Reference Guide for Swarm 2.0

SWARM

http://www.santafe.edu/projects/swarm/

ă

Swarm Development Group

Copyrightăľă1996, 1997, 1998, 1999 Santa Fe Institute
Published 28 August 1999
Release 2.0
Licence terms for Swarm documentation

Reproduction of this documentation requires prior copyright release in writing, from the copyright holder (the Santa Fe Institute); *except* for reasonable personal use or educational purposes. Reproduction for mass distribution or profit, is not permitted. The SGML source and associated utilites needed to generate this documentation can be found in the package: swarmdocs-2.0.tar.gz *(ftp://ftp.santafe.edu/pub/swarm/swarmdocs-2.0.tar.gz)* . Permission to use, copy, modify and distribute both the swarmdocs package *and* the documentation it generates (that is the HTML, TeX, dvi, PostScript and RTF output), must be in accordance with the GNU Public Licence *(http://www.gnu.org/copyleft/gpl.html)* (GPL).

Licence terms for Swarm software

The copyright holders make no representation about the suitability of Swarm for any purpose. It is provided “as is” without expressed or implied warranty. Please refer to the GNU Library Public Licence *(http://www.gnu.org/copyleft/lgpl.html)* (LGPL). Permission to use, copy, modify and distribute Swarm must be in accordance with the LGPL.

Overview

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

Revision History

1999-06-21 Makefile.am *alex*
(sgml.elc): Depend on $(abs_top_builddir)/protocol.elc, rather than just protocol.elc. (refindex.sgml $(PAGES)): Likewise. (graph): Likewise.
1999-06-21 sgml.el *alex*
(load-path): Set to TOP_BUILDDIR, rather than BUILDDIR.
1999-06-21 sgml.el *mgd*
(sgml-method-definitions): Use methodinfo- functions instead of list accessors.
1999-06-20 Makefile.am *mgd*
(protocol.elc): Remove rule.
1999-06-20 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.
1999-06-01 ref00a.sgml *alex*
Add porting notes for 1.4/1.4.1 -> 2.0 changes.
1999-04-22 Makefile.am *mgd*
(EXTRA_DIST): Add graph.el. (MODULES): New variable: (SUBDIRS): Use it. (graph): New target.
1999-04-22 graph.el *mgd*
Moved here from toplevel directory.

1999-04-05 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.
1999-02-27 sgml.el *alex*
(run-all): Set uniquify-method-lists to ‘nil’ when calling load-and-process-modules. Ensures all method documentation called correctly.
1999-02-23 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.
1999-02-23 sgml.el *mgd*
(sgml-generate-indices): Use get-top-builddir instead of get-swarmdocs-build-area.
1999-02-15 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.
1999-02-08 refcont.sgml *alex*
(random-app.sgml): Add ‘Random’ Appendix to list of contents.
1999-01-25 refmeta.sgml *alex*
Move bibliodata entity outside of BOOKBIBLIO - so legalnotice links work.
1999-01-22 ref00a.sgml *mgd*
Remove porting note about .swarmArchiver. Swarm 1.4 will read old files, but then change the syntax.
1999-01-14 ref00a.sgml *alex*
Add porting note on Histogram protocol change.
1999-01-12 refmeta.sgml *alex*
Remove old LEGALNOTICE text. Refer the newly-defined {doc,swarm}-legalnotice entities from global.ent. Move CORPAUTHOR inside BOOKBIBLIO.
1999-01-12 gpl-app.sgml *alex*
Refer to the Swarm documentation as being under the terms of the GPL.
1999-01-12 ref00a.sgml *alex*
(SECT1): Add porting notes for Swarm 1.4.
1999-01-06 ref00a.sgml *alex*
Make SIMPLELIST an ITEMIZEDLIST.
1999-01-06 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.
1999-01-06 lgpl-app.sgml *alex*
Likewise.
1999-01-06 ref00.sgml *alex*
Replace SECT1 with two SIMPLESECTs. Tidying and reformatting. (Acknowledgements): Updated.
1998-12-21 probes-app.sgml *alex*
({customized,complete}-probe-map): Add IDs to FIGUREs

1998-10-13 refbook.tex.in *mgd*
Use top_dossrcdir instead of top_srcdir.
1998-10-08 refbook.tex.in *mgd*
Include tex/macros.tex.
1998-08-24 ref00a.sgml *mgd*
Add 1.2 -> 1.3 porting notes.
1998-07-22 ref00a.sgml *mgd*
Put items related to HeatbugObserverSwarm.h in their own list.
1998-07-17 ref00a.sgml *mgd*
Clarify wording about not using statically typed Swarm objects.
1998-06-24 Makefile.am *alex*
(refindex.sgml $(PAGES)): Set temporary environment variable SWARMSRCDIR before invocation of batch-mode e-lisp program ‘protocol.el’.
1998-06-23 ref00a.sgml *alex*
Tidied 1.1 => 1.2 porting notes.
1998-06-22 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.
1998-06-16 refmeta.sgml *mgd*
Use refbookrevhistory.sgml instead of srcrevhistory.sgml. Scale graphic to 100%.
1998-06-16 refbook.sgml *mgd*
Use refbook.ent instead of src.ent.
1998-06-16 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.
1998-06-16 refbook.ent *mgd*
Renamed from src.ent.
1998-06-15 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.
1998-06-14 Makefile.am *mgd*
Include $(top_srcdir)/src/Makefile.rules. (SGML): Move ENT to SGML_FILES. (EXTRA_DIST): New variable.
1998-06-14 Makefile.rules *mgd*
New file.
1998-06-11 refmeta.sgml *mgd*
Scale graphic to 75%.
1998-06-11 Makefile.am *mgd*
(gridexamples.sgml): Update IDs to SWARM.module.SGML.type.
1998-06-11 conventions-app.sgml, gpl-app.sgml, grid-app.sgml, lgpl-app.sgml, probes-app.sgml, ref00.sgml, ref00a.sgml, refmeta.sgml *mgd*
Likewise.

1998-06-10 refbook.sgml *mgd*
Use jpeg instead of jpg for notation and local.notation.class.
1998-06-09 lgpl-app.sgml, gpl-app.sgml, ref00a.sgml *alex*
Fixed the IDs to have the appropriate "SWARM.SRC." prefix in the content ID.
1998-06-09 ref00.sgml *alex*
Made SIMPLESECTs into SECT2s. Fixed redundant "SWARM.SRC." in PREFACE id.
1998-06-09 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.
1998-06-08 refmeta.sgml *mgd*
Change CORPAUTHOR to SDP. Overview needlessly wordy.
1998-06-08 refmeta.sgml *alex*
Change CORPAUTHOR to 'corpauthor' - an SFI Hive global entity.
1998-06-07 refbook.sgml.in *mgd*
Use public identifer for global.ent, and src.ent. Add versions.ent and figs.ent.
1998-06-07 Makefile.am *mgd*
(GENERATED_SGML): Add versions.ent.
1998-06-06 config.ent *alex*
Removed.
1998-06-06 src.ent *alex*
Removed reference to config.ent.
1998-06-06 src.ent.in *mgd*
Define graphic entities per html/print.
1998-06-06 refbook.sgml.in *mgd*
Add local.notation.class entity and notation for JPG.
1998-06-05 src.ent.in *mgd*
Don't use extracted locations, as module entities now all reference public identifiers.
1998-06-04 src.ent.in *mgd*
Add srcrevhistory.sgml.
1998-06-04 refmeta.sgml *mgd*
Replace REVHISTORY with srcrevhistory.sgml.
1998-06-04 Makefile.am *mgd*
(SUBDIRS): Add tech. (swarm_ChangeLog): Add (empty). (GENERATED_SGML): Add srcrevhistory.sgml.
1998-06-02 Makefile.am *mgd*
(gridexamples.sgml): Remove unnecessary subshell for 'for' loop and don't cd to $(srcdir) when done (it's in a subshell).
1998-06-02 Makefile.am *alex*
(CLEANFILES): Changed hardcoded refbook.rtf to $(NAME).rtf. Added Local variable mode for emacs makefile-mode.
1998-05-31 grid-app.sgml *alex*

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

1998-05-31 Makefile.am *mgd*

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

1998-05-31 grid *alex*

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

1998-05-31 Makefile.am *alex*

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

1998-05-28 probes-app.sgml *mgd*

Update LINKENDs per new ID conventions.

1998-05-25 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.

1998-05-22 refbook.sgml.in *mgd*

New file.

1998-05-22 refbook.tex.in *mgd*

New file.

1998-05-22 refbook.sgml *mgd*

Removed.

1998-05-22 Makefile.am *mgd*

New file.

1998-05-22 Makefile *mgd*

Remove.

1998-05-22 src.ent.in *mgd*

New file.

1998-05-22 src.ent *mgd*

Removed.

1998-05-21 *mgd*

Begin revision log.

Table of Contents

List of Tables

List of Figures

List of Examples

Preface

1. The Swarm Hive

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

Send comments to the Swarm team at:

Swarm Project
Santa Fe Institute

1399 Hyde Park Road
Santa Fe NM 87501
USA
`<swarm@santafe.edu>`

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

2. Acknowledgments

The Swarm project was initiated by Chris Langton, formerly of the Santa Fe Institute now at the Swarm Corporation, New Mexico. The current Swarm team is Roger Burkhart (John Deere), Marcus Daniels (SFI), Vladimir Jojic (SFI) and Alex Lancaster (SFI). Irene Lee (also formerly of SFI) continues to be involved in documentation and Swarm user community efforts such as SwarmFest. 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 1.4 or 1.4.1 to 2.0

- The method `setNumBins:`, found in the analysis library protocol EZBin *(see page 332)* and also in the underlying Histogram *(see page 302)* 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:`.

2. 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 84)* protocol in place of the `ListShuffler` protocol.
- The `-setLabels::` and `-setColors::` methods in the Histogram *(see page 302)* 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];
```

3. 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`.

4. Porting from 1.1 to 1.2

4.1. 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.

4.2. 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.

4.3. 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;
```

4.4. 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)

5. Porting from 1.0.5 to 1.1

5.1. 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.

5.2. 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.

5.3. 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. *(*)*

6. 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.

7. 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.)

8. 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*.

9. 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.

See the Release Page *(http://www.santafe.edu/projects/swarm/release.html#changes)* for other differences.

10. 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 190)*.
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 156)*.
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 116)*. 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 `Zones` has been greatly improved and expanded upon. Most of the aspects of the idea behind `Zones` 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 459)* 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 24)* 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 24)*, 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 459)* 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

1999-08-21 defobj.h *mgd*

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

1999-08-08 defobj.h *mgd*

(types_t): Add Class.

1999-08-07 defobj.h *mgd*

(types_t): Add _long_double.

1999-08-04 defobj.h *mgd*

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

1999-06-27 defobj.h *alex*

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

1999-06-12 defobj.h *mgd*

(types_t): Add signed and unsigned types.

1999-06-08 defobj.h *alex*

(initDefobj): Declare to accept new ‘appName’ argument.

1999-06-07 defobj.h *alex*

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

1999-06-04 defobj.h *alex*

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

1999-05-28 defobj.h *mgd*

Import externvar.h.

1999-05-27 defobj.h *mgd*

Use ‘externvar’ for external variable declarations.

1999-05-19 defobj.h *alex*

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

1999-04-28 defobj.h *mgd*

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

1999-04-21 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.

1999-04-20 defobj.h *mgd*

(MAKE_PARSE_FUNCTION_NAME): New macro.

1999-04-20 defobj.h *mgd*

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

1999-04-15 defobj.h *mgd*

(types_t): New typedef.

1999-03-22 defobj.h *mgd*

(FCall): Add getReturnVal.

1999-03-22 defobj.h *vjojic*

(FCall): Mark phases in FCall protocol.

1999-03-16 defobj.h *vjojic*

Add new protocol FCall.

1999-02-26 defobj.h *mgd*

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

1999-02-19 defobj.h *mgd*

Disable CreateDrop protocol; it is already an @interface.

1999-02-18 defobj.h *vjojic*

Add CreateDrop protocol.

1999-02-15 defobj.h *alex*

(generate_class_name): Prefix with ‘extern’.

1999-01-13 defobj.h *mgd*

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

1999-01-11 defobj.h *mgd*

(HDF5): New protocol.

1999-01-09 defobj.h *mgd*

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

1999-01-09 defobj.h *mgd*

(LoadError, SaveError): New error types.

1999-01-07 defobj.h *mgd*

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

1999-01-05 defobj.h *mgd*

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

1999-01-05 defobj.h *mgd*

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

1998-12-27 defobj.h *mgd*

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

1998-12-20 defobj.h *mgd*

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

1998-12-18 defobj.h *mgd*

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

1998-12-17 defobj.h *mgd*

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

1998-12-17 defobj.h *mgd*

(Symbol): Remove setName:.

1998-12-16 defobj.h *mgd*

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

1998-11-16 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.

1998-11-12 defobj.h *mgd*

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

1998-11-11 defobj.h *mgd*

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

1998-07-21 defobj.h *mgd*

Replace @deftype with @protocol throughout.

1998-06-17 defobj00.sgml *alex*

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

1998-06-16 Makefile.am *mgd*

Include from refbook/ instead of src/.

1998-06-14 Makefile.am *mgd*

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

1998-06-13 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.

1998-06-11 defobjpages.sgml, defobjcont.sgml *mgd*

Update IDs to SWARM.module.SGML.type.

1998-06-05 defobj.ent *mgd*

Use public identifiers.

1998-06-04 Makefile.am *mgd*

(swarm_ChangeLog): Add.

1998-06-02 defobj.h *mgd*

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

1998-05-31 defobj.h *alex*

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

1998-05-27 defobj.h *mgd*

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

1998-05-25 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.

1998-05-25 defobj.ent.in *alex*

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

1998-05-25 defobj.ent.in *mgd*

Make defobjrevhistory be a build-directory path.

1998-05-22 Makefile.am *mgd*

New file.

1998-05-22 defobj.ent.in *mgd*

New file.

1998-05-22 defobj.ent *mgd*

Removed.

1998-05-21 *mgd*

Begin revision log.

1998-05-05 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.

1998-05-03 defobj.h *mgd*

Add CREATING and USING tags where absent.

1998-04-26 defobj.h *mgd*

Add documentation tags.

1998-04-16 defobj.h *mgd*

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

1998-01-26 defobj.h *mgd*

Declare nameToObject function.

1997-12-07 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.
1997-12-07 defobj.h *mgd*
Reenable LibraryUsage, DefaultAssumed, and ObsoleteFeature. Put back ObsoleteMessage (gepr argues they are important for a minor release).
1997-12-06 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.

Archiveră

Name

`Archiver` — High level abstract serialization interface.

Description

High level abstract serialization interface.

Protocols adopted by Archiver

Create *(see page 37)*
Drop *(see page 45)*

Methods

Phase: Creating

- – **setInhibitLoadFlag:** (BOOL)*inhibitLoadFlag*
  Make the Archiver ignore any file found in the specified path
- – **setPath:** (const char *)*path*
  Set the physical path for the Archiver to read/write
- – **setSystemArchiverFlag:** (BOOL)*systemArchiverFlag*
  Make the Archiver expect application metadata, such as 'mode' information
- – **setDefaultPath**
  Specify that the Archiver instance use the default system path
- – **setDefaultAppPath**
  Specify that the Archiver to use the default application path

Phase: Using

- – **registerClient:** *client*
- – **unregisterClient:** *client*
- – **putDeep:** (const char *)*key* **object:** *object*
  Register with the Archiver a deep serialization of the object (serialization only occurs when Archiver is saved)

- – **putShallow:** `(const char *)`*key* **object:** *object*
  `As per -putDeep, but only make a shallow version`
- – **getObject:** `(const char *)`*key*
  `Create the object with ‘key’ using the Archiver’s own Zone`
- – **getWithZone:** `(id <Zone>)`*aZone* **object:** `(const char *)`*key*
  `Create the object with ‘key’ in the specified Zone`
- – **save**
  `Execute all the requested -putShallow: and -putDeep: requests using the requested backend`

Argumentsă

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 37)*
Drop *(see page 45)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

- + **createBegin:** `(id <Zone>)`*aZone*
- – **createEnd**
- – **setArgc:** `(int)`*theArgc* **Argv:** `(const char **)`*theArgv*
- – **setAppName:** `(const char *)`*appName*
- – **setAppModeString:** `(const char *)`*appModeString*
- – **setOptionFunc:** `(int (*) (int, const char *))`*optionFunc*

- `-` **setBugAddress:** `(const char *)`*bugAddress*
- `-` **setVersion:** `(const char *)`*version*
- `-` **addOptions:** `(struct argp_option *)`*options*
- `+` **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*
- `-` `(int)`**parseKey:** `(int)`*key* **arg:** `(const char *)`*arg*
  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.

  This method is called for each option that occurs.

Phase: Setting

- `-` **setInhibitArchiverLoadFlag:** `(BOOL)`*inhibitArchiverLoadFlag*
- `-` **setBatchModeFlag:** `(BOOL)`*batchModeFlag*
- `-` **setVarySeedFlag:** `(BOOL)`*varySeedFlag*
- `-` **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.
- `-` **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.

Phase: Using

- `-` `(BOOL)`**getBatchModeFlag**
- `-` `(BOOL)`**getVarySeedFlag**
- `-` `(const char *)`**getAppName**

- − (const char *)**getAppModeString**
- − (int)**getArgc**
- − (const char **)**getArgv**
- − (const char *)**getExecutablePath**
- − (const char *)**getSwarmHome**
- − (const char *)**getConfigPath**
- − (const char *)**getDataPath**
- − (const char *)**getAppDataPath**
  A path where application-specific data files can be expected to be found.
- − (const char *)**getAppConfigPath**
  A path where application-
  specific configuration files can be expected to be found.
- − (BOOL)**getShowCurrentTimeFlag**
- − (BOOL)**getInhibitArchiverLoadFlag**

Examples

Example #1

```
Let's say you want to add a new argument, say 'protocol' to your standard
list of command.  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             Run with a random seed
  -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@santafe.edu.
---------------

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;
}
```

BehaviorPhaseă

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 39)*

Methods

Phase: Creating

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

Phase: Using

- `-` **`getNextPhase`**

CREATABLEă

Name

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

Description

Declare that a defined type supports creation.

Protocols adopted by CREATABLE

None

Methods

None

Copyă

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 re-
  sulting behavior as a starting object, except that an indepen-
  dent copy of the contents of the starting object is created so that fur-
  ther changes to one object do not affect the other.  The zone argu-
  ment specifies the source of storage for the new object.  The message re-
  turns the id of the new object created.
  ```

Createĕ

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 43)*
Customize *(see page 40)*

Methods

Phase: Creating

- + **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.

- + **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.

- - **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.

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
```

CreatedClassă

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 37)*
DefinedClass *(see page 42)*

Methods

Phase: Creating

- – **setName:** (const char *)*name*
- – **setClass:** *aClass*
- – **setSuperclass:** *aClass*
- – **setDefiningClass:** *aClass*
- – **at:** (SEL)*aSel* **addMethod:** (IMP)*aMethod*
- – **lispInCreate:** *expr*
- – **hdf5InCreate:** *hdf5Obj*
- – **lispOutShallow:** *stream*
- – **hdf5OutShallow:** *hdf5Obj*
- – **updateArchiver:** *archiver*

Phase: Using

- – **getDefiningClass**

Customizeă

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

- + **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.
- – **customizeEnd**
  Returns the new, customized version of the original type.
- – **customizeCopy:** *aZone*
  The customizeCopy: message creates a new copy of the interim object returned by customizeBegin: which may be used for further customiza-

```
tions that do not affect the customization al-
ready in progress.  It may be used to branch off a path of a customiza-
tion in progress to create an alternate final customization.
 customizeCopy may be used only on an interim object returned by customize-
Begin: and not yet finalized by customizeEnd.  The new version of the in-
terim object being customized may be allocated in the same or differ-
ent zone as the original version, using the zone argument required by cus-
tomizeCopy:
```

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
```

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
```

DefinedClassă

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 43)*
GetName *(see page 51)*

Methods

Phase: Using

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

DefinedObjectă

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

None

Methods

Phase: Using

- – (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.
- + (BOOL)**conformsTo:** (Protocol *)*protocol*
- – **getClass**
  getClass returns the class that implements the current behavior of an object.
- – (const char *)**getTypeName**
  getTypeName returns the name of the originating type of this 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.
- – (void)**removeRef:** (ref_t)*refVal*
  Remove an external reference to an object.
- – (int)**compare:** *anObject*
  A local implementation of an Object method.
- – **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.
- – (void)**setDisplayName:** (const char *)*displayName*
  Assigns a character string as a name that identifies an object for display or debug purposes.
- – (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)**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.

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

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

- – (void)**xprint**

  Like describe:, but output goes to standard output.

- – (void)**xprintid**

  Like describeID:, but output goes to standard output.

- – (void)**xfprint**

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

- – (void)**xfprintid**

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

- – (id <Zone>)**getZone**

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

Dropă

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 sub-
  type of Zone used to provide storage for the ob-
  ject.  For some zone types, the drop message immediately deallocates stor-
  age for the object and makes the freed storage avail-
  able for other use.  Subsequent use could include the alloca-
  tion of a new object at precisely the same location, resulting in a new ob-
  ject id identical to a previously dropped one.
   The Drop type may be inherited by any type that provides drop sup-
  port for its instances.  In addition to freeing the storage and invalidat-
  ing the object, a drop message may release other resources ac-
  quired or held within the object.  Not every object which can be cre-
  ated can also be dropped, and some objects can be dropped which are not di-
  rectly creatable.  Some objects may be created as a side ef-
  fect of other operations and still be droppable, and some ob-
  jects may be created with links to other objects and not drop-
  pable on their own.  A type independently inherits Cre-
  ate or Drop types, or both, to indicate its support of these standard in-
  terfaces to define the endpoints of an object lifecycle.
  ```

Erroră

Name

`Error` — A condition which prevents further execution.

Description

A condition which prevents further execution.

Protocols adopted by Error

Warning *(see page 61)*
CREATABLE *(see page 35)*

Methods

None

Macros

• **`deferror`**`(`*`name`*`, `*`message`*`)`
`macro used to create and initialize a Warning symbol`

Globals

id <Error> SourceMessage

ă message in the source defines error

id <Error> NotImplemented
ă requested behavior not implemented by object

id <Error> SubclassMustImplement
ă requested behavior must be implemented by subclass

id <Error> InvalidCombination
ă invalid combination of set messages for create

id <Error> InvalidOperation
ă invalid operation for current state of receiver

id <Error> InvalidArgument
ă argument value not valid

id <Error> CreateSubclassing
ă improper use of Create subclassing framework

id <Error> CreateUsage
ă incorrect sequence of Create protocol messages

id <Error> OutOfMemory
ă no more memory available for allocation

id <Error> InvalidAllocSize

ă no more memory available for allocation

id <Error> InternalError
ă unexpected condition encountered in program

id <Error> BlockedObjectAlloc
ă method from Object with invalid allocation

id <Error> BlockedObjectUsage
ă method inherited from Object superclass

id <Error> ProtocolViolation
ă object does not comply with expected protocol

id <Error> LoadError
ă unable to access a resource

id <Error> SaveError
ă unable to save a resource

EventTypeă

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 59)*

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

FArgumentsă

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 37)*
Drop *(see page 45)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

- \+ **createBegin:** (id <Zone>)*aZone*
- − **setJavaFlag:** (BOOL)*javaFlag*
- − **addArgument:** (void *)*value* **ofObjCType:** (char)*type*
- − **addChar:** (char)*value*
- − **addUnsignedChar:** (unsigned char)*value*
- − **addShort:** (short)*value*
- − **addUnsignedShort:** (unsigned short)*value*
- − **addInt:** (int)*value*

- – **addUnsigned:** (unsigned)*value*
- – **addLong:** (long)*value*
- – **addUnsignedLong:** (unsigned long)*value*
- – **addLongLong:** (long long)*value*
- – **addUnsignedLongLong:** (unsigned long long)*value*
- – **addFloat:** (float)*value*
- – **addDouble:** (double)*value*
- – **addLongDouble:** (long double)*value*
- – **addString:** (const char *)*value*
- – **addObject:** *value*
- – **setObjCReturnType:** (char)*type*
- – **createEnd**

Phase: Using

- – (void *)**getResult**

FCallă

Name

FCall — A language independent interface to dynamic calls.

Description

A language independent interface to dynamic calls.

Protocols adopted by FCall

Create *(see page 37)*
Drop *(see page 45)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

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

Phase: Setting

- − **setArguments:** *args*
- − **getArguments**
- − **setFunctionPointer:** (func_t)*fn*
- − **setMethod:** (SEL)*method* **inObject:** *object*
- − **setJavaMethod:** (const char *)*methodName* **inObject:** (void *)*obj*
- − **setJavaMethod:** (const char *)*methodName* **inClass:** (const char *)*className*

Phase: Using

- − (void)**performCall**
- − (void *)**getResult**
- − (retval_t)**getRetVal:** (retval_t)*retVal* **buf:** (types_t *)*buf*

GetNameĕ

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 iden-
  tifies an object in some context of use.  This message is com-
  monly used for objects that are created once in some fixed con-
  text where they are also assigned a unique name.  Constant objects de-
  fined as part of a program or library are examples.  This message is in-
  tended only for returning a name associated with an object through-
  out its lifetime.  It does not return any data that ever changes.
  ```

GetOwnerǎ

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 consid-
  ```

```
ered as the owner of an initial object.  What is considered as an owner de-
pends on its specific object type, but might be a larger ob-
ject of which the local object is a part, or an object that has exclu-
sive control over the local object.  The principal constraint estab-
lished by an ownership structure is that a given ob-
ject can have only a single other object as its unambiguous owner.
```

HDF5ă

Name

`HDF5` — HDF5 interface

Description

HDF5 interface

Protocols adopted by HDF5

Create *(see page 37)*
Drop *(see page 45)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

- + **createBegin:** (id <Zone>)*aZone*
- – **setCreateFlag:** (BOOL)*createFlag*
- – **setDatasetFlag:** (BOOL)*datasetFlag*
- – **setParent:** *parent*
- – **setCompoundType:** *compoundType*
- – **setCount:** (unsigned)*count*
- – **createEnd**

Phase: Setting

- – **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.
- – **setBaseTypeObject:** *baseTypeObject*

Phase: Using

- – **iterate:** (int (*) (id <HDF5>hdf5Obj))*iterateFunc*
- – (BOOL)**getDatasetFlag**
- – (size_t)**getDatasetRank**
- – (size_t)**getDatasetDimension:** (unsigned)*dimNumber*
- – (const char *)**getName**
- – (unsigned)**getCount**
- – **getCompoundType**
- – **getClass**
- – **assignIvar:** *obj*
- – (BOOL)**checkName:** (const char *)*name*
- – (BOOL)**checkDatasetName:** (const char *)*datasetName*
- – **loadDataset:** (void *)*ptr*
- – **storeAsDataset:** (const char *)*name* **typeName:** (const char *)*typeName* **type:** (const char *)*type* **ptr:** (void *)*ptr*
- – **storeTypeName:** (const char *)*typeName*
- – **storeComponentTypeName:** (const char *)*typeName*
- – **shallowLoadObject:** *obj*
- – **shallowStoreObject:** *obj*
- – **nameRecord:** (unsigned)*recordNumber* **name:** (const char *)*recordName*
- – **numberRecord:** (unsigned)*recordNumber*
- – **selectRecord:** (unsigned)*recordNumber*
- – (const char **)**readRowNames**
- – **writeRowNames**
- – **writeLevels**
- – **storeAttribute:** (const char *)*attributeName* **value:** (const char *)*valueString*

- – **iterateAttributes:** (int (*) (const char *key, const char *value))*iterateFunc*
- – (const char *)**getAttribute:** (const char *)*attributeName*
- – (void)**drop**

HDF5Archiverǎ

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 30)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

- + **create:** (id <Zone>)*aZone* **setPath:** (const char *)*path*
  `Convenience method to create an HDF5Archiver from a specified path`

HDF5CompoundTypeǎ

Name

`HDF5CompoundType` — HDF5 composite type interface

Description

HDF5 composite type interface

Protocols adopted by HDF5CompoundType

Create *(see page 37)*
Drop *(see page 45)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

- `+` **`createBegin:`** `(id <Zone>)`*`aZone`*
- `-` **`setClass:`** *`class`*
- `-` **`createEnd`**

Phase: Using

- `-` **`getClass`**

LispArchiverǎ

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 30)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

- + **create:** (id <Zone>)*aZone* **setPath:** (const char *)*path*
  Convenience method to create LispArchiver from a specified path

RETURNABLEă

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

Serializationă

Name

Serialization — Object serialization protocol.

Description

Object serialization protocol.

Protocols adopted by Serialization

None

Methods

Phase: Creating

- – **lispInCreate:** *expr*

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

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

  Process HDF5 object to set create-time parameters.

Phase: Setting

- – **lispIn:** *expr*

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

- – **hdf5In:** (id <HDF5>)*hdf5Obj*

  Load instance variables from an HDF5 object.

Phase: Using

- – **lispOutShallow:** *stream*

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

- – **lispOutDeep:** *stream*

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

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

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

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

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

- – **hdf5OutDeep:** (id <HDF5>)*hdf5obj*

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

- – **updateArchiver:** *archiver*

SetInitialValueă

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 argu-
ment, from which the value of a newly created object is to be taken.  Un-
like a copy message, the ob-
ject used as the source of the new value need not have the identi-
cal type as the new object to be created.  A particular object type de-
fines the types of initial value objects which it can ac-
cept, along with any special conversion or interpretation it might ap-
ply to such a value.
```

Symbolă

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 37)*

GetName *(see page 51)*

CREATABLE *(see page 35)*

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`

Warningă

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 48)*
CREATABLE *(see page 35)*

Methods

Phase: Using

- `– (void)`**`setMessageString:`**` (const char *)`*`messageString`*
  `Associate a message string with this warning.`
- `– (const char *)`**`getMessageString`**
  `Return the message associated with this warning.`

Macros

- **`defwarning`**`(`*`name`*`, `*`message`*`)`
  `macro used to create and initialize an Error symbol`

Globals

id <Warning> WarningMessage

ă message in the source defines warning

id <Warning> ResourceAvailability

ă resource from runtime environment not available

id <Warning> LibraryUsage

ă invalid usage of library interface

id <Warning> DefaultAssumed

ă non-silent use of default

id <Warning> ObsoleteFeature

ă using feature which could be removed in future

id <Warning> ObsoleteMessage

ă using message which could be removed in future

Zoneă

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 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 Drop 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. Several such zones are predefined as part of the defobj library; see predefined zones for a summary.

