex*

Added reference to setrevhistory.sgml. Changed local notion class to JPEG.

1998-06-11 set.sgml *alex*

(entity): Included versions.ent, figs.ent and updated all entity references to be consistent with filenames.

1998-06-09 setmeta.sgml *alex*

Used global SFI entity 'corpauthor' in CORPAUTHOR SGML tag.

1998-06-08 set.sgml *mgd*

Use public identifiers (file renamed from set.sgml.in).

1998-05-23 set.sgml.in *mgd*

Make ID of set be "set".

1998-05-23 set.sgml.in, set.tex.in *mgd*

New files.

1998-05-23 Makefile.am *mgd*

New file.

1998-05-23 Makefile *mgd*

Remove.

1998-05-22 *mgd*

Begin revision log.

Table of Contents

Brief Overview of Swarm

Swarm Development Group

A Top-Down Introduction To Implementing an Experiment Using Swarm

This document attempts to explain the logical structure of a Swarm experiment application. Starting with a very general outline of an idealized experimental procedure, we successively increase the level of specification of each stage of this idealized structure until we arrive at details of an actual running Swarm application.

Along the way, we introduce and describe (very briefly) some of the tools currently available in Swarm to help users build experiments. The tools presented here are only suggestive: the Swarm library documentation and example applications will give a more detailed view of using specific components of Swarm

Revision History

2000-03-28 *mgd*

Swarmdocs 2.1.1 frozen.

2000-03-01 *alex*

Swarmdocs 2.1 frozen.

2000-03-01 Makefile.am *alex*

(noinst_DATA): Remove entirely.

1999-04-06 overbook.sgml *alex*

Change PUBLIC identifier for DTD to "-//OASIS//DTD DocBook V3.1//EN". Remove JPEG notation class: now part of the 3.1 DTD.

1999-02-17 over04.sgml *alex*

(PROGRAMLISTING: main): Change comment that says batchmode is indicated by swarmGUIMode being 1; batchmode is indicated by swarmGUIMode being 0.

1999-01-26 overmeta.sgml *alex*

Move bibliodata entity outside of BOOKBIBLIO - so legalnotice links work.

1999-01-13 overmeta.sgml *alex*

Include the newly-defined {doc,swarm}-legalnotice entities from global.ent. Move CORPAUTHOR inside BOOKBIBLIO.

1998-12-27 over01.sgml, over02.sgml, over03.sgml, over04.sgml, over05.sgml *alex*

Standardize use of IDs according to conventions. Add IDs for all SECT1 elements.

1998-10-14 overbook.tex.in *mgd*

Use top_dossrcdir instead of top_srcdir.

1998-10-09 over04.sgml *mgd*

Remove duplicated "Scheduling a Model Swarm" section.

1998-10-09 overbook.tex.in *mgd*

Include tex/macros.tex.

1998-06-17 overmeta.sgml *mgd*

Use overbookrevhistory.sgml instead of overrevhistory.sgml. Scale graphic to 100%.

1998-06-17 Makefile.am *mgd*

(GENERATED_SGML): Use $(NAME)revhistory.sgml instead of overrevhistory.sgml. (EXTRA_DIST): over.ent renamed to overbook.ent.

1998-06-15 overbook.sgml *mgd*

Add JPEG notation.

1998-06-15 Makefile.am *mgd*

Include $(top_srcdir)/Makefile.common.

1998-06-12 overbook.sgml *mgd*

Include versions entity.

1998-06-12 overmeta.sgml *mgd*

Scale graphic to 75%.

1998-06-11 over.ent *alex*

Made all SYSTEM identifiers PUBLIC (listed in catalog now).

1998-06-11 Makefile.am *alex*

(GENERATED_SGML): Added for revhistory. (noinst_DATA): Added. (SGML, SGML_FILES): Separated out generated/non-gen files.

1998-06-11 over.ent *alex*

Created overrevhistory.sgml entity.

1998-06-11 overmeta.sgml *alex*

Replaced hardcoded revhistory with the new entity overrevhistory.sgml.

1998-06-09 overmeta.sgml *alex*

Changed corpauthor to be inserted from Global public identifiers.

1998-06-08 overbook.sgml *mgd*

Use public identifiers.

1998-06-04 over02.sgml, over03.sgml, over04.sgml *alex*

Fixed inappropriately positioned PARA tags.

1998-05-23 Makefile.am *mgd*

New file.

1998-05-23 Makefile *mgd*

Remove.

1998-05-23 overbook.tex.in *mgd*

New file.

1998-05-23 overbook.sgml *mgd*

Set ID of book to "overbook".

1998-05-22 *mgd*

Begin revision log.

Table of Contents

Chapter 1. Mag 1x: Experimental Procedure in a Computer

At the highest level of abstraction (= the lowest level of magnification), most experiments look like this:

1. Set up the physical system to be studied.
2. Set up and calibrate the instrumentation
3. Run the experimental system and record the outputs of the instrumentation.
4. Analyze results.
5.
   - Change experimental and instrumental setup
   - Go to 3.
6. Publish paper -> tenure -> fame -> etc.....

The important part of step 6) is that the published paper includes enough detail about the experimental setup and how it was run so that other labs with access to the same equipment can recreate the experiment and test the repeatability of the results. This is hardly ever done (or even possible) in the context of experiments run in computers, and the crucial process of independent verification via replication of results is almost unheard of in computer simulation. One goal of Swarm is to bring simulation writing up to a higher level of expression, writing applications with reference to a standard set of simulation tools.

First, let's look at what happens when we port the above stages into the world of a computer. In a computer, you don't just drag the pieces of your experiment in from the outside world and hook them up. You have to create a world with space and time, a bunch of objects in that world (stuff to study and stuff to look at it with), schedules of events over those objects, all sorts of computer widgetry to interact with that artificial world and to manage multiple experimental runs and the data that they generate, and so forth. In other words, in a computer, one usually has to first *create* from scratch all of the bits and pieces of the experimental setup - the virtual equivalent of beakers, bunsen burners, microscopes etc.

Perhaps the most important difference between an experiment in the "real" world and an experiment inside of a computer is the nature of time. In the real world, everything in one's experimental setup is moved forward in time via a very concurrency courtesy of the laws of physics. In a computer experiment, however, the experimenter has to explicitly move every object in his/her artificial universe forward in time, making sure that everything remains within some well-understood state of synchronization. Many fundamental problems in computer science have arisen in the course of trying to understand how to control and use concurrency. Furthermore, most people who implement computer simulations aren't even aware of the subtle, but quite-possibly dominating, impacts of assumptions that they aren't even aware that they are making about concurrency in their model when they code it up and run it.

Therefore, a very important aspect of setting up an experiment in a computer is how one weaves the multiple threads of time that must be woven together coherently in order to produce reliable, repeatable results. Much of our work on Swarm has been devoted to not only making the task of managing concurrency manageable, but towards mechanisms to make people aware that they are always making implicit assumptions about how multiple threads of time are interacting with one another in their

experimental setups. Swarm forces experimenters to make their concurrency assumptions explicit, so that others can reproduce their results by implementing the same assumptions about the flow of time.

Chapter 2. Mag 2x: Basis of Swarm Computation

Here is a first approximation to embedding the above outline of an experimental procedure in Swarm code:Swarm is implemented in the Object-Oriented Programming language Objective-C. Computation in a Swarm application takes place by having objects send messages to each other. The basic message syntax is:

```
[targetObject message Arg1: var1 Arg2: var2]
```

Where "targetObject" is the recipient of the message, "messageArg1:Arg2:" is the message to send to that object, and "var1", "var2", etc, are arguments to pass along with the message. Objective C's messages are keyword/value oriented, which is why the message name "messageArg1:Arg2:" is interspersed with the arguments.

The whole idea of Swarm is to provide an execution context within which a large number of objects can "live their lives" and interact with one another in a distributed, concurrent manner. Furthermore, we wish to insulate the user from having to master all of the highly baroque computer-science wizardry usually required to implement such massively distributed systems of autonomous agents reliably and robustly.

In the context of the Swarm simulation system, the generic outline of an experimental procedure takes the following form:

1. Create an artificial universe replete with space, time, and objects that can be located, within reason, to certain "points" in the overall structure of space and time within the universe., and allow these objects to determine their own behavior according to their own rules and internal state in concert with sampling the state of the world, usually only sparsely.
2. Create a number of objects which will serve to observe, record, and analyze data produced by the behavior of the objects in the artificial universe implemented in step 1).
3. Run the universe, moving both the simulation and observation objects forward in time under some explicit model of concurrency.
4. Interact with the experiment via the data produced by the instrumentation objects to perform a series of controlled experimental runs of the system.
5. Depending on what is observed in stage 4), alter the experimental or instrumental "apparatus" and go back to 3).
6. Publish paper *including* detailed specification of the experimental set-up so that others can recreate your experiment and verify your results.

Chapter 3. Mag 3x: Swarm Structures

Swarm applications are structured around the concept of the Swarm. Swarms are the basic building blocks of Swarm simulations: a Swarm is a combination of a collection of objects and a schedule of activity over those objects. The collection are like the matter of the Swarm and the schedule is like the arrow of time moving the objects forward.

3.1. Model Swarms

In our current demos, Swarm applications contain two swarms. At the core is the *model swarm*, the Swarm that encapsulates the simulated model. Everything in the model swarm corresponds to objects in the world being modeled. For instance, in Heatbugs the HeatbugModelSwarm contains a collection of Heatbug agents, a HeatSpace to represent a physical property of the world, antion of Heatbug agents, a HeatSpace to represent a physical property of the world, and a Grid2d to store agent position.

In addition to the object collection, the model swarm also contains a schedule of activity on the model. The schedule defines the effect of passing time on the model. For the simple heatbugs schedule, the execution is simply to update the HeatSpace (diffusing heat across the world) and then telling each Heatbug agent to move itself.

Model swarms consist of a set of inputs and outputs. The inputs to the HeatbugModelSwarm are model parameters: things like the size of the world, the number of HeatBugs, and the diffusion rate of heat. The outputs of the HeatbugModelSwarm are the observables of the model: the individual Heatbugs, the distribution of heat across the world, etc.

3.2. Observer Swarms

The model swarm alone defines the simulated world. But an experiment does not just consist of the objects being experimented upon, it also includes the experimental apparatus used for observation and measurements. In Swarm computer simulations, those observation objects are placed in an *observer swarm*.

The most important object in an observer swarm is the model swarm that is being studied. The model swarm is one component of the observer, kind of like a little world in a petri dish on the lab bench. Other observer objects can then input data into the model swarm (setting simulation parameters, for instance) and read data out of the model swarm (collecting statistics of the behavior of agents).

Just as in setting up a model swarm, an observer swarm has a collection of objects (the instrumentation), a schedule of activity, and a set of inputs and outputs. The activity of the observer schedule is to drive data collection - read this number out of the model, draw it on a graph. The inputs to the observer swarm are configurations of the observer tools: what sorts of graphs to generate, for instance. The outputs are the observations.

When running in graphics mode, the observer swarm objects are largely used to mediate user interface. For instance, in Heatbugs the HeatbugObserverSwarm creates Raster widgets, EZGraphs, and Probes. All of these objects are connected into the HeatbugModel swarm to read data, and to graphical interface objects so the human sitting in front of the computer can observe the world.

Interactive, graphical experimentation with models is useful for coming up with intuitions. But for serious experimentation it is necessary to collect statistics, which means doing many runs and storing data for analysis. As an alternative to a graphical observer swarm, you can also create *batch swarms*, observer swarms that are intended to be run without any interaction at all. Instead of putting up fancy graphics, batch swarms read data from files to control the model and write the data out to other files for analysis. The key here is that the model swarm used in the batch swarm is the exact same model as that used in a graphical observer swarm: the only difference is what tools the observer (batch or graphical) connects to the model.

3.3. Summary

Multiple Swarms are used to create an experimental apparatus and control it. The use of multiple Swarms is not restricted only to this use, though: in particular, a model Swarm could itself contain its own subswarms, building a hierarchical simulation. In future Swarm development, we intend to use the power of the multiple Swarm modeling approach to build complicated and flexible models.

Chapter 4. Mag 4x:Sketch of Code

Now that we have multiple Swarms and an experimental apparatus, it's time to learn how to use the objects themselves inside an application. Some examples are provided here: to understand this better, it will be necessary to read through example applications and the library documentation. These examples come from the Heatbugs application.

4.1. Building a Model Swarm

The key component of a simulation is the model Swarm. Here is the definition of a HeatbugModelSwarm, from HeatbugModelSwarm.h

```
@interface HeatbugModelSwarm : Swarm {
 int numBugs; // simulation parameters
 double evaporationRate;
 double diffuseConstant;
 int worldXSize, worldYSize;
 int minIdealTemp, maxIdealTemp;
 int minOutputHeat, maxOutputHeat;
 double randomMoveProbability;

 id modelActions; // scheduling data structures
 id modelSchedule;

 id heatbugList; // list of all the heatbugs
 Grid2d * world; // objects representing
 HeatSpace * heat; // the world
}

-getHeatbugList; // access methods into the
-(Grid2d *) getWorld; // model swarm. These methods
-(HeatSpace *) getHeat; // allow the model swarm to be observed.

+createBegin: aZone; // extra methods you
-createEnd; // provide for Swarms
-buildObjects;
-buildActions;
-activateIn: swarmContext;
```

The first section of code says that a HeatbugModelSwarm is a kind of Swarm. HeatbugModelSwarm inherits a lot of behavior from generic Swarm, but also adds new variables and methods.

The new variables are enclosed in the braces in the definition of HeatbugModelSwarm. They are split into three general classes of things: simulation parameters, schedule data structures, and objects in the world. This is a typical sort of model swarm.

Finally, a HeatbugModelSwarm defines new methods. The first few methods are used to allow the model to be observed: a HeatbugModelSwarm will give out its list of Heatbugs, for instance, or its HeatSpace. Observers use these methods to monitor the model.

In addition to the observation methods, there are several Swarm-specific methods for the building of Swarms. These are fairly stereotyped. The `createBegin` and `createEnd` messages are used to create the Swarm object itself. `buildObjects` builds the model objects, and `buildActions` builds the model schedule - more on these later. Finally, `activateIn` arranges for the execution machinery to execute the Swarm itself.

4.2. Defining an Agent

The agents are typically the real focus of a modeling effort. Most of the work in a simulation comes in defining the agent behavior so that the computer agents resemble the real world phenomena you are trying to create.

In the case of Heatbugs, agents are pretty simple. Here is their definition, from Heatbug.h:

```
@interface Heatbug: SwarmObject {
 double unhappiness; // my current unhappiness
 int x, y; // my spatial coordinates
 HeatValue idealTemperature; // my ideal temperature
 HeatValue outputHeat; // how much heat I put out
 float randomMoveProbability; // chance of moving randomly

 Grid2d * world; // the world I live in
 int worldXSize, worldYSize; // how big that world is
 HeatSpace * heat; // the heat for the world
 Color bugColor; // my colour (display)
}

-setWorld: (Grid2d *) w Heat: (HeatSpace *) h; // which world are we in?
-createEnd;

-(double) getUnhappiness;

-setIdealTemperature: (HeatValue) i;
-setOutputHeat: (HeatValue) o;
-setRandomMoveProbability: (float) p;
-setX: (int) x Y: (int) y; // bug's position
-setBugColor: (Color) c; // bug's colour (display)

-step;

-drawSelfOn: (id <Raster>) r;
```

Heatbug is a subclass of SwarmObject. SwarmObjects have very little behavior of their own - they are defined as the root class of most objects and control computer science aspects like memory allocation and probability.

Heatbug carry with them a variety of state variables. For instance, each Heatbug has a notion of its ideal temperature, which will affect it's behavior. In addition, Heatbugs have variables that let them know about the world: these agents are storing references to the HeatSpace object, for example.

Most of the Heatbug methods have to do with setting up the agents state - the inputs to a Heatbug. Every heatbug must set up its world and heat objects, via the `setWorld:Heat:` method. In addition when Heatbugs are created they have their ideal temperature set, their output heat, etc. Heatbugs are also observable. Heatbugs define a `getUnhappiness` method - the unhappiness is the major measurable aspect of a heatbug, how well optimized it is at the moment. They also have a `drawSelfOn` method that directs the heatbug to draw itself on the specified graphics widget.

Finally, and most importantly, a Heatbug has a `step` method. `step` is where the Heatbugs behavior is defined: every time the Heatbug is told to step it performs its internal calculations, choosing where to move. Each heatbug is told to `step` when appropriate by the model schedule. The code for `step` is the real intellectual input into the model, and is worth reading as an example of an agent's behavior.

4.3. Building Agents

Now that Heatbugs have been defined, the model swarm needs to create them. This code fragment is from the `buildObjects` method on HeatbugModelSwarm.

```
// A loop to create a bunch of heatbugs.
for (i = 0; i < numBugs; i++)
 {
  Heatbug * hbug;
  int idealTemp, outputHeat;

  // Choose a random ideal temperature, output heat from the specified
  // range (model parameters).
  idealTemp = [uniformRandom rMin: minIdealTemp Max: maxIdealTemp];
  outputHeat = [uniformRandom rMin: minOutputHeat Max: maxOutputHeat];

  // Create the heatbug, set the creation time variables
  hbug = [Heatbug createBegin: [self getZone]];
  [hbug setWorld: world Heat: heat];
  hbug = [hbug createEnd];

  // Add the bug to the end of the list.
  [heatbugList addLast: hbug];

  // Now initialize the rest of the heatbug's state.
  [hbug setIdealTemperature: idealTemp];
  [hbug setOutputHeat: outputHeat];
  [hbug setX: [uniformRandom rMax: worldXSize] // random position
         Y: [uniformRandom rMax: worldYSize]];
}
```

The details of this code are best explained in reading the documentation for the libraries and the heatbugs demo application itself. Essentially, we first generate two random numbers: an ideal temperature and an output heat for the new Heatbug. We then create the Hheatbug itself with

`createBegin` and fill in the required parameters of world and heat. Once those are set, we can send `createEnd` to the Heatbug and it is finished being created. After it's done being created we add it into a list of Heatbugs in the model and set a few parameters on it like the ideal temperature and the initial position.

4.4. Building Space objects

In Swarm, spaces are really just another kind of agent. In the heatbugs model we create a HeatSpace, a subclass of a diffusion object from the Swarm space libaries (specified in HeatSpace.m). Here is the code from `buildObjects` in the HeatbugModelSwarm.

```
heat = [HeatSpace createBegin: [self getZone]];
[heat setSizeX: worldXSize Y: worldYSize];
[heat setDiffusionConstant: diffuseConstant];
[heat setEvaporationRate: evaporationRate];
heat = [heat createEnd];
```

the object is created, a few parameters are set, and then the creation is finalized.

4.5. Scheduling a Model Swarm

Once all of the simulated objects are created in `buildObjects`, the next task is to schedule them in the method `buildActions`.

```
modelActions = [ActionGroup create: [self getZone]];
[modelActions createActionTo: heat message: M(stepRule)];
[modelActions createActionForEach: heatbugList message: M(step)];
[modelActions createActionTo: heat message: M(updateLattice)];

modelSchedule = [Schedule createBegin: [self getZone]];
[modelSchedule setRepeatInterval: 1];
modelSchedule = [modelSchedule createEnd];
[modelSchedule at: 0 createAction: modelActions];
```

The heatbug model schedule actually consists of two components: an ActionGroup called `modelActions` and a Schedule called `modelSchedule`. The ActionGroup is a tightly coupled list of three messages: every time the action group is executed, it will send three messages in a row:

```
[heat stepRule];
[heatbugList forEach: step];
[heat updateLattice];
```

The ActionGroup alone specifies three messages to send - in order to put it in the simulation, that ActionGroup is then dropped into a Schedule. The Schedule itself only has one action - to execute `modelActions` itself. That action takes place at time 0. But because we've set a repeat interval on the

schedule of 1, the schedule itself loops, executing every 1 time step. The final result is that `modelActions` is executed at time 0, time 1, etc.

4.6. Building a Graphical Observer Swarm

With the model swarm defined, arranging for a graphical observer Swarm is the next step. For Heatbugs, the code is in HeatbugObserverSwarm. The structure of an observer swarm is almost exactly like building a model swarm.

```
@interface HeatbugObserverSwarm : GUISwarm {
  int displayFrequency; // one parameter: update freq

  id displayActions; // schedule data structs
  id displaySchedule;

  HeatbugModelSwarm * heatbugModelSwarm; // the Swarm we're observing

  // Lots of display objects. First, widgets
  XColormap * colormap; // allocate colours
  ZoomRaster * worldRaster; // 2d display widget
  EZGraph * unhappyGraph; // graphing widget

  // Now, higher order display and data objects
  Value2dDisplay * heatDisplay; // display the heat
  Object2dDisplay * heatbugDisplay; // display the heatbugs
}
```

Again we have input parameters (display frequency), schedule data structures, and resident objects (model swarm, display widgets). The important exception is that HeatbugObserverSwarm is a subclass not just of the generic Swarm class, but specifically a GUISwarm. That implies that the HeatbugObserverSwarm will contain a control panel to allow the user to stop execution, and will also have a special `go` method to set everything running.

4.7. Building a Data Graph

An example of an object inside the HeatbugObserverSwarm is a data graph, the graph of average unhappiness. Here is the code necessary to create that object:

```
// Create the graph widget to display unhappiness.
unhappyGraph = [EZGraph createBegin: [self getZone]];
[unhappyGraph setTitle: "Unhappiness of bugs vs. time"];
[unhappyGraph setAxisLabelsX: "time" Y: "unhappiness"];
unhappyGraph = [unhappyGraph createEnd] ;

[unhappyGraph createAverageSequence: "unhappiness"
 withFeedFrom: [heatbugModelSwarm getHeatbugList]
 andSelector: M(getUnhappiness)] ;
```

The first step is to build an instance of an EZGraph and set its captions. Then a Sequence is created inside that graph (in this case, an AverageSequence). In general, the Sequence requires a target object and a message with which to extract data from that object -- the data is then plotted as one line in the graph.

In the case of an AverageSequence, an entire collection of objects is presented to it. The AverageSequence then extracts data from all the objects in the collection (in this case a List) using the provided message (in this case `getUnhappiness`), and generate a datapoint from these values by averaging them.

4.8. The main() function

The last main type of code needed for an application is the function `main()` the first function called in your program. All the real work has been done already - all that's left is to create the objects at the right time.

```
int main(int argc, const char** argv)
{
 id theTopLevelSwarm ;

 // Swarm initialization: all Swarm apps must call this first.
 initSwarm(argc, argv);

 // swarmGUIMode is set in initSwarm(). It's set to be 0 if you
 // typed `heatbugs --batchmode' or `heatbugs -b', Otherwise, it's set to
 // 1.

 if (swarmGUIMode == 1)
 // We've got graphics, so make a full ObserverSwarm to get GUI objects
   theTopLevelSwarm = [HeatbugObserverSwarm create: globalZone];
 else
 // No graphics - make a batchmode swarm and run it.
   theTopLevelSwarm = [HeatbugBatchSwarm create: globalZone];

 [theTopLevelSwarm buildObjects];
 [theTopLevelSwarm buildActions];
 [theTopLevelSwarm activateIn: nil];
 [theTopLevelSwarm go];

 // theTopLevelSwarm has finished processing, so it's time to quit.
 return 0;
}
```

`main()` calls `initSwarm` (required in all Swarm applications). It then detects if it should do graphics or not, creates the appropriate top level Swarm to contain the model, and sets it to running. Simple as that!

Chapter 5. Conclusion

Swarm tries to help computer simulation authors by making it easier to write simulations by making it more formal. The text above gives a narrative introduction into using Swarm for your own models, but a real understanding of Swarm will only come when you start to read through our examples and try to write your own applications. Good luck, and may your simulations be successful!

Getting Started with Swarm

Swarm Development Group

Overview

Learning to program with Swarm will require reading example applications, studying the technical reference material and sometimes even getting down to the level of reviewing the header files in libraries. It does currently require a good hands-on knowledge of object-oriented programming and software development processes in general.

Swarm is not yet a `shrink-wrapped' simulation toolkit. There are many of those kind of products on the market. However, with these packages, the ease of use comes at a price - you are locked into the that vendor's particular modelling paradigm. Swarm was intended to embrace many different types of modelling - consequently, it can be more difficult for a novice user - but more powerful in the long-run.

Revision History

2000-03-28 *mgd*

Swarmdocs 2.1.1 frozen.

2000-03-01 *alex*

Swarmdocs 2.1 frozen.

2000-03-01 Makefile.am *alex*

(noinst_DATA): Remove entirely.

2000-02-29 install01.sgml *mgd*

Updates for 2.1.

1999-09-25 install01.sgml *mgd*

Update versions for fcall, BLT. Add note about Emacs.

1999-04-20 install01.sgml *mgd*

Update versions for ffcall, Tcl/Tk, BLT, and libpng.

1999-04-06 installbook.sgml *alex*

Change PUBLIC identifier for DTD to "-//OASIS//DTD DocBook V3.1//EN". Remove JPEG notation class: now part of the 3.1 DTD.

1999-01-27 installmeta.sgml *alex*

(booktitlelogo): Scale GRAPHIC by 98%, so all recto-mode elements fit on one page in print backend.

1999-01-26 installmeta.sgml *alex*

Move bibliodata entity outside of BOOKBIBLIO - so legalnotice links work.

1999-01-13 installmeta.sgml *alex*

Include the newly-defined {doc,swarm}-legalnotice entities from global.ent. Move CORPAUTHOR inside BOOKBIBLIO.

1998-11-01 install01.sgml *alex*

Changed URL to Paul Johnson's Swarm FAQ to new location.

1998-10-14 installbook.tex.in *mgd*

Use top_dossrcdir instead of top_srcdir.

1998-10-09 installbook.tex.in *mgd*

Include tex/macros.tex.

1998-10-06 installmeta.sgml *mgd*

Make more conservative. Remove statements of intent.

1998-10-06 install01.sgml *mgd*

Update library version facts, compatibility, etc. Add URLs for both versions and general package information.

1998-07-17 install02.sgml *alex*

Fixed ID of ARTICLE to `INSTALLBOOK' rather than `INSTALL'.

1998-07-17 install01.sgml *alex*

(SIMPLESECT): Updated ULINK http and ftp locations for download of Tcl/Tk to `scriptics.com'. `smli.com' is now out-of-date. Also changed link to BLT to ftp location, rather than the webpage. (ARTICLE): Added missing ID.

1998-06-25 install01.sgml *alex*

Updated installation instruction to reflect new `configure' process.

1998-06-17 Makefile.am *mgd*

(GENERATED_SGML): Use $(NAME)revhistory.sgml instead of installrevhistory.sgml. (EXTRA_DIST): Use installbook.ent instead of install.ent.

1998-06-17 installmeta.sgml *mgd*

Use installbookrevhistory.sgml instead of installrevhistory.sgml. Scale graphic to 100%.

1998-06-16 install01.sgml *alex*

Updated `Supported Systems' to make mention of Windows NT. Updated `Prerequisite Libraries' to mention libraries required as of Swarm 1.1/1.2.

1998-06-15 Makefile.am *mgd*

Include $(top_srcdir)/Makefile.common. (EXTRA_DIST): New variable.

1998-06-15 install.ent *alex*

Removed install00.sgml from list of ENTITIES.

1998-06-15 installbook.sgml *alex*

Added new JPEG notation class and notation.

1998-06-15 Makefile.am *alex*

(SGML): Removed install00.sgml.

1998-06-15 installcont.sgml *alex*

Removed include of install00.sgml.

1998-06-15 install02.sgml *alex*

Removed links to 'reference book' elements - made all linkends be FORMALPARAs.

1998-06-15 install01.sgml *alex*

Commented-out LINK to reference book 'debug' section. Moved original PREFACE material in install00.sgml into new SIDEBAR.

1998-06-15 install00.sgml *alex*

Removed.

1998-06-12 installbook.sgml *mgd*

Include versions and figures entities.

1998-06-12 installmeta.sgml *mgd*

Scale graphic to 75%.

1998-06-12 install00.sgml, install01.sgml, install02.sgml, installmeta.sgml *mgd*

Update all IDs to SWARM.module.SGML.type.

1998-06-11 install.ent *alex*

Made all SYSTEM identifiers PUBLIC (listed in catalog now).

1998-06-11 Makefile.am *alex*

(GENERATED_SGML): Added for revhistory. (noinst_DATA): Added. (SGML, SGML_FILES): Separated out generated/non-gen files.

1998-06-11 install.ent *alex*

Created installrevhistory.sgml entity.

1998-06-11 installmeta.sgml *alex*

Replaced hardcoded revhistory with the new entity installrevhistory.sgml.

1998-06-09 installmeta.sgml *alex*

Changed corpauthor to be inserted from Global public identifiers.

1998-06-08 installbook.sgml *mgd*

Use public identifiers.

1998-05-23 installbook.tex.in *mgd*

New file.

1998-05-23 Makefile.am *mgd*

New file.

1998-05-23 installbook.sgml *mgd*

Make ID of book be "installbook".

1998-05-23 Makefile *mgd*

Remove.

1998-05-22 *mgd*

Begin revision log.

Table of Contents

Unless you are using a Windows or a Unix system with pre-built package manager support (Debian GNU/Linux 2.2, Redhat 6.1, Solaris 2.7), installing Swarm will take some time: various external libraries need to be acquired, compiled, and installed, and then Swarm itself needs to be compiled. Please report any problems during installation back so that we can try to fix them in the future.

If you are using a Unix system with binary package manager support, please read the manual appropriate to the manager. Respectively, these are `dpkg', `rpm', and `pkgadd'.

Swarm uses quite a few libraries and is intended to run under all major flavors of Unix. This presents the unpleasant but largely unavoidable side-effect of making Swarm hard to install. Ideally, you should get the sys-admin for your site to do the job. In any event, the new online *Swarm FAQ* where we've tried to compile some of the common obstacles to installation, may be useful.

- **Online FAQ *(http://lark.cc.ukans.edu/~pauljohn/SwarmFaq/SwarmOnlineFaq.html)* .** Paul Johnson's active Swarm FAQ. *Share your wisdom!*

1. Supported Systems

- **Unix.** Our intention is that Swarm will run on any modern Unix system. Ideally, Swarm itself should be 100% portable. Swarm has been known to run on SunOS 4.1.3, Solaris 2.[567], GNU/Linux systems for Intel, Sparc, and PPC, FreeBSD, HPUX 9, 10, and 11, IRIX 5.3 & 6.5, Digital Unix for Alpha and MachTen/68k.
- **Windows NT.** Swarm can be used on Windows. Installation on Windows is a simple matter of running the InstallShield package. Please see the release page *(http://www.swarm.org/release-swarm.html)* , for more details.

See the Platform News *(http://www.swarm.org/release-platforms.html)* web page for the most up-to-date information.

2. Prerequisite Programs

We assume you already have Unix and X Windows on your system: the rest of this software is freely available. Expect to spend some time compiling and installing these packages. URLs on this page are to the home distribution site: as a convenience, the Swarm ftp site contains copies of the necessary packages in needed-software *(ftp://ftp.swarm.org/pub/swarm/needed-software)* .

- **GNU gcc.** gcc is the FSF's GNU Compiler Collection. You need gcc 2.95.2 or greater installed on your system, including the Objective C support. The latest version is available from ftp://ftp.gnu.org/pub/gnu/gcc *(ftp://ftp.gnu.org/pub/gnu)* .
- **GNU make.** Make is used to automate building programs: every OS ships their own version of make with some random set of features. The Swarm makefiles currently use features that are not supported by all makes: GNU make is the only one guaranteed to work. Linux systems probably use GNU make already. The latest version of GNU make is available here *(ftp://ftp.gnu.org/pub/gnu/make)* .
- **GNU gdb.** gdb is the GNU debugger: Swarm doesn't require this, but you will probably want gdb on your system to debug programs. gdb is available here *(ftp://ftp.gnu.org/pub/gnu/gdb)* .

- **GNU Emacs.** Emacs is a programmable text processing system and editor. Emacs is needed if you want to build Swarm for Java from source code. Emacs is also a good program for editing Objective C and Java code. Emacs is available here *(ftp://ftp.gnu.org/pub/gnu/emacs/)* .

3. Prerequisite Libraries

Swarm uses a variety of freely available software libraries. All of these libraries need to be configured and installed independently of Swarm. When you configure Swarm itself, take a look at the output of "configure --help" to see what "--with-*dir" options are available for locating the installations of these libraries.

- **XPM Library *(http://www.inria.fr/koala/lehors/xpm.html)* .** The XPM library adds pixmap (coloured bitmap) support to X11. XPM is a common X library, many systems already have it installed. A reasonably modern version is needed: we've used version 3.4f. If you get complaints about "Object" being multiply defined, your version of libXpm is too old. The library is available from ftp://ftp.x.org/contrib/libraries/xpm-3.4k.tar.gz *(ftp://ftp.x.org/contrib/libraries/xpm-3.4k.tar.gz)* .
- **Tcl/Tk *(http://www.scriptics.com)* .** Swarm works with Tcl/Tk versions 8.2.3 *(ftp://ftp.scriptics.com/pub/tcl/tcl8_2/tcl8.2.3.tar.gz)* /8.2.3 *(ftp://ftp.scriptics.com/pub/tcl/tcl8_2/tk8.2.3.tar.gz)* and later.
- **BLT *(http://www.tcltk.com/blt/index.html)* .** BLT is an add-on to Tk that provides more widgets. BLT 2.4o *(http://ftp.tcltk.com/pub/blt/BLT2.4o.tar.gz)* is the current version.
- **ffcall *(http://clisp.cons.org/~haible/packages-ffcall.html)* / libffi *(http://www.cygnus.com/~green/libffi.html)* .** Either ffcall or libffi can be used in Swarm to provide the underlying support for message probing. ffcall is provided with Swarm and works for most platforms. Both libraries provide a portable, high level programming interface to various platform calling conventions. This allows a programmer to call any function specified by a call interface description at run time. ffi stands for Foreign Function Interface. A foreign function interface is the popular name for the interface that allows code written in one language to call code written in another language.

The latest version of ffcall is 1.6 *(ftp://ftp.swarm.org/pub/gnu/ffcall-1.6.tar.gz)* The latest version of libffi is: 1.20 *(ftp://ftp.cygnus.com:/pub/green/libffi-1.20.tar.gz)*

- **libpng *(http://www.cdrom.com/pub/png)* .** Swarm requires support for pixmap images - `png' provides that support.

The current version is 1.0.5 *(ftp://swrinde.nde.swri.edu/pub/png/src/libpng-1.0.5.tar.gz)* **zlib *(http://www.cdrom.com/pub/infozip/zlib)* .** png requires data compression which zlib, a general data compression library, provides. The current version is 1.1.3 *(ftp://ftp.cdrom.com/pub/infozip/zlib/zlib.tar.gz)* .

4. Configuring and Installing Swarm

Once you have all of the above software installed, the next step is to configure Swarm. First unpack the Swarm .tar.gz file into a convenient place `(via gzip -dc swarm-xx.yy.zz.tar.gz | tar xf -)`.

As of the Swarm 1.2 release, the compilation and installation of Swarm is handled by a GNU **configure** script. This removes a large burden from the user, as **configure** has some intelligence which automatically sets many common options.

You first need to read the file `INSTALL` with the release of Swarm, in the top-level source directory, to determine the appropriate and recent options to give to the **configure** script (this is analogous to what you needed to do when editing the *INCDIR macros in the obsolete `Makefile.conf` in earlier releases).

5. Compiling Swarm Libraries -- (not required for binary distributions)

You've done the hard part, now type **make**, as in earlier releases from the top of the source directory, and watch the fun! If running **make** in the root Swarm directory does nothing, you probably aren't running GNU Make.

There is now an explicit "installation" step to install Swarm -- **make install** so that binaries and libraries can be installed cleanly to a specified location. On many Unix systems this likely to somewhere like `/usr/local/`.

The Makefiles included by user-created libraries, and user-created applications are `Makefile.lib`, and `Makefile.appl`, respectively. These distributed with earlier versions are now generated by **configure** . If you run into problems, the right thing to do is to re-run **configure**, and re-install Swarm, tinkering with these generated Makefiles directly is not recommended.

6. Compiling and Running Swarm Applications

Swarm applications are distributed separately: you will need to download and unpack applications independently. After the application is unpacked, you will have to set the `SWARMHOME` environment variable to where you installed Swarm. All you need to do now, to compile an application is type **make**.

Once the application is compiled, just run it out of its own directory. A control panel and a couple of parameter windows should pop up; press the "Go" button to watch it run. Congratulations!

7. Post-compile cleanup

After you've tried out the Swarm libraries for awhile you may want to clean up the intermediate .o files, and other generated files, in the original source directory. Once you have done the **make install** you can remove the entire source directory, or if you might want to re-install later with different option, just type **make clean** to clean-up all files generated by the original make. See your the file `INSTALL` in the Swarm distribution, for more details.

Just remember to set `$(SWARMHOME)` to the right directory in your application's makefile. If you have installed Swarm, but for some reason, need to recompile the library and want to start from a completely clean source directory, you can run **make uninstall** and then **make clean**- see `INSTALL` file again here.

1. Libraries

The main part of Swarm is a set of libraries, one per source directory. Briefly, the directory organization is as follows:

lib, include, bin. Installed libraries, include files, and helper binaries.

src. Swarm library source files.

- **defobj.** Support of Swarm style Objective C programming
- **collections.** Collection library -- various objects that "collect" other objects
- **activity.** Schedule execution mechanism -- the "process"
- **objectbase.** Base class for simulated objects -- the "agents"
- **space.** Spatial environment library (currenly mostly 2D spaces are supported).
- **random.** Random number library.
- **simtools.** Collected tools (non-GUI) useful for developing Swarm simulations
- **simtoolsgui.** Further tools useful for interactive "GUI-oriented" simulations
- **gui.** Graphical Interface library
- **analysis.** Objects that help with data processing

2. Sample applications

In addition, there is a set of sample Swarm applications that are distributed separately. Applications are the best current roadmap for the Swarm code: much of what is possible with Swarm is demonstrated in the sample applications. The most important applications are:

- **template.** Template simulation. The code itself here is trivial, but provides a nice base for new Swarm programmers to start from.
- **tutorial.** A step-by-step tutorial to using Swarm. The tutorial starts with a simple implementation of a cellular automaton in ordinary "C" and proceeds up to a full-blown GUI-oriented Swarm application in Swarm Objective C.
- **heatbugs.** Heatbugs, our canonical, simple complex system. The code here is thoroughly commented for use as a guide to swarm programming.
- **mousetrap.** Mousetrap is a discrete event simulation of a room full of mousetraps loaded with ping-pong balls. The triggering of one of the traps sets off a chain reaction supposedly similar to fission. It is also thoroughly commented.
- **market.** Market is an Objective-C wrapped piece of legacy code originally written in C. It is a useful example of how to convert legacy code into Swarm; but, it is not very instructive on Swarm programming in general.

Reference Guide for Swarm 2.2

Swarm Development Group

Overview

Swarm is a collection of libraries against which you link your simulation code. This document describes the interface for those core libraries.

> **Important:** The probe, random and technical appendices formerly part of the *Reference Guide for Swarm* have been removed and relocated to the new *Swarm User Guide (http://www.swarm.org)* . In the interests of conserving paper, the grid turtle example programs are available in `tar.gz` format from the Swarm ftp site (see Appendix A) and pointers to the full text of the LGPL and GPL licenses (see Appendix C) are provided rather than the full text itself.

Revision History

2000-03-28 *mgd*

Swarmdocs 2.1.1 frozen.

2000-03-01 *mgd*

Swarmdocs 2.1 frozen.

2000-03-01 Makefile.am, Makefile.rules *mgd*

(nodist_noinst_DATA): Add. Needed in order to get the dependency for generated SGML. (gridexamples.sgml): Remove. (GENERATED_SGML): Remove gridexamples.sgml. (EXAMPLES): Remove. (refindex.sgml $(PAGES), graph): Quote SWARMSRCDIR and SWARMDOCS.

2000-03-01 grid-app.sgml *mgd*

Add LINK to IMPORTANT for all files.

2000-03-01 ref00a.sgml *mgd*

Remove obsolete reference to release changes.

2000-03-01 refcont.sgml *alex*

Remove {lgpl,gpl}-app.sgml inclusion in content.

2000-03-01 refbook.ent *alex*

Likewise for the relevant ENTITIES.

2000-03-01 lgpl-app.sgml, gpl-app.sgml *alex*

Remove files.

2000-03-01 Makefile.am *alex*

(SGML): Add license-app.sgml.

2000-03-01 refmeta.sgml *alex*

Add admonition regarding reorganization and deletion of appendices in ABSTRACT.

2000-03-01 grid-app.sgml *alex*

Remove the inline gridturtle code. Add admonition specifying location of tar.gz file for download.

2000-03-01 Makefile.am *alex*

(noinst_DATA): Remove variable entirely, since we neither install nor distribute the GENERATED_SGML files.

2000-03-01 Makefile.rules *alex*

(noinst_DATA): Likewise for generated revhistory.sgml files

2000-02-29 ref00a.sgml *mgd*

Add 2.1 porting notes.

2000-02-15 ref00.sgml *alex*

Update Acknowledgment section and remove swarm@santafe.edu mailto link.

2000-02-15 refcont.sgml *alex*

Comment-out random-app.sgml, probes-app.sgml, techcont.sgml APPENDIXes. All material either outdated or relocated to the Swarm User Guide.

1999-09-22 graph.el *mgd*

Use TOP_BUILDDIR, rather than BUILDDIR to load protocol.

1999-06-22 sgml.el *mgd*

(sgml-method-definitions): Use methodinfo- functions instead of list accessors.

1999-06-22 Makefile.am *alex*

(sgml.elc): Depend on $(abs_top_builddir)/protocol.elc, rather than just protocol.elc. (refindex.sgml $(PAGES)): Likewise. (graph): Likewise.

1999-06-22 sgml.el *alex*

(load-path): Set to TOP_BUILDDIR, rather than BUILDDIR.

1999-06-21 sgml.el *alex*

(sgml-refmeta): If deprecated protocol, print Deprecated in square braces after REFENTRYTITLE. (sgml-refsect1-text-list): Print out the deprecated doc-string if set, as a PARA element with EMPHASIS before description text. (sgml-refsect1-description): Pass object to `sgml-refsect1-text-list', so it can determine whether to print deprecated doc-string.

1999-06-21 Makefile.am *mgd*

(protocol.elc): Remove rule.

1999-06-02 ref00a.sgml *alex*

Add porting notes for 1.4/1.4.1 -> 2.0 changes.

1999-04-23 Makefile.am *mgd*

(EXTRA_DIST): Add graph.el. (MODULES): New variable: (SUBDIRS): Use it. (graph): New target.

1999-04-23 graph.el *mgd*

Moved here from toplevel directory.

1999-04-06 refbook.sgml *alex*

Update PUBLIC identifier for DTD to "-//OASIS//DTD DocBook V3.1//EN" Remove JPEG notation class - now part of the 3.1 DTD.

1999-02-28 sgml.el *alex*

(run-all): Set uniquify-method-lists to `nil' when calling load-and-process-modules. Ensures all method documentation called correctly.

1999-02-24 Makefile.am *mgd*

(protocol.elc): Dependency on $(swarm_srcdir)/etc/protocol.el. (sgml.elc): Dependency on common.elc and protocol.elc. Both rely on implicit rules. (EXTRA_DIST): Add sgml.el.

1999-02-24 sgml.el *mgd*

(sgml-generate-indices): Use get-top-builddir instead of get-swarmdocs-build-area.

1999-02-16 Makefile.am *mgd*

(refindex.sgml $(PAGES)): Depend on $(swarm_srcdir)/etc/{protocol,common}.el instead of $(top_srcdir)/{protocol,common}.el. Load sgml.el instead of protocol.el.

1999-02-09 refcont.sgml *alex*

(random-app.sgml): Add `Random' Appendix to list of contents.

1999-01-26 refmeta.sgml *alex*

Move bibliodata entity outside of BOOKBIBLIO - so legalnotice links work.

1999-01-23 ref00a.sgml *mgd*

Remove porting note about .swarmArchiver. Swarm 1.4 will read old files, but then change the syntax.

1999-01-15 ref00a.sgml *alex*

Add porting note on Histogram protocol change.

1999-01-13 refmeta.sgml *alex*

Remove old LEGALNOTICE text. Refer the newly-defined {doc,swarm}-legalnotice entities from global.ent. Move CORPAUTHOR inside BOOKBIBLIO.

1999-01-13 gpl-app.sgml *alex*

Refer to the Swarm documentation as being under the terms of the GPL.

1999-01-13 ref00a.sgml *alex*

(SECT1): Add porting notes for Swarm 1.4.

1999-01-07 ref00a.sgml *alex*

Make SIMPLELIST an ITEMIZEDLIST.

1999-01-07 gpl-app.sgml *alex*

Remove SIDEBAR from wrapping the licence text, SIDEBAR can't run over a page in the printed backend. Make introductory description a SIDEBAR.

1999-01-07 lgpl-app.sgml *alex*

Likewise.

1999-01-07 ref00.sgml *alex*

Replace SECT1 with two SIMPLESECTs. Tidying and reformatting. (Acknowledgements): Updated.

1998-12-22 probes-app.sgml *alex*

({customized,complete}-probe-map): Add IDs to FIGUREs

1998-10-14 refbook.tex.in *mgd*

Use top_dossrcdir instead of top_srcdir.

1998-10-09 refbook.tex.in *mgd*

Include tex/macros.tex.

1998-08-25 ref00a.sgml *mgd*

Add 1.2 -> 1.3 porting notes.

1998-07-23 ref00a.sgml *mgd*

Put items related to HeatbugObserverSwarm.h in their own list.

1998-07-18 ref00a.sgml *mgd*

Clarify wording about not using statically typed Swarm objects.

1998-06-25 Makefile.am *alex*

(refindex.sgml $(PAGES)): Set temporary environment variable SWARMSRCDIR before invocation of batch-mode e-lisp program `protocol.el'.

1998-06-24 ref00a.sgml *alex*

Tidied 1.1 => 1.2 porting notes.

1998-06-23 ref00a.sgml *alex*

Added porting notes for Swarm 1.1 => 1.2. Expanded porting notes for Swarm 1.0.5 => 1.1. Added intro SIDEBAR.

1998-06-17 refmeta.sgml *mgd*

Use refbookrevhistory.sgml instead of srcrevhistory.sgml. Scale graphic to 100%.

1998-06-17 refbook.sgml *mgd*

Use refbook.ent instead of src.ent.

1998-06-17 Makefile.am *mgd*

Include refbook/Makefile.rules instead of src/Makefile.rules. (GENERATED_SGML): Rename srcrevhistory.sgml to refbookrevhistory.sgml. (ENT, EXTRA_DIST): Use refbook.ent instead of src.ent.

1998-06-17 refbook.ent *mgd*

Renamed from src.ent.

1998-06-16 probes-app.sgml *alex*

(complete-probe-map): Scale graphic to 50%. (customized-probe-map): Scale graphic to 75 %. In 'Support for Probing' section - make ITEMIZEDLIST spacing=compact.

1998-06-15 Makefile.am *mgd*

Include $(top_srcdir)/src/Makefile.rules. (SGML): Move ENT to SGML_FILES. (EXTRA_DIST): New variable.

1998-06-15 Makefile.rules *mgd*

New file.

1998-06-12 refmeta.sgml *mgd*

Scale graphic to 75%.

1998-06-12 Makefile.am *mgd*

(gridexamples.sgml): Update IDs to SWARM.module.SGML.type.

1998-06-12 conventions-app.sgml, gpl-app.sgml, grid-app.sgml, lgpl-app.sgml, probes-app.sgml, ref00.sgml, ref00a.sgml, refmeta.sgml *mgd*

Likewise.

1998-06-11 refbook.sgml *mgd*

Use jpeg instead of jpg for notation and local.notation.class.

1998-06-10 Makefile.am *mgd*

Move CLEANFILES to Makefile.rules. (GENERATED_SGML): Move versions.ent to Makefile.rules. (ENT): New variable, the list of .ent files for this module. (SGML): Add ENT.

1998-06-10 lgpl-app.sgml, gpl-app.sgml, ref00a.sgml *alex*

Fixed the IDs to have the appropriate "SWARM.SRC." prefix in the content ID.

1998-06-10 ref00.sgml *alex*

Made SIMPLESECTs into SECT2s. Fixed redundant "SWARM.SRC." in PREFACE id.

1998-06-09 refmeta.sgml *alex*

Change CORPAUTHOR to 'corpauthor' - an SFI Hive global entity.

1998-06-09 refmeta.sgml *mgd*

Change CORPAUTHOR to SDP. Overview needlessly wordy.

1998-06-08 refbook.sgml.in *mgd*

Use public identifer for global.ent, and src.ent. Add versions.ent and figs.ent.

1998-06-08 Makefile.am *mgd*

(GENERATED_SGML): Add versions.ent.

1998-06-07 src.ent.in *mgd*

Define graphic entities per html/print.

1998-06-07 refbook.sgml.in *mgd*

Add local.notation.class entity and notation for JPG.

1998-06-07 config.ent *alex*

Removed.

1998-06-07 src.ent *alex*

Removed reference to config.ent.

1998-06-06 src.ent.in *mgd*

Don't use extracted locations, as module entities now all reference public identifiers.

1998-06-05 src.ent.in *mgd*

Add srcrevhistory.sgml.

1998-06-05 refmeta.sgml *mgd*

Replace REVHISTORY with srcrevhistory.sgml.

1998-06-05 Makefile.am *mgd*

(SUBDIRS): Add tech. (swarm_ChangeLog): Add (empty). (GENERATED_SGML): Add srcrevhistory.sgml.

1998-06-03 Makefile.am *alex*

(CLEANFILES): Changed hardcoded refbook.rtf to $(NAME).rtf. Added Local variable mode for emacs makefile-mode.

1998-06-03 Makefile.am *mgd*

(gridexamples.sgml): Remove unnecessary subshell for `for' loop and don't cd to $(srcdir) when done (it's in a subshell).

1998-06-01 grid *alex*

Moved the grid subdir to the root of the Swarm application source tree and renamed to gridturtle.

1998-06-01 Makefile.am *alex*

(gridexamples.sgml): After changing directories into $(gridturtle_srcdir) - change directory back to $(srcdir) to restore location.

1998-06-01 Makefile.am *mgd*

(swarm_srcdir, gridturtle_srcdir): New variables which get substituted assignments from configure. Use them instead of $(SWARMHOME) and $(srcdir)/grid.

1998-06-01 grid-app.sgml *alex*

Fixed incorrect LINKENDS and descriptions for grid5.m and grid6.m.

1998-05-29 probes-app.sgml *mgd*

Update LINKENDs per new ID conventions.

1998-05-26 Makefile.am *mgd*

(refindex.sgml $(PAGES)): Add $(top_srcdir)/protocol.el and $(top_srcdir)/common.el as dependents. (refindex.sgml $(PAGES)): Set SWARMDOCS environment variable to the fully-resolved top_srcdir.

1998-05-23 refbook.sgml.in *mgd*

New file.

1998-05-23 refbook.tex.in *mgd*

New file.

1998-05-23 refbook.sgml *mgd*

Removed.

1998-05-23 Makefile.am *mgd*

New file.

1998-05-23 Makefile *mgd*

Remove.

1998-05-23 src.ent.in *mgd*

New file.

1998-05-23 src.ent *mgd*

Removed.

1998-05-22 *mgd*

Begin revision log.

Table of Contents

List of Examples

Preface

1. The Swarm Hive

The Swarm home page *(http://www.swarm.org)* is the place to obtain both new software releases and documentation for Swarm. We welcome comments about Swarm and it's documentation.

You can also mail comments to the `Swarm Support` mailing list. See the Swarm mailing lists page *(http://www.swarm.org/community-mailing-lists.html)* for instructions on how to join this list.

2. Acknowledgments

The Swarm project was initiated by Chris Langton (now at the Swarm Corporation). Roger Burkhart (John Deere, now at VantagePoint Network), Nelson Minar (now at the MIT Media Lab), Glen Ropella (now at Swarm Corporation), Manor Askenazi (now at Coopers & Lybrand), Irene Lee (SDG), Vladimir Jojic (UIUC), and Alex Lancaster (Santa Fe Institute) have all been active in the design and programming of Swarm. Marcus Daniels is the current maintainer and lead developer at the SDG.

Much of the early development of Swarm was done by Nelson Minar, who is now at the MIT Media Lab, and Manor Askenazi and Glen Ropella (Swarm Corporation), both formerly of the SFI. The random library was written by Sven Thommesen and graciously contributed. Eric Carr and JJ Merelo have been patient Swarm users and developers. Howard Gutowitz and David Hiebeler have participated in the design process. (David is also the implementor of the original Swarm prototype.) And, many thanks to Simon Fraser, Swarm graphic designer extraordinaire. Finally, many thanks go out to the Swarm user community, particularly those who have stuck with us from the beginning.

Swarm Changes and Compatibility

Notes on Porting

New versions provide new features, but may have the unfortunate side-effect of breaking many existing applications. This document is intended to provide a step-by-step guide to updating your Swarm applications to use the latest version as quickly and painlessly as possible.

This document contains detailed examples for porting versions equal to, or later than, Swarm 1.0.5.

Users trying to port from versions *earlier* than 1.0.5, are advised to first view the compatibility section of the 1.0.5 swarmdocs. First perform those changes in sequence, (i.e. if you have apps compatible with Swarm 1.0.3 - first make the 1.0.3 => 1.0.4 changes, followed by the 1.0.4 => 1.0.5 changes) and then perform the changes described here. You may notice that some of the changes overlap as earlier versions may have merely deprecated some functionality, whilst later versions actually disabled it. This has been done to phase new functionality in, and old functionality out over several releases, so the user isn't hit with an enormous burden every minor release.

We intend that most application changes should be covered, but given that we can never know in advance how a user has employed the software, we cannot guarantee that all potential application changes will be covered in the document explicitly.

Key for changes:

- (*) = absolutely necessary changes (if not performed, will fail to compile)
- (x) = not-strictly necessary, but highly recommended changes (deprecated coding practice, or will produce a compiler warning, if not performed)

1. Porting from 2.0 or 2.0.1 to 2.1

- Backward-compatibility Random module protocol names have been removed. (Generators have the name suffix "gen" and distributions have the suffix "Dist".)
- The Archiver method `getWithZone:object:` was renamed to `getWithZone:key:`.

2. Porting from 1.4 or 1.4.1 to 2.0

- The method `setNumBins:`, found in the analysis library protocol EZBin *(see page 369)* and also in the underlying Histogram *(see page 336)* protocol from the gui library, has changed its name to `setBinCount:`. You will need to change all invocations of `setNumBins:` on objects that conform to either of these protocols, to `setBinCount:`.

3. Porting from 1.3 or 1.3.1 to 1.4

- The `Arguments` protocol has moved from the `objectbase` library to the `defobj` library. If you were subclassing from the `Arguments` class you will now need to subclass from `Arguments_c` and import the `defobj.h` header file.
- The `ListShuffler` class has been moved from `simtools` to `collections`. You must now import `collections.h`. Alternatively you can use the new `[beginPermuted: aZone]` index creation method on a collection from the Collection *(see page 103)* protocol in place of the `ListShuffler` protocol.
- The `-setLabels::` and `-setColors::` methods in the Histogram *(see page 336)* protocol now each must be given a new `count: (unsigned)labelCount` and `count: (unsigned)colorCount` argument, respectively. So, for example, the code formerly in `MarketObserverSwarm.m` in the market application was:

```
[useHisto setLabels: pred];
[useHisto setColors: predictorColors];
```

is now changed to:

```
[useHisto setLabels: pred count: numPredictorsToShow];
[useHisto setColors: predictorColors count: numPredictorsToShow];
```

4. Porting from 1.2 to 1.3

- The location for `Makefile.appl` is now `$(SWARMHOME)/etc/swarm` instead of `$(SWARMHOME)`. Change the `include' directive in your application or library make file to the new pathname.
- ActiveGraph and ActiveOutFile have been moved to the analysis library. The accommodate this, ensure that any source file that uses these protocols includes `analysis.h`.

5. Porting from 1.1 to 1.2

5.1. Major changes

There are really only two main changes which are likely to affect existing users, and all other changes required in user's applications mostly flow from these two changes:

- All Swarm functionality is now exported through a single, well-defined interface: Objective C protocols. Essentially this means that all creatable or subclass-able protocols now follow the same conventions as those in the defobj, activity and collections libraries. This means that static typing of Swarm protocols is now obsolete across the whole package.
- Library header files no longer include any of their individual class header files.
- The `XPixmap` protocol has now been changed to `Pixmap` to divorce itself from it's association with X-Windows.

5.2. Porting Guide

- Any subclass of a class defined as a Swarm protocol now requires the *specific* importation of that protocol's header file. Previously, only a warning was raised. This particularly affects `Swarm` and `SwarmObject`. If you define something like:

  ```
  @interface MyObject: SwarmObject
  ```

  then you need to include:

  ```
  #import <objectbase/SwarmObject.h>
  ```

  Similarly, if you subclass from Swarm, you will require `objectbase/Swarm.h`

- All references to objects conforming to Swarm protocols should either be defined to conform to the appropriate Swarm protocol, or be left untyped. In no circumstance should it be statically typed. In practice, this means that you can write, either:

  ```
  id <Grid2d> world;  OR
  id world;
  ```

  but NOT

  ```
  Grid2d *world;
  ```

  Note that this only applies to protocols defined by Swarm proper, not to your own classes, although it is good practice to adopt a convention and stick to it.

5.3. Porting example: heatbugs

`HeatSpace.h`

- Include header file for `space/Diffuse2d` (x)

`Heatbug.h`

- Make instance variable world conform to protocol: `id <Grid2d> world` NOT `id Grid2d *;` (x)

`Heatbug.m`

- In `[Heatbug -setWorld:]` method: make cast to protocol not static (x)

`HeatbugModelSwarm.h`

- HeatbugObserverSwarm.h
  - Make instance variable `id <Grid2d> world` NOT `Grid2d *world` (x)
  - Change static cast of `[HeatbugObserverSwarm -getWorld:]` method to protocol version (x)
  - Change all the below to conform to their respective protocols (x)

```
EZGraph *unhappyGraph;
Value2dDisplay *heatDisplay;
Object2dDisplay *heatbugDisplay;
```

5.4. Porting example: mousetrap

`MousetrapModelSwarm.h`

- Import `<objectbase/Swarm.h>` for subclassing from Swarm (*)
- Remove imports of `<activity.h> <collections.h> <simtools.h> <objectbase.h>` irrelevant, as not used in the interface. (x)
- Make instance of `Grid2d` conform to protocol, not static typed. (x)

`MousetrapObserverSwarm.h`

- Make instances of `EZGraph, Object2dDisplay` conform to protocol, rather than be statically typed. (x)

`Mousetrap.h`

- Remove import of `<objectbase.h>` (x)

6. Porting from 1.0.5 to 1.1

6.1. Major changes

The major changes which are likely to affect existing users are the following:

- Addition of a new library class "gui" which replaces the existing "tkobjc" library.
- All direct references to any Tcl/Tk code such as the "globalTkInterp" variable have been completely removed from any library code and should not be used in any application.
- Splitting of simtools into two separate libraries: simtools and simtoolsgui. Classes that intrinsically depend on a GUI toolkit being present (either Tcl/Tk or Java AWT) were put into simtoolsgui. This allows a user who never intends to use a GUI toolkit, to be able to compile, link and run Swarm applications *without any* GUI toolkit installed at all. This was not previously possible.
- The above change has been made possible by the fact that the dependency of Tcl/Tk in the Probing mechanism has been completely removed. It has been replaced by the libffi/ffcall libraries. Other than the fact that the user will need to install this new library if they are not using a binary distribution, this new dependence should *not* break any user code.
- The header file to the random library is no longer included in simtools.h you need to explicitly import random.h when you use a default random number generator.
- The XColormap class is now just Colormap to divorce it from its association with X-Windows. Similarly, BLTGraph is now just Graph.
- The class named Histo is now named Histogram.
- Certain classes now enforce their defining by the protocol method. For example an instance of the Raster class must be defined as `(id <Raster> r)` rather than `(Raster * r)`.
- Backwardly-compatible references to the old swarmobject library are no longer supported. You should always use objectbase.

6.2. Porting guide

- Always use the "gui" protocol when calling doing any GUI events:
  - Replace all occurences of tkobjc.h with gui.h

  - *never* explicitly reference any Tcl/Tk-specific code, in particular module with a call to globalTkInterp will no longer compile.

- Using simtools/simtoolsgui:
  - Add the header file simtoolsgui.h to your list of imports whenever you are referencing any of the following classes:

    ```
    ActionCache, ControlPanel, SimpleProbeDisplay, ActionHolder,
    GUIComposite, VarProbeWidget, ActiveGraph, GUISwarm,
    ClassDisplayWidget, MessageProbeWidget, CommonProbeDisplay,
    ProbeDisplay, CompleteProbeDisplay,
    ProbeDisplayManager
    ```

- Explictly import the header when you are subclassing from a given class:
  - You also need to import the header file for any class for which you are subclassing. In particular, when you are creating a GUISwarm you are subclassing from GUISwarm so you need to explicitly import both <simtoolsgui.h> AND <simtoolsgui/GUISwarm.h>. The same is true for <objectbase/SwarmObject.h> and <objectbase/Swarm.h>
- Colormap class name change:
  - Change all references of XColormap to Colormap as it is no longer specific to X11.
  - Change all method references to set the Colormap for the Value2dDisplay class ([Value2dDisplay -Colormap]) to lowercase ([Value2dDisplay -colormap]) to avoid namespace conflicts with Colormap class. For example, in heatbugs:

    ```
    [heatDisplay setDisplayWidget: worldRaster Colormap: colormap];

                    should now be:

    [heatDisplay setDisplayWidget: worldRaster colormap: colormap];
    ```

- Ensure that all required classes conform to their protocol:
  - Make all occurences of (Raster *) to the protocol i.e. (id <Raster>)
  - Similarly change any references to Colormap, ZoomRaster, InFile and OutFile.

- Import random.h explicitly:
  - The header file <random.h> is no longer included by <simtools.h> should always *explicitly* reference the random library if you use it in a given module (.m) file. This again reduces the inter-library dependence, if you don't need to use the random library in your application, you shouldn't be including it.

- ActionCache and ControlPanel:
  - Make all references to -doTkEvents and -waitForControlEvent be to actionCache NOT controlPanel.

6.3. Porting example: heatbugs

To help users port their applications to 1.1, I have included a checklist of changes that were required to update heatbugs from 1.0.5 to 1.1. This may help some users as a kind of "template" for changes they may require for their applications. The `ChangeLog` entries in in the heatbug-1.1 distribution also provide further specific information.

`Heatbug.h`

- Replace <tkobjc/Raster.h> with <gui.h> *(*)*
- Make all occurences of (Raster *) to the protocol (id <Raster>) *(*)*

`Heatbug.m`

- Make all occurences of (Raster *) to the protocol (id <Raster>) *(*)*
- Removed <simtools.h> altogether - not used. *(*)*
- Added <random.h> - no longer included by <simtools.h> - should always *explicitly* reference the random library if you use it in your code. *(*)*

`HeatbugBatchSwarm.h`

- Changed <swarmobject.h> to <objectbase.h> *(*)*

`HeatbugBatchSwarm.m`

- Removed redundant <collections.h> *(x)*

`HeatbugModelSwarm.h`

- Removed <tkobjc.h> irrelevant in this context - tkobjc should never be included directly in any case, if required use <gui.h> *(*)*
- Changed <swarmobject.h> to <objectbase.h>. *(*)*
- Note we need to *separately* include <objectbase/Swarm.h> since you always need to the header file for a class if you need to subclass from it. *(*)*

`HeatbugModelSwarm.m`

- Include <random.h> explicitly since we use the default random number generators. *(*)*

`HeatbugObserverSwarm.h`

- Change <simtools.h> to <simtoolsgui.h> since we are using GUI widgets *(*)*
- Explicitly import <simtoolsgui/GUISwarm.h> since we subclass from it *(*)*
- Remove: <swarmobject.h> <space.h> <activity.h> <collections.h> all are irrelevant in the header file. *(x)*
- <tkobjc.h> has been relocated to the (.m) file as no gui classes are referenced directly in the header (.h) file. It is now changed to <gui.h>. *(*)*

- Change all references of XColormap to Colormap and use protocol form: *(*)*

```
XColormap * colormap TO
id <Colormap> colormap
```

- Make ZoomRaster conform to protocol, ie: *(*)*

```
ZoomRaster * worldRaster TO
id <ZoomRaster> worldRaster
```

`HeatbugObserverSwarm.m`

- Change swarmobject to objectbase *(*)*
- Import the <gui.h> in the implementation file - since it is not referenced in the header file *(*)*
- Change XColormap to Colormap - no longer specific to X11 - so name should not suggest so. *(*)*
- Message to set colormap for worldRaster changed name from (uppercase) Colormap to (lowercase) colormap.*(*)*
- Call -enableDestroyNotification method on worldRaster after createEnd. *(x)*

`main.m`

- Need to import <simtoolsgui.h> in addition to <simtools.h> since we reference GUISwarm methods. *(*)*

7. Porting from 1.0.4 to 1.0.5

- EZGraph's setGraphWindowGeometryRecordName and GUISwarm's setControlPanelGeometryRecordName have been retired. The macro SET_WINDOW_GEOMETRY_RECORD_NAME can now be used in any geometry archiving context.
- After adjusting an application per previous item, be aware that the new internal naming conventions used by `GUIComposite' classes (e.g. EZBin and GUISwarm) will probably differ from the archiving keys in your application. Such widgets will probably lose their saved placements.