Protocols adopted by Zone

Create *(see page 37)*
Drop *(see page 45)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

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

```
 PageSize specifies the size of pages within which a zone manages its in-
ternal allocation.  Its default is typically a natural page size (per-
haps 4K) for the local machine architecture.  The default should be over-
ridden only when tuning storage allocation for very specific situa-
tions.  Allocations within a zone are not lim-
ited to the page size, since any requests that ex-
ceed the page size are sim-
ply passed up to the owner zone within which the zone was allocated.
```

Phase: Using

- `-` **`getReclaimPolicy`**
- `- (BOOL)`**`getStackedSubzones`**

- – (size_t)**getPageSize**
- – **allocIVars:** *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.
- – **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.
- – (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.
- – **allocIVarsComponent:** *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.
- – **copyIVarsComponent:** *anObject*
  Like allocateIVarsComponent, except it copies the storage that holds the instances variables for an object.
- – (void)**freeIVarsComponent:** *anObject*
  Frees the instance variable storage for an object.
- – **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 *)**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.

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

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

- – **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.

- – (BOOL)**containsAlloc:** (void *)*alloc*

  containsAlloc: tests if a particular allocation was made by the local zone, or any zone which obtains storage from that zone. The pointer argument must point to an allocation previously made by any zone. The message returns true if the allocation was made by either the local zone or any of its subzones.

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

  Generate debug description for each member of the zone population.

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

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

Generală

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. sign-char + roundup (log (10)/log(2) = 3.3219).
- **FREEBLOCK**(*block*)
- **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
- PTRFMT
  The printf-style format for displaying a pointer.
- **SFREEBLOCK**(*block*)
- **SSTRDUP**(*str*)
- **STRDUP**(*str*)
- **ZFREEBLOCK**(*aZone*, *block*)

- **ZSTRDUP**(*aZone*, *str*)
- 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.
- const char ***generate_class_name**(void)
  Create a class name.
- void **initDefobj**(int *argc*, const char ***argv*, const char **appName*, const char **version*, const char **bugAddress*, Class *argumentsClass*, struct argp_option **options*, int (**optionFunc*)
  initialize defobj with application info, custom Arguments class, and optional (or NULL) extra options w/ processing function.
- 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

- types_t union { id object; SEL selector; Class class; const char *string; 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; double _long_double; }

Globals

id arguments

ă The singleton Arguments object.

id <HDF5Archiver> hdf5Archiver
ă The singleton HDF5 system Archiver object.

id <LispArchiver> lispArchiver
ă The singleton Lisp system Archiver object.

id <HDF5Archiver> hdf5AppArchiver
ă The singleton HDF5 application Archiver object.

id <LispArchiver> lispAppArchiver
ă The singleton Lisp application Archiver object.

id <Symbol> t_ByteArray
ă Predefined type descriptors for allocated blocks.

id <Symbol> t_LeafObject
ă Predefined type descriptors for allocated blocks.

id <Symbol> t_PopulationObject
ă Predefined type descriptors for allocated blocks.

id _obj_globalZone
ă internal variable for globalZone macro

id _obj_scratchZone

ă internal variable for scratchZone macro

BOOL _obj_debug
ă if true then perform all debug error checking

FILE * _obj_xerror
ă output file for error messages

FILE * _obj_xdebug
ă 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 459)* of the Defobj Library *(see page 21)*. 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 360)* 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 459)* 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 360)* for the most complete examples of collections usage.

The collections library follows the documentation structure suggested by the Defobj Library *(see page 21)* 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 360)*, 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

1999-08-21 collections.h *mgd*

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

1999-08-21 collections.h *mgd*

Add objects-conforming-to-Zone argument types.

1999-07-31 collections.h *alex*

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

1999-07-23 collections.h *mgd*

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

1999-07-02 collections.h *mgd*

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

1999-06-29 collections.h *mgd*

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

1999-06-21 collections.h *mgd*

Reflect these changes.

1999-06-07 collections.h *alex*

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

1999-06-07 collections.h *alex*

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

1999-06-04 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.

1999-06-03 collections.h *mgd*

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

1999-05-28 collections.h *mgd*

Include externvar.h.

1999-05-27 collections.h *mgd*

Use ‘externvar’ for external variable declarations.

1999-05-23 collections.h *alex*

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

1999-05-23 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.

1999-01-14 collections.h *mgd*

(String): Remove setLiteralFlag: and getLiteralFlag.

1999-01-11 collections.h *vjojic*

(PermutedIndex): Declare -reshuffle.

1999-01-06 collectionsmeta.sgml *alex*

(End): Fixed missing end comment.

1999-01-05 collections.h *mgd*

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

1999-01-05 collections.h *alex*

(PermutationItem): Add phase tags and documentation strings.

1998-12-27 collections.h *mgd*

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

1998-12-25 collections.h *mgd*

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

1998-12-21 collections.h *vjojic*

Update descriptions of Permutation and PermutedIndex.

1998-12-16 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.

1998-12-13 collections.h *mgd*

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

1998-12-10 collections.h *vjojic*

(ListShuffler): ListShuffler protocol moved from simtools to collections

1998-11-30 collections.h *mgd*

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

1998-11-17 collections.h *mgd*

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

1998-11-16 collections.h *mgd*

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

1998-11-15 collections.h *mgd*

Add corresponding protocols.

1998-11-10 collections.h *mgd*

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

1998-11-01 collections.h *mgd*

(Collection): Note that copies are shallow.

1998-10-09 collections.h *mgd*

(Sorted): Moved to design document.

1998-09-07 collections.h *mgd*

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

1998-07-21 collections.h *mgd*

Replace @deftype with @protocol throughout.

1998-07-15 collections.h *mgd*

Remove Stack and Queue (now in design document).

1998-07-08 collections.h *mgd*

(Map): Remove mention of DupOption.

1998-07-07 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.

1998-06-16 Makefile.am *mgd*

Include from refbook/ instead of src/.

1998-06-14 Makefile.am *mgd*

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

1998-06-13 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:.

1998-06-11 collections00.sgml, collectionscont.sgml, collectionsmeta.sgml *mgd*

Update IDs to SWARM.module.SGML.type.

1998-06-05 collections.ent *mgd*

Use public identifiers.

1998-06-04 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.

1998-06-04 Makefile.am *mgd*

(swarm_ChangeLog): Add.

1998-06-02 collections.h *mgd*

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

1998-05-31 collections.h *mgd*

Make LiteralString a @class.

1998-05-31 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.

1998-05-22 Makefile.am *mgd*

New file.

1998-05-22 collections.ent.in *mgd*

New file:

1998-05-22 collections.ent *mgd*

Removed.

1998-05-21 *mgd*

Begin revision log.

1998-05-05 collections.h *mgd*

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

1998-05-03 collections.h *mgd*

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

1998-04-29 collections.h *mgd*

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

1998-04-27 collections.h *mgd*

Add documentation tags.

1998-04-27 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.

1998-04-10 collections.h *mgd*

Make archiver symbols extern, not common.

1997-12-03 collections.h *mgd*

(OutputStream, String): Constify string arguments.

1997-11-28 collections.h *mgd*

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

1997-11-28 collections.h *mgd*

Declare [gs]etLiteralFlag methods.

1997-11-28 collections.h *mgd*

Append <Collection> to KeyedCollection deftype.

ArchiverArrayă

Name

`ArchiverArray` — Array encapsulation for serialization.

Description

Array encapsulation for serialization.

Protocols adopted by ArchiverArray

Create *(see page 37)*
Drop *(see page 45)*
RETURNABLE *(see page 57)*

Methods

Phase: Creating

- `–` **`setArray:`** *`array`*

Phase: Using

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

ArchiverKeywordă

Name

ArchiverKeyword — Keyword encapsulation for serialization.

Description

Keyword encapsulation for serialization.

Protocols adopted by ArchiverKeyword

Create *(see page 37)*
Drop *(see page 45)*
RETURNABLE *(see page 57)*

Methods

Phase: Creating

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

Phase: Using

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

ArchiverListă

Name

ArchiverList — Archiver list encapsulation for serialization.

Description

Archiver list encapsulation for serialization.

Protocols adopted by ArchiverList

List *(see page 98)*
RETURNABLE *(see page 57)*

Methods

Phase: Using

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

ArchiverPairă

Name

`ArchiverPair` — List pair encapsulation for serialization.

Description

List pair encapsulation for serialization.

Protocols adopted by ArchiverPair

Create *(see page 37)*
Drop *(see page 45)*
RETURNABLE *(see page 57)*

Methods

Phase: Creating

- `– **setCar:** *car*`

- – **setCdr:** *cdr*
- – **setConsFormatFlag:** (BOOL)*theConsFormatFlag*

Phase: Using

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

ArchiverValueĕ

Name

`ArchiverValue` — Value encapsulation for serialization.

Description

Value encapsulation for serialization.

Protocols adopted by ArchiverValue

Create *(see page 37)*
Drop *(see page 45)*
RETURNABLE *(see page 57)*

Methods

Phase: Creating

- – **setDouble:** (double)*val*
- – **setLongDouble:** (long double)*val*
- – **setFloat:** (float)*val*
- – **setLongLong:** (long long)*val*

- – **setChar:** (unsigned char)*val*
- – **setBoolean:** (BOOL)*val*
- – **setNil**

Phase: Using

- – (char)**getValueType**
- – (double)**getDouble**
- – (long double)**getLongDouble**
- – (float)**getFloat**
- – (long long)**getLongLong**
- – (int)**getInteger**
- – (unsigned)**getUnsigned**
- – (unsigned char)**getChar**
- – (BOOL)**getBoolean**
- – **getObject**
- – **lispOutShallow:** (id <OutputStream>)*stream*
- – **lispOutDeep:** (id <OutputStream>)*stream*
- – (void)**drop**

Arrayă

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.

While the IndexSafety option of an Array defaults to Unsafe, indexes are inherently safe under all operations with the exception of resizing an array or resetting its member allocation. (The only individual member operation which modifies contents of an array is replacing a member value, and this does not affect existing indexes.) The UnsafeAtMember value of IndexSafety is not supported since individual members cannot be added or removed. A value of SafeAlways makes existing indexes safe even under resizing or reallocation. If an existing index referred to a member location that is no longer contained in the array, the index location is reset to the End location.

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 84)*

DefaultMember *(see page 88)*

MemberBlock *(see page 104)*

CREATABLE *(see page 35)*

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 collec-
tion.  Any non-negative value including zero is valid.  If the array al-
ready exists, the any existing members up to the new count will pre-
serve their existing values.  If the new count is greater than the exist-
ing count, or a new array is being created, all members will be as-
signed an initial default value of either nil, or a value previously speci-
fied for DefaultMember.
```

Collectionă

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 37)*
SetInitialValue *(see page 58)*
Copy *(see page 36)*
Drop *(see page 45)*
Offsets *(see page 106)*
ForEach *(see page 89)*

Methods

Phase: Creating

- `- (void)`**`setReplaceOnly:`**` (BOOL)`*`replaceOnly`*

```
 This boolean-valued option restricts valid usage of a collection by ex-
cluding all operations which add or remove members.  For some collec-
```

tion 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.

- – (void)**setIndexFromMemberLoc:** (int)*byteOffset*

Phase: Using

- – **copy:** (id <Zone>)*aZone*
  Note: copies are shallow; members inside the collection are not copied.
- – (BOOL)**getReplaceOnly**
- – (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)**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.
- – **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.
- – (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.
- – (void)**deleteAll**
  Like removeAll:, but drops the member(s) as well.

- – (BOOL)**allSameClass**
  `Returns YES if all members are of the same class.`
- – **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.`
- – **beginPermuted:** (id <Zone>)*aZone*

CompareFunctionă

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

- – **setCompareFunction:** (compare_t)*aFunction*
- – **setCompareCStrings**
- – **setCompareIDs**

- − **setCompareIntegers**
- − **setCompareUnsignedIntegers**

Phase: Using

- − (compare_t)**getCompareFunction**

Typedefs

- compare_t int (*) (id, id)

DefaultMemberă

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**

ForEachă

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)**forEach:** (SEL)*aSelector*
- – (void)**forEach:** (SEL)*aSelector* : *arg1*
- – (void)**forEach:** (SEL)*aSelector* : *arg1* : *arg2*
- – (void)**forEach:** (SEL)*aSelector* : *arg1* : *arg2* : *arg3*

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

ForEachKeyă

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*

Indexă

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 collection type documents which kinds of changes can be made without affecting or invalidating existing indexes. The IndexSafety option of Collection provides additional ways to protect indexes against possible effects of independent updates.

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 43)*
Drop *(see page 45)*

Methods

Phase: Using

- – **getCollection**
  getCollection returns the collection referred to by an index. This collection never changes during the lifetime of the index.
- – **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.
- – **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.
- – **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.
- – **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.
- – **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.
- – **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.

- `- 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.

- `- (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 = [in-
dex next])
     {
      // do something with member ...
```

```
    }
  [index drop];
}
```

- – (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.

- – (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.

- – **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)**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.

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

ă values for index location

id <Symbol> End
ă values for index location

id <Symbol> Between
ă values for index location

id <Symbol> Removed
ă values for index location

id <Symbol> Member
ă values for index location

id <Error> OffsetOutOfRange
ă error types for collections

id <Error> NoMembers
ă error types for collections

id <Error> AlreadyAtEnd

ă error types for collections

id <Error> AlreadyAtStart
ă error types for collections

id <Error> InvalidIndexLoc
ă error types for collections

id <Error> InvalidLocSymbol
ă error types for collections

InputStreamă

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 37)*
Drop *(see page 45)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

- `+ create: (id <Zone>)aZone setFileStream: (FILE *)file`
- `- setFileStream: (FILE *)fileStream`

Phase: Using

- `- (FILE *)getFileStream`
- `- getExpr`

KeyedCollectionă

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 84)*
ForEachKey *(see page 90)*

Methods

Phase: Using

- `-` **`createIndex:`** `(id <Zone>)`*`aZone`* **`fromMember:`** *`anObject`*
- `-` **`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.`
- `-` **`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 collec-`

```
tion which matches.  If more than one en-
try was present for the key value, it removes and returns the first mem-
ber in the internal collection created for duplicate members.
```

• – (BOOL)**containsKey:** *aKey*

```
 The containsKey: message returns true if the key value passed as its argu-
ment is contained in the collection, and false otherwise.
```

KeyedCollectionIndexă

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 90)*

Methods

None

Listă

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.

Unless the EndsOnly option is specified, the default value of IndexSafety is UnsafeAtMember. This means that indexes are inherently safe under all operations except those which affect the member at an index or immediately adjacent to an index between members. If the EndsOnly option is specified, the default value of IndexSafety is Unsafe. In this case, indexes could be invalidated by any operation that adds or removes members. The SafeAlways can be specified for any case to get a guarantee of full index safety.

Protocols adopted by List

Collection *(see page 84)*

CREATABLE *(see page 35)*

Methods

Phase: Using

- `- (void)`**`addFirst:`** *`anObject`*

  `The addFirst: message adds a new member to the front of the list.`

- `- (void)`**`addLast:`** *`anObject`*

  `The addLast: message adds a new member to the end of the list.`

- `-` **`removeFirst`**

  `Removes the first member from the list and returns it.`

- `-` **`removeLast`**

  `Removes the last member from the list and returns it.`

ListIndexă

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.

If either the addAfter: or addBefore: message is requested with the EndsOnly option, and the index location is not Start or End, or if remove is requested on an index that is not positioned at either the first or last member, an invalid operation error is raised.

Protocols adopted by ListIndex

Index *(see page 90)*

Methods

Phase: Using

- – (void)**addAfter:** *anObject*
  Add a member after the index.
- – (void)**addBefore:** *anObject*
  Add a member before the index.

ListShufflerǎ

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 37)*
Drop *(see page 45)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

- – **setUniformRandom:** *dist*
  the setUniformRandom: method connects the supplied uniform distribution to the Shuffler (run after createBegin:).
- – **createEnd**
- \+ **create:** (id <Zone>)*aZone* **setUniformRandom:** *dist*

The create:setUniformRandom method creates the Shuffler and connects the supplied distribution object.

Phase: Using

- – **shuffleWholeList:** *list*
  the shuffleWholeList method randomizes the whole list.
- – **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).

Mapă

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 96)*
CompareFunction *(see page 87)*
CREATABLE *(see page 35)*

Methods

Phase: Using

- – (BOOL)**at:** *aKey* **insert:** *anObject*
  at:insert: inserts an entry into a Map containing the key and member val-

`ues 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.`

- – **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.`

MapIndexă

Name

`MapIndex` — The index behavior for a Map.

Description

The index behavior for a Map.

Protocols adopted by MapIndex

KeyedCollectionIndex *(see page 98)*

Methods

Phase: Using

- – **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.`
- – **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.`
- – **next:** (id *)*key*
  `Return the next item and it's key.`

- – **prev:** (id *)*key*
  Return the previous item and it's key.
- – **get:** (id *)*key*
  Return the current item and it's key.

MemberBlockă

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**

MemberSlotă

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]; }

Offsetsă

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

- – **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.`
- – **getFirst**
  `Equivalent to [aCollection atOffset: 0].`
- – **getLast**
  `Equivalent to [aCollection atOffset: [aCollection getCount] - 1].`

OrderedSetă

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 112)*
List *(see page 98)*

CREATABLE *(see page 35)*

Methods

None

OutputStreamă

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 37)*
Drop *(see page 45)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

- `+` **`create:`** `(id <Zone>)`*`aZone`* **`setFileStream:`** `(FILE *)`*`fileStream`*
- `-` **`setFileStream:`** `(FILE *)`*`fileStream`*

Phase: Using

- `-` `(FILE *)`**`getFileStream`**
- `-` `(void)`**`catC:`** `(const char *)`*`cstring`*
  `Writes character string to stream`

- – (void)**catChar:** (char)*ch*
  Writes a character to stream in Lisp archiver format
- – (void)**catFloat:** (float)*flt*
  Writes a float to stream in Lisp archiver format
- – (void)**catDouble:** (double)*dbl*
  Writes a double to stream in Lisp archiver format
- – (void)**catLongDouble:** (long double)*dbl*
  Writes a double to stream in Lisp archiver format
- – (void)**catInt:** (int)*i*
  Writes an integer to stream in Lisp archiver format
- – (void)**catUnsigned:** (unsigned)*un*
  Writes an unsigned to stream in Lisp archiver format
- – (void)**catShort:** (short)*sht*
- – (void)**catUnsignedShort:** (unsigned short)*usht*
- – (void)**catLong:** (long)*lng*
- – (void)**catUnsignedLong:** (unsigned long)*ulng*
- – (void)**catLongLong:** (long long)*lnglng*
- – (void)**catUnsignedLongLong:** (unsigned long long)*ulnglng*
- – (void)**catPointer:** (void *)*ptr*

Permutationă

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 84)*

Create *(see page 37)*
Array *(see page 82)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

- + **`createBegin:`** `(id <Zone>)`*`aZone`*
- − **`setCollection:`** *`collection`*
- − **`setUniformRandom:`** *`rnd`*

Phase: Using

- − **`generatePermutation`**

PermutationItemă

Name

`PermutationItem` — An element of a Permutation

Description

An element of a Permutation

Protocols adopted by PermutationItem

Create *(see page 37)*
RETURNABLE *(see page 57)*

Methods

Phase: Creating

- − **`setItem:`** *`item`*

- − **setPosition:** (unsigned)*position*

Phase: Using

- − **getItem**
- − (unsigned)**getPosition**

PermutedIndexă

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 90)*

Methods

Phase: Creating

- + **createBegin:** (id <Zone>)*aZone*
- − **setCollection:** *aCollection*
- − **setUniformRandom:** *rnd*
- − **createEnd**

Phase: Using

- − **reshuffle**
- − **next**
- − **prev**

- – **findNext:** *anObject*
- – **findPrev:** *anObject*
- – **get**
- – (id <Symbol>)**getLoc**
- – (void)**setLoc:** (id <Symbol>)*locSymbol*
- – (int)**getOffset**
- – **setOffset:** (unsigned)*offset*

Setă

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 112)*

KeyedCollection *(see page 96)*

CREATABLE *(see page 35)*

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. It returns true if member added matched the key of any member already contained in the Set.

Stringă

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 37)*

Drop *(see page 45)*

Copy *(see page 36)*

CREATABLE *(see page 35)*

Methods

Phase: Creating

- + **create:** (id <Zone>)*aZone* **setC:** (const char *)*cstring*

Phase: Setting

- − (void)**setC:** (const char *)*cstring*

Phase: Using

- − (const char *)**getC**
- − (void)**catC:** (const char *)*cstring*
- − (unsigned)**getCount**
- − (int)**compare:** *aString*

Generală

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*)

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

ă Tokens used by the archiving parser.

id <Symbol> ArchiverQuote
ă Tokens used by the archiving parser.

id <Symbol> ArchiverEOL
ă Tokens used by the archiving parser.

id <Symbol> ArchiverDot
ă 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 21)* of the Defobj Library *(see page 21)*. The activity library relies heavily on the basic collection types defined in the Collections Library *(see page 70)*, 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. Only top-level function calls or message sends that initiate model processing are the concern of the activity library. 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 156)* 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 21)*. 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 360)*, 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

1999-08-21 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:*.

1999-07-27 activity.h *alex*

(Schedule): Add +create:setRepeatInterval: and +create:setAutoDrop: convenience methods to CREATING phase of protocol.

1999-07-11 activity.h *vjojic*

(getCurrentSchedule): Typo.

1999-07-10 activity.h *vjojic*

Make Action, ActionCall, ActionTo, ActionForEach, Activity, ScheduleActivity, SwarmActivity, ForEachActivity CREATABLE protocols.

1999-06-15 activity.h *mgd*

(RelativeTime): Make setRelativeTime: return self. (RepeatInterval): Likewise.

1999-06-02 activity.h *alex*

(ActionCreatingTo): Fix missing message receiver in the example code provided. Thanks to Ken Cline <kcline@c3i.saic.com> for the report.

1999-05-28 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>).

1999-05-27 activity.h *mgd*

Use 'externvar' for external variable declarations. (TimebaseMax): Remove; it's unused.

1999-02-04 activity.h *mgd*

(Schedule): Declare -setKeepEmptyFlag:.

1998-12-21 activity.h *mgd*

(ActionType): Remove activate:.

1998-12-19 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.

1998-12-16 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:.

1998-12-10 activity.h *vjojic*

(DefaultOrder): DefaultOrder reinserted.

1998-10-28 activity.h *mgd*

(ActionCreatingTo): Fix example syntax for createActionTo:message:.

1998-09-07 activity.h *mgd*

(COMPLETEDP, HOLDINGP, INITIALIZEDP, RELEASEDP, RUNNINGP, STOPPEDP, TERMINATEDP): New macros.

1998-07-21 activity.h *mgd*

Replace @deftype with @protocol throughout.

1998-07-11 activity.h *mgd*

(Schedule): Declare -insertGroup:.

1998-06-16 Makefile.am *mgd*

Include from refbook/ instead of src/.

1998-06-14 Makefile.am *mgd*

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

1998-06-13 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.

1998-06-11 activity00.sgml, activitycont.sgml, activitymeta.sgml *mgd*

Update IDs to SWARM.module.SGML.type.

1998-06-05 activity.ent *mgd*

Use public identifiers.

1998-06-04 activity.h *mgd*

Add/update documentation tags. Declare addLast: and remove: (in using phase).

1998-06-04 Makefile.am *mgd*

(swarm_ChangeLog): Add.

1998-05-31 activity.h *alex*

(ActionType): Added method -activate: anActionType.

1998-05-22 Makefile.am *mgd*

New file.

1998-05-22 activity.ent.in *mgd*

New file.

1998-05-22 activity.ent *mgd*

Removed.

1998-05-21 activity.h *alex*

(getCurrentTime,getTopLevelActivity): Added (//#) documentation strings. (_activity_zone, _activity_trace): Made existing inline comments into (//G) doc strings:

1998-05-21 *mgd*

Begin revision log.

1998-05-05 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.

1998-04-28 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.

1997-12-16 activity.h *mgd*

Constify argument to _activity_context_error.

Actionă

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

GetOwner *(see page 52)*

RETURNABLE *(see page 57)*

Methods

None

ActionArgsă

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

Action *(see page 126)*

Methods

Phase: Using

- `- (int)`**`getNArgs`**
- `- (void)`**`setArg1:`** *`arg1`*
- `- `**`getArg1`**
- `- (void)`**`setArg2:`** *`arg2`*
- `- `**`getArg2`**
- `- (void)`**`setArg3:`** *`arg3`*
- `- `**`getArg3`**

ActionCallă

Name

`ActionCall` — An action defined by calling a C function.

Description

An action defined by calling a C function.

Protocols adopted by ActionCall

ActionArgs *(see page 126)*
RETURNABLE *(see page 57)*

Methods

Phase: Using

- `- (void)`**`setFunctionPointer:`**` (func_t)`*`fptr`*
- `- (func_t)`**`getFunctionPointer`**

ActionCreatingă

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

ActionCreatingCall *(see page 129)*

ActionCreatingTo *(see page 130)*

ActionCreatingForEach *(see page 130)*

Methods

Phase: Using

• - **createAction:** *anActionType*

```
 The createAction: message specifies that processing of another ac-
tion type is to be performed by the action.  The referenced ac-
tion type is performed in its entirety, from start to finish, as the ef-
fect of the single created action.
```

ActionCreatingCallă

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

- – **createActionCall:** (func_t)*fptr*
- – **createActionCall:** (func_t)*fptr* : *arg1*
- – **createActionCall:** (func_t)*fptr* : *arg1* : *arg2*
- – **createActionCall:** (func_t)*fptr* : *arg1* : *arg2* : *arg3*

ActionCreatingForEachă

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

- – **createActionForEach:** *target* **message:** (SEL)*aSel*
- – **createActionForEach:** *target* **message:** (SEL)*aSel* : *arg1*
- – **createActionForEach:** *target* **message:** (SEL)*aSel* : *arg1* : *arg2*
- – **createActionForEach:** *target* **message:** (SEL)*aSel* : *arg1* : *arg2* : *arg3*

ActionCreatingToă

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

- `-` **`createActionTo:`** *`target`* **`message:`** `(SEL)`*`aSel`*
- `-` **`createActionTo:`** *`target`* **`message:`** `(SEL)`*`aSel`* `:` *`arg1`*
- `-` **`createActionTo:`** *`target`* **`message:`** `(SEL)`*`aSel`* `:` *`arg1`* `:` *`arg2`*
- `-` **`createActionTo:`** *`target`* **`message:`** `(SEL)`*`aSel`* `:` *`arg1`* `:` *`arg2`* `:` *`arg3`*

Examples

Example #1

```
[anActionGroup createActionTo: aTurtle message: M(move:) : obj];
```

ActionForEachă

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 133)*
DefaultOrder *(see page 142)*
RETURNABLE *(see page 57)*

Methods

None

ActionGroupă

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 139)*
ActionCreating *(see page 128)*
DefaultOrder *(see page 142)*

CREATABLE *(see page 35)*

Methods

None

ActionToă

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

ActionArgs *(see page 126)*
RETURNABLE *(see page 57)*

Methods

Phase: Using

- `- (void)`**`setTarget:`** *`target`*
- `-` **`getTarget`**
- `- (void)`**`setMessageSelector:`** `(SEL)`*`aSel`*
- `- (SEL)`**`getMessageSelector`**

ActionTypeă

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 execut-
ing the actions of an ActionType.  This process is controlled by an ob-
ject called an Activity.  The activateIn message initializes an activ-
ity to run under the execution context passed as the swarmContext argu-
ment, and return the activity object just created.  If the execution con-
text is nil, an activity is returned that allows complete execution con-
trol by the caller.  Otherwise, the execution context must be either an in-
stance of SwarmProcess or SwarmActivity.  (These objects are always main-
tained in one-to-one association with each other, ei-
ther one of the pair is equivalent to the other as a swarmContext argu-
ment.)
 If a top-level activity is created (swarmContext is nil), the created ac-
tivity may be processed using activity processing com-
mands such as run, step, etc.  If an activity is created to run un-
der a swarm context, the swarm itself has responsibility for advanc-
ing the subactivity according to its requirements for synchroniza-
tion and control among all its activities.  Activating a plan for execu-
tion under a swarm turns over control to the swarm to execute the subactiv-
```

ity as a more-or-less autonomous activity.

ActivationOrderă

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 132)*
RETURNABLE *(see page 57)*

Methods

Phase: Using

- `- (void)`**`addLast:`** *`mergeAction`*
  `Method to sort concurrent merge actions in the order of swarm activation.`
- `-` **`remove:`** *`mergeAction`*
  `Method to remove concurrent merge action from sorted group`

Activityă

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 43)*
Drop *(see page 45)*
RETURNABLE *(see page 57)*

Methods

Phase: Using

- `- (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 activ-`

ity has when it is no longer eligible to be run further.

- - **stop**
  The stop message causes the a currently running tree of activities to stop further processing and return a Stopped status.
- - (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.
- - (id <Symbol>)**next**
  The next executes actions within a single compound action while skipping over all processing of any complete action plans executed by those actions.
- - (id <Symbol>)**step**
  The step message executes a single action within a tree of activities.
- - (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>)**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.)
- - **getAction**
  Get action containing parameter bindings for the local activity.
- - **getActionType**
  Get action type of action being performed by activity.
- - (void)**setOwnerActivity:** *ownerActivity*
  Change owner from one swarm activity to another.
- - **getOwnerActivity**
  Return activity under which this activity is running.
- - **getSubactivities**
  Return set of subactivities pending to be run.
- - **getControllingActivity**
  Return activity that issued current run request on top-level activity.
- - **getTopLevelActivity**

`Return top of activity tree running the local activity.`

- – `(id <SwarmActivity>)`**`getSwarmActivity`**
  `Return most immediately containing Swarm activity.`
- – `(id <ScheduleActivity>)`**`getScheduleActivity`**
  `Return most immediately containing Schedule activity.`
- – **`setSerialMode:`** `(BOOL)`*`serialMode`*
  `Set serial execution mode.`
- – `(BOOL)`**`getSerialMode`**
  `Return indicator for serial execution mode.`
- – **`getCurrentSubactivity`**
  `Get running subactivity or next subactivity to run.`

AutoDropă

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

• – (void)**setAutoDrop:** (BOOL)*autoDrop*

Phase: Using

• – (BOOL)**getAutoDrop**

CompoundActionă

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 133)*

Collection *(see page 84)*

AutoDrop *(see page 138)*

DefaultOrder *(see page 142)*

Methods

None

ConcurrentGroupă

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 132)*
RETURNABLE *(see page 57)*

Methods

None

ConcurrentGroupTypeă

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`**

ConcurrentScheduleă

Name

`ConcurrentSchedule` — Time-based map usable for concurrent group.

Description

Time-based map usable for concurrent group.

Protocols adopted by ConcurrentSchedule

ActionGroup *(see page 132)*
RETURNABLE *(see page 57)*

Methods

None

DefaultOrderă

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: Creating

- `- (void)`**`setDefaultOrder:`** *`aSymbol`*

Phase: Using

- `-` **`getDefaultOrder`**

ForEachActivityă

Name

`ForEachActivity` — State of execution within a ForEach action.

Description

State of execution within a ForEach action.

Protocols adopted by ForEachActivity

Activity *(see page 135)*
RETURNABLE *(see page 57)*

Methods

Phase: Using

- `-` **`getCurrentMember`**

GetSubactivityActionă

Name

`GetSubactivityAction` — Declare an internal method for getCurrentAction().

Description

Declare an internal method for getCurrentAction().

Protocols adopted by GetSubactivityAction

None

Methods

Phase: Using

- `- _getSubactivityAction_`

ProcessTypeă

Name

`ProcessType` — ProcessType – specification of a process

Description

ProcessType will eventually define support for parameterization of all processes. A process is a uniquely identified course of events that conforms to a specification of external and internal behavior.

Processes include both objects and executable actions. Objects change their state and behavior based on a sequence of externally initiated actions. An action, in contrast, typically has no external behavior other than to be executed in its entirety as a unit (yielding either an observable result or some change of state in a surrounding environment).

Parameterization will be based a uniform framework that defines all input parameters, internal state variables, and/or final results, and that provides for binding of these data in a context of execution. This parameterization framework is still being established, but ProcessType objects will eventually record the specification of any process, to the extent that this specification is defined.

Protocols adopted by ProcessType

None

Methods

None

RelativeTimeă

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**

RepeatIntervală

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`**

Scheduleă

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

CompoundAction *(see page 139)*
ActionCreating *(see page 128)*
RelativeTime *(see page 145)*
RepeatInterval *(see page 145)*
ConcurrentGroupType *(see page 140)*
SingletonGroups *(see page 149)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

- `+` **`create:`** `(id <Zone>)`*`aZone`* **`setRepeatInterval:`** `(timeval_t)`*`rptInterval`*
  `Convenience method for creating a repeating Schedule`
- `+` **`create:`** `(id <Zone>)`*`aZone`* **`setAutoDrop:`** `(BOOL)`*`autoDrop`*
  `Convenience method for creating an AutoDrop Schedule`

Phase: Using

- – **at:** (timeval_t)*tVal* **createAction:** *anActionType*
- – **at:** (timeval_t)*tVal* **createActionCall:** (func_t)*fptr*
- – **at:** (timeval_t)*tVal* **createActionCall:** (func_t)*fptr* : *arg1*
- – **at:** (timeval_t)*tVal* **createActionCall:** (func_t)*fptr* : *arg1* : *arg2*
- – **at:** (timeval_t)*tVal* **createActionCall:** (func_t)*fptr* : *arg1* : *arg2* : *arg3*
- – **at:** (timeval_t)*tVal* **createActionTo:** *target* **message:** (SEL)*aSel*
- – **at:** (timeval_t)*tVal* **createActionTo:** *target* **message:** (SEL)*aSel* : *arg1*
- – **at:** (timeval_t)*tVal* **createActionTo:** *target* **message:** (SEL)*aSel* : *arg1* : *arg2*
- – **at:** (timeval_t)*tVal* **createActionTo:** *target* **message:** (SEL)*aSel* : *arg1* : *arg2* : *arg3*
- – **at:** (timeval_t)*tVal* **createActionForEach:** *target* **message:** (SEL)*aSel*
- – **at:** (timeval_t)*tVal* **createActionForEach:** *target* **message:** (SEL)*aSel* : *arg1*
- – **at:** (timeval_t)*tVal* **createActionForEach:** *target* **message:** (SEL)*aSel* : *arg1* : *arg2*
- – **at:** (timeval_t)*tVal* **createActionForEach:** *target* **message:** (SEL)*aSel* : *arg1* : *arg2* : *arg3*
- – **remove:** *anAction*
  Remove action from either schedule or concurrent group.
- – **insertGroup:** *aKey*
- – **setKeepEmptyFlag:** (BOOL)*keepEmptyFlag*
  Indicate whether an empty schedule should be dropped and ignored or or kept and attended to (default is YES).

ScheduleActivityă

Name

`ScheduleActivity` — State of execution within a Schedule.

Description

State of execution within a Schedule.

Protocols adopted by ScheduleActivity

Activity *(see page 135)*
RETURNABLE *(see page 57)*

Methods

Phase: Using

- – (timeval_t)**getCurrentTime**
  Get current time of activity (pending time if holding).
- – **stepUntil:** (timeval_t)*tVal*
  Advance activity until requested time has been reached.

SingletonGroupsă

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**

SwarmActivityă

Name

SwarmActivity — A collection of started subactivities.

Description

A collection of started subactivities.

Protocols adopted by SwarmActivity

ScheduleActivity *(see page 148)*

RETURNABLE *(see page 57)*

Methods

Phase: Using

- – **getSwarm**

  Return swarm object containing this swarm activity, if any.

- – **getSynchronizationSchedule**

SwarmProcessă

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 133)*
Zone *(see page 62)*
SynchronizationType *(see page 152)*

Methods

Phase: Creating

- `-` **`setInternalZoneType:`** *`aZoneType`*
  ```
  The InternalZoneType option sets the type of zone which is cre-
  ated by the swarm to hold objects cre-
  ated within the swarm.  If set to nil, no internal zone is cre-
  ated within the swarm, and all use of the swarm as if it were a zone raises an er-
  ror.  The default of this option is stan-
  dard Zone type.  (.. Since there is no other Zone type yet, there’s no rea-
  son to set this option yet except to turn off the internal zone. ..)
  ```

Phase: Using

- `-` **`getInternalZone`**
  ```
  getInternalZone returns a Zone ob-
  ject that is used by the swarm to hold its internal ob-
  ```

jects. 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.

- `- (id <Activity>)`**`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.

SynchronizationTypeă

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**

Generală

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> Concurrent

ă values for DefaultOrder

id <Symbol> Sequential
ă values for DefaultOrder

id <Symbol> Randomized
ă values for DefaultOrder

id <Symbol> Initialized
ă Values returned by getStatus.

id <Symbol> Running
ă Values returned by getStatus.

id <Symbol> Holding
ă Values returned by getStatus.

id <Symbol> Released
ă Values returned by getStatus.

id <Symbol> Stopped

ă Values returned by getStatus.

id <Symbol> Terminated
ă Values returned by getStatus.

id <Symbol> Completed
ă Values returned by getStatus.

id <Symbol> HoldStart
ă Values returned by getHoldType.

id <Symbol> HoldEnd
ă Values returned by getHoldType.

id _activity_current
ă Internal variable used by current context macros.

id <Error> InvalidSwarmZone
ă Error issued when an internal zone is expected, but absent.

id _activity_zone
ă _activity_zone – zone in which activity objects created

BOOL (*) (id) _activity_trace
ă 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 21)* and *Activity Library (see page 116)* 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 260)* 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. This *Probing and Displaying the Contents of a SwarmObject (see page 445)* 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 459)*.

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 Objective C - Swarm Style *(see page 452)* in the section entitled *The Create Phase In Principle*.

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 161)* and the Implementation Notes *(see page 164)*; 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 161)* 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 116)* 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 158)*

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 21)* 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.

Revision History

1999-08-24 objectbase.h *mgd*

(VarProbe): Declare DefaultString, CharString, IntString.

1999-08-22 objectbase.h *mgd*

Add VarProbe, MessageProbe, and ProbeMap typing.

1999-08-21 objectbase.h *mgd*

Add Zone typing to +create:* methods. (Swarm): Return Activity with activateIn:

1999-08-08 objectbase.h *alex*

(EmptyProbeMap): Add +create:forClass: convenience create message to protocol.

1999-08-07 objectbase.h *alex*

(ActivityControl): Move -attachToActivity: to USING phase, doesn't need to be a CREATING phase method.

1999-07-14 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.

1999-05-28 objectbase.h *mgd*

Include externvar.h.

1999-05-27 objectbase.h *mgd*

Use 'externvar' for external variables.

1999-04-28 objectbase.h *mgd*

(MessageProbe), MessageProbe.[hm] ([MessageProbe -longDynamicCallOn:]): Replaces intDynamicCallOn:

1999-04-21 objectbase.h *mgd*

(MessageProbe): Add setting tag.

1999-04-15 objectbase.h *mgd*

(val_t): Use types_t instead of included union.

1999-04-06 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.

1999-03-31 objectbase.h *vjojic*

Protocol Swarm inherits protocols SwarmProcess and CREATABLE.

1999-03-20 objectbase.h *mgd*

Make SwarmObject creatable.

1999-03-19 objectbase.h *mgd*

Add @class DefaultProbeMap.

1999-03-07 objectbase.h *mgd*

Fix return type of getArg: (val_t), and correct documentation for getArg: and getArgName:.

1999-02-25 objectbase.h *mgd*

Add CREATABLE tags to all non-abstract protocols.

1999-02-22 objectbase.h *mgd*

(VarProbe, _VarProbe): Merge.

1998-11-11 objectbase.h *mgd*

(Arguments): Remove (moved to defobj.h).

1998-09-03 objectbase.h *mgd*

(val_t): Add _short and _ushort.

1998-09-02 objectbase.h *mgd*

(val_t): Add _uint, _ulong, and _long.

1998-08-19 objectbase.h *mgd*

Declare setDefaultAppDataPath:.

1998-08-18 objectbase.h *mgd*

Declare setDefaultAppConfigPath:.

1998-08-18 objectbase.h *mgd*

Do it here.

1998-08-06 objectbase.h *mgd*

Add @class SwarmObject.

1998-07-14 objectbase.h *mgd*

Split VarProbe into new features vs. user presentation.

1998-07-13 objectbase.h *mgd*

Minor reformatting of documentation.

1998-07-06 objectbase.h *mgd*

(Probe): Add argument to setProbeClass.

1998-07-05 objectbase.h *alex*

(Arguments): Further clarify use of Arguments with main.m program fragment in Example.

1998-06-23 objectbase.h *alex*

(Arguments): Add protocol. Add documentation description for Arguments, including long example marked-up with //E:.

1998-06-16 objectbase.h *mgd*

Document Swarm.

1998-06-16 objectbase00.sgml *alex*

Added missing reference to import of activity.h.

1998-06-16 objectbasemeta.sgml *alex*

Removed redundant text from ABSTRACT.

1998-06-16 Makefile.am *mgd*

Include from refbook/ instead of src/.

1998-06-14 Makefile.am *mgd*

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

1998-06-11 objectbase00.sgml, objectbasecont.sgml *mgd*

Update IDs to SWARM.module.SGML.type.

1998-06-05 objectbase.ent *mgd*

Use public identifiers.

1998-06-04 Makefile.am *mgd*

(swarm_ChangeLog): Add.

1998-06-02 objectbase.h *mgd*

Update documentation tags.

1998-05-27 objectbase.h *mgd*

Include defobj.h.

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

Remove @srcdir@ in revhistory.

1998-05-22 Makefile.am *mgd*

New file.

1998-05-22 objectbase.ent.in *mgd*

New file.

1998-05-22 objecbase.ent *mgd*

Removed.

1998-05-21 *mgd*

Begin revision log.

1998-05-05 objectbase.h *mgd*

Spacing changes in method declarations throughout. (ProbeConfig, DefaultProbeMap): Add description string. (CompleteProbeMap): Add phase tags. (EmptyProbeMap): Remove -createEnd. (CompleteVarMap): Add //S.

1998-04-21 objectbase.h *mgd*

Remove includes of SwarmObject.h and Swarm.h.

1998-04-14 objectbase.h *mgd*

Add //D: documentation comment tag for module.

1998-03-18 objectbase.h *mgd*

Add const char * slot to val_t union.

1998-03-17 objectbase.h *mgd*

Declare swarm_version.

1998-03-01 objectbase.h *mgd*

(MessageProbe): Declare doubleDynamicCallOn:.

1998-02-25 objectbase.h *mgd*

(val_t): Add _uchar.

1998-02-22 objectbase.h *mgd*

(val_t): Define. Change declarations per MessageProbe.h changes below.

1998-02-03 objectbase.h *mgd*

Change header comment to objectbase.h. Include from swarmobject/ to objectbase/.

1998-01-23 objectbase.h *mgd*

In MesageProbe protocol, declare setArg:ToObjectName:. Constify To: argument of setArg:To:. Constify second arugment to dynamicCallOn:resultStorage:.

1997-12-08 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/.

ActivityControlă

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 184)*
CREATABLE *(see page 35)*

Methods

Phase: Using

- – **attachToActivity:** *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.

- – **run**

  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.

- – **stop**

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.

- – **next**

  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.

- – **step**

  The step method sends a step message to the activity if the conditions are appropriate. It causes the execution of a single action.

- – **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.

- – (void)**terminateActivity**

  The terminate method sends the terminate message to the activity, which causes all actions and action groups to be removed from its schedule. Note: if terminate is sent to an activity and subsequently, a run is attempted on that activity, the program will exit with an error.

- – **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.

- – **getStatus**

  The getStatus method returns the status of the activity.

CompleteProbeMapă

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 181)*
CREATABLE *(see page 35)*

Methods

None

CompleteVarMapă

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 181)*
CREATABLE *(see page 35)*

Methods

None

CustomProbeMapă

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 181)*
CREATABLE *(see page 35)*

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 delim-
  ited 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 delim-
ited string list to be added *after* the CustomProbeMap has been created
```

DefaultProbeMapă

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 181)*
CREATABLE *(see page 35)*

Methods

None

EmptyProbeMapă

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 173)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

- + **create:** (id <Zone>)*aZone* **forClass:** (Class)*aClass*
  `Convenience method for creating an EmptyProbeMap`

MessageProbeă

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 177)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

- – **setProbedSelector:** (SEL)*aSel*
  `The setProbedSelector: method sets the message to be probed given the selector.`
- + **create:** (id <Zone>)*aZone* **setProbedSelector:** (SEL)*aSel*

Convenience factory method for common case.

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

- – (BOOL)**isResultId**
  The isResultId method returns 1 if the return value of the message is of type object, and returns 0 otherwise.
- – (BOOL)**isArgumentId:** (int)*which*
  The isArgumentId: method returns 1 if a given argument of the message is of type object, and returns 0 otherwise.
- – (const char *)**getProbedMessage**
  The getProbedMessage method returns the string matching the identifier of the message being probed.
- – (int)**getArgCount**
- – (val_t)**getArg:** (int)*which*
  The getArg: method returns the argument type for a given index.
- – (const char *)**getArgName:** (int)*which*
  The getArgName: method returns a string representation of the argument key with the given index.
- – (BOOL)**getHideResult**
  The getHideResult method returns 1 if the result field is "hidden".
- – **setArg:** (int)*which* **ToString:** (const char *)*what*
  The setArg:ToString: method sets the nth argument of the message. The argument must be provided in string form.
- – (val_t)**dynamicCallOn:** *target*
  The dynamicCallOn: method generates a dynamic message call on the target object.
- – (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.

- – (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.

- – (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 *.

- – **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.

Probeă

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 184)*

ProbeConfig *(see page 178)*

Methods

Phase: Creating

- – **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

- – **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.
- – **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.

Phase: Using

- – **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.
- – (Class)**getProbedClass**
  The getProbedClass method returns the class of the object the probe points at as a Class pointer.
- – (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.

ProbeConfigă

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

- – **setObjectToNotify:** *anObject*
- – **getObjectToNotify**

ProbeLibraryă

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 37)*
Drop *(see page 45)*
ProbeConfig *(see page 178)*
CREATABLE *(see page 35)*

Methods

Phase: Using

- – **setDisplayPrecision:** (int)*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.
- – (int)**getDisplayPrecision**
  The getDisplayPrecision method gets the current display precision set in the ProbeLibrary instance.
- – **setSavedPrecision:** (int)*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.
- – (int)**getSavedPrecision**
  The getSavedPrecision method gets the current saved precision set in the ProbeLibrary instance.
- – (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.
- – (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.
- – (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 mes-

sages). The current implementation of ProbeLibrary does not cache CompleteProbeMaps.

- – (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 <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 <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.
- – **setProbeMap:** *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.

ProbeMapă

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 184)*
ProbeConfig *(see page 178)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

- – **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

- – (int)**getNumEntries**
  The getNumEntries method returns the number of probes in the ProbeMap.
- – (Class)**getProbedClass**
  The getProbedClass method returns the class of the object that the set of probes that constitute the probe map points at.
- – (id <VarProbe>)**getProbeForVariable:** (const char *)*aVariable*
  The getProbeForVariable: method returns the Probe corresponding to the given variable name.
- – (id <MessageProbe>)**getProbeForMessage:** (const char *)*aMessage*
  The getProbeForMessage: method returns the Probe corresponding to the specified message name.
- – **addProbe:** *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.
- – **addProbeMap:** *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.
- – **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.

- **- 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.

- **- dropProbeMap:** *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.

- **- 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.

- **- 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.

Swarmă

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 150)*
CREATABLE *(see page 35)*

Methods

Phase: Using

- `- buildObjects`
  `Override this to let your Swarm create the objects that it  contains.`
- `- buildActions`
  `Override this to let your Swarm build its actions.`
- `- (id <Activity>)activateIn: (id <Swarm>)swarmContext`
  `Override this to activate any actions you built in buildAc- tions. Note, you must activate yourself first before you can activate ac- tions inside you.`

  Example -activateIn: #1

```
[super activateIn: swarmContext];
[fancySchedule activateIn: self];
return [self getSwarmActivity];
```

- `- (id <ProbeMap>)getProbeMap`
  `Needed to support probing of Swarms.`
- `- (id <ProbeMap>)getCompleteProbeMap`
  `Needed to support probing of Swarms.`
- `- (id <VarProbe>)getProbeForVariable: (const char *)aVariable`
  `Needed to support probing of Swarms.`

SwarmObjectă

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 37)*
Drop *(see page 45)*
CREATABLE *(see page 35)*

Methods

Phase: Using

- `- (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.
- `- (id <ProbeMap>)`**`getCompleteProbeMap`**
  The getCompleteProbeMap method returns a newly created CompleteProbeMap for an object.
- `- (id <VarProbe>)`**`getProbeForVariable:`** `(const char *)`*`aVariable`*
  The getProbeForVariable: method returns the VarProbe indexed in the ProbeMap by the string aVariable.
- `- (id <MessageProbe>)`**`getProbeForMessage:`** `(const char *)`*`aMessage`*
  The getProbeForMessage: method returns the MessageProbe indexed in the ProbeMap by the string aMessage.

VarProbeă

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 177)*
CREATABLE *(see page 35)*

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.
- – **createEnd**

Phase: Setting

- – **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 & drop capability of the objects into the VarProbe field.
- – **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.
- – **setFloatFormat:** (const char *)*format*
  The setFloatFormat: method sets the floating-point format of a GUI display widget when given a sprintf-style formatting string.

Phase: Using

- – (const char *)**getProbedVariable**
  The getProbedVariable method returns a string matching the identifier of variable being probed.

- – (BOOL)**getInteractiveFlag**
  The getInteractiveFlag method returns the interactivity state of the VarProbe.
- – (void *)**probeRaw:** *anObject*
  The probeRaw: method returns a pointer to the probed variable.
- – (void *)**probeAsPointer:** *anObject*
  The probeAsPointer: method returns a pointer to the probed variable based on the probeType.
- – **probeObject:** *anObject*
  A field probed with probeAsObject: must be an object.
- – (int)**probeAsInt:** *anObject*
  The probeAsInt: method returns a pointer to the probed variable as an integer.
- – (double)**probeAsDouble:** *anObject*
  The probeAsDouble: method returns a pointer to the probed variable as a double.
- – (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:** (int)*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.
- – (unsigned)**getRank**
  Returns rank of array, or 0 for scalar objects.
- – (const char *)**getBaseType**
  In the case of arrays, returns the base type.
- – (unsigned *)**getDims**
  Returns a vector equal to length returned by getRank: with the dimensions of the array (major to minor).
- – **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.
- – **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.

- `-` **`setData:`** *`anObject`* **`To:`** `(void *)`*`newValue`*
  The setData:To: method sets the probedVariable using the pointer to the new value.
- `- (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'").

Globals

id <Symbol> DefaultString

ă No description available.

id <Symbol> CharString

ă No description available.

id <Symbol> IntString

ă No description available.

Generală

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.

Typedefs

- val_t struct val { char type; types_t val; }

Globals

id <ProbeLibrary> probeLibrary

ă The global librarian for ProbeMaps.

const char * swarm_version

ă 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 has useful notes (see the Bibliography *(see page 443)*.) 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. User Guide for Beginners

Please see the Randomness for neophytes *(see page 410)*. This section may go away in the future to be relocated in the Swarm User Guide.

3.3. Usage Guide to the Generators

Please see the Usage Guide for Generators *(see page 416)*.

3.4. Usage Guide to the Distributions

Please see the Usage Guide for Distributions *(see page 423)*.

4. Advanced Usage Guide

Please see the Advanced Usage Guide *(see page 426)*.

5. 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.

6. Implementation Notes

Please see the Implementation details *(see page 436)*.

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

1999-02-09 random-app.sgml *alex*

Brought all idrefs into line with naming convention.

1999-02-09 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 random00.sgml *sthomme*

minor textual editing.

1999-02-08 Makefile.am *alex*

(SGML): Add, set to random-app.sgml. (EXTRA_DIST): Add SGML onto the dependencies generated by 'Makefile.rules'.

1999-02-08 random00.sgml *alex*

Moved most content into random-app.sgml APPENDIX.

1999-02-08 random.ent *alex*

Made all ENTITY references to the SGML in the 'extra' subdirectory.

1999-02-08 random-app.sgml *alex*

Add file. Content container for 'Random' APPENDIX.

1999-02-02 random.ent *sthomme*

Replaced references to README.Generators.v075 and README.Distributions.v07 with 15 new references to file entities in extra SUBDIR.

1999-02-02 $ *sthomme*

(SWARMDOCS)/catalog.in: replaced the two old references with the 15 new ones.

1999-02-02 random00.sgml *sthomme*

major textual revision. Expanded outline which imports 15 textual items.

1999-02-02 randommeta.sgml *sthomme*

revised the text. Deleted commented-out section (old file index).

1999-01-25 random00.sgml *alex*

Make all references to $(SWARMHOME)/src/random/docs be to
$(SWARMDOCS)/refbook/random/extra.

1999-01-25 random.ent *alex*

Add references to new text file entitites README.Generators.v075 and README.Distributions.v07.