8. Porting from 1.0.3 to 1.0.4

- doTkEvents has been moved from ControlPanel to ActionCache. Be sure to direct all doTkEvent messages to the Swarm's instance of ActionCache and not the ControlPanel (a warning message will be generated otherwise).
- The swarmobject library has been renamed to objectbase. Although the build procedure creates a link to maintain compatibility, applications should include <objectbase.h> instead of <swarmobject.h>.
- Avoid calling globalTkInterp in all future applications, it will not be supported from version 1.1 onwards.
- If there exists a <LIBRARY.h> file, e.g. <simtools.h>, that file is the advertised interface to LIBRARY, and it will defined in terms of protocols. Whenever possible, define variables in terms of the protocol they respond to:

```
id <ProbeMap> probeMap;
```

rather than using a static type:

```
ProbeMap *probeMap;
```

(One advantage of this is that fewer imports will be needed in your application.)

9. Porting from 1.0.2 to 1.0.3

- Activity Library: Reference Release-Note #3: The AutoDrop option for any concurrent groups is automatically set to the same as the schedule in which it is contained. (Any existing setting of the option on the concurrent group type is ignored.) So, in 1.0.2, if you didn't set the AutoDrop flag for a concurrentGroup initialized under a schedule, it defaulted to NO or false.
- Random Library: 13, 14

  1. Reference Release-Note #13 This release includes Random v0.7, written by Sven Thommesen. This version adds several new bit generators and distributions and rearranges the library. (Ref. $SWARMHOME/src/random/docs/WHATS.CHANGED.in.0.7 and WHATS.NEW.in.0.7)
  2. Reference Release-Note #14: With the addition of the ability to make use of the default random number generators and distributions such that different runs start with a different seed or with the same seed, the static seed was changed. This means that runs with 1.0.3 will not match runs made with the static seed for 1.0.2. This is acceptable because the use of the default generators and distributions is *DEPRECATED*.

10. Porting from 1.0.0 to 1.0.1

There should not be any imcompatibilities between 1.0.1 and 1.0.0. There are a couple of changes that affect the behavior of Swarm, however. The big ones are:

- The automatic dropping of probe displays upon the dropping of an object to which those probe displays were attached. This could break your application code if you leave in the

  ```
  dropProbeDisplaysFor:
  ```

  message where you drop such an object.
- The default probeMap has changed. This is only an issue when

  ```
  createProbeDisplayFor:
  ```

  is called without having previously created and installed a probeMap for that class. The new behavior is to create a probeMap on-the-fly that contains only the instance variables for that class.

11. Beta to 1.0.0

As can be expected with any software package, it is sometimes unavoidable that changes in the functionality of the package will cause incompatibilities with earlier versions. This is especially true when a package is a "proof-of-principle" package like Swarm.

We've made an attempt to compile all the problems a user might have moving to the new release and put them here. Please read this thoroughly to decide what you might need to do to your app to get it to work with the new release.

1. The biggest and most pervasive problems will be due to the new *random* library. Notes on how to deal with this problem are provided in the Random Library *(see page 215)*.
2. A rather benign problem results from the repackaging of the *swarmobject* library. The interface to this library was brought into sync with the *defobj*and *collections* libraries. (The rest of the errant libraries will follow in a later release.) The solutions to the problems associated with this interface change are detailed in the Objectbase Library *(see page 183)*.
3. There are a couple rather benign incompatibility introduced with 1.0 in the new *activity* library. Many parts of *activity* have changed. But, for the most part, everything works exactly the same. For details on the incompatibilities please see the Activity Library *(see page 133)*. Briefly, the incompatibilities are:
   - The high-level structural changes in the activity library has led to the renaming of the variable `swarmActivity`. It is recommended that `[self getActivity]` be used in its place (if your app even used this variable, it was probably in the *BatchSwarm*). This is a new message and it takes the place of the `getSwarmActivity` message. But, the obsolete message has been left in place for backwards compatibility.
   - The `getCurrentActivity()` macro is gone. If you used the old getCurrentActivity() in your code, it won't work now. Use of this macro was not very widespread, since its main use is to access activity library internals. One of the other macros should be sufficient for any application.
4. The functionality of `Zone`s has been greatly improved and expanded upon. Most of the aspects of the idea behind `Zone`s are now in place. However, there is one incompatibility that must be noted in case your code is fairly old. The dropFrom: message has been removed. Even though it was still present in recent releases, its behavior was identical to drop. Any existing usage should be replaced by a simple drop message without any zone argument. SwarmObject subclasses are now restricted from accessing the zone that was once contained in an instance variable; the message getZone must be used instead.

Defobj Library

Overview

The defobj library supports the style of object-oriented programming that is used throughout Swarm. It defines a specific style for using the Objective C language that includes its own standard conventions for creating objects and for storage allocation, error handling, and debugging support.

1. Dependencies

The defobj library is defined and documented using the Library Interface Conventions *(see page 406)* established by the defobj library. It imports the Object superclass and other standard type definitions of the GNU Objective C runtime system.

The collections library must always be linked along with the defobj library. Even though the interface definitions of defobj do not depend directly on collections, the implementations of these libraries both require the presence of the other. Only the collections library should be initialized directly; initialization of the collections library automatically initializes the defobj library as well.

2. Compatibility

No explicit compatibility issues for particular versions of Swarm

3. Usage Guide

3.1. Why Swarm uses Objective C

Swarm uses object-oriented programming not only because this is a good way to build general-purpose software libraries, but because the very concept of an object is at the base of how you build a model in Swarm. To build a simulation in Swarm, the first step to define the various kinds of objects that can inhabit some real or artificial world, and the second step is to define the kinds of events that can occur to these objects, including all the different ways that the objects can interact with each other.

The basic concept of an object is that it responds to external events, and that the only thing that really matters about an object are the ways it can be observed responding to these events. In an object-oriented programming system, an object is represented by some uniquely identified chunk of data, and the events are the dynamic operations that can be made to occur on these objects.

Different object-oriented programming systems define the operations that occur on objects in different ways. For example, in C++ the operations are called "member functions" and look much like an ordinary function call in the C language. In Objective C, an operation on an object is called a "message" and has a special syntax by which you call it, but a message also has arguments and behaves in many respects like a call to a function.

The key requirement for Swarm is to be able to make all these operations happen to any object at any time, whenever they're supposed to occur within some simulated world. It stores these operations inside

its own data structures, and when you run a simulation in Swarm, the Swarm system itself traverses these data structures to make the necessary operations happen at the proper time.

Swarm needs to be able to make any kind of action happen to any kind of object at any time, and to do this it needs very general-purpose structures that hold some kind of representation of the actions themselves. That's a special requirement that not every object language provides. You already have enough work to define the kinds of objects that can exist inside your model, and Swarm doesn't want you to have to define an even larger number of objects for every kind of event that might occur to your objects. Instead, it wants the object language to provide it with a representation of the operations on objects that you've already defined. Swarm can then store these representations in its own data structures, and make the operations happen whenever it needs to.

In Swarm, the representation of events that happen on objects are just as fundamental to the model as the objects themselves. That's why Swarm is a "discrete-event" simulation system. But if the object system doesn't provide a general-purpose enough representation of events as well as objects, there would be a lot more work to do. The Objective C system, through its special data type called a "message selector," provides a representation of operations that is flexible enough to do this, but the C++ language does not. (In C++, the compiler requires more information about an operation than the event structures in Swarm would be able to provide.)

There's a host of additional reasons why Objective C has also been a good match for the requirements of the Swarm system. Like C++, the operations on objects can be compiled to a very efficient form (though Objective C does require a little more overhead than C++ to get the operation started). This efficiency can be very important for a simulation, since simulations can run for very long periods of time to explore all the behavior that might occur within their simulated worlds. Other languages, such as Lisp, Smalltalk, and Java, also have the "dynamic message dispatch" feature that would make general-purpose event structures possible, but they still carry significant added overhead compared to C.

Objective C is also a very simple extension to the C language. Basic knowledge of C is already widespread, and a few days are typically all that is needed to learn the few additions of Objective C. (Really learning the concepts of objects, however, can take much longer, no matter what object language you try to use.)

Objective C was also a good choice for Swarm because it has a high quality, freely distributable implementation in the GNU C compiler. One of the main concerns about Objective C is that it isn't nearly as widely known or used as other languages like C++, but at least the GNU C compiler assures it will be available on machines where Swarm needs to run. The OpenStep system now part of the Apple Rhapsody project (formerly NeXTStep of the NeXT corporation), and the parallel GNUStep project, also help assure that Objective C is a living language.

There are even more powerful aspects of Objective C that Swarm takes advantage of. Some of these are described in the Advanced Usage Guide *(see page 29)* of the defobj library. But many others are at the heart of how Swarm uses Objective C, and so are dealt with in the rest of this Usage Guide. The entire purpose of the defobj library is to define a standard style for the use of Objective C in Swarm. This style is backed up by a library of foundation classes that Swarm provides to support this style.

Swarm provides its own foundation classes even for such basic operations as creating an object, and other support that user classes ordinarily receive from a builtin Object superclass. A good understanding of the defobj library is essential for Objective C programming in Swarm. Like any programming language, Objective C requires learning not only the rules of the language itself, but also a standard library of initial capabilities that you build from. Objective C doesn't really have a single official

standard library (or language definition either, for that matter), but the OpenStep foundation libraries (and the OpenStep language definition) come very close to this status.

For a variety of reasons explained here and in the Advanced Usage Guide *(see page 29)*, Swarm doesn't use a standard library based on the NextStep foundation interface. You can still learn the Objective C language from other available sources, but you have to be careful to sort out the language level from the standard object and library support they also discuss.

The index page of the Swarm documentation provides a variety of links to other Objective C resources. In particular, it provides a link to a complete on-line reference for the Objective C language as defined by OpenStep (formerly NextStep) and implemented by the GNU C compiler. None of the Swarm documentation attempts to duplicate this coverage of the basic language; Swarm assumes that you learn Objective C from other available sources. The Objective C reference (*Object-Oriented Programming and the Objective C Language (http://devworld.apple.com/techpubs/rhapsody/ObjectiveC/index.html)*) is the single best available source, and it can also be ordered as a hardcopy book from the Rhapsody Developer Documentation site *(http://devworld.apple.com/techpubs/rhapsody/rhapsody.html)* . Don't try to program in Objective C without it.

3.2. Swarm style of Objective C Programming

No matter where you've learned to program in Objective C, don't try to use Objective C in Swarm without understanding the special ways in which Swarm adapts its use of the language and the language runtime system. Your place to find this information is right here in the defobj library. Defobj not only provides the most basic layer of foundation class libraries for Swarm, but also serves as the place where all the rules and guidelines for the use of Objective C in Swarm are gathered together.

Some of these rules and guidelines don't even require any specific software to implement them, but are just conventions and recommended style that anyone programming in Objective C can follow when their goals are similar to those of Swarm, and to a large extent often do. These kinds of conventions are the focus of this section; following sections discuss aspects of Swarm Objective C style that depend on specific software support.

More to come... For now, see Library Interface Conventions *(see page 406)* for some of this information, and Swarm tutorials for much of the rest.

4. Advanced Usage Guide

Empty

5. Subclassing Reference

The defobj library currently includes the CreateDrop and Object_s superclasses used by other libraries. For user objects in Swarm simulations the SwarmObject superclass in the objectbase library packages all required behavior of these superclasses into a single, simpler superclass.

The defobj library also provides support for custom-generated classes, such as those which implement separate phases of the standard create protocol. (See generated phase classes in the Advanced Usage Guide.) Subclassing from custom-generated classes is still not officially supported, because the framework for custom class generation is still being finalized. Custom-generated classes are currently

used only in the defobj, collections, and activity libraries; these libraries document the specific classes that provide subclassing support.

In general, classes that implement a library are not automatically available for use as superclasses. Any library must document those specific superclasses that are valid for user classes to subclass from, along all the rules that a user subclass must follow when implementing new behavior.

6. Interface Design Notes

Nothing

7. Implementation Notes

Nothing

Documentation and Implementation Status

The current interfaces defined in defobj are expected to remain stable, with the exception of those relating to package structure and custom-generated classes. Those facilities are in process of being totally replaced, and will not be documented further until that replacement is complete. Details of support for defining a customized type will also be provided later.

All the interfaces defined in the header file are currently implemented, except for the various types of zones that would hold all allocated storage in a collection of local pages, and that would provide automatic reclaim of unused storage. The current zone implementation supports all defined allocation messages, and also maintains the population of the zone that consists of all objects created directly within it.

The Usage Guide section is only a start, but it does include subsections that discuss the style of Objective C programming adopted for use throughout Swarm. There is a full set of Interface Reference sections, though some details remain to be filled in. Throughout the documentation, a parenthesized comment that starts with (.. indicates an editorial comment on the current status of implementation or documentation.

Revision History

2004-07-16 defobj.h *schristley*

([FArguments -addObject:]): Definition conflicts with NSArray, so use NSArray definition for GNUstep.

2003-09-03 defobj.h *alex*

Use '//E:' syntax instead of '//M:' to tag examples which are intended to be inline elements of code. In this way ampersands which cause problems for the XML DocBook backend are escaped properly.

2003-05-10 defobj.h *pauljohn*

inserted headers and explanation for all methods listed in next entries.

2002-01-02 defobj.h *mgd*

(raiseEvent): Avoid concatenation to __FUNCTION__.

2001-12-17 defobj.h *mgd*

Remove const from COMOBJECT.

2001-10-10 defobj.h *mgd*

Declare getDatasetType.

2001-02-23 defobj.h *mgd*

(fcall_type_t): Add fcall_type_jselector. (FCALL_TYPE_COUNT): Increment.

2001-01-28 defobj.h *mgd*

(val_t): Moved from objectbase.h. (LanguageJS): New symbol.

2001-01-27 defobj.h *mgd*

(fcall_type_t): Add fcall_type_iid. (FCALL_TYPE_COUNT): Increment. (types_t): Add iid.

2001-01-24 defobj.h *mgd*

Use const void * for COMOBJECT.

2001-01-05 defobj.h *mgd*

(callTypes): Add JScall.

2000-12-17 defobj.h *mgd*

Add getLastArgIndex.

2000-10-14 defobj.h *mgd*

(DefinedObject): Add -conformsTo:.

2000-09-25 defobj.h *mgd*

(types_t): Move to be visible to C++.

2000-09-24 defobj.h *mgd*

(fcall_type_t): Swap fcall_type_ulonglong and fcall_type_slonglong. (FCALL_TYPE_COUNT): Add.

2000-09-23 defobj.h *mgd*

Add preprocessor hair to make fcall_type_t available to C++.

2000-09-22 defobj.h *mgd*

Add SaveWarning.

2000-09-13 defobj.h *mgd*

(arguments): Add <Arguments> qualifier.

2000-09-13 defobj.h *mgd*

(LanguageCOM, LanguageJava, LanguageObjc): Add.

2000-09-12 defobj.h *mgd*

(COMOBJECT): New typedef.

2000-08-15 defobj.h *mgd*

(Arguments): Call setArgc argument count "count" and make it unsigned.

2000-07-24 defobj.h *mgd*

(Arguments): Fix declaration of setInhibitExecutableSearchFlag:. Move getReclaimPolicy and getStackedSubzones to design document.

2000-07-16 defobj.h *mgd*

(Archiver): Adopt RETURNABLE.

2000-07-11 defobj.h *mgd*

(GSTRDUP): Add.

2000-06-29 defobj.h *mgd*

(CreatedClass): Remove return type from updateArchiver:.

2000-06-22 defobj.h *mgd*

(types_t): Make _long_double of that type.

2000-05-18 defobj.h *mgd*

(GetName): Move before DefinedObject. (DefineObject): Adopt it. (DefinedClass): Don't adopt it (it's inherited). (HDF5): Remove -getName.

2000-04-27 defobj.h *mgd*

([Arguments +createBegin, -createEnd]): Remove. ([HDF5 +createBegin:, -creatEnd, -drop]): Remove. ([{HDF5CompoundType,FArguments,FCall} +createBegin: -createEnd]): Remove.

2000-03-28 *mgd*

Swarmdocs 2.1.1 frozen.

2000-03-25 defobj.h *mgd*

Remove PTRFMT.

2000-03-24 defobj.h *mgd*

(Zone): Note that dropping a zone doesn't drop block allocations, only objects.

2000-02-29 *mgd*

Swarmdocs 2.1 frozen.

2000-02-18 defobj.h *mgd*

(FArguments, FCall): Add Zone argument conformance to +create:* methods.

2000-01-22 defobj.h *mgd*

(Zone): Remove containsAlloc:.

2000-01-19 defobj.h *mgd*

Don't declare generate_class_name.

1999-12-21 defobj.h *mgd*

(types_t): Add boolean.

1999-11-19 defobj.h *mgd*

New types call_t and JOBJECT.

1999-10-29 defobj.h *mgd*

Documentation updates.

1999-08-22 defobj.h *mgd*

Add (id <Zone>) argument and return types. Reorganize. Add (id <HDF5>) argument types.

1999-08-09 defobj.h *mgd*

(types_t): Add Class.

1999-08-08 defobj.h *mgd*

(types_t): Add _long_double.

1999-08-05 defobj.h *mgd*

(ZSTRDUP, SSTRDUP, OSTRDUP, STRDUP, OFREEBLOCK, ZFREEBLOCK): New macros.

1999-06-28 defobj.h *alex*

Reflect Archiver changes. Make `Archiver' an abstract protocol. (HDF5Archiver, LispArchiver): New protocols now conform to and CREATABLE.

1999-06-13 defobj.h *mgd*

(types_t): Add signed and unsigned types.

1999-06-09 defobj.h *alex*

(initDefobj): Declare to accept new `appName' argument.

1999-06-08 defobj.h *alex*

(Archiver): Add ([Archiver_c +create:from{Lisp,HDF5}Path:]) method to protocol.

1999-06-05 defobj.h *alex*

(Archiver): Add method [Archiver_c getWithZone:object:] to protocol. Document all existing methods.

1999-05-29 defobj.h *mgd*

Import externvar.h.

1999-05-28 defobj.h *mgd*

Use `externvar' for external variable declarations.

1999-05-20 defobj.h *alex*

Declare extern {lisp,hdf5}AppArchiver. (Archiver): Add -setDefaultApp{Lisp,HDF5}Path to protocol definition.

1999-04-29 defobj.h *mgd*

(GetName): Add instance -getName, remove +getName:.

1999-04-22 defobj.h *mgd*

Clarify docs on addRef:withArgument:. (Zone): Switch return type of getPageSize from int to size_t.
(FArguments): Add using phase tag before getResult.

1999-04-21 defobj.h *mgd*

(MAKE_PARSE_FUNCTION_NAME): New macro.

1999-04-21 defobj.h *mgd*

(Serialization): Move -lispIn: and -hdf5In: to setting phase. (FArguments): Add.

1999-04-16 defobj.h *mgd*

(types_t): New typedef.

1999-03-23 defobj.h *vjojic*

(FCall): Mark phases in FCall protocol.

1999-03-23 defobj.h *mgd*

(FCall): Add getReturnVal.

1999-03-17 defobj.h *vjojic*

Add new protocol FCall.

1999-02-27 defobj.h *mgd*

Put all setters needed for createEnd to creating phase. Remove duplicate setAppModeString: in setting.

1999-02-20 defobj.h *mgd*

Disable CreateDrop protocol; it is already an @interface.

1999-02-19 defobj.h *vjojic*

Add CreateDrop protocol.

1999-02-16 defobj.h *alex*

(generate_class_name): Prefix with `extern'.

1999-01-14 defobj.h *mgd*

(DSIZE): New macro for sizing decimal scratch buffers.

1999-01-12 defobj.h *mgd*

(HDF5): New protocol.

1999-01-10 defobj.h *mgd*

(Serialization): Add deep: option to lispOut and hdf5Out. (Archiver): Add deep argument to lispArchiverPut, hdf5ArchvierPut.

1999-01-10 defobj.h *mgd*

(LoadError, SaveError): New error types.

1999-01-08 defobj.h *mgd*

(DefinedClass): Declare lispInCreate:, lispIn:, lispOut:, updateArchiver, and copyClass.

1999-01-06 defobj.h *mgd*

(Arguments): Declare +createBegin:, -createEnd, -setArgc:Argv:, -setAppModeString:, -setOptionFunc:, -setBugAddress:, and -setVersion:. Move addOptions: to creating phase.

1999-01-06 defobj.h *mgd*

(DefinedClass): Declare addVariable. (Serialization): Declare updateArchiver.

1998-12-28 defobj.h *mgd*

(Archiver, Serialization): Add protocol summary and description strings.

1998-12-21 defobj.h *mgd*

(archiver{Register,Unregister}, {lisp,HDF5}Archiver{Get,Put}): Prefix declaration with `extern'.

1998-12-19 defobj.h *mgd*

(Archiver): New protocol. Move archiver functions inside this protocol declaration.

1998-12-18 defobj.h *mgd*

Add archiver{Register,Unregister}, and {HDF5,lisp}Archiver{Get,Put} to Serialization protocol.

1998-12-18 defobj.h *mgd*

(Symbol): Remove setName:.

1998-12-17 defobj.h *mgd*

Remove readOnly accessors from SetInitialValue protocol (moved to design document).

1998-11-17 defobj.h *mgd*

(Serialization): New protocol. Put lispIn, lispInQuotedExpr, and archvierSave here. Put lispInCreate: in creating phase. Rename lisp{in,out}: to lisp{In,Out}:. Declare defobj_lookup_type.

1998-11-13 defobj.h *mgd*

(DefinedObject): Declare +lispin:expr: in creating phase. (lispinQuotedExpr): Declare. (MAKE_OBJC_FUNCTION_NAME): Define (was confined to Archiver.m).

1998-11-12 defobj.h *mgd*

(Dataset): New protocol. (Arguments): Add protocol (from objectbase).

1998-07-22 defobj.h *mgd*

Replace @deftype with @protocol throughout.

1998-06-18 defobj00.sgml *alex*

Put CITETITLE tag around reference to the `Object-Oriented Programming and the Objective C Language' volume.

1998-06-17 Makefile.am *mgd*

Include from refbook/ instead of src/.

1998-06-15 Makefile.am *mgd*

(MODULE): New variable. Include Makefile.rules from src. Remove everything else.

1998-06-14 defobj.h *mgd*

Remove mention of what might happen with other (nonexistent) zone types. Remove mention of status of current Zone implementation. Remove mention of -setReclaimPolicy:, -setStackedSubzones:, -getSubzones, -mergeWithOwner, -getSubzone:. Remove -reclaimStorage, -releaseStorage, xfprint, and xfprintid declarations. Improve description of xsetname, xprint, xprintid, xfprint, xfprintid, xexec, and xfexec.

1998-06-12 defobjpages.sgml, defobjcont.sgml *mgd*

Update IDs to SWARM.module.SGML.type.

1998-06-06 defobj.ent *mgd*

Use public identifiers.

1998-06-05 Makefile.am *mgd*

(swarm_ChangeLog): Add.

1998-06-03 defobj.h *mgd*

Updated documentation tags. (PTRFMT): Use %p.

1998-06-01 defobj.h *alex*

([DefinedObject -{xfprint,xfprintid}): Added method and doc tags that should exist to DefinedObject protocol.

1998-05-28 defobj.h *mgd*

Fix //# doc strings (following colon required).

1998-05-26 defobj.ent.in *mgd*

Make defobjrevhistory be a build-directory path.

1998-05-26 defobj.ent.in *alex*

Added entity (defobjrevhistory) for the automatically generated revision history.

1998-05-26 defobj.h *alex*

(raiseEvent,M(), initModule, globalZone, scratchZone, defsymbol, defwarning, deferror): Added (//#) doc strings. (<{Warning,Error}>): Added (//G) doc strings after each definition. (<Symbol>, _obj_formatIDString,

objc_get_class, _obj_debug, (_obj_xerror, *_obj_xdebug, xsetname, xprint, xprintid, xfprint, xfprintid, xexec, xfexec): Added (//G) doc strings before each definition.

1998-05-23 Makefile.am *mgd*

New file.

1998-05-23 defobj.ent.in *mgd*

New file.

1998-05-23 defobj.ent *mgd*

Removed.

1998-05-22 *mgd*

Begin revision log.

1998-05-06 defobj.h *mgd*

Remove instances of <p> in the documentation. Minor changes to method spacing. (Create): Move some documentation from createBegin: to be general documentation. (Error): Add a description.

1998-05-04 defobj.h *mgd*

Add CREATING and USING tags where absent.

1998-04-27 defobj.h *mgd*

Add documentation tags.

1998-04-17 defobj.h *mgd*

(GetName): getName is factory method. (DefinedClass): All methods are factory methods.

1998-01-27 defobj.h *mgd*

Declare nameToObject function.

1997-12-08 defobj.h *mgd*

Constify argument to setDisplayName. Constify return of getDisplayName. Constify return of getName in GetName deftype. Constify setName argument to Symbol +create. Constify argument to Symbol setName. Constify argument to Warning setMessageString. Constify return of Warning getMessageString. Constify argument to CreatedClass setName. Constify argument to objc_get_class. Constify name argument to xsetname, exec, and xfexec. Drop APIChange warning. Reformatting throughout.

1997-12-08 defobj.h *mgd*

Reenable LibraryUsage, DefaultAssumed, and ObsoleteFeature. Put back ObsoleteMessage (gepr argues they are important for a minor release).

1997-12-07 defobj.h *mgd*

Add APIChange to the standard error types. Delete ObsoleteMessage, since it appears to be redundant with ObsoleteFeature. Disable LibraryUsage, DefaultAssumed, and ObsoleteFeature because they aren't used.

Archiver

Name

`Archiver` — High level abstract serialization interface.

Description

High level abstract serialization interface.

Protocols adopted by Archiver

Create *(see page 46)*

Drop *(see page 54)*

RETURNABLE *(see page 66)*

Methods

Phase: Creating

- - **setDefaultAppPath**

  Specify that the Archiver to use the default application path
- - **setDefaultPath**

  Specify that the Archiver instance use the default system path
- - **setSystemArchiverFlag:** (BOOL)*systemArchiverFlag*

  Make the Archiver expect application metadata, such as `mode' information
- - **setPath:** (const char *)*path*

  Set the physical path for the Archiver to read/write
- - **setInhibitLoadFlag:** (BOOL)*inhibitLoadFlag*

  Make the Archiver ignore any file found in the specified path

Phase: Using

- - (void)**sync**

  Ensure that that all registered the requested backend
- - **getWithZone:** (id <Zone>)*aZone* **key:** (const char *)*key*

  Create the object with `key' in the specified Zone
- - **getObject:** (const char *)*key*

  Create the object with `key' using the Archiver's own Zone
- - (void)**putShallow:** (const char *)*key* **object:** *object*

As per -putDeep, but only make a shallow version

- - (void)**putDeep:** (const char *)*key* **object:** *object*

  Register with the Archiver a deep serialization of the object (serialization only occurs when Archiver is saved)

- - (void)**unregisterClient:** *client*
- - (void)**registerClient:** *client*

Arguments

Name

`Arguments` — A class that provides customizable command line argument parsing support

Description

A class that provides customizable command line argument parsing support

Protocols adopted by Arguments

Create *(see page 46)*

Drop *(see page 54)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- `- (int)`**`parseKey:`** `(int)`*`key`* **`arg:`** `(const char *)`*`arg`*

  This method is called for each option that occurs.

- `+` **`createArgc:`** `(int)`*`argc`* **`Argv:`** `(const char **)`*`argv`* **`appName:`** `(const char *)`*`appName`* **`version:`** `(const char *)`*`version`* **`bugAddress:`** `(const char *)`*`bugAddress`* **`options:`** `(struct argp_option *)`*`options`* **`optionFunc:`** `(int (*)(int, const char *))`*`optionFunc`* **`inhibitExecutableSearchFlag:`** `(BOOL)`*`inhibitExecutableSearchFlag`*

- `- (void)`**`addOption:`** `(const char *)`*`name`* **`key:`** `(int)`*`key`* **`arg:`** `(const char *)`*`arg`* **`flags:`** `(int)`*`flags`* **`doc:`** `(const char *)`*`doc`* **`group:`** `(int)`*`group`*

  Takes an option specification that includes the following information:

  - The name of the option specification

  - The key of the option. This an integer that, if printiable, is the single-character use of the option. For example, `-p' vs. `--protocol' are the different versions of the same thing. One is intended to be mnemonic, the other convenient.

  - If non-NULL, an argument label that says that the option requires an argument (in this case, the protocol name).

  - Flags that change the visibility and parsing of the option

  - Documentation for the option

  - A sorting integer; relative placement of the option in the help screen.

- `-` **`addOptions:`** `(struct argp_option *)`*`options`*

- `-` **`setVersion:`** `(const char *)`*`version`*

- - **setBugAddress:** (const char *)*bugAddress*
- - **setOptionFunc:** (int (*) (int, const char *))*optionFunc*
- - **setAppModeString:** (const char *)*appModeString*
- - **setAppName:** (const char *)*appName*
- - **setArgc:** (unsigned)*count* **Argv:** (const char **)*theArgv*

Phase: Setting

- - **setFixedSeed:** (unsigned)*seed*
- - **setDefaultAppDataPath:** (const char *)*path*

  Specify a default path to use for data files when installed location of Swarm cannot be determined. Defaults to current directory.
- - **setDefaultAppConfigPath:** (const char *)*path*

  Specify a default path to use for configuration files when installed location of Swarm cannot be determined. Defaults to current directory.
- - **setVerboseFlag:** (BOOL)*verboseFlag*
- - **setVarySeedFlag:** (BOOL)*varySeedFlag*
- - **setBatchModeFlag:** (BOOL)*batchModeFlag*
- - **setInhibitExecutableSearchFlag:** (BOOL)*theInhibitExecutableSearchFlag*
- - **setInhibitArchiverLoadFlag:** (BOOL)*inhibitArchiverLoadFlag*

Phase: Using

- - (BOOL)**getInhibitArchiverLoadFlag**
- - (BOOL)**getShowCurrentTimeFlag**
- - (const char *)**getAppConfigPath**

  A path where application-specific configuration files can be expected to be found.
- - (const char *)**getAppDataPath**

  A path where application-specific data files can be expected to be found.
- - (const char *)**getDataPath**
- - (const char *)**getConfigPath**
- - (const char *)**getSwarmHome**
- - (const char *)**getExecutablePath**
- - (const char **)**getArgv**
- - (int)**getLastArgIndex**
- - (int)**getArgc**
- - (const char *)**getAppModeString**
- - (const char *)**getAppName**

- - (BOOL)**getVerboseFlag**
- - (unsigned)**getFixedSeed**
- - (BOOL)**getFixedSeedFlag**
- - (BOOL)**getVarySeedFlag**
- - (BOOL)**getBatchModeFlag**

Examples

Example #1

```
Let's say you want to add a new argument, say `protocol' to your standard
list of commands.  In other words you want the following to happen at the
command line when you type --help.
------------------------
mgd@wijiji[/opt/src/mgd/src/mySwarmApp] $ ./mySwarmApp --help
Usage: mySwarmApp [OPTION...]

  -s, --varyseed             Select random number seed from current time
  -S, --seed=INTEGER         Specify seed for random numbers
  -b, --batch                Run in batch mode
  -m, --mode=MODE            Specify mode of use (for archiving)
  -p, --protocol=PROTOCOL    Set protocol
  -?, --help                 Give this help list
      --usage                Give a short usage message
  -V, --version              Print program version

Mandatory or optional arguments to long options are also mandatory or
optional for any corresponding short options.

Report bugs to bug-swarm@swarm.org.
------------------------

To implement this you need to make your own subclass of Arguments
like the following:

#import <defobj/Arguments.h>

@interface MySwarmAppArguments: Arguments_c
{
  const char *protocolArg;
}
- (const char *)getProtocolArg;
@end

@implementation MySwarmAppArguments

+ createBegin: (id <Zone>)aZone
{
  static struct argp_option options[] = {
    {"protocol", 'p', "PROTOCOL", 0, "Set protocol", 3},
    { 0 }
```

```
  };

  MySwarmAppArguments *obj = [super createBegin: aZone];

  [obj addOptions: options];
  return obj;
}

- (int)parseKey: (int)key arg: (const char *)arg
{
  if (key == 'p')
    {
      protocolArg = arg;
      return 0;
    }
  else
    return [super parseKey: key arg: arg];
}

- (const char *)getProtocolArg
{
  return protocolArg;
}

@end
```

To actually invoke this in the main.m program, you do the following:

```
int
main (int argc, const char ** argv)
{
  initSwarmArguments (argc, argv, [MySwarmAppArguments class]);

  // the usual - buildObjects:, - buildActions:, - activateIn: calls

  return 0;
}
```

BehaviorPhase

Name

`BehaviorPhase` — Created class which implements a phase of object behavior.

Description

Created class which implements a phase of object behavior.

Protocols adopted by BehaviorPhase

CreatedClass *(see page 48)*

Methods

Phase: Creating

- `- (void)` **`setNextPhase:`** *`aClass`*

Phase: Using

- `-` **`getNextPhase`**

CREATABLE

Name

`CREATABLE` — Declare that a defined type supports creation.

Description

Declare that a defined type supports creation.

Protocols adopted by CREATABLE

None

Methods

None

Copy

Name

`Copy` — Copy all state defined as part of object.

Description

An object type that supplies the copy operation defines what it includes as the contents of an object copied. There is no global rule for what is considered "inside" a copied object vs. merely referenced by it. (There is no fixed notion of "shallow" vs. "deep" copy found in some object libraries.) After copying, the new object may still contain some references to other elements also referenced by the starting object, but in general the new object minimizes any dependencies shared with the starting object. Any object type supplying the copy message should also supply documentation on its rules for copied objects.

Protocols adopted by Copy

None

Methods

Phase: Using

- `- **copy:** (id <Zone>)`*`aZone`*

  The copy message creates a new object that has the same contents and resulting behavior as a starting object, except that an independent copy of the contents of the starting object is created so that further changes to one object do not affect the other. The zone argument specifies the source of storage for the new object. The message returns the id of the new object created.

Create

Name

`Create` — Create an instance of a type with optional customization.

Description

The Create supertype defines standard messages that provide a general-purpose protocol for creating new objects. These messages may be used either to create a new instance of a type in one message, or to bracket a series of messages that customize available options for an object to be created. The separation of create-time specifications from later behavior of an object gives substantial flexibility to adapt a generic type to particular needs.

These create messages may be implemented either by a type that hides its implementing classes, or directly by a class that adopts these messages as a uniform interface for creating objects. If implemented directly by a class, then the class object serves as the type object in all message descriptions that follow. Otherwise, a type object might be implemented in a variety of ways that guarantee only that published messages on a type are accepted.

In addition to the create messages defined by Create, an object type may support any other messages of any other names or calling conventions. These messages define only a standard method for creating new objects that other types are free to inherit and implement in conformance with a uniform convention. Further conventions are established elsewhere create combination messages for standard ways in which create messages that combine several steps can be combined.

Any interim object returned by either createBegin: or customizeBegin: supports the getZone and drop messages that a finalized instance may also support. These messages are defined by the Drop type, which is normally inherited by a type to declare these messages on a finalized instance. This type is not inherited by the Create type because the messages would then apply to the finalized instance, not to the interim object. Even though not declared, the messages are available on the interim objects nonetheless. The drop message on an interim object may be used if it turns out that a finalized version is no longer required after creation or customization has already begun.

The createBegin: and createEnd messages bracket a series of messages that specify options for an object being created. The intermediate messages can set values defined as parameters of the type, or provide other forms of specification using available messages. A particular object type defines the specific messages that are valid for sending during this interim creation phase.

Protocols adopted by Create

DefinedObject *(see page 52)*

Customize *(see page 49)*

Methods

Phase: Creating

- `-` **`createEnd`**

The createEnd message completes the process of specifying available options for an object being created. Typically it validates that requested options are valid and consistent with one another, and raises an error if they are not. The standard, predefined error InvalidCombination may be raised by createEnd to indicate an invalid combination of requests, or other, more specific forms of error handling may be used.

If all requests received since the initial createBegin: are valid, both individually and in combination with each other, then createEnd determines a finalized form of object that satisfies all requests received and then returns this object. Any additional storage required for the finalized object is taken from the same zone originally passed to createBegin. The object may have whatever implementation is selected to best satisfy a particular request. Different requests may result in entirely different implementations being returned. The only guarantee is that a returned object supports the messages defined for further use of the finalized object. If a type was defined by a @protocol declaration, these messages are those appearing in either the SETTING or USING sections.

On return from createEnd, the id of the interim object returned by createBegin: is no longer guaranteed to be valid for further use, and should no longer be referenced. A variable which holds this such an id can be reassigned the new id returned by createEnd, so that the same variable holds successive versions of the object being created.

- + **createBegin:** (id <Zone>)*aZone*

  createBegin: returns an interim object intended only for receiving create-time messages. If a type was defined by a @protocol declaration, these messages are those appearing in either the CREATING or SETTING sections. Otherwise, the messages valid as create-time messages are defined by the type without any specific syntactic marker.

- + **create:** (id <Zone>)*aZone*

  The create: message creates a new instance of a type with default options. The zone argument specifies the source of storage for the new object. The receiving object of this message is a previously defined type object. The message is declared as a class message (with a + declaration tag) to indicate that the message is accepted only by the type object itself rather than an already created instance of the type (which a - declaration tag otherwise defines).

  The create: message returns the new object just created. This object is an instance of some class selected to implement the type. The class which a type selects to implement an object may be obtained by the getClass message, but is not otherwise visible to the calling program. A caller never refers to any class name when creating objects using these messages, only to type names, which are automatically published as global constants from any @protocol declaration.

Examples

Example #1

```
newArray = [Array createBegin: aZone];
[newArray setInitialValue: aList];
```

```
[newArray setDefaultMember: UnsetMember];
[newArray setCount: [aList getCount] * 2 );
newArray = [newArray createEnd];   // ! note reassignment of newArray
```

CreatedClass

Name

`CreatedClass` — Class with variables and/or methods defined at runtime.

Description

Class with variables and/or methods defined at runtime.

Protocols adopted by CreatedClass

Create *(see page 46)*

DefinedClass *(see page 51)*

Methods

Phase: Creating

- - (void)**updateArchiver:** (id <Archiver>)*archiver*
- - (void)**hdf5OutShallow:** (id <HDF5>)*hdf5Obj*
- - (void)**lispOutShallow:** *stream*
- - **hdf5InCreate:** *hdf5Obj*
- - **lispInCreate:** *expr*
- - **at:** (SEL)*aSel* **addMethod:** (IMP)*aMethod*
- - **setDefiningClass:** *aClass*
- - **setSuperclass:** *aClass*
- - **setClass:** (Class)*aClass*
- - **setName:** (const char *)*name*

Phase: Using

- - **getDefiningClass**

Customize

Name

`Customize` — Create-phase customization.

Description

Some types accept create-time messages not only when creating a new instance, but to customize a new version of the type itself. Objects created from a customized type will have all options preset that create-time messages sent to the customized type object have already set. If many objects all need the same create-time options, it is often simpler (and can also be faster) to create a customized version of a type first, and then create further instances from that type.

Customizing a type object does not modify the original type object, but instead creates a new type object that has the customizations built-in. A create: message on the new type object creates a new instance as if the same sequence of create-time messages had been sent to the original type object using createBegin: and createEnd. A type is customized by bracketing the sequence of create-time messages not with the createBegin: and createEnd messages used to create a new instance, but with customizeBegin: and customizeEnd messages instead.

Whether a customized version of a type can be created depends on the implementation of the type itself. If a type does not support customization, a customizeBegin: message on the type raises an error. All types defined by an @protocol declaration may be relied on to support at least one cycle of customization to create a new type object. Whether an already customized type object (returned by customizeEnd) supports a further cycle of customization (by another sequence of customizeBegin:/customizeEnd) depends on the implementation of the original starting type. A type should not be relied on to support more than one cycle of customization unless it is specifically documented to do so.

Protocols adopted by Customize

None

Methods

Phase: Creating

- `-` **`customizeCopy:`** *`aZone`*

  The customizeCopy: message creates a new copy of the interim object returned by customizeBegin: which may be used for further customizations that do not affect the customization already in progress. It may be used to branch off a path of a customization in progress to create an alternate final customization.

  customizeCopy may be used only on an interim object returned by customizeBegin: and not yet finalized by customizeEnd. The new version of the interim object being customized may be allocated in the same or different zone as the original version, using the zone argument required by customizeCopy:

Example -customizeCopy: #1

```
newArrayType1 = [Array customizeBegin: aZone];
[newArrayType1 setCount: 100];
newArrayType2 = [newArrayType2 customizeCopy: aZone];
[newArrayType2 setDefaultMember: UnsetMember];

newArrayType1 = [newArrayType1 customizeEnd];
 newArrayType2 = [newArrayType2 customizeEnd];
array1 = [newArrayType1 create: aZone];  // no DefaultMember option
array2 = [newArrayType create: aZone];   // DefaultMember option set
```

- \- **customizeEnd**

  Returns the new, customized version of the original type.

- \+ **customizeBegin:** *aZone*

  Returns an interim value for receiving create-time messages much like createBegin:.

  The zone passed to customizeBegin: is the same zone from which storage for the new, finalized type object will be taken. This zone need not be the same as any instance later created from that type, since a new zone argument is still passed in any subsequent create message on that type.

Examples

Example #1

```
newArrayType = [Array customizeBegin: aZone];
[newArrayType setCount: 100];
newArrayType = [newArrayType customizeEnd];
array1 = [newArrayType create: aZone];
array2 = [newArrayType create: aZone];
// [array1 getCount] and [array2 getCount] are both 100
```

DefinedClass

Name

`DefinedClass` — Class which implements an interface of a type.

Description

Class which implements an interface of a type.

Protocols adopted by DefinedClass

DefinedObject *(see page 52)*

Methods

Phase: Using

- `+ (IMP)` **`getMethodFor:`** `(SEL)` *`aSel`*
- `+` **`getTypeImplemented`**
- `+ (void)` **`setTypeImplemented:`** *`aType`*
- `+ (BOOL)` **`isSubclass:`** *`aClass`*
- `+` **`getSuperclass`**

DefinedObject

Name

`DefinedObject` — Object with defined type and implementation.

Description

DefinedObject is the top-level supertype for all objects that follow the object programming conventions of the defobj library. The messages defined by this type are the only messages which should be assumed to be automatically available on objects that follow these conventions. In particular, use of messages defined by the Object superclass of the GNU Objective C runtime should not generally be assumed because future implementations of some objects might not give continued access to them.

The DefinedObject type defines a minimum of standard messages, and leaves to other types the definition of message that might or might not apply in any general way to particular objects.

Protocols adopted by DefinedObject

GetName *(see page 60)*

Methods

Phase: Using

- `- (id <Zone>)`**`getZone`**

  The getZone message returns the zone in which the object was created.

- `- (void)`**`xfprintid`**

  print id for each member of a collection on debug output stream

- `- (void)`**`xfprint`**

  print description for each member of a collection on debug output stream

- `- (void)`**`xprintid`**

  Like describeID:, but output goes to standard output.

- `- (void)`**`xprint`**

  Like describe:, but output goes to standard output.

- `- (void)`**`describeID:`** *`outputCharStream`*

  Prints a one-line describe string, consisting of the built-in default to outputCharStream.

- `- (void)`**`describe:`** *`outputCharStream`*

The describe: message prints a brief description of the object for debug purposes to the object passed as its argument. The object passed as the outputCharStream argument must accept a catC: message as defined in String and OutputStream in the collections library. Particular object types may generate object description strings with additional information beyond the built-in default, which is just to print the hex value of the object id pointer along with the name of its class, and the display name of the object, if any.

- - (const char *)**getDisplayName**

  Return a string that identifies an object for external display purposes, either from a previously assigned string or an identification string default

- - (void)**setDisplayName:** (const char *)*displayName*

  Assigns a character string as a name that identifies an object for display or debug purposes.

- - **perform:** (SEL)*aSel*

  A local implementation of an Object method.

- - **perform:** (SEL)*aSel* **with:** *anObject1*

  A local implementation of an Object method.

- - **perform:** (SEL)*aSel* **with:** *anObject1* **with:** *anObj2*

  A local implementation of an Object method.

- - **perform:** (SEL)*aSel* **with:** *anObject1* **with:** *anObj2* **with:** *anObj3*

  Perform a selector with three object arguments.

- - (int)**compare:** *anObject*

  A local implementation of an Object method.

- - (void)**removeRef:** (ref_t)*refVal*

  Remove an external reference to an object.

- - (ref_t)**addRef:** (notify_t)*notifyFunction* **withArgument:** (void *)*arg*

  Adds an external reference to an object that is notified when a an object is dropped.

- - (const char *)**getTypeName**

  getTypeName returns the name of the originating type of this object.

- - (Class)**getClass**

  getClass returns the class that implements the current behavior of an object.

- + (BOOL)**conformsTo:** (Protocol *)*protocol*

- - (BOOL)**conformsTo:** (Protocol *)*protocol*

- - (BOOL)**respondsTo:** (SEL)*aSel*

The respondsTo: message returns true if the object implements the message identified by the selector argument. To implement a message means only that some method will receive control if the message is sent to the object. (The method could still raise an error.) The respondsTo: message is implemented by direct lookup in a method dispatch table, so is just as fast as a normal message send. It provides a quick way to test whether the type of an object includes a particular message.

Drop

Name

`Drop` — Deallocate an object allocated within a zone.

Description

The Drop supertype defines the drop message, which is a standard message for indicating that an object no longer exists and will never again be referenced. Any future attempt to reference a dropped object is an error. This error may or not produce predictable effects depending on the level of debug checking and other factors.

Protocols adopted by Drop

None

Methods

Phase: Using

- `- (void)`**`drop`**

  Immediate effects of the drop message depends on the subtype of Zone used to provide storage for the object. For some zone types, the drop message immediately deallocates storage for the object and makes the freed storage available for other use. Subsequent use could include the allocation of a new object at precisely the same location, resulting in a new object id identical to a previously dropped one.

  The Drop type may be inherited by any type that provides drop support for its instances. In addition to freeing the storage and invalidating the object, a drop message may release other resources acquired or held within the object. Not every object which can be created can also be dropped, and some objects can be dropped which are not directly creatable. Some objects may be created as a side effect of other operations and still be droppable, and some objects may be created with links to other objects and not droppable on their own. A type independently inherits Create or Drop types, or both, to indicate its support of these standard interfaces to define the endpoints of an object lifecycle.

Error

Name

`Error` — A condition which prevents further execution.

Description

A condition which prevents further execution.

Protocols adopted by Error

Warning *(see page 74)*

CREATABLE *(see page 44)*

Methods

None

Macros

- **`deferror`**`(`*`name`*`, `*`message`*`)`

  `macro used to create and initialize a Warning symbol`

Globals

id <Error> SourceMessage

message in the source defines error

id <Error> NotImplemented

requested behavior not implemented by object

id <Error> SubclassMustImplement

requested behavior must be implemented by subclass

id <Error> InvalidCombination

invalid combination of set messages for create

id <Error> InvalidOperation

invalid operation for current state of receiver

id <Error> InvalidArgument

argument value not valid

id <Error> CreateSubclassing

improper use of Create subclassing framework

id <Error> CreateUsage

incorrect sequence of Create protocol messages

id <Error> OutOfMemory

no more memory available for allocation

id <Error> InvalidAllocSize

no more memory available for allocation

id <Error> InternalError

unexpected condition encountered in program

id <Error> BlockedObjectAlloc

method from Object with invalid allocation

id <Error> BlockedObjectUsage

method inherited from Object superclass

id <Error> ProtocolViolation

object does not comply with expected protocol

id <Error> LoadError

unable to access a resource

id <Error> SaveError

unable to save a resource

EventType

Name

`EventType` — A report of some condition detected during program execution.

Description

A report of some condition detected during program execution.

Protocols adopted by EventType

Symbol *(see page 73)*

Methods

Phase: Using

- `- (void)`**`raiseEvent`**

  Raise an event noting the event symbol type.

- `- (void)`**`raiseEvent:`** `(const void *)`*`eventData`* **`:`** `...`

  Raise an event noting the event symbol type using a format string and arguments.

Macros

- **`raiseEvent`**`(`*`eventType`*`, `*`formatString`*`, `*`args...`*`)`

  macro to raise Warning or Error with source location strings

FArguments

Name

`FArguments` — A language independent interface to dynamic call argument construction.

Description

A language independent interface to dynamic call argument construction.

Protocols adopted by FArguments

Create *(see page 46)*

Drop *(see page 54)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- - **setBooleanReturnType**
- - **setReturnType:** (fcall_type_t)*retType*
- - **setObjCReturnType:** (char)*type*
- - **addJavaObject:** (JOBJECT)*obj*
- - **addSelector:** (SEL)*aSel*
- - (void)**addObject:** *obj*
- - **addString:** (const char *)*value*
- - **addLongDouble:** (long double)*value*
- - **addDouble:** (double)*value*
- - **addFloat:** (float)*value*
- - **addUnsignedLongLong:** (unsigned long long)*value*
- - **addLongLong:** (long long)*value*
- - **addUnsignedLong:** (unsigned long)*value*
- - **addLong:** (long)*value*
- - **addUnsigned:** (unsigned)*value*
- - **addInt:** (int)*value*
- - **addUnsignedShort:** (unsigned short)*value*
- - **addShort:** (short)*value*
- - **addUnsignedChar:** (unsigned char)*value*

- - **addBoolean:** (BOOL)*value*
- - **addChar:** (char)*value*
- - **addArgument:** (void *)*value* **ofObjCType:** (char)*type*
- - **addArgument:** (types_t *)*value* **ofType:** (fcall_type_t)*type*
- + **create:** (id <Zone>)*aZone* **setSelector:** (SEL)*aSel*
- - **setJavaSignature:** (const char *)*javaSignature*
- - **setSelector:** (SEL)*aSel*

  The selector is used to set argument types. Some languages won't have any, and so for those languages this need not be called.
- - **setLanguage:** (id <Symbol>)*languageType*

Phase: Using

- - (void *)**getResult**
- - (id <Symbol>)**getLanguage**
- - (val_t)**getRetVal**

FCall

Name

`FCall` — A language independent interface to dynamic calls.

Description

A language independent interface to dynamic calls.

Protocols adopted by FCall

Create *(see page 46)*

Drop *(see page 54)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- `- **setJavaMethodFromName:** (const char *)*methodName* **inClass:** (const char *)*className*`
- `- **setJavaMethodFromName:** (const char *)*methodName* **inObject:** (JOBJECT)*jObj*`
- `- **setMethodFromName:** (const char *)*methodName* **inObject:** *object*`
- `- **setMethodFromSelector:** (SEL)*method* **inObject:** *object*`
- `- **setFunctionPointer:** (func_t)*fn*`
- `- **setArguments:** *args*`
- `+ **create:** (id <Zone>)*aZone* **target:** *obj* **methodName:** (const char *)*methodName* **arguments:** (id <FArguments>)*fa*`
- `+ **create:** (id <Zone>)*aZone* **target:** *obj* **selector:** (SEL)*sel* **arguments:** (id <FArguments>)*fa*`

Phase: Using

- `- (func_t)**getFunctionPointer**`
- `- (retval_t)**getRetVal:** (retval_t)*retVal* **buf:** (types_t *)*buf*`
- `- (void *)**getResult**`
- `- (void)**performCall**`
- `- **getArguments**`
- `- (call_t)**getCallType**`

GetName

Name

`GetName` — Get name which identifies object in its context of use.

Description

Get name which identifies object in its context of use.

Protocols adopted by GetName

None

Methods

Phase: Using

- `- (const char *)`**`getName`**

  The getName message returns a null-terminated character string that identifies an object in some context of use. This message is commonly used for objects that are created once in some fixed context where they are also assigned a unique name. Constant objects defined as part of a program or library are examples. This message is intended only for returning a name associated with an object throughout its lifetime. It does not return any data that ever changes.

GetOwner

Name

`GetOwner` — Get object on which existence of object depends.

Description

Ownership hierarchies arrange themselves in a strict, single-rooted tree. The top-level node of an ownership hierarchy typically returns nil as its owner. If an object is regarded merely as one part of another object defined as its owner, then copying or dropping the owner object should copy or drop the member object as well. Owner and member are neutral terms for a generic relationship sometimes called parent vs. child, but it is up to a particular object type to define specifically what it means by a getOwner relationship.

Protocols adopted by GetOwner

None

Methods

Phase: Using

- \- **`getOwner`**

  The getOwner message returns another object which is considered as the owner of an initial object. What is considered as an owner depends on its specific object type, but might be a larger object of which the local object is a part, or an object that has exclusive control over the local object. The principal constraint established by an ownership structure is that a given object can have only a single other object as its unambiguous owner.

HDF5

Name

HDF5 — HDF5 interface

Description

HDF5 interface

Protocols adopted by HDF5

Create *(see page 46)*

Drop *(see page 54)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- - **setCount:** (unsigned)*count*
- - **setCompoundType:** *compoundType*
- - **setParent:** *parent*
- - **setExtensibleDoubleVector**
- - **setExtensibleVectorType:** (fcall_type_t)*type*
- - **setDatasetFlag:** (BOOL)*datasetFlag*
- - **setWriteFlag:** (BOOL)*writeFlag*

Phase: Setting

- - **setBaseTypeObject:** *baseTypeObject*
- - **setName:** (const char *)*name*

  Create-time use is to name the file or group. Setting-time use is to rename component datasets that don't parent's name.

Phase: Using

- - (void)**flush**
- - (const char *)**getAttribute:** (const char *)*attributeName*
- - (void)**iterateAttributes:** (int (*) (const char *key, const char *value))*iterateFunc*
- - (void)**storeAttribute:** (const char *)*attributeName* **value:** (const char *)*valueString*

- - (void)**writeLevels**
- - (void)**writeRowNames**
- - (const char **)**readRowNames**
- - (void)**selectRecord:** (unsigned)*recordNumber*
- - (void)**numberRecord:** (unsigned)*recordNumber*
- - (void)**nameRecord:** (unsigned)*recordNumber* **name:** (const char *)*recordName*
- - (void)**shallowStoreObject:** *obj*
- - (void)**shallowLoadObject:** *obj*
- - (void)**storeComponentTypeName:** (const char *)*typeName*
- - (void)**storeTypeName:** (const char *)*typeName*
- - (void)**addDoubleToVector:** (double)*val*
- - (void)**storeAsDataset:** (const char *)*name* **typeName:** (const char *)*typeName* **type:** (fcall_type_t)*type* **rank:** (unsigned)*rank* **dims:** (unsigned *)*dims* **ptr:** (void *)*ptr*
- - (void)**loadDataset:** (void *)*ptr*
- - (BOOL)**checkDatasetName:** (const char *)*datasetName*
- - (BOOL)**checkName:** (const char *)*name*
- - (void)**assignIvar:** *obj*
- - (Class)**getClass**
- - **getCompoundType**
- - (const char *)**getHDF5Name**
- - (unsigned)**getCount**
- - (fcall_type_t)**getDatasetType**
- - (size_t)**getDatasetDimension:** (unsigned)*dimNumber*
- - (size_t)**getDatasetRank**
- - (BOOL)**getWriteFlag**
- - (BOOL)**getDatasetFlag**
- - (void)**iterate:** (int (*) (id <HDF5>hdf5Obj))*iterateFunc*
- - (void)**iterate:** (int (*) (id <HDF5>hdf5Obj))*iterateFunc* **drop:** (BOOL)*dropFlag*

HDF5Archiver

Name

`HDF5Archiver` — Protocol for creating HDF5 instances of the Archiver

Description

Protocol for creating HDF5 instances of the Archiver Default system path is ~/swarmArchiver.hdf Default application path is <swarmdatadir>/<appname>/<appname>.hdf or the current directory.

Protocols adopted by HDF5Archiver

Archiver *(see page 38)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- `+` **`create:`** `(id <Zone>)`*`aZone`* **`setPath:`** `(const char *)`*`path`*

  `Convenience method to create an HDF5Archiver from a specified path`

HDF5CompoundType

Name

`HDF5CompoundType` — HDF5 composite type interface

Description

HDF5 composite type interface

Protocols adopted by HDF5CompoundType

Create *(see page 46)*

Drop *(see page 54)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- \- **`setPrototype:`** *`prototype`*

Phase: Using

- \- **`getPrototype`**

LispArchiver

Name

`LispArchiver` — Protocol for creating Lisp instances of the Archiver

Description

Protocol for creating Lisp instances of the Archiver Default system path is ~/.swarmArchiver.scm. Default application path is <swarmdatadir>/<appname>/<appname>.scm or the current directory.

Protocols adopted by LispArchiver

Archiver *(see page 38)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- `+` **`create:`** `(id <Zone>)`*`aZone`* **`setPath:`** `(const char *)`*`path`*

  `Convenience method to create LispArchiver from a specified path`

RETURNABLE

Name

`RETURNABLE` — Declare that a defined type may be created as a side-effect

Description

Declare that a defined type may be created as a side-effect

Protocols adopted by RETURNABLE

None

Methods

None

Serialization

Name

`Serialization` — Object serialization protocol.

Description

Object serialization protocol.

Protocols adopted by Serialization

None

Methods

Phase: Creating

- - **hdf5InCreate:** (id <HDF5>)*hdf5Obj*

  Process HDF5 object to set create-time parameters.

- - **lispInCreate:** *expr*

  Process keyword parameters in expression in order to get create-time parameters.

Phase: Setting

- - **hdf5In:** (id <HDF5>)*hdf5Obj*

  Load instance variables from an HDF5 object.

- - **lispIn:** *expr*

  Process an archived Lisp representation of object state from a list of instance variable name / value pairs.

Phase: Using

- - (void)**updateArchiver:** (id <Archiver>)*archiver*
- - (void)**hdf5OutDeep:** (id <HDF5>)*hdf5obj*

  Output a deep HDF5 representation of object state to a stream.

- - (void)**hdf5OutShallow:** (id <HDF5>)*hdf5obj*

  Output a shallow HDF5 representation of object state to a stream.

- - (void)**lispStoreDoubleArray:** (double *)*ptr* **Keyword:** (const char *)*keyword* **Rank:** (unsigned)*rank* **Dims:** (unsigned *)*dims* **Stream:** *stream*

  Lisp save array of doubles, see lispStoreIntegerArray:Keyword:Rank:Dims.

- - (void)**lispStoreFloatArray:** (float *)*ptr* **Keyword:** (const char *)*keyword* **Rank:** (unsigned)*rank* **Dims:** (unsigned *)*dims* **Stream:** *stream*

  Lisp save array of floats, see lispStoreIntegerArray:Keyword:Rank:Dims.

- - (void)**lispStoreUnsignedLongLongArray:** (unsigned long long *)*ptr* **Keyword:** (const char *)*keyword* **Rank:** (unsigned)*rank* **Dims:** (unsigned *)*dims* **Stream:** *stream*

  Lisp save array of unsigned long long, see lispStoreIntegerArray:Keyword:Rank:Dims.

- - (void)**lispStoreLongLongArray:** (long long *)*ptr* **Keyword:** (const char *)*keyword* **Rank:** (unsigned)*rank* **Dims:** (unsigned *)*dims* **Stream:** *stream*

  Lisp save array of long long, see lispStoreIntegerArray:Keyword:Rank:Dims.

- - (void)**lispStoreUnsignedLongArray:** (unsigned long *)*ptr* **Keyword:** (const char *)*keyword* **Rank:** (unsigned)*rank* **Dims:** (unsigned *)*dims* **Stream:** *stream*

  Lisp save array of unsigned long, see lispStoreIntegerArray:Keyword:Rank:Dims.

- - (void)**lispStoreLongArray:** (long *)*ptr* **Keyword:** (const char *)*keyword* **Rank:** (unsigned)*rank* **Dims:** (unsigned *)*dims* **Stream:** *stream*

  Lisp save array of long, see lispStoreIntegerArray:Keyword:Rank:Dims.

- - (void)**lispStoreUnsignedArray:** (unsigned *)*ptr* **Keyword:** (const char *)*keyword* **Rank:** (unsigned)*rank* **Dims:** (unsigned *)*dims* **Stream:** *stream*

  Lisp save array of unsigned integers, see lispStoreIntegerArray:Keyword:Rank:Dims.

- - (void)**lispStoreShortArray:** (short int *)*ptr* **Keyword:** (const char *)*keyword* **Rank:** (unsigned)*rank* **Dims:** (unsigned *)*dims* **Stream:** *stream*

  Lisp save array of short integers, see lispStoreIntegerArray:Keyword:Rank:Dims.

- - (void)**lispStoreCharArray:** (char *)*ptr* **Keyword:** (const char *)*keyword* **Rank:** (unsigned)*rank* **Dims:** (unsigned *)*dims* **Stream:** *stream*

  Lisp save array of characters, see lispStoreIntegerArray:Keyword:Rank:Dims.

- - (void)**lispStoreBooleanArray:** (BOOL *)*ptr* **Keyword:** (const char *)*keyword* **Rank:** (unsigned)*rank* **Dims:** (unsigned *)*dims* **Stream:** *stream*

  Lisp save array of Booleans, see lispStoreIntegerArray:Keyword:Rank:Dims.

- - (void)**lispStoreIntegerArray:** (int *)*ptr* **Keyword:** (const char *)*keyword* **Rank:** (unsigned)*rank* **Dims:** (unsigned *)*dims* **Stream:** *stream*

  For customized archiving of dynamically allocated arrays within objects. To use this method, override the object's lispOutDeep: method as follows. The array is assumed allocated in one long piece of memory, but it can be treated in segments to make it two dimensional. The number of rows is "rank" and the length of the i'th row is dims[i], thus allowing for a ragged array. In this example, Attribute is a class, culture is a dynamically allocated array of numDims integers, so rank is 1 and a pointer to numDims is passed through for the length of that row.

  Example -lispStoreIntegerArray:Keyword:Rank:Dims:Stream: #1

```
- (void)lispOutDeep: stream
{
 [stream catStartMakeInstance: "Attribute"];
```

```
  [super lispOutVars: stream deep: NO]; //saves all ints, doubles, BOOLs,
and static arrays. Saves the values of all other objects as nil
  // Note one can use the previous "lispSaveStream:..." methods to
  // customize the choice of variables to be saved.
  // Now save an array called "culture", which has 1 row
  // and "numDims" columns, onto the stream.
  [super lispStoreIntegerArray: culture Keyword: "culture" Rank: 1 Dims:
&numDims Stream: stream];
  [stream catEndMakeInstance];
```

}

- - (void)**lispSaveStream:** *stream* **Double:** (const char *)*aName* **Value:** (double)*val*

  On the given stream, save a double valued variable called "aName" which has value "val".

- - (void)**lispSaveStream:** *stream* **Float:** (const char *)*aName* **Value:** (double)*val*

  On the given stream, save a float valued variable called "aName" which has value "val".

- - (void)**lispSaveStream:** *stream* **UnsignedLongLong:** (const char *)*aName* **Value:** (unsigned long long)*val*

  On the given stream, save an unsigned long long variable called "aName" which has value "val".

- - (void)**lispSaveStream:** *stream* **LongLong:** (const char *)*aName* **Value:** (long long)*val*

  On the given stream, save a long long variable called "aName" which has value "val".

- - (void)**lispSaveStream:** *stream* **UnsignedLong:** (const char *)*aName* **Value:** (unsigned long)*val*

  On the given stream, save an unsigned long variable called "aName" which has value "val".

- - (void)**lispSaveStream:** *stream* **Long:** (const char *)*aName* **Value:** (long)*val*

  On the given stream, save a long variables called "aName" which has value "val".

- - (void)**lispSaveStream:** *stream* **Unsigned:** (const char *)*aName* **Value:** (unsigned)*val*

  On the given stream, save an unsigned integer variable called "aName" which has value "val".

- - (void)**lispSaveStream:** *stream* **Integer:** (const char *)*aName* **Value:** (int)*val*

  On the given stream, save an integer variable called "aName" which has value "val".

- - (void)**lispSaveStream:** *stream* **UnsignedShort:** (const char *)*aName* **Value:** (unsigned short)*val*

  On the given stream, save an unsigned short integer variable called "aName" which has value "val".

- - (void)**lispSaveStream:** *stream* **Short:** (const char *)*aName* **Value:** (short)*val*

  On the given stream, save a short integer variable called "aName" which has value "val".

- - (void)**lispSaveStream:** *stream* **Char:** (const char *)*aName* **Value:** (char)*val*

  On the given stream, save a character variable called "aName" which has value "val".

- - (void)**lispSaveStream:** *stream* **Boolean:** (const char *)*aName* **Value:** (int)*val*

On the given stream, save a Boolean variable called "aName" which has value "val". Explanation: The Swarm lisp serialization approach assumes that objects have lispOutDeep: and lispOutShallow: methods which indicate which variables are supposed to be saved. If an object is subclassed from SwarmObject, there are default lispOutDeep: and lispOutShallow: methods. Those methods employ on the method, lispOutVars:deep:, which is the "default" approach to try to save all variables, either deep or shallow. Sometimes one needs to selectively list particular instance variables to be saved.

This is necessary, for example, if one wants to save a Swarm itself, because the usage of lispOutVars: will result in a variable "activity" being saved as nil, and so when the saved values are read back in, the "activity" variable will be erased and nil will appear in its place.

Here is an example of how a subclass called "BFagent" might override lispOutDeep: to customize the selection of variables to be saved. Note that the same could be used to override lispOutShallow:.

The key thing to remember is that when one tries to do a deep save on a high level object, such as a Swarm, then the Swarm libraries will try to track from top to bottom, finding all collections and objects, and all objects and collections inside them, and so forth, and each will be told to execute its lispOutDeep: method. So all objects you want to save need a lispOutDeep: method, or else the default will try to save all variables. If you omit some objects or variables from your lispOutDeep: method, then they will not appear in the saved file, which is what you want if you want to be sure that pre-existing interited values of variables are not obliterated by bogus saved values.

Example -lispSaveStream:Boolean:Value: #1

```
- (void)lispOutDeep: stream
{
  [stream catStartMakeInstance: "BFagent"];
  [self lispSaveStream: stream Double: "demand" Value: demand];
  [self lispSaveStream: stream Double: "profit" Value: profit];
  [self lispSaveStream: stream Double: "wealth" Value: wealth];
  [self lispSaveStream: stream Double: "position" Value: position];
  [self lispSaveStream: stream Double: "cash" Value: cash];
  [self lispSaveStream: stream Double: "price" Value: price];
  [self lispSaveStream: stream Double: "dividend" Value: dividend];
  [self lispSaveStream: stream Integer: "myID" Value: myID];
  [stream catEndMakeInstance];

}
```

An example of such a usage can be found in version 2.4 of the

Artificial Stock Market (http://ArtStkMkt.sourceforge.net).