1999-01-25 random00.sgml *alex*

Use new entities in place of text in NOTE markup.

1999-01-25 Makefile.am *alex*

(SUBDIRS): Add ‘extra’ subdirectory.

1999-01-25 random00.sgml *alex*

Change outdated references of the command line ‘-varySeed’ to ‘–varyseed’.

1999-01-06 random00.sgml *alex*

({Generator,Distribution} Usage Notes): Made SIDEBAR, NOTE markup to overcome the one page limitation of SIDEBARs in print backend.

1998-10-27 random.h *mgd*

Include objectbase.h instead of defobj.h. Include random{vars,defs}.h instead of Random{Vars,Defs}.h.

1998-06-16 Makefile.am *mgd*

Include from refbook/ instead of src/.

1998-06-14 Makefile.am *mgd*

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

1998-06-11 random00.sgml, randomcont.sgml *mgd*

Update IDs to SWARM.module.SGML.type.

1998-06-05 random.ent *mgd*

Use public identifiers.

1998-06-04 Makefile.am *mgd*

(swarm_ChangeLog): Add.

1998-06-02 random.h *mgd*

Add summary and description tags for module.

1998-05-22 Makefile.am *mgd*

New file.

1998-05-22 random.ent.in *mgd*

New file.

1998-05-22 random.ent *mgd*

Removed.

1998-05-21 *mgd*

Begin revision log.

1997-12-07 random.h *mgd*

(InternalState): Likewise.

ACGgenă

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 229)*
CREATABLE *(see page 35)*

Methods

None

BernoulliDistă

Name

`BernoulliDist` — Bernoulli Distribution

Description

A distribution returning YES with a given probability.

Protocols adopted by BernoulliDist

BooleanDistribution *(see page 197)*

CREATABLE *(see page 35)*

Methods

Phase: Creating

- + **create:** (id <Zone>)*aZone* **setGenerator:** *generator* **setProbability:** (double)*p*
  Use this create message if the generator to be attached is a Simple one:
- + **create:** (id <Zone>)*aZone* **setGenerator:** *generator* **setVirtualGenerator:** (unsigned)*vGen* **setProbability:** (double)*p*
  Use this create message if the generator to be attached is a Split one:

Phase: Setting

- – **setProbability:** (double)*p*
  The setProbability: method sets the probability of returning YES.

Phase: Using

- – (double)**getProbability**
  The getProbability method returns the probability of returning YES.
- – (BOOL)**getSampleWithProbability:** (double)*p*
  The getSampleWithProbability: returns a sample YES or NO value.

BooleanDistributionă

Name

`BooleanDistribution` — Boolean Distribution

Description

A probability distribution that returns YES/NO sample values.

Protocols adopted by BooleanDistribution

ProbabilityDistribution *(see page 221)*

Methods

Phase: Using

- – (BOOL)**getBooleanSample**
  `The getBooleanSample method returns a YES or NO sample value.`
- – (int)**getIntegerSample**

C2LCGXgenă

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 234)*
CREATABLE *(see page 35)*

Methods

None

C2MRG3genă

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 229)*
CREATABLE *(see page 35)*

Methods

None

C2TAUS1genă

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 201)*
CREATABLE *(see page 35)*

Methods

None

C2TAUS2genă

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 201)*
CREATABLE *(see page 35)*

Methods

None

C2TAUS3genă

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 201)*
CREATABLE *(see page 35)*

Methods

None

C2TAUSgenă

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 229)*

Methods

None

C3MWCgenă

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 229)*
CREATABLE *(see page 35)*

Methods

None

C4LCGXgenă

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 234)*
CREATABLE *(see page 35)*

Methods

None

DoubleDistributionă

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 221)*

Methods

Phase: Using

- – (double)**getDoubleSample**
  The getDoubleSample method returns a double-precision floating point value.

ExponentialDistă

Name

`ExponentialDist` — Exponential distribuiton

Description

A well-known continuous probability distribution returning doubles.

Protocols adopted by ExponentialDist

DoubleDistribution *(see page 202)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

- + **create:** (id <Zone>)*aZone* **setGenerator:** *generator* **setMean:** (double)*mean*
  Use this create message if the generator to be attached is a Simple one:
- + **create:** (id <Zone>)*aZone* **setGenerator:** *generator* **setVirtualGenerator:** (unsigned)*vGen* **setMean:** (double)*mean*
  Use this create message if the generator to be attached is a Split one:

Phase: Setting

- – **setMean:** (double)*mean*
  The setMean: method sets the mean of the distribution.

Phase: Using

- – (double)**getMean**
  The getMean method returns the mean of the distribution.
- – (double)**getSampleWithMean:** (double)*mean*
  The getSampleWithMean: method returns a sample value from a distribution with the specified mean.

GammaDistă

Name

GammaDist — Gamma distribution

Description

A well-known continuous probability distribution returning doubles

Protocols adopted by GammaDist

DoubleDistribution *(see page 202)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

- \+ **create:** (id <Zone>)*aZone* **setGenerator:** *generator* **setAlpha:** (double)*alpha* **setBeta:** (double)*beta*
  Use this create message if the generator to be attached is a Simple one:
- \+ **create:** (id <Zone>)*aZone* **setGenerator:** *generator* **setVirtualGenerator:** (unsigned)*vGen* **setAlpha:** (double)*alpha* **setBeta:** (double)*beta*
  Use this create message if the generator to be attached is a Split 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)**getAlpha**
  The getAlpha method returns the alpha value.
- – (double)**getBeta**
  The getBeta method returns the beta value.
- – (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.

IntegerDistributionă

Name

IntegerDistribution — Integer Distribution

Description

A probability distribution that returns integer sample values.

Protocols adopted by IntegerDistribution

ProbabilityDistribution *(see page 221)*

Methods

Phase: Using

- – (int)**getIntegerSample**
  The getIntegerSample method returns an integer sample value.

InternalStateă

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)`**`getStateSize`**

  `Specifies the minimum buffer size needed.`

- `– (void)`**`putStateInto:`** `(void *)`*`buffer`*
- `– (void)`**`setStateFrom:`** `(void *)`*`buffer`*
- `– (void)`**`describe:`** *`outStream`*
- `– (const char *)`**`getName`**
- `– (unsigned)`**`getMagic`**

LCG1genă

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 208)*
CREATABLE *(see page 35)*

Methods

None

LCG2genă

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 208)*
CREATABLE *(see page 35)*

Methods

None

LCG3genă

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 208)*
CREATABLE *(see page 35)*

Methods

None

LCGgenă

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 229)*

Methods

None

LogNormalDistă

Name

`LogNormalDist` — Log-Normal distribution

Description

A well-known continuous probability distribution returning doubles.

Protocols adopted by LogNormalDist

Normal *(see page 214)*
CREATABLE *(see page 35)*

Methods

None

MRG5genă

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 211)*
CREATABLE *(see page 35)*

Methods

None

MRG6genă

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 211)*
CREATABLE *(see page 35)*

Methods

None

MRG7genă

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 211)*
CREATABLE *(see page 35)*

Methods

None

MRGgenă

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 229)*

Methods

None

MT19937genă

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 229)*
CREATABLE *(see page 35)*

Methods

None

MWCAgenă

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 229)*
CREATABLE *(see page 35)*

Methods

None

MWCBgenă

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 229)*
CREATABLE *(see page 35)*

Methods

None

MultiSeedă

Name

`MultiSeed` — Internal

Protocols adopted by MultiSeed

None

Methods

Phase: Creating

- `+` **`create:`** `(id <Zone>)`*`aZone`* **`setStateFromSeeds:`** `(unsigned *)`*`seeds`*

Phase: Setting

- `-` **`setStateFromSeeds:`** `(unsigned *)`*`seeds`*

```
 The setStateFromSeeds method initializes the seed depen-
dent part of the  state.
```

Phase: Using

- – (unsigned)**lengthOfSeedVector**
  The lengthOfSeedVector method returns the number of seeds required (the size of the array).
- – (unsigned *)**getMaxSeedValues**
  The getMaxSeedValue method returns the upper limit on the seed value that can be supplied.
- – (unsigned *)**getInitialSeeds**
  The getInitialSeed method returns the generator's starting values.

Normală

Name

`Normal` — Internal

Protocols adopted by Normal

DoubleDistribution *(see page 202)*

Methods

Phase: Creating

- + **create:** (id <Zone>)*aZone* **setGenerator:** *generator* **setMean:** (double)*mean* **setVariance:** (double)*variance*
  Use this create message if the generator to be attached is a Simple one:
- + **create:** (id <Zone>)*aZone* **setGenerator:** *generator* **setVirtualGenerator:** (unsigned)*vGen* **setMean:** (double)*mean* **setVariance:** (double)*variance*
  Use this create message if the generator to be attached is a Split one:

Phase: Setting

- – **setMean:** (double)*mean* **setVariance:** (double)*variance*
  The setMean:setVariance: method sets the mean and the variance of the distribution.

Phase: Using

- – (double)**getMean**
  The getMean method returns the mean of the distribution.
- – (double)**getVariance**
  The getVariance method returns the variance of the distribution.
- – (double)**getStdDev**
  The getStdDev method returns the standard deviation of the distribution.
- – (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.

NormalDistă

Name

`NormalDist` — Normal (Gaussian) distribution

Description

A well-known continuous probability distribution returning doubles.

Protocols adopted by NormalDist

Normal *(see page 214)*
CREATABLE *(see page 35)*

Methods

None

PMMLCG1genă

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 220)*
CREATABLE *(see page 35)*

Methods

None

PMMLCG2genă

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 220)*
CREATABLE *(see page 35)*

Methods

None

PMMLCG3genă

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 220)*

CREATABLE *(see page 35)*

Methods

None

PMMLCG4genă

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 220)*

CREATABLE *(see page 35)*

Methods

None

PMMLCG5genă

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 220)*
CREATABLE *(see page 35)*

Methods

None

PMMLCG6genă

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 220)*
CREATABLE *(see page 35)*

Methods

None

PMMLCG7genă

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 220)*
CREATABLE *(see page 35)*

Methods

None

PMMLCG8genă

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 220)*
CREATABLE *(see page 35)*

Methods

None

PMMLCG9genă

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 220)*
CREATABLE *(see page 35)*

Methods

None

PMMLCGgenă

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 229)*

Methods

None

PSWBgenă

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 229)*
CREATABLE *(see page 35)*

Methods

None

ProbabilityDistributionă

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 184)*
InternalState *(see page 205)*

Methods

Phase: Creating

- + **createWithDefaults:** (id <Zone>)*aZone*
  The createWithDefaults method creates a distribution object with a default set of seeds and parameters, and its own private generator.
- + **create:** (id <Zone>)*aZone* **setGenerator:** *generator*
  Use this create message if the generator to be attached is a Simple one:
- + **create:** (id <Zone>)*aZone* **setGenerator:** *generator* **setVirtualGenerator:** (unsigned)*vGen*
  Use this create message if the generator to be attached is a Split one:
- + **createBegin:** (id <Zone>)*aZone*
- – **createEnd**

Phase: Setting

- – **setGenerator:** *generator*
  Use this message if the generator to be attached is a Simple one:
- – **setGenerator:** *generator* **setVirtualGenerator:** (unsigned)*vGen*
  Use this message if the generator to be attached is a Split one:
- – **reset**
  The reset method resets the currentCount and other state data.

Phase: Using

- – **getGenerator**
  The getGenerator method returns the id of the generator.
- – (unsigned)**getVirtualGenerator**
  The getVirtualGenerator returns the number of the virtual generator used.

- `– (BOOL)`**`getOptionsInitialized`**
  `The getOptionsInitialized returns the value of the parameter.`
- `– (unsigned long long int)`**`getCurrentCount`**
  `The getCurrentCount method returns the count of variates generated.`

RWC2genă

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 229)*
CREATABLE *(see page 35)*

Methods

None

RWC8genă

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 229)*
CREATABLE *(see page 35)*

Methods

None

RandomBitDistă

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 197)*
CREATABLE *(see page 35)*

Methods

Phase: Using

- `- (BOOL)`**`getCoinToss`**
  `The getCoinToss method returns a YES or NO value.`

SCGgenă

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 229)*

CREATABLE *(see page 35)*

Methods

None

SWB1genǎ

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 226)*

CREATABLE *(see page 35)*

Methods

None

SWB2genă

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 226)*
CREATABLE *(see page 35)*

Methods

None

SWB3genă

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 226)*
CREATABLE *(see page 35)*

Methods

None

SWBgenă

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 229)*

Methods

None

Simpleă

Name

`Simple` — Internal

Protocols adopted by Simple

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).

Phase: Using

- – (BOOL)**getAntithetic**
  The getAntithetic method returns the current values of the parameter.
- – (unsigned long long int)**getCurrentCount**
  The getCurrentCount method returns the count of variates generated.
- – **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.

SimpleOută

Name

`SimpleOut` — Internal

Protocols adopted by SimpleOut

None

Methods

Phase: Using

- – (unsigned)**getUnsignedMax**
  The getUnsignedMax method returns the highest value that will ever be returned by -getUnsignedSample (the lowest is 0).
- – (unsigned)**getUnsignedSample**
  The getUnsignedSample method returns a random unsigned integer uniformly distributed over [0,unsignedMax].
- – (float)**getFloatSample**

```
 The getFloatSample method returns a random floating point num-
ber of size float, uniformly dis-
tributed in the range [0.0, 1.0). It uses 1 call to -
getUnsignedSample to fill the mantissa.
```

- – (double)**getThinDoubleSample**

```
 The getThinDoubleSample method returns a random floating point num-
ber of size double, uniformly dis-
tributed in the range [0.0, 1.0). It uses 1 call to -
getUnsignedSample to fill the mantissa.
```

- – (double)**getDoubleSample**

```
 The getDoubleSample method returns a random floating point num-
ber of size double, uniformly dis-
tributed in the range [0.0, 1.0). It uses 2 calls to -
getUnsignedSample to fill the mantissa.
```

- – (long double)**getLongDoubleSample**

```
 The getLongDoubleSample method returns a random floating point num-
ber of size long double, uniformly dis-
tributed in the range [0.0, 1.0). It uses 2 calls to -
getUnsignedSample to fill the man-
tissa. Note: use of this method is not portable between architectures.
```

SimpleRandomGeneratorǎ

Name

`SimpleRandomGenerator` — A Simple (non-split) generator.

Description

This protocol covers all implemented non-split generators.

Protocols adopted by SimpleRandomGenerator

SwarmObject *(see page 184)*
InternalState *(see page 205)*
SimpleOut *(see page 228)*
Simple *(see page 227)*
SingleSeed *(see page 230)*
MultiSeed *(see page 213)*

Methods

None

SingleSeedă

Name

`SingleSeed` — Internal

Protocols adopted by SingleSeed

None

Methods

Phase: Creating

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

Phase: Setting

- – **setStateFromSeed:** (unsigned) *seed*

  The setStateFromSeeds method initializes the seed dependent part of the state.

Phase: Using

- – (unsigned) **getMaxSeedValue**

  The getMaxSeedValue method returns the upper limit on the seed value.

- – (unsigned) **getInitialSeed**

  The getInitialSeed method returns the generator's starting values.

Splită

Name

`Split` — Internal

Protocols adopted by Split

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).
- `-` **`initGenerator:`** `(unsigned)`*`vGen`*

  The initGenerator method resets the state of a virtual generator to the start of segment #0.
- `-` **`initAll`**

  The initAll method resets the state of all the virtual generators to the start of segment #0.

Phase: Using

- `-` `(unsigned)`**`getNumGenerators`**

  The getNumGenerators method returns the current number of virtual generators (A).
- `-` `(unsigned)`**`getNumSegments`**

  The getNumSegments method returns log2(the current number of segments) = v.
- `-` `(unsigned)`**`getSegmentLength`**

  The getSegmentLength method returns log2(the current segment length) = w.

- – (BOOL)**getAntithetic**
  The getAntithetic method returns the current value of the parameter.
- – **restartGenerator:** (unsigned)*vGen*
  The restartGenerator method resets the state of a virtual generator to the start of the current segment.
- – **advanceGenerator:** (unsigned)*vGen*
  The advanceGenerator method resets the state of a virtual generator to the start of the next 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.
- – **restartAll**
  The restartAll method resets the state of all the virtual generators to the start of their current segment.
- – **advanceAll**
  The advanceAll method resets the state of all the virtual generators to the start of their next segment.
- – **jumpAllToSegment:** (unsigned long long int)*seg*
  The jumpAlltoSegment: method resets the state of all the virtual generators to the start of the specified segment.
- – (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.
- – **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.

SplitMultiSeedă

Name

SplitMultiSeed — Internal

Protocols adopted by SplitMultiSeed

None

Methods

Phase: Creating

- + **create:** (id <Zone>)*aZone* **setA:** (unsigned)*A* **setV:** (unsigned)*v* **setW:** (unsigned)*w* **setStateFromSeeds:** (unsigned *)*seeds*

Phase: Setting

- – **setStateFromSeeds:** (unsigned *)*seeds*
  The setStateFromSeeds method initializes the generator from a vector of seed values.

Phase: Using

- – (unsigned)**lengthOfSeedVector**
  The lengthOfSeedVector method returns the number of seeds required.
- – (unsigned *)**getMaxSeedValues**
  The getMaxSeedValue method returns a vector of highest allowable seed values. The lowest value is always 1 (0 is not a valid seed).
- – (unsigned *)**getInitialSeeds**
  The getInitialSeed method returns the generator's starting seeds.

SplitOută

Name

`SplitOut` — Internal

Protocols adopted by SplitOut

None

Methods

Phase: Using

- – (unsigned)**getUnsignedMax**

  The getUnsignedMax method returns the highest value that will ever be returned by -getUnsignedSample (the lowest is 0).

- – (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.

- – (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.

- – (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.

- – (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.

- – (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.

SplitRandomGeneratoră

Name

`SplitRandomGenerator` — A Split generator.

Description

This protocol covers the implemented split generators (C2LCGX and C4LCGX.)

Protocols adopted by SplitRandomGenerator

SwarmObject *(see page 184)*

InternalState *(see page 205)*

SplitOut *(see page 233)*

Split *(see page 230)*

SplitSingleSeed *(see page 235)*

SplitMultiSeed *(see page 232)*

Methods

None

SplitSingleSeedă

Name

`SplitSingleSeed` — Internal

Protocols adopted by SplitSingleSeed

None

Methods

Phase: Creating

- `+ ` **`create:`** `(id <Zone>)`*`aZone`* **`setA:`** `(unsigned)`*`A`* **`setV:`** `(unsigned)`*`v`* **`setW:`** `(unsigned)`*`w`* **`setStateFromSeed:`** `(unsigned)`*`seed`*

Phase: Setting

- `- ` **`setStateFromSeed:`** `(unsigned)`*`seed`*

  `The setStateFromSeed method initializes the generator from a single seed value. If more are needed, they are generated internally.`

Phase: Using

- `- (unsigned)`**`getMaxSeedValue`**

  `The getMaxSeedValue method returns the highest allowable seed value. The lowest value is 1 (0 is not a legal seed value).`

- `- (unsigned)`**`getInitialSeed`**

  `The getInitialSeed method returns the generator's starting seed.`

TGFSRgenă

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 229)*

Methods

None

TT403genă

Name

`TT403gen` — A single long generator recommended for use.

Description

A single long generator recommended for use.

Protocols adopted by TT403gen

TGFSRgen *(see page 236)*
CREATABLE *(see page 35)*

Methods

None

TT775genă

Name

`TT775gen` — A single long generator recommended for use.

Description

A single long generator recommended for use.

Protocols adopted by TT775gen

TGFSRgen *(see page 236)*
CREATABLE *(see page 35)*

Methods

None

TT800genă

Name

`TT800gen` — A single long generator recommended for use.

Description

A single long generator recommended for use.

Protocols adopted by TT800gen

TGFSRgen *(see page 236)*
CREATABLE *(see page 35)*

Methods

None

UniformDoubleDistă

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 202)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

- + **create:** (id <Zone>)*aZone* **setGenerator:** *generator* **setDoubleMin:** (double)*minValue* **setMax:** (double)*maxValue*
  Use this create message if the generator to be attached is a Simple one:
- + **create:** (id <Zone>)*aZone* **setGenerator:** *generator* **setVirtualGenerator:** (unsigned)*vGen* **setDoubleMin:** (double)*minValue* **setMax:** (double)*maxValue*
  Use this create message if the generator to be attached is a Split 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)**getDoubleMin**
  The getDoubleMin method returns the minimum floating point value in the specified range.
- - (double)**getDoubleMax**
  The getDoubleMax method returns the maximum floating point value in the specified range.
- - (double)**getDoubleWithMin:** (double)*minValue* **withMax:** (double)*maxValue*
  The getDoubleWithMin:withMax: method returns a floating point value within the range [min, max).

UniformIntegerDistă

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 205)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

- + **create:** (id <Zone>)*aZone* **setGenerator:** *generator* **setIntegerMin:** (int)*minValue* **setMax:** (int)*maxValue*
  Use this create message if the generator to be attached is a Simple one:
- + **create:** (id <Zone>)*aZone* **setGenerator:** *generator* **setVirtualGenerator:** (unsigned)*vGen* **setIntegerMin:** (int)*minValue* **setMax:** (int)*maxValue*
  Use this create message if the generator to be attached is a Split 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)**getIntegerMin**
  The getIntegerMin method returns the minimum integer value.
- − (int)**getIntegerMax**
  The getIntegerMax method returns the maximum integer value.
- − (int)**getIntegerWithMin:** (int)*minValue* **withMax:** (int)*maxValue*
  The getIntegerWithMin:withMax: returns an integer within the interval [min, max].

UniformUnsignedDistă

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 241)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

- + **create:** (id <Zone>)*aZone* **setGenerator:** *generator* **setUnsignedMin:** (unsigned)*minValue* **setMax:** (unsigned)*maxValue*

  Use this create message if the generator to be attached is a Simple one:

- + **create:** (id <Zone>)*aZone* **setGenerator:** *generator* **setVirtualGenerator:** (unsigned)*vGen* **setUnsignedMin:** (unsigned)*minValue* **setMax:** (unsigned)*maxValue*

  Use this create message if the generator to be attached is a Split 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)**getUnsignedMin**

  The getUnsignedMin method returns the minimum unsigned value.

- − (unsigned)**getUnsignedMax**

  The getUnsignedMax method returns the maximum unsigned value.

- − (unsigned)**getUnsignedWithMin:** (unsigned)*minVal* **withMax:** (unsigned)*maxVal*

  The getUnsignedWithMin:withMax: returns an unsigned integer within the interval [min, max].

UnsignedDistributionă

Name

`UnsignedDistribution` — Unsigned Distribution

Description

A probability distribution that returns non-negative integer sample values.

Protocols adopted by UnsignedDistribution

ProbabilityDistribution *(see page 221)*

Methods

Phase: Using

- `– (unsigned)`**`getUnsignedSample`**
  `The getUnsignedSample method returns a non-negative integer sample value.`

Generală

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 260)* 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 260)*. 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

1999-07-04 simtools.h *alex*

(initSwarmAppArguments): Fixed macro, STRINGIFY(APPNAME) -> APPNAME_STRING.

1999-06-22 simtools.h *mgd*

Reflect these changes.

1999-06-20 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.

1999-06-20 simtools.h *alex*

(_initSwarm_, STRINGIFY, STRINGIFYSYM, APPNAME_STRING, APPVERSION_STRING, BUGADDRESS_STRING): Remove, hide from user in new include'd initSwarm.h.

1999-06-09 simtools.h *mgd*

Stringify application name, version, and bug address. (Don't require quoting from make.)

1999-06-08 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.

1999-05-28 simtools.h *mgd*

Include externvar.h.

1999-05-27 simtools.h *mgd*

Use 'externvar' for swarmGUIMode.

1999-02-25 simtools.h *mgd*

Add CREATABLE tags to all non-abstract protocols.

1999-02-06 simtools.h *mgd*

(InFile): Add a warning about error return behavior.

1998-12-10 simtools.h *vjojic*

Remove ListShuffler protocol and ListShuffler class definition

1998-08-22 simtools.h *mgd*

(ListShuffler): New protocol.

1998-08-19 simtools.h *mgd*

(ObjectLoader): Add it.

1998-07-07 simtools.h *mgd*

(ActiveOutFile): Remove protocol and class object.

1998-06-16 Makefile.am *mgd*
Include from refbook/ instead of src/.
1998-06-14 Makefile.am *mgd*
(MODULE): New variable. Include Makefile.rules from src. Remove everything else.
1998-06-11 simtools00.sgml, simtoolscont.sgml *mgd*
Update IDs to SWARM.module.SGML.type.
1998-06-05 simtools.ent *mgd*
Use public identifiers.
1998-06-04 Makefile.am *mgd*
(swarm_ChangeLog): Add.
1998-06-02 simtools.h *mgd*
Update documentation tags.
1998-05-22 Makefile.am *mgd*
New file.
1998-05-22 simtools.ent.in *mgd*
New file.
1998-05-22 simtools.ent *mgd*
Removed.
1998-05-21 *mgd*
Begin revision log.
1998-04-15 simtools.h *mgd*
Move tagged documentation to the respective protocols definition.
1998-04-05 simtools.h *mgd*
Declare new function initSwarmArguments.
1998-02-25 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.
1998-01-26 simtools.h *mgd*
Likewise.
1998-01-23 simtools.h *mgd*
Don't include tkobjc.h.
1998-01-19 simtools.h *mgd*
Add copyright header.
1998-01-14 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.

1998-01-13 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.

1998-01-12 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.

1998-01-12 simtools.h *mgd*

Add dropProbeDisplaysFor: to the ProbeDisplayManager protocol.

1998-01-11 simtools.h *mgd*

Declare buildWindowGeometryRecordName. New protocol WindowGeometryRecordName, derive ActionCache, and GUISwarm from it.

1998-01-05 simtools.h *mgd*

(Action): Rename setControlPanelGeometryRecordName to setWindowGeometryRecordName.
(ControlPanel): Remove setControlPanelGeometryRecordName.

1997-12-18 simtools.h *mgd*

(ActionCache): Make setScheduleContext a Using phase method. Add setControlPanelGeometryRecordName. (InFile): Add getline to InFile protocol. Reformatting throughout (remove gratuitous space).

1997-12-17 simtools.h *mgd*

Add setWindowGeometryRecordName to ProbeDisplay protocol.

1997-12-09 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.

1997-12-06 simtools.h *mgd*

(CompleteProbeDisplay): Add getMarkedForDropFlag to protocol.

1997-12-03 simtools.h *mgd*

(ProbeDisplayManager): Add setDropImmediatelyFlag.

1997-11-30 simtools.h *alex*

Added AppendFile and added reverseOrder method to QSort protocol.

1997-11-28 simtools.h *mgd*

Make Object{Loader,Saver} fromFileNamed: arguments const.

1997-11-28 simtools.h *mgd*

Add methods below to ProbeDisplayManager protocol.

1997-11-19 simtools.h *mgd*

(library-wide): Brought simtools library into conformance with protocol interface standard.

AppendFile [Deprecated]ă

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 254)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

- `+ `**`create:`**` `*`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.`

- `+ `**`create:`**` `*`aZone`*` `**`withName:`**` (const char *)`*`theName`*

  `for backward compatibility`

InFile [Deprecated]ă

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 184)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

- + **create:** *aZone* **setName:** (const char *)*theName*
  This is the create method for In-Files, where theName is, of course the name of the file to open.
- + **create:** *aZone* **withName:** (const char *)*theName*
  for backward compatibility

Phase: Using

- − (int)**getWord:** (char *)*aWord*
  The getWord: method returns a string that does not contain spaces, tabs, and newlines.
- − (int)**getLine:** (char *)*aLine*
  The getLine: method loads the argument string with the characters up to, but not including a newline character.
- − (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)**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)**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)**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)**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)**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)**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)**unGetChar:** (char)*aChar*
  Returns a character for re-reading.
- – (int)**skipLine**
  Skips a line.
- – (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.

NSelectă

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 184)*

CREATABLE *(see page 35)*

Methods

Phase: Using

• + (void)**select:** (int)*n* **from:** *aCollection* **into:** *bCollection*

```
 The select:from:into: method selects exactly N elements from a collec-
tion without repetition into another collection.  The selection algo-
rithm is from Knuth.
```

ObjectLoader [Deprecated]ă

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 184)*

CREATABLE *(see page 35)*

Methods

Phase: Creating

- + **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.
- + **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* **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* **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.

Phase: Setting

- – **setFileObject:** *aFileObject*
  The setFileObject: method sets the source fileObject which the instance of the ObjectLoader class should use by sending it this message.
- – **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.

Phase: Using

- – **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.
- – **updateCache:** *exampleTarget*

```
The updateCache: method should be called if an ObjectLoader in-
stance is  going to initialize a large number of ob-
jects from the same class.
```

ObjectSaver [Deprecated]ă

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 184)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

- + **save**: *anObject* **to**: *aFileObject*

```
The save:to: method saves the entire target object without actually  re-
turning 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  vari-
ables specified in a template from anObject without actually  return-
ing an instance of ObjectSaver to the user.
```

- + **save**: *anObject* **toFileNamed**: (const char *)*aFileName*

```
The save:toFileNamed: method saves the entire target ob-
ject 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.

Phase: Setting

- `- setFileObject: aFileObject`
  The setFileObject: method sets the target fileObject which the instance of the ObjectSaver class should use.
- `- 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.

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]ă

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 184)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

- + **create:** *aZone* **setName:** (const char *)*theName*
  The create:setName: method opens a file named theName and creates an Outfile object.
- + **create:** *aZone* **withName:** (const char *)*theName*
  for backward compatibility

Phase: Using

- – **putString:** (const char *)*aString*
  The putString: method takes an instance of type string 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.
- – **putUnsigned:** (unsigned)*anUnsigned*
  The putUnsigned: method takes an instance of type unsigned 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.
- – **putUnsignedLong:** (unsigned long)*anUnsLong*
  The putUnsignedLong: method takes an instance of type unsigned long 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.
- – **putFloat:** (float)*aFloat*
  The putFloat: method takes an instance of type float and writes it into the open file.
- – **putChar:** (char)*aChar*
  The putChar: method takes an instance of type char and writes it into the open file.
- – **putNewLine**
  The putNewline method writes a newline into the open file.
- – **putTab**
  The putTab method writes a tab into the open file.

- – (void)**drop**
  The drop method closes the open file. This method must be called to close the file.

QSortă

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 184)*
CREATABLE *(see page 35)*

Methods

Phase: Using

- + (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.
- + (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`* **`us-`**
  `ing:` `(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)`**`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.

UNameǎ

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 184)*

CREATABLE *(see page 35)*

Methods

Phase: Creating

- + **create:** *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.
- + **create:** *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.
- − **setBaseName:** (const char *)*aString*
  The setBaseName: method is used to set the base name given a const char *.
- − **setBaseNameObject:** *aStringObject*
  The setBaseNameObject: method is used to set the base name given an object of class String.

Phase: Using

- − (const char *)**getNewName**
  The getNewName method generates a new name as a character string.
- − **getNewNameObject**
  The getNewNameObject generates a new name as a String Object.
- − **resetCounter**
  Resets the counter used as a suffix in the unique names generated.

Generală

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.

Globals

BOOL swarmGUIMode

ă Flag for whether we're in graphics mode or not. Default is NO.

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 260)* 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

1999-08-21 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.

1999-07-20 simtoolsgui.h *vjojic*

Make CommonProbeDisplay protocol CREATABLE. Make SingleProbeDisplay protocol CREATABLE. Add SimpleProbeDisplay protocol and make it CREATABLE.

1999-05-28 simtoolsgui.h *mgd*

Include externvar.h.

1999-05-27 simtoolsgui.h *mgd*

Use ‘externvar’ for external variables.

1999-04-25 simtoolsgui.h *alex*

(GUIComposite): Add compliance to SwarmObject protocol. Revert this change, since protocol is abstract, confirmation added to instantiatable protocols.

1999-03-31 simtoolsgui.h *vjojic*

Protocol GUISwarm inherits protocol Swarm.

1999-03-30 simtoolsgui.h *vjojic*

Add methods getActionCache and getControlPanel to GUISwarm.

1999-02-25 simtoolsgui.h *mgd*

Add CREATABLE tags to all non-abstract protocols.

1999-01-30 simtoolsgui.h *mgd*

(ProbeDisplayManager): Declare -getDropImmediateFlag.

1998-12-30 simtoolsgui.h *mgd*

(ActionCache, GUISwarm): Declare -drop.

1998-09-02 simtoolsgui.h *mgd*

(CommonProbeDisplay): Declare -getTopLevel.

1998-08-17 simtoolsgui.h *mgd*

(SingleProbeDisplay): Add //S and //D.

1998-07-14 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.

1998-07-13 simtoolsgui.h *mgd*

Add MultiVarProbeWidget protocol and class object.
1998-07-09 simtoolsgui.h *alex*
(MessageProbeWidget): Add phase tags.
1998-07-07 simtoolsgui.h *mgd*
(ActiveGraph): Remove protocl and class object.
1998-07-06 simtoolsgui.h *mgd*
(MessageProbeWidget): New protocol and class object.
1998-06-16 simtoolsgui00.sgml *alex*
Added missing description of activity.h dependency.
1998-06-16 Makefile.am *mgd*
Include from refbook/ instead of src/.
1998-06-14 Makefile.am *mgd*
(MODULE): New variable. Include Makefile.rules from src. Remove everything else.
1998-06-11 simtoolsgui00.sgml, simtoolsguicont.sgml *mgd*
Update IDs to SWARM.module.SGML.type.
1998-06-05 simtoolsgui.ent *mgd*
Use public identifiers.
1998-06-04 Makefile.am *mgd*
(swarm_ChangeLog): Add.
1998-06-02 simtoolsgui.h *mgd*
Update documentation tags. Prefix global references with extern.
1998-05-27 simtoolsgui.h *mgd*
Include activity.h.
1998-05-22 Makefile.am *mgd*
New file.
1998-05-22 simtoolsgui.ent.in *mgd*
New file.
1998-05-22 simtoolsgui.ent *mgd*
Removed.
1998-05-21 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.
1998-05-21 *mgd*
Begin revision log.
1998-05-06 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.

1998-05-05 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.

1998-04-15 simtoolsgui.h *mgd*

Add comment tags with documentation.

ActionCacheă

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 267)*
RETURNABLE *(see page 57)*

Methods

Phase: Creating

- – **setControlPanel:** *cp*
- – **createEnd**
- – **createProcCtrl**

Phase: Using

- – **setScheduleContext:** *context*
- – **insertAction:** *actionHolder*
- – **deliverActions**
- – **sendActionOfType:** (id <Symbol>)*type* **toExecute:** (const char *)*cmd*
- – **sendStartAction**
- – **sendStopAction**
- – **sendStepAction**
- – **sendNextAction**
- – **sendQuitAction**

- – **verifyActions**
- – **getPanel**
- – **doTkEvents**
- – **waitForControlEvent**
- – (void)**drop**

CommonProbeDisplayă

Name

`CommonProbeDisplay` — A protocol underlying ProbeDisplay and CompleteProbeDisplay

Description

This protocol provides the common interface to all kinds of ProbeDisplays.

Protocols adopted by CommonProbeDisplay

WindowGeometryRecordName *(see page 277)*

Methods

Phase: Using

- – **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.
- – (BOOL)**getMarkedForDropFlag**
- – **getTopLevel**

CompleteProbeDisplayă

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 277)*
RETURNABLE *(see page 57)*

Methods

None

CompositeWindowGeometryRecordNameă

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 277)*

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`*`)`

ControlPanelă

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 184)*
RETURNABLE *(see page 57)*

Methods

Phase: Creating

- `-` **`createEnd`**

Phase: Using

- – **getState**
  Get the current button state of the controlpanel. Is one of Control-StateRunning, ControlStateStopped, ControlStateStepping, ControlStateNext-Time, or ControlStateQuit.
- – **setState:** *s*
- – **startInActivity:** *activityID*
- – **setStateRunning**
  Sets the state to 'running'.
- – **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).
- – **setStateStepping**
  Stop the running activity, and then set state to 'ControlStateStepping'.
- – **setStateQuit**
  Terminate activities, and set state to 'ControlStateQuit'.
- – **setStateNextTime**
  Stop the running activity, and then set state to 'ControlStateNextTime'.
- – **setStateSave**
  Saves the objects that are registered for archiving.

GUICompositeă

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 267)*

Methods

Phase: Using

- `-` **`enableDestroyNotification:`** *`notificationTarget`* **`notification-Method:`** `(SEL)`*`notificationMethod`*
- `-` **`disableDestroyNotification`**

GUISwarmă

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 183)*

WindowGeometryRecordName *(see page 277)*

CREATABLE *(see page 35)*

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 <ActionCache>)**getActionCache**
- − (id <ControlPanel>)**getControlPanel**
- − (void)**drop**

MessageProbeWidgetă

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 35)*

Methods

Phase: Creating

- + **createBegin:** (id <Zone>)*aZone*
- − **setParent:** *parent*
- − **setObject:** *object*
- − **createEnd**
- − **setProbe:** *probe*

Phase: Using

- − **pack**

MultiVarProbeDisplayă

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 266)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

- − **setObjectList:** (id <List>)*objectList*
  `Sets the list of objects to display.`
- − **setProbeMap:** (id <ProbeMap>)*probeMap*
  `Sets the probe map (i.e. list of fields) to display.`
- − **setObjectNameSelector:** (SEL)*objectNameSelector*
  `Sets the selector to send for labeling the object.`
- + **createBegin:** (id <Zone>)*aZone*
- − **createEnd**

Phase: Using

- − (void)**drop**

MultiVarProbeWidgetă

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

CREATABLE *(see page 35)*

Methods

Phase: Creating

- `+` **`createBegin:`** `(id <Zone>)`*`aZone`*
- `-` **`createEnd`**
- `-` **`setParent:`** *`parent`*
- `-` **`setFieldLabelingFlag:`** `(BOOL)`*`labelingFlag`*
  `Determines if the fields (probes) are labeled (horizontal).`
- `-` **`setObjectNameSelector:`** `(SEL)`*`objectNameSelector`*
  `Sets the method to use get the label for each object (vertical).`
- `-` **`setObjectList:`** `(id <List>)`*`objectList`*
  `Sets the objects to show on the widget.`
- `-` **`setProbeMap:`** `(id <ProbeMap>)`*`probeMap`*
  `Sets the fields (probes) to show on the widget.`

Phase: Using

- `-` **`update`**
- `-` **`pack`**
- `-` `(void)`**`drop`**

ProbeDisplayă

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 277)*
CREATABLE *(see page 35)*

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.`
- `-` **`createEnd`**

Phase: Using

- `- (id <ProbeMap>)`**`getProbeMap`**
  `Gets the probedMap.`

ProbeDisplayManageră

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 184)*
CREATABLE *(see page 35)*

Methods

Phase: Using

- `– (BOOL)`**`getDropImmediatelyFlag`**
- `– (id <ProbeDisplay>)`**`createProbeDisplayFor:`** *`anObject`*
- `– (id <ProbeDisplay>)`**`createArchivedProbeDisplayFor:`** *`anObject`* **`variableName:`** `(const char *)`*`variableName`*
- `– (id <ProbeDisplay>)`**`createDefaultProbeDisplayFor:`** *`anObject`*
- `– (id <ProbeDisplay>)`**`createArchivedDefaultProbeDisplayFor:`** *`anObject`* **`variableName:`** `(const char *)`*`variableName`*
- `– (id <CompleteProbeDisplay>)`**`createCompleteProbeDisplayFor:`** *`anObject`*
- `– (id <CompleteProbeDisplay>)`**`createArchivedCompleteProbeDisplayFor:`** *`anObject`* **`variableName:`** `(const char *)`*`variableName`*
- `–` **`addProbeDisplay:`** *`probeDisplay`*
  Add a probe display to be managed by the ProbeDisplayManager.
- `–` **`removeProbeDisplay:`** *`probeDisplay`*
  Remove a probe display from management by the ProbeDisplayManager.
- `–` **`dropProbeDisplaysFor:`** *`anObject`*
  Remove and drop probe displays associated with a given object.
- `–` **`update`**
  This method will recursively send an update message to all the Probe Displays managed by the ProbeDisplayManager.
- `–` **`setDropImmediatelyFlag:`** `(BOOL)`*`dropImmediateFlag`*

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

SimpleProbeDisplayă

Name

SimpleProbeDisplay —

Description

Protocols adopted by SimpleProbeDisplay

SingleProbeDisplay *(see page 277)*
RETURNABLE *(see page 57)*

Methods

Phase: Creating

- – **setProbeMap:** *probeMap*

- – **createEnd**

Phase: Using

- – (id <ProbeMap>)**getProbeMap**

SingleProbeDisplayă

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 266)*

Methods

Phase: Creating

- – **setProbedObject:** *anObject*

  This method must be called.

- – **createEnd**

Phase: Using

- – **getProbedObject**

  Gets the probed object.

WindowGeometryRecordNameă

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

- `-` **`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.`

Generală

Name

`simtoolsgui` — GUI-related features for simulation.

Functions

- `void` **`initSimtoolsGUI`**`(void)`
  `Initialize the library and create a ProbeDisplayManager.`

Globals

id <Symbol> Control

ă Type Symbols for ActionCache

id <Symbol> Probing
ă Type Symbols for ActionCache

id <Symbol> Spatial
ă Type Symbols for ActionCache

id <Symbol> InvalidActionType
ă Error Symbols for ActionCache

id <Symbol> ActionTypeNotImplemented
ă Error Symbols for ActionCache

id <ProbeDisplayManager> probeDisplayManager
ă Manager that keeps track of active probes to be updated

id ControlStateRunning
ă State Symbols for the ControlPanel.

id ControlStateStopped
ă State Symbols for the ControlPanel.

id ControlStateStepping
ă State Symbols for the ControlPanel.

id ControlStateNextTime
ă State Symbols for the ControlPanel.

id ControlStateQuit
ă 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.

Revision History

1999-07-19 gui.h *mgd*

(Form): Change Boolean: argument type to unsigned * (from BOOL *). (InputWidget): Likewise.

1999-06-04 gui.h *mgd*

(Colormap): New method unsetColor:. Thanks to Ken Cline.

1999-04-30 gui.h *mgd*

(Histogram): Change setNumBins: to setBinCount:. Change type of setActiveOutlierText:count: arguments to unsigned.

1999-03-19 gui.h *mgd*

(WindowGeometryRecord): Don't tag as CREATABLE. Add @class for CheckButton, Form, CanvasIteim, NodeItem, and Rectangle.

1999-02-25 gui.h *mgd*

Add CREATABLE tags for all non-abstract protocols.

1999-02-22 gui.h *mgd*

Merge internal protocols into advertised protocols.

1999-02-07 gui.h *mgd*

(_ArchivedGeometryWidget): Add archiver argument to updateArchiver:. Version 1.4.1.

1999-01-05 gui.h *mgd*

(_Histogram): Change argument to setNumBins: to unsigned. Add count argument to setColors and setLabels.

1998-11-17 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.

1998-11-16 gui.h *mgd*

(_WindowGeometryRecord): Adopt Serialization protocol; don't redeclare methods.

1998-11-15 gui.h *mgd*

Remove creating phase +in:. Rename in: and out: to lispin: and lispout:.

1998-09-29 gui.h *mgd*

(_Histogram): -setNumBins:, -setLabels:, -setColors:, +createBegin:, createEnd: Split out setNumPoints:Labels:Colors: into different methods.

1998-09-27 gui.h *mgd*

(_Canvas, _Frame): Move Frame's assertGeometry to Canvas as checkGeometry:.

1998-09-27 gui.h *alex*

(ScheduleItem): Fixed incorrect documentation markup of 'Description' entry for protocol.

1998-09-24 gui.h *mgd*

(_Canvas): Use addWidget:X:Y:centerFlag: and removeWidget:.

1998-09-23 gui.h *mgd*
(_ScheduleItem): Add trigger:X:Y:. (_CompleteProbeDisplayLabel): Remove setProbeDisplayManager: and rename setProbeDisplay: to setTargetWidget:.
1998-09-22 gui.h *mgd*
(_ScheduleItem): Add at:owner:widget:x:y:.
1998-09-21 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:.
1998-09-16 gui.h *mgd*
(Pixmap): New method setDecorationsFlag:.
1998-09-02 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:.
1998-08-18 gui.h *mgd*
(Pixmap): Add setDirectory:.
1998-08-17 gui.h *mgd*
(Pixmap): Revert create:widget: and create:file: addition and use setFile: and setWidget: instead.
1998-08-05 gui.h *mgd*
(_Pixmap): Add create:widget: and create:file:. Move setRaster: to using phase (still need to make implementation synchronize). Add save:.
1998-07-21 gui.h *mgd*
(Graph): Add setRangesXMin:Max:.
1998-07-20 gui.h *mgd*
(Graph): Arguments to setScaleModeX:Y: are now boolean, not integers.
1998-07-14 gui.h *mgd*
(_VarProbeEntry): Replace setProbeType: with setVarProbe:. Add getVarProbe.
1998-07-06 gui.h *mgd*
(_GraphElement): New method -setWidth:.
1998-06-30 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.
1998-06-16 guimeta.sgml *alex*
Removed redundant text from ABSTRACT.
1998-06-16 gui00.sgml *alex*
Added objectbase.h to list of dependencies.
1998-06-16 Makefile.am *mgd*

Include from refbook/ instead of src/.

1998-06-14 Makefile.am *mgd*

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

1998-06-13 gui.h *mgd*

Make separate ifdefs for GUI macros to work with the protocol script. Use #if 0 for awt.

1998-06-11 gui00.sgml, guicont.sgml *mgd*

Update IDs to SWARM.module.SGML.type.

1998-06-05 gui.ent *mgd*

Use public identifiers.

1998-06-04 gui.h *alex*

Fixed (#if 1) around the tk macros - erroneosly set to (#if 0).

1998-06-04 Makefile.am *mgd*

(swarm_ChangeLog): Add.

1998-06-02 gui.h *mgd*

Update documentation tags. Use #if 0 for macros to avoid multiple definitions in protocol.el.

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

Fix typo in revhistory.

1998-05-22 Makefile.am *mgd*

New file.

1998-05-22 gui.ent.in *mgd*

New file.

1998-05-22 gui.ent *mgd*

Removed.

1998-05-21 gui.h *mgd*

(_ArchivedGeometryWidget): Declare -loadWindowGeometryRecord, -registerAndLoad, +createBegin:, -updateArchiver, -getDestroyedFlag, and -drop.

1998-05-21 *mgd*

Begin revision log.

1998-05-12 gui.h *mgd*

InputWidget: new protocol. Make CheckButton and Entry inherit from InputWidget, not _InputWidget.

1998-05-05 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.

1998-04-22 gui.h *mgd*

Add documentation tags to all protocols. Bring in sync with tkobjc changes.

1998-04-16 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.

1998-04-12 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.

1998-04-09 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.

1998-02-26 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:.

1998-02-23 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.

1998-02-19 gui.h *mgd*

Change javaobjc to awtobjc throughout. Use initAWTObjc instead of initJavaObjc.

1998-02-17 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:.

1998-01-26 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.

1998-01-25 gui.h *mgd*

Remove some USING tags.

1998-01-24 gui.h *mgd*

New file.

1998-01-24 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.

ArchivedGeometryWidgetă

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 319)*

Methods

Phase: Creating

- + **createBegin:** (id <Zone>)*aZone*
- – **setWindowGeometryRecordName:** (const char *)*recordName*
  Called to set a name for archiving.
- – **loadWindowGeometryRecord**
- – **registerAndLoad**

Phase: Using

- – **updateArchiver:** *archiver*
- – (void)**drop**

Buttonă

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 319)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

- `-` **`createEnd`**

Phase: Using

- `-` **`setText:`** `(const char *)`*`text`*
  `Set the text for button.`
- `-` **`setButtonTarget:`** *`target`* **`method:`** `(SEL)`*`method`*
  `Set the target and selector for button.`

ButtonPanelă

Name

`ButtonPanel` — Several buttons bound together in one frame.

Description

Several buttons bound together in one frame.

Protocols adopted by ButtonPanel

Frame *(see page 299)*
CREATABLE *(see page 35)*

Methods

Phase: Using

- `-` **`setButtonTarget:`** *`target`*
  `Set a default target for use with addButtonName:method:.`
- `-` **`addButtonName:`** `(const char *)`*`name`* **`target:`** *`target`* **`method:`** `(SEL)`*`sel`*
  `Create a new button, and set both a target and method.`
- `-` **`addButtonName:`** `(const char *)`*`name`* **`method:`** `(SEL)`*`sel`*
  `Create a new button, and set the method, using the default target.`

Canvasă

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 287)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

- `-` **`createEnd`**
  `Create the canvas.`

Phase: Using

- `-` **`addWidget:`** *`widget`* **`X:`** `(int)`*`x`* **`Y:`** `(int)`*`y`* **`centerFlag:`** `(BOOL)`*`centerFlag`*
  `Position a widget inside the canvas`

- – **removeWidget:** *widget*
  Remove a widget from the canvas.
- – **checkGeometry:** *frame*
  Make sure the that the geometry is ‘reasonable’.

CanvasAbstractItemă

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 37)*
Drop *(see page 45)*

Methods

Phase: Creating

- – **createItem**
  Method to be implemented by subclass.
- – **createBindings**
  Method to be implemented by subclass.
- – **createEnd**
  Calls createItem and createBindings.
- – **setCanvas:** *canvas*
  Designates the id of the Canvas in which this item resides.

Phase: Using

- – **setTargetId:** *target*
  Designates the object to which this item refers.

- – **setClickSel:** (SEL)*sel*
  Sets the message that will be sent upon a click on this item.
- – **setMoveSel:** (SEL)*sel*
  Sets the message that will effect the motion of the item on the canvas.
- – **setPostMoveSel:** (SEL)*sel*
  Sets the message that will dictate what happens after the item is moved.
- – **clicked**
  Called when a mouse click occurs.
- – **initiateMoveX:** (long)*delta_x* **Y:** (long)*delta_y*
  Method to be implemented by subclass.
- – **getCanvas**
  Return the canvas.

CanvasItemă

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 290)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

- – **createBindings**
  Establishes the bindings for the buttons.

Phase: Using

- − **initiateMoveX:** (long)*delta_x* **Y:** (long)*delta_y*
  `Prepares for movement of the item within the canvas.`

CheckButtonă

Name

`CheckButton` — A check box on/off selection widget.

Description

A check box on/off selection widget.

Protocols adopted by CheckButton

InputWidget *(see page 304)*
CREATABLE *(see page 35)*

Methods

Phase: Using

- − (BOOL)**getBoolValue**
  `Get on/off status.`
- − **setBoolValue:** (BOOL)*v*
  `Turn the widget value and check button on or off.`

Circleă

Name

`Circle` — A CanvasItem that displays a circle.

Description

A CanvasItem that displays a circle.

Protocols adopted by Circle

CanvasItem *(see page 291)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

- – **setX:** (int)*x* **Y:** (int)*y*
  `Set the x, y coordinates for the center of the circle.`
- – **setRadius:** (unsigned)*r*
  `Set the radius of the circle.`
- – **createItem**
  `Create the Circle.`

ClassDisplayHideButtonă

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 287)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

- – **`setSubWidget:`** *`subWidget`*
- – **`setUser:`** *`user`*
- – **`setOwner:`** *`owner`*

ClassDisplayLabelă

Name

`ClassDisplayLabel` — A label for displaying class names.

Description

This widget is used internally by ClassDisplayWidget.

Protocols adopted by ClassDisplayLabel

Label *(see page 305)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

- – **`createEnd`**
  `Create a blue, left-justified label.`

Colormapă

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 37)*

CREATABLE *(see page 35)*

Methods

Phase: Creating

- `–` **`createEnd`**

Phase: Using

- `– (PixelValue *)`**`map`**

  `The current palette, per color-index.`

- `– (PixelValue)`**`black`**

  `The pixel value for black.`

- `– (PixelValue)`**`white`**

  `The pixel value for white.`

- `–`
  `(BOOL)`**`setColor:`** `(Color)`*`c`* **`ToRed:`** `(double)`*`r`* **`Green:`** `(double)`*`g`* **`Blue:`** `(double)`*`b`*

  `Add color index ‘c’ to the color map, using a certain per-`
  `cent of  red, green, and blue.`

- `– (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`* **`ToGrey:`** `(double)`*`g`*

  `Add a color of a certain level of grey.`

- `–` **`unsetColor:`** `(Color)`*`c`*

  `Remove color at index ‘c’ from the color map.`

CompleteProbeDisplayLabelǎ

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 305)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

- `-` **`setTargetWidget:`** *`targetWidget`*

  `Sets target widget which to report (e.g. probeDisplay).`
- `-` **`setProbedObject:`** *`probedObject`*

  `Sets the object that the probe display represents.`
- `-` **`createEnd`**

  `Performs drag and drop configruation, Button3 to get a CompleteProbeDisplay, and highlighting.`

CompositeItemǎ

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 290)*

Methods

Phase: Using

- `-` **`moveX:`** `(long)`*`delta_x`* **`Y:`** `(long)`*`delta_y`*
  `Must be implemented by subclass.`
- `-` **`initiateMoveX:`** `(long)`*`delta_x`* **`Y:`** `(long)`*`delta_y`*
  `Prepares for movement of the item within the canvas.`

Drawerǎ

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

- `-` **`drawX:`** `(int)`*`x`* **`Y:`** `(int)`*`y`*

Entryă

Name

`Entry` — Handles text-field input.

Description

Handles text-field input.

Protocols adopted by Entry

InputWidget *(see page 304)*
CREATABLE *(see page 35)*

Methods

Phase: Using

- `-` **`setHeight:`** `(unsigned)`*h*
  `This method aborts`

Formă

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 319)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

- – **`createEnd`**

Phase: Using

- – **`setEntryWidth:`** `(int)`*ew*
  `The width of all the Entry widgets.`
- – **`addLineName:`** `(const char *)`*n* **`Boolean:`** `(unsigned *)`*p*
  `Add a boolean CheckButton widget.`
- – **`addLineName:`** `(const char *)`*n* **`Int:`** `(int *)`*p*
  `Add an Entry to get an integer.`
- – **`addLineName:`** `(const char *)`*n* **`Double:`** `(double *)`*p*
  `Add an Entry to get a double.`

Frameă

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 287)*

CREATABLE *(see page 35)*

Methods

Phase: Creating

- – **setBorderWidth:** `(int)`*width*
  `Determines the width of the border, if any.`
- – **setReliefFlag:** `(BOOL)`*reliefFlag*
  `Determines whether or not a frame has a border.`
- – **createEnd**

Phase: Using

- – **withdraw**
  `Take the frame off screen.`
- – **deiconify**
  `Deiconify the frame.`

Graphă

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 287)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

- – **createEnd**

Phase: Using

- – **setTitle:** (const char *)*title*
  Set the title for the graph.
- – **setAxisLabelsX:** (const char *)*xl* **Y:** (const char *)*yl*
  Set the axis labels for the graph.
- – (id <GraphElement>)**createElement**
  Builds a new GraphElement to plot data with.
- – **setScaleModeX:** (BOOL)*xs* **Y:** (BOOL)*ys*
  Whether to autoscale every timestep or instead to jump scale.
- – **setRangesXMin:** (double)*minx* **Max:** (double)*maxx*
  Sets the X ranges for the graph. Turns off autoscaling.
- – **setRangesYMin:** (double)*miny* **Max:** (double)*maxy*
  Sets the Y ranges for the graph. Turns off autoscaling.
- – **setRangesXMin:** (double)*minx* **Max:** (double)*maxx* **YMin:** (double)*miny* **Max:** (double)*maxy*
  Sets the ranges for the graph. Turns off autoscaling.

GraphElementă

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 37)*

Drop *(see page 45)*

Methods

Phase: Creating

- − **setOwnerGraph:** *ownerGraph*
- − **createEnd**
- + **createOwnerGraph:** *ownerGraph*

Phase: Using

- − **setLabel:** (const char *)*label*
  Set the label for the element.
- − **setColor:** (const char *)*colorName*
  Set the color for the element.
- − **setDashes:** (int)*dashesVal*
  Set the dash style, 0 means solid.
- − **setSymbol:** (const char *)*symbol*
  Set the symbol for the element.
- − **addX:** (double)*x* **Y:** (double)*y*
  Add a new data point.
- − **resetData**
  Clear the data to be displayed.
- − **setWidth:** (unsigned)*w*
  Set the line width of the element.

Histogramă

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 287)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

- + **createBegin:** (id <Zone>)*aZone*
- − **setBinCount:** (unsigned)*n*
  Set the number of bins to use (bars to draw).
- − **createEnd**
  Initialize the histogram.

Phase: Using

- − **setColors:** (const char * const *)*c* **count:** (unsigned)*colorCount*
  Set colors for the histogram bars. If not set, all are blue. Colors are arrays of strings (one per bin/bar) of color names.
- − **setLabels:** (const char * const *)*l* **count:** (unsigned)*labelCount*
  Set labels for the histogram bars. If not set, they remain blank. Labels are arrays of strings, one per bin/bar.
- − **setTitle:** (const char *)*title*
  Set the title of the histogram.
- − **setBarWidth:** (double)*step*
  Set the width of the bars.
- − **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.
- − **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.
- − **setAxisLabelsX:** (const char *)*xl* **Y:** (const char *)*yl*

`Set the axis labels.`

- – **setActiveOutlierText:** `(unsigned)`*outliers* **count:** `(unsigned)`*count*

  `Set the text that describes a specified number of outliers.`

- – **hideLegend**

  `Hide the legend on the histogram.`

- – **drawHistogramWithInt:** `(int *)`*points*

  `Draw the (integer) data in the histogram.`

- – **drawHistogramWithDouble:** `(double *)`*points*

  `Draw the (double) data in the histogram.`

- – **drawHistogramWithInt:** `(int *)`*points* **atLocations:** `(double *)`*locations*

  `Draw the (integer) data in the histogram at particular offsets.`

- – **drawHistogramWithDouble:** `(double *)`*points* **atLocations:** `(double *)`*locations*

  `Draw the (double) data in the histogram at particular offsets.`

- – **setupZoomStack**

- – **setupActiveOutlierMarker**

- – **setupActiveItemInfo**

InputWidgetă

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 319)*

Methods

Phase: Using

- `- (const char *)`**`getValue`**
  `Get the string value of the widget.`
- `-` **`setValue:`** `(const char *)`*`v`*
  `Set the string value of the widget.  This must be implemented by a sub-class.`
- `-` **`linkVariableInt:`** `(int *)`*`p`*
  `Attach the widget value to an integer.`
- `-` **`linkVariableDouble:`** `(double *)`*`p`*
  `Attach the widget value to a double.`
- `-` **`linkVariableBoolean:`** `(unsigned *)`*`p`*
  `Attach the widget value to a boolean.`

Labelă

Name

`Label` — A widget with text.

Description

A widget with text.

Protocols adopted by Label

Widget *(see page 319)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

- `-` **`createEnd`**

Phase: Using

- `- setText: (const char *)text`
  `Set the text to write in the label.`

Lineă

Name

`Line` — A CanvasItem that displays a line.

Description

A CanvasItem that displays a line.

Protocols adopted by Line

CanvasItem *(see page 291)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

- `- setTX: (int)tx TY: (int)ty LX: (int)lx LY: (int)ly`
  `Set the end points of the line.`
- `- createItem`
  `Create the Line.`

LinkItemă

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 296)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

- `–` **`setFrom:`** *`from`*
  `Designate the node that will be the source of the link.`
- `–` **`setTo:`** *`to`*
  `Designate the node that will be the destination of the link.`
- `–` **`createItem`**
  `Create a the lines that make up the link item.`
- `–` **`createBindings`**
  `A LinkItem is passive; disable the mouse bindings.`

Phase: Using

- `–` **`setColor:`** `(const char *)`*`aColor`*
  `Set the color of the link.`
- `–` **`update`**
  `Redraw the link (especially due to the motion of nodes).`
- `– (void)`**`drop`**
  `Delete the lines that make up a LinkItem.`

MessageProbeEntryă

Name

`MessageProbeEntry` — A widget for arguments to a message probe.

Description

An Entry widget for MessageProbe arguments.

Protocols adopted by MessageProbeEntry

Entry *(see page 297)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

- – **setIdFlag:** (BOOL)*idFlag*
  `Indicates whether the type of this entry is an id.`
- – **setArg:** (int)*arg*
  `Indicates the argument number.`
- + **createBegin:** (id <Zone>)*aZone*
- – **createEnd**

NodeItemă

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 296)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

- – **createBindings**
  Set the mouse bindings for a NodeItem (e.g. dragging).

Phase: Using

- – **setX:** (int)*x* **Y:** (int)*y*
  Set the position of the node.
- – (int)**getX**
  Get the x position of the node on the canvas.
- – (int)**getY**
  Get the y position of the node on the canvas.
- – **setString:** (const char *)*string*
  Set the label to put on the node.
- – **setFont:** (const char *)*the_font*
  Set the font with which to draw the label.
- – **setColor:** (const char *)*aColor*
  Set the color of the node.
- – **setBorderColor:** (const char *)*aColor*
  Set the border color of the node.
- – **setBorderWidth:** (int)*aVal*
  Set the width of the border.
- – **createText**
  Create the text for the node.
- – **createPaddedText**
  Create the space for the text for the node.

OvalNodeItemă

Name

OvalNodeItem — A circular NodeItem.

Description

A NodeItem with a circular appearance.

Protocols adopted by OvalNodeItem

NodeItem *(see page 308)*

CREATABLE *(see page 35)*

Methods

Phase: Creating

- – **`createItem`**

  `Create the OvalNodeItem.`

Pixmapă

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 297)*

Create *(see page 37)*

CREATABLE *(see page 35)*

Methods

Phase: Creating

- \+ **createBegin:** (id <Zone>)*aZone*
- − **setFile:** (const char *)*filename*
  Create a pixmap from a PNG file.
- − **setDirectory:** (const char *)*path*
  Specify the directory to find the PNG file.
- − **setWidget:** (id <Widget>)*widget*
  Create a pixmap from a widget, or from the root window if widget is nil.
- − **setDecorationsFlag:** (BOOL)*decorationsFlag*
  Specify whether or not window manager decorations for a widget should be included.
- − **createEnd**

Phase: Using

- − **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.
- − (unsigned)**getWidth**
  Get the width of the bitmap in pixels.
- − (unsigned)**getHeight**
  Get the height of the bitmap in pixels.
- − **drawX:** (int)*x* **Y:** (int)*y*
  Draw the pixmap on the current raster at the given position.
- − **save:** (const char *)*filename*
  Save the pixmap to a file.
- − (void)**drop**

ProbeCanvasă

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 289)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

- – **setHorizontalScrollbarFlag:** (BOOL)*horizontalScrolbarFlag*
  `Indicates the presence or absence of a horizontal scroll bar.`
- – **createEnd**

Rasteră

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 287)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

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

Phase: Using

- – **setColormap:** (id <Colormap>)*c*

  Set the palette for this raster.
- – **drawPointX:** (int)*x* **Y:** (int)*y* **Color:** (Color)*c*

  Draw a point at the given coordinates with the given color.
- – **drawSelf**

  Draw the raster to the display.
- – **setWidth:** (unsigned)*newWidth* **Height:** (unsigned)*newHeight*

  Set the size of the raster.
- – **setButton:** (int)*n* **Client:** *c* **Message:** (SEL)*sel*

  Configure at mouse button to send a message to a given client object.
- – **fillRectangleX0:** (int)*x0* **Y0:** (int)*y0* **X1:** (int)*x1* **Y1:** (int)*y1* **Color:** (Color)*color*

  Fill a rectangle of given geometry and color.
- – **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.
- – **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.
- – **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.
- – **draw:** (id <Drawer>)*drawer* **X:** (int)*x* **Y:** (int)*y*

  Draw an object at a given position.
- – **erase**

  Erase the raster.

Rectangleă

Name

`Rectangle` — A CanvasItem that displays a rectangle.

Description

A CanvasItem that displays a rectangle.

Protocols adopted by Rectangle

CanvasItem *(see page 291)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

- `-` **`setTX:`** `(int)`*`tx`* **`TY:`** `(int)`*`ty`* **`LX:`** `(int)`*`lx`* **`LY:`** `(int)`*`ly`*
  `Set the diagonal corner coordinates of the rectangle.`
- `-` **`createItem`**
  `Create the Rectangle.`

RectangleNodeItemă

Name

`RectangleNodeItem` — A rectangular NodeItem.

Description

A rectangular NodeItem.

Protocols adopted by RectangleNodeItem

NodeItem *(see page 308)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

- – **createItem**
  A NodeItem with a rectangular appearance.

ScheduleItemă

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 296)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

- + **createBegin:** (id <Zone>)*aZone*
- – **setSchedule:** *schedule*
  Set the schedule to be inspected.
- – **setStep:** (unsigned)*step*
  Set the horizontal spacing of a time step.
- – **setX:** (int)*x* **Y:** (int)*y*
  Position the item on the canvas.

- – **createItem**

Phase: Using

- – **update**
  Redraw widget with current values from Schedule.
- – **at:** (timeval_t)*tval* **owner:** *owner* **widget:** *wid-get* **x:** (int)*sourceX* **y:** (int)*sourceY*
  Record the screen coordinates associated with a scheduling event.
- – **trigger:** *widget* **X:** (int)*x* **Y:** (int)*y*
  Send visual message indicator from browser to some target.
- – (void)**drop**
  Remove the schedule inspector from the canvas.

SimpleProbeDisplayHideButtonă

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 287)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

- – **setProbeDisplay:** *probeDisplay*
  The probe display in use.
- – **createEnd**

SuperButtonă

Name

`SuperButton` — Request superclass in ClassDisplayWidget.

Description

A button used by ClassDisplayWidget to ask for superclass.

Protocols adopted by SuperButton

Button *(see page 287)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

- `-` **`createEnd`**
- `-` **`setSuperWidget:`** *`superWidget`*
- `-` **`setOwner:`** *`owner`*
- `-` **`setUser:`** *`user`*

TextItemă

Name

`TextItem` — A CanvasItem that displays text.

Description

A CanvasItem that displays text.

Protocols adopted by TextItem

CanvasItem *(see page 291)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

- + **createBegin:** (id <Zone>)*aZone*
- – **setX:** (int)*x* **Y:** (int)*y*
  Set the coordinate for the center of the text.
- – **setText:** (const char *)*the_text*
  Set the text to display.
- – **setFont:** (const char *)*the_font*
  Set the font with which to display the text.
- – **setCenterFlag:** (BOOL)*centerFlag*
  Determine whether text is centered or not.
- – **createItem**
  Create the TextItem.

VarProbeEntryă

Name

`VarProbeEntry` — A widget for variable probes.

Description

An Entry widget for VarProbes.

Protocols adopted by VarProbeEntry

Entry *(see page 297)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

- – **setInteractiveFlag:** (BOOL)*interactiveFlag*
  Indicates whether the entry is editable or not.
- – **setOwner:** *owner*
  Indicate the object that is using this widget.
- – **setVarProbe:** *varProbe*
  Set the variable probe associated with this widget.
- – **createEnd**

Phase: Using

- – **getVarProbe**

VarProbeLabelǎ

Name

`VarProbeLabel` — A label for displaying variable names.

Description

This widget is used internally by VarProbeWidget.

Protocols adopted by VarProbeLabel

Label *(see page 305)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

- – **createEnd**
  Create a right-justified label.

Widgetă

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 37)*
Drop *(see page 45)*

Methods

Phase: Creating

- + **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
- − **setParent:** *parent*
  Set the containing window of the widget.
- − (const char *)**makeWidgetNameFor:** *widget*
  Compute the widget name for a component widget.
- − **setWidgetNameFromParent:** *parent*
  Set the widget name using the parent as context.
- − **setWidgetNameFromParentName:** (const char *)*parentWidgetName*
  Set the widget name using a hypothetical parent name.
- − **createEnd**

Phase: Using

- – **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.
- – **packFill**
- – **packBeforeAndFillLeft:** *widget* **expand:** (BOOL)*expandFlag*
- – **packFillLeft:** (BOOL)*expandFlag*
- – **packToRight:** *widget*
- – **packForgetAndExpand**
- – **setActiveFlag:** (BOOL)*activeFlag*
  Enable or disable the widget.
- – **setWidth:** (unsigned)*width*
  Set the width of the widget.
- – **setHeight:** (unsigned)*height*
  Set the height of the widget.
- – **setWidth:** (unsigned)*width* **Height:** (unsigned)*height*
  Set the width and height of the widget.
- – **setX:** (int)*x* **Y:** (int)*y*
  Set the position of the widget.
- – **setWindowTitle:** (const char *)*title*
  Set the title on the widget.
- – **getParent**
  Get the containing window of the widget.
- – **getTopLevel**
  Get top level frame
- – (const char *)**getWidgetName**
  Get the widget name.
- – (unsigned)**getHeight**
  Get the height of the widget.
- – (unsigned)**getWidth**
  Get the widget the widget.
- – (int)**getX**
  Get the X position of the widget.
- – (int)**getY**
  Get the Y position of the widget.

- – (const char *)**getWindowGeometry**
- – **setWindowGeometry:** (const char *)*s*
- – **enableDestroyNotification:** *notificationTarget* **notification-Method:** (SEL)*destroyNotificationMethod*
  Call a method if we are destroyed.
- – **disableDestroyNotification**
  Prevent calling the destroy notification method.
- – (BOOL)**getDestroyedFlag**
- – (void)**drop**

WindowGeometryRecordă

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 57)*
Create *(see page 37)*
Drop *(see page 45)*

Methods

Phase: Using

- – **setX:** (int)*x* **Y:** (int)*y*
  Set the window position.
- – **setWidth:** (unsigned)*w* **Height:** (unsigned)*h*
  Set the window size.
- – (BOOL)**getSizeFlag**
  Get the flag that indicates if the size has been set.

- – (BOOL)**getPositionFlag**
  Get the flag that indicates if the position has been set.
- – (unsigned)**getWidth**
  Get the window's horizontal size.
- – (unsigned)**getHeight**
  Get the window's vertical size.
- – (int)**getX**
  Get the window's horizontal position.
- – (int)**getY**
  Get the window's vertical position.

ZoomRasterǎ

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 312)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

- – **createEnd**

Phase: Using

- – **increaseZoom**
  Make the raster bigger.

- – **decreaseZoom**
  Make the raster smaller.
- – (unsigned)**getZoomFactor**
  Get the current zoom factor.
- – **setZoomFactor:** (unsigned)*z*
  Set the zoom factor.
- – **handleConfigureWidth:** (unsigned)*newWidth* **Height:** (unsigned)*newHeight*
  Reconfigures the ZoomRaster when the window is resized.

Generală

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

Revision History

1999-08-21 analysis.h *mgd*

Add Zone typing to +create:* methods. (EZSequence): Change from CREATABLE to RETURNABLE.

1999-08-08 analysis.h *alex*

(EZGraph): Add two convenience methods for batch-mode (non-graphical) instances of EZGraph (+create:setFileOutput:, +create:setFileName:).

1999-08-01 analysis.h *alex*

(EZGraph): Add new convenience factory method to protocol.

1999-07-20 analysis.h *vjojic*

Add EZSequence protocol.

1999-04-25 analysis.h *alex*

(EZBin, EZGraph): Add compliance to SwarmObject protocol.

1999-02-25 analysis.h *mgd*

Add CREATABLE tags to all non-abstract protocols.

1999-01-15 analysis.h *alex*

([EZGraph -setFileName:]): Update method documentation.

1998-12-09 analysis.h *mgd*

(EZBin, EZGraph): Declare getFilename and setFilename:. Add getTitle.

1998-09-29 analysis.h *mgd*

(EZBin): Add setMonoColorBars:. (EZGraph): Change getGraph return type, and include createEnd.

1998-09-17 analysis.h *mgd*

(EZBin): Note that the upper bound is not inclusive.

1998-09-02 analysis.h *mgd*

Likewise.

1998-08-23 analysis.h *mgd*

(EZGraph): Return value is now a sequence for create{,Average,Count,Min,Max,Total}Sequence:withFeedFrom: methods. Declare new method -dropSequence:.

1998-07-09 analysis.h *alex*

(FunctionGraph): Add phase tags CREATING and USING.

1998-07-07 analysis.h *mgd*

(ActiveGraph, ActiveOutFile, FunctionGraph): Add protocols and class objects.

1998-06-16 analysis00.sgml *alex*

Added missing description of dependencies - objectbase.h and simtoolsgui.h.

1998-06-16 Makefile.am *mgd*

Include from refbook/ instead of src/.

1998-06-14 Makefile.am *mgd*

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

1998-06-11 analysis00.sgml, analysiscont.sgml *mgd*
Update IDs to SWARM.module.SGML.type.
1998-06-05 analysis.ent *mgd*
Use public identifiers.
1998-06-04 Makefile.am *mgd*
(swarm_ChangeLog): Add.
1998-06-02 analysis.h *mgd*
Updated documentation tags.
1998-05-22 Makefile.am *mgd*
New file.
1998-05-22 analysis.ent.in *mgd*
New file:
1998-05-22 analysis.ent *mgd*
Removed.
1998-05-21 *mgd*
Begin revision log.
1998-05-05 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.
1998-04-22 analysis.h *mgd*
Remove includes of analysis things. Replace with protocols for Averager, EZBin, EZDistribution, EZGraph, and Entropy. Add a @class for each.

ActiveGraphă

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 175)*
CREATABLE *(see page 35)*

Methods

Phase: Using

- – **setElement:** *ge*
  `Sets the graph element used to draw on.`
- – **setDataFeed:** *d*
  `Sets the object that will be probed for data.`
- – **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.`

ActiveOutFileă

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 175)*
CREATABLE *(see page 35)*

Methods

Phase: Using

- – **`setFileObject:`** *`aFileObj`*
  `The setFileObject: method sets the file object to which the data will be sent.`
- – **`setDataFeed:`** *`d`*
  `The setDataFeed: method sets the object that will be probed for data.`
- – **`step`**
  `The step method fires the probe, reads the value from the object, and  sends the value to the file.`

Averagerǎ

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 175)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

- – **setCollection:** *aCollection*
  Sets the collection of objects that will be probed.
- – **createEnd**

Phase: Using

- – **update**
  The update method runs through the collection calling the selector on each object.
- – (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.
- – (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)**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.
- – (unsigned)**getCount**
  The getCount method returns the number of values the averager collects.

EZBină

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 184)*
GUIComposite *(see page 269)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

- + **createBegin:** (id <Zone>)*aZone*
- − **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.
- − **setCollection:** *aCollection*
  The setCollection method sets the collection of target objects which will be requested to generate the dataset for the histogram.
- − **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.)
- − **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.
- − **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.)
- − **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.
- − **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.)

- – **setMonoColorBars:** (BOOL)*mcb*
  The setMonoColorBars method specifies whether all bars should be shown in a single color (blue). The default is differently colored bars.
- – **setBinCount:** (unsigned)*theBinCount*
  The setBinCount method sets the number of bins the histogram will have.
- – **setLowerBound:** (double)*theMin*
  The setLowerBound method sets the inclusive lower bound on the histogram range.
- – **setUpperBound:** (double)*theMax*
  The setUpperBound method sets the non-inclusive upper bound on the histogram range.
- – **setColors:** (const char * const *)*colors* **count:** (unsigned)*nc*
  Set a custom vector of colors for the histogram bars
- – **createEnd**

Phase: Using

- – **setPrecision:** (unsigned)*precision*
  Sets the number of significant figures shown for major-tick labels.
- – **reset**
  The reset method resets the histogram.
- – **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.
- – **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.
- – **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.

- \- **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.

- \- (unsigned *)**getDistribution**
  The getDistribution method returns an array of integers containing the number of entries which landed in each bin of the histogram.

- \- (unsigned)**getCount**
  The getCount method gets the number of entries which landed within the bounds of 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)**getBinCount**
  The getBinCount method gets the number of bins in the histogram.

- \- (unsigned)**getBinColorCount**
  The getBinColorCount method gets the number of distinct bin colors allocated (by default, or by the user).

- \- (double)**getLowerBound**
  The getLowerBound method gets the lower bound on the histogram range.

- \- (double)**getUpperBound**
  The getUpperBound method gets the upper bound on the histogram range.

- \- (double)**getMin**
  The getMin method gets the minimum value in the dataset.

- \- (double)**getMax**
  The getMax method gets the maximum value in the dataset.

- \- (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)**getStd**
  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.

- \- (id <Histogram>)**getHistogram**
  Return the histogram widget.

- `– (const char *)`**`getTitle`**
  `Return the title string.`
- `– (const char *)`**`getFileName`**
  `Return the filename string.`

EZDistributionă

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 332)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

- `–` **`createEnd`**

Phase: Using

- `–` **`update`**
  `The update method polls the bins and updates the entropy of the distribution as well as the probabilities associated with the individual bins.`
- `–` **`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 dis-`

play is enabled (the state of setGraphics is set to 1), the histogram will be drawn.

- – (double *)**getProbabilities**

  The getProbabilities method returns an array of doubles representing the probability of every bin in the distribution.

- – (double)**getEntropy**

  The getEntropy method returns the entropy of the distribution as calculated in the previous call to update.

EZGraphă

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 184)*
GUIComposite *(see page 269)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

- + **create:** (id <Zone>)*aZone* **setTitle:** (const char *)*aTitle* **setAxisLabelsX:** (const char *)*xl* **Y:** (const char *)*yl*

  Convenience method for creating ‘graphical’ EZGraph instances

- + **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* **setFileName:** (const char *)*aFileName*
  Convenience method for creating a non-graphical EZGraph, in this case, the filename is explicitly set by the user
- – **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.
- – **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.
- – **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.
- – **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.)
- – **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.)
- – **setColors:** (const char * const *)*colors* **count:** (unsigned)*nc*
  Set a custom vector of colors for the graph lines
- – **createEnd**

Phase: Using

- – **setRangesXMin:** (double)*xmin* **Max:** (double)*xmax*
  Fix the range of X values on the graph between some range.
- – **setRangesYMin:** (double)*ymin* **Max:** (double)*ymax*
  Fix the range of Y values on the graph between some range.

- \- **setScaleModeX:** (BOOL)*xs* **Y:** (BOOL)*ys*

  Whether to autoscale every timestep or instead to jump scale.

- \- (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.)

- \- **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.

- \- **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 (as opposed to a single object as in createSequence) The method returns an id which can be used later with -dropSequence.

- \- **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 (as opposed to a single object as in createSequence) The method returns an id which can be used later with -dropSequence.

- \- **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 (as opposed to a single object as in createSequence) The method returns an id which can be used later with -dropSequence.

- \- **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 (as opposed to a single object as in createSequence) The method returns an id which can be used later with -

dropSequence.

- \- **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 (as opposed to a single object as in createSequence) The method returns an id which can be used later with -dropSequence.
- \- **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.
- \- (const char *)**getTitle**
  Return the title string.
- \- (const char *)**getFileName**
  Return the file name prefix string.
- \- **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.
- \- **outputGraph**
  the outputGraph method updates the graph with the data obtained from the last call to -update. If setGraphics==0, nothing is done.
- \- **outputToFile**
  the outputToFile method sends to the disk file data obtained from the last call to -update. If setFileOutput==0, nothing is done.
- \- **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.

EZSequenceă

Name

EZSequence —

Description

Protocols adopted by EZSequence

SwarmObject *(see page 184)*
RETURNABLE *(see page 57)*

Methods

None

Entropyă

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 175)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

- `–` **`setCollection:`** *`aCollection`*