- - (void)**lispOutVars:** *stream* **deep:** (BOOL)*deepFlag*

  Output just key/variable pairs, where variables are serialized deep or shallow per deepFlag.

- - (void)**lispOutDeep:** *stream*

  Output a deep Lisp representation of object state to a stream.

- - (void)**lispOutShallow:** *stream*

  Output a shallow Lisp representation of object state to a stream.

SetInitialValue

Name

SetInitialValue — Create using initial value from an existing object.

Description

The SetInitialValue type defines a variety of messages relating to an initial or unmodifiable value established as part of an object. This message is typically provided when creation of a new object might be more easily accomplished by copying the value of an existing object rather than establishing a new value from scratch. As with the copy message, precisely what is considered the value of an existing object to copy is defined only by the particular object type that supplies these messages.

If an object has a value which can be established at create time, it is often useful (and can also enable significant optimization) to declare that no further modification will occur to this value during further use of the object. A restriction against modifying a value is referred to as a "read-only" restriction. This type supplies messages to declare a read-only restriction along with any initial value. For some object types, a read-only restriction can also be added or removed after an object has already been created.

Protocols adopted by SetInitialValue

None

Methods

Phase: Creating

- - (void)**setInitialValue:** *initialValue*

  The setInitialValue: message requires another object as its argument, from which the value of a newly created object is to be taken. Unlike a copy message, the object used as the source of the new value need not have the identical type as the new object to be created. A particular object type defines the types of initial value objects which it can accept, along with any special conversion or interpretation it might apply to such a value.

Symbol

Name

`Symbol` — Object defined as a distinct global id constant.

Description

A Symbol is an object created with a fixed name. It has no behavior except to get the name with which it was created. A Symbol is typically used to define unique id values which are assigned to global constant names. These names, capitalized according to the recommended convention for global object constants, are used by some libraries as flags or enumerated value codes in arguments or return values of messages.

Ordinarily, a symbol is created with its character string name matching the global id constant to which it is assigned. These global program constants can then provide a minimal level of self documentation as objects. Subtypes of Symbol can extend the base of a named, global id constant to establish further components of a global, constant definition.

A symbol is fully creatable using standard Create messages. A character string name must be supplied for any new symbol; there is no default. Symbol inherits the getName message, which returns the symbol name.

Protocols adopted by Symbol

Create *(see page 46)*

GetName *(see page 60)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- `+` **`create:`** `(id <Zone>)`*`aZone`* **`setName:`** `(const char *)`*`name`*

  `create:setName: is a combination message defined as a caller convenience. See combination messages for a summary of conventions on combination messages.`

Macros

- **`defsymbol`**`(`*`name`*`)`

  `macro used to create and initialize a symbol`

Warning

Name

`Warning` — A condition of possible concern to a program developer.

Description

A condition of possible concern to a program developer.

Protocols adopted by Warning

EventType *(see page 56)*

CREATABLE *(see page 44)*

Methods

Phase: Using

- `- (const char *)`**`getMessageString`**

  `Return the message associated with this warning.`

- `- (void)`**`setMessageString:`** `(const char *)`*`messageString`*

  `Associate a message string with this warning.`

Macros

- **`defwarning`**`(`*`name`*`, `*`message`*`)`

  `macro used to create and initialize an Error symbol`

Globals

id <Warning> WarningMessage

message in the source defines warning

id <Warning> ResourceAvailability

resource from runtime environment not available

id <Warning> LibraryUsage

invalid usage of library interface

id <Warning> DefaultAssumed

non-silent use of default

id <Warning> ObsoleteFeature

using feature which could be removed in future

id <Warning> ObsoleteMessage

using message which could be removed in future

id <Warning> SaveWarning

non-fatal problem saving a resource

Zone

Name

`Zone` — Modular unit of storage allocation.

Description

A zone is a source of storage for objects or other allocated data. Whenever a new object is created, a zone must be identified from which the storage for its instance variables, or other internal data, is obtained. A program may establish multiple zones to ensure that objects with similar lifetime or storage needs are allocated together, and in general to optimize allocation and reuse of storage.

Zones also maintain a collection of all objects allocated within the zone. This collection, referred to as the "population" of a zone, is a set of all objects which have been created but not yet dropped within the zone. Collections maintained automatically by zones can eliminate a need for other, separately maintained collections in applications that need to keep track of entire populations of objects. Collections of allocated objects can provide support for object query, external object storage, and automatic storage reclamation.

A zone may be used to obtain storage not only for objects, but also for raw storage blocks like those provided by the C malloc function. All objects and storage blocks allocated in a zone remain local to that zone. This means that allocation of storage in other zones does not affect the efficiency of storage allocation within a particular zone. For most zone types, individual allocations may still be freed within a zone, and total storage of a zone may grow and shrink according to aggregate needs. In addition to freeing individual allocations, an entire zone may also dropped. Dropping a zone automatically frees all object allocations made within it, including final drop processing on any allocated objects that need it. Release of an entire zone can be much faster than individual release of each object within it.

The Zone type is a fully implemented type that provides default storage management support for objects and other allocated storage. It is also a supertype for other zones that implement alternative policies for use in specialized situations.

A zone is created using standard create messages just like other objects. This means that a zone must identify another zone from which it obtains its storage. Storage is typically obtained from this other zone in large units called pages, which are then managed by the local zone to support internal allocations. The getZone message of the DefinedObject type returns the zone which provides these base pages.

Since a new zone always requires that an existing zone be identified, no new zones could be created unless there were some zones that already existed. Two such zones are predefined as part of the defobj library: globalZone and scratchZone.

Protocols adopted by Zone

Create *(see page 46)*

Drop *(see page 54)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- - (void)**setPageSize:** (size_t)*pageSize*

  PageSize specifies the size of pages within which a zone manages its internal allocation. Its default is typically a natural page size (perhaps 4K) for the local machine architecture. The default should be overridden only when tuning storage allocation for very specific situations. Allocations within a zone are not limited to the page size, since any requests that exceed the page size are simply passed up to the owner zone within which the zone was allocated.

Phase: Using

- - (void)**describeForEachID:** *outputCharStream*

  Generate debug id description for each member of the zone population.

- - (void)**describeForEach:** *outputCharStream*

  Generate debug description for each member of the zone population.

- - (id <List>)**getPopulation**

  getPopulation returns a collection all objects allocated in a zone using either allocIVars: or copyIVars: and not yet freed using freeIVars:. getObjects returns nil if the ObjectCollection option is false. The collection returned has the type OrderedSet as defined in the collections library, with the ReadOnly option set true and the IndexSafety option set to SafeAlways. The members of this collection may change as objects are allocated and freed, but may not added or removed directly within the collection.

- - (void)**freeBlock:** (void *)*aBlock* **blockSize:** (size_t)*size*

  freeBlock:blockSize: must be used to free any block previously allocated by allocBlock:.

- - (void *)**allocBlock:** (size_t)*size*

  allocBlock: allocates a new storage block similar to alloc:, except that the size of the block allocated must be passed as an argument when freeing the block.

- - (void)**free:** (void *)*aBlock*

  free: releases a block of storage previously allocated using alloc:. The size of the block is not required as an argument because alloc: has saved this size as necessary as part of the initial allocation. free: may be used only to free a block allocated by alloc:, and a block allocated by alloc: may be freed only by free:.

- - (void *)**alloc:** (size_t)*size*

  alloc: allocates a new storage block much like the malloc function of the C library. The storage is aligned according to the most restrictive requirements for any data type on the local machine architecture. The storage is not initialized to any known contents.

- - **getComponentZone**

  Returns a specially qualified version of the zone that automatically allocates all its objects with the internal component qualification, even if allocated with allocIVars: or copyIVars:. This qualified zone may be passed as an argument to a create: or createBegin: message so that it will create the new object as an internal component object.

- - (void)**freeIVarsComponent:** *anObject*

  Frees the instance variable storage for an object.

- - **copyIVarsComponent:** *anObject*

  Like allocateIVarsComponent, except it copies the storage that holds the instances variables for an object.

- - **allocIVarsComponent:** (Class)*aClass*

  These messages allocate, copy, and free

  This message allocates the storage that holds the instance variables for an object. It allocates the object as an internal component of the zone that is not included in the zone population. It is used by classes that allocate additional objects as part of the implementation of another object, and that control the mapping of this storage separately from the zone level objects.

- - (void)**freeIVars:** *anObject*

  freeIVars: releases storage that was previously allocated to hold the instance variable structure of an object. The first word of the object must be a class pointer that correctly describes the size of the structure. Storage allocated by allocIVars: or copyIVars: may be freed only by freeIVars:, and freeIVars: may be used only to free storage allocated by one of these messages.

- - **copyIVars:** *anObject*

  copyIVars: creates copies an existing instance variable structure into a new allocation made within the local zone. The existing instance variable structure may be in any zone, but must contain a class pointer in its first word that correctly describes the size of the structure.

- - **allocIVars:** (Class)*aClass*

  allocIVars: allocates the instance variable structure for a new object. The initial word of this structure is set to class id passed as its argument. The class also determines the size of the structure allocated. All remaining contents of this structure are initialized to binary zeroes.

- - (size_t)**getPageSize**

General

Name

defobj — Standard objects for GNU Objective C extensions

Description

The defobj library supports the style of object-oriented programming that is used throughout Swarm. It defines a specific style for using the Objective C language that includes its own standard conventions for creating objects and for storage allocation, error handling, and debugging support.

Macros

- **DSIZE**(*type*)

  Conservative approximation of the number of decimal digits for a object of a given type, not including terminator. signchar + roundup (log (10)/log(2) = 3.3219).

- FCALL_TYPE_COUNT
- **FREEBLOCK**(*block*)
- **GSTRDUP**(*str*)
- **M**(*messageName*)

  Abbreviation for @selector().

- MAKE_CLASS_FUNCTION_NAME

Name to use for Lisp archiving class-creation function

- MAKE_INSTANCE_FUNCTION_NAME

Name to use for Lisp archiving object-creation function

- **OFREEBLOCK**(*obj, block*)
- **OSTRDUP**(*obj, str*)
- PARSE_FUNCTION_NAME

Name to use for Lisp archive custom-parse function

- **SFREEBLOCK**(*block*)
- **SSTRDUP**(*str*)
- **STRDUP**(*str*)
- **ZFREEBLOCK**(*aZone, block*)
- **ZSTRDUP**(*aZone, str*)
- __swarm_defobj_h
- globalZone

A zone for allocating global objects.

- **initModule**(*module*)

  module initialization macro

- scratchZone

A zone for allocating temporary objects.

Functions

- void **_obj_formatIDString**(char *_buffer_, id _anObject_)

  Function to generate object id string in standard format (Up to 78 characters of the supplied buffer argument could be filled.)

- void **_obj_initModule**(void *_module_)

  internal module initialization function

- id **defobj_lookup_type**(const char *_name_)

  Lookup a defobj type object by name.

- void **initDefobj**(id <Arguments> _arguments_)

- id **nameToObject**(const char *_name_)

  Get an object from textual pointer description.

- Class **objc_get_class**(const char *_name_)

  Declaration to enable use of @class declaration for message receiver without compile error.

- void **xexec**(id _anObject_, const char *_name_)

  Debug function to perform message on an object.

- void **xfexec**(id _anObject_, const char *_name_)

  Debug function to perform message on each member of a collection.

- void **xfprint**(id _anObject_)

  Print description for each member of a collection on debug output stream.

- void **xfprintid**(id _anObject_)

  Print id for each member of a collection on debug output stream.

- void **xprint**(id _anObject_)

  Print description of object on debug output stream.

- void **xprintid**(id _anObject_)

  Print only the id string for an object on debug output stream.

- void **xsetname**(id _anObject_, const char *_name_)

  Set the display name.

- char ***zstrdup**(id <Zone> _aZone_, const char *_str_)

Typedefs

- COMOBJECT void *
- JOBJECT void *
- call_t enum callTypes { ccall, COMcall, JScall, javacall, javastaticcall, objccall }

- fcall_type_t enum {fcall_type_void = 0, fcall_type_boolean, fcall_type_uchar, fcall_type_schar, fcall_type_ushort, fcall_type_sshort, fcall_type_uint, fcall_type_sint, fcall_type_ulong, fcall_type_slong, fcall_type_ulonglong, fcall_type_slonglong, fcall_type_float, fcall_type_double, fcall_type_long_double, fcall_type_object, fcall_type_class, fcall_type_string, fcall_type_selector, fcall_type_jobject, fcall_type_jstring, fcall_type_jselector, fcall_type_iid }
- types_t union { id object; SEL selector; Class _class; const char *string; BOOL boolean; char schar; unsigned char uchar; short sshort; unsigned short ushort; int sint; unsigned int uint; long slong; unsigned long ulong; long long slonglong; unsigned long long ulonglong; float _float; double _double; long double _long_double; void *iid; }
- val_t struct { fcall_type_t type; types_t val; }

Globals

id <Arguments> arguments

The singleton Arguments object.

id <HDF5Archiver> hdf5Archiver

The singleton HDF5 system Archiver object.

id <LispArchiver> lispArchiver

The singleton Lisp system Archiver object.

id <HDF5Archiver> hdf5AppArchiver

The singleton HDF5 application Archiver object.

id <LispArchiver> lispAppArchiver

The singleton Lisp application Archiver object.

id <Symbol> t_ByteArray

Predefined type descriptors for allocated blocks.

id <Symbol> t_LeafObject

Predefined type descriptors for allocated blocks.

id <Symbol> t_PopulationObject

Predefined type descriptors for allocated blocks.

id <Symbol> LanguageCOM

Language tags (e.g. for use in FArguments)

id <Symbol> LanguageJS

Language tags (e.g. for use in FArguments)

id <Symbol> LanguageJava

Language tags (e.g. for use in FArguments)

id <Symbol> LanguageObjc

Language tags (e.g. for use in FArguments)

id _obj_globalZone

internal variable for globalZone macro

id _obj_scratchZone

internal variable for scratchZone macro

BOOL _obj_debug

if true then perform all debug error checking

FILE * _obj_xerror

output file for error messages

FILE * _obj_xdebug

output file for debugging messages

Collections Library

Overview

The object types of the collections library establish a general-purpose foundation to maintain object references or other values as members of structured collections. Customization options consolidate a wide range of basic collection structures into a few core types (Array, List, Set, Map). These types are defined strictly by their interface, not their internal implementation. Specialized options, however, give control and flexibility for efficient, low-level use (such as implementing other libraries).

1. Dependencies

Following are the other header files imported by <collections.h>:

```
#import <defobj.h>
```

The collections library follows the Library Interface Conventions *(see page 406)* of the Defobj Library *(see page 26)*. It also depends on standard supertypes and classes defined by this library. Initialization of the collections library automatically initializes the defobj library as well. Since defobj also requires the collections library, both must always be linked into an application together.

2. Compatibility

No explicit incompatibilites for particular versions of Swarm

3. Usage Guide

This section of documentation is not yet available. In the meantime, see the GridTurtle Test Programs *(see page 404)* for the most complete examples of collections library usage. If you can't find an example there that exercises a message or option that you want to use, chances are it's not implemented.

4. Advanced Usage Guide

Unavailable

5. Subclassing Reference

Until the collections library has been fully implemented, subclassing conventions from collections implementation classes are still in flux. In general, these classes will be among the most complex uses of multiple classes selected to implement an independent object type. (See Library Interface Conventions *(see page 406)* for a summary of the distinction between types and classes.) New methods are being developed to simplify subclassing from such implementations. In the meantime, if you need to use to a collection within the implementation of your own class, just put an instance variable in your class and

put the collection in that, and pass through the messages of the collection you want to have available on your class to this variable. In many if not most cases, this is better design anyway, because you control all use of the underlying structure.

6. Interface Design Notes

A collections library is one of the most important foundation services for object-oriented programming. Most object-oriented systems provide at least the start of a general-purpose collections library. The GNUSTEP, project, for example, provides the libobject library (currently in alpha test at *(ftp://alpha.prep.ai.mit.edu/)*) which includes a collections library along with other services intended to parallel those of the OpenStep framework developed by Next.

Swarm has implemented its own collection library to meet the specialized needs of its agent simulation framework.

7. Implementation Notes

Unavailable

Documentation and Implementation Status

The collections library has an almost complete set of Interface Reference documents, and the design of the interface is almost entirely final. The implementation of the collections library, however, doesn't yet implement many of the more advanced features of this interface.

Collections library development has concentrated on the specific features needed by the activity library. This library uses collections for its underlying support of actions to be executed in schedules. The collections library itself is still being completed for use as a general-purpose library.

The Array and List types have all basic capability fully implemented. Set and Map have basic messages defined, but currently rely on a crude implementation based on sorted lists. (This implementation is more than adequate for the usually small and relatively static schedule structures in the activity library, and is the most forgiving when change does take place within members being traversed.) Much more efficient implementations based on both balanced trees and hash tables are in the works.

OrderedSet is currently supported only with the low-level option for an internal member slot, and the implemented messages do not match the ones documented in the Interface Reference. Support for groups of duplicate members is missing from both Set and Map.

None of the special options for restricted usage modes on any type of collection (e.g., read-only restriction) has been implemented. Stack and Queue are not implemented, but are nothing more than restricted uses of a List. There is no support for any member type except id or smaller (other member types will depend on a data type facility to be supplied by defobj).

Even though incomplete, the portion of capability that is implemented has been exercised very heavily. The interfaces to Set and Map structures will remain the same even as their underlying implementations improve, so there is no harm in using them. See the GridTurtle Test Programs *(see page 404)* for the most complete examples of collections usage.

The collections library follows the documentation structure suggested by the Defobj Library *(see page 26)* of the defobj library. There are placeholders at least for each section of documentation (some of which merely indicate that the section is not available yet) so that all links should at least link up with something, whether or not there's anything there.

Throughout the documentation, a parenthesized comment that starts with (.. indicates an editorial comment on the current status of implementation or documentation.

The documentation priority for all libraries is to complete at least their interface reference documents, so that there is the equivalent of Unix "man pages" that summarize all basic capability. A second priority is to complete the complementary "Usage Guide" documents. Unlike the reference documents, the Usage Guide will have a task-oriented organization, and will lead the initial user through actual code examples in the rough order a typical user is likely to need them. It will serve the role of a tutorial on each library.

The Usage Guide code examples have not yet been developed. For the time being, a directory of test programs (GridTurtle Test Programs *(see page 404)*, contained within the documentation release directory) provides code examples of many of the basic features of the defobj, collections, and activity libraries. These code examples also help indicate the portions of the libraries which are fully implemented and working, since they are run on each new release of these libraries.

Revision History

2002-05-14 collections.h *mgd*

(Array, List, Map): Adopt Serialization.

2002-01-16 collections.h *mgd*

(INDEX{STARTP,ENDP}, REMOVEDP, ARCHIVERDOTP): Cast to void * to avoid warning. (ARCHIVEREOLP): New macro.

2001-04-12 collections.h *mgd*

Remove Index safety info from Index, Array, and List. Remove EndsOnly info from ListIndex.

2001-01-24 collections.h *mgd*

(OutputStream): Use const void * for catPointer: argument.

2000-06-23 collections.h *mgd*

(Permutation): Declare setLastPermutation:.

2000-05-18 collections.h *mgd*

([Collection copy:]): Protect with #ifndef IDL. ([String compare:]): Remove (since there's no documentation, anyway).

2000-04-27 collections.h *mgd*

([Index -compare:]): Protect with #ifndef IDL. ([PermutedIndex +createBegin:, -createEnd, -next, -prev, -findNext, -findPrev, -get, -getLoc, -setLoc:, getOffset, -setOffset:]): Remove. ([ListShuffler -createEnd]): Remove. ([Permutation +createBegin:]): Remove.

2000-03-28 *mgd*

Swarmdocs 2.1.1 frozen.

2000-02-29 *mgd*

Swarmdocs 2.1 frozen.

2000-02-15 collections.h *mgd*

(_Set): Remove bogus documentation about return value of Set's add:. Thanks to Michael Stillwell.

2000-02-10 collections.h *mgd*

(Permutation): Remove -generatePermuation.

1999-09-07 collections.h *alex*

(Collection): Make -begin:, -beginPermuted: conform to Index and PermutedIndex protocol, respectively. (Index, PermutedIndex, MapIndex, ListIndex): Make RETURNABLE. Reformatting to avoid forward declarations, throughout.

1999-08-22 collections.h *mgd*

(ArchiverKeyword, ArchiverArray, ArchiverValue, ArchvierPair, ArchiverList, PermutationItem): Change from CREATABLE to RETURNABLE.

1999-08-22 collections.h *mgd*

Add objects-conforming-to-Zone argument types.

1999-08-01 collections.h *alex*

(ForEachKey): New protocol. (KeyedCollection): Adopt it.

1999-07-24 collections.h *mgd*

(_Set): Split out common Set features into subprotocol. (Set, OrderedSet): Adopt it.

1999-07-03 collections.h *mgd*

(InputStream): Declare -setLong: and -getLong.

1999-06-30 collections.h *mgd*

(OutputStream), OutputStream.[hm]: Add long long output methods.

1999-06-22 collections.h *mgd*

Reflect these changes.

1999-06-08 collections.h *alex*

(ArchiverPair): Add -{set,get}ConsFormatFlag method to protocol.

1999-06-08 collections.h *alex*

(ArchiverList): Add protocol, conform to List, CREATABLE. (OutputStream): Remove -catExpr: method. Add -cat{Short,UnsignedShort,Long,UnsignedLong}: methods.

1999-06-05 collections.h *alex*

(OutputStream): [OutputStream_c -cat{Expr, Double, Float, Int, Unsigned}:] Add and document new methods. (Archiver{Keyword,Value,Array,Pair): Add new methods to protocol.

1999-06-04 collections.h *mgd*

([Set add:]): Remove remark about DupOption (moved to design document). ([Map at:replace:]): Remove remark about potential multiple duplicate keys.

1999-05-29 collections.h *mgd*

Include externvar.h.

1999-05-28 collections.h *mgd*

Use `externvar' for external variable declarations.

1999-05-24 collections.h *alex*

(Index): -getLoc, -setLoc: Make these methods accept and return (id <Symbol>). (PermutedIndex): Likewise.

1999-05-24 collections.h *alex*

(Map): Make protocol comply with CompareFunction. Remove redundant declaration of -removeKey: (already defined in KeyedCollection protocol). (KeyedCollection): Remove compliance with CompareFunction. (Set): Re-enable compliance with KeyedCollection. (InputStream): Add docs on support for Lisp comments.

1999-01-15 collections.h *mgd*

(String): Remove setLiteralFlag: and getLiteralFlag.

1999-01-12 collections.h *vjojic*

(PermutedIndex): Declare -reshuffle.

1999-01-07 collectionsmeta.sgml *alex*

(End): Fixed missing end comment.

1999-01-06 collections.h *alex*

(PermutationItem): Add phase tags and documentation strings.

1999-01-06 collections.h *mgd*

(PermutationItem): New protocol. (PermutedIndex): Remove no-update qualification.

1998-12-28 collections.h *mgd*

Change all count arguments to unsigned. (PermutedIndex): Remove generatePermutation. (ArchiverKeyword, ArchiverArray, ArchiverValue, ArchiverPair): Add protocol summary and description strings.

1998-12-26 collections.h *mgd*

(Collection): Declare -beginPermuted:. (PermutedIndex): Declare generatePermutation.

1998-12-22 collections.h *vjojic*

Update descriptions of Permutation and PermutedIndex.

1998-12-17 collections.h *mgd*

(Index): Don't adopt Copy protocol. (KeyedCollection): Remove -createIndex:setMember: and -createIndex:at:. (ListShuffler): Remove +create:withUniformRandom:. Don't adopt CREATABLE or Create.

1998-12-14 collections.h *mgd*

(ListShuffler): Adopt Create, Drop, and CREATABLE.

1998-12-11 collections.h *vjojic*

(ListShuffler): ListShuffler protocol moved from simtools to collections

1998-12-01 collections.h *mgd*

(Index): Change example to avoid processing the End location.

1998-11-18 collections.h *mgd*

(List): The methods here aren't create-time; mark as USING phase.

1998-11-17 collections.h *mgd*

(List, Map): Adopt Serialization protocol. (ArchiverValue): Add setBoolean: and getBoolean.

1998-11-16 collections.h *mgd*

Add corresponding protocols.

1998-11-11 collections.h *mgd*

Remove creating -setDefaultMember:; there is already a setting method.

1998-11-02 collections.h *mgd*

(Collection): Note that copies are shallow.

1998-10-10 collections.h *mgd*

(Sorted): Moved to design document.

1998-09-08 collections.h *mgd*

(INDEXSTARTP, INDEXENDP, REMOVEDP, ARCHIVERLITERALP): New macros.

1998-07-22 collections.h *mgd*

Replace @deftype with @protocol throughout.

1998-07-16 collections.h *mgd*

Remove Stack and Queue (now in design document).

1998-07-09 collections.h *mgd*

(Map): Remove mention of DupOption.

1998-07-08 collections.h *alex*

(KeyedCollection): Removed to design document -getCountAtKey:, -containsKey: methods, all are unimplemented. Removed comments for -{get,set}IndexFromMember: (KeyCollectionsIndex): Removed to design document commented-out -setMember method. (Map): Removed to design document -setKeyType:, setKeySize: methods, unimplemented.

1998-06-17 Makefile.am *mgd*

Include from refbook/ instead of src/.

1998-06-15 Makefile.am *mgd*

(MODULE): New variable. Include Makefile.rules from src. Remove everything else.

1998-06-14 collections.h *mgd*

Remove MemberType; don't adopt it in Collection. Remove IndexSafety. Remove mention of MemberType in Drop. Remove EndsOnly; don't adopt it in List. Remove DupOption, BucketFunction, PartiallyOrdered, PartialOrderContext, and PartialOrderRelations; don't adopt in KeyedCollection. Remove mention of duplicate key, partial ordering, and index safety from KeyedCollection. Remove disabled -setIndexFromKey:, -getIndexFromKey, -getKeyAllocSize, -at:insert:setIndex:, -insertGroup, -removeKey:getKey:, -replaceKey:, -createIndex:setKey:, -createIndex:setMember:.

1998-06-12 collections00.sgml, collectionscont.sgml, collectionsmeta.sgml *mgd*

Update IDs to SWARM.module.SGML.type.

1998-06-06 collections.ent *mgd*

Use public identifiers.

1998-06-05 Makefile.am *mgd*

(swarm_ChangeLog): Add.

1998-06-05 collections.h *alex*

(LiteralString): Made an extern id <String>, rather than @class variable. Added doc tag. (DupOption): Put space between global variable tag and @end directive - causing problems for make-h2x script.

1998-06-03 collections.h *mgd*

Updated documentation tags. (BucketFunction): -getBucketFunction now returns bucket_t. (CompareFunction): -getCompareFunction now returns compare_t.

1998-06-01 collections.h *alex*

(Collection): Added method -setIndexFromMemberLoc: to protocol. (Index): Added doc string (//G) to Symbol and Error global variables - made these inside the @end protocol declaration for Index. (KeyedCollection): Added method -createIndex:fromMember: to protocol.

1998-06-01 collections.h *mgd*

Make LiteralString a @class.

1998-05-23 Makefile.am *mgd*

New file.

1998-05-23 collections.ent.in *mgd*

New file:

1998-05-23 collections.ent *mgd*

Removed.

1998-05-22 *mgd*

Begin revision log.

1998-05-06 collections.h *mgd*

(IndexSafety, Offsets, ForEach, DefaultMember, MemberBlock, Array, EndsOnly, DupOption, Sorted, CompareFunction, BucketFunction, PartiallyOrdered, PartialOrderContext, MapIndex, InputStream): Add //S.

1998-05-04 collections.h *mgd*

Remove NextPrev from Index protocol. Tweak comments for the sake of documentation processing.

1998-04-30 collections.h *mgd*

Augment the Index documentation. Move the existing Index info to the Collections protocol, as it is a bit more general.

1998-04-28 collections.h *mgd*

Add documentation tags.

1998-04-28 collections.h *mgd*

New protocols: MemberType, IndexSafety, Offsets, ForEach. (Collection): Include them. New protocols: DefaultMember, MemberBlock. (Array): Include them. New protocol: EndsOnly. (List): Include it. New protocols: DupOption, Sorted, CompareFunction, BucketFunction, PartiallyOrdered, PartialOrderContext, PartialOrderRelations. (KeyedCollection): Include them.

1998-04-11 collections.h *mgd*

Make archiver symbols extern, not common.

1997-12-04 collections.h *mgd*

(OutputStream, String): Constify string arguments.

1997-11-29 collections.h *mgd*

Add @deftype for InputStream, and declare symbols that getExpr can return.

1997-11-29 collections.h *mgd*

Declare [gs]etLiteralFlag methods.

1997-11-29 collections.h *mgd*

Append <Collection> to KeyedCollection deftype.

ArchiverArray

Name

`ArchiverArray` — Array encapsulation for serialization.

Description

Array encapsulation for serialization.

Protocols adopted by ArchiverArray

Create *(see page 46)*

Drop *(see page 54)*

RETURNABLE *(see page 66)*

Methods

Phase: Creating

- - **setArray:** *array*

Phase: Using

- - (void)**drop**
- - (void)**lispOutDeep:** (id <OutputStream>)*stream*
- - (void)**lispOutShallow:** (id <OutputStream>)*stream*
- - **convertToType:** (char)*destType* **dest:** (void *)*ptr*
- - (fcall_type_t)**getArrayType**
- - (unsigned)**getElementCount**
- - (size_t)**getElementSize**
- - (unsigned *)**getDims**
- - (unsigned)**getRank**
- - (void *)**getData**

ArchiverKeyword

Name

`ArchiverKeyword` — Keyword encapsulation for serialization.

Description

Keyword encapsulation for serialization.

Protocols adopted by ArchiverKeyword

Create *(see page 46)*

Drop *(see page 54)*

RETURNABLE *(see page 66)*

Methods

Phase: Creating

- `- setKeywordName: (const char *)name`

Phase: Using

- `- (void)lispOutDeep: (id <OutputStream>)stream`
- `- (void)lispOutShallow: (id <OutputStream>)stream`
- `- (const char *)getKeywordName`

ArchiverList

Name

`ArchiverList` — Archiver list encapsulation for serialization.

Description

Archiver list encapsulation for serialization.

Protocols adopted by ArchiverList

List *(see page 116)*

RETURNABLE *(see page 66)*

Methods

Phase: Using

- `- (void)`**`lispOutDeep:`** `(id <OutputStream>)`*`stream`*
- `- (void)`**`lispOutShallow:`** `(id <OutputStream>)`*`stream`*

ArchiverPair

Name

`ArchiverPair` — List pair encapsulation for serialization.

Description

List pair encapsulation for serialization.

Protocols adopted by ArchiverPair

Create *(see page 46)*

Drop *(see page 54)*

RETURNABLE *(see page 66)*

Methods

Phase: Creating

- `- **setConsFormatFlag:** (BOOL)`*`theConsFormatFlag`*
- `- **setCdr:** `*`cdr`*
- `- **setCar:** `*`car`*

Phase: Using

- `- (void)**lispOutDeep:** (id <OutputStream>)`*`stream`*
- `- (void)**lispOutShallow:** (id <OutputStream>)`*`stream`*
- `- (BOOL)**getConsFormatFlag**`
- `- **getCdr**`
- `- **getCar**`

ArchiverQuoted

Name

`ArchiverQuoted` — Archiver serialization object for (quote x) or 'x

Description

Archiver serialization object for (quote x) or 'x

Protocols adopted by ArchiverQuoted

RETURNABLE *(see page 66)*

Methods

Phase: Creating

- `-` **`setQuotedObject:`** *`value`*

Phase: Using

- `-` `(void)`**`lispOutDeep:`** `(id <OutputStream>)`*`stream`*
- `-` **`getQuotedObject`**

ArchiverValue

Name

`ArchiverValue` — Value encapsulation for serialization.

Description

Value encapsulation for serialization.

Protocols adopted by ArchiverValue

Create *(see page 46)*

Drop *(see page 54)*

RETURNABLE *(see page 66)*

Methods

Phase: Creating

- - **setNil**
- - **setClass:** (Class)*class*
- - **setBoolean:** (BOOL)*val*
- - **setChar:** (char)*val*
- - **setLongLong:** (long long)*val*
- - **setFloat:** (float)*val*
- - **setLongDouble:** (long double)*val*
- - **setDouble:** (double)*val*

Phase: Using

- - (void)**drop**
- - (void)**lispOutDeep:** (id <OutputStream>)*stream*
- - (void)**lispOutShallow:** (id <OutputStream>)*stream*
- - (Class)**getClass**
- - **getObject**
- - (BOOL)**getBoolean**
- - (char)**getChar**
- - (unsigned)**getUnsigned**
- - (int)**getInteger**
- - (long long)**getLongLong**

- - (float)**getFloat**
- - (long double)**getLongDouble**
- - (double)**getDouble**
- - (fcall_type_t)**getValueType**

Array

Name

`Array` — Collection supporting access only by relative position.

Description

An array is a collection of members that are all created as members of the collection at the same time. Existing member values may be replaced with new values, but the members themselves are fixed at constant offsets within the collection. The fixed structure of an array permits very fast access to members by integer offset positions, since the location of each member may be directly calculated.

The Array type is one of the simplest collection types in the collections library, and the closest to a data structure directly supported in C. Unlike C arrays, the group of members belonging to the array is not necessarily fixed for the lifetime of the array, but may be dynamically resized to contain a different number of members. When an array is dynamically resized, existing member values are preserved as much as possible.

The Array type adds few messages to the generic messages inherited from Collection. This type is provided partly so that a fixed-structure array can be accessed with the same uniform set of basic messages as any other kind of object collection. It also handles all required memory allocation within the collection. As an option, however, the Array type can be used to wrap an existing C array for external access as an object collection. It can also provide access to an internal C array for direct manipulation using C expressions. These forms of low-level access support hybrid modes of use in which advantages of both low-level manipulation and external object access can be combined.

The Array type is directly creatable, and supports all standard messages of Collection except removal of individual members. The messages based on an integer offset, either on the collection (atOffset:, atOffset:put:), or an index (setOffset:) all execute in fast constant time. Members of an array are fully ordered according to these integer offsets. Sequential access to members through its members is also fully supported. The Array type disables the remove message inherited from Collection; the message is defined, but any attempt to remove a member will raise an error that the operation is not supported.

The default value of the ReplaceOnly option is true, and cannot be overridden.

The type of index returned by begin: on an array is simply Index. There is no special index type for Array because there are no additional messages beyond those already defined by Index.

All the Array create-time options can also be set after the array is already created, subject to restrictions noted below.

Protocols adopted by Array

Collection *(see page 103)*

DefaultMember *(see page 105)*

MemberBlock *(see page 122)*

Serialization *(see page 67)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- + **create:** (id <Zone>)*aZone* **setCount:** (unsigned)*count*

Phase: Setting

- - **setCount:** (unsigned)*count*

  The Count option sets the number of members which belong to the collection. Any non-negative value including zero is valid. If the array already exists, the any existing members up to the new count will preserve their existing values. If the new count is greater than the existing count, or a new array is being created, all members will be assigned an initial default value of either nil, or a value previously specified for DefaultMember.

Collection

Name

`Collection` — A generic collection interface.

Description

A collection is a grouping of object references or other data values which are assigned explicitly as members of the collection. Depending on the subtype, collection members may also be maintained in various associations with each other, such as an ordering of members or an association of members with identifying key values. Major Collection subtypes include Array, List, Set and Map. The Collection supertype establishes common conventions (and associated messages) supported by all types of collections.

All collections support traversal over their members using a separate object called an Index. All collections also follow common rules regarding the types of data values which may be added as members. The next two subsections summarize these basic features of all collections.

An index is a special type of object that references a current position in an enumeration sequence over a collection. An enumeration sequence contains every member of a collection exactly once. Every collection defines an associated type of object for its index. The index type of a collection defines additional ways in which the members of a collection may be processed beyond any messages available directly on the collection. Often the operations of an index provide the most basic and flexible means for manipulating individual members of a collection.

An index object into a collection may be created at any time. An index is the basic means to traverse through all members of a collection. Multiple indexes on the same collection may all exist at the same time and may reference the same or different positions. Depending on the collection type, it may be possible to modify a collection through its indexes.

Once an index is created, the sequence of members in its enumeration sequence is guaranteed to remain the same, provided that no new members are added to the underlying collection, or existing members removed. If a member is located once at a particular position, it is guaranteed to remain at that position as long as the index itself remains.

Many collection types define an explicit ordering over their members. For such collections, the sequence of members referred to by an index will always be consistent with this ordering. An explicit, total ordering also guarantees that all indexes of the same collection have the same member sequence.

If no such ordering is defined, however, some particular sequence of all the collection still becomes associated with each created index. All collection members are guaranteed to be contained somewhere in the enumeration sequence for any particular index, but two indexes on the same collection are not guaranteed to have the same sequence.

The Index type corresponds to the iterator types defined as part of many other object-oriented libraries. The name Index is shorter and emphasizes the more abstract and multi-function role of these basic support objects for any collection. For more background on design of indexes and iterators, see the Interface Design Notes for the collections library.

Protocols adopted by Collection

Create *(see page 46)*

SetInitialValue *(see page 72)*

Copy *(see page 45)*

Drop *(see page 54)*

Offsets *(see page 123)*

ForEach *(see page 106)*

Methods

Phase: Creating

- - (void)**setIndexFromMemberLoc:** (int)*byteOffset*
- - (void)**setReplaceOnly:** (BOOL)*replaceOnly*

  This boolean-valued option restricts valid usage of a collection by excluding all operations which add or remove members. For some collection subtypes, a replace-only restriction can obtain many of the same performance advantages as a read-only collection, but without disabling replace operations as well. Just like the ReadOnly option, the ReplaceOnly option may be reset after a collection is created, provided it was not originally set to true.

Phase: Using

- - (id <PermutedIndex>)**beginPermuted:** (id <Zone>)*aZone*
- - (id <Index>)**begin:** (id <Zone>)*aZone*

  The begin: message is the standard method for creating a new index for traversing the elements of a collection. All further information about indexes is documented under the Index type.

- - (BOOL)**allSameClass**

  Returns YES if all members are of the same class.

- - (void)**deleteAll**

  Like removeAll:, but drops the member(s) as well.

- - (void)**removeAll**

  The removeAll message removes all existing members of a collection and sets its member count to zero. The collection then remains valid for further members to be added. This message has no effect on the objects which might be referenced by any removed member values. If resources consumed by these objects also need to be released, such release operations (such as drop messages) can be performed prior to removing the member values.

- - **remove:** *aMember*

The remove: message removes the first member in the collection with a value matching the value passed as its argument. If there is no such member, a nil value is returned. As with the contains: message, the speed of this operation may vary from very low to linear in the number of members, depending on the collection subtype.

- - (BOOL)**contains:** *aMember*

  The contains: message returns true if the collection contains any member value which matches the value passed as its argument. Depending on the collection subtype, this may require traversing sequentially through all members of the collection until a matching member is found. For other subtypes, some form of direct indexing from the member value may be supported. The message is supported regardless of its speed.

- - (unsigned)**getCount**

  getCount returns the integer number of members currently contained in the collection. All collections maintain their count internally so that no traversal of collection members is required simply to return this value.

- - (BOOL)**getReplaceOnly**

- - **copy:** (id <Zone>)*aZone*

  Note: copies are shallow; members inside the collection are not copied.

CompareFunction

Name

`CompareFunction` — Interface for defining the compare function to use when comparing to members in a collection.

Description

The function given will be called whenever one key value needs to be compared with another. Multiple calls to the function typically occur whenever members are added or removed from the collection, until the correct member for insertion or removal is determined.

The compare function is called repeatedly by the collection to compare two key values. The function should return zero if the key values are equal, -1 for the first key argument less than the second, and +1 for the first greater than the second. If a keyed collection is not sorted, either -1 or +1 may be returned for unequal keys, regardless of whether one might be taken as greater or less than the other.

Protocols adopted by CompareFunction

None

Methods

Phase: Creating

- - **setCompareUnsignedIntegers**
- - **setCompareIntegers**
- - **setCompareIDs**
- - **setCompareCStrings**
- - **setCompareFunction:** (compare_t)*aFunction*

Phase: Using

- - (compare_t)**getCompareFunction**

Typedefs

- compare_t int (*) (id, id)

DefaultMember

Name

`DefaultMember` — Methods for setting and getting the default member in a collection.

Description

When this option is set, the initial value of all new members will be set to the member value given (otherwise the default is nil). This option gives a convenient way to distinguish members which have never been set from any other valid member value, which could include nil. This option may be reset after array creation only if some setting for the option was given at create time. (The initial, explicitly set value can still be the default nil, but a value must be set explicitly for the option to be resettable later). The get message for this option always retrieves the current setting, but this value has no effect except when the count of an array is increased, so that new members need to be initialized.

Protocols adopted by DefaultMember

None

Methods

Phase: Setting

- `- (void)` **`setDefaultMember:`** *`memberValue`*

Phase: Using

- `-` **`getDefaultMember`**

ForEach

Name

`ForEach` — Messages for performing the same message on objects in a collection.

Description

The forEach messages supply a convenient shorthand for repeatedly performing the same message on all objects contained as members in a collection. The message to be sent is identified by the argument aSelector. This selector must define the same number of arguments as contained in any remaining argument slots of the forEach message. The argument types of the message to be sent must be either the id type, or some other type that will fit in the same number of bits as the id type. By global portability assumptions, the argument type could be as large as an int (signed or unsigned), but not necessarily as large as a long. To use the message, any non-id value must be cast to the id type as part of the call expression.

The forEach: messages are implemented by a simple loop through all members of a collection, using an internal, temporary index. If any operation more complex than a simple message send is required, this operation should just be coded directly using a loop that traverses its own index.

Protocols adopted by ForEach

None

Methods

Phase: Using

- `- (void)`**`describeForEachID:`** *`outputCharStream`*
- `- (void)`**`describeForEach:`** *`outputCharStream`*
- `- (void)`**`forEach:`** `(SEL)`*`aSelector`*
- `- (void)`**`forEach:`** `(SEL)`*`aSelector`* **`:`** *`arg1`*
- `- (void)`**`forEach:`** `(SEL)`*`aSelector`* **`:`** *`arg1`* **`:`** *`arg2`*
- `- (void)`**`forEach:`** `(SEL)`*`aSelector`* **`:`** *`arg1`* **`:`** *`arg2`* **`:`** *`arg3`*

ForEachKey

Name

`ForEachKey` — Exactly the same as the ForEach protocol, but only for KeyedCollections.

Description

Works identically to the ForEach protocol, but loops through the keys in a KeyedCollection, rather than the members.

Protocols adopted by ForEachKey

None

Methods

Phase: Using

- `- (void)` **`forEachKey:`** `(SEL)` *`aSelector`*
- `- (void)` **`forEachKey:`** `(SEL)` *`aSelector`* **`:`** *`arg1`*
- `- (void)` **`forEachKey:`** `(SEL)` *`aSelector`* **`:`** *`arg1`* **`:`** *`arg2`*
- `- (void)` **`forEachKey:`** `(SEL)` *`aSelector`* **`:`** *`arg1`* **`:`** *`arg2`* **`:`** *`arg3`*

Index

Name

`Index` — Reference into the enumeration sequence for a collection.

Description

An index is a reference into an enumeration sequence of a collection. Such an enumeration sequence contains all members of the collection in some order. This order will always be consistent with ordering of members in the collection, assuming there is such an ordering. Otherwise, the sequence will still contain all members in some order that remains fixed provided that new members are not added or removed from the collection.

An index is created by a begin: or createIndex: message against a collection. Each major collection type has its own corresponding index type, which supports specialized types of processing against the valid contents of that kind of collection. Once created, an index is a separate object from the collection itself, but it remains valid only so long as the collection itself still exists. Multiple indexes may exist at the same time against the same collection, and each index maintains its own position within an enumeration sequence for the collection.

Many indexes provde the ability modify the collection they refer to, in addition to simply traversing members. An index often provides the complete means for maintaining the contents of a collection, more than could otherwise be performed on the collection itself. The position or other status of the index is automatically updated to reflect any changes made through the index itself.

If changes to a collection are made while other indexes exist, those other indexes could be affected in potentially catastrophic ways.

Each index is a stand-alone object allocated within a zone passed as an argument in the message that created it. This zone need not match the zone of a collection. It is common for index lifetimes to be shorter than their collection. For example, indexes can be created in a temporary scratch zone for use only within a local loop.

Because messages to a collection are the only valid way to create an index, create messages and create-time options are not used with index types. All valid processing on an index is determined by characteristics of the collection from which it is created. Index types are typically named after the type of collection they are created from, and serve principally to define the specific messages valid for an index on that type of collection.

Index objects support the universal messages of the top-level DefinedObject supertype, along with the standard drop: and getZone messages. Even though they cannot be created except from a collection, new index objects can be created from an existing index using the standard copy: message. Each copy refers to the same collection as the initial index, and starts at the same position in its enumeration sequence. In all other respects, however, the new copy is an independent index that maintains its own position under any further processing.

Protocols adopted by Index

DefinedObject *(see page 52)*

Drop *(see page 54)*

RETURNABLE *(see page 66)*

Methods

Phase: Using

- - (int)**compare:** *anIndex*

  The compare: message compares the current location of one index with the current location of another index passed as its argument. If the two indexes have the same location, compare: returns zero. Otherwise, compare: returns +1 or -1 according to whether the argument index precedes or follows the receiver index in the enumeration sequence of the collection. If either of the two indexes has an unknown offset, and the location of the other index is anything other than Start or End or an immediately adjacent member, compare: returns the UnknownOffset integer value.

- - **setOffset:** (unsigned)*offset*

  Using the setOffset: message, an index may be positioned directly to a member using the offset of the member within its enumeration sequence. The speed of this operation depends on the specific type of the collection, just as noted for the atOffset: message on Collection. In the worst case, this operation is linear in the magnitude of the offset.

- - (int)**getOffset**

  Provided there is no major computational cost, an index also maintains the integer offset of its current member within the enumeration sequence of the collection. These integer offset values have the same definition as in the atOffset: messages of Collection. The getOffset message returns this current offset value. If the index is current positioned at the Start or End location, getOffset returns -1. If the index is positioned at a Between location, getOffset returns the offset of the immediately preceding member. If the offset is not currently available from the index, getOffset returns the special value UnknownOffset. This value is defined by a macro as the maximally negative value of a 32-bit, 2's-complement integer.

  An offset is always available from an index if its current position has been reached by repeated operations from an initial Start or End position, and there has been no other modification to the underlying collection. Some forms of direct member access operations supported by some index types, however, may result in an integer offset not being available. These restrictions are noted with the individual index type.

- - (void)**setLoc:** (id <Symbol>)*locSymbol*

  The setLoc: message may be used to reset the current location of an index to either Start or End. It may be used to reprocess a collection using an existing index after some other location has already been reached. It may also be used to position an index at the end of all members prior to traversing members in reverse order using prev.

Besides Start and End, setLoc: accepts the special argument values of BetweenAfter and BetweenBefore, which are also defined symbols. These argument values are only valid if the index is positioned at a current member. They reposition the index to the special location between the current member and its immediately following or preceding member.

- - (id <Symbol>)**getLoc**

  The getLoc message returns a symbol constant which indicates the type of location at which an index is currently positioned. This index location symbol has one of the following values: Start, End,

  and Member.

  The Start symbol indicates the special position preceding all members in the enumeration sequence for the collection. This is the location at which an index is positioned when it is first created. The End symbol indicates the special position following all members in the collection. This is the location at which an index is positioned just after a next message has returned nil, as a result of moving beyond the last member. The Member symbol indicates that the index is positioned at some current member in the enumeration sequence of a collection.

  The getLoc message is needed to traverse a collection which could contain nil values for its members. Without getLoc, there would be no way to distinguish a nil value returned by next as either a valid member value or the special value returned at the end of members. With getLoc, a loop that traverses a collection can test specifically for the end (or start) of members.

Example -getLoc #1

Following is a simple loop which illustrates such usage:

```
{
  id <Index> index = [aCollection begin: aZone];
  id member;

  for (member = [index next]; [index getLoc] == Member; member = [index
next])
    {
     // do something with member ...
    }
  [index drop];
```

```
}
```

- - **remove**

  The remove message removes the member at the current location of an index, and returns the member value removed. The index position is set to a special position between the members which previously preceded and followed the removed member. If there is no preceding or following member, the index is set to the special location before the start or after the end of all members. After a current member is removed, there is no member at the current index location, but a subsequent next or prev message will continue with the same member that would have been accessed had the current member not been removed. An InvalidIndexLoc error is raised if the index is not positioned at a current member.

- - **put:** *anObject*

  The put: message replaces the member value at the current index position with its argument. An InvalidIndexLoc error is raised if the index is not positioned at a current member.

- - **get**

  get returns the member value at which the index is currently positioned, or nil if the index is not positioned at a member.

  The get message provides an alternate way to obtain the current member value in a loop that traverses a collection; its return value is the same as next or prev would return when first positioning to a new member.

- - **findPrev:** *anObject*

  findPrev: repeatedly performs prev until the member value of the index matches the argument. nil is returned if the index reaches the end of valid members without matching the argument.

- - **findNext:** *anObject*

  findNext: repeatedly performs next until the member value of the index matches the argument. nil is returned if the index reaches the end of valid members without matching the argument.

- - **prev**

  The prev message works similarly, but positions to a valid member preceding the current position, or to a special position preceding all valid members.

- - **next**

  The next message positions the index to the next valid member after its current position, or to a special position after the end of all valid members. In addition to repositioning the index, both messages return the new member value to which they are positioned, or nil if there is no such member.

- - **getCollection**

  getCollection returns the collection referred to by an index. This collection never changes during the lifetime of the index.

Macros

- **INDEXENDP**(*obj*)

  Predicate to test if index is at the end.

- **INDEXSTARTP**(*obj*)

  Predicate to test if index is at the start.

- **REMOVEDP**(*obj*)

  Predicate to test if item at index has been removed.

- UndefinedOffset

Globals

id <Symbol> Start

values for index location

id <Symbol> End

values for index location

id <Symbol> Between

values for index location

id <Symbol> Removed

values for index location

id <Symbol> Member

values for index location

id <Error> OffsetOutOfRange

error types for collections

id <Error> NoMembers

error types for collections

id <Error> AlreadyAtEnd

error types for collections

id <Error> AlreadyAtStart

error types for collections

id <Error> InvalidIndexLoc

error types for collections

id <Error> InvalidLocSymbol

error types for collections

InputStream

Name

`InputStream` — Stream of input data.

Description

This type reads Lisp-like expressions into lists. Supports Lisp comments: semi-colons `;'

Protocols adopted by InputStream

Create *(see page 46)*

Drop *(see page 54)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- `-` **`setFileStream:`** `(FILE *)`*`fileStream`*
- `-` **`setExpr:`** *`expr`*
- `+` **`create:`** `(id <Zone>)`*`aZone`* **`setExpr:`** *`expr`*
- `+` **`create:`** `(id <Zone>)`*`aZone`* **`setFileStream:`** `(FILE *)`*`file`*

Phase: Using

- `-` **`getExpr`**
- `-` `(FILE *)`**`getFileStream`**

KeyedCollection

Name

`KeyedCollection` — Member identity definition shared by Set and Map types.

Description

A keyed collection is a collection in which each member can be compared with some other value that identifies the member. This value is referred to as the member key. The key value may be determined either by the member value itself, which defines a Set, or by external association with the member when the member is first added, which defines a Map.

The KeyedCollection type inherits all standard behavior of Collection. The KeyedCollection type is not itself creatable; it only serves as a common supertype for Set and Map collection types.

The keyed collection type establishes the common behavior shared by both Set and Map. Standard options are provided to declare ordering of members in the collection.

Protocols adopted by KeyedCollection

Collection *(see page 103)*

ForEachKey *(see page 107)*

Methods

Phase: Using

- `- (BOOL)`**`containsKey:`** *`aKey`*

  The containsKey: message returns true if the key value passed as its argument is contained in the collection, and false otherwise.

- `-` **`removeKey:`** *`aKey`*

  The removeKey: message removes a member matching a key value from the collection, and returns the member just removed. It returns nil if there is no key value in the collection which matches. If more than one entry was present for the key value, it removes and returns the first member in the internal collection created for duplicate members.

- `-` **`at:`** *`aKey`*

  The at: message returns the existing member of the collection which matches the key value passed as its argument, or nil if there is no key value in the collection which matches. If duplicate entries for this key exist, the entire collection of duplicate members created for the key value is returned instead.

- `-` **`createIndex:`** `(id <Zone>)`*`aZone`* **`fromMember:`** *`anObject`*

KeyedCollectionIndex

Name

`KeyedCollectionIndex` — Index behavior shared by Set and Map types.

Description

An index to a keyed collection traverses all members of the collection, regardless of whether these members belong to collections of members entered under duplicate key values. Internally, however, an index keeps track of any specific subcollection it is currently processing.

Protocols adopted by KeyedCollectionIndex

Index *(see page 108)*

Methods

None

List

Name

`List` — Collection of members in an externally assigned linear sequence.

Description

A list is a collection of members that are all maintained at some externally assigned position in a linear sequence of all members. The sequence is established by the position at which members are added: members can be added at the start of list, at the end, or at any point in the middle.

A list is also one of the most dynamic of basic collections, in that the cost of adding and removing members is very low. Members can be removed from any position just as easily as they can be added. A list automatically grows and shrinks to reflect the number of members at any one time, and there is no fixed capacity which limits the size to which a list may grow.

The List type is supports all messages of Collection. If created with default options, it provides no special speedup of accesses by integer offset.

Protocols adopted by List

Collection *(see page 103)*

Serialization *(see page 67)*

CREATABLE *(see page 44)*

Methods

Phase: Using

- `- (id <ListIndex>)`**`listBegin:`** `(id <Zone>)`*`aZone`*

  `Returns a ListIndex, the special index for the List type`

- `-` **`removeLast`**

  `Removes the last member from the list and returns it.`

- `-` **`removeFirst`**

  `Removes the first member from the list and returns it.`

- `- (void)`**`addLast:`** *`anObject`*

  `The addLast: message adds a new member to the end of the list.`

- `- (void)`**`addFirst:`** *`anObject`*

  `The addFirst: message adds a new member to the front of the list.`

ListIndex

Name

`ListIndex` — Index with insertion capability at any point in list.

Description

The addAfter: and addBefore: messages add members at a particular point in the sequence of members maintained by a list. The current location of an index determines the point at which a new member will be added. The addAfter: message adds a member at the list position immediately following the current index location. addBefore: adds a member to the immediately preceding location. Neither message changes the current location of the index, except that an index can change from a Start or End location to a location of Between.

Since an index may be positioned to any location in a list, these messages enable the construction of any desired sequence of members. Since the current index location remains unchanged, multiple members may all be inserted successively at some point in a list; previously added members are just pushed out one-by-one as new members are added.

An index with a location of Start, End, or Between is just as valid a location for addAfter: or addBefore: as an index positioned at a member. In these cases, there is no member at the current location of the index, so the new member is just inserted directly at the current index location, and the index is left positioned between the new member and the member that was previously adjacent in the opposite direction. If the previous location was Start and the message addAfter:, or the location was End and the message addBefore:, the index location remains Start or End.

Protocols adopted by ListIndex

Index *(see page 108)*

RETURNABLE *(see page 66)*

Methods

Phase: Using

- `- (void)`**`addBefore:`** *`anObject`*

  `Add a member before the index.`

- `- (void)`**`addAfter:`** *`anObject`*

  `Add a member after the index.`

ListShuffler

Name

ListShuffler — A class to randomize the order of a given Swarm List

Description

ListShuffler randomizes the order of the elements in a List; either the whole list or the num lowest elements. The list must be supplied. An uniform distribution can be supplied, or the system- supplied uniformUnsRand is used. The algorithm is from Knuth. All these methods modify the underlying collection, so any indexes should always be regenerated.

Protocols adopted by ListShuffler

Create *(see page 46)*

Drop *(see page 54)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- \+ **create:** (id <Zone>)*aZone* **setUniformRandom:** *dist*

  The create:setUniformRandom method creates the Shuffler and connects the supplied distribution object.

- \- **setUniformRandom:** *dist*

  the setUniformRandom: method connects the supplied uniform distribution to the Shuffler (run after createBegin:).

Phase: Using

- \- **shufflePartialList:** *list* **Num:** (unsigned)*num*

  the shufflePartialList:Num method randomizes the order of the 'num' lowest elements of the list, or the whole list if (num > size of list).

- \- **shuffleWholeList:** *list*

  the shuffleWholeList method randomizes the whole list.

Map

Name

`Map` — Collection of associations from key objects to member objects.

Description

Map is a subtype of KeyedCollection in which the key value associated with each member is independent of the member itself. Whenever a new member is added to the collection, a key value to be associated with the member must be supplied also. A Map defines a mapping from key values to member values.

For the Map type, key values are independent of the member values with which they are associated. Map defines two additional options to document information about its key values. Map also defines its own messages to distinguish the key value from member value in any operation which involves both.

Protocols adopted by Map

KeyedCollection *(see page 115)*

CompareFunction *(see page 104)*

Serialization *(see page 67)*

CREATABLE *(see page 44)*

Methods

Phase: Using

- `- (id <MapIndex>)`**`mapBegin:`** `(id <Zone>)`*`aZone`*

  Returns a MapIndex, the special index for the Map type

- `-` **`at:`** *`aKey`* **`replace:`** *`anObject`*

  Replaces an existing member value associated with a key value by a new value given as its final argument. The message returns the member value which was formerly associated with the key value.

- `- (BOOL)`**`at:`** *`aKey`* **`insert:`** *`anObject`*

  at:insert: inserts an entry into a Map containing the key and member values given as its arguments. It returns true if the key was not previously contained in the collection. An attempt to insert a duplicate key is simply rejected and false is returned.

MapIndex

Name

`MapIndex` — The index behavior for a Map.

Description

The index behavior for a Map.

Protocols adopted by MapIndex

KeyedCollectionIndex *(see page 115)*

RETURNABLE *(see page 66)*

Methods

Phase: Using

- - **get:** (id *)*key*

  Return the current item and it's key.

- - **prev:** (id *)*key*

  Return the previous item and it's key.

- - **next:** (id *)*key*

  Return the next item and it's key.

- - (unsigned long)**getKeyValue**

  The getKeyValue message returns the integer value associated with the current location of the index. It is exactly like getKey, except the value is returned as an integer. (A common application of this method is to get a Schedule time value associated with an Action in Java.)

- - **getKey**

  The getKey message returns the key value associated with the current location of the index. It returns nil if the index is not currently positioned at a member.

- - **setKey:** *aKey*

  The setKey: messages repositions the index to an entry having a key value that matches its argument. If there is more than one entry matching this key value, the index is positioned to the first entry that matches.

MemberBlock

Name

`MemberBlock` — A way to wrap an existing C array for access as an object collection.

Description

This option provides a means to wrap an existing C array for access as an object collection. If this option is given, dynamic resizing is not supported. The current C array being wrapped, however, can be replaced by giving a new setting for MemberBlock after an array already exists. A new setting can be given only if some setting was initially given at create time.

Even if a setting was not given for MemberBlock at create time, the get message for MemberBlock always returns a pointer to whatever internal memory array is currently being used by the array object. If a MemberBlock setting was given, the pointer returned will be the same as the one previously given. In either case, the pointer returned may be used to manipulate member values in any way desired using native C expressions. After an array has been created, there is no way to determine whether the MemberBlock pointer was established externally or by internal allocation. If the pointer was established by internal allocation, however, the external program must make no attempt to free or otherwise reallocate this memory.

A count must always be supplied with an external member allocation, using the setCount: argument of the compound message. If an external allocation is being used, the only way to reset the count is also to reset MemberBlock; any attempt to use the setCount: message by itself will raise an error. Whenever an external member allocation is being used, the external program is entirely responsible for assuring that the MemberBlock value is a pointer to valid allocated memory containing at least the number of member slots given by setCount:.

With an external member allocation, the array itself will not attempt to either allocate this memory or free it when the array is dropped. Dropping the array only removes its reference to the external allocation.

Since an array neither allocates nor frees an external member allocation, the same region of allocated memory may be referenced by multiple arrays, including overlapping member ranges each defined by starting location and count. This flexibility enables alternate subrange views of a single, contiguous initial allocation by means of separately created external collections.

Protocols adopted by MemberBlock

None

Methods

Phase: Creating

- `+ create: (id <Zone>)aZone setMemberBlock: (id *)members setCount: (unsigned)count`

Phase: Setting

- `- (void)setMemberBlock: (id *)members setCount: (unsigned)count`

Phase: Using

- `- (void *)getMemberBlock`

MemberSlot

Name

`MemberSlot` — Allocation in member/key for fast setMember:/setKey:

Description

The MemberSlot option indicates that space has been reserved within each member that allows it to contain a link directly to its position in the enumeration for a collection. If such space has been reserved, special messages can be used that rapidly position an index directly to the member. Operations to remove members are also much faster.

The value of MemberSlot specifies the offset in bytes from the start of each member where the space for its position link has been reserved.

The offset of the position link from the start of a member may be either positive or negative, in the range of -2048 to +2047. The default value of MemberSlot is UnknownOffset (a large negative value), which specifies that no slot for an internal position link is available within each member.

Protocols adopted by MemberSlot

None

Methods

None

Typedefs

- dupmember_t struct { void *memberData[2]; id owner; }
- member_t struct memberData { void *memberData[2]; }

Offsets

Name

`Offsets` — Methods for accessing collection members by position.

Description

An offset is an integer value that gives relative position of a member in the enumeration sequence of a collection. Offsets start the count of the first member at zero, just like C array indexing.

Offsets provide an alternate means to access the members of a collection, without creating a separate index object. Some collection subtypes (such as Array) support fast, direct access by integer member offset, while others support member offsets only as a shorthand for sequential access through every preceding member. Access by offsets is supported on all collections regardless of whether its speed on a particular collection type.

atOffset: and atOffset:put: raise the error OffsetOutOfRange if the offset is greater than or equal to the count of members in the collection.

Protocols adopted by Offsets

None

Methods

Phase: Using

- - **`getLast`**

  Equivalent to [aCollection atOffset: [aCollection getCount] - 1].

- - **`getFirst`**

  Equivalent to [aCollection atOffset: 0].

- - **`atOffset:`** (unsigned)*`offset`*

  Returns the member at a particular member offset.

- - **`atOffset:`** (unsigned)*`offset`* **`put:`** *`anObject`*

  The atOffset: put: message replaces the member at a particular offset with a new value, and returns the previous member value at this offset.

OrderedSet

Name

`OrderedSet` — A set of members in an externally assigned linear sequence.

Description

An OrderedSet is a totally ordered collection of members in which every member also has a distinct identity as defined by comparison against a key value.

(.. This type is currently implemented only using the low-level option of an internal member slot, and the messages for that option do not match the documentation in KeyedCollection. If you need one of these objects, then either use a List or wait for some other implementation.)

The sequence of members of an OrderedSet is established using the same messages that maintain member sequence in a List. An OrderedSet supports customization and access by key as defined by Set and KeyedCollection. The union of messages from all these sources defines the total interface of an OrderedSet. Members with duplicate keys, however, are not valid for an OrderedSet. Each member must have a unique position within the member sequence

Protocols adopted by OrderedSet

Set *(see page 130)*

List *(see page 116)*

CREATABLE *(see page 44)*

Methods

None

OutputStream

Name

`OutputStream` — Stream of output bytes.

Description

The OutputStream type currently supports the writing of types to a Lisp-like format. It is a placeholder for more general stream types. A stream is a collection that supports only sequential addition of members (an output stream) or sequential removal of members (an input stream). With the exception of the -catC: method, all messages write to stream in Lisp archiver format.

Protocols adopted by OutputStream

Create *(see page 46)*

Drop *(see page 54)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- - **setExprFlag:** (BOOL)*exprFlag*
- - **setFileStream:** (FILE *)*fileStream*
- + **create:** (id <Zone>)*aZone* **setFileStream:** (FILE *)*fileStream*

Phase: Using

- - (void)**catNil**
- - (void)**catUnsignedPair:** (unsigned)*a* **:** (unsigned)*b*
- - (void)**catEndMakeClass**
- - (void)**catStartMakeClass:** (const char *)*className*
- - (void)**catEndMakeInstance**
- - (void)**catStartMakeInstance:** (const char *)*typeName*
- - (void)**catEndParse**
- - (void)**catStartParse**
- - (void)**catEndQuotedList**
- - (void)**catStartQuotedList**
- - (void)**catEndList**
- - (void)**catStartList**

- - (void)**catEndCons**
- - (void)**catStartCons**
- - (void)**catEndFunction**
- - (void)**catStartFunction:** (const char *)*functionName*
- - (void)**catClass:** (Class)*class*
- - (void)**catType:** (const char *)*type*
- - (void)**catEndArray**
- - (void)**catArrayRank:** (unsigned)*rank*
- - (void)**catSeparator**
- - (void)**catString:** (const char *)*str*
- - (void)**catSymbol:** (const char *)*symbol*
- - (void)**catKeyword:** (const char *)*keyword*
- - (void)**catEndExpr**
- - (void)**catStartExpr**
- - (void)**catLiteral:** (const char *)*str*
- - (void)**catPointer:** (const void *)*ptr*
- - (void)**catUnsignedLongLong:** (unsigned long long)*ulnglng*
- - (void)**catLongLong:** (long long)*lnglng*
- - (void)**catUnsignedLong:** (unsigned long)*ulng*
- - (void)**catLong:** (long)*lng*
- - (void)**catUnsignedShort:** (unsigned short)*usht*
- - (void)**catShort:** (short)*sht*
- - (void)**catUnsigned:** (unsigned)*un*

  Writes an unsigned to stream in Lisp archiver format
- - (void)**catInt:** (int)*i*

  Writes an integer to stream in Lisp archiver format
- - (void)**catLongDouble:** (long double)*dbl*

  Writes a double to stream in Lisp archiver format
- - (void)**catDouble:** (double)*dbl*

  Writes a double to stream in Lisp archiver format
- - (void)**catFloat:** (float)*flt*

  Writes a float to stream in Lisp archiver format
- - (void)**catChar:** (char)*ch*

  Writes a character to stream in Lisp archiver format
- - (void)**catBoolean:** (BOOL)*bool*

Writes a boolean to stream in Lisp archiver format

- - (void)**catC:** (const char *)*cstring*

  Writes character string to stream

- - **getExpr**
- - (FILE *)**getFileStream**

Permutation

Name

Permutation — A class that represents a permutation of elements of a collection

Description

Permutation is used to generate a permutation of elements of a a collection and store them in an array for fast access. Permutation only mirrors the original collection. Updates of contents of Permutation will not reflect on the original collection.

Protocols adopted by Permutation

Collection *(see page 103)*

Create *(see page 46)*

Array *(see page 99)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- - **setUniformRandom:** *rnd*
- - **setLastPermutation:** (id <Permutation>)*permutation*
- - **setCollection:** (id <Collection>)*collection*

PermutationItem

Name

`PermutationItem` — An element of a Permutation

Description

An element of a Permutation

Protocols adopted by PermutationItem

Create *(see page 46)*

RETURNABLE *(see page 66)*

Methods

Phase: Creating

- - **`setPosition:`** `(unsigned)`*`position`*
- - **`setItem:`** *`item`*

Phase: Using

- - `(unsigned)`**`getPosition`**
- - **`getItem`**

PermutedIndex

Name

`PermutedIndex` — General PermutedIndex class.

Description

PermutedIndex class may be used for randomized traversals of a collection. Methods implemented offer the same functionality as Index class does, except that traversal is randomized.

Protocols adopted by PermutedIndex

Index *(see page 108)*

RETURNABLE *(see page 66)*

Methods

Phase: Creating

- - **`setUniformRandom:`** *`rnd`*
- - **`setCollection:`** *`aCollection`*

Phase: Using

- - **`reshuffle`**

Set

Name

`Set` — Collection of members each having a defined identity.

Description

Set is a subtype of KeyedCollection in which the key value associated with each member is determined by the member value itself. The key value may be identical to the member itself, or may be defined as a a function of the member using a create-time option.

The Set type inherits most of its interface from the KeyedCollection supertype. Set defines no create-time options beyond those already defined by KeyedCollection. If a custom compare or bucket function is specified, the member value is passed as the key value arguments of these functions. These functions determine what part of the member value is part of the key value, by determining which key values will compare equal to any member.

Protocols adopted by Set

Set *(see page 130)*

KeyedCollection *(see page 115)*

CREATABLE *(see page 44)*

Methods

Phase: Using

- `- ` **`replace:`** *`anObject`*

  The replace: message replaces a member stored at a given key with another member value that matches the same key. The new value to replace the existing one is passed as the argument. replace: returns the member value that was replaced, or nil if the collection contained no member with a matching key.

- `- (BOOL)` **`add:`** *`anObject`*

  The add: message adds a new member to a set.

String

Name

`String` — Character string object (later to support collection behavior).

Description

The String object type packages a null-terminated, C-format character string into an object. All memory allocation needed to hold the string value is handled by the object. This type currently defines only the most rudimentary operations for initializing and appending C-format character strings. These are sufficient for its current limited roles in places that need a uniformity between character strings and other kinds of allocated objects.

Protocols adopted by String

Create *(see page 46)*

Drop *(see page 54)*

Copy *(see page 45)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- `+ **create:** (id <Zone>)`*`aZone`* `**setC:** (const char *)`*`cstring`*

Phase: Setting

- `- (void)**setC:** (const char *)`*`cstring`*

Phase: Using

- `- (unsigned)**getCount**`
- `- (void)**catC:** (const char *)`*`cstring`*
- `- (const char *)**getC**`

General

Name

`collections` — Standard collection types

Description

The collections library follows the library interface conventions of the defobj library. It also depends on standard supertypes and classes defined by this library. Initialization of the collections library automatically initializes the defobj library as well. Since defobj also requires the collections library, both must always be linked into an application together.

Macros

- **ARCHIVERDOTP**(*obj*)
- **ARCHIVEREOLP**(*obj*)

Functions

- int **compareCStrings**(id, id)

  A routine for comparing C strings.

- int **compareIDs**(id, id)

  A routine for comparing objects. Only useful for equality (EQ) discrimination.

- int **compareIntegers**(id, id)

  A routine for comparing integers.

- int **compareUnsignedIntegers**(id, id)

  A routine for comparing unsigned integers.

Globals

id <Symbol> ArchiverLiteral

Tokens used by the archiving parser.

id <Symbol> ArchiverQuote

Tokens used by the archiving parser.

id <Symbol> ArchiverEOL

Tokens used by the archiving parser.

id <Symbol> ArchiverDot

Tokens used by the archiving parser.

Activity Library

Overview

The activity library is responsible for scheduling actions to occur within a simulated world, and for making these actions actually happen at the right time in the right order. It provides the foundation of dynamic, object-oriented simulation within Swarm.

Actions consist of messages to objects, calls to functions, or groups of actions in some defined order. The activity library guarantees that all these actions, and the state changes they produce, occur at predictable points in time. Time is defined by the relative order of actions, and may also be indexed by the discrete values of a world clock.

1. Dependencies

Following are the other header files imported by <activity.h>:

```
#import <collections.h>
```

The activity library follows the Swarm Defobj Library *(see page 26)* of the Defobj Library *(see page 26)*. The activity library relies heavily on the basic collection types defined in the Collections Library *(see page 83)*, and the collection interfaces are an integral part of the interfaces defined within this library. Initialization of the activity library is done automatically by the containing Swarm libraries, and automatically initializes the collections and defobj libraries as well.

2. Compatibility

No explicit compatibility issues for particular versions of Swarm

3. Usage Guide

3.1. Role of the activity library in Swarm

The activity library provides a foundation for dynamic, object-oriented simulation in Swarm. Swarm assumes that a user defines an object-oriented representation for the structure of a world to be simulated. Once such a representation is established, activity library components are responsible for generating all state changes and flow of information within it. These state changes must be carefully controlled to ensure that they occur only at appropriate times and places in the model. The activity library provides mechanisms to establish this control.

Once the simulation of a dynamic model begins, everything that happens to the model occurs as a direct result of messages sent to world objects by activity library components. These components are also represented by objects, since this is the most effective means for their representation as well, but their purpose is not to represent the current state of the simulated world, but rather to generate changes in world objects by sending messages in a valid order. The activity library has two basic categories of

components: those that represent messages to be sent, including constraints on permissible order for sending them, and those that execute the message sends these representations specify, making sure they conform with all constraints.

All changes to a model occur as a result of messages sent to objects within it. These messages invoke compiled methods of receiving objects, which may change local state or send messages to other objects to propagate effects as widely as needed. Because the activity library finally interacts with the world model just by invoking its defined methods, the model can prepackage as much behavior as it can in the form of fast compiled methods. The activity library just triggers the prepackaged behavior at proper times under its own explicit, dynamically interpreted representation.

As conditions shift during execution of a model, the model can also examine and alter the representation of its future behavior contained in the activity structures that drive it. Any changes to this future behavior, however, still occur only as a result of some currently executing action initiated by the activity library.

3.2. Activity library components

The activity library defines one set of basic object types to provide a very rich representation of the kinds of message sending patterns it could generate. Its other major category of object types controls and tracks a current state of processing within these representations.

Both these representations are more abstract than typical object representations, since they deal not with any constant state which can be statically analyzed, but with shifting patterns of messages to be fired to generate such state. While the basic structure of a Swarm simulation is that of a discrete event simulation, its activity representation is also more complicated than most such systems.

There are two basic reasons for the potential complexity of a Swarm activity representation. One is that Swarm supports a decentralized representation of activity to be generated, both to reflect the nature of much of the behavior that it simulates, and so that execution may be distributed across multiple parallel processors. To avoid any need for synchronizing its decoupled activities more often than necessary, Swarm enables a model designer to avoid overspecifying constraints on the patterns of message sends which might potentially be valid. The partial order representation it adopts for a distributed and decoupled plan of activity is inherently more complicated than the single centralized event list adopted by many discrete event simulation systems.

The other reason for potential added complexity is that Swarm's representation of intended activity may be broken into many separate, modular components, which can be bound together in various ways to create larger components. The Swarm composition structure, when fully implemented, will be fundamentally as powerful as the modular abstractions found in most programming languages, with the added complexity of controlling the time at which various events occur.

In spite of this high potential complexity, none of these features is needed for many simple models, such as those that contain only a small variety of basic behaviors, or which define all their behavior to occur at regular, repeating timesteps. The Swarm representation is extremely rich to enable it to scale to large, multi-level models with a variety of dependent behaviors built into the model, and also to facilitate running models on massively parallel machines. Many of the features which seem complex, moreover, are also especially well-suited to building modular and reusable library components. It is anticipated that many of the more advanced features will find their heaviest usage in pre-built libraries that hide internal complexity from applications that use them.

No matter how complex the structures built from them, all the activity library components finally result in the direct execution of messages sends to objects in a model. Because of this direct execution, very precise meaning can be defined for each of them, in terms of message sends that can or must occur. Every component of a structure to be executed, and every event of its execution, can also be accessed using the interfaces of an object-oriented representation. This means that tools can be built to understand and interact with any components to be executed, and also to trace and debug activity as it occurs. None of these tools have been built yet as part of Swarm, but many of the activity library interfaces are present to support them, not because normal use of the library requires the added components.

3.3. Action plan components

The first set of activity library components represent messages to be sent to objects in a model, together with constraints in the order in which they may be sent. All these components are defined as subtypes of the one generic type called ActionPlan.

The two basic kinds of action plans are a simple group of actions to be performed in some order, called an ActionGroup, and a series of actions to be performed at discrete points in time, called a Schedule. The basic element of an action plan is a simple object called an Action. An action defines a particular message to be sent to an object or objects.

In its representation of action plans, Swarm relies heavily on the dynamic message sending capabilities of the Objective C language. The support of Objective C for dynamic message sends is absolutely crucial to Swarm's implementation of generic activity structures. Objective C defines a special kind of data value called a selector, which identifies a message according to its name. One of these selector values is stored in each action of an action plan. During execution of the plan, Objective C machinery performs the actual message send using its selector.

Action plans are free-standing objects that may be created directly by a user program, using a variety of selectable create-time options. Once created, action plans may also refer to each other, by containing special kinds of actions to start another action plan or to perform all the actions within it.

Individual actions can be created only as completely controlled components of some action plan. They are created not by a standard create message, but by special messages (each containing the phrase createAction in its name) sent to an action plan that creates the action as part of the action plan. If the same action is needed in more than one plan, it has to be created in each plan where it is needed. If the entire action plan is dropped, all its actions are also automatically dropped. So the only action plan components which need to be directly managed by an application are the action plans themselves.

Action plans are relatively straightforward components, because they are entirely passive. The only actions they ever contain are those which an application has created in them. These actions stay in them indefinitely unless a special option is requested to clean them out as each one is executed. Because these plans are passive, read-only components to all execution machinery, if the same pattern of actions needs to be triggered at multiple points in a model, it's perfectly valid to create one copy of the actions in an action plan, and then refer to the plan anywhere the actions may be needed.

Even though action plans are passive, containing only what has been placed in them, there is no requirement that their contents remain fixed. Both of the action groups and schedules are implemented directly as collections of their actions. New actions may be added at any time, and existing ones may be dropped or moved from position to another. The contents of action plans may be as dynamic as they need to be to represent the future actions needed in a model. None of these shifting contents has any effect on the model until actually processed during model execution.

3.4. Model execution component

The main responsibility of model execution is just to perform the actions specified by an action plan, in an order of execution consistent with any requirements of the plan. Each action plan may specify as few or as many constraints as it likes over the possible order in which its actions could be performed. Given these specifications, the execution objects are entirely responsible for selecting each action to be performed and then performing them.

There are two simple cases of ordering that account for most all usage, including that of the current Swarm demo programs. One is to permit the actions to be performed in any order, including concurrent execution if hardware and software support were available to do this. The other is to require them to be performed in some specific sequence, one after another, so that the effects of one action could depend on actions which preceded it. This sequence could be either fixed and predetermined, or dynamically established each time the action plan is performed. If a dynamically determined sequence is needed, perhaps even selecting which actions are to be performed at all, users can provide custom subclasses for an action plan and an execution object that performs it. A builtin option is to generate an entirely random sequence each time a plan is executed.

Each time an action plan is being processed, a special kind of object called an activity is created entirely automatically by the runtime processing machinery. These activity objects implement the internal machinery of a virtual processor which has the ability to execute action plans. To get any activity started on a model, a processor is first initialized to run a single top-level plan. All other activity during the lifetime of a model must occur as a result of actions initiated by this plan (which may include the startup of other plans).

Because the activity objects are internal to the processing machinery, an application can usually just ignore their existence. They come and go dynamically as various action plans are started and completed, all arranged in a stack or tree of current activities controlled by a single top-level processor. The activity objects are potentially useful, however, to obtain information about the context in which an action is currently running, or to build debugging or tracing tools to understand actions being performed.

One of the kinds of context information available from an activity object is the current time of a clock value incremented as a schedule is processed. A schedule is a kind of action plan in which all actions occur at specific points in time explicitly established within the schedule. As a schedule is executed, the activity object keeps a current time clock, which holds a global time from the start of all model execution, regardless of the time when the schedule itself was started. An activity object that processes a schedule is called a timeline activity, because its time continually increases from a global base time regardless of times contained in the schedule.

New activities are created whenever an action plan being processed contains an action to perform another action plan, or to start another action plan for autonomous execution. If one action plan performs another, its own processing is stopped until a new activity processing the other action plan completes. If one action plan starts another, the new action plan is started as an autonomous activity controlled only by a higher-level, containing activity.

A Swarm is an activity that exists only to control and coordinate other started subactivities. Unless the subactivities have some special form of explicit synchronization, none of their internal actions has any required ordering relative to those of other plans except as explicitly established during activity execution. The swarm can serve as a simple container of started subactivities which only occasionally synchronize for messages they send to each other. The swarm can be used to hold collections of objects as well as its subactivities as needed to help them coordinate with one another.

One special form of synchronization within a swarm is built into the virtual processing machinery. This synchronization interleaves the actions that occur for every successive time value during processing of timeline subactivities. This merging of actions is often relied on to interleave display and analysis processing with the scheduled actions of a base model. Since no other mechanisms for subactivity coordination are implemented in the current version of Swarm, synchronizing subschedule activities is the major current role of the Swarm activity type. In later versions of Swarm, a swarm will also serve as an important means for organizing a large models into clusters of more densely interacting components, and will also provide a basis of decomposition for parallel execution.

4. Advanced Usage Guide

Unavailable

5. Subclassing Reference

Subclassing is supported by the activity library as an integral technique for extending the framework it implements. There are two specific places where subclassing is expected as the normal technique of extension: definition of concurrent action groups, and definition of customized swarm objects. Concurrent action groups are not yet fully documented because their full interface is still being finalized. Swarm subclasses should inherit from the Swarm superclass defined in the Objectbase Library *(see page 183)* library; the activity library provides the underlying support packaged by this superclass.

6. Interface Design Notes

Unavailable

7. Implementation Notes

Unavailable

Documentation and Implementation Status

The activity library is in process of being converted to draw even more of its support from underlying types and classes in the defobj and collections libraries. The basic interface has already been changed to reflect this consolidation, so the progrmming interface for currently available function is expected to remain stable.

The activity library follows the documentation structure of the Swarm Defobj Library *(see page 26)*. There are placeholders for each section of documentation so that all links should at least link up with something, whether or not there's anything there. The Interface Reference section is already fairly complete, and there is a start toward a more complete Usage Guide section. The Advanced Usage Guide and other sections will come at a later time.

An attempt has been made to remove all discussion of future capabilities and to document instead only the capability that is already there.

Source for code examples in the Usage Guide is included in the text where it occurs. Source consists of fragments taken from example programs. The example programs themselves may be downloaded from the web pages and compiled on your own system, using links provided for that purpose. (.. currently there are no code examples in the Usage Guide, which so far is only a general overview of the activity library ..)

There is also a directory of test programs (GridTurtle Test Programs *(see page 404)*, contained within the documentation release directory) that provides additional code examples of many basic features of the defobj, collections, and activity libraries. These code examples are very rough and subject to change, however, since their main use is with each new release of the libraries to test various new or existing features. They are sometimes the only source, however, for examples of some library features that have not yet been used anywhere else.

Revision History

2001-06-11 activity.h *mgd*

(ActionSelector): Move setMessageSelector: to setting phase.

2000-11-27 activity.h *mgd*

(ActionConcurrent): New protocol.

2000-07-24 activity.h *mgd*

(Action): Don't adopt RETURNABLE.

2000-07-20 activity.h *mgd*

([Schedule -setKeepEmptyFlag:]): Make create-time.

2000-06-23 activity.h *mgd*

(ConcurrentGroup): Add _setActionConcurrent_ and _getEmptyConcurrent_. (ConcurrentSchedule): Adopt ConcurrentGroup instead of ActionGroup.

2000-05-23 activity.h *mgd*

(ActionSelector): Split out of ActionTo. (ActionForEachHomogeneous): New protocol.

2000-05-18 activity.h *mgd*

([{ActivationOrder,Schedule} remove:]): Hide in a #ifndef IDL.

2000-04-02 *mgd*

Swarmdocs 2.1.1 frozen.

2000-04-02 activity00.sgml *mgd*

Delete sentence about top-level functions calls and methods being a "concern" to the activity library. Huh?

2000-02-29 *mgd*

Swarmdocs 2.1 frozen.

1999-12-01 activity.h *mgd*

(ActionChanged): New type.

1999-11-23 activity.h *mgd*

(ActivityIndex): New protocol. (Activity): Move before Action. Add _performAction_:. (ActionTo, ActionCall): Adopt Action. (ActionArgs): Don't adopt Action. (COMPLETEDP, HOLDINGP, INITIALIZEDP, RELEASEDP, RUNNINGP, STOPPEDP, TERMINATEDP): Remove (now) unnecessary casting. (ProcessType): Moved to design document.

1999-10-31 activity.h *mgd*

(ActionCreatingFAction): New protocol. (ActionCreating): Adopt it. (Schedule): Add at:createFAction: declaration.

1999-10-31 activity.h *mgd*

(FAction): Advertise. (Action): Adopt Create and Drop. (ActionArgs, ActionTo, ActionCall): Organize methods into creating and using sections.

1999-09-16 activity.h *mgd*

Add ActionTo, ActionCall, and ActionForEach protocol-conforming return types. (DefaultOrder): Move setDefaultOrder: to setting phase.

1999-09-07 activity.h *alex*

(Schedule): Re-enable conformance to Map.

1999-08-22 activity.h *mgd*

(Action, ActionCall, ActionTo, ActionForEach, Activity, ScheduleActivity, SwarmActivity, ForEachActivity, ConcurrentGroup, ConcurrentSchedule, ActivationOrder): Change from CREATABLE to RETURNABLE. (SwarmProcess): Remove CREATABLE. Return id <Activity> for getActivity. (Schedule): Add Zone typing for +create:*.

1999-07-28 activity.h *alex*

(Schedule): Add +create:setRepeatInterval: and +create:setAutoDrop: convenience methods to CREATING phase of protocol.

1999-07-12 activity.h *vjojic*

(getCurrentSchedule): Typo.

1999-07-11 activity.h *vjojic*

Make Action, ActionCall, ActionTo, ActionForEach, Activity, ScheduleActivity, SwarmActivity, ForEachActivity CREATABLE protocols.

1999-06-16 activity.h *mgd*

(RelativeTime): Make setRelativeTime: return self. (RepeatInterval): Likewise.

1999-06-03 activity.h *alex*

(ActionCreatingTo): Fix missing message receiver in the example code provided. Thanks to Ken Cline <kcline@c3i.saic.com> for the report.

1999-05-29 activity.h *mgd*

Include externvar.h. (ActionForEach, CompoundAction, ActionGroup): Add DefaultOrder compliance. (ActionType): Add (id <Activity>) return types to activate* methods. Rearrange protocols so that the Activity protocol will be declared for (id <Activity>).

1999-05-28 activity.h *mgd*

Use `externvar' for external variable declarations. (TimebaseMax): Remove; it's unused.

1999-02-05 activity.h *mgd*

(Schedule): Declare -setKeepEmptyFlag:.

1998-12-22 activity.h *mgd*

(ActionType): Remove activate:.

1998-12-20 activity.h *mgd*

(Activity): Remove stepEntry and stepExit methods (moved to design document). (ScheduleActivity): Remove setTerminateAtEnd:, getTerminateAtEnd, getSynchronizedMode, and getCurrentTimebase (moved to design document). Add creating phase tag. (ActionType): Remove -getActionType.

1998-12-17 activity.h *mgd*

(ActionGroup): Don't adopt OrderedSet. (Schedule): Don't adopt Map. (RelativeTime): Remove create:setRelativveTime:. (RepeatInterval): Remove create:setRepeatInterval:. (Action): Disable declaration of -getActionType. (ActionType): Disable declaration of activate:.

1998-12-11 activity.h *vjojic*

(DefaultOrder): DefaultOrder reinserted.

1998-10-29 activity.h *mgd*

(ActionCreatingTo): Fix example syntax for createActionTo:message:.

1998-09-08 activity.h *mgd*

(COMPLETEDP, HOLDINGP, INITIALIZEDP, RELEASEDP, RUNNINGP, STOPPEDP, TERMINATEDP): New macros.

1998-07-22 activity.h *mgd*

Replace @deftype with @protocol throughout.

1998-07-12 activity.h *mgd*

(Schedule): Declare -insertGroup:.

1998-06-17 Makefile.am *mgd*

Include from refbook/ instead of src/.

1998-06-15 Makefile.am *mgd*

(MODULE): New variable. Include Makefile.rules from src. Remove everything else.

1998-06-14 activity.h *mgd*

Remove DefaultOrder; don't adopt it in CompoundAction. Remove example that uses it in ConcurrentGroupType. Remove mention of partially ordered sets from ActionGroup, Schedule, and Action. Remove InternalTimeMultiplier; don't adopt it in SwarmProcess.

1998-06-12 activity00.sgml, activitycont.sgml, activitymeta.sgml *mgd*

Update IDs to SWARM.module.SGML.type.

1998-06-06 activity.ent *mgd*

Use public identifiers.

1998-06-05 Makefile.am *mgd*

(swarm_ChangeLog): Add.

1998-06-05 activity.h *mgd*

Add/update documentation tags. Declare addLast: and remove: (in using phase).

1998-06-01 activity.h *alex*

(ActionType): Added method -activate: anActionType.

1998-05-23 Makefile.am *mgd*

New file.

1998-05-23 activity.ent.in *mgd*

New file.

1998-05-23 activity.ent *mgd*

Removed.

1998-05-22 *mgd*

Begin revision log.

1998-05-22 activity.h *alex*

(getCurrentTime,getTopLevelActivity): Added (//#) documentation strings. (_activity_zone, _activity_trace): Made existing inline comments into (//G) doc strings:

1998-05-06 activity.h *mgd*

Remove instances of <p> Minor spacing changes to methods. (ActionType, ActionCreatingCall, ActionCreatingTo, ActionCreatingForEach, ActionCreating, Action, ActionArgs, ActionCall, ActionTo, Activity, ForEachActivity, ScheduleActivity, SwarmActivity): Add phase tags. (GetSubactivityAction): Add //S and //D tags. (SynchronizationType, InternalTimeMultiplier): Add //S.

1998-04-29 activity.h *mgd*

Include new protocols DefaultOrder, AutoDrop in CompoundAction. Include new protocols ActionCreatingTo, ActionCreatingForEach in ActionCreating. Include new protocols RelativeTime, RepeatInterval, ConcurrentGroupType, SingletonGroups in Schedule. Include SynchronizationType and InternalTimeMultiplier in SwarmProcess. Disable setTerminateAtEnd:, getTerminateAtEnd, setSynchronizedMode:, getSynchronizedMode, and getCurrentTimebase from ScheduleActivity protocol. Add documentation tags throughout.

1997-12-17 activity.h *mgd*

Constify argument to _activity_context_error.

Action

Name

`Action` — An action type that has been customized for direct execution by an action interpreter.

Description

Action is a common supertype of all action types which may be created within an action plan. Each action is always controlled by a single action plan to which it belongs. This action plan is referred to as its owner. Given the action object, its owner plan may be obtained using the inherited getOwner message.

Actions are allocated in the same zone as their owner plan, and may be created only using one of the createAction messages on an ActionGroup or Schedule. Each of these messages returns the action that was created as its return value. Actions in an action plan may also be obtained by processing the plan using any of its messages inherited from its underlying collection. Actions may also be removed from an action plan using a remove message on the underlying collection.

(.. Note: currently, an action cannot be removed while it is currently being executed. This means that the function or message called by an action cannot itself remove that same action from its action plan. This restriction will be removed in the future.)

Separate subtypes of Action are defined for each of the various forms of createAction messages that create them. The current representation of these actions will be undergoing change as support for their parameter and return types is migrated into more basic support from the defobj library. Each action type provides messages to retrieve and set the values of all argument values bound into the action. These capabilities will remain, but different messages will eventually be supported. This documentation will be completed once the messages supported on Action types are finalized.

Protocols adopted by Action

Create *(see page 46)*

Drop *(see page 54)*

GetOwner *(see page 61)*

Methods

Phase: Using

- `- (void)` **`_performAction_:`** `(id <Activity>)`*`activity`*

ActionArgs

Name

`ActionArgs` — Supertype of ActionCall, ActionTo, and ActionForEach.

Description

The ActionArgs subtypes all implement a specific, hard-coded method for binding an action type to a fixed number of arguments. All the arguments must have types compatible with id type. Eventually, more generic methods for binding an action type to any number and types of arguments and return values will also be provided.

Protocols adopted by ActionArgs

None

Methods

Phase: Creating

- `- (void)`**`setArg3:`** *`arg3`*
- `- (void)`**`setArg2:`** *`arg2`*
- `- (void)`**`setArg1:`** *`arg1`*

Phase: Using

- `-` **`getArg3`**
- `-` **`getArg2`**
- `-` **`getArg1`**
- `- (unsigned)`**`getNArgs`**

ActionCall

Name

`ActionCall` — An action defined by calling a C function.

Description

An action defined by calling a C function.

Protocols adopted by ActionCall

Action *(see page 143)*

ActionArgs *(see page 144)*

RETURNABLE *(see page 66)*

Methods

Phase: Creating

- `- (void)`**`setFunctionPointer:`** `(func_t)`*`fptr`*

Phase: Using

- `- (func_t)`**`getFunctionPointer`**

ActionChanged

Name

`ActionChanged` — An action generated when actions changes from single to concurrent.

Description

An action generated when actions changes from single to concurrent.

Protocols adopted by ActionChanged

Action *(see page 143)*

RETURNABLE *(see page 66)*

Methods

None

ActionConcurrent

Name

`ActionConcurrent` — An action generated when two or more actions co-occur.

Description

An action generated when two or more actions co-occur.

Protocols adopted by ActionConcurrent

Action *(see page 143)*

RETURNABLE *(see page 66)*

Methods

Phase: Using

- `- (id <ActionGroup>)` **`getConcurrentGroup`**

ActionCreating

Name

`ActionCreating` — Protocol shared by ActionGroup and Schedule.

Description

ActionCreating defines the createAction messages for ActionGroup just so that this protocol may be shared with Schedule, where they provide a convenience interface for the creation of actions in the schedule at time zero.

The createAction messages declare all arguments of the message to be of object id type, but you are free to cast other pointers and values up to the limits defined by the global portability assumptions. These is not portable across all machine architectures, but is expected to be portable across the 32-bit and 64-bit architectures on which Swarm will be supported. The message you send must still be declared to receive the type of argument you actually pass, before you cast it to the id type.

(.. Alternative approaches to argument typing are currently in development, but these will supplement rather than replace the current forms of createAction messages.)

Each of the createAction messages returns the action object which it creates. Each different kind of createAction message returns a different type of Action object with a matching name. These Action objects provide access to all the information with which the Action was initialized. The complete set of Action object types is defined below together with the messages that may be used to access their contents. (.. The implementation of the Action objects is currently undergoing change as the responsibility for parameter and return value typing gets taken over by ActionType in defobj.)

Protocols adopted by ActionCreating

FActionCreating *(see page 167)*

ActionCreatingCall *(see page 148)*

ActionCreatingTo *(see page 150)*

ActionCreatingForEach *(see page 149)*

FActionCreatingForEachHeterogeneous *(see page 168)*

FActionCreatingForEachHomogeneous *(see page 168)*

Methods

Phase: Using

- `-` **`createAction:`** *`anActionType`*

The createAction: message specifies that processing of another action type is to be performed by the action. The referenced action type is performed in its entirety, from start to finish, as the effect of the single created action.

ActionCreatingCall

Name

`ActionCreatingCall` — An action that calls a C function.

Description

The createActionCall: messages are similar to the createActionTo messages, except they specify the action to be performed as a binding of a C function to a list of argument values. The correct number of arguments for the function pointer passed as the initial argument must be supplied.

Protocols adopted by ActionCreatingCall

None

Methods

Phase: Using

- `- (id <ActionCall>)`**`createActionCall:`** `(func_t)`*`fptr`*
- `- (id <ActionCall>)`**`createActionCall:`** `(func_t)`*`fptr`* **`:`** *`arg1`*
- `- (id <ActionCall>)`**`createActionCall:`** `(func_t)`*`fptr`* **`:`** *`arg1`* **`:`** *`arg2`*
- `- (id <ActionCall>)`**`createActionCall:`** `(func_t)`*`fptr`* **`:`** *`arg1`* **`:`** *`arg2`* **`:`** *`arg3`*

ActionCreatingForEach

Name

`ActionCreatingForEach` — Send a message to every item in target, which is assumed to be a collection.

Description

The createActionForEach: messages define a message to be sent in the same way as the createActionTo messages, but they assume that the object passed as the target argument is a collection object. They specify that each object available from that collection, using the standard messages of the Collection type in the collections library, is to receive the specified message.

Protocols adopted by ActionCreatingForEach

None

Methods

Phase: Using

- `- (id <ActionForEach>)`**`createActionForEach:`** *`target`* **`message:`** `(SEL)`*`aSel`*
- `- (id <ActionForEach>)`**`createActionForEach:`** *`target`* **`message:`** `(SEL)`*`aSel`* **`:`** *`arg1`*
- `- (id <ActionForEach>)`**`createActionForEach:`** *`target`* **`message:`** `(SEL)`*`aSel`* **`:`** *`arg1`* **`:`** *`arg2`*
- `- (id <ActionForEach>)`**`createActionForEach:`** *`target`* **`message:`** `(SEL)`*`aSel`* **`:`** *`arg1`* **`:`** *`arg2`* **`:`** *`arg3`*
- `- (id <ActionForEachHomogeneous>)`**`createActionForEachHomogeneous:`** *`target`* **`message:`** `(SEL)`*`aSel`*

ActionCreatingTo

Name

`ActionCreatingTo` — An action that sends a message to an object.

Description

A createActionTo: message specifies that the action to be performed is defined by an Objective C message selector to be performed on a receiving object plus any required arguments. The message selector is specified by the message: argument and the receiving object is specified as the first argument, target.

Protocols adopted by ActionCreatingTo

None

Methods

Phase: Using

- `- (id <ActionTo>)`**`createActionTo:`** *`target`* **`message:`** `(SEL)`*`aSel`*
- `- (id <ActionTo>)`**`createActionTo:`** *`target`* **`message:`** `(SEL)`*`aSel`* **`:`** *`arg1`*
- `- (id <ActionTo>)`**`createActionTo:`** *`target`* **`message:`** `(SEL)`*`aSel`* **`:`** *`arg1`* **`:`** *`arg2`*
- `- (id <ActionTo>)`**`createActionTo:`** *`target`* **`message:`** `(SEL)`*`aSel`* **`:`** *`arg1`* **`:`** *`arg2`* **`:`** *`arg3`*

Examples

Example #1

```
[anActionGroup createActionTo: aTurtle message: M(move:) : obj];
```

ActionForEach

Name

`ActionForEach` — An action defined by sending a message to every member of a collection.

Description

An action defined by sending a message to every member of a collection.

Protocols adopted by ActionForEach

ActionTo *(see page 155)*

DefaultOrder *(see page 167)*

RETURNABLE *(see page 66)*

Methods

None

ActionForEachHomogeneous

Name

`ActionForEachHomogeneous` — Like ActionForEach, except that the collection must be homogeneous.

Description

Like ActionForEach, except that the collection must be homogeneous.

Protocols adopted by ActionForEachHomogeneous

Action *(see page 143)*

ActionTarget *(see page 154)*

ActionSelector *(see page 154)*

DefaultOrder *(see page 167)*

RETURNABLE *(see page 66)*

Methods

None

ActionGroup

Name

`ActionGroup` — A collection of actions under total or partial order constraints.

Description

An action group is an action plan whose basic representation is a sequence of actions that have been created within it.

An action group inherits its underlying representation from the OrderedSet type of the collections library. All the members of the ordered set must consist only of actions that are created by one of the createAction messages defined on ActionGroup itself. Once the actions are created, they may be accessed or traversed using standard messages of the OrderedSet type.

The action objects are an integral, controlled component of the action plan in which they are created. If they are removed from the action plan collection using a remove message, the only collection in which they may be reinserted is the same collection from which they came. It is permissible, however, to modify the base representation sequence by removing from one position and reinserting at another.

Protocols adopted by ActionGroup

CompoundAction *(see page 162)*

ActionCreating *(see page 147)*

DefaultOrder *(see page 167)*

CREATABLE *(see page 44)*

Methods

None

ActionSelector

Name

`ActionSelector` — Messages for actions involving a selector.

Description

Messages for actions involving a selector.

Protocols adopted by ActionSelector

None

Methods

Phase: Setting

- `- (void)`**`setMessageSelector:`** `(SEL)`*`aSel`*

Phase: Using

- `- (SEL)`**`getMessageSelector`**

ActionTarget

Name

`ActionTarget` — Messages common to actions that are sent to an object.

Description

Messages common to actions that are sent to an object.

Protocols adopted by ActionTarget

None

Methods

Phase: Creating

- `- (void)`**`setTarget:`** *`target`*

Phase: Using

- `-` **`getTarget`**

ActionTo

Name

`ActionTo` — An action defined by sending an Objective C message.

Description

An action defined by sending an Objective C message.

Protocols adopted by ActionTo

Action *(see page 143)*

ActionTarget *(see page 154)*

ActionSelector *(see page 154)*

ActionArgs *(see page 144)*

RETURNABLE *(see page 66)*

Methods

None

ActionType

Name

`ActionType` — Specification of an executable process.

Description

An action type is a type of process that may be initiated as a unit of execution by an external request. A typical action has a well-defined duration determined by a fixed set of actions that execute within it. Externally initiated interaction typically occurs only at the start or end of the overall process. A typical action is executed in its entirety once an external request that initiates it has occurred. Some actions may also have internal events that cannot begin or complete until other actions from a containing environment have also begun or completed their execution. Such ordering constraints can be defined either within an action type or as part of a dynamic context of execution.

Executable actions include both actions compiled in a host language (such as C functions or Objective C messages) and compound actions built at runtime for interpretation by the Swarm abstract machine.

(.. For now, the only subtype of ActionType is CompoundAction. Types for compiled actions such as functions and messages have not been defined yet. ..)

Protocols adopted by ActionType

None

Methods

Phase: Using

- `- (id <Activity>)`**`activateIn:`** `(id <Swarm>)`*`swarmContext`*

  The activateIn: message is used to initialize a process for executing the actions of an ActionType. This process is controlled by an object called an Activity. The activateIn message initializes an activity to run under the execution context passed as the swarmContext argument, and return the activity object just created. If the execution context is nil, an activity is returned that allows complete execution control by the caller. Otherwise, the execution context must be either an instance of SwarmProcess or SwarmActivity. (These objects are always maintained in one-to-one association with each other, either one of the pair is equivalent to the other as a swarmContext argument.)

  If a top-level activity is created (swarmContext is nil), the created activity may be processed using activity processing commands such as run, step, etc. If an activity is created to run under a swarm context, the swarm itself has responsibility for advancing the subactivity according to its requirements for synchronization and control among all its activities. Activating a plan for execution under a swarm turns over control to the swarm to execute the subactivity as a more-or-less autonomous activity.

ActivationOrder

Name

`ActivationOrder` — Default type used as concurrent group of a swarm.

Description

Concurrent group to order merge by activation order within swarm.

Protocols adopted by ActivationOrder

ActionGroup *(see page 153)*

RETURNABLE *(see page 66)*

Methods

Phase: Using

- `- ` **`remove:`** *`mergeAction`*

  `Method to remove concurrent merge action from sorted group`

- `- (void)` **`addLast:`** *`mergeAction`*

  `Method to sort concurrent merge actions in the order of swarm activation.`

Activity

Name

`Activity` — A level of processing by the interpreter of an action type.

Description

A object of type Activity implements the processing of actions within an action plan. Each subtype of action plan has a corresponding subtype of Activity that implements processing of its respective type of plan.

The Activity types are part of the virtual processing machinery of an action plan interpreter. All the important elements of this machinery, including their current state of processing, are exposed to aid in development of general-purpose tools such as debuggers. Except for applications that need to create and run their own reflective activities, direct use of activity types by a modeling application should be rare.

A new activity object is created by each activateIn: message sent to an action type. activateIn: initializes a new activity object and prepares it for processing, but does not itself execute any actions. Subsequent messages, such as run, must be used to initiate processing within an activity. If an activity is activated to run under a swarm context, the owning swarm itself controls all processing within the subactivity.

An Activity type holds a tree of currently running, or a potentially runnable, subactivities. Each activity object represents a state of processing within a single action plan. The structure is a tree because multiple activities could be running or eligible to run at the same time. This occurs whenever two activities are specified as concurrent, either because are being performed by concurrent actions in some other plan, or because they were started for autonomous execution under the same swarm. When parallel processing machinery is present, each activity could also be advancing its own state independent of those of any other activity.

The current implementation supports only a single serial processor. Under serial execution, only one activity may be active at one time. This means that the current state of processing may be represented by a single stack of current activities that traces a path to a single leaf of the runnable activity tree. When running in serial mode, messages are available to obtain the currently running leaf activity or useful context information such as the current time available from it.

Protocols adopted by Activity

DefinedObject *(see page 52)*

Drop *(see page 54)*

RETURNABLE *(see page 66)*

Methods

Phase: Using

- `- (id <Activity>)`**`getCurrentSubactivity`**

  `Get running subactivity or next subactivity to run.`

- - (BOOL)**getSerialMode**

  Return indicator for serial execution mode.

- - **setSerialMode:** (BOOL)*serialMode*

  Set serial execution mode.

- - (id <ScheduleActivity>)**getScheduleActivity**

  Return most immediately containing Schedule activity.

- - (id <SwarmActivity>)**getSwarmActivity**

  Return most immediately containing Swarm activity.

- - (id <Activity>)**getTopLevelActivity**

  Return top of activity tree running the local activity.

- - (id <Activity>)**getControllingActivity**

  Return activity that issued current run request on top-level activity.

- - **getSubactivities**

  Return set of subactivities pending to be run.

- - (id <Activity>)**getOwnerActivity**

  Return activity under which this activity is running.

- - (void)**setOwnerActivity:** *ownerActivity*

  Change owner from one swarm activity to another.

- - **getActionType**

  Get action type of action being performed by activity.

- - (id <Action>)**getAction**

  Get action containing parameter bindings for the local activity.

- - (id <Symbol>)**getHoldType**

  The getHoldType returns a code for the particular hold constraint under which an activity is currently holding (HoldStart or HoldEnd). It returns nil if the basic status of the activity is not Holding.

  (.. Currently no hold constraints other than merging within a swarm are supported, and this message always returns nil.)

- - (id <Symbol>)**getStatus**

  The getStatus message returns one of the following codes for the current run status of a particular activity: Initialized, Running, Stopped, Holding, Released, Terminated, Completed.

- - (id <Symbol>)**stepAction**

  The step message executes a single action within a tree of activities.

- - (id <Symbol>)**nextAction**

  The next executes actions within a single compound action while skipping over all processing of any complete action plans executed by those actions.

- - (void)**terminate**

Terminate also stops a running tree of activities, but sets all activities within the tree to a status of Completed whenever the tree is next run. terminate may be used on either a running or stopped tree of activities. It is the standard way to terminate schedule that is endlessly repeating under the RepeatInterval option.

- - (void)**stop**

  The stop message causes the a currently running tree of activities to stop further processing and return a Stopped status.

- - (id <Symbol>)**run**

  The run message continue processing of an activity from any state in which its execution has been suspended. An error is raised if the activity is already running. run returns either a Stopped or Completed status that the activity has when it is no longer eligible to be run further.

ActivityIndex

Name

ActivityIndex — Additional methods used by indexes over activities.

Description

Additional methods used by indexes over activities.

Protocols adopted by ActivityIndex

Index *(see page 108)*

RETURNABLE *(see page 66)*

Methods

Phase: Using

- - (id <Action>)**nextAction:** (id *)*status*
- - (id <Symbol>)**getHoldType**

AutoDrop

Name

`AutoDrop` — Specify that an action is dropped after being processed.

Description

The AutoDrop option specifies that as soon as any action been processed by a running activity, the action is removed from the plan and dropped so that it will never appear again. This option is useful for plans that are created for a one-time use, never to be used again. This option is especially useful for a dynamic schedule that continually receives new actions to be executed at future times, but will never repeat actions that were previously scheduled. Depending on the underlying implementation of the schedule, making sure the old actions get dropped using AutoDrop can considerably improve the performance of the schedule.

When an option like AutoDrop is used, or whenever the contents of an action plan undergo a significant amount of dynamic update, the action plan would ordinarily be intended only for a single point of use. If AutoDrop is specified, a restriction against multiple active references is enforced. An error will be raised if two activities ever attempt to process a plan with AutoDrop enabled at the same time.

Protocols adopted by AutoDrop

None

Methods

Phase: Creating

- `- ` **`setAutoDrop:`** `(BOOL)`*`autoDrop`*

Phase: Using

- `- (BOOL)` **`getAutoDrop`**

CompoundAction

Name

`CompoundAction` — A collection of actions to be performed in any order consistent with a set of ordering constraints.

Description

An compound action is the supertype of ActionGroup and Schedule. A compound action defines an executable process that is composed from the execution of a set of actions in some defined order.

CompoundAction is not directly creatable. One of its subtypes must be created instead. ActionPlan inherits the basic ability to be activated for execution from ActionType.

Protocols adopted by CompoundAction

ActionType *(see page 156)*

Collection *(see page 103)*

AutoDrop *(see page 161)*

DefaultOrder *(see page 167)*

Methods

None

ConcurrentGroup

Name

`ConcurrentGroup` — Default type used as concurrent group of a schedule.

Description

Default type used as concurrent group of a schedule.

Protocols adopted by ConcurrentGroup

ActionGroup *(see page 153)*

RETURNABLE *(see page 66)*

Methods

Phase: Using

- `- _getEmptyActionConcurrent_`
- `- (void)_setActionConcurrent_:` *`action`*

ConcurrentGroupType

Name

`ConcurrentGroupType` — Handle actions scheduled at same time value.

Description

The ConcurrentGroupType option is used to control handling of multiple actions which end up being scheduled at the same time value. As far the schedule representation is concerned, these actions are assumed by default to be concurrently executable, and the processing machinery is free to process them as such if no ConcurrentGroupType option is specified.

If a different interpretation of actions at the same time step is needed, the ConcurrentGroupType option may be specified. The argument of this option must be an object that when given a standard create: message will return an object having all the structure of a standard ActionGroup object.

In addition to overriding the standard ActionGroup type, the concurrent group type may be implemented by a custom subclass of the ActionGroup implementation which you supply yourself. A custom subclass is free to implement custom rules for how to combine all the actions which happen to arrive at the same time value. For example, it could decide that some actions are not to be executed at all, or it could create entirely new actions to be executed instead of or in addition to those which were originally scheduled.

(.. Specific rules for writing a custom ActionGroup subclass will be documented at a future time, but all the apparatus to do so is present today. A concurrent action group is currently used to maintain the proper order of execution among subswarms of an owner swarm.)

Protocols adopted by ConcurrentGroupType

None

Methods

Phase: Setting

- `- (void)` **`setConcurrentGroupType:`** *`groupType`*

Phase: Using

- `-` **`getConcurrentGroupType`**

ConcurrentSchedule

Name

`ConcurrentSchedule` — Time-based map usable for concurrent group.

Description

Time-based map usable for concurrent group.

Protocols adopted by ConcurrentSchedule

ConcurrentGroup *(see page 163)*

Schedule *(see page 173)*

CREATABLE *(see page 44)*

Methods

None

DefaultOrder

Name

`DefaultOrder` — The DefaultOrder option indicates the ordering to be assumed among actions of the plan when no other explicit ordering has been assigned.

Description

The DefaultOrder option indicates the ordering to be assumed among actions of the plan when no other explicit ordering has been assigned. Beyond this initial ordering, additional ordering constraints can be added selectively using partial order specifications on individual actions. (.. Partial order order constraints are not yet implemented.)

The value for DefaultOrder is a symbol that may have one of the following values: Concurrent, Sequential, Randomized;

The Concurrent value of the DefaultOrder option indicates that can actions be run in any order (including at the same time, if hardware and software to do this is available) without no impact on the net outcome of the actions. The claim that action results are independent of their execution order gives the processing machinery explicit leeway to execute them in any order it chooses. In the current implementation on a single, serial processor, actions are always processed sequentially even if marked concurrent, because that is the only way they can be. In future versions, however, special runtime processing modes may be defined even for a serial processor, which would mix up execution order just to confirm the independence of model results.

The Sequential value for the DefaultOrder option is the default. It specifies that the actions must always be executed in the same order as they occur in the plan. This order is ordinarily the same order in which actions are first created in the plan, unless actions are explicitly added elsewhere the collection that underlies a plan. This option is always the safest to assure predictability of results, but it excludes the ability to run the actions in parallel. To better understand and document a model design, it is worth annotating action plans with an explicit indication as to whether they do or do not depend on a Sequential order.

The Randomized value for the DefaultOrder option specifies that the model results do depend on execution order, but that the order in which the actions were created or added has no special significance. Instead, the method of dealing with order dependence is to generate a random order each time a collection of same-time actions is processed. The random order will be generated from an random number generator internal to the processing machinery.

Protocols adopted by DefaultOrder

None

Methods

Phase: Setting

- `-` **`setDefaultOrder:`** `(id <Symbol>)`*`aSymbol`*

Phase: Using

- `- (id <Symbol>)`**`getDefaultOrder`**

FAction

Name

`FAction` — An action defined by sending a FCall.

Description

An action defined by sending a FCall.

Protocols adopted by FAction

Action *(see page 143)*

RETURNABLE *(see page 66)*

Methods

Phase: Creating

- `-` **`setCall:`** *`fcall`*

Phase: Setting

- `-` **`setAutoDrop:`** `(BOOL)`*`autoDrop`*

FActionCreating

Name

`FActionCreating` — An action that calls a FCall.

Description

The createFAction: message creates an action that runs a FCall closure.

Protocols adopted by FActionCreating

None

Methods

Phase: Using

- `- (id <FAction>)`**`createFAction:`** `(id <FCall>)`*`call`*

FActionCreatingForEachHeterogeneous

Name

`FActionCreatingForEachHeterogeneous` — Invoke a FCall for every item in the target collection, which can include objects of various types.

Description

Invoke a FCall for every item in the target collection, which can include objects of various types.

Protocols adopted by FActionCreatingForEachHeterogeneous

None

Methods

Phase: Using

- `- (id <FActionForEachHeterogeneous>)`**`createFActionForEachHeterogeneous:`** *`target`* **`call:`** `(id <FCall>)`*`call`*

FActionCreatingForEachHomogeneous

Name

`FActionCreatingForEachHomogeneous` — Invoke a FCall for every item in the target collection. All members must be of the same type.

Description

Invoke a FCall for every item in the target collection. All members must be of the same type.

Protocols adopted by FActionCreatingForEachHomogeneous

None

Methods

Phase: Using

- `- (id <FActionForEachHomogeneous>)`**`createFActionForEachHomogeneous:`** *`target`* **`call:`** `(id <FCall>)`*`call`*

FActionForEach

Name

`FActionForEach` — Base protocol for FActionForEach{Homogeneous,Heterogeneous}.

Description

Base protocol for FActionForEach{Homogeneous,Heterogeneous}.

Protocols adopted by FActionForEach

FAction *(see page 167)*

ActionTarget *(see page 154)*

DefaultOrder *(see page 167)*

Methods

None

FActionForEachHeterogeneous

Name

`FActionForEachHeterogeneous` — An action defined by applying a FAction to every member of a collection.

Description

An action defined by applying a FAction to every member of a collection.

Protocols adopted by FActionForEachHeterogeneous

FActionForEach *(see page 169)*

RETURNABLE *(see page 66)*

Methods

None

FActionForEachHomogeneous

Name

`FActionForEachHomogeneous` — An action defined by applying a FAction to every member of a collection All members of the collection must be of the same type.

Description

An action defined by applying a FAction to every member of a collection. All members of the collection must be of the same type.

Protocols adopted by FActionForEachHomogeneous

FActionForEach *(see page 169)*

RETURNABLE *(see page 66)*

Methods

None

ForEachActivity

Name

`ForEachActivity` — State of execution within a ForEach action.

Description

State of execution within a ForEach action.

Protocols adopted by ForEachActivity

Activity *(see page 158)*

RETURNABLE *(see page 66)*

Methods

Phase: Using

- `-` **`getCurrentMember`**

GetSubactivityAction

Name

`GetSubactivityAction` — Declare an internal method for getCurrentAction().

Description

Declare an internal method for getCurrentAction().

Protocols adopted by GetSubactivityAction

None

Methods

Phase: Using

- `- _getSubactivityAction_`

RelativeTime

Name

`RelativeTime` — Specifies that time is relative to when the schedule started.

Description

The RelativeTime option specifies that all the times in the schedule are relative to the time when processing of the entire schedule begins. Otherwise, the times are assumed to be absolute times, with their base in the starting time of the entire model.

Protocols adopted by RelativeTime

None

Methods

Phase: Setting

- `- setRelativeTime: (BOOL)relativeTime`

Phase: Using

- `- (BOOL)getRelativeTime`

RepeatInterval

Name

`RepeatInterval` — Reschedule actions after a period of time.

Description

The RepeatInterval option specifies that as soon as all actions in the schedule have completed, it is to be rescheduled at a time computed as the time at which the schedule was last started, plus the value specified as the RepeatInterval argument. This option overrides the normal default that times are considered absolute, as if the RelativeTime option had also been specified at the same time. All scheduled times must be less than the specified repeat interval, or an error will be raised. The RepeatInterval option can continue to be reassigned to different values after a schedule has been created, but the interval value must always be greater than the scheduled times of any actions which it contains. (.. This option is currently supported only on schedule, not swarms.)

Protocols adopted by RepeatInterval

None

Methods

Phase: Setting

- `- **setRepeatInterval:** (timeval_t)*repeatInterval*`

Phase: Using

- `- (timeval_t)**getRepeatInterval**`

Schedule

Name

`Schedule` — A collection of actions ordered by time values.

Description

A schedule is compound action whose basic representation is a sorted Map of actions that have been created within it. The key value associated with each of these actions is an unsigned integer value for which the typedef timeval_t is supplied.

A schedule inherits its underlying representation from the Map type of the collections library. All the members of the ordered set must consist only of actions that are created by one of the createAction messages defined on Schedule itself. Once the actions are created, they may be accessed or traversed using standard messages of the Map type. The key values of this collection, however, must be cast to and from the id type defined for key values by the Map type.

The messages to create actions within a schedule are essentially the same as those for ActionGroup, except for the presence of an initial at: argument indicating the time at which an action is to be performed. Except for the time associated with each action, meaning of the createAction messages is the same as for ActionGroup.

When multiple actions are all scheduled at the same time, they are all inserted into a concurrent action group created for that time value. The ConcurrentGroupType option may be used to override the default action group for these concurrent actions by a custom user-defined subclass. (.. Details of doing this are not yet documented, but there are examples.)

Protocols adopted by Schedule

Map *(see page 119)*

CompoundAction *(see page 162)*

ActionCreating *(see page 147)*

RelativeTime *(see page 171)*

RepeatInterval *(see page 172)*

ConcurrentGroupType *(see page 164)*

SingletonGroups *(see page 176)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- - **setKeepEmptyFlag:** (BOOL)*keepEmptyFlag*

  Indicate whether an empty schedule should be dropped and ignored or or kept and attended to (default is YES).

- + **create:** (id <Zone>)*aZone* **setAutoDrop:** (BOOL)*autoDrop*

  Convenience method for creating an AutoDrop Schedule

- + **create:** (id <Zone>)*aZone* **setRepeatInterval:** (timeval_t)*rptInterval*

  Convenience method for creating a repeating Schedule

Phase: Using

- - (id <ActionGroup>)**insertGroup:** (timeval_t)*tVal*
- - **remove:** *anAction*

  Remove action from either schedule or concurrent group.

- - (id <FActionForEachHomogeneous>)**at:** (timeval_t)*tVal* **createFActionForEachHomogeneous:** *target* **call:** (id <FCall>)*call*
- - (id <FActionForEachHeterogeneous>)**at:** (timeval_t)*tVal* **createFActionForEachHeterogeneous:** *target* **call:** (id <FCall>)*call*
- - (id <ActionForEach>)**at:** (timeval_t)*tVal* **createActionForEach:** *target* **message:** (SEL)*aSel*
- - (id <ActionForEach>)**at:** (timeval_t)*tVal* **createActionForEach:** *target* **message:** (SEL)*aSel* **:** *arg1*
- - (id <ActionForEach>)**at:** (timeval_t)*tVal* **createActionForEach:** *target* **message:** (SEL)*aSel* **:** *arg1* **:** *arg2*
- - (id <ActionForEach>)**at:** (timeval_t)*tVal* **createActionForEach:** *target* **message:** (SEL)*aSel* **:** *arg1* **:** *arg2* **:** *arg3*
- - (id <ActionForEachHomogeneous>)**at:** (timeval_t)*tVal* **createActionForEachHomogeneous:** *target* **message:** (SEL)*aSel*
- - (id <ActionTo>)**at:** (timeval_t)*tVal* **createActionTo:** *target* **message:** (SEL)*aSel*
- - (id <ActionTo>)**at:** (timeval_t)*tVal* **createActionTo:** *target* **message:** (SEL)*aSel* **:** *arg1*
- - (id <ActionTo>)**at:** (timeval_t)*tVal* **createActionTo:** *target* **message:** (SEL)*aSel* **:** *arg1* **:** *arg2*
- - (id <ActionTo>)**at:** (timeval_t)*tVal* **createActionTo:** *target* **message:** (SEL)*aSel* **:** *arg1* **:** *arg2* **:** *arg3*
- - (id <ActionCall>)**at:** (timeval_t)*tVal* **createActionCall:** (func_t)*fptr*
- - (id <ActionCall>)**at:** (timeval_t)*tVal* **createActionCall:** (func_t)*fptr* **:** *arg1*
- - (id <ActionCall>)**at:** (timeval_t)*tVal* **createActionCall:** (func_t)*fptr* **:** *arg1* **:** *arg2*
- - (id <ActionCall>)**at:** (timeval_t)*tVal* **createActionCall:** (func_t)*fptr* **:** *arg1* **:** *arg2* **:** *arg3*

- `- (id <FAction>)`**`at:`**`(timeval_t)`*`tVal`* **`createFAction:`**`(id <FCall>)`*`call`*
- `-` **`at:`**`(timeval_t)`*`tVal`* **`createAction:`** *`anActionType`*

ScheduleActivity

Name

`ScheduleActivity` — State of execution within a Schedule.

Description

State of execution within a Schedule.

Protocols adopted by ScheduleActivity

Activity *(see page 158)*

RETURNABLE *(see page 66)*

Methods

Phase: Using

- `-` **`stepUntil:`**`(timeval_t)`*`tVal`*

  `Advance activity until requested time has been reached.`
- `- (timeval_t)`**`getCurrentTime`**

  `Get current time of activity (pending time if holding).`

SingletonGroups

Name

`SingletonGroups` — Indicates that an action group should be created for every time value which is present.

Description

The SingletonGroups option indicates that an action group should be created for every time value which is present, even when only a single action is present at the time value.

Ordinarily, a concurrent action group is created to process actions at the same timestep only if more than one action is scheduled at that timestep. The overhead of these action groups is relatively low, because it just creates a single new object to which actions are directly linked, but it is still faster to avoid creating them if only one action is present at a timestep. If a custom subclass is being provided however, it may need to examine the actions at a timestep even if there is only one.

Protocols adopted by SingletonGroups

None

Methods

Phase: Setting

- `- (void)`**`setSingletonGroups:`** `(BOOL)`*`singletonGroups`*

Phase: Using

- `- (BOOL)`**`getSingletonGroups`**

SwarmActivity

Name

`SwarmActivity` — A collection of started subactivities.

Description

A collection of started subactivities.

Protocols adopted by SwarmActivity

ScheduleActivity *(see page 175)*

RETURNABLE *(see page 66)*

Methods

Phase: Using

- `- (id <Schedule>)` **`getSynchronizationSchedule`**
- `-` **`getSwarm`**

  `Return swarm object containing this swarm activity, if any.`

SwarmProcess

Name

SwarmProcess — An object that holds a collection of concurrent subprocesses.

Description

SwarmProcess inherits the messages of both ActionType and Zone. Inheritance of zone behavior means that a swarm can be used as the argument of a create: or createBegin: message, for creation of an object within the internal zone of a swarm.

Unlike other action types, swarms and swarm activities always exist in a one-to-one relationship, provided that the swarm has been activated. This restriction to a single activity enables the swarm to do double-duty as a custom object that provides its own interface to the activities running within the swarm.

Protocols adopted by SwarmProcess

ActionType *(see page 156)*

Zone *(see page 76)*

SynchronizationType *(see page 179)*

Methods

Phase: Creating

- - **setInternalZoneType:** *aZoneType*

  The InternalZoneType option sets the type of zone which is created by the swarm to hold objects created within the swarm. If set to nil, no internal zone is created within the swarm, and all use of the swarm as if it were a zone raises an error. The default of this option is standard Zone type. (.. Since there is no other Zone type yet, there's no reason to set this option yet except to turn off the internal zone. ..)

Phase: Using

- - (id <SwarmActivity>)**getActivity**

  getActivity returns the activity which is currently running of subactivities within the swarm. This activity is the same as the value returned by activateIn: when the swarm was first activated. It returns nil if the swarm has not yet been activated.

- - **getInternalZone**

  getInternalZone returns a Zone object that is used by the swarm to hold its internal objects. Even though the swarm itself inherits from Zone and can be used as a Zone for nearly all purposes, this message is also provided so that the zone itself can be obtained independent of all zone behavior.

SynchronizationType

Name

`SynchronizationType` — Synchronization type sets the type of schedule which is used internally by the swarm to synchronize subschedules.

Description

Synchronization type sets the type of schedule which is used internally by the swarm to synchronize subschedules. Its default is a schedule with a concurrent group of ActivationOrder.

The default value for the SynchronizationType option is not a generic action group, but a special predefined subtype of ActionGroup called ActivationOrder. This concurrent group type guarantees that actions scheduled to occur at the same time from different action plans running in the same swarm are executed in the same order in which the action plans were first activated.

Protocols adopted by SynchronizationType

None

Methods

Phase: Creating

- `- (void)`**`setSynchronizationType:`** *`aScheduleType`*

Phase: Using

- `-` **`getSynchronizationType`**

General

Name

`activity` — Processing control over all levels of Swarm execution

Description

The activity library is responsible for scheduling actions to occur within a simulated world, and for making these actions actually happen at the right time in the right order. It provides the foundation of dynamic, object-oriented simulation within Swarm.

Actions consist of messages to objects, calls to functions, or groups of actions in some defined order. The activity library guarantees that all these actions, and the state changes they produce, occur at predictable points in time. Time is defined by the relative order of actions, and may also be indexed by the discrete values of a world clock.

Macros

- **COMPLETEDP**(*status*)
- DEFINED_timeval_t
- **HOLDINGP**(*status*)
- **INITIALIZEDP**(*status*)
- **RELEASEDP**(*status*)
- **RUNNINGP**(*status*)
- **STOPPEDP**(*status*)
- **TERMINATEDP**(*status*)
- **getCurrentAction**()
- **getCurrentActivity**()
- **getCurrentOwnerActivity**()
- **getCurrentSchedule**()
- **getCurrentScheduleActivity**()
- **getCurrentSwarm**()
- **getCurrentSwarmActivity**()
- **getCurrentTime**()

  Macro to get the time of the current activity - only valid when an activity is actually running. Returns a (timeval_t).

- **getTopLevelActivity**()

  Macro to get id of the current topLevelActivity - only valid when an activity is actually running.

Functions

- `id` **`_activity_context_error`**`(const char *`*`macroName`*`)`

  `Internal error message issued when a current activity is missing.`

Typedefs

- timeval_t unsigned long

Globals

id <Symbol> Initialized

Values returned by getStatus.

id <Symbol> Running

Values returned by getStatus.

id <Symbol> Holding

Values returned by getStatus.

id <Symbol> Released

Values returned by getStatus.

id <Symbol> Stopped

Values returned by getStatus.

id <Symbol> Terminated

Values returned by getStatus.

id <Symbol> Completed

Values returned by getStatus.

id <Symbol> Concurrent

values for DefaultOrder

id <Symbol> Sequential

values for DefaultOrder

id <Symbol> Randomized

values for DefaultOrder

id <Symbol> HoldStart

Values returned by getHoldType.

id <Symbol> HoldEnd

Values returned by getHoldType.

id _activity_current

Internal variable used by current context macros.

id <Error> InvalidSwarmZone

Error issued when an internal zone is expected, but absent.

id _activity_zone

_activity_zone -- zone in which activity objects created

BOOL (*) (id) _activity_trace

trace function for activity execution global variable for function to be called on every change in the activity tree Note: support for any specific form of this trace facility is not guaranteed in future versions. Some form of trace facility will remain for debug purposes, however, at least until a full form of event history logging has been implemented as an integral part of the Activity type.

Objectbase Library

Overview

The *objectbase* library encapsulates various fundamental aspects of the *Swarm* object model and defines the probing machinery used to take data from *Swarm* objects. Most of the underlying functionality of the classes defined here is contained in the *Defobj Library (see page 26)* and *Activity Library (see page 133)* libraries.

Probes. Probes are idealized entities that are intended to allow the user to monitor and modify the innards of objects without explicitly providing the functionality to do so at compile time. Hence, they allow dynamic interaction with an objects instance variables and methods. Most of the functionality of probes is implemented here; but, at present, they are intimately linked to the *Simtoolsgui Library (see page 291)* library, which contains all the widgetry needed to use probes from a GUI. Despite this intimacy, probes are intended to be a general purpose mechanism for *any* agent or device to interact dynamically with *Swarm* objects. The Probes section in the *Swarm User Guide (http://www.swarm.org)* will explain more about the reasoning and purpose behind probes.

1. Dependencies

Following are the other header files imported by <objectbase.h>:

```
#import <defobj.h>
#import <activity.h>
```

The *defobj* library interface is included to provide the basic object support.

2. Compatibility

- **1.0.4 => 1.0.5.** No changes.
- **1.0.3 => 1.0.4.** The name of this library is now *`objectbase`*, it is has been renamed from *`swarmobject`* largely to reflect the more generic nature of the library and also motivated by the impending port of Swarm to Windows NT (to avoid filename conflicts with the `SwarmObject` class). There should be little, or no effect on the user, the only visible change is the fact that the actual library (.a) or (.so) file will now have a different name and the header file name has changed. A symbolic link from `objectbase.h` to `swarmobject.h` has been provided in the distribution, to ensure backwards compatibility, however, users should *not* continue to rely on this being so. Users should port references to `swarmobject.h` in their code to `objectbase.h`, because this symlink will be removed in a future release.

Note: this is no way affects the `SwarmObject` class which remains the same as in all previous releases.

- **1.0.0 => 1.0.1.** The interface has changed again! *EmptyProbeMap* is now a subclass of *CustomProbeMap*, which is subclassed from *ProbeMap*. And a shortcut `create:` method was added to that branch.

Also, a new method was added to *ProbeLibrary* called `isProbeMapDefinedFor` that serves to non-invasively test for the existence of a *ProbeMap* for a given class.

- ***Beta* => 1.0.0.** The new interface for the *swarmobject* library might cause some problems for apps that worked under the Beta release of Swarm. To get the whole scoop, read the Library Interface Conventions *(see page 406)*.

3. Usage Guide

3.1. Overview

The *objectbase* library contains the most basic objects users need to design their agents and swarms. It also serves, at present, as a repository for the probe machinery, which is provided for every *SwarmObject*. The way the classes in this library are to be used is varied. But, basically, it is provided so that the user will have something to subclass from for her own objects and Swarms.

3.2. Example Usage of *SwarmObject*

The best way to explain how the library should be used is to walk through an example. So, using Heatbugs, we'll walk through the ways *objectbase* is used and discuss them. Since more documentation is usually better than less, I'm going to explain things at a low level so that those not familiar with Objective C will understand the discussion. If you already are familiar with Objective C, then you should skip this part.

First off, the basic elements of the Heatbugs simulation are the heatbugs, the model swarm (which bundles the heatbugs), and the observer swarm (which bundles the displays of the probes poking into the model swarm and the heatbugs). The interface files for each show what must be imported and the declaration syntax needed to subclass from *SwarmObject*.