  ```
   The setCollection method sets the collection of ob-
  jects that will be  probed.
  ```

- `–` **`createEnd`**

Phase: Using

- – **update**
  The update method polls the collection and updates the entropy. This method should be scheduled prior to collecting the data using getEntropy.
- – (double)**getEntropy**
  The getEntropy method returns the calculated Entropy. The entropy value is read out of the object, not computed everytime it is requested.

FunctionGraphă

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 184)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

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

- – **createEnd**
- – **setElement:** (id <GraphElement>)*graphElement*
  Set the GraphElement to use for plotting.
- – **setDataFeed:** *feed*
  Set the target to send the function method.
- – **setFunctionSelector:** (SEL)*aSel*
  Set the function method.
- – **setArithmeticWarn:** (BOOL)*state*
  If true, raise a warning if the function method failed to compute a value.
- – **setXMin:** (double)*minx* **Max:** (double)*maxx* **Resolution:** (unsigned)*steps*
  Set the range and resolution of X values at which to compute values.
- – **setXMin:** (double)*minx* **Max:** (double)*maxx* **StepSize:** (double)*size*
  Set the range and step size of X values at which to compute values.
- – **setResetFrequency:** (unsigned)*freq*
  Set the frequency at which to clear the graph element.

Phase: Using

- – **graph**
  Draw the graph with the current contents of the graph element.

Generală

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

1999-08-23 space.h *mgd*

Add Discrete2d and Raster typing on convenience factory methods (for use by Java).

1999-08-21 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.

1999-08-02 space.h *alex*

(Diffuse2d): Make initializeLattice a CREATING method to match implementation.

1999-07-31 space.h *alex*

(Object2dDisplay, Value2Display, Discrete2d, Diffuse2d, DblBuffer2d, Grid2d): Add +create: convenience methods to these CREATABLE protocols.

1999-07-08 space.h *mgd*

(Object2dDisplay): Add -makeProbeAtX:Y:.

1999-05-04 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.

1999-04-30 space.h *mgd*

(Discrete2d): Remove setUseObjects and setUseValues.

1999-04-24 space.h *alex*

([Discrete2d -setUseObjects], [Discrete2d -setUseValues]): Add to SETTING phase.

1999-02-25 space.h *mgd*

Merge _Discrete2d with Discrete2d. Add CREATABLE tags for all non-abstract protocols.
1998-10-03 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.
1998-07-14 space.h *mgd*
(_Discrete2d): Split Discrete2d into new and presentation interface.
1998-06-16 space00.sgml *alex*
Added missing description of dependencies - objectbase.h and gui.h.
1998-06-16 Makefile.am *mgd*
Include from refbook/ instead of src/.
1998-06-14 Makefile.am *mgd*
(MODULE): New variable. Include Makefile.rules from src. Remove everything else.
1998-06-11 space00.sgml, spacecont.sgml *mgd*
Update IDs to SWARM.module.SGML.type.
1998-06-05 space.ent *mgd*
Use public identifiers.
1998-06-04 Makefile.am *mgd*
(swarm_ChangeLog): Add.
1998-06-02 space.h *mgd*
Add a module summary documentation tag.
1998-05-22 Makefile.am *mgd*
New file.
1998-05-22 space.ent.in *mgd*
New file.
1998-05-22 space.ent *mgd*
Removed.
1998-05-21 *mgd*
Begin revision log.
1998-05-05 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.
1998-04-23 space.h *mgd*
Now a protocol definition file instead of a full set of includes.

Ca2dă

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 349)*

Methods

Phase: Creating

- `- `**`setNumStates:`**` (unsigned)`*n*
  `Record the number of states the CA understands.`
- `- `**`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.`
- `- `**`createEnd`**
  `Check that numStates has been set.`

Phase: Using

- `- `**`stepRule`**
  `One iteration of the CA rule. Unimplemented in Ca2d; subclass this to implement your CA rule.`

ConwayLife2dă

Name

`ConwayLife2d` — Classic 2d Conway's Life CA.

Description

Classic 2d Conway's Life CA.

Protocols adopted by ConwayLife2d

Ca2d (see page 348)
CREATABLE (see page 35)

Methods

Phase: Creating

- `+ **createBegin:** (id <Zone>)*aZone*`
  `Set number of states to 2.`
- `- **initializeLattice**`
  `Initialize lattice to random 1/3 in state 1.`

Phase: Using

- `- **stepRule**`
  `Run Conway's Life rule (simpleminded version).`

DblBuffer2dǎ

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 351)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

- + **create:** (id <Zone>)*aZone* **setSizeX:** (unsigned)*x* **Y:** (unsigned)*y*
  Convenience constructor for DblBuffer lattice
- − **createEnd**
  Rewrites the method from Discrete2d. Allocate two lattices, makes the offsets.

Phase: Using

- − (id *)**getNewLattice**
  Return a pointer to the newLattice buffer.
- − **updateLattice**
  Copy newLattice to lattice, in effect updating the lattice.
- − **putObject:** *anObject* **atX:** (unsigned)*x* **Y:** (unsigned)*y*
  Overridden so writes happen to newLattice.
- − **putValue:** (long)*v* **atX:** (unsigned)*x* **Y:** (unsigned)*y*
  Overridden so writes happen to newLattice.

Diffuse2dă

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 348)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

- + **create:** (id <Zone>)*aZone* **setSizeX:** (unsigned)*x* **Y:** (unsigned)*y* **setDiffusionConstant:** (double)*d* **setEvaporationRate:** (double)*e*
  `Convenience constructor for Diffuse2d`
- + **createBegin:** (id <Zone>)*aZone*
  `Set diffusion constant and evaporation rate to 1.0, numStates to 0x7fff.`
- − **initializeLattice**
  `Initialize world to 0.`

Phase: Setting

- − **setDiffusionConstant:** (double)*d*
  `Set the diffusion constant. Values over 1.0 might not be valid.`
- − **setEvaporationRate:** (double)*e*
  `Set the evaporation rate. Values over 1.0 don't make much sense`

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.`

Discrete2dă

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 184)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

- + **create:** (id <Zone>)*aZone* **setSizeX:** (unsigned)*x* **Y:** (unsigned)*y*
  Convenience constructor for Discrete2d lattice
- – **setSizeX:** (unsigned)*x* **Y:** (unsigned)*y*
  Set the world size.
- – (id *)**allocLattice**
  Allocate memory for the lattice.
- – **makeOffsets**
  Given an array size, compute the offsets array that caches the multiplication by ysize. See the discrete2dSiteAt macro.
- – **createEnd**
  Create the lattice, precompute the offsets based on Y coordinate.

Phase: Setting

- – **setLattice:** (id *)*lattice*

Phase: Using

- – (unsigned)**getSizeX**
  Get the size of the lattice in the X dimension.
- – (unsigned)**getSizeY**
  Get the size of the lattice in the Y dimension.
- – **getObjectAtX:** (unsigned)*x* **Y:** (unsigned)*y*
  Return the pointer stored at (x,y).
- – (long)**getValueAtX:** (unsigned)*x* **Y:** (unsigned)*y*
  Return the integer stored at (x,y).

- – **putObject:** *anObject* **atX:** (unsigned)*x* **Y:** (unsigned)*y*
  Put the given pointer to (x,y) overwriting whatever was there.
- – **putValue:** (long)*v* **atX:** (unsigned)*x* **Y:** (unsigned)*y*
  Put the given integer to (x,y) overwriting whatever was there.
- – **fastFillWithValue:** (long)*aValue*
  Directly fills the lattice with a value.
- – **fastFillWithObject:** *anObj*
  Directly fills the lattice with an object.
- – **fillWithValue:** (long)*aValue*
  Fills the space using putValue.
- – **fillWithObject:** *anObj*
  Fills the space using putObject.
- – (id *)**getLattice**
  Returns the lattice pointer - use this for fast access.
- – (int)**setDiscrete2d:** (id <Discrete2d>)*a* **toFile:** (const char *)*filename*
  This method reads a PGM formatted file and pipes the data into a Discrete2d object.
- – **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.
- – (long *)**getOffsets**

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"))))))
```

Grid2dă

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 351)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

- + **create:** (id <Zone>)*aZone* **setSizeX:** (unsigned)*x* **Y:** (unsigned)*y*
  Convenience constructor for Grid2d

Phase: Using

- − **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.
- − **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.