We'll use `Heatbug.h` for our discussion here. The first part of the file shows the C-preprocessor imports needed:

```
#import <objectbase/SwarmObject.h>
#import <space.h>
#import "HeatSpace.h"
#import <tkobjc/Raster.h>
```

The `#import <objectbase/SwarmObject.h>;` is included in order to subclass from *SwarmObject*. However, to provide backwards compatibility, we've placed this import in the library interface file `objectbase.h` as well, which means one could subclass from *SwarmObject* by simply importing the `objectbase.h` file. This is discouraged in order to make the library interfaces as standard as possible.

The next *objectbase* relevant piece of code in this file is:

```
@interface Heatbug: SwarmObject
{
  double unhappiness;
  int x, y;
  HeatValue idealTemperature;
  HeatValue outputHeat;
  float randomMoveProbability;
```

```
  Grid2d * world;
  int worldXSize, worldYSize;
  HeatSpace * heat;
  Color bugColor;
}
```

The `@interface` keyword indicates that you are beginning the definition of the part of an object (a *Heatbug* in this case) that will be visible to other objects. The `Heatbug: SwarmObject` indicates that you are calling this object *Heatbug* and it is a subclass of *SwarmObject*. What follows between the curly braces ({}) are the instance variables defined for the *Heatbug* class *above and beyond* those inherited from the *SwarmObject* class.

Inside this "agent," we have defined several parameters associated with either the agent, itself, or the space in which it sits. Any data that will need to be present throughout all the behavior and lifetime of the agent should be declared here. Also, anything declared here will be accessible to the probe machinery, and so will be capable of being manipulated and viewed from outside the agent.

Next come the message prototypes to which this agent will respond. And it is worth noting again that these are *in addition to* those declared in the *SwarmObject* superclass. So, not only will other objects be able to send messages to this agent that are declared here, but other objects will be able to send all the messages declared in the `objectbase/SwarmObject.h` imported previously. The messages prototyped here will dictate what the compiler thinks this object can respond to. Hence, if any part of any of these prototypes differs from the corresponding function definition in the `Heatbug.m` file, then the compiler will say something like `Object: aHeatbug does not respond to xyz`, where "xyz" is the name of the message that is being sent to the `Heatbug`. A script is provided with the Swarm distribution that fixes header file prototypes to match the message declarations in the corresponding ".m" file. This script should be in the `$SWARMHOME/bin` directory and is called `m2h`.

One more thing to notice about these prototypes is that some of them are duplicates of what appears in the `objectbase/SwarmObject.h` file. This means that when the message is called on a *Heatbug* object, it will execute the method defined here and not the one in the *SwarmObject* class. In the *objectbase* library, the following messages are intended to be overridden, as necessry: `create:, createBegin:, createEnd, customizeBegin:, customizeEnd, customizeCopy:, describe:, and getInstanceName.` Each of these messages do specific things that may change from subclass to subclass of *SwarmObject*. In this case, however, we're only overriding `createEnd`. The differences between we implement it in *Heatbugs* and the default is not that significant. But, it should be pointed out that when overriding certain messages, like `createBegin:` and `createEnd`, the new method should call the superclass' version of the message, as well. This is done using the default pointer to the superclass, designated *super*. The syntax in the *Heatbugs* case is:

```
[super createEnd];
```

The reasons for doing this are related to the object phase protocols used by *defobj*. If you would like more info on that, see the *Swarm User Guide (http://www.swarm.org)* .

Finally, the `@end` keyword signifies the end of the interface definition. GNU Objective C allows one to leave this off; but, it is not good practice.

And that's it. Of course, there're a few tricky aspects to using the *objectbase* library that weren't mentioned here. Some of them will be mentioned in the Advanced Usage Guide *(see page 190)* and the Implementation Notes *(see page 190)*; but, the best way to learn is to examine the way the demo applications do it and try to make some changes yourself.

3.3. Subclassing from *Swarm*

Subclassing from the *Swarm* class works very similar to subclassing from *SwarmObject*.

3.4. ActivityControl

The *ActivityControl* object provides much more finely grained control over the execution of an interactive simulation. It addresses both the problems of not being able to stop the simulation at any given point in any given activity and provides an initial step towards a Swarm debugger.

An activity controller can be attached to *any* activity that is created in a Swarm simulation, including those that are created for use only by the Swarm kernel. The controller then provides the basic activity manipulation messages on that activity, which are: `run, stop, next, step, stepUntil,` and `terminate`.

The presence of the *ActivityControl* object might cause some confusion about what role the *ControlPanel* should play in the controlled execution of the various schedules. The *ControlPanel* should still be used for the top-level control of any simulation that is running in a context where *random* interference is expected (like under a GUISwarm where the user may click a button at any time). The reason this is true is because the *ControlPanel* sends pseudo-interrupts to the infinite loop we use to perpetuate execution of the top level Swarm (which can only be seen in the form of the `go` message on a *GUISwarm* at present). *This type of control may change in the future!* But, for now, it is how we monitor the control state of the simulation.

Now, having said that, the *ControlPanel* should no longer be used to run the simulation. It should only be used to instantiate the control context and quit the entire simulation. That means that sometime in the future, the `Go` and the `Stop` buttons will be removed from the *ControlPanel* display. They have been left in for backwards compatibility so that applications that do not use the new *ActivityControl* will retain their (albeit handicapped) controllability. Also, the current `Time Step` button will be renamed to `Start` to be consistent with it's new purpose.

In order to use the new control mechanism, you must place code like the following in the top-level Swarm. (This code was taken from a modified mousetrap demo app.)

```
observerActCont = [ActivityControl createBegin: [self getZone]];
observerActCont = [observerActCont createEnd];
[observerActCont attachToActivity: [self getSwarmActivity]];
[probeDisplayManager createProbeDisplayFor: observerActCont];
```

This creates an *ActivityControl* and attaches it to the top-level activity (in this case an `observerSwarm`). It also creates a display for the controller. (The probe map for the *ActivityControl* class is designed within the *ActivityControl*, itself. This is done because all of these objects are expected to look the same to any outside object.) With this activity controller, you will then be able to `run, stop, next, step, stepUntil,` and `terminate` that activity.

There are some tricky aspects to successfully using an *ActivityControl* object that the Advanced Usage Guide *(see page 190)* will cover.

4. Advanced Usage Guide

4.1. ProbeMap design

When designing a *ProbeMap* for a given (subclass of) *SwarmObject*, inclusion of instance variables or messages defined in the super class might be desirable. The normal *ProbeMap* design code might look like (this code was taken from the tutorial app called "hello-world"):

```
probeMap = [CustomProbeMap createBegin: [self getZone]];
[probeMap setProbedClass: [person class]];
probeMap = [probeMap createEnd];
[probeMap addProbe: [probeLibrary getProbeForVariable: "room"
                                  inClass: [person class]]];
[probeMap addProbe: [probeLibrary getProbeForVariable: "party"
                                  inClass: [person class]]];
[probeMap addProbe: [probeLibrary getProbeForVariable: "name"
                                  inClass: [person class]]];
[probeMap addProbe: [probeLibrary getProbeForVariable: "stillhere"
                                  inClass: [person class]]];
[probeMap addProbe: [probeLibrary getProbeForVariable: "listOfFriends"
                                  inClass: [person class]]];
[probeMap addProbe: [probeLibrary getProbeForVariable: "myColor"
                                  inClass: [person class]]];
[probeLibrary setProbeMap: probeMap For: [person class]];
[probeDisplayManager createProbeDisplayFor: person];
```

where `room, party, name, stillhere, listOfFriends,` and `myColor` are instance variables declared in the interface to the *Person* subclass. And *Person* is a subclass of *Agent2d*, which is a subclass of *SwarmObject*.

Now let's say that there are two variables declared in *Agent2d* that you want to put into this custom probe in addition to the ones you've picked out of *Person*. Call them `x` and `y`. The way to add them to the `probeMap` is to add the following two lines of code to the above.

```
[probeMap addProbe: [probeLibrary getProbeForVariable: "x"
                                  inClass: [Agent2d class]]];
[probeMap addProbe: [probeLibrary getProbeForVariable: "y"
                                       inClass: [Agent2d class]]];
```

And that's it! The two superclass-declared variables, which are, in fact, instance variables of the instance of the subclass, are now included in the probe.

In addition, a convenience message has been added to the *CustomProbeMap* interface to compress the above rather cluttered mechanism into one message. This convenience message can be used in the usual case where a *ProbeMap* will consist of variables and messages from the same class. For example, the first part of the custom probe creation above can be shortened to:

```
probeMap = [CustomProbeMap create: [self getZone] forClass: [person class]
      withIdentifiers: "room", "party", "name", "stillhere",
                       "listOfFriends", "myColor", NULL];
```

And if the user wanted messages in the probe as well, it could be extended to:

```
probeMap = [CustomProbeMap create: [self getZone]
                           forClass: [person class]
                     withIdentifiers: "room", "party", "name",
                         "stillhere", "listOfFriends", "myColor",
                         ":",
                         "setWorld:Room:Party:",
                         "setPerson:Topic_array:ShowSpeech:",
                         NULL];
```

At present, this message doesn't search the superclasses for the message names listed here. But, that will soon be rectified.

4.2. ActivityControl Issues

It is completely reasonable to assume that explicit control can be had over all the activities in a simulation. However, at present, this control is limited because the context in which an activity runs determines how it behaves. To understand how an *ActivityControl* is to be used, we will have to explore the behavior of activities in context. (For a more complete explanation of the behavior of activities, see the *Activity Library (see page 133)* documentation.)

There are two ways to get an activity started, one can activate the activity in `nil` or in some other activity. So called "top-level" activities are activated in `nil` and are intended to be independent of the sophisticated scheduling activity that dictates the execution of actions in any other context in the simulation. I.e. the only activities that should be activated in `nil` are those sets of events that are expected to preserve the same behavior no matter what goes on in any other part of the simulation.

The other type of activity, those activated in some other activity, is intended to be an integral part of its owner activity. However, this doesn't mean that it *must* depend on the outcome of actions happening in the owner activity. In fact, an *ActionPlan* can be designated as containing actions that are capable of being processed in parallel, via code like the following:

```
[anActionPlan setDefaultOrder: Concurrent];
```

But these activities are *still* intended to be meshed with their owner activities. In other words, they are part and parcel of the same model or simulation.

Now, the operational effect of activating an activity in `nil` is that it will not be meshed with the rest of the Swarm activity structure. This gives the user (or process) complete control over the execution of that activity. A `run` on that activity will run either to completion or until a stop flag is set by a sequence of events purely internal to that activity. Or, one can stop it from the outside with a message from something like an *ActivityControl*.

What all this means is that, while one can attach an *ActivityControl* to *any* activity, only the "top-level" activities (those having been activated in `nil`) are going to respond well to it. Any sub-activity will respond half-heartedly, if at all. For example, in the *Mousetrap* demo distributed with Swarm, an *ActivityControl* has been placed on both the *ObserverSwarm* and the *ModelSwarm* activities. Now, if one sends a `run` message to the *ActivityControl* that is attached to the `observerSwarm`'s activity, the entire model continues to run to completion, unless the user sends a `stop` message. However, if the sim is stopped at some point, a `run` message to the `modelSwarm`'s activity will have no effect at all. (*Note:* If you do this via the activity controllers, you see the `currentTime` variable get updated; but, the actual `run` message to the activity, itself, has no effect.)

So, the rule of thumb, for the present, is to attach *ActivityControl* objects only to "top-level" activities, like the *ObserverSwarm*.

5. Subclassing Reference

The main classes defined here that are intended to be subclassed by users are Swarm and SwarmObject. The probing functionality provided here is mainly for use within the *Swarm* kernel. However, this probe machinery should be used when designing any interface between *Swarm* and any other agent or device.

SwarmObject. *SwarmObject* is the base class for all interactive agents in *Swarm*. It defines the standard behavior for Swarmstyle agents, which includes hooks for creation, probing, zoned memory allocation, and destruction.

Swarm. The *Swarm* class encapsulates all of the abstract execution machinery in the *activity* library, as well as the notion of a group of related objects. Metaphorically, a "Swarm" is a combination of a collection of objects and a schedule of activity over those objects. Hence, the *Swarm* class provides the behavior necessary for creating such an object and starting up the schedule.

Further details on subclassing are also described in the Usage Guide *(see page 190)*

6. Interface Design Notes

Unfortunately, this interface has not undergone a rigorous design review. As such it is subject to change in the future. However, there are rumours that this library will be integrated into the *Defobj Library* *(see page 26)* anyway. So, even though little thought was given to the design of this interface (and it is not likely to be worthwhile designing a robust interface at this time), it was implemented in order to provide a first step towards bringing all the various libraries in line with the standard set by *defobj*.

Along these lines, a few notes are relevant.

1. *Probe*s may become an inherent part of any object.
2. *ActivityControl*s will become a part of a larger set of tools used for debugging Swarm models.

7. Implementation Notes

1. Right now, *Probe*'s rely on a special method, `getInstanceName`, that has to be implemented in any probe-able object in order to get anything other than the class name of that object into the object designation widget. However, a more general capability has been added to *defobj* to give a meaningful name to any object. *Probe*s will be changed to take advantage of this new capability.
2. In the *ActivityControl* the frequency of the message `updateStateVar` is very high. It is sent at least once every cycle and *every* time a message is sent to the *ActivityControl*. This is unsatisfactory. Some of these messages can be pruned out of the object.
3. The `probeMap` designed for use with an *ActivityControl* was chosen fairly arbitrarily. Right now, it serves as a default for the class. A user can override it by designing a new one and inserting it into the `probeLibrary`.
4. Errors specific to objects in the *objectbase* library need to be gathered and initialized like those in the *defobj* library.

Documentation and Implementation Status

Revision History

2001-11-27 objectbase.h *alex*

(setNonInteractive): Fix doc string to not use an ampersand between `drag' and `drop', can confuse some XML processors.

2001-01-28 objectbase.h *mgd*

(val_t): Remove. (Moved to defobj.h.) (ProbeMap): Add setProbedObject:.

2001-01-27 objectbase.h *mgd*

(val_t): Change type to fcall_type_t.

2000-10-14 objectbase.h *mgd*

(ProbeMap): Make argument to addProbe: id <Probe>. (ProbeLibrary): Make first argument to setProbeMap:For: id <ProbeMap>.

2000-07-02 objectbase.h *mgd*

(ProbeMap): Change return type on begin: to Index.

2000-05-18 objectbase.h *mgd*

([Swarm -activateIn:]): Hide in a #ifndef IDL.

2000-04-27 objectbase.h *mgd*

([VarProbe -createEnd]): Remove. ([ProbeMap -getProbeForVariable:, -getProbeForMessage:]): Remove.

2000-03-28 *mgd*

Swarmdocs 2.1.1 frozen.

2000-03-28 objectbase.h *mgd*

Declare terminate and getActivity. Accomodate changes above.

2000-02-29 *mgd*

Swarmdocs 2.1 frozen.

2000-02-17 objectbase.h *mgd*

(ActivityControl): Remove terminateActivity.

2000-02-15 objectbase00.sgml *alex*

Remove LINK to probe APPENDIX, refer reader to Swarm User Guide.

1999-08-25 objectbase.h *mgd*

(VarProbe): Declare DefaultString, CharString, IntString.

1999-08-23 objectbase.h *mgd*

Add VarProbe, MessageProbe, and ProbeMap typing.

1999-08-22 objectbase.h *mgd*

Add Zone typing to +create:* methods. (Swarm): Return Activity with activateIn:

1999-08-09 objectbase.h *alex*

(EmptyProbeMap): Add +create:forClass: convenience create message to protocol.

1999-08-08 objectbase.h *alex*

(ActivityControl): Move -attachToActivity: to USING phase, doesn't need to be a CREATING phase method.

1999-07-15 objectbase.h *alex*

(CustomProbeMap): Document existing methods. Add new SETTING method (addProbesForClass:withIdentifiers:) which allows post-create time addition of variables and method names via the list-delimiter form.

1999-05-29 objectbase.h *mgd*

Include externvar.h.

1999-05-28 objectbase.h *mgd*

Use `externvar' for external variables.

1999-04-29 objectbase.h *mgd*

(MessageProbe), MessageProbe.[hm] ([MessageProbe -longDynamicCallOn:]): Replaces intDynamicCallOn:

1999-04-22 objectbase.h *mgd*

(MessageProbe): Add setting tag.

1999-04-16 objectbase.h *mgd*

(val_t): Use types_t instead of included union.

1999-04-07 objectbase00.sgml *alex*

Fixed erroneous example code that referred to the global probeLibrary instance rather than the intended CustomProbeMap instance. Thanks to Albert-Jan Brouwer <ajbrouw@casema.net> for reporting this.

1999-04-01 objectbase.h *vjojic*

Protocol Swarm inherits protocols SwarmProcess and CREATABLE.

1999-03-21 objectbase.h *mgd*

Make SwarmObject creatable.

1999-03-20 objectbase.h *mgd*

Add @class DefaultProbeMap.

1999-03-08 objectbase.h *mgd*

Fix return type of getArg: (val_t), and correct documentation for getArg: and getArgName:.

1999-02-26 objectbase.h *mgd*

Add CREATABLE tags to all non-abstract protocols.

1999-02-23 objectbase.h *mgd*

(VarProbe, _VarProbe): Merge.

1998-11-12 objectbase.h *mgd*

(Arguments): Remove (moved to defobj.h).

1998-09-04 objectbase.h *mgd*

(val_t): Add _short and _ushort.

1998-09-03 objectbase.h *mgd*

(val_t): Add _uint, _ulong, and _long.

1998-08-20 objectbase.h *mgd*

Declare setDefaultAppDataPath:.

1998-08-19 objectbase.h *mgd*

Declare setDefaultAppConfigPath:.

1998-08-19 objectbase.h *mgd*

Do it here.

1998-08-07 objectbase.h *mgd*

Add @class SwarmObject.

1998-07-15 objectbase.h *mgd*

Split VarProbe into new features vs. user presentation.

1998-07-14 objectbase.h *mgd*

Minor reformatting of documentation.

1998-07-07 objectbase.h *mgd*

(Probe): Add argument to setProbeClass.

1998-07-06 objectbase.h *alex*

(Arguments): Further clarify use of Arguments with main.m program fragment in Example.

1998-06-24 objectbase.h *alex*

(Arguments): Add protocol. Add documentation description for Arguments, including long example marked-up with //E:.

1998-06-17 Makefile.am *mgd*

Include from refbook/ instead of src/.

1998-06-17 objectbase00.sgml *alex*

Added missing reference to import of activity.h.

1998-06-17 objectbasemeta.sgml *alex*

Removed redundant text from ABSTRACT.

1998-06-17 objectbase.h *mgd*

Document Swarm.

1998-06-15 Makefile.am *mgd*

(MODULE): New variable. Include Makefile.rules from src. Remove everything else.

1998-06-12 objectbase00.sgml, objectbasecont.sgml *mgd*

Update IDs to SWARM.module.SGML.type.

1998-06-06 objectbase.ent *mgd*

Use public identifiers.

1998-06-05 Makefile.am *mgd*

(swarm_ChangeLog): Add.

1998-06-03 objectbase.h *mgd*

Update documentation tags.

1998-05-28 objectbase.h *mgd*

Include defobj.h.

1998-05-27 objectbase.ent.in *mgd*

Remove @srcdir@ in revhistory.

1998-05-23 Makefile.am *mgd*

New file.

1998-05-23 objectbase.ent.in *mgd*

New file.

1998-05-23 objecbase.ent *mgd*

Removed.

1998-05-22 *mgd*

Begin revision log.

1998-05-06 objectbase.h *mgd*

Spacing changes in method declarations throughout. (ProbeConfig, DefaultProbeMap): Add description string. (CompleteProbeMap): Add phase tags. (EmptyProbeMap): Remove -createEnd. (CompleteVarMap): Add //S.

1998-04-22 objectbase.h *mgd*

Remove includes of SwarmObject.h and Swarm.h.

1998-04-15 objectbase.h *mgd*

Add //D: documentation comment tag for module.

1998-03-19 objectbase.h *mgd*

Add const char * slot to val_t union.

1998-03-18 objectbase.h *mgd*

Declare swarm_version.

1998-03-02 objectbase.h *mgd*

(MessageProbe): Declare doubleDynamicCallOn:.

1998-02-26 objectbase.h *mgd*

(val_t): Add _uchar.

1998-02-23 objectbase.h *mgd*

(val_t): Define. Change declarations per MessageProbe.h changes below.

1998-02-04 objectbase.h *mgd*

Change header comment to objectbase.h. Include from swarmobject/ to objectbase/.

1998-01-24 objectbase.h *mgd*

In MesageProbe protocol, declare setArg:ToObjectName:. Constify To: argument of setArg:To:. Constify second arugment to dynamicCallOn:resultStorage:.

1997-12-09 objectbase.h *mgd*

Constify argument to getProbedForVariable and getProbeForMessage (SwarmObject). Constify return of getProbedType (Probe). Constify argument to setProbedVariable (VarProbe). Constify return of getProbedVariable (VarProbe). Constify argument to setFloatFormat (VarProbe). Constify return of probeAsString (VarProbe, both). Constify ToString argument to setData (VarProbe). Constify argument to setProbedMessage (MessageProbe). Constify return of getArgName (MessageProbe). Constify argument to getProbeForVariable and getProbeForMessage (ProbeMap). Constify argument to dropProbeForVariable and dropProbeForMessage (ProbeMap). Constify first argument to getProbeForVariable and getProbeForMessage (ProbeLibrary). Reformatting throughout.

1997-12-08 objectbase.h *gepr*

Moved all swarmobject files from swarmobject directory to objectbase directory. Renamed swarmobject.h to objectbase.h. Changed all instances of swarmobject.h to objectbase.h and all instances of swarmobject/ to objectbase/.

ActivityControl

Name

`ActivityControl` — A class that provides an object interface to an activity.

Description

The ActivityControl class specifies an object that can be attached to an activity (regardless of how or where that activity is created) for the purpose of explicitly controlling the execution of actions on that activity's action plan. There is nothing that available through this class that is not already available through the variables or messages available via the activity itself. However, it packages up the main control messages and makes them available to other objects that may need control over an activity, thereby shielding the activity from directly receiving messages from outside objects and saving the user from having to parse the more complex interface to the activity.

Protocols adopted by ActivityControl

SwarmObject *(see page 211)*

CREATABLE *(see page 44)*

Methods

Phase: Using

- `- (id <ScheduleActivity>)`**`getActivity`**

  Return the controlled activity.

- `- (void)`**`terminate`**

  Recursively removes all subactivities.

- `- (id <Symbol>)`**`getStatus`**

  The getStatus method returns the status of the activity.

- `- (void)`**`updateStateVar`**

  The updateStateVar method updates the ActivityControl instance variables and tests for the continued existence of the activity that is being controlled. This message is sent on each cycle of the schedule for the activity being controlled.

- `- (id <Symbol>)`**`stepUntil:`** `(timeval_t)`*`stopTime`*

  The stepUntil: method sends a stepUntil: message to the activity if conditions are appropriate. This causes all actions on the activity's schedule, including any actions on subactivities' schedules, to be executed until the activity's relative time is equal to stopTime - 1.

- `- (id <Symbol>)`**`stepAction`**

The step method sends a step message to the activity if the conditions are appropriate. It causes the execution of a single action.

- - (id <Symbol>)**nextAction**

  The next method sends a next message to the activity if the conditions are appropriate. It runs an activity forward through as many actions as necessary until it hits a breakFunction, at which point it walks back up the tree of activities and returns Stopped. In most cases, this means that an entire action or action group on the activity under control will be executed, including completion of all subactivities.

- - (id <Symbol>)**stopActivity**

  The stop method sends a stop message to the activity if the conditions are appropriate. This message causes the control to move back up the run-stack and resume at the place in the code where the run was first executed. The next action on the super-activity will begin without finishing the rest of the current activity's actions.

- - (id <Symbol>)**runActivity**

  The run method sends a run message to the activity if the conditions are appropriate. This message causes the activity to continue executing the actions on its schedule until either no other actions are waiting, or until the execution of actions is stopped by a subactivity or stopped by a stop message to the activity. If the activity completes executing all the actions on its schedule, the run method returns Completed.

- - (void)**attachToActivity:** (id <ScheduleActivity>)*anActivity*

  The attachToActivity: method sets an instance variable inside the ActivityControl object that points to the Activity to be controlled. It then creates a Schedule upon which it places a message to itself to update its own variables.

CompleteProbeMap

Name

`CompleteProbeMap` — A subclass of ProbeMap whose initial state contains the VarProbes and MessageProbes of the requested target class but also those of all its subclasses.

Description

Upon creation, this subclass of the ProbeMap will contain all the variables and all the messages of a given class (including the inherited ones).

Protocols adopted by CompleteProbeMap

ProbeMap *(see page 208)*

CREATABLE *(see page 44)*

Methods

None

CompleteVarMap

Name

`CompleteVarMap` — A subclass of ProbeMap, whose initial state contains no MessageProbes.

Description

A subclass of ProbeMap, whose initial state contains no MessageProbes, but does contain all the VarProbes of the requested target class and those of all its superclasses.

Protocols adopted by CompleteVarMap

ProbeMap *(see page 208)*

CREATABLE *(see page 44)*

Methods

None

CustomProbeMap

Name

`CustomProbeMap` — A subclass of ProbeMap, whose initial state is empty unlike the default probeMap initial state which contains all the VarProbes of the requested target class.

Description

This subclass of the ProbeMap is used to create probe maps which are initialised in an emtpy state or with the VarProbes and MessageProbes intended. In other words, the probed class is set, as is the case with the normal ProbeMap class but upon createEnd no VarProbes or MessageProbes will be present within it. This feature is useful when creating a probe map from scratch (e.g. to be used in conjunction with the setProbeMap:For: message of the ProbeLibrary).

Protocols adopted by CustomProbeMap

ProbeMap *(see page 208)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- `+` **`create:`** `(id <Zone>)`*`aZone`* **`forClass:`** `(Class)`*`aClass`* **`withIdentifiers:`** `(const char *)`*`vars`* **`:`** `...`

  Convenience method for creating a CustomProbeMap in which the user specifies the list of variables and methods to be probed this by passing a delimited list of strings of the form: "var1", "var2", ..., ":", "method1", "method2",..., NULL

Phase: Setting

- `-` **`addProbesForClass:`** `(Class)`*`aClass`* **`withIdentifiers:`** `(const char *)`*`vars`* **`:`** *`...`*

  Allows further probes specified in the delimited string list to be added *after* the CustomProbeMap has been created

DefaultProbeMap

Name

`DefaultProbeMap` — A subclass of ProbeMap, whose initial state contains all the VarProbes of the requested target class and also those of all its superclasses.

Description

A subclass of ProbeMap, whose initial state contains all the VarProbes of the requested target class and also those of all its superclasses.

Protocols adopted by DefaultProbeMap

ProbeMap *(see page 208)*

CREATABLE *(see page 44)*

Methods

None

EmptyProbeMap

Name

`EmptyProbeMap` — A CustomProbeMap to be used for building up ProbeMaps from scratch.

Description

A CustomProbeMap to be used for building up ProbeMaps from scratch.

Protocols adopted by EmptyProbeMap

CustomProbeMap *(see page 199)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- `+ **create:** (id <Zone>)`*`aZone`* `**forClass:** (Class)`*`aClass`*

  `Convenience method for creating an EmptyProbeMap`

MessageProbe

Name

`MessageProbe` — A class that allows the user to call a given message on any candidate that is an instance of, or inherits from, a given class.

Description

This is a specialized subclass of the abstract class Probe. It completes the specification of a probe that refers to a message element of an object.

Protocols adopted by MessageProbe

Probe *(see page 203)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- + **create:** (id <Zone>)*aZone* **setProbedSelector:** (SEL)*aSel*

  Convenience factory method for common case.

- - **setProbedMethodName:** (const char *)*methodName*

  The setProbedMessage: method sets the message to be probed given the message name. In dynamically-typed languages like JavaScript selectors don't make sense, since method argument types aren't fixed.

- - **setProbedSelector:** (SEL)*aSel*

  The setProbedSelector: method sets the message to be probed given the selector.

Phase: Setting

- - **setHideResult:** (BOOL)*val*

  The setHideResult: method is used to set the visibility of the result field. When set to 1, the user is indicating that the result field in a graphical representation of the message probe should not be shown.

Phase: Using

- - **objectDynamicCallOn:** *target*

  The objectDynamicCallOn: method generates a dynamic message call on the target object. This method assumes the user knows the return type to be id.

- - (const char *)**stringDynamicCallOn:** *target*

The stringDynamicCallOn: method generates a dynamic message call on the target object. This method assumes the user knows the return type to be const char *.

- - (long)**longDynamicCallOn:** *target*

  The longDynamicCallOn: method generates a dynamic message call on the target object. This method assumes the user knows the return type to be numeric and would like a direct translation into type logn.

- - (double)**doubleDynamicCallOn:** *target*

  The doubleDynamicCallOn: method generates a dynamic message call on the target object. This method assumes the user knows the type to be numeric and would like a direct translation into type double.

- - (val_t)**dynamicCallOn:** *target*

  The dynamicCallOn: method generates a dynamic message call on the target object.

- - **setArg:** (unsigned)*which* **ToUnsigned:** (unsigned)*x*

  The setArg:ToUnsigned: method sets the nth argument of the message used by the probe to an unsigned integer value. The user is responsible for matching the unsigned integer type of this argument with the argument type of the method being probed.

- - **setArg:** (unsigned)*which* **ToString:** (const char *)*what*

  The setArg:ToString: method sets the nth argument of the message. The argument must be provided in string form.

- - (BOOL)**getHideResult**

  The getHideResult method returns 1 if the result field is "hidden".

- - (const char *)**getArgName:** (unsigned)*which*

  The getArgName: method returns a string representation of the argument key with the given index.

- - (val_t)**getArg:** (unsigned)*which*

  The getArg: method returns the argument type for a given index.

- - (unsigned)**getArgCount**

- - (const char *)**getProbedMessage**

  The getProbedMessage method returns the string matching the identifier of the message being probed.

- - (BOOL)**isArgumentId:** (unsigned)*which*

  The isArgumentId: method returns 1 if a given argument of the message is of type object, and returns 0 otherwise.

- - (BOOL)**isResultId**

  The isResultId method returns 1 if the return value of the message is of type object, and returns 0 otherwise.

Probe

Name

`Probe` — An abstract superclass of both VarProbe and MessageProbe.

Description

A Probe is simply an object that contains pointers to an element (instance variable or message description) of another object. The Probe contains instance variables that describe the referent's class and type. It's actually an abstract class that is further subdivided into VarProbe and MessageProbe, which represent the two basic types of elements of any object. The Probes are collected into a ProbeMap and subsequently installed in the ProbeLibrary.

Protocols adopted by Probe

SwarmObject *(see page 211)*

ProbeConfig *(see page 204)*

Methods

Phase: Creating

- - **setProbedObject:** *object*
- - **setProbedClass:** (Class)*class*

  The setProbedClass: method sets the class of the object the probe points at and must be called at create time.

Phase: Setting

- - **unsetSafety**

  The unsetSafety method turns off the option of checking the compatibility of the class of the object before any actions are performed on the object.

- - **setSafety**

  The setSafety method turns on the option of checking the compatibility of the class of the object before any actions are performed on the object.

Phase: Using

- - (const char *)**getProbedType**

  The getProbedType method returns the typing of the probed variable or message. The typing is represented using the string-format provided by the Objective-C runtime system.

- - (Class)**getProbedClass**

  The getProbedClass method returns the class of the object the probe points at as a Class pointer.

- - **clone:** (id <Zone>)*aZone*

  The clone: method returns a clone of the probe. If the initial probe was created by Library Generation or by the default version of Object generation, the probe should be cloned prior to making changes to it to avoid having the changes affect the other potential users of the probe.

ProbeConfig

Name

ProbeConfig — Protocol for configuration of Probes, ProbeMaps, and the ProbeLibrary.

Description

Protocol for configuration of Probes, ProbeMaps, and the ProbeLibrary.

Protocols adopted by ProbeConfig

None

Methods

Phase: Using

- - **getObjectToNotify**
- - **setObjectToNotify:** *anObject*

ProbeLibrary

Name

`ProbeLibrary` — A (singleton) Class, whose instance is used as a container for a global mapping between classnames and their 'default' ProbeMaps. These defaults can be changed by the user, thus allowing him/her to customize the default contents of the ProbeDisplays generated when probing objects.

Description

The normal Swarm simulation will probably only ever contain one instance of this class, namely the probeLibrary object. This object is used for Library Generation of Probes and ProbeMaps: its role is to cache one unique "official" ProbeMap for every Class ever probed during a run of Swarm. These ProbeMaps are generated as they are requested.

Protocols adopted by ProbeLibrary

Create *(see page 46)*

Drop *(see page 54)*

ProbeConfig *(see page 204)*

CREATABLE *(see page 44)*

Methods

Phase: Using

- `- setProbeMap: (id <ProbeMap>)aMap ForObject: anObject`
- `- setProbeMap: (id <ProbeMap>)aMap For: (Class)aClass`

  The setProbeMap:For: method sets the standard probe map as the probe map. The returned Probe will be cached as though it was produced by the library itself.
- `- (id <MessageProbe>)getProbeForMessage: (const char *)aMessage inObject: anObject`
- `- (id <MessageProbe>)getProbeForMessage: (const char *)aMessage inClass: (Class)aClass`

  The getProbeForMessage:inClass: method returns a probe that has been "checked out" from the appropriate Probes in the probe library. Note: The returned probe will be cached so to avoid affecting the results of future requests for the same probes, clone the probe prior to making modifications to the probe.
- `- (id <VarProbe>)getProbeForVariable: (const char *)aVar inObject: anObject`

- - (id <VarProbe>)**getProbeForVariable:** (const char *)*aVar* **inClass:** (Class)*aClass*

  The getProbeForVariable:inClass: method returns a probe that has been "checked out" from the appropriate Probes in the probe library. Note: The returned probe will be cached so to avoid affecting the results of future requests for the same probes, clone the probe prior to making modifications to the probe.

- - (id <ProbeMap>)**getCompleteVarMapForObject:** *anObject*

- - (id <ProbeMap>)**getCompleteVarMapFor:** (Class)*aClass*

  The getCompleteVarMapFor: method returns a ProbeMap containing Probes for all the instance variables of the given Class (including inherited variables) but does not include any MessageProbes.

- - (id <ProbeMap>)**getCompleteProbeMapForObject:** *anObject*

- - (id <ProbeMap>)**getCompleteProbeMapFor:** (Class)*aClass*

  The getCompleteProbeMapFor: method returns a ProbeMap containing Probes for all the instance variables and messages of the given Class (including inherited variables and messages). The current implementation of ProbeLibrary does not cache CompleteProbeMaps.

- - (id <ProbeMap>)**getProbeMapForObject:** *anObject*

- - (id <ProbeMap>)**getProbeMapFor:** (Class)*aClass*

  The getProbeMapFor: method returns a ProbeMap for the aClass class. If a specific ProbeMap has been designed and installed in the ProbeLibrary for that class, then that specific ProbeMap is returned. If a custom ProbeMap was not designed and installed, then a CompleteProbeMap is created and returned.

- - (BOOL)**isProbeMapDefinedForObject:** *anObject*

- - (BOOL)**isProbeMapDefinedFor:** (Class)*aClass*

  The isProbeMapDefinedFor: method returns True if there is a non-nil value in the ProbeLibrary for that class and False otherwise.

- - (unsigned)**getSavedPrecision**

  The getSavedPrecision method gets the current saved precision set in the ProbeLibrary instance.

- - **setSavedPrecision:** (unsigned)*nSigSaved*

  The setSavedPrecision: method sets the number of significant digits saved for floating-point and double floating-point numbers through ObjectSaver. This function sets the global default precision for all floating point numbers, including double floating point numbers. This floating point precision affects all numbers saved via the ObjectSaver class. There is currently no way to override this global default for an individual probe.

- - (unsigned)**getDisplayPrecision**

  The getDisplayPrecision method gets the current display precision set in the ProbeLibrary instance.

- - **setDisplayPrecision:** (unsigned)*nSigDisplay*

```
 The setDisplayPrecision: method sets the number of significant digits for
floating point and double floating point numbers displayed on GUI widgets.
This method is currently only implemented for VarProbes. It has not been
implemented for MessageProbes yet.  The setDisplayPrecision method allows
all probes checked out from the  global ProbeLibrary instance to access
this displayed precision. However,  individual probes can vary from this
global default, by using the  setFloatFormat method on a exisiting probe.
```

ProbeMap

Name

`ProbeMap` — A container class for Probes used to specify the contents of a ProbeDisplay.

Description

A ProbeMap is a Map-type collection of Probes. They are used to gather several Probes, who usually have a common referent, into a single bundle. For example, all the instance variables of a ModelSwarm might be gathered into a single ProbeMap. Each ProbeMap is then installed into the global ProbeLibrary.

Protocols adopted by ProbeMap

SwarmObject *(see page 211)*

ProbeConfig *(see page 204)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- `- **setProbedObject:** *object*`
- `- **setProbedClass:** (Class)*class*`

  The setProbedClass: method sets the class of the object that the set of probes that constitute the probe map points at. This message must be sent before createEnd.

Phase: Using

- `- **clone:** (id <Zone>)*aZone*`

  The clone: method returns a clone of the probe map. If the initial probe map created by Library Generation or by the default version of Object generation, the probe map should be cloned prior to making changes to it to avoid having the changes affect the other potential users of the probe map.

- `- (id <Index>)**begin:** (id <Zone>)*aZone*`

  The begin: method returns an iterator (index) over the ProbeMap. This index is used in the exact same way any Map index is used.

- `- **dropProbeMap:** (id <ProbeMap>)*aProbeMap*`

  The dropProbeMap: method is used to drop a probe from a probe map. It is equivalent to callling dropProbeForVariable for each variable name present in the ProbeMap being dropped, followed by a call to dropProbeForMessage for each message name present in the ProbeMap being dropped.

- - (void)**dropProbeForMessage:** (const char *)*aMessage*

  The dropProbeForMessage: method is used to drop a Probe from the ProbeMap. No class verification takes place since the probe is dropped based on its messageName, not its actual id value.

- - (void)**dropProbeForVariable:** (const char *)*aVariable*

  The dropProbeForVariable: method is used to drop a Probe from the ProbeMap. No class verification takes place since the probe is dropped based on its variableName, not its actual id value.

- - **addProbeMap:** (id <ProbeMap>)*aProbeMap*

  The addProbeMap: method is used to tailor the contents of a ProbeMap by performing "set inclusion" with another ProbeMap. The typing is verified prior to inclusion.

- - **addProbe:** (id <Probe>)*aProbe*

  The addProbe: method adds a probe to the contents of the ProbeMap. The ProbeMap will always make sure that the probedClass of the Probe being added corresponds to its own probedClass.

- - (Class)**getProbedClass**

  The getProbedClass method returns the class of the object that the set of probes that constitute the probe map points at.

- - (unsigned)**getCount**

  The getCount method returns the number of probes in the ProbeMap.

Swarm

Name

`Swarm` — A temporal container.

Description

A Swarm is a community of agents sharing a common timescale as well as common memory pool.

Protocols adopted by Swarm

SwarmProcess *(see page 178)*

CREATABLE *(see page 44)*

Methods

Phase: Using

- `- (id <VarProbe>)`**`getProbeForVariable:`** `(const char *)`*`aVariable`*

  Needed to support probing of Swarms.

- `- (id <ProbeMap>)`**`getCompleteProbeMap`**

  Needed to support probing of Swarms.

- `- (id <ProbeMap>)`**`getProbeMap`**

  Needed to support probing of Swarms.

- `- (id <Activity>)`**`activateIn:`** `(id <Swarm>)`*`swarmContext`*

  Override this to activate any actions you built in buildActions. Note, you must activate yourself first before you can activate actions inside you.

  Example -activateIn: #1

```
[super activateIn: swarmContext];
[fancySchedule activateIn: self];
return [self getSwarmActivity];
```

- `-` **`buildActions`**

  Override this to let your Swarm build its actions.

- `-` **`buildObjects`**

  Override this to let your Swarm create the objects that it contains.

SwarmObject

Name

`SwarmObject` — A superclass of most objects in a Swarm simulation that provides support for probing.

Description

A SwarmObject is an object that is intended to be a member of a Swarm. It's behavior will be perpetuated by messages sent from the schedule of events defined in the context of Swarm object.

The SwarmObject is where the models of all the agents of a simulation will reside. Hence, most of the burden on defining the messages that can be sent to any agent lies with the user. SwarmObject inherits its basic functionality from the Create and Drop object types defined in the defobj library.

Protocols adopted by SwarmObject

Create *(see page 46)*

Drop *(see page 54)*

CREATABLE *(see page 44)*

Methods

Phase: Using

- `- (id <MessageProbe>)`**`getProbeForMessage:`** `(const char *)`*`aMessage`*

  The getProbeForMessage: method returns the MessageProbe indexed in the ProbeMap by the string aMessage.

- `- (id <VarProbe>)`**`getProbeForVariable:`** `(const char *)`*`aVariable`*

  The getProbeForVariable: method returns the VarProbe indexed in the ProbeMap by the string aVariable.

- `- (id <ProbeMap>)`**`getCompleteProbeMap`**

  The getCompleteProbeMap method returns a newly created CompleteProbeMap for an object.

- `- (id <ProbeMap>)`**`getProbeMap`**

  The getProbeMap method returns a pointer to the ProbeMap for the object if there has been one creaded for that object's class. If it hasn't been created, then it creates a default ProbeMap.

VarProbe

Name

`VarProbe` — A class that allows the user to inspect a given variable in any candidate that is an instance of, or inherits from, a given class.

Description

This is a specialized subclass of the abstract class Probe. It completes the specification of a probe that refers to an instance variable element of an object.

Protocols adopted by VarProbe

Probe *(see page 203)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- `- **setProbedVariable:** (const char *)`*`aVariable`*

  The setProbedVariable: sets the variable being probed. The aVariable identifier is simply a character string consisting of the identifier of the variable referent. This method must be called during the create phase.

Phase: Setting

- `- **setFloatFormat:** (const char *)`*`format`*

  The setFloatFormat: method sets the floating-point format of a GUI display widget when given a sprintf-style formatting string.

- `- **setStringReturnType:** `*`returnType`*

  The setStringReturnType: method sets the format that will be used to print the variable. When the probedVariable is of type unsigned char or char, the method probeAsString will, by default, return a string of the format: "'%c' %d". This is meant to reflect the commonplace use of an unsigned char as a small int.

- `- **setNonInteractive**`

  The setNonInteractive method sets a VarProbe to be non-interactive. This ensures that the user will not be able to change the value of a probe, only observe it. Setting the VarProbe to be non-interactive will not interfere with the drag and drop capability of the objects into the VarProbe field.

Phase: Using

- `- (void)**setData:** `*`anObject`* `**ToDouble:** (double)`*`val`*

Sets the probeVariable value using a double. This requires that the value is numeric.

- - (BOOL)**setData:** *anObject* **ToString:** (const char *)*s*

  The setData:ToString: sets the probedVariable using a string which the probe reads and converts appropriately. When setting the value of an unsigned char or a char using this method, the expected format of the string is always "%i" unless CharString was chosen (in which case the format should be "'%c'").

- - (void)**setData:** *anObject* **To:** (void *)*newValue*

  The setData:To: method sets the probedVariable using the pointer to the new value.

- - **iterateAsInteger:** *anObject* **using:** (void (*) (unsigned rank, unsigned *vec, int val))*func*

  Iterates through the elements in an array, calling the argument function with the rank, position vector, and array element cast as an integer.

- - **iterateAsDouble:** *anObject* **using:** (void (*) (unsigned rank, unsigned *vec, double val))*func*

  Iterates through the elements in an array, calling the argument function with the rank, position vector, and array element cast as a double.

- - (unsigned *)**getDims**

  Returns a vector equal to length returned by getRank: with the dimensions of the array (major to minor).

- - (const char *)**getBaseType**

  In the case of arrays, returns the base type.

- - (unsigned)**getRank**

  Returns rank of array, or 0 for scalar objects.

- - (id <String>)**probeAsString:** *anObject*

  The probeAsString: method prints the value of the variable into a new String object.

- - (const char *)**probeAsString:** *anObject* **Buffer:** (char *)*buffer*

  The probeAsString:Buffer: method prints the value of the variable into the buffer. The buffer should be pre-allocated.

- - (const char *)**probeAsString:** *anObject* **Buffer:** (char *)*buf* **withFullPrecision:** (BOOL)*precision*

  The probeAsString:Buffer:withFullPrecision: method prints the value of the variable into the buffer. The buffer should be pre-allocated. This version of probeAsString is used internally by ObjectSaver to use the "saved as" precision form which may differ from the "displayed" precision.

- - (double)**probeAsDouble:** *anObject*

  The probeAsDouble: method returns a pointer to the probed variable as a double.

- - (int)**probeAsInt:** *anObject*

The probeAsInt: method returns a pointer to the probed variable as an integer.

- - **probeObject:** *anObject*

  A field probed with probeAsObject: must be an object.

- - (void *)**probeAsPointer:** *anObject*

  The probeAsPointer: method returns a pointer to the probed variable based on the probeType.

- - (void *)**probeRaw:** *anObject*

  The probeRaw: method returns a pointer to the probed variable.

- - (BOOL)**getInteractiveFlag**

  The getInteractiveFlag method returns the interactivity state of the VarProbe.

- - (const char *)**getProbedVariable**

  The getProbedVariable method returns a string matching the identifier of variable being probed.

Globals

id <Symbol> DefaultString

No description available.

id <Symbol> CharString

No description available.

id <Symbol> IntString

No description available.

General

Name

objectbase — Support for Swarm objects and probing

Description

The objectbase library contains the most basic objects users need to design their agents and swarms. It also serves, at present, as a repository for the probe machinery, which is provided for every SwarmObject.

Globals

id <ProbeLibrary> probeLibrary

The global librarian for ProbeMaps.

const char * swarm_version

The version of Swarm being used.

Random Library

Overview

First, a few thoughts on random number generation. It's hard to do right! The root cause, of course, is that computer algorithms themselves are not truly random. (Hence this library contains pseudo-random generators only.) There are many problems in implementing algorithms correctly and efficiently, and in coming up with good tests for generators and distributions. The history of pseudorandom number generation in simulation work is mostly embarrassing. This library attempts to do a decent job of generating random numbers, as well as documenting how things work and what shortcomings there are. If you want to learn more about random number generation, the bibliography in the *Swarm User Guide (http://www.swarm.org)* has useful notes. Knuth is the main reference in this realm, but too old to describe most of the particular generators used here, many of which are drawn from recent literature.

1. Dependencies

Following are the other header files imported by <random.h>:

```
#import <objectbase.h>
#import <random/generators.h>
#import <random/distributions.h>
#import <random/randomdefs.h>
#import <random/randomvars.h>
```

The *objectbase* library interface is included to provide the basic object support. `randomdefs.h` contains some C preprocessor macros and typedefs used in the library.

This reference guide contains the object definitions for generators and distributions (see the list above) and also encodes the inheritance structure through the "Protocols that this protocol uses" section of each protocol. Just click on a (sub-)protocol name to see what methods it implements. (You may want to review the section on Protocols in the Objective-C book here!)

In the protocols described, any protocol that ultimately inherits from CREATABLE defines an object that you can use in your program. (This is part of the Swarm defobj machinery.) In other words, while 'InternalState' is a normal protocol (a list of method definitions), the name `ACGgen' refers to both a protocol and a class that implements that protocol. Similarly, 'GammaDist' defines both a protocol and a class that implements that protocol.

All generators and distributions ultimately inherit from SwarmObject.

2. Compatibility

- **1.0.2 -> 1.0.3.** *Note:* The new random library does not work in the same way as the old one. This means that some applications that used the random library provided with the 1.0.2 release will be *broken*. However, porting these applications to the new random library will be fairly easy since large efforts were made to adhere to the standard set with the last version and some backwards compatibility hooks were incorporated.

- **1.0, 1.1, 1.2, 1.3, 1.3.1, 1.4.** There were no major compatibility issues in these releases.

3. Usage Guide

3.1. Overview

The *random* library contains two kinds of objects, the *generators* which implement different pseudo-random-number algorithms, and the *distributions* which transform the (uniform) output from the generators into the desired simulated statistical distributions. (The Swarm random library does not implement any *true* random number generators at this time.)

3.2. Usage Guide for Beginners, Advanced Usage Guide and Guide to Generators and Distributions

All these sections have been relocated to the *Swarm User Guide (http://www.swarm.org)*

4. Subclassing Reference

Random library objects do not do anything exotic during the create phase. The competent programmer may subclass these objects in the normal manner.

5. Implementation Notes

This section provides implementation details for the current version of the random library.

5.1. General Implementation Notes

This is release 1.4.1 of the Swarm libraries. It contains version 0.8 of the random-number library and version 0.81 of the random library documentation.

Look at the Random Library *(see page 215)* for the objects defined in this library.

`Fat' vs. `Thin' doubles. Note that distributions which use floating point variates from their generators by default draw `fat' doubles (-getDoubleSample) which use two calls to the basic (32-bit) unsigned int sample method (-getUnsignedSample) in order to fill the 53-bit mantissa of a double. If you don't need this much precision, or want to speed up the distributions, you can make the distributions use `thin' samples (-getThinDoubleSample) instead. See the note at the top of `random/distributions.h` for how to change this behavior. If you do, be sure to remake Swarm (**`make`**, **`make install`**).

5.2. Implementation notes for Generators

Version 0.8: Changes since version 0.75.

1. The code was rearranged to conform to create-phase protocol (CREATING-SETTING-USING) ordering.
2. Some for-loop indices were changed to unsigned integers to eliminate compiler warnings.

3. A few objects (C2LCGXgen, C4LCGXgen, SWBgen, TGFSRgen) were given their own -drop methods to drop internally allocated arrays properly.
4. A new -reset method was added to all generators. This method resets the state of the generator to what it was at creation, or at the point when -setStateFromSeed(s) was last used. Counters are also reset.

Version 0.75: Changes since version 0.7.

1. The method '-getDoubleSample' was redefined to use only double variables in its implementation (instead of long doubles).
2. The macros used for starting seed generation were changed to avoid a situation where many new generators would be created the same starting seed (if '--varyseed' was not specified.) See the Generator Usage Guide and the Reference Guide for details.

Version 0.7: Improvements over version 0.6.

- A host of new generators, located on the web or in the literature, have been added since the last version of Random. There is now a total of 36 different generators defined! Some of these have immense periods, some are very fast, and some have much better statistical properties than the old generators.
- A new *type* of generator, the `split' generator, has been introduced in the form of L'Ecuyer's C2LCGXgen and C4LCGXgen generators.
- A `split' generator is a long-period generator for which we are able to split the period into arbitrary sub-periods, which we can access quickly. We then configure the generator as having a number (A) of `virtual generators', each of which can address a number (2^v) of sub-segments of length 2^w. These parameters (A,v,w) are user selectable when the generator is created. (As an example, for C4LCGXgen the default values are A=128, v=31, w=41.) The advantage is that the subsegments act as statistically independent streams of random numbers.
- In addition to the -getUnsignedSample method, generators now also supply floating point output in the range [0.0,1.0), in the form of these methods:

```
-(float)       getFloatSample;         // using 1 unsigned value
-(double)      getThinDoubleSample;          // using 1 unsigned value
-(double)      getDoubleSample;              // using 2 unsigned values
-(long double) getLongDoubleSample;          // using 2 unsigned values
```

  Note that the last method is not portable across architectures, since the length of a long double varies between machines.
- Generators may now be started with a single seed, *or* with a vector of seeds whose length is generator dependent. (PMMLCG requires 1 integer for a seed, while MT19937 needs 624 of them.)
- Generators now remember what seed values they were started with. They also count how many variates they have delivered (i.e., how many calls to -getUnsignedSample they have serviced.)
- There are a few arbitrary seed values, DEFAULTSEED, DEFAULTSEED1, DEFAULTSEED2, DEFAULTSEED3, DEFAULTSEED4 defined. There is also the value FIRSTSEED, which returns the value that the default generator `randomGenerator' was started with.
- The macro NEXTSEED will generate a deterministic sequence of seed values, using and inline LCG and starting with FIRSTSEED. There is the macro RANDOMSEED, which will be different every time it is invoked because it depends on program time. And there is value STARTSEED, which will

by default equal NEXTSEED, but will instead be equal to RANDOMSEED if you start your program with the `--varyseed` or `-s` command line parameter.

- The generators have gained a new creation method, '+createWithDefaults: aZone', which creates the generator and initializes it with STARTSEED. Split generators get default values for A,v,w.

Version 0.7: Changes since version 0.6.

- The generator classes have changed names to where they all end in '-gen'. A simple search-and-replace in your code will get you up and running again.(Or perhaps you'll want to try one of the new generators?)
- A bug in SWBgen was corrected. Code for ACG and SCG was also changed.
- The `-verifySelf` method is gone. Instead see the test program located in `/random/testR0`. (Available in a separate tarball at the SFI ftp site.)
- The `getState:' method has been named `putStateInto: (void *) buffer', and the `setState:' method is now `setStateFrom: (void *) buffer'. A quick search-and-replace fixes things in your code.
- Note: these methods have also changed somewhat, as has the size of the data being saved. As a result, v. 0.7 generators will refuse to `setStateFrom' data saved by v. 0.6 objects.
- There should be fewer changes like this in the next release.

Testing Generators. Since v. 0.6 we have done some rudimentary statistical testing of the implemented generators, using Marsaglia's Diehard tests and the ENT tests. The results of these tests are summarized in Generator quality table (now found in the *Swarm User Guide (http://www.swarm.org)*), where test results as well as period length, state size and execution times are listed. You can use these data to select a generator that suits your simulation. Some brief comments:

a. the tests show that old generators SCG and LCG are of poor quality and should be avoided.

b. the lagged-Fibonacci based generators (ACG, SWB, PSWB) all fail Diehard's `Birthday spacings test', for reasons having to do with their lattice structure. These generators are only conditionally recommended.

c. The rest of the 32-bit generators (i.e. generators that fill all 32 bits of an unsigned int) pass all tests, and are recommended at this time. (Note that while a test may show that a generator is bad, passing a number of tests does not prove that a generator is good!)

d. The 31-bit generators all fail the same set of tests. Some of these cannot be passed by a generator whose output has a `stuck' bit. Until I clear up with Prof. Marsaglia how to interpret these results, I believe all the 31-bit generators are in the `recommended' category.

e. However, a cautionary note: while the PMMLCG generators pass the tests, they have a very short period (less than 2^31) and should only be used for `toy' simulations. You don't want your generator(s) to `go around' and start repeating themselves !

f. For what it's worth, Professor L'Ecuyer recommends his own C4LCGX and C2MRG3 generators as well as Matsumoto's TT800 (the monster MT19937 hadn't been released yet), and Prof. Marsaglia recommends his own Multiply-With-Carry generators (MWCA, MWCB, C3MWC, RWC2, RWC8="Mother").

5.3. Implementation notes for Distributions

Version 0.8: Changes over version 0.75. No functional changes were made. Code was rearranged to conform to create-phase protocol (CREATING-SETTING-USING) ordering.

Version 0.75: Changes over version 0.7. No functional changes were made.

Version 0.7: Improvements over version 0.6.

- One new distribution class, BernoulliDist, has been added. It returns binary values (yes/true/1) with a given probability (while the old RandomBitDist has a fixed 50% probability, a fair coin toss.)
- Distributions now have a new create method, '+createWithDefaults: aZone'. This method creates the distribution object, and also a new generator object for its exclusive use. Each distribution class has a different default generator class assigned. These generators are initialized with STARTSEED, which by default equals the fixed value DEFAULTSEED, but will be equal to the varying RANDOMSEED if you start your program with the command line parameter `--varyseed` or `-s`.
- All distributions have code to interact with the new `split' generators.
- UniformIntegerDist and UniformUnsignedDist now allow you to set parameter minValue equal to maxValue. In this case that value is returned every time.
- UniformDoubleDist also allows this, even if the set [x,x) is mathematically suspect ...
- NormalDist and LogNormalDist now allow you to specify zero Variance, in which case the values returned are the Mean and exp(Mean) respectively.

Version 0.7: Changes since version 0.6.

- The distribution classes have changed names to where they all end in `Dist'. A simple search-and-replace in your code will get you back up and running.
- The strong distinction between `frozen' and `un-frozen' distribution objects in v. 0.6 has been softened considerably. You may now set and reset the default parameters as often as you wish, and you may make calls for variates with given parameters even if different default parameters have been set.
- The generation of uniform(0,1) floating point values has been moved from the distribution objects into the generator objects. Thus, if all you need is a uniform(0,1) double, you have no need of a distribution but can get what you desire from a generator.
- Note that the generators fill the mantissa of a double from two 32-bit unsigned values in a different manner from v. 0.6 distributions, so output will be a bit different in the new version.
- A bug in LogNormalDist has been fixed.
- The `getState:' method has been named `putStateInto: (void *) buffer', and the `setState:' method is now `setStateFrom: (void *) buffer. A quick search-and-replace fixes things in your code.
- But note: these methods have also changed somewhat, as has the size of the data being saved. As a result, v. 0.7 distributions will refuse to `setStateFrom' data saved by v. 0.6 objects.

Utility Objects Provided. The following objects have been defined in `random/random.m`. They may be accessed and used from anywhere in your program.

```
id <SimpleRandomGenerator>   randomGenerator;
id <UniformIntegerDist>      uniformIntRand;
id <UniformUnsignedDist>     uniformUnsRand;
```

```
id <UniformDoubleDist>       uniformDblRand;
```

The 3 distribution objects all draw their random numbers from the MT19937 generator, which has a period of 2^19937 (10^6001) and is quite fast.

5.4. Programming yet to do

Like many Open Source projects, this random-number library is a work in process. Further developments still on the to-do list are detailed below.

The following changes and additions are contemplated for the next release of Random for Swarm (though I don't promise they'll all make it in -- nor when that next release will be):

a. ADD a few more generators. It's good to have many different types of generators, so you can test your model's results with different generators and ensure that the results aren't artifacts of the generator used. And people seem to insist on inventing new, better ones with longer periods!

b. ADD more distributions. NOTE: if you have any strong opinions about what distributions or generators need to be added, please e-mail me!

c. TEST the generators, using Marsaglia's Diehard battery, or L'Ecuyer's tests when/if those become available. This will allow us (a) to make a choice between the implemented generators on the basis of their statistical quality, (b) to decide what old and bad generators to remove, and (c) to detect any bugs in my implementation.

d. TEST the distributions, to make sure they actually put out numbers according to the probability distribution and parameters used.

e. ELIMINATE 'bad' old generators on the basis of statistical tests

f. RETAIN PMMLCG as the only short-period generator, for convenience

g. ADD an 'empirical' distribution, whose f is defined by a set of user- supplied data

h. IMPLEMENT a version of getState/setState that is portable across machine architectures, so that simulations may be moved to or duplicated on other machines. (The problem: integers and doubles are stored in different byte orders on different systems.)

i. IMPLEMENT a proper -drop method for generators that allocate their state vectors dynamically, freeing the state vector memory to avoid `memory leakage'

j. REVIEW all objects for ways to make the crucial methods run faster

k. ADD code to make all objects meter their own usage and send the author monthly e-mails in a stealthy manner, so he can monitor usage and perhaps start collecting a usage fee for his efforts ... ;-) (Suggested by Rick Riolo. Thanks, Rick!)

Can you think of anything else? Drop me a note! -- Sven Thommesen
<sthomme@humsci.auburn.edu>

Documentation and Implementation Status

This is version 0.8 of Random. It was donated by Sven Thommesen. Version 0.6 was a reimplementation of most of Nelson Minar's original random with many changes and a new interface. Versions 0.7 and 0.75 added many more generators and distributions and changed the interface somewhat. This version cleaned up the protocol interface definitions and fixed a few small bugs. The documentation was also improved a bit.

We are reasonably sure that the generators and distributions included here have been correctly implemented. The generators have been subjected to a battery of statistical tests, and the results are described in the documentation. The distributions have not been subjected to statistical tests yet. As with any pseudo-random number generation library, the results obtained should be examined closely. A set of test programs which exercises the objects is available on the Swarm web site, and the statistical tests are also available on the web.

Revision History

2000-05-18 random.h *mgd*

([InternalState -describe:, -getName]): Remove.

2000-03-28 *mgd*

Swarmdocs 2.1.1 frozen.

2000-02-29 *mgd*

Swarmdocs 2.1 frozen.

2000-02-15 random00.sgml *alex*

Removed all LINKs to `Random Appendix', refer reader to Swarm User Guide. Add `Implementation Notes' back into this PARTINTRO from old Appendix, as the Implementation details should always be in-place with the Reference Guide.

1999-02-10 random00.sgml *sthomme*

minor textual editing.

1999-02-10 random-app.sgml *alex*

Brought all idrefs into line with naming convention.

1999-02-10 random00.sgml *alex*

Removed `Implementation Design Notes' - no longer relevant. Make link to User-Guide.For-Beginners section. Brought all idrefs into line with naming convention.

1999-02-09 Makefile.am *alex*

(SGML): Add, set to random-app.sgml. (EXTRA_DIST): Add SGML onto the dependencies generated by `Makefile.rules'.

1999-02-09 random00.sgml *alex*

Moved most content into random-app.sgml APPENDIX.

1999-02-09 random.ent *alex*

Made all ENTITY references to the SGML in the `extra' subdirectory.

1999-02-09 random-app.sgml *alex*

Add file. Content container for `Random' APPENDIX.

1999-02-03 random.ent *sthomme*

Replaced references to README.Generators.v075 and README.Distributions.v07 with 15 new references to file entities in extra SUBDIR.

1999-02-03 $ *sthomme*

(SWARMDOCS)/catalog.in: replaced the two old references with the 15 new ones.

1999-02-03 random00.sgml *sthomme*

major textual revision. Expanded outline which imports 15 textual items.

1999-02-03 randommeta.sgml *sthomme*

revised the text. Deleted commented-out section (old file index).

1999-01-26 random00.sgml *alex*

Make all references to $(SWARMHOME)/src/random/docs be to $(SWARMDOCS)/refbook/random/extra.

1999-01-26 random.ent *alex*

Add references to new text file entitites README.Generators.v075 and README.Distributions.v07.

1999-01-26 random00.sgml *alex*

Use new entities in place of text in NOTE markup.

1999-01-26 Makefile.am *alex*

(SUBDIRS): Add `extra' subdirectory.

1999-01-26 random00.sgml *alex*

Change outdated references of the command line `-varySeed' to `--varyseed'.

1999-01-07 random00.sgml *alex*

({Generator,Distribution} Usage Notes): Made SIDEBAR, NOTE markup to overcome the one page limitation of SIDEBARs in print backend.

1998-10-28 random.h *mgd*

Include objectbase.h instead of defobj.h. Include random{vars,defs}.h instead of Random{Vars,Defs}.h.

1998-06-17 Makefile.am *mgd*

Include from refbook/ instead of src/.

1998-06-15 Makefile.am *mgd*

(MODULE): New variable. Include Makefile.rules from src. Remove everything else.

1998-06-12 random00.sgml, randomcont.sgml *mgd*

Update IDs to SWARM.module.SGML.type.

1998-06-06 random.ent *mgd*

Use public identifiers.

1998-06-05 Makefile.am *mgd*

(swarm_ChangeLog): Add.

1998-06-03 random.h *mgd*

Add summary and description tags for module.

1998-05-23 Makefile.am *mgd*

New file.

1998-05-23 random.ent.in *mgd*

New file.

1998-05-23 random.ent *mgd*

Removed.

1998-05-22 *mgd*

Begin revision log.

1997-12-08 random.h *mgd*

(InternalState): Likewise.

ACGgen

Name

`ACGgen` — Additive Congruential Generator

Description

ACG is in the Lagged Fibonacci class of generators. These generators use a basic algorithm of the form X_n = f(X_(n-r),X_(n-s)) mod m; r>s The function f is typically xor, addition, subtraction, multiplication or subtraction with carry. It uses simpler math than a basic LCG, but keeps a larger state.

NOT recommended for serious use; these are included for historical reasons (compatibility with earlier releases).

Protocols adopted by ACGgen

SimpleRandomGenerator *(see page 261)*

CREATABLE *(see page 44)*

Methods

None

BasicRandomGenerator

Name

`BasicRandomGenerator` — The common functionality of simple and split generators.

Description

This protocol covers methods common to simple and split generators.

Protocols adopted by BasicRandomGenerator

SwarmObject *(see page 211)*

InternalState *(see page 238)*

CommonGenerator *(see page 234)*

Methods

None

BernoulliDist

Name

`BernoulliDist` — Bernoulli Distribution

Description

A distribution returning YES with a given probability.

Protocols adopted by BernoulliDist

BooleanDistribution *(see page 228)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- `+` **`create:`** `(id <Zone>)`*`aZone`* **`setGenerator:`** `(id <SplitRandomGenerator>)`*`splitGenerator`* **`setVirtualGenerator:`** `(unsigned)`*`vGen`* **`setProbability:`** `(double)`*`p`*

  Use this create message if the generator to be attached is a Split one:

- `+` **`create:`** `(id <Zone>)`*`aZone`* **`setGenerator:`** `(id <SimpleRandomGenerator>)`*`simpleGenerator`* **`setProbability:`** `(double)`*`p`*

  Use this create message if the generator to be attached is a Simple one:

Phase: Setting

- `-` **`setProbability:`** `(double)`*`p`*

  The setProbability: method sets the probability of returning YES.

Phase: Using

- `-` `(BOOL)`**`getSampleWithProbability:`** `(double)`*`p`*

  The getSampleWithProbability: returns a sample YES or NO value.

- `-` `(double)`**`getProbability`**

  The getProbability method returns the probability of returning YES.

BinomialDist

Name

`BinomialDist` — Binomial distribution

Description

The binomial distribution gives the discrete probability of obtaining exactly n successes out of N Bernoulli trials

Protocols adopted by BinomialDist

UnsignedDistribution *(see page 271)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- + **create:** (id <Zone>)*aZone* **setGenerator:** (id <SimpleRandomGenerator>)*generator*

  Use this create message if the generator to be attached is a Simple one:

- + **create:** (id <Zone>)*aZone* **setGenerator:** (id <SplitRandomGenerator>)*generator* **setVirtualGenerator:** (unsigned)*vGen*

  Use this create message if the generator to be attached is a Split one:

- + **create:** (id <Zone>)*aZone* **setGenerator:** (id <SplitRandomGenerator>)*generator* **setVirtualGenerator:** (unsigned)*vGen* **setNumTrials:** (unsigned)*aNumTrials* **setProbability:** (double)*aProbability*

  Use this create message if the generator to be attached is a Split one and both the number of trials and the probability are to be set at create time:

- + **create:** (id <Zone>)*aZone* **setGenerator:** (id <SimpleRandomGenerator>)*generator* **setNumTrials:** (unsigned)*aNumTrials* **setProbability:** (double)*aProbability*

  Use this create message if the generator to be attached is a Simple one: and both the number of trials and the probability are to be set at create time:

Phase: Setting

- - **setNumTrials:** (unsigned)*aNumTrials*

  The setNumTrials only sets the numTrials parameter; the probability parameter is left unchanged from its previous or initialized value

- - **setNumTrials:** (unsigned)*aNumTrials* **setProbability:** (double)*aProbability*

The setNumTrials:setProbability sets both the number of trials rate and the probability parameters.

Phase: Using

- - (unsigned)**getUnsignedSampleWithNumTrials:** (unsigned)*aNumTrials* **withProbability:** (double)*aProbability*

  The getUnsignedSampleWithOccurRate:andInterval return a sample value for the specified number of trials and probability. Does not change the the distribution's parameter values.

- - (unsigned)**getUnsignedSampleWithProbability:** (double)*aProbability*

  The getIntegerSampleWithInterval returns a sample value using the distribution's current number of trials and new probability value. Causes an error if the number of trials has not been previously set.

- - (unsigned)**getUnsignedSample**

  The getIntegerSample returns a sample value using the distribution's current number of trials and probability parameters; causes an error if these parameters have not been previously set.

- - (double)**getProbability**

  The getProbability returns probability parameter.

- - (unsigned)**getNumTrials**

  The getNumTrials returns number of trials parameter.

BooleanDistribution

Name

BooleanDistribution — Boolean Distribution

Description

A probability distribution that returns YES/NO sample values.

Protocols adopted by BooleanDistribution

ProbabilityDistribution *(see page 256)*

Methods

Phase: Using

- - (int)**getIntegerSample**
- - (BOOL)**getBooleanSample**

  The getBooleanSample method returns a YES or NO sample value.

C2LCGXgen

Name

C2LCGXgen — A short component based generator with splitting facilities. Recommended. This combined random generator uses 2 (PMM)LGC generators.

Description

This portable generator is based on a backbone generator which is a combination of 2 (PMM)LCG generators. It has a period length of almost 2^61 (2.3e18). The backbone generator's period can be split up into a number of 'virtual generators' (A), each of which can be set to access a number of 'segments' (V) of length W, subject to the constraint that A * V * W <= 2^60.

Protocols adopted by C2LCGXgen

SplitRandomGenerator *(see page 264)*

CREATABLE *(see page 44)*

Methods

None

C2MRG3gen

Name

C2MRG3gen — Combined Multiple Recursive Generator. A combination of 2 multiple recursive LCG generators.

Description

Combinations of like generators are shown to have better statistical properties than single generators. The components of this generator each has two nonzero multipliers (and one that's zero). They use different moduli (2^31-1, 2145483479.)

Protocols adopted by C2MRG3gen

SimpleRandomGenerator *(see page 261)*

CREATABLE *(see page 44)*

Methods

None

C2TAUS1gen

Name

`C2TAUS1gen` — Combined Tausworthe generator 1

Description

Component 1 parameters: P = 31, S = 12, Q = 13 Component 2 parameters: P = 29, S = 17, Q = 2 With these parameters, this generator has a single full cycle of length ~ 2^60.

Protocols adopted by C2TAUS1gen

C2TAUSgen *(see page 231)*

CREATABLE *(see page 44)*

Methods

None

C2TAUS2gen

Name

`C2TAUS2gen` — Combined Tausworthe generator 2

Description

Component 1 parameters: P = 31, S = 21, Q = 3 Component 2 parameters: P = 29, S = 17, Q = 2 With these parameters, this generator has a single full cycle of length ~ 2^60.

Protocols adopted by C2TAUS2gen

C2TAUSgen *(see page 231)*

CREATABLE *(see page 44)*

Methods

None

C2TAUS3gen

Name

`C2TAUS3gen` — Combined Tausworthe generator 3

Description

Component 1 parameters: P = 31, S = 13, Q = 13 Component 2 parameters: P = 29, S = 20, Q = 2 With these parameters, this generator has a single full cycle of length ~ 2^60.

Protocols adopted by C2TAUS3gen

C2TAUSgen *(see page 231)*

CREATABLE *(see page 44)*

Methods

None

C2TAUSgen

Name

`C2TAUSgen` — Combined Tausworthe generator

Description

This generator is based on 2 component generators of periods 2^31-1 and 2^29-1.

Protocols adopted by C2TAUSgen

SimpleRandomGenerator *(see page 261)*

Methods

None

C3MWCgen

Name

`C3MWCgen` — Combined Multiply With Carry generator

Description

This generator is a combination of 3 MWC generators, each of which is a combination of 2 16-bit Multiply-With-Carry generators.

Protocols adopted by C3MWCgen

SimpleRandomGenerator *(see page 261)*

CREATABLE *(see page 44)*

Methods

None

C4LCGXgen

Name

`C4LCGXgen` — Combined random generator using 4 (PMM)LGC generators.

Description

This portable generator is based on a backbone generator which is a combination of 4 (PMM)LCG generators. It has a period length of (m1-1)(m2-1)(m3-1)(m4-1) / 8, or almost 2^121 (2.6e36).

Protocols adopted by C4LCGXgen

SplitRandomGenerator *(see page 264)*

CREATABLE *(see page 44)*

Methods

None

CommonGenerator

Name

CommonGenerator — Internal

Protocols adopted by CommonGenerator

None

Methods

Phase: Creating

- + **createWithDefaults:** (id <Zone>)*aZone*

Phase: Setting

- - **setAntithetic:** (BOOL)*antiT*

  The setAntithetic method turns on or off antithetic output (default=off). Antithetic output is (unsignedMax - u) or (1.0 - d).

- - **setStateFromSeeds:** (unsigned *)*seeds*

  The setStateFromSeeds method initializes the seed dependent part of the state from a vector of seed values.

- - **setStateFromSeed:** (unsigned)*seed*

  The setStateFromSeed method initializes the seed dependent part of the state from a single seed value.

Phase: Using

- - (unsigned)**getUnsignedMax**

  The getUnsignedMax method returns the highest value that will ever be returned by -getUnsignedSample (the lowest is 0).

- - **reset**

  The -reset method sets the generator back to the state it had at start or at the last use of -setStateFromSeed(s). CurrentCount is zeroed.

- - (unsigned)**lengthOfSeedVector**

  The lengthOfSeedVector method returns the number of seeds required if you wish to set the state directly.

- - (unsigned *)**getInitialSeeds**

  The getInitialSeeds method returns a vector of the generator's starting seed values.

- - (unsigned)**getInitialSeed**

  The getInitialSeed method returns the value of the generator's starting seed.

- `- (unsigned *)`**`getMaxSeedValues`**

  The getMaxSeedValues method returns a vector of upper limits on the seed values that can be supplied.

- `- (unsigned)`**`getMaxSeedValue`**

  The getMaxSeedValue method returns the upper limit on the seed value that can be supplied.

- `- (BOOL)`**`getAntithetic`**

  The getAntithetic method returns the current values of the parameter.

DoubleDistribution

Name

`DoubleDistribution` — Double Distribution

Description

A probability distribution that returns an approximation of continuous values as represented by double-precision floating point values.

Protocols adopted by DoubleDistribution

ProbabilityDistribution *(see page 256)*

Methods

Phase: Using

- `- (double)`**`getDoubleSample`**

  The getDoubleSample method returns a double-precision floating point value.

ExponentialDist

Name

`ExponentialDist` — Exponential distribuiton

Description

A well-known continuous probability distribution returning doubles.

Protocols adopted by ExponentialDist

DoubleDistribution *(see page 234)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- `+` **`create:`** `(id <Zone>)`*`aZone`* **`setGenerator:`** `(id <SplitRandomGenerator>)`*`splitGenerator`* **`setVirtualGenerator:`** `(unsigned)`*`vGen`* **`setMean:`** `(double)`*`mean`*

  Use this create message if the generator to be attached is a Split one:

- `+` **`create:`** `(id <Zone>)`*`aZone`* **`setGenerator:`** `(id <SimpleRandomGenerator>)`*`simpleGenerator`* **`setMean:`** `(double)`*`mean`*

  Use this create message if the generator to be attached is a Simple one:

Phase: Setting

- `-` **`setMean:`** `(double)`*`mean`*

  The setMean: method sets the mean of the distribution.

Phase: Using

- `-` `(double)`**`getSampleWithMean:`** `(double)`*`mean`*

  The getSampleWithMean: method returns a sample value from a distribution with the specified mean.

- `-` `(double)`**`getMean`**

  The getMean method returns the mean of the distribution.

GammaDist

Name

GammaDist — Gamma distribution

Description

A well-known continuous probability distribution returning doubles

Protocols adopted by GammaDist

DoubleDistribution *(see page 234)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- + **create:** (id <Zone>)*aZone* **setGenerator:** (id <SplitRandomGenerator>)*splitGenerator* **setVirtualGenerator:** (unsigned)*vGen* **setAlpha:** (double)*alpha* **setBeta:** (double)*beta*

  Use this create message if the generator to be attached is a Split one:

- + **create:** (id <Zone>)*aZone* **setGenerator:** (id <SimpleRandomGenerator>)*simpleGenerator* **setAlpha:** (double)*alpha* **setBeta:** (double)*beta*

  Use this create message if the generator to be attached is a Simple one:

Phase: Setting

- - **setAlpha:** (double)*alpha* **setBeta:** (double)*beta*

  The setAlpha:setBeta: method sets the alpha and beta values for the gamma distribution.

Phase: Using

- - (double)**getSampleWithAlpha:** (double)*alpha* **withBeta:** (double)*beta*

  The getSampleWithAlpha:withBeta: method returns a sample value from a Gamma distribution with the specified alpha and beta values.

- - (double)**getBeta**

  The getBeta method returns the beta value.

- - (double)**getAlpha**

  The getAlpha method returns the alpha value.

IntegerDistribution

Name

`IntegerDistribution` — Integer Distribution

Description

A probability distribution that returns integer sample values.

Protocols adopted by IntegerDistribution

ProbabilityDistribution *(see page 256)*

Methods

Phase: Using

- `- (int)`**`getIntegerSample`**

  `The getIntegerSample method returns an integer sample value.`

InternalState

Name

`InternalState` — Archiving routines for internal generator and distribution state.

Description

Methods to save the internal state of an object (generator, distribution) to a memory buffer allocated by the calling program, and to set the state of an object from previously saved state data, provided in a memory buffer.

NOTE: the putStateInto/setStateFrom methods are NOT portable across architectures, since they store integers and doubles using different byte orders. A portable storage method may be provided in the next release.

Protocols adopted by InternalState

None

Methods

Phase: Using

- `- (unsigned)`**`getMagic`**
- `- (void)`**`setStateFrom:`** `(void *)`*`buffer`*
- `- (void)`**`putStateInto:`** `(void *)`*`buffer`*
- `- (unsigned)`**`getStateSize`**

  `Specifies the minimum buffer size needed.`

LCG1gen

Name

`LCG1gen` — Linear Congruential Generator 1

Description

With the parameters: a = 1,664,525 and c = 1,013,904,223 this generator has a single full cycle of length m.

Protocols adopted by LCG1gen

LCGgen *(see page 240)*

CREATABLE *(see page 44)*

Methods

None

LCG2gen

Name

`LCG2gen` — Linear Congruential Generator 2

Description

With the parameters: a = 69,069 and c = 1,013,904,223 this generator has a single full cycle of length m.

Protocols adopted by LCG2gen

LCGgen *(see page 240)*

CREATABLE *(see page 44)*

Methods

None

LCG3gen

Name

`LCG3gen` — Linear Congruential Generator 3

Description

With the parameters: a = 1,664,525 and c = 152,193,325 this generator has a single full cycle of length m.

Protocols adopted by LCG3gen

LCGgen *(see page 240)*

CREATABLE *(see page 44)*

Methods

None

LCGgen

Name

`LCGgen` — Linear Congruential Generator

Description

This classic generator relies on controlled overflow at 32 bits. This requires that unsigned be a 32bit value that follows ANSI C rules. Knuth claims that the adder c does not matter much, as long as it has no factors in common with the modulus 2^32.

NOT recommended for serious use; these are included for historical reasons (compatibility with earlier releases).

Protocols adopted by LCGgen

SimpleRandomGenerator *(see page 261)*

Methods

None

LogNormalDist

Name

`LogNormalDist` — Log-Normal distribution

Description

A well-known continuous probability distribution returning doubles.

Protocols adopted by LogNormalDist

Normal *(see page 245)*

CREATABLE *(see page 44)*

Methods

None

MRG5gen

Name

`MRG5gen` — Multiple Recursive [LCG] Generator 5

Description

This generator has a single full cycle of length (2^31-1)^5 - 1, i.e. 2^154 < cycle < 2^155.

Protocols adopted by MRG5gen

MRGgen *(see page 243)*

CREATABLE *(see page 44)*

Methods

None

MRG6gen

Name

`MRG6gen` — Multiple Recursive [LCG] Generator 6

Description

This generator has a single full cycle of length (2^31-1)^6 - 1, i.e. 2^185 < cycle < 2^186.

Protocols adopted by MRG6gen

MRGgen *(see page 243)*

CREATABLE *(see page 44)*

Methods

None

MRG7gen

Name

`MRG7gen` — Multiple Recursive [LCG] Generator 7

Description

This generator has a single full cycle of length (2^31-1)^7 - 1, i.e. 2^216 < cycle < 2^217.

Protocols adopted by MRG7gen

MRGgen *(see page 243)*

CREATABLE *(see page 44)*

Methods

None

MRGgen

Name

`MRGgen` — Multiple Recursive [LCG] Generator

Description

These generators require k multipliers and k past values to be kept. In their paper, the authors investigate MRG's of order k from 1 to 7. They provide several sets of parameters which they recommend out of a large number that were tested. Generally, the quality of the generators increases with k.

Protocols adopted by MRGgen

SimpleRandomGenerator *(see page 261)*

Methods

None

MT19937gen

Name

`MT19937gen` — 'Mersenne Twister' Twisted GFSR generator

Description

This generator has a single cycle of length 2^19937-1.

Protocols adopted by MT19937gen

SimpleRandomGenerator *(see page 261)*

CREATABLE *(see page 44)*

Methods

None

MWCAgen

Name

`MWCAgen` — Multiply-With-Carry generator

Description

This generator is claimed to be strictly periodic, with a period > 2^59. (There's possibly two such cycles.)

Protocols adopted by MWCAgen

SimpleRandomGenerator *(see page 261)*

CREATABLE *(see page 44)*

Methods

None

MWCBgen

Name

`MWCBgen` — Multiply-With-Carry generator

Description

This generator implements an alternate manner of conjoining the two components (differs from MWCA). This generator is claimed to be strictly periodic, with a period > 2^59. (There's possibly two such cycles.)

Protocols adopted by MWCBgen

SimpleRandomGenerator *(see page 261)*

CREATABLE *(see page 44)*

Methods

None

Normal

Name

Normal — Internal

Protocols adopted by Normal

DoubleDistribution *(see page 234)*

Methods

Phase: Creating

- + **create:** (id <Zone>)*aZone* **setGenerator:** (id <SplitRandomGenerator>)*splitGenerator* **setVirtualGenerator:** (unsigned)*vGen* **setMean:** (double)*mean* **setStdDev:** (double)*sdev*

  Use this create message if the generator to be attached is a Split one and you wish to specify the standard deviation:

- + **create:** (id <Zone>)*aZone* **setGenerator:** (id <SplitRandomGenerator>)*splitGenerator* **setVirtualGenerator:** (unsigned)*vGen* **setMean:** (double)*mean* **setVariance:** (double)*variance*

  Use this create message if the generator to be attached is a Split one and you wish to specify the variance:

- + **create:** (id <Zone>)*aZone* **setGenerator:** (id <SimpleRandomGenerator>)*simpleGenerator* **setMean:** (double)*mean* **setStdDev:** (double)*sdev*

  Use this create message if the generator to be attached is a Simple one and you wish to specify the standard deviation:

- + **create:** (id <Zone>)*aZone* **setGenerator:** (id <SimpleRandomGenerator>)*simpleGenerator* **setMean:** (double)*mean* **setVariance:** (double)*variance*

  Use this create message if the generator to be attached is a Simple one and you wish to specify the variance:

Phase: Setting

- - **setMean:** (double)*mean* **setStdDev:** (double)*sdev*

  The setMean:setStdDev: method sets the mean and the standard deviation of the distribution.

- - **setMean:** (double)*mean* **setVariance:** (double)*variance*

  The setMean:setVariance: method sets the mean and the variance of the distribution.

Phase: Using

- - (double)**getSampleWithMean:** (double)*mean* **withStdDev:** (double)*sdev*

The getSampleWithMean:withStdDev: method returns a sample value drawn from a distribution with the specified mean and standard deviation.

- - (double)**getSampleWithMean:** (double)*mean* **withVariance:** (double)*variance*

  The getSampleWithMean:withVariance: method returns a sample value drawn from a distribution with the specified mean and variance.

- - (double)**getStdDev**

  The getStdDev method returns the standard deviation of the distribution.

- - (double)**getVariance**

  The getVariance method returns the variance of the distribution.

- - (double)**getMean**

  The getMean method returns the mean of the distribution.

NormalDist

Name

NormalDist — Normal (Gaussian) distribution

Description

A well-known continuous probability distribution returning doubles.

Protocols adopted by NormalDist

Normal *(see page 245)*

CREATABLE *(see page 44)*

Methods

None

PMMLCG1gen

Name

`PMMLCG1gen` — Prime Modulus Multiplicative Linear Congruential Generator 1

Description

With parameters a = 16,807 and m = 2,147,483,647, this generator has a single full cycle of length (m-1).

Protocols adopted by PMMLCG1gen

PMMLCGgen *(see page 251)*

CREATABLE *(see page 44)*

Methods

None

PMMLCG2gen

Name

`PMMLCG2gen` — Prime Modulus Multiplicative Linear Congruential Generator 2

Description

With parameters a = 48,271 and m = 2,147,483,647, this generator has a single full cycle of length (m-1).

Protocols adopted by PMMLCG2gen

PMMLCGgen *(see page 251)*

CREATABLE *(see page 44)*

Methods

None

PMMLCG3gen

Name

`PMMLCG3gen` — Prime Modulus Multiplicative Linear Congruential Generator 3

Description

With parameters a = 69,621 and m = 2,147,483,647, this generator has a single full cycle of length (m-1).

Protocols adopted by PMMLCG3gen

PMMLCGgen *(see page 251)*

CREATABLE *(see page 44)*

Methods

None

PMMLCG4gen

Name

`PMMLCG4gen` — Prime Modulus Multiplicative Linear Congruential Generator 4

Description

With parameters a = 45,991 and m = 2,147,483,647, this generator has a single full cycle of length (m-1). This is one of the component generators of the CLOG4.

Protocols adopted by PMMLCG4gen

PMMLCGgen *(see page 251)*

CREATABLE *(see page 44)*

Methods

None

PMMLCG5gen

Name

`PMMLCG5gen` — Prime Modulus Multiplicative Linear Congruential Generator 5

Description

With parameters a = 207,707 and m = 2,147,483,543, this generator has a single full cycle of length (m-1). This is one of the component generators of the CLOG4.

Protocols adopted by PMMLCG5gen

PMMLCGgen *(see page 251)*

CREATABLE *(see page 44)*

Methods

None

PMMLCG6gen

Name

`PMMLCG6gen` — Prime Modulus Multiplicative Linear Congruential Generator 6

Description

With parameters a = 138,556 and m = 2,147,483,423, this generator has a single full cycle of length (m-1). This is one of the component generators of the CLOG4.

Protocols adopted by PMMLCG6gen

PMMLCGgen *(see page 251)*

CREATABLE *(see page 44)*

Methods

None

PMMLCG7gen

Name

`PMMLCG7gen` — Prime Modulus Multiplicative Linear Congruential Generator 7

Description

With parameters a = 49,689 and m = 2,147,483,323, this generator has a single full cycle of length (m-1). This is one of the component generators of the CLOG4.

Protocols adopted by PMMLCG7gen

PMMLCGgen *(see page 251)*

CREATABLE *(see page 44)*

Methods

None

PMMLCG8gen

Name

`PMMLCG8gen` — Prime Modulus Multiplicative Linear Congruential Generator 8

Description

With parameters a = 40,014 and m = 2,147,483,563, this generator has a single full cycle of length (m-1). This is one of the component generators of the C2LOGX.

Protocols adopted by PMMLCG8gen

PMMLCGgen *(see page 251)*

CREATABLE *(see page 44)*

Methods

None

PMMLCG9gen

Name

`PMMLCG9gen` — Prime Modulus Multiplicative Linear Congruential Generator 9

Description

With parameters a = 40,692 and m = 2,147,483,399, this generator has a single full cycle of length (m-1). This is one of the component generators of the C2LOGX.

Protocols adopted by PMMLCG9gen

PMMLCGgen *(see page 251)*

CREATABLE *(see page 44)*

Methods

None

PMMLCGgen

Name

`PMMLCGgen` — Prime Modulus Multiplicative Linear Congruential Generator

Description

These generator have single full cycle of length (m-1).

Protocols adopted by PMMLCGgen

SimpleRandomGenerator *(see page 261)*

Methods

None

PSWBgen

Name

`PSWBgen` — Subtract-with-borrow Congruential Generator with prime modulus

Description

PSWB is an improvement on SWB in that the use of a prime modulus guarantees a single full cycle. It's slower, of course.

Protocols adopted by PSWBgen

SimpleRandomGenerator *(see page 261)*

CREATABLE *(see page 44)*

Methods

None

PoissonDist

Name

PoissonDist — Poisson distribution

Description

A distribution used to model the integer number of occurrences of some event over an interval of time or space.

Protocols adopted by PoissonDist

UnsignedDistribution *(see page 271)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- + **create:** (id <Zone>)*aZone* **setGenerator:** (id <SplitRandomGenerator>)*generator* **setVirtualGenerator:** (unsigned)*vGen* **setOccurRate:** (double)*anOccurRate* **setInterval:** (double)*anInterval*

  Use this create message if the generator to be attached is a Split one and both the occurrence rate and the interval are to be set at create time:

- + **create:** (id <Zone>)*aZone* **setGenerator:** (id <SimpleRandomGenerator>)*generator* **setOccurRate:** (double)*anOccurRate* **setInterval:** (double)*anInterval*

  Use this create message if the generator to be attached is a Simple one and both the occurrence rate and the interval are set at create time:

Phase: Setting

- - **setOccurRate:** (double)*anOccurRate*

  The setOccurRate method only sets the occurRate parameter; the interval parameter is left unchanged from its previous or initialized value

- - **setOccurRate:** (double)*anOccurRate* **setInterval:** (double)*anInterval*

  The setOccurRate:setInterval method sets both the occurrence rate and the interval parameters.

- - **setInterval:** (double)*anInterval*

  The setInterval method only sets the interval parameter; the occurRate parameter is left unchanged from its previous or initialized value

Phase: Using

- - (unsigned)**getUnsignedSampleWithOccurRate:** (double)*anOccurRate* **withInterval:** (double)*anInterval*

  The getUnsignedSampleWithOccurRate:andInterval method returns a sample value for the specified occurrence rate and interval. Does not change the the distribution's parameter values set by the setter methods.

- - (unsigned)**getUnsignedSampleWithInterval:** (double)*anInterval*

  The getUnsignedSampleWithInterval method returns a sample value using the distribution's current occurrence rate and new interval value. Causes an error if the occurrence rate has not been previously set.

- - (double)**getInterval**

  The getInterval method returns the interval parameter.

- - (double)**getOccurRate**

  The getOccurRate method returns the occurrence rate parameter.

ProbabilityDistribution

Name

ProbabilityDistribution — Probability Distribution

Description

A process for generating a sequence of random numbers matching the frequencies defined by a specific distribution function. The process is driven by input from a supplied uniform random generator.

Protocols adopted by ProbabilityDistribution

SwarmObject *(see page 211)*

InternalState *(see page 238)*

Methods

Phase: Creating

- + **create:** (id <Zone>)*aZone* **setGenerator:** (id <SimpleRandomGenerator>)*simpleGenerator*

  Use this create message if the generator to be attached is a Simple one:

- + **create:** (id <Zone>)*aZone* **setGenerator:** (id <SplitRandomGenerator>)*splitGenerator* **setVirtualGenerator:** (unsigned)*vGen*

  Use this create message if the generator to be attached is a Split one:

- + **createWithDefaults:** (id <Zone>)*aZone*

  The createWithDefaults method creates a distribution object with a default set of seeds and parameters, and its own private generator.

Phase: Setting

- - **reset**

  The reset method resets the currentCount and other state data.

- - **setGenerator:** (id <SimpleRandomGenerator>)*simpleGenerator*

  Use this message if the generator to be attached is a Simple one:

- - **setGenerator:** (id <SplitRandomGenerator>)*splitGenerator* **setVirtualGenerator:** (unsigned)*vGen*

  Use this message if the generator to be attached is a Split one:

Phase: Using

- - (unsigned long long int)**getCurrentCount**

  The getCurrentCount method returns the count of variates generated.

- - (BOOL)**getOptionsInitialized**

  The getOptionsInitialized returns the value of the parameter.

- - (unsigned)**getVirtualGenerator**

  The getVirtualGenerator returns the number of the virtual generator used.

- - (id <BasicRandomGenerator>)**getGenerator**

  The getGenerator method returns the id of the generator.

RWC2gen

Name

RWC2gen — 2-lag Recursion With Carry generator

Description

This generator is a 2-lag MWC generator implemented using 64-bit math.

Protocols adopted by RWC2gen

SimpleRandomGenerator *(see page 261)*

CREATABLE *(see page 44)*

Methods

None

RWC8gen

Name

RWC8gen — Multiply With Carry generator ("The Mother of all RNG's")

Description

This generator is a combination of 2 16-bit 8-lag Recursion-With-Carry generators.

Protocols adopted by RWC8gen

SimpleRandomGenerator *(see page 261)*

CREATABLE *(see page 44)*

Methods

None

RandomBitDist

Name

`RandomBitDist` — Random Bit Distribution

Description

A generator that returns uniformly distributed single bit values (i.e. fair coin tosses).

Protocols adopted by RandomBitDist

BooleanDistribution *(see page 228)*

CREATABLE *(see page 44)*

Methods

Phase: Using

- `- (BOOL)` **`getCoinToss`**

  `The getCoinToss method returns a YES or NO value.`

SCGgen

Name

`SCGgen` — Subtractive Congruential Generator

Description

SCG is in the Lagged Fibonacci class of generators. These generators use a basic algorithm of the form X_n = f(X_(n-r),X_(n-s)) mod m; r>s The function f is typically xor, addition, subtraction, multiplication or subtraction with carry. It uses simpler math than a basic LCG, but keeps a larger state.

NOT recommended for serious use; these are included for historical reasons (compatibility with earlier releases).

Protocols adopted by SCGgen

SimpleRandomGenerator *(see page 261)*

CREATABLE *(see page 44)*

Methods

None

SWB1gen

Name

`SWB1gen` — Subtract-with-borrow Congruential Generator 1

Description

With the parameters r = 37 and s = 24, this generator has 64 cycles of length 10^354.

Protocols adopted by SWB1gen

SWBgen *(see page 259)*

CREATABLE *(see page 44)*

Methods

None

SWB2gen

Name

`SWB2gen` — Subtract-with-borrow Congruential Generator 2

Description

With the parameters r = 24 and s = 19, this generator has 1536 cycles of length 10^228.

Protocols adopted by SWB2gen

SWBgen *(see page 259)*

CREATABLE *(see page 44)*

Methods

None

SWB3gen

Name

`SWB3gen` — Subtract-with-borrow Congruential Generator 3

Description

With the parameters r = 21 and s = 6, this generator has 192 cycles of length 10^200.

Protocols adopted by SWB3gen

SWBgen *(see page 259)*

CREATABLE *(see page 44)*

Methods

None

SWBgen

Name

`SWBgen` — Subtract-with-borrow Congruential Generator

Description

These generators use a basic algorithm of the form X_n = f(X_(n-r),X_(n-s)) mod m; r>s The function f is typically xor, addition, subtraction, multiplication or subtraction with carry. It uses simpler math than a basic LCG, but keeps a larger state.

Protocols adopted by SWBgen

SimpleRandomGenerator *(see page 261)*

Methods

None

SimpleGenerator

Name

SimpleGenerator — Internal

Protocols adopted by SimpleGenerator

None

Methods

Phase: Creating

- + **create:** (id <Zone>)*aZone* **setStateFromSeeds:** (unsigned *)*seeds*
- + **create:** (id <Zone>)*aZone* **setStateFromSeed:** (unsigned)*seed*

Phase: Using

- - (unsigned long long int)**getCurrentCount**

  The getCurrentCount method returns the count of variates generated.

- - (long double)**getLongDoubleSample**

  The getLongDoubleSample method returns a random floating point number of size long double, uniformly distributed in the range [0.0, 1.0). It uses 2 calls to -getUnsignedSample to fill the mantissa. Note: use of this method is not portable between architectures.

- - (double)**getDoubleSample**

  The getDoubleSample method returns a random floating point number of size double, uniformly distributed in the range [0.0, 1.0). It uses 2 calls to -getUnsignedSample to fill the mantissa.

- - (double)**getThinDoubleSample**

  The getThinDoubleSample method returns a random floating point number of size double, uniformly distributed in the range [0.0, 1.0). It uses 1 call to -getUnsignedSample to fill the mantissa.

- - (float)**getFloatSample**

  The getFloatSample method returns a random floating point number of size float, uniformly distributed in the range [0.0, 1.0). It uses 1 call to -getUnsignedSample to fill the mantissa.

- - (unsigned)**getUnsignedSample**

  The getUnsignedSample method returns a random unsigned integer uniformly distributed over [0,unsignedMax].

SimpleRandomGenerator

Name

`SimpleRandomGenerator` — A Simple (non-split) generator.

Description

This protocol covers all implemented non-split generators.

Protocols adopted by SimpleRandomGenerator

BasicRandomGenerator *(see page 225)*

SimpleGenerator *(see page 260)*

Methods

None

SplitGenerator

Name

SplitGenerator — Internal

Protocols adopted by SplitGenerator

None

Methods

Phase: Creating

- + **create:** (id <Zone>)*aZone* **setA:** (unsigned)*A* **setV:** (unsigned)*v* **setW:** (unsigned)*w* **setStateFromSeeds:** (unsigned *)*seeds*
- + **create:** (id <Zone>)*aZone* **setA:** (unsigned)*A* **setV:** (unsigned)*v* **setW:** (unsigned)*w* **setStateFromSeed:** (unsigned)*seed*

Phase: Setting

- - **initAll**

  The initAll method resets the state of all the virtual generators to the start of segment #0.

- - **initGenerator:** (unsigned)*vGen*

  The initGenerator method resets the state of a virtual generator to the start of segment #0.

Phase: Using

- - (unsigned long long int)**getCurrentCount:** (unsigned)*vGen*

  The getCurrentCount method returns the current count of the specified virtual generator (i.e. the number of variates delivered).

- - (unsigned long long int)**getCurrentSegment:** (unsigned)*vGen*

  The getCurrentSegment method returns the number of the current segment of the specified virtual generator.

- - (long double)**getLongDoubleSample:** (unsigned)*vGen*

  The getLongDoubleSample method returns a random floating-point number of size long double, uniformly distributed in the range [0.0,1.0), from virtual generator (data stream) vGen. This method uses 2 calls to -getUnsignedSample to fill the mantissa. Warning: use of this method is not portable between architectures.

- - (double)**getDoubleSample:** (unsigned)*vGen*

The getDoubleSample method returns a random floating-point number of size double, uniformly distributed in the range [0.0,1.0), from virtual generator (data stream) vGen. This method uses 2 calls to -getUnsignedSample to fill the mantissa.

- - (double)**getThinDoubleSample:** (unsigned)*vGen*

  The getThinDoubleSample method returns a random floating-point number of size double, uniformly distributed in the range [0.0,1.0), from virtual generator (data stream) vGen. This method uses 1 call to -getUnsignedSample to fill the mantissa.

- - (float)**getFloatSample:** (unsigned)*vGen*

  The getFloatSample method returns a random floating-point number of size float, uniformly distributed in the range [0.0,1.0), from virtual generator (data stream) vGen. This method uses 1 call to -getUnsignedSample to fill the mantissa.

- - (unsigned)**getUnsignedSample:** (unsigned)*vGen*

  The getUnsignedSample method returns a random unsigned integer uniformly distributed over the interval [0,unsignedMax] from virtual generator (data stream) vGen.

- - **jumpAllToSegment:** (unsigned long long int)*seg*

  The jumpAlltoSegment: method resets the state of all the virtual generators to the start of the specified segment.

- - **advanceAll**

  The advanceAll method resets the state of all the virtual generators to the start of their next segment.

- - **restartAll**

  The restartAll method resets the state of all the virtual generators to the start of their current segment.

- - **jumpGenerator:** (unsigned)*vGen* **toSegment:** (unsigned long long int)*seg*

  The jumpGenerator:toSegment: method resets the state of a virtual generator to the start of the specified segment.

- - **advanceGenerator:** (unsigned)*vGen*

  The advanceGenerator method resets the state of a virtual generator to the start of the next segment.

- - **restartGenerator:** (unsigned)*vGen*

  The restartGenerator method resets the state of a virtual generator to the start of the current segment.

- - (unsigned)**getSegmentLength**

  The getSegmentLength method returns log2(the current segment length) = w.

- - (unsigned)**getNumSegments**

  The getNumSegments method returns log2(the current number of segments) = v.

- - (unsigned)**getNumGenerators**

```
 The getNumGenerators method returns the current number of  virtual
generators (A).
```

SplitRandomGenerator

Name

`SplitRandomGenerator` — A split generator.

Description

This protocol covers the implemented split generators (C2LCGX and C4LCGX.)

Protocols adopted by SplitRandomGenerator

BasicRandomGenerator *(see page 225)*

SplitGenerator *(see page 263)*

Methods

None

TGFSRgen

Name

`TGFSRgen` — Twisted GFSR generator

Description

With properly chosen parameters, these generators have a single cycle of length 2^(w*N) -1.

Protocols adopted by TGFSRgen

SimpleRandomGenerator *(see page 261)*

Methods

None

TT403gen

Name

`TT403gen` — A single long generator recommended for use.

Description

A single long generator recommended for use.

Protocols adopted by TT403gen

TGFSRgen *(see page 264)*

CREATABLE *(see page 44)*

Methods

None

TT775gen

Name

`TT775gen` — A single long generator recommended for use.

Description

A single long generator recommended for use.

Protocols adopted by TT775gen

TGFSRgen *(see page 264)*

CREATABLE *(see page 44)*

Methods

None

TT800gen

Name

`TT800gen` — A single long generator recommended for use.

Description

A single long generator recommended for use.

Protocols adopted by TT800gen

TGFSRgen *(see page 264)*

CREATABLE *(see page 44)*

Methods

None

UniformDoubleDist

Name

UniformDoubleDist — Uniform Double Distribution

Description

A generator of floating point values uniformly distributed across a half-open interval [min,max). (The interval includes the lower endpoint but excludes the upper endpoint.) NOTE: Setting minValue == maxValue is allowed (and returns minValue).

Protocols adopted by UniformDoubleDist

DoubleDistribution *(see page 234)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- + **create:** (id <Zone>)*aZone* **setGenerator:** (id <SplitRandomGenerator>)*splitGenerator* **setVirtualGenerator:** (unsigned)*vGen* **setDoubleMin:** (double)*minValue* **setMax:** (double)*maxValue*

  Use this create message if the generator to be attached is a Split one:

- + **create:** (id <Zone>)*aZone* **setGenerator:** (id <SimpleRandomGenerator>)*simpleGenerator* **setDoubleMin:** (double)*minValue* **setMax:** (double)*maxValue*

  Use this create message if the generator to be attached is a Simple one:

Phase: Setting

- - **setDoubleMin:** (double)*minValue* **setMax:** (double)*maxValue*

  The setDoubleMin:setMax method sets the minimum and maximum floating point values of the distribution.

Phase: Using

- - (double)**getDoubleWithMin:** (double)*minValue* **withMax:** (double)*maxValue*

  The getDoubleWithMin:withMax: method returns a floating point value within the range [min, max).

- - (double)**getDoubleMax**

  The getDoubleMax method returns the maximum floating point value in the specified range.

- - (double)**getDoubleMin**

The getDoubleMin method returns the minimum floating point value in the specified range.

UniformIntegerDist

Name

`UniformIntegerDist` — Uniform Integer Distribution

Description

A generator of integral values uniformly distributed across a closed interval [min,max]. (The interval includes both its endpoints.) Setting minValue == maxValue is allowed (and returns minValue).

Protocols adopted by UniformIntegerDist

IntegerDistribution *(see page 237)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- `+ **create:** (id <Zone>)*aZone* **setGenerator:** (id <SplitRandomGenerator>)*splitGenerator* **setVirtualGenerator:** (unsigned)*vGen* **setIntegerMin:** (int)*minValue* **setMax:** (int)*maxValue*`

  Use this create message if the generator to be attached is a Split one:

- `+ **create:** (id <Zone>)*aZone* **setGenerator:** (id <SimpleRandomGenerator>)*simpleGenerator* **setIntegerMin:** (int)*minValue* **setMax:** (int)*maxValue*`

  Use this create message if the generator to be attached is a Simple one:

Phase: Setting

- `- **setIntegerMin:** (int)*minValue* **setMax:** (int)*maxValue*`

  The setIntegerMin:setMax: method sets the minimum and maximum integer values to be returned

Phase: Using

- `- (int)**getIntegerWithMin:** (int)*minValue* **withMax:** (int)*maxValue*`

  The getIntegerWithMin:withMax: returns an integer within the interval [min, max].

- `- (int)**getIntegerMax**`

  The getIntegerMax method returns the maximum integer value.

- `- (int)**getIntegerMin**`

  The getIntegerMin method returns the minimum integer value.

UniformUnsignedDist

Name

UniformUnsignedDist — Uniform Unsigned Distribution

Description

A generator of non-negative integral values uniformly distributed across a closed interval [min,max]. (The interval includes both its endpoints.) Setting minValue == maxValue is allowed (and returns minValue).

Protocols adopted by UniformUnsignedDist

UnsignedDistribution *(see page 271)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- + **create:** (id <Zone>)*aZone* **setGenerator:** (id <SplitRandomGenerator>)*splitGenerator* **setVirtualGenerator:** (unsigned)*vGen* **setUnsignedMin:** (unsigned)*minValue* **setMax:** (unsigned)*maxValue*

  Use this create message if the generator to be attached is a Split one:

- + **create:** (id <Zone>)*aZone* **setGenerator:** (id <SimpleRandomGenerator>)*simpleGenerator* **setUnsignedMin:** (unsigned)*minValue* **setMax:** (unsigned)*maxValue*

  Use this create message if the generator to be attached is a Simple one:

Phase: Setting

- - **setUnsignedMin:** (unsigned)*minValue* **setMax:** (unsigned)*maxValue*

  The setUnsignedMin:setMax: method sets the minimum and maximum unsigned values to be returned

Phase: Using

- - (unsigned)**getUnsignedWithMin:** (unsigned)*minVal* **withMax:** (unsigned)*maxVal*

  The getUnsignedWithMin:withMax: returns an unsigned integer within the interval [min, max].

- - (unsigned)**getUnsignedMax**

  The getUnsignedMax method returns the maximum unsigned value.

- - (unsigned)**getUnsignedMin**

  The getUnsignedMin method returns the minimum unsigned value.

UnsignedDistribution

Name

`UnsignedDistribution` — Unsigned Distribution

Description

A probability distribution that returns non-negative integer sample values.

Protocols adopted by UnsignedDistribution

ProbabilityDistribution *(see page 256)*

Methods

Phase: Using

- `- (unsigned)` **`getUnsignedSample`**

  `The getUnsignedSample method returns a non-negative integer sample value.`

General

Name

`random` — Module for random number generation

Description

This module consists of a set of random number generation classes and a set of distribution classes for transforming random number sequences into various simulated probability distributions.

Simtools Library

Overview

Simtools is the stdlib of Swarm. In other words, it is the library where we have parked many miscellaneous classes which, while very useful, are not specific enough to be placed in any other library. So, for example, simtools contains InFile, ObjectLoader and other I/O classes. Simtools contains all *non-GUI* classes - for the miscellaneous classes that do depend on a GUI toolkit being present at link-time, see the documentation for the Simtoolsgui Library *(see page 291)* library.

1. Dependencies

Following are the other header files imported by <simtools.h>:

```
#import <objectbase.h>
```

The *objectbase* library interface is included to provide the basic object support. The *random* library is *no longer included* by default. You will need to explicitly include it to use the default random number generators.

Special global functions - global.h. Users need to include simtools.h in their code in order to call `initSwarm()` but also in order to get access to a set of important pre-initialized objects which are generated in every simulation (e.g. `probeDisplayManager`).

2. Compatibility

- **1.0.5 -> 1.1.** simtools has been split into two: simtools and Simtoolsgui Library *(see page 291)*. The latter now contains all the classes which were GUI-related, so that users can compile and link pure-batch mode simulations (i.e. simulations that don't require Tk/Tcl/BLT, Java AWT or any GUI toolkit).
- **1.0.4 -> 1.0.5.** GUISwarm now inherits from GUIComposite. Because GUIComposite handles the passthru of archiving keys to tkobjc primitives, the method setControlPanelGeometryRecordName is no longer needed; instead, the macro SET_WINDOW_GEOMETRY_RECORD_NAME is provided.
- **1.0.3 -> 1.0.4.** All functions maintain backward compatibility. There are additional features, however, and features previously undocumented.

Documentation and Implementation Status

The simtools library has undergone an upgrade to the status of a library as of 1.0.4. This means the interface now conforms to the library interfaces specifications, the format of the documentation now reflects this. Other than new features, there should be no affect on the user.

Revision History

2003-06-21 simtools.h *mgd*

(__objc_exec_class_for_all_init_modules): Declare. (initSwarmArguments, initSwarmAppArguments): Call __objc_exec_class_for_all_init_modules before _initSwarm_.

2000-07-11 simtools.h *mgd*

Remove inclusion of externvar.h and declaration of swarmGUIMode (moved to swarm.h). Include swarm.h.

2000-03-28 *mgd*

Swarmdocs 2.1.1 frozen.

2000-02-29 *mgd*

Swarmdocs 2.1 frozen.

2000-02-18 simtools.h *mgd*

Add Zone argument conformance to +create:* methods throughout.

1999-09-16 simtools.h *mgd*

Add inhibitExecutableSearchFlag argument.

1999-07-05 simtools.h *alex*

(initSwarmAppArguments): Fixed macro, STRINGIFY(APPNAME) -> APPNAME_STRING.

1999-06-23 simtools.h *mgd*

Reflect these changes.

1999-06-21 simtools.h *alex*

(InFile, OutFile, AppendFile, ObjectSaver, ObjectLoader): Markup with `//x:' doc-string as deprecated protocols. Move explanatory text originally in `//D:' markup to new tag, where appropriate.

1999-06-21 simtools.h *alex*

(_initSwarm_, STRINGIFY, STRINGIFYSYM, APPNAME_STRING, APPVERSION_STRING, BUGADDRESS_STRING): Remove, hide from user in new include'd initSwarm.h.

1999-06-10 simtools.h *mgd*

Stringify application name, version, and bug address. (Don't require quoting from make.)

1999-06-09 simtools.h *alex*

(initSwarm): Redefine as a macro. Call new internal function _initSwarm_(). If not explicitly set by the macro, pass APPNAME, BUGADDRESS, APPVERSION which will be passed from the application Makefile to the internal _initSwarm_(). (initSwarmBatch): Likewise. (initSwarmApp): Likewise. (initSwarmAppBatch): Likewise. (initSwarmAppOptions): Likewise. (initSwarmAppOptionsBatch): Likewise (_initSwarm_): Define internal function.

1999-05-29 simtools.h *mgd*

Include externvar.h.

1999-05-28 simtools.h *mgd*

Use `externvar' for swarmGUIMode.

1999-02-26 simtools.h *mgd*

Add CREATABLE tags to all non-abstract protocols.

1999-02-07 simtools.h *mgd*

(InFile): Add a warning about error return behavior.

1998-12-11 simtools.h *vjojic*

Remove ListShuffler protocol and ListShuffler class definition

1998-08-23 simtools.h *mgd*

(ListShuffler): New protocol.

1998-08-20 simtools.h *mgd*

(ObjectLoader): Add it.

1998-07-08 simtools.h *mgd*

(ActiveOutFile): Remove protocol and class object.

1998-06-17 Makefile.am *mgd*

Include from refbook/ instead of src/.

1998-06-15 Makefile.am *mgd*

(MODULE): New variable. Include Makefile.rules from src. Remove everything else.

1998-06-12 simtools00.sgml, simtoolscont.sgml *mgd*

Update IDs to SWARM.module.SGML.type.

1998-06-06 simtools.ent *mgd*

Use public identifiers.

1998-06-05 Makefile.am *mgd*

(swarm_ChangeLog): Add.

1998-06-03 simtools.h *mgd*

Update documentation tags.

1998-05-23 Makefile.am *mgd*

New file.

1998-05-23 simtools.ent.in *mgd*

New file.

1998-05-23 simtools.ent *mgd*

Removed.

1998-05-22 *mgd*

Begin revision log.

1998-04-16 simtools.h *mgd*

Move tagged documentation to the respective protocols definition.

1998-04-06 simtools.h *mgd*

Declare new function initSwarmArguments.

1998-02-26 simtools.h *mgd*

Remove ControlPanel, WindowGeometryRecordName, CompositeWindowGeometryRecordName, ActioonCache, ProbeDisplay, CompleteProbeDisplay, ProbeDisplayManager, GUIComposite, GUISwarm, ActiveGraph protocols (moved to simtoolsgui). Remove declaration of probeDisplayManager. Remove mention of @classes ActionCache, ProbeDisplay, CompleteProbeDisplay, ProbeDisplayManager, GUISwarm, and ActiveGraph. Don't include GUISwarm.h.

1998-01-27 simtools.h *mgd*

Likewise.

1998-01-24 simtools.h *mgd*

Don't include tkobjc.h.

1998-01-20 simtools.h *mgd*

Add copyright header.

1998-01-15 simtools.h *mgd*

(SET_WINDOW_GEOMETRY_RECORD_NAME): Add convenience macro. ProbeDisplay and CompleteProbeDisplay derive from WindowGeometryRecordName protocol. CompositeWindowGeometryRecordName: New protocol. ActionCache and GUIComposite protocols derive from it.

1998-01-14 simtools.h *mgd*

Drop windowGeometryRecordNameForComponent and windowGeometryRecordName fro WindowGeometryRecordName protocol.
(SET_COMPONENT_WINDOW_GEOMETRY_RECORD_NAME_FOR): New macro.
(SET_COMPONENT_WINDOW_GEOMETRY_RECORD_NAME): New macro. (CREATE_PROBE_DISPLAY, CREATE_COMPLETE_PROBE_DISPLAY, CREATE_ARCHIVED_PROBE_DISPLAY, CREATE_ARCHIVED_COMPLETE_PROBE_DISPLAY): Rename to be in caps.

1998-01-13 simtools.h *mgd*

(_createProbeDisplay, _createCompleteProbeDisplay, createArchivedProbeDisplayNamed, createArchivedCompleteProbeDisplayNamed): Declare new functions.
(CREATE_ARCHIVED_PROBE_DISPLAY): New macro to pass variable variable name through to archiving functions above. (CREATE_ARCHIVED_COMPLETE_PROBE_DISPLAY): Likewise. (ProbeDisplayManager): Replace setWindowGeometryRecordName versions of createProbeDisplayFor with createArchivedProbeDisplayFor:variableName: and createArchivedCompleteProbeDisplayFor:variableName.
(CREATE_PROBE_DISPLAY, CREATE_COMPLETE_PROBE_DISPLAY): Define macros.

1998-01-13 simtools.h *mgd*

Add dropProbeDisplaysFor: to the ProbeDisplayManager protocol.

1998-01-12 simtools.h *mgd*

Declare buildWindowGeometryRecordName. New protocol WindowGeometryRecordName, derive ActionCache, and GUISwarm from it.

1998-01-06 simtools.h *mgd*

(Action): Rename setControlPanelGeometryRecordName to setWindowGeometryRecordName. (ControlPanel): Remove setControlPanelGeometryRecordName.

1997-12-19 simtools.h *mgd*

(ActionCache): Make setScheduleContext a Using phase method. Add setControlPanelGeometryRecordName. (InFile): Add getline to InFile protocol. Reformatting throughout (remove gratuitous space).

1997-12-18 simtools.h *mgd*

Add setWindowGeometryRecordName to ProbeDisplay protocol.

1997-12-10 simtools.h *mgd*

Consify toExecute argument of ActionCache sendActionOfType. Consify setBaseName argument of UName create. Consify argument to UName setBaseName. Consify return of UName getNewName. Consify withName argument of InFile create. Consify withName argument of OutFile create. Consify argument to OutFile putString. Consify withName argument of AppendFile create. Consify argument to AppendFile putString.

1997-12-07 simtools.h *mgd*

(CompleteProbeDisplay): Add getMarkedForDropFlag to protocol.

1997-12-04 simtools.h *mgd*

(ProbeDisplayManager): Add setDropImmediatelyFlag.

1997-12-01 simtools.h *alex*

Added AppendFile and added reverseOrder method to QSort protocol.

1997-11-29 simtools.h *mgd*

Make Object{Loader,Saver} fromFileNamed: arguments const.

1997-11-29 simtools.h *mgd*

Add methods below to ProbeDisplayManager protocol.

1997-11-20 simtools.h *mgd*

(library-wide): Brought simtools library into conformance with protocol interface standard.

AppendFile [Deprecated]

Name

`AppendFile` — A class for appended file output.

Description

Deprecated: Warning: the error return behavior of these methods is fragile, only end of file is reported as an error. It is probably not wise to use use this inteface unless your text processing needs are very simple.

This class subclasses from OutFile, the only functional difference being that it opens a given file in Append Mode rather than in Overwrite mode.

Protocols adopted by AppendFile

OutFile [Deprecated] *(see page 286)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- `+ create: (id <Zone>)aZone withName: (const char *)theName`

  for backward compatibility

- `+ create: (id <Zone>)aZone setName: (const char *)theName`

  The create:setName: method is the create method for AppendFiles, where theName is the name of the file to open.

InFile [Deprecated]

Name

`InFile` — Class to perform file input.

Description

Deprecated: Warning: the error return behavior of these methods is fragile, only end of file is reported as an error. It is probably not wise to use use this inteface unless your text processing needs are very simple.

This class is (was) intended to simplify the input file-I/O in Swarm. It essentially deals with the detailed file opening and closing routines thus alleviating the need for C file I/O procedure calls.

Protocols adopted by InFile

SwarmObject *(see page 211)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- `+ create: (id <Zone>)aZone withName: (const char *)theName`

  for backward compatibility

- `+ create: (id <Zone>)aZone setName: (const char *)theName`

  This is the create method for InFiles, where theName is, of course the name of the file to open.

Phase: Using

- `- (void)drop`

  The drop method must be called when the user wants to close the file. Only by dropping the InFile object will the file truly be closed.

- `- (int)skipLine`

  Skips a line.

- `- (int)unGetChar: (char)aChar`

  Returns a character for re-reading.

- `- (int)getChar: (char *)aChar`

  The getChar: method takes a pointer of type char and loads it with an instance of that type from the open file. In case of failure, the method returns 0.

- `- (int)getFloat: (float *)aFloat`

The getFloat: method takes a pointer of type float and loads it with an instance of that type from the open file. In case of failure, the method returns 0.

- - (int)**getDouble:** (double *)*aDouble*

  The getDouble: method takes a pointer of type double and loads it with an instance of that type from the open file. In case of failure, the method returns 0.

- - (int)**getUnsignedLong:** (unsigned long *)*anUnsLong*

  The getUnsignedLong: method takes a pointer of type unsigned long and loads it with an instance of that type from the open file. In case of failure, the method returns 0.

- - (int)**getLong:** (long *)*aLong*

  The getLong: method takes a pointer of type long and loads it with an instance of that type from the open file. In case of failure, the method returns 0.

- - (int)**getUnsigned:** (unsigned *)*anUnsigned*

  The getUnsigned: method takes a pointer of type unsigned and loads it with an instance of that type from the open file. In case of failure, the method returns 0.

- - (int)**getInt:** (int *)*anInt*

  The getInt: method takes a pointer of type Int and loads it with an instance of that type from the open file. In case of failure, the method returns 0.

- - (int)**getLine:** (char *)*aLine*

  The getLine: method loads the argument string with the characters up to, but not including a newline character.

- - (int)**getWord:** (char *)*aWord*

  The getWord: method returns a string that does not contain spaces, tabs, and newlines.

NSelect

Name

`NSelect` — A class to select exactly N elements at random from a collection.

Description

NSelect selects exactly N elements from a collection without repetition. A target collection must be provided.

Protocols adopted by NSelect

SwarmObject *(see page 211)*

CREATABLE *(see page 44)*

Methods

Phase: Using

- `+ (void)` **`select:`** `(int)`*`n`* **`from:`** *`aCollection`* **`into:`** *`bCollection`*

  The select:from:into: method selects exactly N elements from a collection without repetition into another collection. The selection algorithm is from Knuth.

ObjectLoader [Deprecated]

Name

`ObjectLoader` — A class to load an object's instance variables from a file.

Description

Deprecated: Use the Archiver protocol in the defobj library as a replacement for this ad-hoc format

This class is used to initialize the variables of a target object from a data file. The data file is required to have a very simple format.

Protocols adopted by ObjectLoader

SwarmObject *(see page 211)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- + **load:** *anObject* **fromAppDataFileNamed:** (const char *)*aFileName*

  The load:fromAppConfigFileNamed: method loads anObject from th application-specific data file named aFileName. The ObjectLoader class will open the file, initialize the object with its contents and then close the file.

- + **load:** *anObject* **fromAppConfigFileNamed:** (const char *)*aFileName*

  The load:fromAppConfigFileNamed: method loads anObject from th application-specific configuration file named aFileName. The ObjectLoader class will open the file, initialize the object with its contents and then close the file.

- + **load:** *anObject* **fromFileNamed:** (const char *)*aFileName*

  The load:fromFileNamed: method loads anObject from the file named aFileName. The ObjectLoader class will open the file, initialize the object with its contents and then close the file.

- + **load:** *anObject* **from:** *aFileObject*

  The load:from: method loads anObject from the previously opened aFileObject without returning an actual instance of the ObjectLoader class. The FileObject remains open after the method has been called.

Phase: Setting

- - **setTemplateProbeMap:** (id <ProbeMap>)*probeMap*

  The setTemplateProbeMap: method is used to specify which variables of the target object(s) should be loaded by the ObjectLoader instance to which this message was sent.

- - **setFileObject:** *aFileObject*

  The setFileObject: method sets the source fileObject which the instance of the ObjectLoader class should use by sending it this message.

Phase: Using

- - **updateCache:** *exampleTarget*

  The updateCache: method should be called if an ObjectLoader instance is going to initialize a large number of objects from the same class.

- - **loadObject:** *anObject*

  The loadObject: message must be sent to an instance of the ObjectLoader class in order to initialize the target object from the requested file.

ObjectSaver [Deprecated]

Name

`ObjectSaver` — A class to save an object's instance variables to a file.

Description

Deprecated: Use the Archiver protocol in the defobj library as a replacement for this ad-hoc format

This class is used to write an object's variables to a specified file. If only a subset of the variables should be written out, the set is specified by a template ProbeMap (where the ProbeMap will contain Probes for those variables which should be saved).

Protocols adopted by ObjectSaver

SwarmObject *(see page 211)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- + **save:** *anObject* **toFileNamed:** (const char *)*aFileName*

  The save:toFileNamed: method saves the entire target object to the file aFileName.

- + **save:** *anObject* **toFileNamed:** (const char *)*aFileName* **withTemplate:** (id <ProbeMap>)*aProbeMap*

  The save:toFileNamed:withTemplate: method saves the subset of variables specified in a template from the target object to the file aFileName.

- + **save:** *anObject* **to:** *aFileObject*

  The save:to: method saves the entire target object without actually returning an instance of ObjectSaver to the user.

- + **save:** *anObject* **to:** *aFileObject* **withTemplate:** (id <ProbeMap>)*aProbeMap*

  The save:to:withTemplate: method saves the subset of target object variables specified in a template from anObject without actually returning an instance of ObjectSaver to the user.

Phase: Setting

- - **setTemplateProbeMap:** (id <ProbeMap>)*aProbeMap*

  The setTemplateProbeMap: method is used to specify which variables of the source object(s) should be saved by the ObjectSaver instance to which this message was sent.

- - **setFileObject:** *aFileObject*

The setFileObject: method sets the target fileObject which the instance of the ObjectSaver class should use.

Phase: Using

- - **saveObject:** *anObject*

  The saveObject: message tells an instance of the ObjectSaver class to save the state of the target object into the requested file.

OutFile [Deprecated]

Name

`OutFile` — A class to perform file output.

Description

Deprecated: Warning: the error return behavior of these methods is fragile, only end of file is reported as an error. It is probably not wise to use use this inteface unless your text processing needs are very simple.

This class is intended to simplify output file-I/O in Swarm. It essentially deals with the detailed file opening and closing routines thus alleviating the need for C file I/O procedure calls.

Protocols adopted by OutFile

SwarmObject *(see page 211)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- `+ create: (id <Zone>)aZone withName: (const char *)theName`

  for backward compatibility

- `+ create: (id <Zone>)aZone setName: (const char *)theName`

  The create:setName: method opens a file named theName and creates an Outfile object.

Phase: Using

- `- (void)drop`

  The drop method closes the open file. This method must be called to close the file.

- `- putTab`

  The putTab method writes a tab into the open file.

- `- putNewLine`

  The putNewline method writes a newline into the open file.

- `- putChar: (char)aChar`

  The putChar: method takes an instance of type char and writes it into the open file.

- `- putFloat: (float)aFloat`

The putFloat: method takes an instance of type float and writes it into the open file.

- - **putDouble:** (double)*aDouble*

  The putDouble: method takes an instance of type double and writes it into the open file.

- - **putUnsignedLong:** (unsigned long)*anUnsLong*

  The putUnsignedLong: method takes an instance of type unsigned long and writes it into the open file.

- - **putLong:** (long)*aLong*

  The putLong: method takes an instance of type long and writes it into the open file.

- - **putUnsigned:** (unsigned)*anUnsigned*

  The putUnsigned: method takes an instance of type unsigned and writes it into the open file.

- - **putInt:** (int)*anInt*

  The putInt: method takes an instance of type int and writes it into the open file.

- - **putString:** (const char *)*aString*

  The putString: method takes an instance of type string and writes it into the open file.

QSort

Name

QSort — A class to sort a collection.

Description

QSort is simply a "wrapper" for the C native "qsort" function, as applied to a Swarm collection. The values will appear in ascending order by default. Reversing the order of a collection can be made by calling reverseOrderOf. All these methods modify the underlying collection, so any indexes should always be regenerated.

Protocols adopted by QSort

SwarmObject *(see page 211)*

CREATABLE *(see page 44)*

Methods

Phase: Using

- + (void)**reverseOrderOf:** *aCollection*

  The reverseOrderOf: method reverses the current order of a collection. To make a "reversed" sort, simply call one of the appropriate "sort" methods on a collection then call this function on the same collection.

- + (void)**sortNumbersIn:** *aCollection*

  The sortNumbersIn: method performs integer comparisons on the objects in the collection using the default "compare" function. The default assumes that the numbers should be monotonically increasing.

- + (void)**sortNumbersIn:** *aCollection* **using:** (int (*) (const void*, const void*))*comp_fun*

  The sortNumbersIn:using: method performs integer comparisons on the objects in the collection with the specified comparison function for the object.

- + (void)**sortObjectsIn:** *aCollection*

  The sortObjectsIn: method will sort the objects in the collection with the "compare' function for the object. If the objects don't provide a compare function, sortObjectsIn uses the default from the defobj library.

- + (void)**sortObjectsIn:** *aCollection* **using:** (SEL)*aSelector*

  The sortObjectsIn:using: method will sort the objects in the collection with the specified comparison function for the object.

UName

Name

UName — A class used to generate unique names (e.g. "critter1", "critter2" etc.)

Description

This class is used to generate unique names (agent0, agent1, agent2...) for objects in a simulation. The user will typically create an instance of the UName class initialized with a baseName presented either as a (const char *) or an object of class String. The user can then request new names, again either as (const char *)'s or as instances of the String class. The user can also reset the counter used to generate the names in case s/he wants to restart naming objects with the same baseName.

Note: Both in the case of initialization by (const char *) and initialization by an instance of the String class, the original is copied not stored internally so it is up to the user to free the original (const char *) or String instance if/when necessary!

Protocols adopted by UName

SwarmObject *(see page 211)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- - **setBaseNameObject:** *aStringObject*

  The setBaseNameObject: method is used to set the base name given an object of class String.

- - **setBaseName:** (const char *)*aString*

  The setBaseName: method is used to set the base name given a const char *.

- + **create:** (id <Zone>)*aZone* **setBaseNameObject:** *aStringObject*

  The create:setBaseNameObject: method is used to create an instance of the UName class and set the base name given an object of class String. This method will automatically reset the counter.

- + **create:** (id <Zone>)*aZone* **setBaseName:** (const char *)*aString*

  The create:setBaseNameObject: method is used to create an instance of the UName class and set the base name given a const char *. This method will automatically reset the counter.

Phase: Using

- - **resetCounter**

  Resets the counter used as a suffix in the unique names generated.

- - **getNewNameObject**

The getNewNameObject generates a new name as a String Object.

- - (const char *)**getNewName**

  The getNewName method generates a new name as a character string.

General

Name

simtools — General simulation tools

Description

A collection of tools that are only loosely related to each other. the class hierarchy is virtually flat.

Macros

- **initSwarm**(*argc*, *argv*)

  Initializes the Swarm libraries without version or bug-report-address information.

- **initSwarmApp**(*argc*, *argv*, *version*, *bugAddress*)

  Initializes the Swarm libraries for an application.

- **initSwarmAppArguments**(*argc*, *argv*, *version*, *bugAddress*, *argumentsClass*)

  Like initSwarmApp, but specifies what class to use for argument parsing, typically this will be a subclass of Arguments.

- **initSwarmAppBatch**(*argc*, *argv*, *version*, *bugaddress*)

  Like initSwarmApp, but initializes in batch-mode only

- **initSwarmAppOptions**(*argc*, *argv*, *version*, *bugAddress*, *options*, *optionFunc*)

  Like initSwarmApp, but specifies a parsing function .

- **initSwarmAppOptionsBatch**(*argc*, *argv*, *version*, *bugAddress*, *options*, *optionFunc*)

  Like initSwarmAppOptions, but initializes in batch-mode only

- **initSwarmArguments**(*argc*, *argv*, *argumentsClass*)

  Like initSwarm, but specifies what class to use for argument parsing, typically this will be a subclass of Arguments.

- **initSwarmBatch**(*argc*, *argv*)

  Initializes the Swarm libraries for batch mode without version or bug report address information.

Functions

- void **__objc_exec_class_for_all_initial_modules**()

Simtoolsgui Library

Overview

Simtoolsgui contains all miscellaneous *GUI* classes. It depends on a GUI toolkit being present at link-time, see the documentation for the Simtoolsgui Library *(see page 291)* library for non-GUI classes.

1. Dependencies

Following are the other header files imported by <simtoolsgui.h>:

```
#import <objectbase.h>
#import <activity.h>
```

2. Compatibility

- **1.0.5 -> 1.1.** This new library has been created it contains all the classes which were GUI-related, so that users can compile and link pure-batch mode simulations (i.e. simulations that don't require Tk/Tcl/BLT, Java AWT or any GUI toolkit).

Documentation and Implementation Status

First creation of simtoolsgui documentation.

Revision History

2001-02-07 simtoolsgui.h *mgd*

(GUISwarm): Add GETTERS section.

2000-09-20 simtoolsgui.h *mgd*

(WindowGeometryRecordName): Declare -setSaveSizeFlag:.

2000-09-18 simtoolsgui.h *mgd*

Include gui.h. (MessageProbeWidget, MultiVarProbeWidget): Qualify setParent: argument with Frame. (MessageProbeWidget): Qualify argument to setProbe: with Probe. (SimpleProbeDisplay): Qualify argument to setProbeMap:. (ProbeDisplayManager): Qualify addProbeDisplay: and removeProbeDisplay: arguments with CommonProbeDisplay.

2000-08-31 simtoolsgui.h *alex*

Add doc string to "doTkEvents" method.

2000-05-18 simtoolsgui.h *mgd*

(SimpleProbeDisplay, ProbeDisplay): Remove -getProbemap.

2000-04-28 simtoolsgui.h *mgd*

(ActionCache, CommonProbeDisplay, MultiVarProbeWidget): Adopt SwarmObject.

2000-03-28 *mgd*

Swarmdocs 2.1.1 frozen.