Int2dFilerǎ

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 184)*
CREATABLE *(see page 35)*

Methods

Phase: Using

- – **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).
- – **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.
- – **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 written to the file. The default background value is 0.
- – **fileTo:** (const char *)*aFileName*
  When the Int2dFiler receives this message, it opens a file called fileName, stores the state of a prespecified space into it, and then closes the file.

Object2dDisplayă

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 184)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

- + **create:** `(id <Zone>)`*aZone* **setDisplayWidget:** `(id <Raster>)`*r* **setDiscrete2dToDisplay:** `(id <Discrete2d>)`*c* **setDisplayMessage:** `(SEL)`*s*
  Convenience constructor for Object2dDisplay
- – **setDisplayWidget:** `(id <Raster>)`*r*
  Set the display widget to use for drawing.
- – **setDiscrete2dToDisplay:** `(id <Discrete2d>)`*c*
  Set the 2d array to draw.
- – **setDisplayMessage:** `(SEL)`*s*
  Set the message to be sent to each object in the grid to make it draw itself.

Phase: Using

- – **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.
- – **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.
- – **makeProbeAtX:** `(unsigned)`*x* **Y:** `(unsigned)`*y*

Make a probe for an object at a specific point.

Value2dDisplayă

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 184)*
CREATABLE *(see page 35)*

Methods

Phase: Creating

- + **create:** (id <Zone>)*aZone* **setDisplayWidget:** (id <Raster>)*r* **colormap:** (id <Colormap>)*c* **setDiscrete2dToDisplay:** (id <Discrete2d>)*d*
  Convenience constructor for Value2dDisplay
- – **setDisplayWidget:** (id <Raster>)*r* **colormap:** (id <Colormap>)*c*
  Set the display widget and the colourmap to use to draw the value array.
- – **setDiscrete2dToDisplay:** (id <Discrete2d>)*c*
  Set which array to draw.
- – **createEnd**

Phase: Using

- – **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.
- – **display**

```
 Draw the array on the given wid-
get. Note that you still have to tell the widget to draw itself after-
wards. The code for display uses the fast macro access in Dis-
crete2d on the cached return value from getLat-
tice. It also caches the drawPointX:Y: method lookup on the display wid-
get - this is a nice trick that you might want to look at.
```

Generală

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.

Appendix A. GridTurtle Test Programs

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 459)* of the Defobj Library *(see page 21)* 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 373)*.** 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 *(see page 373)*, grid1b.m *(see page 376)*, grid1c.m *(see page 378)* .** Tests of Array type. The setInitialValue:, setDefaultMember:, and setMemberBlock: options are each tested by variants of the same basic program.
- **grid2.m *(see page 380)*, grid2b.m *(see page 382)*.** 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** ***(see page 385)*****, grid3b.m** ***(see page 386)*****.** Tests of Map type. grid3b uses an alternate compare function to handle integers rather than id values as keys.
- **grid4.m** ***(see page 387)*****, grid4b.m** ***(see page 389)*****.** 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 390)*****.** Tests of ActionGroup execution independent of any schedule, both by itself and within a swarm.
- **grid6.m** ***(see page 393)*****.** Test of running an entire top-level activity within an action of an already running activity.
- **grid7.m** ***(see page 395)*****.** Tests of schedule execution and schedule merging, using various combinations of absolute, relative, and repeating schedules.
- **grid8.m** ***(see page 398)*****.** Basic test of schedules running within a simulation swarm that is nested within an observer swarm.
- **grid9.m** ***(see page 401)*****.** 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 Library *(see page 156)* library.)
- **mousetraps.m** ***(see page 403)*****, Mousetrap.m** ***(see page 365)*****.** 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.m** ***(see page 405)*****, Mousetrap2.m** ***(see page 368)*****.** 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 407)*****.** Simple test of basic operations on the String type.

A.2.2. Support files

- **Makefile** ***(see page 363)*****.** 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 372)*****.** 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 372)*****.** 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 362)*** **.** 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 459)* 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.

A.3. GridTurtle Code examples

Example A-1 GridTurtle.h

```
#import "grid.h"
#import <defobj/Create.h>

@interface GridTurtle_c : CreateDrop_s
{
@public
  int          x_coord;
  int          y_coord;
  direction_t  direction;
  member_t     listMembership;
}
@end
```

Example A-2 GridTurtle.m

```
#import "GridTurtle.h"

#define __USE_FIXED_PROTOTYPES__  // for gcc headers
#include <stdio.h>


@implementation GridTurtle_c

- (void) move: (int) distance
{
  switch (direction) {
  case north: y_coord += distance; break;
  case east:  x_coord += distance; break;
  case south: y_coord -= distance; break;
  case west:  x_coord -= distance; break;
  }
}

- (void) turn: (int) angle
```

```
{
  direction = ( direction - angle ) % 4;
}

- (void) print
{
  printf( "x coord: %d  y coord: %d  direction: %d\n",
           x_coord, y_coord, direction );
}

@end
```

Example A-3 Makefile

```
ifeq ($(SWARMHOME),)
SWARMHOME=../../../swarm
endif
include $(SWARMHOME)/etc/swarm/Makefile.common
include $(SWARMHOME)/etc/swarm/Makefile.rules

CFLAGS+=-Werror
OTHERINCDIRS=-I.
SWARMINCLUDES+=$(OTHERINCDIRS)

LDFLAGS=$(SWARMLDFLAGS) $(SUPPORTLDFLAGS)
LINK=$(SWARMHOME)/bin/libtool-swarm -mode link $(CC) $(LDFLAGS)

LIBS_C= -lswarm
LIBS_A=  $(LIBS_C)

TARGETS_C= strtest grid0 grid1a grid1b grid1c grid2 grid2b grid3 grid3b \
           grid4 grid4b
TARGETS_A= grid5 grid6 grid7 grid8 grid9
TARGETS_M= mousetraps mousetraps2

OBJECTS= grid.o GridTurtle.o

all: $(TARGETS_C) $(TARGETS_A) $(TARGETS_M)

grid.o: grid/classes.h grid.xm

grid.xm: grid.h
OBJC="$(OBJC)" OBJCFLAGS="$(OTHERINCDIRS) -I$(includedir) $(CPPFLAGS) \
$(OBJCFLAGS)" OBJECTS="$(OBJECTS)" $(bindir)/make-h2x $@ .

grid/classes.h:
if [ ! -x grid ]; then mkdir grid; fi
```

```
cd grid; ln -s ../grid.xc classes.h

install:
echo "no default install"
clean:
- rm -f grid.x[cmt] *.o $(TARGETS_C) $(TARGETS_A) $(TARGETS_M) core
- rm grid/classes.h; rmdir grid

c: $(TARGETS_C)
a: $(TARGETS_A)
m: $(TARGETS_M)

$(TARGETS_C): %: grid.o %.o $(OBJECTS)
$(LINK) $@.o $(OBJECTS) -o $@ $(LIBS_C)

$(TARGETS_A): %: grid.o %.o $(OBJECTS)
$(LINK) $@.o $(OBJECTS) -o $@ $(LIBS_A)

mousetraps: grid.o mousetraps.o Mousetrap.o
$(LINK) $@.o Mousetrap.o -o $@ $(LIBS_A)

mousetraps2: grid.o mousetraps2.o Mousetrap2.o
$(LINK) $@.o Mousetrap2.o -o $@ $(LIBS_A)
```

Example A-4 Mousetrap.h

```
#import <activity.h>
#import <defobj/Create.h>

@interface Mousetrap : CreateDrop
{
  int  xCoord;
  int  yCoord;
  int  triggered;
  id   displayWidget;
}
+ create: aZone setXCoord: (int)x setYCoord: (int)y;
- (int) getXCoord;
- (int) getYCoord;
- (void) trigger;
- (BOOL) getTriggered;
- (void) printTriggered;
/*
-               setDisplayWidget: w;
*/
@end
```

```
@interface Grid2D : CreateDrop
{
  int  gridSize;
  id   *cells;      // array of gridSize X gridSize cells
}+ create: aZone setGridSize: (int)size;
- (int) getGridSize;
- atX: (int)xCoord atY: (int)yCoord;
- (void) atX: (int)xCoord atY: (int)yCoord put: anObject;
@end

// global variables currently used as world parameters
//
extern int triggerLikelihood;
extern int numberOutputTriggers;
extern int maxTriggerDistance;
extern int maxTriggerTime;
extern int gridSize;
extern int printTriggerEvents;
extern int printSameTickTriggers;

// (trigger distance and trigger time are currently uncorrelated.)

// global variable containing root object for world (eventually in context?)
//
extern id grid;
```

Example A-5 Mousetrap.m

```
#import "Mousetrap.h"

#define __USE_FIXED_PROTOTYPES__   // for gcc headers
#include <stdio.h>
#include <stdlib.h>

extern id  grid, worldSchedule;

extern unsigned  numTriggered;

@implementation Mousetrap

+ create: aZone setXCoord: (int)x setYCoord: (int)y
{
  Mousetrap  *newTrap;

  newTrap = [self create: aZone];
  newTrap->xCoord = x;
  newTrap->yCoord = y;
```

```
  return newTrap;
}

- (void) trigger
{
  int       n, xTrigger, yTrigger;
  unsigned  bits, xbits, ybits, tbits, currentTick, triggerTick;

  if ( ! triggered ) {

    currentTick = getCurrentTime();
    if ( printTriggerEvents ) {
      printf( "mousetrap at x: %.2d, y: %.2d triggered at time: %d\n",
              xCoord, yCoord, currentTick );
    }
    // nelson hacks
    // [displayWidget drawPointX: xCoord Y: yCoord Color: 2];
    numTriggered++;

    bits = rand();
    if ( ( ( bits & 0xff ) ) < triggerLikelihood ) {
      triggered = 1;

      for ( n = numberOutputTriggers; n > 0; n– ) {
        bits = rand();
        xbits = bits & 0xff;
        ybits = bits » 8 & 0xff;
        bits = rand();
        tbits = bits & 0xff;
        bits »= 8;

if ( bits & 1 ) {
  xTrigger = ( xCoord + ( xbits & ( maxTriggerDistance - 1 ) ) )
                     % gridSize;
} else {
  xTrigger = ( xCoord - ( xbits & ( maxTriggerDistance - 1 ) )
                       + gridSize ) % gridSize;
}
if ( bits & 2 ) {
  yTrigger = ( yCoord + ( ybits & (maxTriggerDistance - 1) ) )
                     % gridSize;
} else {
  yTrigger = ( yCoord - ( ybits & (maxTriggerDistance - 1) )
                       + gridSize ) % gridSize;
}
```

```
        triggerTick = currentTick + ( tbits & (maxTriggerTime - 1) );

        if ( printSameTickTriggers && triggerTick == currentTick )
          printf( "triggering mousetrap at same timestep at time: %d\n",
                  currentTick );

[worldSchedule at: triggerTick
          createActionTo: [grid atX: xTrigger atY: yTrigger]
          message: M( trigger )];
      }
    }
  }
}

- (int) getXCoord
{
  return xCoord;
}

- (int) getYCoord
{
  return yCoord;
}

- (BOOL) getTriggered
{
  return triggered;
}

- (void) printTriggered
{
  timeval_t  currentTime;

  currentTime = getCurrentTime();
  printf( "trigger status of mouse-
trap at x: %.2d, y: %.2d at time: %lu is %d\n",
             xCoord, yCoord, currentTime, triggered );
}

/*
- setDisplayWidget: w {
  displayWidget = w;
}
*/
```

```
@end


@implementation Grid2D

+ create: aZone setGridSize: (int)size
{
  Grid2D  *newGrid;

  newGrid = [aZone allocIVars: self];
  newGrid->gridSize = size;
  newGrid->cells = [aZone allocBlock:
                     newGrid->gridSize * newGrid->gridSize * sizeof (id)];
  return newGrid;
}

- (int) getGridSize
{
  return gridSize;
}

- atX: (int)xCoord atY: (int)yCoord
{
  return cells[ xCoord + yCoord * gridSize];
}

- (void) atX: (int)xCoord atY: (int)yCoord put: anObject
{
  cells[ xCoord + yCoord * gridSize] = anObject;
}

@end
```

Example A-6 Mousetrap2.m

```
#import "Mousetrap.h"

#define __USE_FIXED_PROTOTYPES__   // for gcc headers
#include <stdio.h>
#include <stdlib.h>

extern id  grid, worldSchedule;

extern unsigned  numTriggered;

@implementation Mousetrap
```

```
+ create: aZone setXCoord: (int)x setYCoord: (int)y
{
  Mousetrap  *newTrap;

  newTrap = [self create: aZone];
  newTrap->xCoord = x;
  newTrap->yCoord = y;
  return newTrap;
}

- (void) trigger
{
  int       n, xTrigger, yTrigger;
  unsigned  bits, xbits, ybits, tbits, currentTick, ownerTick, triggerTick;
  id        subgroup, ownerActivity;

  if ( ! triggered ) {

    // get subclock schedule and its current time

    // (Could still have a concurrent group within the subclock schedule,
    // but the initial getCurrentScheduleActivity() will skip over this
    // to get the running activity of the subclock schedule.)

    ownerActivity = getCurrentScheduleActivity();
    currentTick   = [ownerActivity getCurrentTime];

    // get activity running the subclock schedule activity, and its time

    ownerActivity = [ownerActivity getOwnerActivity];
    ownerTick     = [ownerActivity getCurrentTime];

    if ( printTriggerEvents ) {
      printf( "mousetrap at x: %.2d, y: %.2d triggered at time: %d,%d\n",
              xCoord, yCoord, ownerTick, currentTick );
    }
    // nelson hacks
    // [displayWidget drawPointX: xCoord Y: yCoord Color: 2];
    numTriggered++;

    bits = rand();
    if ( ( ( bits & 0xff ) ) < triggerLikelihood ) {
      triggered = 1;

      for ( n = numberOutputTriggers; n > 0; n- ) {
```

```
        bits = rand();
        xbits = bits & 0xff;
        ybits = bits » 8 & 0xff;
        bits = rand();
        tbits = bits & 0xff;
        bits »= 8;

if ( bits & 1 ) {
  xTrigger = ( xCoord + ( xbits & ( maxTriggerDistance - 1 ) ) )
                     % gridSize;
} else {
  xTrigger = ( xCoord - ( xbits & ( maxTriggerDistance - 1 ) )
                       + gridSize ) % gridSize;
}
if ( bits & 2 ) {
  yTrigger = ( yCoord + ( ybits & (maxTriggerDistance - 1) ) )
                     % gridSize;
} else {
  yTrigger = ( yCoord - ( ybits & (maxTriggerDistance - 1) )
                       + gridSize ) % gridSize;
}
        triggerTick = ( ownerTick * 10 ) + currentTick +
                        ( tbits & (maxTriggerTime - 1) );

        if ( printSameTickTriggers && triggerTick == currentTick )
          printf( "triggering mousetrap at same timestep at time: %d\n",
                  currentTick );

subgroup = [worldSchedule insertGroup: (id)( triggerTick / 10 )];
        [subgroup
          at: (triggerTick % 10)
          createActionTo: [grid atX: xTrigger atY: yTrigger]
          message: M( trigger )];
      }
    }
  }
}

- (int) getXCoord
{
  return xCoord;
}

- (int) getYCoord
{
```

```
  return yCoord;
}

- (BOOL) getTriggered
{
  return triggered;
}

- (void) printTriggered
{
  timeval_t  currentTime;

  currentTime = getCurrentTime();
  printf("trigger status of mouse-
trap at x: %.2d, y: %.2d at time: %lu is %d\n",
         xCoord, yCoord, currentTime, triggered );
}

/*
- setDisplayWidget: w {
  displayWidget = w;
}
*/

@end


@implementation Grid2D

+ create: aZone setGridSize: (int)size
{
  Grid2D  *newGrid;

  newGrid = [aZone allocIVars: self];
  newGrid->gridSize = size;
  newGrid->cells = [aZone allocBlock:
                      newGrid->gridSize * newGrid->gridSize * sizeof (id)];
  return newGrid;
}

- (int) getGridSize
{
  return gridSize;
}
```

```
- atX: (int)xCoord atY: (int)yCoord
{
  return cells[ xCoord + yCoord * gridSize];
}

- (void) atX: (int)xCoord atY: (int)yCoord put: anObject
{
  cells[ xCoord + yCoord * gridSize] = anObject;
}

@end
```

Example A-7 grid.h

```
#import <collections.h>

@protocol GridTurtle <Create, Drop, CREATABLE>
- (void) move: (int)distance;
- (void) turn: (int)angle;     /* angle measured in units of pi/2 */
- (void) print;
@end

typedef enum direction { north, east, south, west } direction_t;

externvar id <Symbol>S1;

externvar id <Warning>W1;

externvar id <Error>E1;

#import "grid.xt"
```

Example A-8 grid.m

```
#import "grid.xm"

void _grid_implement( void )
{
  initModule( collections );

  [id_GridTurtle_c setTypeImplemented: GridTurtle];
}

void _grid_initialize( void )
{
  defsymbol( S1 );
  defwarning( W1, 0 );
  deferror( E1, 0 );
}
```

Example A-9 grid0.m

```
#import "GridTurtle.h"

int main(void)
{
  id   aZone, t1;

  initModule( grid );

  aZone = [Zone create: globalZone];

  t1 = [GridTurtle create: aZone];
  [t1 drop];
  t1 = [GridTurtle create: aZone];
  [t1 move: 10];
  [t1 turn: -1];
  [t1 move: 10];
  [t1 print];

  [t1 turn: 23];
  [t1 move: 10];
  [t1 print];

  [aZone xprint];
  [aZone xfprint];
  [aZone drop];
  return 0;
}
```

Example A-10 grid1a.m

```
#import "grid.h"

#define __USE_FIXED_PROTOTYPES__   // for gcc headers
#include <stdio.h>

int main(void)
{
  id   aZone, anArray, newArray, aTurtle, aMember, index;
  int  i;

  initModule( grid );

  aZone = [Zone create: globalZone];

  anArray = [Array create: aZone setCount: 12];
```

```
printf( "count: %d\n", [anArray getCount] );

for ( i = 0; i < 10; i++ ) {
  aTurtle = [GridTurtle create: aZone];
  [aTurtle turn: -1];
  [aTurtle move: i * 10];
  [anArray atOffset: i put: aTurtle];
}

printf( "count: %d\n", [anArray getCount] );

index = [anArray begin: scratchZone];
while ( (aMember = [index next]) || [index getLoc] != End ) {
  if ( ! aMember ) {
    printf( "member is nil\n" );
  } else if ( [aMember respondsTo: M(print)] ) {
    [aMember print];
  } else {
    printf( "%s\n", [aMember getName] );
  }
}
while ( (aMember = [index prev]) || [index getLoc] != Start ) {
  if ( ! aMember ) {
    printf( "member is nil\n" );
  } else if ( [aMember respondsTo: M(print)] ) {
    [aMember print];
  } else {
    printf( "%s\n", [aMember getName] );
  }
}

[anArray setCount: 15];
index = [anArray begin: scratchZone];
while ( (aMember = [index next]) || [index getLoc] != End ) {
  if ( ! aMember ) {
    printf( "member is nil\n" );
  } else if ( [aMember respondsTo: M(print)] ) {
    [aMember print];
  } else {
    printf( "%s\n", [aMember getName] );
  }
}
[index setLoc: Start];

[anArray setCount: 8];
```

```
index = [anArray begin: scratchZone];
while ( (aMember = [index next]) || [index getLoc] != End ) {
  if ( ! aMember ) {
    printf( "member is nil\n" );
  } else if ( [aMember respondsTo: M(print)] ) {
    [aMember print];
  } else {
    printf( "%s\n", [aMember getName] );
  }
}

[anArray atOffset: 2 put: nil];
newArray = [anArray copy: aZone];

index = [newArray begin: scratchZone];
[newArray atOffset: 3 put: nil];
while ( (aMember = [index next]) || [index getLoc] != End ) {
  if ( ! aMember ) {
    printf( "member is nil\n" );
  } else if ( [aMember respondsTo: M(print)] ) {
    [aMember print];
  } else {
    printf( "%s\n", [aMember getName] );
  }
}
[newArray drop];

[anArray atOffset: 5 put: nil];
newArray = [Array createBegin: aZone];
[newArray setInitialValue: anArray];
newArray = [newArray createEnd];
[newArray atOffset: 6 put: nil];

index = [newArray begin: scratchZone];
while ( (aMember = [index next]) || [index getLoc] != End ) {
  if ( ! aMember ) {
    printf( "member is nil\n" );
  } else if ( [aMember respondsTo: M(print)] ) {
    [aMember print];
  } else {
    printf( "%s\n", [aMember getName] );
  }
}
[index drop];
```

```
  index = [anArray begin: scratchZone];
  while ( (aMember = [index next]) || [index getLoc] != End ) {
    if ( ! aMember ) {
      printf( "member is nil\n" );
    } else if ( [aMember respondsTo: M(print)] ) {
      [aMember print];
    } else {
      printf( "%s\n", [aMember getName] );
    }
  }

  for ( i = 0; i < 8; i++ ) {
    aMember = [anArray atOffset: i];
    if ( ! aMember ) {
      printf( "member is nil\n" );
    } else if ( [aMember respondsTo: M(print)] ) {
      [aMember print];
    } else {
      printf( "%s\n", [aMember getName] );
    }
  }
  [anArray xprint];
  [aZone xprint];
  [anArray drop];
  [newArray drop];
  [aZone xprint];
  [aZone xfprint];
  [aZone drop];
  exit(0);
}
```

Example A-11 grid1b.m

```
#import "grid.h"

#define __USE_FIXED_PROTOTYPES__   // for gcc headers
#include <stdio.h>

int main(void)
{
  id   aZone, anArray, aTurtle, aMember, index;
  int  i;

  initModule( grid );

  aZone = [Zone create: globalZone];
```

```
anArray = [Array createBegin: aZone];
[anArray setCount: 12];
[anArray setDefaultMember: Member];
anArray = [anArray createEnd];

printf( "count: %d\n", [anArray getCount] );

for ( i = 0; i < 10; i++ ) {
  aTurtle = [GridTurtle create: aZone];
  [aTurtle turn: -1];
  [aTurtle move: i * 10];
  [anArray atOffset: i put: aTurtle];
}

printf( "count: %d\n", [anArray getCount] );

index = [anArray begin: scratchZone];
while ( (aMember = [index next]) ) {
  if ( ! aMember ) {
    printf( "member is nil\n" );
  } else if ( [aMember respondsTo: M(print)] ) {
    [aMember print];
  } else {
    printf( "%s\n", [aMember getName] );
  }
}
while ( (aMember = [index prev]) ) {
  if ( ! aMember ) {
    printf( "member is nil\n" );
  } else if ( [aMember respondsTo: M(print)] ) {
    [aMember print];
  } else {
    printf( "%s\n", [aMember getName] );
  }
}

[anArray setCount: 15];
index = [anArray begin: scratchZone];
while ( (aMember = [index next]) ) {
  if ( ! aMember ) {
    printf( "member is nil\n" );
  } else if ( [aMember respondsTo: M(print)] ) {
    [aMember print];
  } else {
    printf( "%s\n", [aMember getName] );
```

```
    }
  }
  [index setLoc: Start];

  [anArray setCount: 8];
  index = [anArray begin: scratchZone];
  while ( (aMember = [index next]) ) {
    if ( ! aMember ) {
      printf( "member is nil\n" );
    } else if ( [aMember respondsTo: M(print)] ) {
      [aMember print];
    } else {
      printf( "%s\n", [aMember getName] );
    }
  }

  for ( i = 0; i < 8; i++ ) {
    [[anArray atOffset: i] print];
  }
  [anArray xprint];
  [anArray drop];
  [aZone xprint];
  [aZone drop];
  exit(0);
}
```