2000-02-29 *mgd*

Swarmdocs 2.1 frozen.

1999-11-23 simtoolsgui.h *mgd*

Make ControlState* symbols conform to Symbol.

1999-08-22 simtoolsgui.h *mgd*

(ControlPanel, ActionCache, CommonProbeDisplay, SimpleProbeDisplay, CompleteProbeDisplay): Switch from CREATABLE to RETURNABLE. (SingleProbeDisplay, GUIComposite): Remove CREATABLE. (GUISwarm): Add ActionCache and ControlPanel return typing to getters.

1999-07-21 simtoolsgui.h *vjojic*

Make CommonProbeDisplay protocol CREATABLE. Make SingleProbeDisplay protocol CREATABLE. Add SimpleProbeDisplay protocol and make it CREATABLE.

1999-05-29 simtoolsgui.h *mgd*

Include externvar.h.

1999-05-28 simtoolsgui.h *mgd*

Use `externvar' for external variables.

1999-04-26 simtoolsgui.h *alex*

(GUIComposite): Add compliance to SwarmObject protocol. Revert this change, since protocol is abstract, confirmation added to instantiatable protocols.

1999-04-01 simtoolsgui.h *vjojic*

Protocol GUISwarm inherits protocol Swarm.

1999-03-31 simtoolsgui.h *vjojic*

Add methods getActionCache and getControlPanel to GUISwarm.

1999-02-26 simtoolsgui.h *mgd*

Add CREATABLE tags to all non-abstract protocols.

1999-01-31 simtoolsgui.h *mgd*

(ProbeDisplayManager): Declare -getDropImmediateFlag.

1998-12-31 simtoolsgui.h *mgd*

(ActionCache, GUISwarm): Declare -drop.

1998-09-03 simtoolsgui.h *mgd*

(CommonProbeDisplay): Declare -getTopLevel.

1998-08-18 simtoolsgui.h *mgd*

(SingleProbeDisplay): Add //S and //D.

1998-07-15 simtoolsgui.h *mgd*

(CommonProbeDisplay): Remove setProbedObject: and getProbedObject from protocol (moved to SingleProbeDisplay). (SingleProbeDisplay): New protocol. (CompleteProbeDisplay, SimpleProbeDisplay): Adopt SingleProbeDisplay instead of CommonProbeDisplay. (MultiVarProbeDisplay): New protocol. (MultiVarProbeWidget): Replace setProbeList: with setProbeMap:. Rename setLabelingFlag: to setFieldLabelingFlag:. Rename setAgentNameSelector: to setObjectNameSelector:. Add +createBegin:. Add MultiVarProbeDisplay class object.

1998-07-14 simtoolsgui.h *mgd*

Add MultiVarProbeWidget protocol and class object.

1998-07-10 simtoolsgui.h *alex*

(MessageProbeWidget): Add phase tags.

1998-07-08 simtoolsgui.h *mgd*

(ActiveGraph): Remove protocl and class object.

1998-07-07 simtoolsgui.h *mgd*

(MessageProbeWidget): New protocol and class object.

1998-06-17 Makefile.am *mgd*

Include from refbook/ instead of src/.

1998-06-17 simtoolsgui00.sgml *alex*

Added missing description of activity.h dependency.

1998-06-15 Makefile.am *mgd*

(MODULE): New variable. Include Makefile.rules from src. Remove everything else.

1998-06-12 simtoolsgui00.sgml, simtoolsguicont.sgml *mgd*

Update IDs to SWARM.module.SGML.type.

1998-06-06 simtoolsgui.ent *mgd*

Use public identifiers.

1998-06-05 Makefile.am *mgd*

(swarm_ChangeLog): Add.

1998-06-03 simtoolsgui.h *mgd*

Update documentation tags. Prefix global references with extern.

1998-05-28 simtoolsgui.h *mgd*

Include activity.h.

1998-05-23 Makefile.am *mgd*

New file.

1998-05-23 simtoolsgui.ent.in *mgd*

New file.

1998-05-23 simtoolsgui.ent *mgd*

Removed.

1998-05-22 *mgd*

Begin revision log.

1998-05-22 simtoolsgui.h *alex*

(ActionCache, probeDisplayManager, initSimtoolsGUI): Added doc tags (//G) to global variables and symbols. (SET_WINDOW_GEOMETRY_RECORD_NAME,CREATE_PROBE_DISPLAY, CREATE_COMPLETE_PROBE_DISPLAY, CREATE_ARCHIVED_PROBE_DISPLAY, CREATE_ARCHIVED_COMPLETE_PROBE_DISPLAY): Added doc tags (//#) for macros.

1998-05-07 simtoolsgui.h *mgd*

(_createProbeDisplay, createArchivedProbeDisplayNamed): Return id conforming to ProbeDisplay protocol. (_createCompleteProbeDisplay, createArchivedCompleteProbeDisplayNamed): Return an object conforming to CompleteProbeDisplay protocol. (-createProbeDisplayFor:, -createArchivedProbeDisplayFor:variableName:): Return an object conforming to ProbeDisplay protocol. (-createCompleteProbeDisplayFor:, -createArchivedCompleteProbeDisplayFor:variableName:): Return an object conforming to the CompleteProbeDisplay protocol. (-createDefaultProbeDisplayFor:, -createArchivedDefaultProbeDisplayFor:variableName:): Declare. Return an object conforming to the CompleteProbeDisplay protocol. (-getMarkedForDrop): Return BOOL.

1998-05-06 simtoolsgui.h *mgd*

(WindowGeometryRecordName, CompositeWindowGeometryRecordName, GUIComposite): Add phase tags. (WindowGeometryRecordName, CompositeWindowGeometryRecordName, ControlPanel, ActionCache, CommonProbeDisplay, ProbeDisplay, CompleteProbeDisplay, ProbeDisplayManager, GUIComposite, GUISwarm, ActiveGraph): Move //S and //D tags inside protocol.

1998-04-16 simtoolsgui.h *mgd*

Add comment tags with documentation.

ActionCache

Name

`ActionCache` — A class to manage threads and Swarms.

Description

A class that provides a smart bag into which actions can be thrown by other threads and Swarms intended for insertion on it's Swarm's schedule.

Protocols adopted by ActionCache

CompositeWindowGeometryRecordName *(see page 300)*

SwarmObject *(see page 211)*

RETURNABLE *(see page 66)*

Methods

Phase: Creating

- - **createProcCtrl**
- - **setControlPanel:** (id <ControlPanel>)*cp*

Phase: Using

- - **waitForControlEvent**
- - **doTkEvents**

  A message that processes any input or output events of the Tk toolkit. Scheduling -doTkEvents ensures Tk keeps the user interface up-to-date. Without scheduling it as part of the GUI code, the Tk events would just queue up and never get processed, resulting in a static, unresponsive user interface.
- - **getPanel**
- - **verifyActions**
- - **sendQuitAction**
- - **sendNextAction**
- - **sendStepAction**
- - **sendStopAction**
- - **sendStartAction**
- - **sendActionOfType:** (id <Symbol>)*type* **toExecute:** (const char *)*cmd*
- - **deliverActions**

- - **insertAction:** *actionHolder*
- - **setScheduleContext:** (id <Swarm>)*context*

CommonProbeDisplay

Name

CommonProbeDisplay — A protocol underlying ProbeDisplay and CompleteProbeDisplay

Description

This protocol provides the common interface to all kinds of ProbeDisplays.

Protocols adopted by CommonProbeDisplay

SwarmObject *(see page 211)*

WindowGeometryRecordName *(see page 311)*

Methods

Phase: Using

- - **getTopLevel**
- - (BOOL)**getMarkedForDropFlag**
- - (void)**update**

  This method maintains consistency between the values of the probedObject's variables and the values which are displayed in the ProbeDisplay. Ideally, this method should be called every time the object is modified by the simulation. In practice, the user schedules an update on the probeDisplayManager which in turn communicates to all the active ProbeDisplays in the system.

CompleteProbeDisplay

Name

`CompleteProbeDisplay` — A class that generates a complete ProbeMap for an object.

Description

A class which generates a GUI to a complete ProbeMap of probes applied to a given target object (by complete we mean that all the probes for the target object's class and its superclasses are included)...

Protocols adopted by CompleteProbeDisplay

SingleProbeDisplay *(see page 310)*

RETURNABLE *(see page 66)*

Methods

None

CompositeWindowGeometryRecordName

Name

`CompositeWindowGeometryRecordName` — Protocol for archiving objects with several GUI components.

Description

Protocol for assigning archiving names to components of an object with several GUI components.

Protocols adopted by CompositeWindowGeometryRecordName

WindowGeometryRecordName *(see page 311)*

Methods

Phase: Creating

- `-` **`setWindowGeometryRecordNameForComponent:`** `(const char *)`*`componentName`* **`widget:`** *`widget`*

  `Update the list of components, and compute the derived archiving name.`

Macros

- **`SET_COMPONENT_WINDOW_GEOMETRY_RECORD_NAME`**`(`*`theWidget`*`)`
- **`SET_COMPONENT_WINDOW_GEOMETRY_RECORD_NAME_FOR`**`(`*`obj, theWidget`*`)`

ControlPanel

Name

`ControlPanel` — Class to control the top level SwarmProcess

Description

ControlPanel keeps track of the users requests to run, stop, quit, or time step the simulation. It cooperates with the GUISwarm to control the execution of activities in Swarm.

Protocols adopted by ControlPanel

SwarmObject *(see page 211)*

RETURNABLE *(see page 66)*

Methods

Phase: Using

- - **setStateSave**

  Saves the objects that are registered for archiving.

- - **setStateNextTime**

  Stop the running activity, and then set state to `ControlStateNextTime'.

- - **setStateQuit**

  Terminate activities, and set state to `ControlStateQuit'.

- - **setStateStepping**

  Stop the running activity, and then set state to `ControlStateStepping'.

- - **setStateStopped**

  The -setStateStopped message is particularly useful since it will cause the simulation to stop until the user interactively sets it back in motion (in other words, this method is useful in generating a software-triggered pause).

- - **setStateRunning**

  Sets the state to `running'.

- - **startInActivity:** (id <SwarmActivity>)*activityID*

- - **setState:** (id <Symbol>)*s*

- - (id <Symbol>)**getState**

  Get the current button state of the controlpanel. Is one of ControlStateRunning, ControlStateStopped, ControlStateStepping, ControlStateNextTime, or ControlStateQuit.

GUIComposite

Name

`GUIComposite` — Base class for objects that use several GUI components.

Description

Base class for objects that use several GUI components.

Protocols adopted by GUIComposite

CompositeWindowGeometryRecordName *(see page 300)*

Methods

Phase: Using

- `- (void)`**`disableDestroyNotification`**
- `- (void)`**`enableDestroyNotification:`** *`notificationTarget`* **`notificationMethod:`** `(SEL)`*`notificationMethod`*

GUISwarm

Name

`GUISwarm` — A version of the Swarm class which is graphics aware.

Description

GUISwarm is a subclass of Swarm that is used as a toplevel Swarm for simulations running with a graphical user interface. The GUISwarm creates a ControlPanel automatically for you and defines a -go method that interprets the state of the ControlPanel to keep things running in response to user input. Users subclass GUISwarm much like they subclass a normal Swarm, implementing the same kind of buildObjects, buildActions, and activateIn methods. When you are done building your Observer Swarm, start it as a toplevel via [myGUISwarm go].

The control panel places a few responsibilities on the GUISwarm subclass author. In particular, a message to [controlPanel doTkEvents] should be scheduled fairly frequently - only when that method is executed will the user interface update (and the stop button be checked). Also, it is often useful to use [controlPanel setStateStopped] to wait for the user to indicate they're ready for execution to proceed.

Protocols adopted by GUISwarm

Swarm *(see page 210)*

WindowGeometryRecordName *(see page 311)*

CREATABLE *(see page 44)*

Methods

Phase: Using

- `- go`

  Start the activity running, and also handle user requests via the control panel. Returns either Completed (the model ran until requested to terminate) or ControlStateQuit (the user pressed the quit button).

- `- (id <ControlPanel>)getControlPanel`
- `- (id <ActionCache>)getActionCache`

MessageProbeWidget

Name

`MessageProbeWidget` — A widget for editing the arguments of a MessageProbe.

Description

A widget for editing the arguments of a MessageProbe.

Protocols adopted by MessageProbeWidget

CREATABLE *(see page 44)*

Methods

Phase: Creating

- `-` **`setProbe:`** `(id <Probe>)`*`probe`*
- `-` **`setObject:`** *`object`*
- `-` **`setParent:`** `(id <Frame>)`*`parent`*

Phase: Using

- `-` `(void)`**`pack`**

MultiVarProbeDisplay

Name

`MultiVarProbeDisplay` — A display for displaying a ProbeMap across a number of objects.

Description

This ProbeDisplay extracts all the variable probes from a probe map and creates a variable probe entry for each object in the list provided by the user.

Protocols adopted by MultiVarProbeDisplay

CommonProbeDisplay *(see page 298)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- - **`setObjectNameSelector:`** `(SEL)`*`objectNameSelector`*

  `Sets the selector to send for labeling the object.`

- - **`setProbeMap:`** `(id <ProbeMap>)`*`probeMap`*

  `Sets the probe map (i.e. list of fields) to display.`

- - **`setObjectList:`** `(id <List>)`*`objectList`*

  `Sets the list of objects to display.`

MultiVarProbeWidget

Name

`MultiVarProbeWidget` — A widget for displaying multiple objects across multiple fields.

Description

A widget for displaying multiple objects across multiple fields.

Protocols adopted by MultiVarProbeWidget

SwarmObject *(see page 211)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- - **setProbeMap:** (id <ProbeMap>)*probeMap*

  Sets the fields (probes) to show on the widget.

- - **setObjectList:** (id <List>)*objectList*

  Sets the objects to show on the widget.

- - **setObjectNameSelector:** (SEL)*objectNameSelector*

  Sets the method to use get the label for each object (vertical).

- - **setFieldLabelingFlag:** (BOOL)*labelingFlag*

  Determines if the fields (probes) are labeled (horizontal).

- - **setParent:** (id <Frame>)*parent*

Phase: Using

- - (void)**pack**
- - (void)**update**

ProbeDisplay

Name

`ProbeDisplay` — A class to display ProbeMaps

Description

A class which generates a GUI to a ProbeMap of probes applied to a given target object.

Protocols adopted by ProbeDisplay

SingleProbeDisplay *(see page 310)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- `-` **`setProbeMap:`** `(id <ProbeMap>)`*`probeMap`*

  `This is an optional create phase method - if no probeMap is specified the ProbeDisplay will ask the probedObject for a ProbeMap using the getProbeMap method described below... The default behaviour of this method will be to return the probeLibrary's copy of the probeMap for the class of the target object.`

ProbeDisplayManager

Name

`ProbeDisplayManager` — The ProbeDisplay manager.

Description

A (singleton) class whose instance is used to manage all the ProbeDisplays created by the user during a GUI run of the simulation.

Protocols adopted by ProbeDisplayManager

SwarmObject *(see page 211)*

CREATABLE *(see page 44)*

Methods

Phase: Using

- `- **setDropImmediatelyFlag:** (BOOL)`*`dropImmediateFlag`*
- `- (void)`**`update`**

  `This method will recursively send an update message to all the Probe Displays managed by the ProbeDisplayManager.`
- `- **dropProbeDisplaysFor:** `*`anObject`*

  `Remove and drop probe displays associated with a given object.`
- `- **removeProbeDisplay:** (id <CommonProbeDisplay>)`*`probeDisplay`*

  `Remove a probe display from management by the ProbeDisplayManager.`
- `- **addProbeDisplay:** (id <CommonProbeDisplay>)`*`probeDisplay`*

  `Add a probe display to be managed by the ProbeDisplayManager.`
- `- (id <CompleteProbeDisplay>)`**`createArchivedCompleteProbeDisplayFor:`** *`anObject`* **`variableName:`** `(const char *)`*`variableName`*
- `- (id <CompleteProbeDisplay>)`**`createCompleteProbeDisplayFor:`** *`anObject`*
- `- (id <ProbeDisplay>)`**`createArchivedDefaultProbeDisplayFor:`** *`anObject`* **`variableName:`** `(const char *)`*`variableName`*
- `- (id <ProbeDisplay>)`**`createDefaultProbeDisplayFor:`** *`anObject`*
- `- (id <ProbeDisplay>)`**`createArchivedProbeDisplayFor:`** *`anObject`* **`variableName:`** `(const char *)`*`variableName`*
- `- (id <ProbeDisplay>)`**`createProbeDisplayFor:`** *`anObject`*
- `- (BOOL)`**`getDropImmediatelyFlag`**

Macros

- **CREATE_ARCHIVED_COMPLETE_PROBE_DISPLAY**(*anObject*)

  This macro creates a complete probe display for the given object, to be saved by the window archiver

- **CREATE_ARCHIVED_PROBE_DISPLAY**(*anObject*)

  This macro creates the probe display for the given object, to be saved by the window archiver

- **CREATE_COMPLETE_PROBE_DISPLAY**(*anObject*)

  This macro creates a complete probe display for the given object

- **CREATE_PROBE_DISPLAY**(*anObject*)

  This macro creates a probe display for the given object

SimpleProbeDisplay

Name

SimpleProbeDisplay —

Description

Protocols adopted by SimpleProbeDisplay

SingleProbeDisplay *(see page 310)*

RETURNABLE *(see page 66)*

Methods

Phase: Creating

- - **setProbeMap:** (id <ProbeMap>)*probeMap*

SingleProbeDisplay

Name

`SingleProbeDisplay` — An abstract protocol underlying single-object probe displays.

Description

This protocol is common to CompleteProbeDisplay and ProbeDisplay.

Protocols adopted by SingleProbeDisplay

CommonProbeDisplay *(see page 298)*

Methods

Phase: Creating

- - **setProbedObject:** *anObject*

  This method must be called.

Phase: Using

- - **getProbedObject**

  Gets the probed object.

WindowGeometryRecordName

Name

`WindowGeometryRecordName` — Protocol for archiving window geometry.

Description

Classes that allow for window geometry archiving must conform this protocol.

Protocols adopted by WindowGeometryRecordName

None

Methods

Phase: Creating

- `-` **`setSaveSizeFlag:`** `(BOOL)`*`saveSizeFlag`*
- `-` **`setWindowGeometryRecordName:`** `(const char *)`*`windowGeometryRecordName`*

  This method is used to give an instance ProbeDisplay a name, which will used by the Archiver when recording its geometry information.

Macros

- **`SET_WINDOW_GEOMETRY_RECORD_NAME`**`(`*`theWidget`*`)`

  This macro uses the instance name of theWidget to set its name in the window geometry record.

General

Name

`simtoolsgui` — GUI-related features for simulation.

Functions

- `void` **`initSimtoolsGUI`**`(void)`

  `Initialize the library and create a ProbeDisplayManager.`

Globals

id <Symbol> Control

Type Symbols for ActionCache

id <Symbol> Probing

Type Symbols for ActionCache

id <Symbol> Spatial

Type Symbols for ActionCache

id <Symbol> InvalidActionType

Error Symbols for ActionCache

id <Symbol> ActionTypeNotImplemented

Error Symbols for ActionCache

id <ProbeDisplayManager> probeDisplayManager

Manager that keeps track of active probes to be updated

id <Symbol> ControlStateRunning

State Symbols for the ControlPanel.

id <Symbol> ControlStateStopped

State Symbols for the ControlPanel.

id <Symbol> ControlStateStepping

State Symbols for the ControlPanel.

id <Symbol> ControlStateNextTime

State Symbols for the ControlPanel.

id <Symbol> ControlStateQuit

State Symbols for the ControlPanel.

Gui Library

Overview

The GUI library is intended to be a toolkit-independent description Swarm-specific GUI widgets. The user specifies a specific back-end (such as Tk or Java AWT) at link time.

1. Dependencies

Following are the other header files imported by <gui.h>:

```
#import <objectbase.h>
```

2. Compatibility

- **1.0.5 -> 1.1.** This library was created with the Swarm 1.1 release, earlier version of Swarm used the tkobjc functions directly.

Documentation and Implementation Status

Revision History

2004-07-16 gui.h *schristley*

(GUI_*): Exclude tk macros for GNUstep.

2003-06-23 gui.h *pauljohn*

declare ZoomRaster method - (void)fillCenteredRectangleX0: (int)x0 Y0: (int)y0 X1: (int)x1 Y1: (int)y1 Color: (Color)color; In case users want to draw rectangles whose position does not change when zooming rasters.

2001-05-13 gui.h *mgd*

(Histogram): Name count: argument to setColors:count: and setLabels:count: "count".

2001-04-18 gui.h *mgd*

(NodeItem): New method -resetString:. (LinkItem): New method -setDirectedFlag:.

2001-03-29 gui.h *mgd*

(Pixmap): Remove drawX:Y:, it's in Drawer. (CanvasItem): Remove initiateMoveX:Y:, it's in CanvasAbstractItem. (NodeItem): Change argument name of setString to label.

2001-03-20 gui.h *mgd*

(Graph, Histogram): Make setTitle and setAxisLabelsX:Y: have object returns because there are analysis methods in creating phase.

2001-03-13 gui.h *mgd*

Add setSymbolSize: to GraphElement.

2001-03-12 gui.h *mgd*

Many changes to remove ugly object returns.

2001-03-03 gui.h *mgd*

(NodeItem): Move setString, setFont, and setX:Y: into create phase. (In the last case, note there is a moveX:Y: method.)

2000-09-20 gui.h *mgd*

(ArchivedGeometryWidget): Declare -updateSize.

2000-06-29 gui.h *mgd*

(ArchivedGeometryWidget): Remove return type on updateArchiver:.

2000-05-23 gui.h *mgd*

(Pixmap): Adopt drop.

2000-04-20 gui.h *mgd*

Note that window geometry record names must not have spaces.

2000-03-28 *mgd*

Swarmdocs 2.1.1 frozen.

2000-02-29 *mgd*

Swarmdocs 2.1 frozen.

2000-02-18 gui.h *mgd*

Remove extra +createBegin and -createEnd throughout. Move +create:setWindowGeometryRecordName: to ArchivedGeometryWidget. Remove from Raster and ZoomRaster.

1999-10-08 gui.h *mgd*

(ArchivedGeometryWidget): Add -setSaveSizeFlag:.

1999-09-14 gui.h *alex*

(Raster, ZoomRaster): Add factory +create:setWindowGeometryRecordName: method to protocols.

1999-07-20 gui.h *mgd*

(Form): Change Boolean: argument type to unsigned * (from BOOL *). (InputWidget): Likewise.

1999-06-05 gui.h *mgd*

(Colormap): New method unsetColor:. Thanks to Ken Cline.

1999-05-01 gui.h *mgd*

(Histogram): Change setNumBins: to setBinCount:. Change type of setActiveOutlierText:count: arguments to unsigned.

1999-03-20 gui.h *mgd*

(WindowGeometryRecord): Don't tag as CREATABLE. Add @class for CheckButton, Form, CanvasIteim, NodeItem, and Rectangle.

1999-02-26 gui.h *mgd*

Add CREATABLE tags for all non-abstract protocols.

1999-02-23 gui.h *mgd*

Merge internal protocols into advertised protocols.

1999-02-08 gui.h *mgd*

(_ArchivedGeometryWidget): Add archiver argument to updateArchiver:. Version 1.4.1.

1999-01-06 gui.h *mgd*

(_Histogram): Change argument to setNumBins: to unsigned. Add count argument to setColors and setLabels.

1998-11-18 gui.h *mgd*

(Widget, WindowGeometryRecord, GraphElement, CanvasAbstractItem): Adopt Create and Drop instead of SwarmObject. (_ClassDisplayLabel, _CompleteProbeDisplayLabel, _ClassDisplayHideButton, _MessageProbeEntry, _TextItem, _Circle): Add USING tag. (_ButtonPanel, _CompositeItem): Add CREATING tag.

1998-11-17 gui.h *mgd*

(_WindowGeometryRecord): Adopt Serialization protocol; don't redeclare methods.

1998-11-16 gui.h *mgd*

Remove creating phase +in:. Rename in: and out: to lispin: and lispout:.

1998-09-30 gui.h *mgd*

(_Histogram): -setNumBins:, -setLabels:, -setColors:, +createBegin:, createEnd: Split out setNumPoints:Labels:Colors: into different methods.

1998-09-28 gui.h *alex*

(ScheduleItem): Fixed incorrect documentation markup of `Description' entry for protocol.

1998-09-28 gui.h *mgd*

(_Canvas, _Frame): Move Frame's assertGeometry to Canvas as checkGeometry:.

1998-09-25 gui.h *mgd*

(_Canvas): Use addWidget:X:Y:centerFlag: and removeWidget:.

1998-09-24 gui.h *mgd*

(_ScheduleItem): Add trigger:X:Y:. (_CompleteProbeDisplayLabel): Remove setProbeDisplayManager: and rename setProbeDisplay: to setTargetWidget:.

1998-09-23 gui.h *mgd*

(_ScheduleItem): Add at:owner:widget:x:y:.

1998-09-22 gui.h *mgd*

(Circle): Make argument to setRadius: unsigned. Provide Circle, Line, ScheduleItem, and TextItem class objects. (_ScheduleItem, ScheduleItem): New protocols. (_TextItem): Add +createBegin: and setCenterFlag:.

1998-09-17 gui.h *mgd*

(Pixmap): New method setDecorationsFlag:.

1998-09-03 gui.h *mgd*

(Widget): Declare -getParent and -getTopLevel. Revert removal of -getWindowGeometry and -setWindowGeometry:, but don't document them. (Histogram): Declare setXaxisMin:max:step:precision:.

1998-08-19 gui.h *mgd*

(Pixmap): Add setDirectory:.

1998-08-18 gui.h *mgd*

(Pixmap): Revert create:widget: and create:file: addition and use setFile: and setWidget: instead.

1998-08-06 gui.h *mgd*

(_Pixmap): Add create:widget: and create:file:. Move setRaster: to using phase (still need to make implementation synchronize). Add save:.

1998-07-22 gui.h *mgd*

(Graph): Add setRangesXMin:Max:.

1998-07-21 gui.h *mgd*

(Graph): Arguments to setScaleModeX:Y: are now boolean, not integers.

1998-07-15 gui.h *mgd*

(_VarProbeEntry): Replace setProbeType: with setVarProbe:. Add getVarProbe.

1998-07-07 gui.h *mgd*

(_GraphElement): New method -setWidth:.

1998-07-01 gui.h *mgd*

(Widget): Rename setPositionX:Y: to setX:Y:. Add new methods -getX and -getY. (_WindowGeometryRecord): Remove getWindowGeometry and setWindowGeometry:. Remove -describe:. Add setX:Y:, setWidth:Height:, getSizeFlag, getPositionFlag, getWidth, getHeight, getX, and getY.

1998-06-17 Makefile.am *mgd*

Include from refbook/ instead of src/.

1998-06-17 guimeta.sgml *alex*

Removed redundant text from ABSTRACT.

1998-06-17 gui00.sgml *alex*

Added objectbase.h to list of dependencies.

1998-06-15 Makefile.am *mgd*

(MODULE): New variable. Include Makefile.rules from src. Remove everything else.

1998-06-14 gui.h *mgd*

Make separate ifdefs for GUI macros to work with the protocol script. Use #if 0 for awt.

1998-06-12 gui00.sgml, guicont.sgml *mgd*

Update IDs to SWARM.module.SGML.type.

1998-06-06 gui.ent *mgd*

Use public identifiers.

1998-06-05 Makefile.am *mgd*

(swarm_ChangeLog): Add.

1998-06-05 gui.h *alex*

Fixed (#if 1) around the tk macros - erroneosly set to (#if 0).

1998-06-03 gui.h *mgd*

Update documentation tags. Use #if 0 for macros to avoid multiple definitions in protocol.el.

1998-05-27 gui.ent.in *mgd*

Fix typo in revhistory.

1998-05-23 Makefile.am *mgd*

New file.

1998-05-23 gui.ent.in *mgd*

New file.

1998-05-23 gui.ent *mgd*

Removed.

1998-05-22 *mgd*

Begin revision log.

1998-05-22 gui.h *mgd*

(_ArchivedGeometryWidget): Declare -loadWindowGeometryRecord, -registerAndLoad, +createBegin:, -updateArchiver, -getDestroyedFlag, and -drop.

1998-05-13 gui.h *mgd*

InputWidget: new protocol. Make CheckButton and Entry inherit from InputWidget, not _InputWidget.

1998-05-06 gui.h *mgd*

(_Widget, _WindowGeometryRecord, _ArchivedGeometryWidget, _Frame, _Canvas, _ProbeCanvas, _GraphElement, _Graph, _Histogram, _Label, _ClassDisplayLabel, _VarProbeLabel, _CompleteProbeDisplayLabel, _Button, _ClassDisplayHideButton, _SimpleProbeDisplayHideButton, _SuperButton, _InputWidget, _Entry, _MessageProbeEntry, _VarProbeEntry, _ButtonPanel, _Form, _Colormap, _Drawer, _Raster, _Pixmap, _CanvasAbstractItem, _CanvasItem, _CompositeItem, _NodeItem, _LinkItem, _OvalNodeItem, _RectangleNodeItem, _TextItem, _Circle, _Rectangle, _Line): Add phase tags. (_Widget, _WindowGeometryRecord, _ArchivedGeometryWidget, _Frame, _ProbeCanvas, _Raster, _ZoomRaster, _Pixmap, _CanvasAbstractItem, _NodeItem, _LinkItem): Reorder definitions for above. Move all //S and //D comments in internal protocols to external protocols. (_GraphElement): Declare -setOwnerGraph:, -createEnd, and +createOwnerGraph:. (_Raster): Declare +createBegin and -createEnd. (_Graph, _Histogram, _Label, _ClassDisplayLabel, _Button, _SimpleProbeDisplayHideButton, _InputWidget, _Form, _Colormap, _ZoomRaster): Declare -createEnd.

1998-04-23 gui.h *mgd*

Add documentation tags to all protocols. Bring in sync with tkobjc changes.

1998-04-17 gui.h *mgd*

(Raster): Add ellipseX0:Y0:X1:Y1:Width:Color, lineX0:Y0:X1:Y1:Width:Color:, and rectangleX0:Y0:X1:Y1:Width:Color:. New protocols ButtonPanel, Form, CheckButton, OvalNodeItem, RectangleNodeItem, CanvasItem, TextItem, Circle, Rectangle, and Line. Split all protocols into `New' and `Usage' parts.

1998-04-13 gui.h *mgd*

Colormap and Pixmap protocols now are drived from Create protocol. (Thanks to Ken Cline.) Add "@class Raster". (Thanks to Pietro Terna.) (Pixmap): Add getWidth and getHeight methods to protocol.

1998-04-10 gui.h *mgd*

Create class object for Raster. (Drawer): Use it to declare drawX:Y: method spec. (Raster): Declare draw:X:Y: method spec. (Pixmap): New protocol. Create class object for Pixmap.

1998-02-27 gui.h *mgd*

(Entry): Add linkVariableInt: to protocol (used in testIPD). (Graph) Add setRangesYMin:Max:. (GraphElement): Add setColor:, setDashes:, setSymbol:, and resetData. (Entry): Add linkVariableDouble: and linkVariableBoolean:.

1998-02-24 gui.h *alex*

(GUI_UPDATE_IDLE_TASKS): Added to call the new parametrized (tkobjc_updateIdleTasks) for the Tk #ifndef and a stub added for the Java version.

1998-02-20 gui.h *mgd*

Change javaobjc to awtobjc throughout. Use initAWTObjc instead of initJavaObjc.

1998-02-18 gui.h *mgd*

Add AWT placeholders for GUI_* macros. Change GUI_INIT to take a single argument, the Arguments object. Switch from #ifdef _TK_ to #ifndef USE_JAVA. (Widget): Add setPositionX:Y:.

1998-01-27 gui.h *mgd*

Remove GUI_ prefixes from GUI_Color and GUI_PixelValue. Notes about XPixmap and XDrawer. Remove GUI_ButtonRight. Add ButtonLeft, ButtonMiddle, and ButtonRight defines. Use #import for tkobjc/common.h and tkobjc/global.h. Rename class XColormap to Colormap and BLTGraph to Graph. Add @class Entry. Add getParent for Widget. Add fillRectangeX0:Y0:X1:Y1:Color: to Raster. Add moveX:Y: to CompositeItem.

1998-01-26 gui.h *mgd*

Remove some USING tags.

1998-01-25 gui.h *mgd*

(Widget): Remove setBorderWidth. Add packToRight. (Frame): Change enableRelief to setRelief:. Add setBorderWidth to Frame. (Label): Remove anchorEast, anchorWest, colorBlue. (ClassDisplayLabel, VarProbeLabel, ClassDisplayHideButton): Added. (SuperButton): Add createEnd, setOwner, setUser. Add ClassDisplayLabel, VarProbeLabel.

1998-01-25 gui.h *mgd*

New file.

ArchivedGeometryWidget

Name

`ArchivedGeometryWidget` — Base class for widgets that archive geometry.

Description

Subclasses of this class inherit the ability to archive their window geometry. This class also provides an interface to destroy notification.

Protocols adopted by ArchivedGeometryWidget

Widget *(see page 354)*

Methods

Phase: Creating

- `- **updateSize**`
- `- **registerAndLoad**`
- `- **loadWindowGeometryRecord**`
- `- **setSaveSizeFlag:** (BOOL)`*`saveSizeFlag`*

  Determines whether or not size is saved in addition to position.
- `- **setWindowGeometryRecordName:** (const char *)`*`recordName`*

  Called to set a name for archiving. recordName must not have spaces.
- `+ **create:** (id <Zone>)`*`aZone`* `**setWindowGeometryRecordName:** (const char *)`*`name`*

Phase: Using

- `- (void)**updateArchiver:** (id <Archiver>)`*`archiver`*

Button

Name

`Button` — A button widget.

Description

A button widget that, when pressed, sends a method to a target object.

Protocols adopted by Button

Widget *(see page 354)*

CREATABLE *(see page 44)*

Methods

Phase: Using

- `- (void)` **`setButtonTarget:`** *`target`* **`method:`** `(SEL)`*`method`*

  `Set the target and selector for button.`

- `-` **`setText:`** `(const char *)`*`text`*

  `Set the text for button.`

ButtonPanel

Name

`ButtonPanel` — Several buttons bound together in one frame.

Description

Several buttons bound together in one frame.

Protocols adopted by ButtonPanel

Frame *(see page 333)*

CREATABLE *(see page 44)*

Methods

Phase: Using

- `- (void)`**`addButtonName:`** `(const char *)`*`name`* **`method:`** `(SEL)`*`sel`*

  `Create a new button, and set the method, using the default target.`

- `- (void)`**`addButtonName:`** `(const char *)`*`name`* **`target:`** *`target`* **`method:`** `(SEL)`*`sel`*

  `Create a new button, and set both a target and method.`

- `- (void)`**`setButtonTarget:`** *`target`*

  `Set a default target for use with addButtonName:method:.`

Canvas

Name

`Canvas` — An interface to Tk canvas semantics.

Description

The Canvas widget allows display of a diverse range of graphical objects.

Protocols adopted by Canvas

ArchivedGeometryWidget *(see page 321)*

CREATABLE *(see page 44)*

Methods

Phase: Using

- - **checkGeometry:** *frame*

  Make sure the that the geometry is `reasonable'.

- - **removeWidget:** *widget*

  Remove a widget from the canvas.

- - **addWidget:** *widget* **X:** (int)*x* **Y:** (int)*y* **centerFlag:** (BOOL)*centerFlag*

  Position a widget inside the canvas

CanvasAbstractItem

Name

`CanvasAbstractItem` — An abstract class for items on a Canvas.

Description

CanvasAbstractItem is the root class of all items drawn on a Canvas.

Protocols adopted by CanvasAbstractItem

Create *(see page 46)*

Drop *(see page 54)*

Methods

Phase: Creating

- `- setCanvas: (id <Canvas>)canvas`

  Designates the id of the Canvas in which this item resides.

- `- (void)createBindings`

  Method to be implemented by subclass.

- `- (void)createItem`

  Method to be implemented by subclass.

Phase: Using

- `- (id <Canvas>)getCanvas`

  Return the canvas.

- `- (void)initiateMoveX: (long)deltaX Y: (long)deltaY`

  Method to be implemented by subclass.

- `- (void)clicked`

  Called when a mouse click occurs.

- `- (void)setPostMoveSel: (SEL)sel`

  Sets the message that will dictate what happens after the item is moved.

- `- (void)setMoveSel: (SEL)sel`

  Sets the message that will effect the motion of the item on the canvas.

- `- (void)setClickSel: (SEL)sel`

  Sets the message that will be sent upon a click on this item.

- `- (void)setTargetId: target`

Designates the object to which this item refers.

CanvasItem

Name

CanvasItem — An abstract superclass for simple Canvas items.

Description

An abstract superclass for non-composite Canvas items.

Protocols adopted by CanvasItem

CanvasAbstractItem *(see page 325)*

CREATABLE *(see page 44)*

Methods

None

CheckButton

Name

CheckButton — A check box on/off selection widget.

Description

A check box on/off selection widget.

Protocols adopted by CheckButton

InputWidget *(see page 338)*

CREATABLE *(see page 44)*

Methods

Phase: Using

- - **setBoolValue:** (BOOL)*v*

  Turn the widget value and check button on or off.

- - (BOOL)**getBoolValue**

  Get on/off status.

Circle

Name

`Circle` — A CanvasItem that displays a circle.

Description

A CanvasItem that displays a circle.

Protocols adopted by Circle

CanvasItem *(see page 326)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- - **`setRadius:`** `(unsigned)`*`r`*

  `Set the radius of the circle.`

- - **`setX:`** `(int)`*`x`* **`Y:`** `(int)`*`y`*

  `Set the x, y coordinates for the center of the circle.`

ClassDisplayHideButton

Name

`ClassDisplayHideButton` — The hide button used by a CompleteProbeDisplay.

Description

A button that handles the dismissal of class widgets on a ClassDisplayWidget (for CompleteProbeDisplay).

Protocols adopted by ClassDisplayHideButton

Button *(see page 322)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- - **`setOwner:`** *`owner`*
- - **`setUser:`** *`user`*
- - **`setSubWidget:`** *`subWidget`*

ClassDisplayLabel

Name

`ClassDisplayLabel` — A label for displaying class names.

Description

This widget is used internally by ClassDisplayWidget.

Protocols adopted by ClassDisplayLabel

Label *(see page 339)*

CREATABLE *(see page 44)*

Methods

None

Colormap

Name

`Colormap` — An class for creating a color palette for use with a Raster.

Description

Mechanism used to map numbers in the range [0, 255] to colour names. Create an XColormap, allocate colours in it, and pass it to a Raster widget for drawing.

Protocols adopted by Colormap

Create *(see page 46)*

CREATABLE *(see page 44)*

Methods

Phase: Using

- `- (void)`**`unsetColor:`** `(Color)`*`c`*

  Remove color at index `c' from the color map.

- `- (BOOL)`**`setColor:`** `(Color)`*`c`* **`ToGrey:`** `(double)`*`g`*

  Add a color of a certain level of grey.

- `- (BOOL)`**`setColor:`** `(Color)`*`c`* **`ToName:`** `(const char *)`*`colorName`*

  Add color index `c' looking up the color name in the color database.

- `- (BOOL)`**`setColor:`** `(Color)`*`c`* **`ToRed:`** `(double)`*`r`* **`Green:`** `(double)`*`g`* **`Blue:`** `(double)`*`b`*

  Add color index `c' to the color map, using a certain percent of red, green, and blue.

- `- (PixelValue)`**`white`**

  The pixel value for white.

- `- (PixelValue)`**`black`**

  The pixel value for black.

- `- (PixelValue *)`**`map`**

  The current palette, per color-index.

CompleteProbeDisplayLabel

Name

`CompleteProbeDisplayLabel` — A class label used in a SimpleProbeDisplay.

Description

This widget is used internally by SimpleProbeDisplay. It is used to set up the mouse bindings to get a CompleteProbeDisplay, and to set up drag and drop.

Protocols adopted by CompleteProbeDisplayLabel

Label *(see page 339)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- `- **setProbedObject:** probedObject`

  `Sets the object that the probe display represents.`

- `- **setTargetWidget:** targetWidget`

  `Sets target widget which to report (e.g. probeDisplay).`

CompositeItem

Name

`CompositeItem` — A CanvasItem with several pieces.

Description

A CompositeItem is a CanvasItem that consists of several pieces. CompositeItem is an abstract superclass.

Protocols adopted by CompositeItem

CanvasAbstractItem *(see page 325)*

Methods

Phase: Using

- `- (void)**moveX:** (long)deltaX **Y:** (long)deltaY`

  `Must be implemented by subclass.`

Drawer

Name

`Drawer` — The interface used by Raster to draw an arbitrary object.

Description

The interface used by Raster to draw an arbitrary object. Pixmap uses this.

Protocols adopted by Drawer

None

Methods

Phase: Using

- `- (void)`**`drawX:`** `(int)`*`x`* **`Y:`** `(int)`*`y`*

Entry

Name

`Entry` — Handles text-field input.

Description

Handles text-field input.

Protocols adopted by Entry

InputWidget *(see page 338)*

CREATABLE *(see page 44)*

Methods

Phase: Using

- `-` **`setHeight:`** `(unsigned)`*`h`*

  ```
  This method aborts
  ```

Form

Name

`Form` — A set of Entry widgets bound together in one frame.

Description

A set of Entry widgets bound together in one frame.

Protocols adopted by Form

Widget *(see page 354)*

CREATABLE *(see page 44)*

Methods

Phase: Using

- `- (void)`**`addLineName:`** `(const char *)`*`n`* **`Double:`** `(double *)`*`p`*

  `Add an Entry to get a double.`

- `- (void)`**`addLineName:`** `(const char *)`*`n`* **`Int:`** `(int *)`*`p`*

  `Add an Entry to get an integer.`

- `- (void)`**`addLineName:`** `(const char *)`*`n`* **`Boolean:`** `(unsigned *)`*`p`*

  `Add a boolean CheckButton widget.`

- `- (void)`**`setEntryWidth:`** `(unsigned)`*`ew`*

  `The width of all the Entry widgets.`

Frame

Name

`Frame` — Encapsulation of toplevels.

Description

Frames are boxes other widgets fit in. They correspond to the Tk "frame" and "toplevel" widgets. Frames can be new windows, or subwindows in an existing window. You only need to create frames yourself if building complicated composite widgets: by default, a frame will be built automatically for widgets without parents.

Protocols adopted by Frame

ArchivedGeometryWidget *(see page 321)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- `- ` **`setReliefFlag:`** `(BOOL)`*`reliefFlag`*

  `Determines whether or not a frame has a border.`

- `- ` **`setBorderWidth:`** `(int)`*`width`*

  `Determines the width of the border, if any.`

Phase: Using

- `- (void)` **`deiconify`**

  `Deiconify the frame.`

- `- (void)` **`withdraw`**

  `Take the frame off screen.`

Graph

Name

`Graph` — A time series graph tool.

Description

A time series graph tool, based on BLT's graph widget. Graph currently implements just a basic graph with multiple datasets, but should eventually support scaling and scrolling. For each Graph you create one or many GraphElements, one per dataset to plot. GraphElements can be configured for appearance, and data can be added to the element to draw.

Protocols adopted by Graph

ArchivedGeometryWidget *(see page 321)*

CREATABLE *(see page 44)*

Methods

Phase: Using

- `- (void)`**`setRangesXMin:`** `(double)`*`minx`* **`Max:`** `(double)`*`maxx`* **`YMin:`** `(double)`*`miny`* **`Max:`** `(double)`*`maxy`*

  Sets the ranges for the graph. Turns off autoscaling.

- `- (void)`**`setRangesYMin:`** `(double)`*`miny`* **`Max:`** `(double)`*`maxy`*

  Sets the Y ranges for the graph. Turns off autoscaling.

- `- (void)`**`setRangesXMin:`** `(double)`*`minx`* **`Max:`** `(double)`*`maxx`*

  Sets the X ranges for the graph. Turns off autoscaling.

- `- (void)`**`setScaleModeX:`** `(BOOL)`*`xs`* **`Y:`** `(BOOL)`*`ys`*

  Whether to autoscale every timestep or instead to jump scale.

- `- (id <GraphElement>)`**`createElement`**

  Builds a new GraphElement to plot data with.

- `-` **`setAxisLabelsX:`** `(const char *)`*`xl`* **`Y:`** `(const char *)`*`yl`*

  Set the axis labels for the graph.

- `-` **`setTitle:`** `(const char *)`*`title`*

  Set the title for the graph.

GraphElement

Name

`GraphElement` — Contains a set of related data for display.

Description

A GraphElement accumulates a related set of data for display, including attributes for the set.

Protocols adopted by GraphElement

Create *(see page 46)*

Drop *(see page 54)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- + **createOwnerGraph:** *ownerGraph*
- - **setOwnerGraph:** *ownerGraph*

Phase: Using

- - **setWidth:** (unsigned)*w*

  Set the line width of the element.
- - (void)**resetData**

  Clear the data to be displayed.
- - (void)**addX:** (double)*x* **Y:** (double)*y*

  Add a new data point.
- - (void)**setSymbolSize:** (unsigned)*size*

  Set the symbol size in pixels.
- - (void)**setSymbol:** (const char *)*symbol*

  Set the symbol for the element.
- - (void)**setDashes:** (int)*dashesVal*

  Set the dash style, 0 means solid.
- - (void)**setColor:** (const char *)*colorName*

  Set the color for the element.
- - (void)**setLabel:** (const char *)*label*

  Set the label for the element.

Histogram

Name

`Histogram` — Histogram display tool.

Description

In Tk, this is based on BLT's barchart. The number of bins is fixed at creation time, then the user hands the Histogram an array of datapoints (double or int) to display (or optionally an array of datapoints and locations where the bars should be drawn (specified as doubles).

Protocols adopted by Histogram

ArchivedGeometryWidget *(see page 321)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- - **setBinCount:** (unsigned)*n*

  Set the number of bins to use (bars to draw).

Phase: Using

- - (void)**setupActiveItemInfo**
- - (void)**setupActiveOutlierMarker**
- - (void)**setupZoomStack**
- - (void)**drawHistogramWithDouble:** (double *)*points*

  Draw the (double) data in the histogram.

- - (void)**drawHistogramWithDouble:** (double *)*points* **atLocations:** (double *)*locations*

  Draw the (double) data in the histogram at particular offsets.

- - (void)**drawHistogramWithInt:** (int *)*points*

  Draw the (integer) data in the histogram.

- - (void)**drawHistogramWithInt:** (int *)*points* **atLocations:** (double *)*locations*

  Draw the (integer) data in the histogram at particular offsets.

- - (void)**hideLegend**

  Hide the legend on the histogram.

- - (void)**setActiveOutlierText:** (unsigned)*outliers* **count:** (unsigned)*count*

  Set the text that describes a specified number of outliers.

- - **setAxisLabelsX:** (const char *)*xl* **Y:** (const char *)*yl*

  Set the axis labels.

- - (void)**setXaxisMin:** (double)*min* **max:** (double)*max* **step:** (double)*step*

  Set the X range and step size for the histogram. Display three significant figures for the major-tick labels.

- - (void)**setXaxisMin:** (double)*min* **max:** (double)*max* **step:** (double)*step* **precision:** (unsigned)*precision*

  Set the X range, step size, and number of major-tick-label significant figures for the histogram.

- - (void)**setBarWidth:** (double)*step*

  Set the width of the bars.

- - **setTitle:** (const char *)*title*

  Set the title of the histogram.

- - (void)**setLabels:** (const char * const *)*l* **count:** (unsigned)*count*

  Set labels for the histogram bars. If not set, they remain blank. Labels are arrays of strings, one per bin/bar.

- - (void)**setColors:** (const char * const *)*c* **count:** (unsigned)*count*

  Set colors for the histogram bars. If not set, all are blue. Colors are arrays of strings (one per bin/bar) of color names.

InputWidget

Name

`InputWidget` — Abstract superclass for widgets that take input.

Description

InputWidgets get their input in one of two ways: by being readable, or by being linked to a C variable.

Protocols adopted by InputWidget

Widget *(see page 354)*

Methods

Phase: Using

- `- (void)`**`linkVariableBoolean:`** `(unsigned *)`*`p`*

  Attach the widget value to a boolean.

- `- (void)`**`linkVariableDouble:`** `(double *)`*`p`*

  Attach the widget value to a double.

- `- (void)`**`linkVariableInt:`** `(int *)`*`p`*

  Attach the widget value to an integer.

- `- (void)`**`setValue:`** `(const char *)`*`v`*

  Set the string value of the widget. This must be implemented by a subclass.

- `- (const char *)`**`getValue`**

  Get the string value of the widget.

Label

Name

`Label` — A widget with text.

Description

A widget with text.

Protocols adopted by Label

Widget *(see page 354)*

CREATABLE *(see page 44)*

Methods

Phase: Using

- `- **setText:** (const char *)`*text*

  `Set the text to write in the label.`

Line

Name

`Line` — A CanvasItem that displays a line.

Description

A CanvasItem that displays a line.

Protocols adopted by Line

CanvasItem *(see page 326)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- `- **setTX:** (int)`*tx* `**TY:** (int)`*ty* `**LX:** (int)`*lx* `**LY:** (int)`*ly*

  `Set the end points of the line.`

LinkItem

Name

`LinkItem` — A canvas item for displaying a link between two nodes.

Description

A CompositeCanvasItem for displaying a link between two NodeItems.

Protocols adopted by LinkItem

CompositeItem *(see page 330)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- - **setDirectedFlag:** (BOOL)*directedFlag*

  For disabling directed link items.

- - **setTo:** (id <NodeItem>)*to*

  Designate the node that will be the destination of the link.

- - **setFrom:** (id <NodeItem>)*from*

  Designate the node that will be the source of the link.

Phase: Using

- - (void)**update**

  Redraw the link (especially due to the motion of nodes).

- - (void)**setColor:** (const char *)*aColor*

  Set the color of the link.

MessageProbeEntry

Name

`MessageProbeEntry` — A widget for arguments to a message probe.

Description

An Entry widget for MessageProbe arguments.

Protocols adopted by MessageProbeEntry

Entry *(see page 331)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- - **setArg:** (int)*arg*

  Indicates the argument number.

- - **setIdFlag:** (BOOL)*idFlag*

  Indicates whether the type of this entry is an id.

NodeItem

Name

`NodeItem` — A class for displaying a node on a Canvas.

Description

A class for displaying a node on a Canvas. A NodeItem has a position, a font, color, border color and width.

Protocols adopted by NodeItem

CompositeItem *(see page 330)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- - **setX:** (int)*x* **Y:** (int)*y*

  Set the position of the node.

- - **setFont:** (const char *)*font*

  Set the font with which to draw the label.

- - **setString:** (const char *)*label*

  Set the label to put on the node.

Phase: Using

- - (void)**createPaddedText**

  Create the space for the text for the node.

- - (void)**createText**

  Create the text for the node.

- - (void)**setBorderWidth:** (int)*aVal*

  Set the width of the border.

- - (void)**setBorderColor:** (const char *)*aColor*

  Set the border color of the node.

- - (void)**setColor:** (const char *)*aColor*

  Set the color of the node.

- - (int)**getY**

  Get the y position of the node on the canvas.

- - (int)**getX**

Get the x position of the node on the canvas.

- - (void)**resetString:** (const char *)*string*

  Change the label on the string after the node is created.

OvalNodeItem

Name

`OvalNodeItem` — A circular NodeItem.

Description

A NodeItem with a circular appearance.

Protocols adopted by OvalNodeItem

NodeItem *(see page 342)*

CREATABLE *(see page 44)*

Methods

None

Pixmap

Name

`Pixmap` — A class for drawing color bitmaps on a Raster.

Description

A class for drawing color bitmaps on a Raster. The bitmaps are stored in the Portable Network Graphics format.

Protocols adopted by Pixmap

Drawer *(see page 331)*

Create *(see page 46)*

Drop *(see page 54)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- `- **setDecorationsFlag:** (BOOL)`*`decorationsFlag`*

  Specify whether or not window manager decorations for a widget should be included.

- `- **setWidget:** (id <Widget>)`*`widget`*

  Create a pixmap from a widget, or from the root window if widget is nil.

- `- **setDirectory:** (const char *)`*`path`*

  Specify the directory to find the PNG file.

- `- **setFile:** (const char *)`*`filename`*

  Create a pixmap from a PNG file.

Phase: Using

- `- (void)**save:** (const char *)`*`filename`*

  Save the pixmap to a file.

- `- (unsigned)**getHeight**`

  Get the height of the bitmap in pixels.

- `- (unsigned)**getWidth**`

  Get the width of the bitmap in pixels.

- `- (void)**setRaster:** (id <Raster>)`*`raster`*

Set the raster that the pixmap will be shown on. It's used to augment raster the color palette as necessary.

ProbeCanvas

Name

ProbeCanvas — A canvas type for probe displays.

Description

ProbeCanvas is a Canvas that implements the general appearance and interface of a probe display.

Protocols adopted by ProbeCanvas

Canvas *(see page 324)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- - **setHorizontalScrollbarFlag:** (BOOL)*horizontalScrolbarFlag*

  Indicates the presence or absence of a horizontal scroll bar.

Raster

Name

`Raster` — A two dimension color display class.

Description

2 dimensional, colour pixel images. Raster is based on a Tk frame widget with our own code for fast display of images. You can draw coloured dots on a Raster, or generic Drawers. Raster widgets are double buffered - the pixels you draw are not actually put on the screen until drawSelf is called. In addition, Rasters handle mouse clicks.

Protocols adopted by Raster

ArchivedGeometryWidget *(see page 321)*

CREATABLE *(see page 44)*

Methods

Phase: Using

- `- (void)` **`erase`**

  Erase the raster.

- `- (void)` **`draw:`** `(id <Drawer>)`*`drawer`* **`X:`** `(int)`*`x`* **`Y:`** `(int)`*`y`*

  Draw an object at a given position.

- `- (void)` **`rectangleX0:`** `(int)`*`x0`* **`Y0:`** `(int)`*`y0`* **`X1:`** `(int)`*`x1`* **`Y1:`** `(int)`*`y1`* **`Width:`** `(unsigned)`*`penWidth`* **`Color:`** `(Color)`*`c`*

  Draw a rectangle of given geometry, pen width, and color.

- `- (void)` **`lineX0:`** `(int)`*`x0`* **`Y0:`** `(int)`*`y0`* **`X1:`** `(int)`*`x1`* **`Y1:`** `(int)`*`y1`* **`Width:`** `(unsigned)`*`penWidth`* **`Color:`** `(Color)`*`c`*

  Draw a line of given geometry, pen width, and color.

- `- (void)` **`ellipseX0:`** `(int)`*`x0`* **`Y0:`** `(int)`*`y0`* **`X1:`** `(int)`*`x1`* **`Y1:`** `(int)`*`y1`* **`Width:`** `(unsigned)`*`penWidth`* **`Color:`** `(Color)`*`c`*

  Draw an ellipse of given geometry, pen width, and color.

- `- (void)` **`fillRectangleX0:`** `(int)`*`x0`* **`Y0:`** `(int)`*`y0`* **`X1:`** `(int)`*`x1`* **`Y1:`** `(int)`*`y1`* **`Color:`** `(Color)`*`color`*

  Fill a rectangle of given geometry and color.

- `- (void)` **`setButton:`** `(int)`*`n`* **`Client:`** *`c`* **`Message:`** `(SEL)`*`sel`*

  Configure at mouse button to send a message to a given client object.

- `- (void)` **`drawSelf`**

  Draw the raster to the display.

- - (void)**drawPointX:** (int)*x* **Y:** (int)*y* **Color:** (Color)*c*

  Draw a point at the given coordinates with the given color.

- - **setColormap:** (id <Colormap>)*c*

  Set the palette for this raster.

Rectangle

Name

Rectangle — A CanvasItem that displays a rectangle.

Description

A CanvasItem that displays a rectangle.

Protocols adopted by Rectangle

CanvasItem *(see page 326)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- - **setTX:** (int)*tx* **TY:** (int)*ty* **LX:** (int)*lx* **LY:** (int)*ly*

  Set the diagonal corner coordinates of the rectangle.

RectangleNodeItem

Name

`RectangleNodeItem` — A rectangular NodeItem.

Description

A rectangular NodeItem.

Protocols adopted by RectangleNodeItem

NodeItem *(see page 342)*

CREATABLE *(see page 44)*

Methods

None

ScheduleItem

Name

ScheduleItem — A canvas item for displaying the time structure of a schedule.

Description

A CompositeCanvasItem for displaying the time structure of a schedule.

Protocols adopted by ScheduleItem

CompositeItem *(see page 330)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- - **setX:** (int)*x* **Y:** (int)*y*

  Position the item on the canvas.

- - **setStep:** (unsigned)*step*

  Set the horizontal spacing of a time step.

- - **setSchedule:** *schedule*

  Set the schedule to be inspected.

Phase: Using

- - (void)**trigger:** *widget* **X:** (int)*x* **Y:** (int)*y*

  Send visual message indicator from browser to some target.

- - (void)**at:** (timeval_t)*tval* **owner:** *owner* **widget:** *widget* **x:** (int)*sourceX* **y:** (int)*sourceY*

  Record the screen coordinates associated with a scheduling event.

- - (void)**update**

  Redraw widget with current values from Schedule.

SimpleProbeDisplayHideButton

Name

`SimpleProbeDisplayHideButton` — The hide button used by a SimpleProbeDisplay.

Description

A button that handles the dismissal of a SimpleProbeDisplay.

Protocols adopted by SimpleProbeDisplayHideButton

Button *(see page 322)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- `-` **`setProbeDisplay:`** *`probeDisplay`*

  `The probe display in use.`

SuperButton

Name

`SuperButton` — Request superclass in ClassDisplayWidget.

Description

A button used by ClassDisplayWidget to ask for superclass.

Protocols adopted by SuperButton

Button *(see page 322)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- `-` **`setUser:`** *`user`*
- `-` **`setOwner:`** `(id <Widget>)` *`owner`*
- `-` **`setSuperWidget:`** `(id <Widget>)` *`superWidget`*

TextItem

Name

`TextItem` — A CanvasItem that displays text.

Description

A CanvasItem that displays text.

Protocols adopted by TextItem

CanvasItem *(see page 326)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- - **setCenterFlag:** (BOOL)*centerFlag*

  Determine whether text is centered or not.

- - **setFont:** (const char *)*font*

  Set the font with which to display the text.

- - **setText:** (const char *)*text*

  Set the text to display.

- - **setX:** (int)*x* **Y:** (int)*y*

  Set the coordinate for the center of the text.

VarProbeEntry

Name

`VarProbeEntry` — A widget for variable probes.

Description

An Entry widget for VarProbes.

Protocols adopted by VarProbeEntry

Entry *(see page 331)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- - **setVarProbe:** *varProbe*

  Set the variable probe associated with this widget.

- - **setOwner:** *owner*

  Indicate the object that is using this widget.

- - **setInteractiveFlag:** (BOOL)*interactiveFlag*

  Indicates whether the entry is editable or not.

Phase: Using

- - **getVarProbe**

VarProbeLabel

Name

`VarProbeLabel` — A label for displaying variable names.

Description

This widget is used internally by VarProbeWidget.

Protocols adopted by VarProbeLabel

Label *(see page 339)*

CREATABLE *(see page 44)*

Methods

None

Widget

Name

`Widget` — Widget base class.

Description

All graphical widgets inherit from the Widget base class. Widget defines most of the behaviour needed: Widgets are created by the user with a particular parent, and then "pack"ed in order to draw them on the screen. All widgets have three essential things: a widget name used when running Tcl code, an Objective C name when sending messages from Tcl to those objects, and a parent.

Protocols adopted by Widget

Create *(see page 46)*

Drop *(see page 54)*

Methods

Phase: Creating

- `- ` **`setWidgetNameFromParentName:`** `(const char *)`*`parentWidgetName`*

  Set the widget name using a hypothetical parent name.

- `- ` **`setWidgetNameFromParent:`** `(id <Widget>)`*`parent`*

  Set the widget name using the parent as context.

- `- (const char *)` **`makeWidgetNameFor:`** *`widget`*

  Compute the widget name for a component widget.

- `- ` **`setParent:`** *`parent`*

  Set the containing window of the widget.

- `+ ` **`createParent:`** *`parent`*

  When a widget is created it needs to be given a parent. The parent widget will be the widget's containing window. If no parent is given (ie, a parent of nil), then a toplevel Frame will be allocated automatically

Phase: Using

- `- (BOOL)` **`getDestroyedFlag`**
- `- (void)` **`disableDestroyNotification`**

  Prevent calling the destroy notification method.

- `- (void)` **`enableDestroyNotification:`** *`notificationTarget`* **`notificationMethod:`** `(SEL)`*`destroyNotificationMethod`*

  Call a method if we are destroyed.

- - (void)**setWindowGeometry:** (const char *)*s*
- - (const char *)**getWindowGeometry**
- - (int)**getY**

  Get the Y position of the widget.
- - (int)**getX**

  Get the X position of the widget.
- - (unsigned)**getWidth**

  Get the widget the widget.
- - (unsigned)**getHeight**

  Get the height of the widget.
- - (const char *)**getWidgetName**

  Get the widget name.
- - **getTopLevel**

  Get top level frame
- - **getParent**

  Get the containing window of the widget.
- - (void)**setWindowTitle:** (const char *)*title*

  Set the title on the widget.
- - **setX:** (int)*x* **Y:** (int)*y*

  Set the position of the widget.
- - **setWidth:** (unsigned)*width* **Height:** (unsigned)*height*

  Set the width and height of the widget.
- - **setHeight:** (unsigned)*height*

  Set the height of the widget.
- - **setWidth:** (unsigned)*width*

  Set the width of the widget.
- - (void)**setActiveFlag:** (BOOL)*activeFlag*

  Enable or disable the widget.
- - (void)**packForgetAndExpand**
- - (void)**packToRight:** *widget*
- - (void)**packFillLeft:** (BOOL)*expandFlag*
- - (void)**packBeforeAndFillLeft:** *widget* **expand:** (BOOL)*expandFlag*
- - (void)**packFill**
- - (void)**pack**

  Roughly, packing a widget makes it draw on the screen. The Tk packer allows complicated options to control widget layout. See documentation on Tk to learn more about packing details.

WindowGeometryRecord

Name

`WindowGeometryRecord` — A container for window geometry information.

Description

A container for window geometry information that implements archiving methods.

Protocols adopted by WindowGeometryRecord

Serialization *(see page 67)*

Create *(see page 46)*

Drop *(see page 54)*

Methods

Phase: Using

- `- (int)`**`getY`**

  `Get the window's vertical position.`

- `- (int)`**`getX`**

  `Get the window's horizontal position.`

- `- (unsigned)`**`getHeight`**

  `Get the window's vertical size.`

- `- (unsigned)`**`getWidth`**

  `Get the window's horizontal size.`

- `- (BOOL)`**`getPositionFlag`**

  `Get the flag that indicates if the position has been set.`

- `- (BOOL)`**`getSizeFlag`**

  `Get the flag that indicates if the size has been set.`

- `-` **`setWidth:`** `(unsigned)`*`w`* **`Height:`** `(unsigned)`*`h`*

  `Set the window size.`

- `-` **`setX:`** `(int)`*`x`* **`Y:`** `(int)`*`y`*

  `Set the window position.`

ZoomRaster

Name

ZoomRaster — A zoomable Raster.

Description

ZoomRaster is a subclass of Raster that implements a zoomable image. It handles translation between logical coordinates and screen coordinates.

Protocols adopted by ZoomRaster

Raster *(see page 346)*

CREATABLE *(see page 44)*

Methods

Phase: Using

- - (void)**handleConfigureWidth:** (unsigned)*newWidth* **Height:** (unsigned)*newHeight*

  Reconfigures the ZoomRaster when the window is resized.

- - **setZoomFactor:** (unsigned)*z*

  Set the zoom factor.

- - (void)**fillCenteredRectangleX0:** (int)*x0* **Y0:** (int)*y0* **X1:** (int)*x1* **Y1:** (int)*y1* **Color:** (Color)*color*

  Special method for ZoomRasters. Like fillRectangleX0:Y0:X1:Y1:Color: in Raster, it will fill a rectangle of given geometry and color. This method makes sure that zooming the window does not change the logical position of the rectangle in relation to the logical coordinates. In other words, if a rectangle includes point (10,10) at one zoom factors, then that same point is included for all zoom factors.

- - (unsigned)**getZoomFactor**

  Get the current zoom factor.

- - (void)**decreaseZoom**

  Make the raster smaller.

- - (void)**increaseZoom**

  Make the raster bigger.

General

Name

gui — GUI interface for Swarm

Description

Tcl/Tk is a scripting language and graphical widget set. TkObjc is a library of wrapper classes around Tk and BLT widgets. It's purpose is to provide a simple graphical interface while hiding most Tk-specific code from library users. To create and use graphical widgets, the user merely needs to create and use objects. Many of the objects here are straightforward wrappings of Tk widgets, but some (ButtonPanel, for instance) are combinations of other widgets, and others (Raster) are novel code.

TkObjc works with most configurations of Tcl, Tk and BLT. It depends on the tclobjc package, the current version is 1.3 (available from Swarm authors). Very little of this code is library-version dependent, however, as most of it works by directly calling the Tk interpreter.

The basic purpose of tkobjc is to package Tk functionality. Therefore, tkobjc's behaviour is similar to the Tk toolkit. For simple usage one should be able to get fairly far just by looking at this document, the header files, and the Swarm examples: more complicated graphical output will require the programmer have some familiarity with Tk.

Macros

- ButtonLeft
- ButtonMiddle
- ButtonRight
- **GUI_BEEP**()
- **GUI_DRAG_AND_DROP**(*source, object*)
- **GUI_DRAG_AND_DROP_OBJECT**()
- **GUI_EVENT_ASYNC**()
- **GUI_EVENT_SYNC**()
- **GUI_FOCUS**(*widget*)
- **GUI_INIT**(*arguments*)
- **GUI_MAKE_FRAME**(*widget*)
- **GUI_PACK**(*widget*)
- **GUI_RELEASE_AND_UPDATE**()
- **GUI_UPDATE**()
- **GUI_UPDATE_IDLE_TASKS**()
- **GUI_UPDATE_IDLE_TASKS_AND_HOLD**()

Functions

- `void` **`initTkObjc`**`(id` *`arguments`*`)`

Typedefs

- Color unsigned char
- PixelValue unsigned long

Analysis Library

Overview

This is the library where tools primarily related to analysis tasks, reside. This includes tools which simplify the task of graphing values or displaying distributions as well as more specific measurement tools (such as Average, Entropy).

1. Dependencies

Following are the other header files imported by <analysis.h>:

```
#import <objectbase.h>
#import <simtoolsgui.h>
```

2. Compatibility

No explicit compatibility issues for particular versions of Swarm

Documentation and Implementation Status

Revision History

2004-07-30 analysis.h *mgd*

(EZAverageSequence, EZSequence): Also exclude in --disable-gui case, as they won't be implemented.

2004-07-21 analysis.h *schristley*

#ifndef DISABLE_GUI classes not applicable for a non-gui Swarm build.

2002-05-15 analysis.h *mgd*

(ActiveGraph): Move getCurrentValue to using phase. (ActiveOutFile): Add getCurrentValue.

2001-05-13 analysis.h *mgd*

(EZBin, EZGraph): For the sake of COM, name the count: argument "count".

2001-01-17 analysis.h *mgd*

(EZSequence): Add setUnsignedArg:. (EZAverageSequence): Adopt EZSequence.

2000-04-27 analysis.h *mgd*

([{Averager,Entropy,EZDistribution,EZGraph} -createEnd]): Remove. ([{EZBin,FunctionGraph] +createBegin:, -createEnd]): Remove. ([EZDistribution -update, -output]): Protect with #ifndef IDL.

2000-03-28 *mgd*

Swarmdocs 2.1.1 frozen.

2000-02-29 *mgd*

Swarmdocs 2.1 frozen.

1999-09-07 analysis.h *mgd*

Add @class EZAverageSequence;

1999-09-07 analysis.h *alex*

(EZAverageSequence): New protocol, make RETURNABLE.

1999-09-07 analysis.h *alex*

(EZGraph), EZGraph.[hm]: ([EZGraph -createSequence:withFeedFrom:andSelector:]) returns an object of type EZSequence. ([EZGraph -create{Average,Min,Max,Count}Sequence:withFeedFrom:andSelector:]): returns an object of type EZAverageSequence.

1999-08-22 analysis.h *mgd*

Add Zone typing to +create:* methods. (EZSequence): Change from CREATABLE to RETURNABLE.

1999-08-09 analysis.h *alex*

(EZGraph): Add two convenience methods for batch-mode (non-graphical) instances of EZGraph (+create:setFileOutput:, +create:setFileName:).

1999-08-02 analysis.h *alex*

(EZGraph): Add new convenience factory method to protocol.

1999-07-21 analysis.h *vjojic*

Add EZSequence protocol.

1999-04-26 analysis.h *alex*

(EZBin, EZGraph): Add compliance to SwarmObject protocol.

1999-02-26 analysis.h *mgd*

Add CREATABLE tags to all non-abstract protocols.

1999-01-16 analysis.h *alex*

([EZGraph -setFileName:]): Update method documentation.

1998-12-10 analysis.h *mgd*

(EZBin, EZGraph): Declare getFilename and setFilename:. Add getTitle.

1998-09-30 analysis.h *mgd*

(EZBin): Add setMonoColorBars:. (EZGraph): Change getGraph return type, and include createEnd.

1998-09-18 analysis.h *mgd*

(EZBin): Note that the upper bound is not inclusive.

1998-09-03 analysis.h *mgd*

Likewise.

1998-08-24 analysis.h *mgd*

(EZGraph): Return value is now a sequence for create{,Average,Count,Min,Max,Total}Sequence:withFeedFrom: methods. Declare new method -dropSequence:.

1998-07-10 analysis.h *alex*

(FunctionGraph): Add phase tags CREATING and USING.

1998-07-08 analysis.h *mgd*

(ActiveGraph, ActiveOutFile, FunctionGraph): Add protocols and class objects.

1998-06-17 Makefile.am *mgd*

Include from refbook/ instead of src/.

1998-06-17 analysis00.sgml *alex*

Added missing description of dependencies - objectbase.h and simtoolsgui.h.

1998-06-15 Makefile.am *mgd*

(MODULE): New variable. Include Makefile.rules from src. Remove everything else.

1998-06-12 analysis00.sgml, analysiscont.sgml *mgd*

Update IDs to SWARM.module.SGML.type.

1998-06-06 analysis.ent *mgd*

Use public identifiers.

1998-06-05 Makefile.am *mgd*

(swarm_ChangeLog): Add.

1998-06-03 analysis.h *mgd*

Updated documentation tags.

1998-05-23 Makefile.am *mgd*

New file.

1998-05-23 analysis.ent.in *mgd*

New file:

1998-05-23 analysis.ent *mgd*

Removed.

1998-05-22 *mgd*

Begin revision log.

1998-05-06 analysis.h *mgd*

(Averager, Entropy, EZBin, EZDistribution, EZGraph): (EZBin): Declare +createBegin:. (Entropy, EZDistribution): Declare -createEnd. (EZBin, EZGraph): -setGraphs and -setFileOutput take a BOOL, not int as an argument. Reorder for phase tags.

1998-04-23 analysis.h *mgd*

Remove includes of analysis things. Replace with protocols for Averager, EZBin, EZDistribution, EZGraph, and Entropy. Add a @class for each.

ActiveGraph

Name

`ActiveGraph` — Provides a continuous data feed between Swarm and the GUI.

Description

An active graph object is the glue between a MessageProbe (for reading data) and a Graph GraphElement. ActiveGraphs are created and told where to get data from and send it to, and then are scheduled to actually do graphic functions. This class is used by EZGraph, and we expect to see less direct usage of it by end-users as more analysis tools (such as EZGraph) internalize its functionality.

Protocols adopted by ActiveGraph

MessageProbe *(see page 202)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- `- ` **`setDataFeed:`** *`d`*

  `Sets the object that will be probed for data.`

- `- ` **`setElement:`** *`ge`*

  `Sets the graph element used to draw on.`

Phase: Using

- `- (double)` **`getCurrentValue`**

  `Returns the value of the graph element..`

- `- (void)` **`step`**

  `Fires the probe, reads the value from the object, and draws it on the graph element. The X value is implicitly the current simulation time. Y is the value read.`

ActiveOutFile

Name

`ActiveOutFile` — An object that actively updates its file stream when updated.

Description

This is the file I/O equivalent of ActiveGraph: it takes an OutFile object, a target (datafeed) object, and a selector, which it uses to extract data from the object and send it to the file.

Protocols adopted by ActiveOutFile

MessageProbe *(see page 202)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- - **setDataFeed:** *d*

  The setDataFeed: method sets the object that will be probed for data.

- - **setHDF5Dataset:** (id <HDF5>)*hdf5Dataset*

  Sets an extensible HDF5 dataset that data will be appended.

- - **setFileObject:** *aFileObj*

  The setFileObject: method sets the file object to which the data will be sent.

Phase: Using

- - (double)**getCurrentValue**

  Returns the last probed value

- - (void)**step**

  The step method fires the probe, reads the value from the object, and sends the value to the file.

Averager

Name

`Averager` — Averages together data, gives the data to whomever asks.

Description

Averager objects read a value (via a MessageProbe) from a collection (typically a list) of objects and collect statistics over them.

Protocols adopted by Averager

MessageProbe *(see page 202)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- `- setWidth: (unsigned)width`

  Set sampling width for target.

- `- setCollection: aTarget`

  Sets the collection of objects that will be probed.

Phase: Using

- `- (unsigned)getCount`

  The getCount method returns the number of values the averager collects.

- `- (double)getMin`

  The getMin method returns the minimum value the averager collects. The value is read out of the object, not computed everytime it is asked for.

- `- (double)getTotal`

  The getTotal method sums the values the averager collects. The value is read out of the object, not computed everytime it is asked for.

- `- (double)getMovingStdDev`

  The returns the square root of -getMovingVariance.

- `- (double)getStdDev`

  The returns the square root of -getVariance.

- `- (double)getMovingVariance`

  The returns the unbiased estimate of sample variance using the specified sampling width.

- `- (double)getVariance`

The returns the unbiased estimate of sample variance per the `corrected' formula (Hays, Statistics 3rd ed, p. 188).

- - (double)**getMovingAverage**

  The getMovingAverage method averages the values the averager collects using the specified sampling width.

- - (double)**getAverage**

  The getAverage method averages the values the averager collects. The total and count are read out of the object to compute the average.

- - (void)**update**

  The update method runs through the collection calling the selector on each object.

EZAverageSequence

Name

EZAverageSequence — Protocol for an EZAverageSequence

Description

An averaging sequence generated using an EZGraph instance returns an object of this type.

Protocols adopted by EZAverageSequence

EZSequence *(see page 377)*

RETURNABLE *(see page 66)*

Methods

Phase: Using

- - (id <Averager>)**getAverager**

  Returns a pointer to the Averager object that provides data to this sequence. This might be useful if one wants to find additional information about the data, because the Averager can calculate not only averages, but also indicators of dispersion, in the ordinary or moving average format.

EZBin

Name

`EZBin` — An easy to use histogram interface.

Description

This class allows the user to easily histogram data generated by a collection of objects. In addition the class will generate some standard statistics over the resulting dataset.

Protocols adopted by EZBin

SwarmObject *(see page 211)*

GUIComposite *(see page 302)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- - **setColors:** (const char * const *)*colors* **count:** (unsigned)*count*

  Set a custom vector of colors for the histogram bars

- - **setUpperBound:** (double)*theMax*

  The setUpperBound method sets the non-inclusive upper bound on the histogram range.

- - **setLowerBound:** (double)*theMin*

  The setLowerBound method sets the inclusive lower bound on the histogram range.

- - **setBinCount:** (unsigned)*theBinCount*

  The setBinCount method sets the number of bins the histogram will have.

- - **setMonoColorBars:** (BOOL)*mcb*

  The setMonoColorBars method specifies whether all bars should be shown in a single color (blue). The default is differently colored bars.

- - **setAxisLabelsX:** (const char *)*xl* **Y:** (const char *)*yl*

  The setAxisLabels:X:Y method sets the horizontal and vertical labels on the histogram in the graphical version of EZBin. (Only relevant if the state of setGraphics is set to 1.)

- - **setFileName:** (const char *)*fileName*

  The setFileName method sets the name used for disk file data output. (Only relevant if the state of seFileOutput is set to 1.) If not set, the filename defaults to be the same as the graph title.

- - **setTitle:** (const char *)*title*

  The setTitle method uses a title string to label a graph window in the graphical version of EZBin. The label appears at the top of the graph window. (Only relevant if the state of setGraphics is set to 1.)

- - **setFileOutput:** (BOOL)*state*

  The setFileOutput method sets the state of file I/O. Set the state to 1 if data for the sequences is to be sent to a file. The default state is 0 meaning that by default no file I/O is carried out by the EZBin class.

- - **setProbedSelector:** (SEL)*aSel*

  The setProbedSelector method sets the selector that will be applied to the objects in the specified collection in order to generate the dataset (inherited from MessageProbe.)

- - **setCollection:** *aCollection*

  The setCollection method sets the collection of target objects which will be requested to generate the dataset for the histogram.

- - **setGraphics:** (BOOL)*state*

  The setGraphics method sets the state of the display. Set the state to 0 if a graphical display of the graph is not required. The default state is 1 meaning that by default the data appears graphically in a window.

Phase: Using

- - (const char *)**getFileName**

  Return the filename string.

- - (const char *)**getTitle**

  Return the title string.

- - (id <Histogram>)**getHistogram**

  Return the histogram widget.

- - (double)**getStdDev**

  The getStd method gets the standard deviation in the dataset. The value is read out of the object, not computed everytime it is asked for.

- - (double)**getAverage**

  The getAverage method gets the average value in the dataset. The value is read out of the object, not computed everytime it is asked for.

- - (double)**getMax**

  The getMax method gets the maximum value in the dataset.

- - (double)**getMin**

  The getMin method gets the minimum value in the dataset.

- - (double)**getUpperBound**

  The getUpperBound method gets the upper bound on the histogram range.

- - (double)**getLowerBound**

The getLowerBound method gets the lower bound on the histogram range.

- - (unsigned)**getBinColorCount**

  The getBinColorCount method gets the number of distinct bin colors allocated (by default, or by the user).

- - (unsigned)**getBinCount**

  The getBinCount method gets the number of bins in the histogram.

- - (unsigned)**getOutliers**

  The getOutliers method gets the number of entries which landed out of the bounds of the histogram. Pressing the "o" key on the graphical representation of the histogram will display this value both as an integer and as a percentage of the total number of attempted entries.

- - (unsigned)**getCount**

  The getCount method gets the number of entries which landed within the bounds of the histogram.

- - (unsigned *)**getDistribution**

  The getDistribution method returns an array of integers containing the number of entries which landed in each bin of the histogram.

- - (void)**output**

  The output: method combines the actions of -outputGraph and -outputToFile. If graph updates and file output need to happen at different frequencies, schedule calls to -outputGraph and -outputToFile instead of -output.

- - (void)**outputToFile**

  The outputToFile method causes the number of entries per bin to be sent to the output file, using the data extracted by the previous call to update. If setFileOutput==0, nothing is done.

- - (void)**outputGraph**

  The ouputGraph method causes the graphical display to be updated with the information extracted by the previous call to update. If setGraphics==0, nothing is done.

- - (void)**update**

  The update method polls the collection of objects and adds the data to the final data set. It is possible to poll the same collection of objects repeatedly, thus increasing the amount of data included in the final dataset, before generating output.

- - (void)**reset**

  The reset method resets the histogram.

- - (void)**setPrecision:** (unsigned)*precision*

  Sets the number of significant figures shown for major-tick labels.

EZDistribution

Name

`EZDistribution` — An EZBin that treats data as a distribution.

Description

This is a subclass of EZBin which normalizes the data and treats it as a distribution. This means that in addition to the statistics it can calculate by virtue of being a subclass of EZBin, it can also calculate the entropy of the distribution as well as return the probabilities associated with the individual bins.

Protocols adopted by EZDistribution

EZBin *(see page 369)*

CREATABLE *(see page 44)*

Methods

Phase: Using

- `- (double)`**`getEntropy`**

  The getEntropy method returns the entropy of the distribution as calculated in the previous call to update.

- `- (double *)`**`getProbabilities`**

  The getProbabilities method returns an array of doubles representing the probability of every bin in the distribution.

- `- (void)`**`output`**

  The output method causes the graphical display to be updated with the information extracted by the previous call to update. When file I/O is enabled (the state of setFileOutput is set to 1), the probability associated with each bin is sent to the output file. When the graphical display is enabled (the state of setGraphics is set to 1), the histogram will be drawn.

- `- (void)`**`update`**

  The update method polls the bins and updates the entropy of the distribution as well as the probabilities associated with the individual bins.

EZGraph

Name

`EZGraph` — A class for easily create graphs.

Description

This class allows the user to easily create graphs of various quantities in the model s/he is investigating. The user first creates the EZGraph, and then creates "Sequences"; (lines) which will appear in the graph. The sequences are generated based on data provided by a single object or a collection of target objects, in reponse to a specified selector. One of the features of the EZGraph is that it will automatically generate average, total, min, max and count sequences without the user having to mess with Averagers amd other low-level classes.

Protocols adopted by EZGraph

SwarmObject *(see page 211)*

GUIComposite *(see page 302)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- - **setColors:** (const char * const *)*colors* **count:** (unsigned)*count*

  Set a custom vector of colors for the graph lines

- - **setAxisLabelsX:** (const char *)*xl* **Y:** (const char *)*yl*

  The setAxisLabels:X:Y method sets the horizontal and vertical labels on the histogram in the graphical version of EZGraph. (Only relevant if the state of setGraphics is set to 1.)

- - **setTitle:** (const char *)*aTitle*

  The setTitle method uses a title string to label a graph window in the graphical version of EZGraph. The label appears at the top of the graph window. (Only relevant if the state of setGraphics is set to 1.)

- - **setFileName:** (const char *)*aFileName*

  The setFileName method sets the name used for disk file data output. (Only relevant if the state of setFileOutput is set to 1.) The name set here is prepended to the names of each data sequence. If file name is NOT set, with this method, the file name for the sequence will default simply to the sequence name.

- - **setHDF5Container:** (id <HDF5>)*hdf5Container*

  The setHDF5Container: method allows one to combine multiple graphs with multiple sequences in a single HDF5 file.

- - **setFileOutput:** (BOOL)*state*

  The setFileOutput method sets the state of file I/O. Set the state to 1 if data for the sequences is to be sent to a file. The default state is 0 meaning that by default no file I/O is carried out by the EZGraph class.

- - **setGraphics:** (BOOL)*state*

  The setGraphics method sets the state of the display. Set the state to 0 if a graphical display of the graph is not required. The default state is 1 meaning that by default the data appears graphically in a window.

- + **create:** (id <Zone>)*aZone* **setHDF5Container:** (id <HDF5>)*hdf5Container* **setPrefix:** (const char *)*prefix*

  Convenience method for creating a non-graphical EZGraph, inside of a HDF5 container.

- + **create:** (id <Zone>)*aZone* **setFileName:** (const char *)*filename*

  Convenience method for creating a non-graphical EZGraph, in this case, the filename is explicitly set by the user

- + **create:** (id <Zone>)*aZone* **setFileOutput:** (BOOL)*fileOutputFlag*

  Convenience method for creating a non-graphical EZGraph, the filename is generated from the sequence name

- + **create:** (id <Zone>)*aZone* **setTitle:** (const char *)*aTitle* **setAxisLabelsX:** (const char *)*xl* **Y:** (const char *)*yl* **setWindowGeometryRecordName:** (const char *)*windowGeometryRecordName*

  Convenience method for creating `graphical' EZGraph instances

- + **create:** (id <Zone>)*aZone* **setTitle:** (const char *)*aTitle* **setAxisLabelsX:** (const char *)*xl* **Y:** (const char *)*yl* **setWindowGeometryRecordName:** (const char *)*windowGeometryRecordName* **setSaveSizeFlag:** (BOOL)*saveSizeFlag*

Phase: Using

- - (void)**step**

  The step method combines -update, -outputGraph and -outputToFile. If you want file output to occur at a different frequency than graph updates, schedule those methods separately instead of using -step.

- - (void)**outputToFile**

  the outputToFile method sends to the disk file data obtained from the last call to -update. If setFileOutput==0, nothing is done.

- - (void)**outputGraph**

  the outputGraph method updates the graph with the data obtained from the last call to -update. If setGraphics==0, nothing is done.

- - (void)**update**

  the -update method causes the underlying sequences to get the next set of data values. If a sequence has a single object attached rather than an Averager, nothing is done.

- - (const char *)**getFileName**

Return the file name prefix string.

- - (const char *)**getTitle**

  Return the title string.

- - **dropSequence:** *aSeq*

  The dropSequence method drops a data sequence (line on the graph), e.g. because the source object no longer exists. The aSeq parameter should be an id previously returned by one of the createSequence: methods. If the drop is successful, the method returns aSeq, otherwise it returns nil.

- - (id <EZAverageSequence>)**createCountSequence:** (const char *)*aName* **withFeedFrom:** *aCollection* **andSelector:** (SEL)*aSel*

  The createCountSequence method takes a collection of objects and generates a sequence based on the count over the responses of the entire object set. The method returns an id which can be used later with -dropSequence.

- - (id <EZAverageSequence>)**createMaxSequence:** (const char *)*aName* **withFeedFrom:** *aCollection* **andSelector:** (SEL)*aSel*

  The createMaxSequence method takes a collection of objects and generates a sequence based on the maximums over the responses of the entire object set. The method returns an id which can be used later with -dropSequence.

- - (id <EZAverageSequence>)**createMinSequence:** (const char *)*aName* **withFeedFrom:** *aCollection* **andSelector:** (SEL)*aSel*

  The createMinSequence method takes a collection of objects and generates a sequence based on the minimum over the responses of the entire object set. The method returns an id which can be used later with -dropSequence.

- - (id <EZAverageSequence>)**createTotalSequence:** (const char *)*aName* **withFeedFrom:** *aCollection* **andSelector:** (SEL)*aSel*

  The createTotalSequence method takes a collection of objects and generates a sequence based on the sum over the responses of the entire object set. The method returns an id which can be used later with -dropSequence.

- - (id <EZAverageSequence>)**createMovingStdDevSequence:** (const char *)*aName* **withFeedFrom:** *aTarget* **andSelector:** (SEL)*aSel* **andWidth:** (unsigned)*width*

  The createMovingStdDevSequence method takes a single object, or collection of objects and generates a sequence based on the variance over the responses of the entire object set for a given width chunk of samples. The method returns an id which can be used later with -dropSequence.

- - (id <EZAverageSequence>)**createStdDevSequence:** (const char *)*aName* **withFeedFrom:** *aCollection* **andSelector:** (SEL)*aSel*

  The createStdDevSequence method takes a collection of objects and generates a sequence based on the sample variance over the responses of the entire object set. The method returns an id which can be used later with -dropSequence.

- - (id <EZAverageSequence>)**createMovingVarianceSequence:** (const char *)*aName* **withFeedFrom:** *aTarget* **andSelector:** (SEL)*aSel* **andWidth:** (unsigned)*width*

The createMovingVarianceSequence method takes a single object, or collection of objects and generates a sequence based on the variance over the responses of the entire object set for a given width chunk of samples. The method returns an id which can be used later with -dropSequence.

- - (id <EZAverageSequence>)**createVarianceSequence:** (const char *)*aName* **withFeedFrom:** *aCollection* **andSelector:** (SEL)*aSel*

  The createVarianceSequence method takes a collection of objects and generates a sequence based on the sample variance over the responses of the entire object set. The method returns an id which can be used later with -dropSequence.

- - (id <EZAverageSequence>)**createMovingAverageSequence:** (const char *)*aName* **withFeedFrom:** *aTarget* **andSelector:** (SEL)*aSel* **andWidth:** (unsigned)*width*

  The createMovingAverageSequence method takes a single object, or collection of objects and generates a sequence based on the average over the responses of the entire object set for a given width chunk of samples. The method returns an id which can be used later with -dropSequence.

- - (id <EZAverageSequence>)**createAverageSequence:** (const char *)*aName* **withFeedFrom:** *aCollection* **andSelector:** (SEL)*aSel*

  The createAverageSequence method takes a collection of objects and generates a sequence based on the average over the responses of the entire object set. The method returns an id which can be used later with -dropSequence.

- - (id <EZSequence>)**createSequence:** (const char *)*aName* **withFeedFrom:** *anObj* **andSelector:** (SEL)*aSel*

  The createSequence method creates a sequence in the EZGraph based on the return value provided by the object anObj when sent the selector aSel. If file I/O is enabled, then the data will be sent to a file with the name aName, otherwise the aName argument is simply used as the legend for the graph element generated by EZGraph. The method returns an id which can be used later with -dropSequence.

- - (id <Graph>)**getGraph**

  The getGraph method lets the user access the graph generated internally by the EZGraph. (Only relevant if the state of setGraphics is set to 1.)

- - (void)**setScaleModeX:** (BOOL)*xs* **Y:** (BOOL)*ys*

  Whether to autoscale every timestep or instead to jump scale.

- - (void)**setRangesYMin:** (double)*ymin* **Max:** (double)*ymax*

  Fix the range of Y values on the graph between some range.

- - (void)**setRangesXMin:** (double)*xmin* **Max:** (double)*xmax*

  Fix the range of X values on the graph between some range.

EZSequence

Name

`EZSequence` — Protocol for an EZSequence

Description

A sequence generated using an EZGraph instance returns an object of this type

Protocols adopted by EZSequence

SwarmObject *(see page 211)*

RETURNABLE *(see page 66)*

Methods

Phase: Using

- `- (double)`**`getCurrentValue`**

  `Returns the current value of the sequence.`

- `- (void)`**`setUnsignedArg:`** `(unsigned)`*`val`*

  `After a sequence has been created and a selector is set, this method allows the user to specify a single unsigned integer argument that is required by the message that the selector implies.`

  Example -setUnsignedArg: #1

```
For  example, suppose you have created an EZGraph called "heightGraph"
If one has an object "dog" in which there is a method
- getFriendHeight: (unsigned)h;
And one wants to create a line to plot the 5th dog,
then the EZsequence can be created with a command like:
{
  id sequence = [heightGraph createSequence: "aName"
                                withFeedFrom: dog
                                andSelector: M(getFriendHeight:);

  [sequence setUnsignedArg: 4];
}
```

Entropy

Name

`Entropy` — Computes entropy via a MessageProbe.

Description

Entropy objects read probabilities (via a MessageProbe) from a collection of objects and calculate the entropy of the underlying distribution.

Protocols adopted by Entropy

MessageProbe *(see page 202)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- `- ` **`setCollection:`** *`aCollection`*

  The setCollection method sets the collection of objects that will be probed.

Phase: Using

- `- (double)` **`getEntropy`**

  The getEntropy method returns the calculated Entropy. The entropy value is read out of the object, not computed everytime it is requested.

- `- (void)` **`update`**

  The update method polls the collection and updates the entropy. This method should be scheduled prior to collecting the data using getEntropy.

FunctionGraph

Name

`FunctionGraph` — A widget for drawing a function over a range of one variable.

Description

The FunctionGraph class is like the ActiveGraph except that instead of plotting values versus time it plots them versus some specified range on the x-axis. Also, instead of plotting one value on each step (as you would with time), FunctionGraph does a complete sampling whenever the `graph' method is called. That is, it graphs f(x) = y for all x in [minX, maxX] where x = minX + n * stepS ize.

The user specifies stuff like minX, maxX, the number of steps between minX and maxX to sample at and a method selector that is a wrapper for the equation being graphed. The method selector must be in a particular format: (BOOL) f: (double *) x : (double *) y If the method returns FALSE then that x value is skipped, otherwise it is assummed that y = f(x) and that value is plotted.

Protocols adopted by FunctionGraph

SwarmObject *(see page 211)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- - **setResetFrequency:** (unsigned)*freq*

  Set the frequency at which to clear the graph element.

- - **setXMin:** (double)*minx* **Max:** (double)*maxx* **StepSize:** (double)*size*

  Set the range and step size of X values at which to compute values.

- - **setXMin:** (double)*minx* **Max:** (double)*maxx* **Resolution:** (unsigned)*steps*

  Set the range and resolution of X values at which to compute values.

- - **setArithmeticWarn:** (BOOL)*state*

  If true, raise a warning if the function method failed to compute a value.

- - **setFunctionSelector:** (SEL)*aSel*

  Set the function method.

- - **setDataFeed:** *feed*

  Set the target to send the function method.

- - **setElement:** (id <GraphElement>)*graphElement*

  Set the GraphElement to use for plotting.

Phase: Using

- `- (void)`**`graph`**

  `Draw the graph with the current contents of the graph element.`

General

Name

`analysis` — Analysis tools

Description

This is the library where tools primarily related to analysis tasks, reside. This includes tools which simplify the task of graphing values or displaying distributions as well as more specific measurement tools (such as Average, Entropy).

Space Library

Overview

The Swarm Space library is the beginnings of a library to assist in building environments for interacting agents. In general, environments can be just as varied as the agents themselves (in one view, the environment itself is simply another agent). However, many simulations have similar types of environments that can be helpfully supported by generic code.

The current space library only addresses simple kinds of discretized 2d space. Improvement is planned in the future. Briefly, coordinates need to be elevated to the status of objects, which should hopefully allow spaces of different scales and boundary conditions to interact through a common reference system. In addition, other types of spaces are desired: continuous coordinates, other dimensions, arbitrary graphs, etc

1. Dependencies

Following are the other header files imported by <space.h>:

```
#import <objectbase.h>
#import <gui.h>
```

2. Compatibility

No explicit compatibility issues for particular versions of Swarm

Documentation and Implementation Status

Swarm is an open ended system which is meant to grow in response to the requirements of the user base, either by inhouse development or through user re-contributions. We are therefore maintaining a list of the most popular requests (both in terms of tools and libraries) so that groups of users can recognize common requirements, make more informed suggestions and so forth:

A 'Double' Space Which could deal with notions of "distance" and answer questions of the form: "which other objects are within X radius of me"? An initial implementation of such a space has been re-contributed by Ginger Booth and may serve as a good foundation for such a space.

Complete Batch-Mode Support Swarm can now run in batch mode, which should allow the user to organize large parameter sweeps over the models s/he is implementing. However, we still need to provide adequate and standardized support for file operations (it should be easy, for example, to load the parameters of an experiment from a file). In order to do this we will provide File objects which will allow users to avoid ad-hoc coding of their file-I/O. This support will be in place well before V1.0.

More Analysis Tools The averager object can generate the Mean, Max, Min, and Count of a given input stream. We would like to add similar tools to calculate entropies, mutual information and other such measures.

Revision History

2004-07-21 space.h *schristley*

#ifndef DISABLE_GUI class not applicable for a non-gui Swarm build.

2001-08-10 space.h *mgd*

(GridData): New protocol. (Discrete2d): Adopt it.

2000-03-28 *mgd*

Swarmdocs 2.1.1 frozen.

2000-02-29 *mgd*

Swarmdocs 2.1 frozen.

1999-08-24 space.h *mgd*

Add Discrete2d and Raster typing on convenience factory methods (for use by Java).

1999-08-22 space.h *mgd*

Add Zone typing to +create* methods. Don't touch arguments that are subclasses of Discrete2d because we don't have a stubbing mechanism for that.

1999-08-03 space.h *alex*

(Diffuse2d): Make initializeLattice a CREATING method to match implementation.

1999-08-01 space.h *alex*

(Object2dDisplay, Value2Display, Discrete2d, Diffuse2d, DblBuffer2d, Grid2d): Add +create: convenience methods to these CREATABLE protocols.

1999-07-09 space.h *mgd*

(Object2dDisplay): Add -makeProbeAtX:Y:.

1999-05-05 space.h *alex*

(Ca2d): Remove CREATABLE tag in protocol definition, since this protocol is intended to be abstract. (Discrete2d): Add example of Lisp output serialization. (Int2dFiler): Note as deprecated protocol, point user to Discrete2d serialization.

1999-05-01 space.h *mgd*

(Discrete2d): Remove setUseObjects and setUseValues.

1999-04-25 space.h *alex*

([Discrete2d -setUseObjects], [Discrete2d -setUseValues]): Add to SETTING phase.

1999-02-26 space.h *mgd*

Merge _Discrete2d with Discrete2d. Add CREATABLE tags for all non-abstract protocols.

1998-10-04 space.h *mgd*

(_Discrete2d): Make x and y arguments to setSizeX:Y:, getObjectAtX:Y: getValueAtX:Y:, putObject:atX:Y:, putValue:atX:Y: unsigned. Make return values of getSize{X,Y} unsigned. (DblBuffer2d): Make x and y arguments to putObject:atX:Y: and putValue:atX:Y: unsigned. (Grid2d): Likewise. (Ca2d): Make argument to setNumStates: unsigned.

1998-07-15 space.h *mgd*

(_Discrete2d): Split Discrete2d into new and presentation interface.

1998-06-17 Makefile.am *mgd*

Include from refbook/ instead of src/.

1998-06-17 space00.sgml *alex*

Added missing description of dependencies - objectbase.h and gui.h.

1998-06-15 Makefile.am *mgd*

(MODULE): New variable. Include Makefile.rules from src. Remove everything else.

1998-06-12 space00.sgml, spacecont.sgml *mgd*

Update IDs to SWARM.module.SGML.type.

1998-06-06 space.ent *mgd*

Use public identifiers.

1998-06-05 Makefile.am *mgd*

(swarm_ChangeLog): Add.

1998-06-03 space.h *mgd*

Add a module summary documentation tag.

1998-05-23 Makefile.am *mgd*

New file.

1998-05-23 space.ent.in *mgd*

New file.

1998-05-23 space.ent *mgd*

Removed.

1998-05-22 *mgd*

Begin revision log.

1998-05-06 space.h *mgd*

(Discrete2d, Ca2d): Add phase tags, reorder in sync. (DblBuffer2d, Value2dDisplay, ConwayLife2d, Diffuse2d, Grid2d, Object2dDisplay, Int2dFiler): Add phase tags. (Value2dDisplay): Declare -createEnd. (Grid2d, Int2dFiler): Declare +createBegin.

1998-04-24 space.h *mgd*

Now a protocol definition file instead of a full set of includes.

Ca2d

Name

`Ca2d` — Defines abstract protocol for cellular automata.

Description

Inherits from DblBuffer2d, defines abstract protocol for cellular automata.

Protocols adopted by Ca2d

DblBuffer2d *(see page 388)*

Methods

Phase: Creating

- - **initializeLattice**

  Use this to set up your CA to a default initial state. Unimplemented in Ca2d; subclass this to set up initial state of lattice.

- - **setNumStates:** (unsigned)*n*

  Record the number of states the CA understands.

Phase: Using

- - **stepRule**

  One iteration of the CA rule. Unimplemented in Ca2d; subclass this to implement your CA rule.

ConwayLife2d

Name

`ConwayLife2d` — Classic 2d Conway's Life CA.

Description

Classic 2d Conway's Life CA.

Protocols adopted by ConwayLife2d

Ca2d *(see page 386)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- - **`initializeLattice`**

  ```
  Initialize lattice to random 1/3 in state 1.
  ```

Phase: Using

- - **`stepRule`**

  ```
  Run Conway's Life rule (simpleminded version).
  ```

DblBuffer2d

Name

`DblBuffer2d` — A double buffered space.

Description

DblBuffer2d augments Discrete2d to provide a form of double buffered space. Two lattices are maintained: lattice (the current state), and newLattice (the future state). All reads take place from lattice, all writes take place to newLattice. newLattice is copied to lattice when updateLattice is called. DblBuffer2d can be used to implement one model of concurrent action, like in Ca2ds. NOTE: be very careful if you're using low-level macro access to the world, in particular be sure that you preserve the write semantics on the newLattice.

Protocols adopted by DblBuffer2d

Discrete2d *(see page 390)*

CREATABLE *(see page 44)*

Methods

Phase: Using

- `- **putValue:** (long)*v* **atX:** (unsigned)*x* **Y:** (unsigned)*y*`

  `Overridden so writes happen to newLattice.`
- `- **putObject:** *anObject* **atX:** (unsigned)*x* **Y:** (unsigned)*y*`
- `- **updateLattice**`

  `Copy newLattice to lattice, in effect updating the lattice.`
- `- (id *)**getNewLattice**`

  `Return a pointer to the newLattice buffer.`

Diffuse2d

Name

`Diffuse2d` — 2d difussion with evaporation.

Description

Discrete 2nd order approximation to 2d diffusion with evaporation. Math is done in integers on the range [0,0x7fff].

Protocols adopted by Diffuse2d

Ca2d *(see page 386)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- `-` **`initializeLattice`**

  `Initialize world to 0.`

- `+` **`create:`** `(id <Zone>)`*`aZone`* **`setSizeX:`** `(unsigned)`*`x`* **`Y:`** `(unsigned)`*`y`* **`setDiffusionConstant:`** `(double)`*`d`* **`setEvaporationRate:`** `(double)`*`e`*

  `Convenience constructor for Diffuse2d`

Phase: Setting

- `-` **`setEvaporationRate:`** `(double)`*`e`*

  `Set the evaporation rate. Values over 1.0 don't make much sense`

- `-` **`setDiffusionConstant:`** `(double)`*`d`*

  `Set the diffusion constant. Values over 1.0 might not be valid.`

Phase: Using

- `-` **`stepRule`**

  `Run discrete approximation to diffusion. Roughly, it's newHeat = evapRate * (self + diffuseConstant*(nbdavg - self)) where nbdavg is the weighted average of the 8 neighbours.`

Discrete2d

Name

`Discrete2d` — Root class of all 2d discrete spaces.

Description

A Discrete2d is basically a 2d array of ids. Subclasses add particular space semantics onto this. Currently Discrete2d grids are accessed by integer pairs of X and Y coordinates.

Protocols adopted by Discrete2d

SwarmObject *(see page 211)*

GridData *(see page 394)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- `- **makeOffsets**`

  Given an array size, compute the offsets array that caches the multiplication by ysize. See the discrete2dSiteAt macro.

- `- (id *)**allocLattice**`

  Allocate memory for the lattice.

- `- **setSizeX:** (unsigned)`*`x`* `**Y:** (unsigned)`*`y`*

  Set the world size.

- `+ **create:** (id <Zone>)`*`aZone`* `**setSizeX:** (unsigned)`*`x`* `**Y:** (unsigned)`*`y`*

  Convenience constructor for Discrete2d lattice

Phase: Setting

- `- **setObjectFlag:** (BOOL)`*`objectFlag`*

  When objectFlag is true, indicates that this lattice is intended only for objects.

- `- **setLattice:** (id *)`*`lattice`*

Phase: Using

- `- **copyDiscrete2d:** (id <Discrete2d>)`*`a`* `**toDiscrete2d:** (id <Discrete2d>)`*`b`*

  This method copies the data in one Discrete2d object to another Discrete2d object. It assumes that both objects already exist.

- `- (int)**setDiscrete2d:** (id <Discrete2d>)`*`a`* `**toFile:** (const char *)`*`filename`*

This method reads a PGM formatted file and pipes the data into a Discrete2d object.

- - **fillWithObject:** *anObj*

  Fills the space using putObject.

- - **fillWithValue:** (long)*aValue*

  Fills the space using putValue.

- - **fastFillWithObject:** *anObj*

  Directly fills the lattice with an object.

- - **fastFillWithValue:** (long)*aValue*

  Directly fills the lattice with a value.

- - **putValue:** (long)*v* **atX:** (unsigned)*x* **Y:** (unsigned)*y*

  Put the given integer to (x,y) overwriting whatever was there.

- - **putObject:** *anObject* **atX:** (unsigned)*x* **Y:** (unsigned)*y*

  Put the given pointer to (x,y) overwriting whatever was there.

Examples

Example #1

Discrete2d instances can now serialize themselves (without needing additional classes such as Int2dFiler).

The result of value (shallow) Lisp serialization of a 4x3 Discrete2d consisting of long values might be:

```
(list
  (cons 'myDiscrete2d
    (make-instance 'Discrete2d
     #:xsize 4 #:ysize 3 #:lattice
      (parse
       #2((1000 1000 1000 1000)
          (1000 1000 1000 1000)
          (1000 1000 10 1000))))))
```

For object (deep) Lisp serialization of the same 4x3 lattice with identical instances of MyClass at each point (except at (2,2) which has an instance of MyClassOther) would look like:

```
(list
  (cons 'myDiscrete2d
    (make-instance 'Discrete2d
      #:xsize 4 #:ysize 3 #:lattice
       (parse
       (cons '(0 . 0)
        (make-instance 'MyClass #:strVal "Hello World"))
       (cons '(0 . 1)
        (make-instance 'MyClass #:strVal "Hello World"))
       (cons '(0 . 2)
```

```
 (make-instance 'MyClass #:strVal "Hello World"))
(cons '(1 . 0)
 (make-instance 'MyClass #:strVal "Hello World"))
(cons '(1 . 1)
 (make-instance 'MyClass #:strVal "Hello World"))
(cons '(1 . 2)
 (make-instance 'MyClass #:strVal "Hello World"))
(cons '(2 . 0)
 (make-instance 'MyClass #:strVal "Hello World"))
(cons '(2 . 1)
 (make-instance 'MyClass #:strVal "Hello World"))
(cons '(2 . 2)
 (make-instance 'MyClassOther #:strVal "Other World"))
(cons '(3 . 0)
 (make-instance 'MyClass #:strVal "Hello World"))
(cons '(3 . 1)
 (make-instance 'MyClass #:strVal "Hello World"))
(cons '(3 . 2)
 (make-instance 'MyClass #:strVal "Hello World"))))))
```

Grid2d

Name

`Grid2d` — A 2d container class for agents.

Description

Grid2d is a generic container class to represent agent position on a 2d lattice. It gets most of its behaviour from Discrete2d, adding extra code to check that you don't overwrite things by accident. Grid2d is pretty primitive: only one object can be stored at a site, no boundary conditions are implied, etc.

Protocols adopted by Grid2d

Discrete2d *(see page 390)*

CREATABLE *(see page 44)*

Methods

Phase: Using

- - **setOverwriteWarnings:** (BOOL)*b*

  If set to true, then if you try to store something at a site that doesn't have 0x0 there, a warning will be generated.

- - **putObject:** *anObject* **atX:** (unsigned)*x* **Y:** (unsigned)*y*

  Replaces the Discrete2d method. First check to see if it should do overwrite warnings, and if so if you're going to overwrite: if both conditions are true, print out a warning message. Regardless of the check, it writes the new object in.

GridData

Name

`GridData` — Methods used by Value2dDisplay and Object2dDisplay for display

Description

Methods required by widgets that display grids. User defined space objects must adopt this or any implementor of it in order to be accepted as data providers in the setDiscrete2dToDisplay method of Value2dDisplay and Object2dDisplay objects. User spaces must also define the macro discrete2dSiteAt(), versions of which can be found in Grid2d.h or Discrete2d.h.

Protocols adopted by GridData

None

Methods

Phase: Using

- `- (long)`**`getValueAtX:`** `(unsigned)`*`x`* **`Y:`** `(unsigned)`*`y`*

  `Return the integer stored at (x,y).`

- `-` **`getObjectAtX:`** `(unsigned)`*`x`* **`Y:`** `(unsigned)`*`y`*

  `Return the pointer stored at (x,y).`

- `- (long *)`**`getOffsets`**
- `- (id *)`**`getLattice`**

  `Returns the lattice pointer - use this for fast access.`

- `- (unsigned)`**`getSizeY`**

  `Get the size of the lattice in the Y dimension.`

- `- (unsigned)`**`getSizeX`**

  `Get the size of the lattice in the X dimension.`

Int2dFiler

Name

`Int2dFiler` — Saves the state of a Discrete2d object [DEPRECATED].

Description

The Int2dFiler class is used to save the state of any Discrete2d object (or a subclass thereof) to a specified file.

Use of this protocol is deprecated, the ability to write the state of a Discrete2d instance to disk (serialize) is now encoded directly to the Discrete2d class, via the lisp and HDF5 archiver features.

Protocols adopted by Int2dFiler

SwarmObject *(see page 211)*

CREATABLE *(see page 44)*

Methods

Phase: Using

- `- **fileTo:** (const char *)`*`aFileName`*

  When the Int2dFiler receives this message, it opens a file called fileName, stores the state of a pre-specified space into it, and then closes the file.

- `- **setBackground:** (int)`*`aValue`*

  This message is optional. It is used when the target Discrete2d contains objects. If a particular location in the space has no resident object, the argument of this message is the value which gets writtent to the file. The default background value is 0.

- `- **setValueMessage:** (SEL)`*`aSelector`*

  This message is optional. It is used when the target Discrete2d contains objects. By sending each object the message specified by the selector, the Int2dFiler is able to get from the object an integer representing its state, which it then writes to the file.

- `- **setDiscrete2dToFile:** (id <Discrete2d>)`*`sSpace`*

  Set the target space to be filled. This message can be used more than once, but often it is useful to keep one Int2dFiler per space (e.g. when the space is saved multiple times).

Object2dDisplay

Name

`Object2dDisplay` — Object2dDisplay displays 2d arrays of objects.

Description

Object2dDisplay helps display 2d arrays of objects. Create a Object2dDisplay, give it a Raster widget to draw on, a Discrete2d, a message to call on each object, and (optionally) a collection of objects and it will dispatch the message to all objects with the Raster widget as an argument. In addition, Object2dDisplay can help you make probees.

Protocols adopted by Object2dDisplay

SwarmObject *(see page 211)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- - **setDisplayMessage:** (SEL)*s*

  Set the message to be sent to each object in the grid to make it draw itself.

- - **setDiscrete2dToDisplay:** (id <GridData>)*c*

  Set the 2d array to draw.

- - **setDisplayWidget:** (id <Raster>)*r*

  Set the display widget to use for drawing.

- + **create:** (id <Zone>)*aZone* **setDisplayWidget:** (id <Raster>)*r* **setDiscrete2dToDisplay:** (id <GridData>)*c* **setDisplayMessage:** (SEL)*s*

  Convenience constructor for Object2dDisplay

Phase: Using

- - **makeProbeAtX:** (unsigned)*x* **Y:** (unsigned)*y*

  Make a probe for an object at a specific point.

- - **display**

  Draw all objects in the array (or optionally, the collection) on the raster widget. All that happens here is the display message is sent to each object - it is the object's responsibility to render itself.

- - **setObjectCollection:** *objects*

Set a collection of objects to be displayed. If this is not given, then Object2dDisplay loops through the 2d grid sending draw messages to all objects it finds there. Giving an explicit collection of objects to draw is more efficient if your grid is sparsely populated.

Value2dDisplay

Name

`Value2dDisplay` — Value2dDisplay displays 2d arrays of values.

Description

Value2dDisplay helps display 2d arrays of values. Value2dDisplay goes through a given Discrete2d array, turn states into colours, and draws them into a Raster widget.

Protocols adopted by Value2dDisplay

SwarmObject *(see page 211)*

CREATABLE *(see page 44)*

Methods

Phase: Creating

- - **setDiscrete2dToDisplay:** (id <GridData>)*c*

  Set which array to draw.

- - **setDisplayWidget:** (id <Raster>)*r* **colormap:** (id <Colormap>)*c*

  Set the display widget and the colourmap to use to draw the value array.

- + **create:** (id <Zone>)*aZone* **setDisplayWidget:** (id <Raster>)*r* **colormap:** (id <Colormap>)*c* **setDiscrete2dToDisplay:** (id <GridData>)*d*

  Convenience constructor for Value2dDisplay

Phase: Using

- - **display**

  Draw the array on the given widget. Note that you still have to tell the widget to draw itself afterwards. The code for display uses the fast macro access in Discrete2d on the cached return value from getLattice. It also caches the drawPointX:Y: method lookup on the display widget - this is a nice trick that you might want to look at.

- - **setDisplayMappingM:** (int)*m* **C:** (int)*c*

  Linear transform of states to colours for drawing. color = state / m + c If not set, assume m == 1 and c == 0.

General

Name

`space` — tools for visualizing objects in various spaces

Description

The Swarm Space library is the beginnings of a library to assist in building environments for interacting agents. In general, environments can be just as varied as the agents themselves (in one view, the environment itself is simply another agent). However, many simulations have similar types of environments that can be helpfully supported by generic code.

The current space library only addresses simple kinds of discretized 2d space. Improvement is planned in the future: see the todo list for ideas. Briefly, coordinates need to be elevated to the status of objects, which should hopefully allow spaces of different scales and boundary conditions to interact through a common reference system. In addition, other types of spaces are desired: continuous coordinates, other dimensions, arbitrary graphs, etc.

Startup protocol

SwarmEnvironment

Name

`SwarmEnvironment` — Container object for Swarm globals

Description

Container object for Swarm globals

Protocols adopted by SwarmEnvironment

CREATABLE *(see page 44)*

Methods

Phase: Creating

- + **initSwarm:** (const char *)*appName* **version:** (const char *)*version* **bugAddress:** (const char *)*bugAddress* **argCount:** (unsigned)*count* **args:** (const char **)*args*
- - **createEnd**
- - **setBatchMode:** (BOOL)*batchMode*
- - **setArguments:** (id <Arguments>)*arguments*
- + **createBegin**

Phase: Using

- - (void)**updateDisplay**
- - (void)**verboseMessage:** (const char *)*message*
- - (const char *)**typeModule:** (const char *)*typeName*
- - (void)**dumpDirectory**
- - (void)**xfprint:** *obj*
- - (void)**xprint:** *obj*
- - (void)**setComponentWindowGeometryRecordName:** *widget* **name:** *name*
- - (void)**setComponentWindowGeometryRecordNameFor:** *obj* **widget:** *widget* **name:** *name*
- - (void)**setWindowGeometryRecordName:** *obj* **name:** (const char *)*name*
- - (void)**createArchivedCompleteProbeDisplay:** *obj* **name:** (const char *)*name*
- - (void)**createArchivedProbeDisplay:** *obj* **name:** (const char *)*name*
- - (void)**createCompleteProbeDisplay:** *obj*
- - (void)**createProbeDisplay:** *obj*

- - (id <SwarmActivity>)**getCurrentSwarmActivity**
- - (timeval_t)**getCurrentTime**
- - (void)**initSwarmUsing:** (const char *)*appName* **version:** (const char *)*version* **bugAddress:** (const char *)*bugAddress* **args:** (const char **)*args*
- - (id <Symbol>)**getLanguageObjc**
- - (id <Symbol>)**getLanguageJava**
- - (id <Symbol>)**getLanguageCOM**
- - (BOOL)**getGuiFlag**
- - (id <Archiver>)**getLispAppArchiver**
- - (id <Archiver>)**getHdf5AppArchiver**
- - (id <Archiver>)**getLispArchiver**
- - (id <Archiver>)**getHdf5Archiver**
- - (id <ProbeDisplayManager>)**getProbeDisplayManager**
- - (id <ProbeLibrary>)**getProbeLibrary**
- - (id <UniformDoubleDist>)**getUniformDblRand**
- - (id <UniformIntegerDist>)**getUniformIntRand**
- - (id <MT19937gen>)**getRandomGenerator**
- - (id <Zone>)**getGlobalZone**
- - (id <Zone>)**getScratchZone**
- - (id <Symbol>)**getControlStateNextTime**
- - (id <Symbol>)**getControlStateQuit**
- - (id <Symbol>)**getControlStateStepping**
- - (id <Symbol>)**getControlStateStopped**
- - (id <Symbol>)**getControlStateRunning**
- - (id <Symbol>)**getSequential**
- - (id <Symbol>)**getRandomized**
- - (id <Symbol>)**getCompleted**
- - (id <Symbol>)**getTerminated**
- - (id <Symbol>)**getReleased**
- - (id <Symbol>)**getHolding**
- - (id <Symbol>)**getStopped**
- - (id <Symbol>)**getRunning**
- - (id <Symbol>)**getInitialized**
- - (id <Symbol>)**getEnd**
- - (id <Symbol>)**getMember**

- `- (id <Symbol>)`**`getStart`**
- `- (id <Arguments>)`**`getArguments`**

General

Name

`swarm` — Top-level Swarm module

Description

Top-level module for controlling startup and providing access to globals

Functions

- `void` **`_initSwarm_`**`(int` *`argc`*`, const char **`*`argv`*`, const char *`*`appName`*`, const char *`*`version`*`, const char *`*`bugAddress`*`, Class` *`argumentsClass`*`, struct argp_option *`*`options`*`, int (*`*`optionFunc`*`)`

Globals

BOOL swarmGUIMode

Flag for whether we're in graphics mode or not. Default is NO.

Appendix A. GridTurtle Test Programs

Important: To prevent the guide from getting overbulky, the actual program listings of the Grid Turtle test programs, that were formerly contained here, can now be found in a separate archive on the Swarm ftp site: gridturtle-2.2.tar.gz *(ftp://ftp.swarm.org/pub/swarm/gridturtle-2.2.tar.gz)*

A.1. Overview

The GridTurtle test programs are a collection of programs which exercise basic capabilities of the defobj, collections, and activity libraries. These programs are always run on each new release of these libraries as a partial test. For the time being they also serve to show examples of working code that exercise basic capabilities or accomplishes particular tasks.

Note, however, that these are not good examples to learn from, or examples to emulate! The programs are not particularly well-designed as either a systematic test or a useful series of tutorial examples, but until there's anything else they help serve for both. To serve their role as test programs some of these examples deliberately make use of the more obscure and low-level features of the libraries they exercise. Mixed in are examples of many basic code fragments an application might need. So be selective in what you use, and read the comments on each program below.

Most of the programs in this directory use a simple type of object called a "GridTurtle." This type of object is an agent that move around on a two-dimensional grid, always moving in a current direction that it maintains internally. This agent is like a "turtle" of the original Logo system, except that its position is constrained to discrete integer values of its X-Y coordinates, and its direction is always one of the four orthogonal directions north, east, south, or west. The GridTurtle object type is implemented using the Library Interface Conventions *(see page 406)* of the Defobj Library *(see page 26)* library, and is itself a defined module that must be initialized by any program.

A.2. Summary of files

A.2.1. Main programs

- **grid0.m** ***(see page 404)*****.** Initialization of library package. (Swarm applications ordinarily just call initSwarm() which takes care of all needed library initialization.) Basic messages on GridTurtle object.
- **grid1a.m, grid1b.m, grid1c.m** ***(see page 404)*****.** Tests of Array type. The setInitialValue:, setDefaultMember:, and setMemberBlock: options are each tested by variants of the same basic program.
- **grid2.m, grid2b.m** ***(see page 404)*****.** Tests of List type. grid2 also tests inheritance from a class that implements the List type (not yet supported, but does work). grid2b uses an internal member slot (a special low-level implementation feature) in what is called a List, but should really be an OrderedSet. The messages being used for this are in the process of being changed, and should not be used.
- **grid3.m, grid3b.m** ***(see page 404)*****.** Tests of Map type. grid3b uses an alternate compare function to handle integers rather than id values as keys.

- **grid4.m, grid4b.m** *(see page 404)***.** Test of Set and OrderedSet types. The OrderedSet creation in grid4b uses message names that differ from the currently documented interface.
- **grid5.m** *(see page 404)***.** Tests of ActionGroup execution independent of any schedule, both by itself and within a swarm.
- **grid6.m** *(see page 404)***.** Test of running an entire top-level activity within an action of an already running activity.
- **grid7.m** *(see page 404)***.** Tests of schedule execution and schedule merging, using various combinations of absolute, relative, and repeating schedules.
- **grid8.m** *(see page 404)***.** Basic test of schedules running within a simulation swarm that is nested within an observer swarm.
- **grid9.m** *(see page 404)***.** Test of two-level nested swarms using a custom subclass for the swarm. (Normal user subclassing should use the Swarm superclass provided in the objectbase *(see page 404)* library.)
- **mousetraps, Mousetrap.m** *(see page 404)***.** Sample application (main program and supporting class) that uses dynamic scheduling. This application is much the same as the mousetrap model of a nuclear chain reaction that is also implemented as a stand-alone application, but it runs outside any user interface framework and links just with the core libraries defobj, collections, and activity.
- **mousetraps2, Mousetrap2.m** *(see page 404)***.** Variant of mousetraps that schedules actions at subclock divisions of a coarser-grained schedule. Used to test this capability; intended only for specialized use when ordinary time units can't be divided finely enough.
- **strtest.m** *(see page 404)***.** Simple test of basic operations on the String type.

A.2.2. Support files

- **Makefile** *(see page 404)***.** Make file for all programs in the directory. Change the SWARMHOME macro setting at the top to reflect installation location of the swarm libraries relative to the local directory. This make file requires the GNU make program; neither Sun nor other makes are compatible with the GNU-specific conventions used.
- **grid.h** *(see page 404)***.** Header file that defines the GridTurtle object type. This file is also compiled to publish external definitions for the grid module. Conventions for coding the header file of a library module are not fully documented yet, but this is a working sample.
- **grid.m** *(see page 404)***.** Initialization of the grid module. Not normally a part of an application program, but included to test module facilities outside of a library. Methods of coding this file are not currently documented, but this is a working sample.
- **GridTurtle.m** *(see page 404)***.** Code for the class that implements the GridTurtle object type. The interface for this type is specified in the file grid.h. Methods for coding classes that implement a type (see library interface conventions *(see page 404)* for a summary of the differences between types and classes) are not currently documented. The important thing to know is simply that a library implements all the messages declared as part of a type.

Appendix B. Library Interface Conventions

B.1. Overview

Some of the basic Swarm libraries are implemented using object definition conventions established by the Defobj Library *(see page 26)*. This document explains how to read and interpret the header files and interface definitions published by such libraries. It also explains the typical structure of documentation provided for libraries that follow these conventions.

B.2. Library Header File

For a library that adopts these conventions, a library is not merely a collection of source files that are compiled into a library archive under control of a make file. Instead, Objective C source files in the library are processed in a special way to publish their definitions not only as header files, but as generated objects available at runtime to make full use of the library. A library processed in this way is referred to as as a "package." (Note: some of the newer Swarm libraries, such as objectbase and random, don't undergo this special processing and yet still follow all the library interface conventions described in this document.)

The Defobj Library *(see page 26)* library documents the full details of special processing performed on a package. For simply using a library, the key fact to keep in mind is that the entire public interface to a library is declared in the one header file having the same name as the package itself, plus a trailing .h suffix. Additionally, the header file of a library normally documents only its public interface, in a way that is completely separated from the implementation of the objects it specifies.

The separation of implementation means that a library publishes its interface entirely without reference to any Objective C classes which implement its objects. Even though classes are often thought of as separating the interface of an object from its implementation, this separation is far from complete. Not only do classes typically contain many internal methods not intended for external use, but they also define a particular storage format for an object defined as instance variables.

A library instead publishes its interface as a set of public object types. These object types may also be supplemented by global object constants called symbols. Both these kinds of definitions normally appear only in the header file of a library. The remaining source files in a library normally contain the classes which implement the object types.

In a library, the files which implement classes (including class header files), need not ever be referenced simply to make use of the implemented capabilities of the library object types. Documentation for the library is normally expressed entirely in terms of the types and symbols published in the library header file. If a feature does not appear in the library header file, it should not be considered part of a supported public interface.

Individual class header files are required to subclass from existing implementations, but interfaces for subclassing are an entirely separate issue from normal public use of an object library. Class inheritance can be a powerful implementation technique, but extension of an existing class framework is typically safe only if performed in explicitly permitted ways. If a library supports subclassing at all, it must carefully state which classes may be subclassed and in what ways. For a library package, this information is supplied outside the library header file. The library header file specifies only the

interfaces by which objects are intended to be used, whether implemented by a local class or an external subclass.

Remaining sections of this document explain the declarations which appear in a library header file, and end with a suggested structure of documentation to be provided for a library. The libraries of Swarm mostly follow this structure.

B.3. Object Type Definitions

The interfaces to objects defined by a library are specified by object type definitions. An object type defines only a set of messages which may be sent to an object over various phases of its lifetime. An object type makes no commitment to the classes that might be used to implement the object.

Multiple classes may all implement the same messages belonging to a type. The independence of types and classes means that different classes can provide alternate implementations of the same object type. For example, a particular implementing class might be selected to optimize the implementation for a particular case.

Object types are similar to protocols defined by Objective C, and the declarations appearing in a library header file are a minor adaptation of Objective C protocol syntax. A key difference from protocols is that object types are published as real external objects that may be used at runtime to create instances of a type. A further difference is that the object types of the Defobj Library *(see page 26)* are divided into separate interfaces that define distinct phases of an object's life cycle.

B.4. GridTurtle example

An example will help illustrate the features of type definitions supported by the defobj library. Throughout documentation of the basic Swarm libraries, a series of running examples will be based on a simple type of object belonging to sample Swarm simulations. This type of object is an agent that move around on a two-dimensional grid, always moving in a current direction that it maintains internally. This agent is like a "turtle" of the original Logo system, except that its position is constrained to discrete integer values of its X-Y coordinates, and its direction is always one of the four orthogonal directions north, east, south, or west.

Following is a complete library header file for a library which defines such an object, called GridTurtle:

```
/*
Name:         GridTurtle.h
Description:  object type for Swarm example programs
Library:      grid
*/

#import <defobj.h>

@deftype GridTurtle <Create, Drop, CREATABLE>
CREATING
- (void)        setXLocMax: (int)xLocMax;
- (void)        setYLocMax: (int)yLocMax;
SETTING
- (void)        setXLoc: (int)xLoc;
```

```
- (void)        setYLoc: (int)yLoc;
- (void)        setXLoc: xLoc setYLoc: yLoc;

- (void)        setDirection: direction;
USING
- (int)         getXLoc;
- (int)         getYLoc;

-               getDirection;

- (void)        move: (int)distance;
- (void)        turn: (int)angle;  // angle measured in units of pi/2 (90
deg.)
- (void)        print;
@end

id <Symbol> North, East, South, West;

#import "grid.xt"
```

An object type is defined by an @deftype declaration. (Note: newer libraries, including objectbase and random, now follow the library interface conventions without using this special @deftype tag. Instead they use just an ordinary @protocol declaration, but otherwise they follow all the structure explained in this document.) The syntax of such declaration is identical to that of an Objective C @protocol definition, except for the entirely uppercase keywords (CREATABLE, CREATING, SETTING, USING) appearing in the GridTurtle example above. All these modifications of Objective C syntax are accomplished by simple preprocessor macros; no extensions to the language compiler are involved.

When this library header file is processed (by a special rule in a make file), an external object id with the name GridTurtle is automatically published. The name of a defined type becomes an ordinary object that accepts specific messages defined by the defobj library. The Defobj Library *(see page 26)* explains the details of such messages; the only purpose here is to explain the basic sections of a deftype declaration.

deftype declarations follow the syntax as Objective C protocols for inheriting messages from each other: a list of names enclosed in angle brackets (e.g., <Create, Drop, ...> above) gives the names of other declared types containing messages to be supported by the new type as well. (These types referenced here are defined by the imported file <defobj.h>.) Like protocols, full multiple inheritance of types is supported. The same messages may be inherited any number of times through any path with no different effect than if inherited or declared just once, so long as no conflicts occur in any of their argument or return types.

The CREATABLE tag appearing in the inherited type list above is a special type which defines no messages of its own, but merely marks the type as one which supports direct creation of instances of the type. Without this tag, the only role of a type is to define messages for inheritance by other types. With this tag, the global type object has a particular implementation that supports object creation using standard messages defined in defobj.

The declared messages of the type may be separated into sections marked by the special uppercase tags such as CREATING, SETTING, and USING above. (Currently, these are the only such tags which may occur.) These sections each define messages belonging to a particular defined "interface" of the object

type, which are further combined into distinct "phases" of an object lifecycle supported by defobj messages. Further explanation of the interfaces and phases defined by this example are provided in the Usage Guide of the defobj library.

B.5. Global Object Symbols

The `grid.h` header file above also contains the declaration:

```
id <Symbol>  North, East, South, West;
```

Lines that declare global id variables of type Symbol, EventType, Warning, or Error (using the angle bracket syntax of id variables conforming to a protocol) are processed somewhat like deftype declarations in that they also produce global id variables initialized to support particular messages defined by defobj. These global variables, however, are not used to define or implement other message interfaces, but only to define certain fixed capabilities referenced through their global object names.

If declared as a Symbol, as in the case here, the generated objects have no particular behavior of their own (other than the character string of their name), but only serve to define unique global id constants which may be used as distinct named values in messages. In this example, the current direction of a GridTurtle object is represented by one of the symbol names North, East, South, or West. These values are like the enum constants of the C language, except that they are defined as full Objective C objects, and may be used with further restrictions. An EventType, or a Warning or Error, defines a further subtype of a Symbol constant with further specialized messages documented in defobj.

B.6. Interface Design Convention

A variety of rules on naming and declaration of object types, symbols, and messages are followed by many of the Swarm libraries. These rules help establish a basic consistency on the library interfaces. Following is a list of such conventions that apply to a public library interface, some but not all of which are derived from standard Smalltalk or Objective C coding practice:

- Names of global object constants are capitalized. In the public interface, such names include types and symbols.
- Recapitalization separates words of a compound name (e.g., GridTurtle). Underscores are generally not used.
- Message names start with a lower-case character, and are named using verbs. Nouns that represent gettable or settable components of object state (e.g., Direction of a GridTurtle), are prefixed by get or set to indicate the action being performed.
- The Smalltalk convention that a message return the receiver of a message if there is no other specific return value is generally *not* followed. If there is no specific return value needed from a message the return type is declared (void).

B.7. Documentation Structure

The following standard sections of documentation are suggested for libraries that follow the strict interface vs. implementation separation of library packages, each with purpose and typical contents as given:

- **Usage Guide.** Tutorial introduction to a library. Focuses on the most common uses in an order reflecting the needs of a first-time user. Most explanation by means of progressively elaborated examples. Serves as a guided tour of major library capabilities. No attempt at reference-style completeness.
- **Advanced Usage Guide.** Continues overview of all significant capabilities of a library, including those which might be needed by advanced users customizing or extending built-in capability. Provides examples of specialized uses extending beyond normal, basic usage, but still using built-in features of the library framework. Helps to simplify the Usage Guide by providing a location for overflow of more advanced features.
- **Interface Reference.** Complete, concise summary of all features of a library. Fills the role a Unix "man page" in providing comprehensive definition of library services. Little or no concern to provide path of tutorial introduction. Does not contain extended examples, at most only fragments of examples which serve needs of specification.
- **Subclassing Reference.** Documents the rules for writing new classes which subclass from classes that implement the library. Because a library might use complex combinations of classes to implement the range of behaviors defined by its types, the classes in a library are not automatically usable as superclasses of user-defined classes. Each library documents which classes are available for use as superclasses, and the specific rules that must be followed when subclassing from these classes.
- **Implementation Notes.** Explains the structure of classes by which the types of a library are implemented. Summarizes the status of implementation if still incomplete and lists items of possible future work. Provides overview and high-level structure of the implementation in whatever ways would best guide a reader of implementation source code. May also discuss tradeoffs considered along with references to related or supporting work.

Appendix C. Licenses for Distribution of Swarm and Applications

- **Swarm libraries.** Swarm is distributed under the GNU Library General Public License *(http://www.gnu.org/copyleft/lgpl.html)* (LGPL).
- **Swarm example applications and documentation.** The Swarm example applications and the Swarm documentation are distributed under the GNU General Public License *(http://www.gnu.org/copyleft/gpl.html)* (GPL).

Important: More information on these GNU LGPL and GPL, and free software in general, please consult the Free Software Foundation *(http://www.gnu.org)* .

Protocol Index

Method Index

Function Index

Global Index

Macro Index

Typedef Index