Example A-12 grid1c.m

```
#import "grid.h"

#define __USE_FIXED_PROTOTYPES__   // for gcc headers
#include <stdio.h>

int main(void)
{
  id   aZone, anArray, aTurtle, aMember, index, newArray;
  int  i;

  initModule( grid );

  aZone = [Zone create: globalZone];

  anArray = [Array createBegin: aZone];
  [anArray setCount: 10];
  [anArray setDefaultMember: Member];
  anArray = [anArray createEnd];
```

```
printf( "count: %d\n", [anArray getCount] );

for ( i = 0; i < 5; i++ ) {
  aTurtle = [GridTurtle create: aZone];
  [aTurtle turn: -1];
  [aTurtle move: i * 10];
  [anArray atOffset: i put: aTurtle];
}

printf( "count: %d\n", [anArray getCount] );

index = [anArray begin: scratchZone];
while ( (aMember = [index next]) ) {
  if ( ! aMember ) {
    printf( "member is nil\n" );
  } else if ( [aMember respondsTo: M(print)] ) {
    [aMember print];
  } else {
    printf( "%s\n", [aMember getName] );
  }
}

newArray = [Array createBegin: aZone];
[newArray setMemberBlock: (id *)[anArray getMemberBlock] + 2 setCount: 6];
newArray = [newArray createEnd];

index = [newArray begin: scratchZone];
while ( (aMember = [index next]) ) {
  if ( ! aMember ) {
    printf( "member is nil\n" );
  } else if ( [aMember respondsTo: M(print)] ) {
    [aMember print];
  } else {
    printf( "%s\n", [aMember getName] );
  }
}

[newArray setMemberBlock: [anArray getMemberBlock] setCount: 6];

index = [newArray begin: scratchZone];
while ( (aMember = [index next]) ) {
  if ( ! aMember ) {
    printf( "member is nil\n" );
  } else if ( [aMember respondsTo: M(print)] ) {
    [aMember print];
```

```
    } else {
      printf( "%s\n", [aMember getName] );
    }
  }
  [newArray xprint];
  [anArray drop];
  [newArray drop];
  [aZone xprint];
  [aZone drop];
  exit(0);
}
```

Example A-13 grid2.m

```
#import "grid.h"

#define __USE_FIXED_PROTOTYPES__   // for gcc headers
#include <stdio.h>

#import <collections/List_linked.h>
#import <defobj/defalloc.h>

extern id _obj_sessionZone;  // to confirm cleanup of display names

@interface MyList : List_linked
+ create: aZone;
@end

@implementation MyList

+ create: aZone
{
  List_linked  *newList;

  newList = [aZone allocIVars: [List_linked self]];
  setMappedAlloc( newList );
  return newList;
}

@end

int main(void)
{
  id    aZone, aList, aTurtle, index, newList;
  int   i;
  char  buffer[50];
```

```
initModule( grid );

aZone = [Zone create: globalZone];
aList = [MyList create: aZone];

printf( "count: %d\n", [aList getCount] );

for ( i = 0; i < 10; i++ ) {
  aTurtle = [GridTurtle create: aZone];
  [aTurtle turn: -1];
  [aTurtle move: i * 10];
  [aList addLast: aTurtle];
  if ( i % 2 ) {
    sprintf( buffer, "turtle number %d", i );
    [aTurtle setDisplayName: buffer];
  }
}

printf( "count: %d\n", [aList getCount] );

printf( "\n[aZone xfprintid]\n" );
[aZone xfprintid];
printf( "\n" );

index = [aList begin: scratchZone];
while ( (aTurtle = [index next]) ) {
  [aTurtle print];
}
[index drop];

[aList addFirst: [aList removeLast]];
[aList addFirst: [aList removeLast]];
[aList addLast: [aList removeFirst]];

printf( "count: %d\n", [aList getCount] );
index = [aList begin: scratchZone];
for ( [index setLoc: End]; (aTurtle = [index prev]); ) {
  [aTurtle print];
}

printf( "turtle located at offset 3: " );
[[index setOffset: 3] print];
printf( "turtle removed at offset 3: " );
[[index remove] print];
```

```
  [index setLoc: Start];
  while ( (aTurtle = [index next]) );

  [aList forEach: @selector( print )];

  newList = [List createBegin: aZone];
  [newList setInitialValue: aList];
  newList = [newList createEnd];
  [aList drop];

  printf( "new list count: %d\n", [newList getCount] );
  [newList forEach: @selector( print )];

  aList = [newList copy: aZone];
  printf( "copied list count: %d\n", [aList getCount] );
  [aList forEach: @selector( print )];

  printf( "\nsession zone before drop of all turtles\n" );
  [_obj_sessionZone xprint];
  [aList deleteAll];
  [aList drop];

  printf( "\nsession zone after drop\n" );
  [_obj_sessionZone xprint];
  printf( "\n" );

  [newList drop];
  [aZone xprint];
  [aZone drop];
  return 0;
}
```

Example A-14 grid2b.m

```
// This is an old test program for safe indexes mode on a list, which is
// not currently supported.  Now it shows how indexes can be affected by
// remove.

#import "grid.h"
#import "GridTurtle.h"

#define __USE_FIXED_PROTOTYPES__   // for gcc headers
#include <stdio.h>

int main(void)
{
  id   aZone, listType, aList, aTurtle, index, index2;
```

```
int  i;

initModule( grid );

aZone = [Zone create: globalZone];

listType = [List customizeBegin: aZone];
[listType setIndexFromMemberLoc: offsetof( GridTurtle_c, listMembership )];
// [listType setSafeIndexes: 1];
listType = [listType customizeEnd];

aList = [listType create: aZone];

printf( "count: %d\n", [aList getCount] );

for ( i = 0; i < 10; i++ ) {
  aTurtle = [GridTurtle create: aZone];
  [aTurtle turn: -1];
  [aTurtle move: i * 10];
  [aList addLast: aTurtle];
}

printf( "count: %d\n", [aList getCount] );

index = [aList begin: scratchZone];
while ( (aTurtle = [index next]) ) {
  [aTurtle print];
}
[index drop];

[aList addFirst: [aList removeLast]];
[aList addFirst: [aList removeLast]];
[aList addLast: [aList removeFirst]];

index = [aList begin: scratchZone];
while ( (aTurtle = [index next]) ) {
  [aTurtle print];
}
[aList addLast: [aList removeFirst]];

printf( "count: %d\n", [aList getCount] );
index = [aList createIndex: scratchZone fromMember: [aList atOffset: 8]];
printf( "offset after createIndex:fromMember: %d\n", [index getOffset] );
[index next];
[index next];
```

```
[index prev];
printf( "offset after next to End then prev: %d\n", [index getOffset] );

index = [aList createIndex: scratchZone fromMember: [aList atOffset: 1]];
[index setOffset: 1];
index2 = [aList createIndex: scratchZone fromMember: [aList atOffset: 0]];
[index remove];
[index2 remove];
printf( "offset after remove on both index and lesser index: %d\n",
        [index getOffset] );
printf( "member after removed offsets 0 and 1:\n  " );
[[index next] print];

index = [aList createIndex: scratchZone fromMember: [aList atOffset: 0]];
index2 = [aList createIndex: scratchZone fromMember: [aList atOffset: 1]];
[index remove];
[index2 remove];
printf( "offset after remove on both index and greater index: %d\n",
        [index getOffset] );
printf( "member after removed offsets 0 and 1:\n  " );
[[index next] print];

index = [aList createIndex: scratchZone fromMember: [aList atOffset: 1]];
index2 = [aList createIndex: scratchZone fromMember: [aList atOffset: 0]];
[index2 remove];
printf( "offset after remove on lesser index: %d\n",
        [index getOffset] );
printf( "member after removed offset 0 and original offset 1:\n  " );
[[index next] print];

index = [aList createIndex: scratchZone fromMember: [aList atOffset: 1]];
index2 = [aList createIndex: scratchZone fromMember: [aList atOffset: 1]];
[index2 remove];
printf( "offset after remove on same index: %d\n",
        [index getOffset] );
printf( "member after removed offset 1:\n  " );
[[index next] print];

index = [aList createIndex: scratchZone fromMember: [aList atOffset: 0]];
index2 = [aList createIndex: scratchZone fromMember: [aList atOffset: 1]];
[index2 remove];
printf( "offset after remove on greater index: %d\n",
        [index getOffset] );
printf( "member after original offset 0 and removed offset 1:\n  " );
[[index next] print];
```

```
  index = [aList createIndex: scratchZone fromMember: [aList atOffset: 0]];
  [index setLoc: Start];
  index2 = [aList createIndex: scratchZone fromMember: [aList atOffset: 0]];
  [index2 remove];
  printf( "offset after move to Start and remove on other index: %d\n",
          [index getOffset] );
  printf( "member after removed offset 0:\n  " );
  [[index next] print];

  printf( "remaining members of list:\n" );
  index = [aList begin: scratchZone];
  while ( (aTurtle = [index next]) ) {
    [aTurtle print];
  }
  printf( "location of index at End: %s\n", [[index getLoc] getName] );

  [index setOffset: 1];
  printf( "location of index at offset 1: %s\n", [[index getLoc] getName] );
  [index remove];
  printf( "location of index after remove: %s\n", [[index getLoc] getName] );
  [index prev];
  printf( "location of index after prev: %s\n", [[index getLoc] getName] );

  [index setOffset: 0];
  printf( "location of index at offset 0: %s\n", [[index getLoc] getName] );
  [index remove];
  printf( "location of index after remove: %s\n", [[index getLoc] getName] );
  [index next];
  printf( "location of index after next: %s\n", [[index getLoc] getName] );

  [aList xprint];
  [aList drop];
  [aZone xprint];
  [aZone drop];
  return 0;
}
```

Example A-15 grid3.m

```
#import "grid.h"

#define __USE_FIXED_PROTOTYPES__   // for gcc headers
#include <stdio.h>

int main(void)
{
```

```
  id    aZone, aMap, aTurtle, key, index;
  int   i;
  char  buffer[10];

  initModule( grid );

  aZone = [Zone create: globalZone];
  aMap  = [Map create: aZone];

  printf( "count: %d\n", [aMap getCount] );

  for ( i = 0; i < 10; i++ ) {
    aTurtle = [GridTurtle create: aZone];
    [aTurtle turn: -1];
    [aTurtle move: i * 10];
    sprintf( buffer, "%d", i );
    key = [String create: aZone setC: buffer];
    [aMap at: key insert: aTurtle];
  }

  printf( "count: %d\n", [aMap getCount] );

  index = [aMap begin: scratchZone];
  while ( (aTurtle = [index next]) ) {
    key = [index getKey];
    printf( "%s: ", [key getC] );
    [aTurtle print];
    [key drop];
  }

  [[aMap getFirst] print];
  [[aMap getLast] print];

  [aMap xprint];
  [aMap drop];
  [aZone xprint];
  [aZone drop];
  return 0;
}
```

Example A-16 grid3b.m

```
#import "grid.h"

#define __USE_FIXED_PROTOTYPES__   // for gcc headers
#include <stdio.h>
```

```
int main(void)
{
  id   aZone, aMap, aTurtle, index;
  int   i;

  initModule( grid );

  aZone = [Zone create: globalZone];
  aMap  = [Map createBegin: aZone];
  [aMap setCompareFunction: compareIntegers];
  aMap = [aMap createEnd];

  printf( "count: %d\n", [aMap getCount] );

  for ( i = 0; i < 10; i++ ) {
    aTurtle = [GridTurtle create: aZone];
    [aTurtle turn: -1];
    [aTurtle move: (10 - i) * 10];
    [aMap at: (id)(10 - i) insert: aTurtle];
  }

  printf( "count: %d\n", [aMap getCount] );

  index = [aMap begin: scratchZone];
  while ( (aTurtle = [index next]) ) {
    printf( "%d: ", (int)[index getKey] );
    [aTurtle print];
  }
  [index drop];

  [[aMap getFirst] print];
  [[aMap getLast] print];

  [aMap xprint];
  [aMap drop];
  [aZone xprint];
  [aZone drop];
  return 0;
}
```

Example A-17 grid4.m

```
#import "grid.h"

#define __USE_FIXED_PROTOTYPES__   // for gcc headers
#include <stdio.h>
```

```
int main(void)
{
  id aZone, aSet, aTurtle, turtles[10], index;
  int i;

  initModule( grid );

  aZone = [Zone create: globalZone];
  aSet  = [Set create: aZone];

  printf( "count: %d\n", [aSet getCount] );

  for ( i = 0; i < 10; i++ ) {
    aTurtle = [GridTurtle create: aZone];
    turtles[i] = aTurtle;
    [aTurtle turn: -1];
    [aTurtle move: i * 10];
    if ( i & 1 ) [aSet add: aTurtle];
  }

  printf( "count: %d\n", [aSet getCount] );

  index = [aSet begin: aZone];
  while ( (aTurtle = [index next]) ) {
    [aTurtle print];
  }
  for ( i = 0; i < 10; i++ ) {
    if ( [aSet at: turtles[i]] != turtles[i] ) {
      printf( "turtle not found at %d\n", i );
    }
  }

  for ( i = 0; i < 10; i++ ) {
    if ( i & 1 ) {
      [aSet remove: turtles[i]];
    } else {
      [aSet add: turtles[i]];
    }
  }
  index = [aSet begin: scratchZone];
  while ( (aTurtle = [index next]) ) {
    [aTurtle print];
  }
  for ( i = 0; i < 10; i++ ) {
    if ( [aSet at: turtles[i]] != turtles[i] ) {
```

```
      printf( "turtle not found at %d\n", i );
    }
  }
  [index drop];

  [aSet xprint];
  [aSet drop];
  [aZone xprint];
  [aZone drop];
  return 0;
}
```

Example A-18 grid4b.m

```
#import "grid.h"
#import "GridTurtle.h"

#define __USE_FIXED_PROTOTYPES__   // for gcc headers
#include <stdio.h>

int main(void)
{
  id   aZone, anOrderedSet, aTurtle, turtles[10], index;
  int  i;

  initModule( grid );

  aZone = [Zone create: globalZone];

  anOrderedSet = [OrderedSet createBegin: aZone];
  [anOrderedSet setIndexFromMemberLoc:
    offsetof( GridTurtle_c, listMembership )];
  anOrderedSet = [anOrderedSet createEnd];

  printf( "count: %d\n", [anOrderedSet getCount] );

  for ( i = 0; i < 10; i++ ) {
    aTurtle = [GridTurtle create: aZone];
    turtles[i] = aTurtle;
    [aTurtle turn: -1];
    [aTurtle move: i * 10];
    if ( i & 1 ) [anOrderedSet add: aTurtle];
  }

  printf( "count: %d\n", [anOrderedSet getCount] );

  index = [anOrderedSet begin: scratchZone];
```

```
  while ( (aTurtle = [index next]) ) {
    [aTurtle print];
  }
  for ( i = 0; i < 10; i++ ) {
    if ( ! [anOrderedSet contains: turtles[i]] ) {
      printf( "turtle not found at %d\n", i );
    }
  }

  for ( i = 0; i < 10; i++ ) {
    if ( i % 2 ) {
      [anOrderedSet remove: turtles[i]];
    } else {
      [anOrderedSet add: turtles[i]];
    }
  }
  index = [anOrderedSet begin: scratchZone];
  while ( (aTurtle = [index next]) ) {
    [aTurtle print];
  }
  for ( i = 0; i < 10; i++ ) {
    if ( ! [anOrderedSet contains: turtles[i]] ) {
      printf( "turtle not found at %d\n", i );
    }
  }

  [anOrderedSet xprint];
  [anOrderedSet drop];
  [aZone xprint];
  [aZone drop];
  return 0;
}
```

Example A-19 grid5.m

```
#import "grid.h"
#import <activity.h>
#import <objectbase.h>
#import <misc.h>

static BOOL trace( id anActivity );

int main(void)
{
  id   aZone, aList, aTurtle;
  id   emptyActions, moveActions, printActions, moveAndPrint;
  id   groupActivity, swarm, swarmActivity;
```

```
id   <Symbol>status;
int  i, stepCount;

initModule( grid );
initModule( activity );

aZone = [Zone create: globalZone];
swarm = [Swarm create: aZone];
_activity_zone = [Zone create: globalZone];

[_activity_zone xprint];
swarmActivity = [swarm activateIn: nil];
[swarmActivity drop];
[_activity_zone xprint];

//  if ( _obj_debug ) exit(0);

aList = [List create: swarm];

for ( i = 0; i < 10; i++ ) {
  aTurtle = [GridTurtle create: swarm];
  [aTurtle turn: -1];
  [aTurtle move: i * 10];
  [aTurtle turn: 1];
  [aList addLast: aTurtle];
}

emptyActions = [ActionGroup create: swarm];

moveActions = [ActionGroup create: swarm];
for ( i = 0; i < 10; i++ ) {
  if ( i % 2 )
    [moveActions
      createActionTo: [aList atOffset: i] message: M(move:) : (id)10 ];
}

printActions = [ActionGroup create: swarm];
for ( i = 0; i < 10; i++ ) {
  [printActions createActionTo: [aList atOffset: i] message: M(print)];
}

moveAndPrint = [ActionGroup create: swarm];
[moveAndPrint createAction: emptyActions];
[moveAndPrint createAction: moveActions];
[moveAndPrint createAction: printActions];
```

```
  for ( i = 0; i < 10; i++ ) {
    if ( i % 2 )
      [moveAndPrint
        createActionTo: [aList atOffset: i] message: M(move:) : (id)10];
  }
  [moveAndPrint createActionForEach: aList message: M(print)];

  groupActivity = [moveAndPrint activateIn: nil];
  printf("about to start schedule\n");
  while ( (status = [groupActivity run]) != Completed );
  printf( "run return status: %s\n\n", [status getName] );
  [groupActivity drop];

  swarmActivity = [swarm activateIn: nil];
  [moveAndPrint activateIn: swarmActivity];

  if ( swarmActivity != [swarm getSwarmActivity] )
    raiseEvent( WarningMessage, nil );

  _activity_trace = trace;

  for ( stepCount = 1; [swarmActivity step] == Stopped; stepCount++ );
  printf( "number of steps: %d\n", stepCount );

  [swarmActivity drop];
  [_activity_zone xprint];

  [emptyActions drop];
  [moveActions drop];
  [printActions drop];
  [moveAndPrint drop];
  [aList deleteAll];
  [aList drop];
  [[swarm getInternalZone] xprint];

  [swarm drop];
  [aZone xprint];
  return 0;
}

static BOOL trace( id anActivity )
{
  char  buffer[100];

  if ( getCurrentAction() ) {
```

```
    _obj_formatIDString( buffer, getCurrentAction() );
    printf( "action trace: %s\n", buffer );
  } else {
    printf( "action trace: activity status %s\n",
      [[anActivity getStatus] getName] );
  }
  return 0;
}
```

Example A-20 grid6.m

```
#import "grid.h"
#import <activity.h>
#import <objectbase.h>
#import <defobj/DefObject.h>      // for dropAllocations: of internal actions
#import <activity/SwarmProcess.h> // activate:

#include <misc.h>

static void terminateFunction( void )
{
  [getTopLevelActivity() terminate];
}

static void nestedFunction( void )
{
  char buf1[100], buf2[100];

  _obj_formatIDString( buf1, getTopLevelActivity() );
  _obj_formatIDString( buf2, [getTopLevelActivity() getControllingActiv-
ity] );

  printf( "top level activity: %s\ncontrolling activity: %s\n", buf1, buf2 );
}

static void runFunction( void )
{
  id  group, groupActivity;

  group = [ActionGroup create: globalZone];
  [group createActionCall: (func_t)printf : (id)"nested group performed\n"];
  [group createActionCall: nestedFunction];
  groupActivity = [group activateIn: nil];
  [groupActivity run];
  [groupActivity drop];
  [group drop];
}
```

```
int main(void)
{
  id  x, aZone, sched1, sched2, action1, action2;
  id  swarm;
  id  <Activity>swarmActivity;
  id  <Symbol>status;

  initModule( grid );
  initModule( activity );

  aZone = [Zone create: globalZone];
  swarm = [Swarm create: aZone];
  _activity_zone = [Zone create: globalZone];

  sched1 = [Schedule create: swarm];
  [sched1 at: 1 createActionCall: terminateFunction];

  x = [Schedule createBegin: swarm];
  [x setRepeatInterval: 10];
  sched2 = [x createEnd];

  [sched2 createActionCall: (func_t)printf :
    (id)"repeating action performed\n"];
  [sched2 createActionCall: (func_t)runFunction];

  // create a concurrent group just to make sure it goes away

  action1 = [sched2 at: 1 createAction-
Call: (func_t)printf : (id)"action 1\n"];
  action2 = [sched2 at: 1 createAction-
Call: (func_t)printf : (id)"action 2\n"];

  [sched2 remove: action1];
  [sched2 remove: action2];

  swarmActivity = [swarm activateIn: nil];
  [swarm activate: sched1];
  [swarm activate: sched2];

  while ( (status = [swarmActivity run]) != Stopped  && (status != Com-
pleted) );

  [swarmActivity drop];
  [_activity_zone xprint];
```

```
#if 0
  [sched1 drop];
  [sched2 drop];
#endif
  [action1 drop];
  [action2 drop];
  [[swarm getInternalZone] xprint];

  [swarm drop];
  [aZone xprint];
  return 0;
}
```

Example A-21 grid7.m

```
#import "grid.h"
#import <activity.h>
#import <activity/XActivity.h>
#import <objectbase.h>

#include <misc.h>

extern BOOL respondsTo( id, SEL );  // from defobj

static BOOL trace( id anActivity );

static id newTurtle( void )
{
  id          aTurtle;
  static int  ycoord;

  aTurtle = [GridTurtle create: globalZone];
  [aTurtle move: ++ycoord];
  return aTurtle;
}

int main(void)
{
  id   aZone, aList, aTurtle;
  id   emptyActions1, emptyActions2;
  id   moveAndPrint, moveActions, printActions;
  id   printRelative, printRepeat, startSchedule;
  id   swarm, swarmActivity;
  id   <Symbol>status;
  int  i;
```

```
initModule( grid );
initModule( activity );

aZone = [Zone create: globalZone];
swarm = [Swarm create: aZone];
_activity_zone = [Zone create: globalZone];

aList = [List create: swarm];

for ( i = 0; i < 10; i++ ) {
  aTurtle = [GridTurtle create: swarm];
  [aTurtle turn: -1];
  [aTurtle move: i * 10];
  [aTurtle turn: 1];
  [aList addLast: aTurtle];
}

moveActions = [ActionGroup create: swarm];
for ( i = 0; i < 10; i++ ) {
  if ( i % 2 )
    [moveActions
      createActionTo: [aList atOffset: i] message: M(move:) : (id)10 ];
}

printActions = [ActionGroup create: swarm];
for ( i = 0; i < 10; i++ ) {
  [printActions createActionTo: [aList atOffset: i] message: M(print)];
}

moveAndPrint = [Schedule createBegin: swarm];
[moveAndPrint setRepeatInterval: 10];
moveAndPrint = [moveAndPrint createEnd];

[moveAndPrint at: 0 createAction: moveActions];
[moveAndPrint at: 0 createAction: printActions];

emptyActions1 = [ActionGroup create: swarm];
emptyActions2 = [ActionGroup create: swarm];

printRelative = [Schedule createBegin: swarm];
[printRelative setRelativeTime: 1];
printRelative = [printRelative createEnd];

[printRelative at: 0 createActionTo: newTurtle() message: M(print)];
[printRelative at: 4 createAction: emptyActions1];
```

```
[printRelative at: 4 createActionTo: newTurtle() message: M(print)];
[printRelative at: 9 createActionTo: newTurtle() message: M(print)];

printRepeat = [Schedule createBegin: swarm];
[printRepeat setRepeatInterval: 10];
printRepeat = [printRepeat createEnd];

[printRepeat at: 0 createActionTo: newTurtle() message: M(print)];
[printRepeat at: 1 createActionTo: newTurtle() message: M(print)];

startSchedule = [Schedule create: swarm];
[startSchedule at: 1 createAction: printRepeat];
[startSchedule at: 2 createActionTo: newTurtle() message: M(print)];
[startSchedule at: 2 createAction: emptyActions2];

[startSchedule at: 2 createAction: printRelative];
[startSchedule at: 18 createAction: printRelative];
[startSchedule at: 21 createAction: printRelative];
[startSchedule at: 34 createAction: printRelative];

[startSchedule at: 5 createActionTo: newTurtle() message: M(print)];
[startSchedule at: 12 createActionTo: newTurtle() message: M(print)];
[startSchedule at: 18 createActionTo: newTurtle() message: M(print)];
[startSchedule at: 10 createActionTo: newTurtle() message: M(print)];
[startSchedule at: 20 createActionTo: newTurtle() message: M(print)];

_activity_trace = trace;
swarmActivity = [swarm activateIn: nil];
[startSchedule activateIn: swarmActivity];
//  [swarm activate: startSchedule];

while ( (status = [swarmActivity run]) != Stopped && status != Completed );
printf( "run return status: %s\n", [status getName] );

[swarmActivity drop];
[_activity_zone xprint];

[moveActions drop];
[printActions drop];
[moveAndPrint drop];
[emptyActions1 drop];
[emptyActions2 drop];

[printRelative drop];
[printRepeat drop];
```

```
  [startSchedule drop];

  [aList deleteAll];
  [aList drop];
  [[swarm getInternalZone] xprint];
  [swarm drop];
  return 0;
}

static BOOL trace( id anActivity )
{
  id          swarmActivity, string;
  int         level;
  timeval_t   currentTime = 999999;
  char        buffer[100];

  for ( swarmActivity = anActivity, level = 0;
        ! respondsTo( swarmActivity, @selector( getSynchronizationSched-
ule ) );
        swarmActivity = [swarmActivity getOwnerActivity], level++ );

  currentTime = getCurrentTime();

  if ( swarmActivity != getCurrentSwarmActivity() ||
       currentTime != [getCurrentScheduleActivity() getCurrentTime] )
    raiseEvent( SourceMessage, "context macro mismatch\n" );

  _obj_formatIDString( buffer, anActivity );
  if ( [anActivity getStatus] == Running ) {
    [getCurrentAction() describe: (string = [String create: scratchZone])];
    printf( "at time: %lu, level %d, %s, %s",
             currentTime, level, buffer, [string getC] );
    [string drop];
  } else {
    printf( "at time: %lu, level %d, %s is %s\n",
             currentTime, level, buffer, [[anActivity getStatus] getName] );
  }
  if ( currentTime < 100 ) return 0;
  ((Activity_c *)anActivity)->status = Stopped;
  return 1;
}
```

Example A-22 grid8.m

```
#import "grid.h"
#import <activity/SwarmProcess.h>
#import <activity.h>
```

```
#import <objectbase.h>
#include <misc.h>

static void terminateFunction( void )
{
  [getTopLevelActivity() terminate];
}

static void stopFunction( void )
{
  printf( "stopping activity at time: %lu\n", getCurrentTime() );
  [getTopLevelActivity() stop];
}

int main(void)
{
  id   aZone, aTurtle;
  id   simSched1, simSched2, displaySchedule, terminateSchedule;
  id   simSwarm, obsSwarm;
  id   <Activity>obsSwarmActivity;

  // initialize

  initModule( grid );
  initModule( activity );

  aZone = [Zone create: globalZone];
  obsSwarm = [Swarm create: aZone];
  simSwarm = [Swarm create: obsSwarm];

  _activity_zone = [Zone create: globalZone];

  // create a turtle to turn and move during simulation

  aTurtle = [GridTurtle create: simSwarm];

  // create a pair of simulation schedules (repeat intervals of 2 and 4)

  simSched1 = [Schedule createBegin: simSwarm];
  [simSched1 setRepeatInterval: 2];
  simSched1 = [simSched1 createEnd];
  [simSched1 at: 0 createActionTo: aTurtle message: M(move:) : (id)10];

  simSched2 = [Schedule createBegin: simSwarm];
  [simSched2 setRepeatInterval: 4];
```

```
  simSched2 = [simSched2 createEnd];
  [simSched2 at: 0 createActionTo: aTurtle message: M(turn:) : (id)1];

  // create a display schedule to print turtle at every time step

  displaySchedule = [Schedule createBegin: obsSwarm];
  [displaySchedule setRepeatInterval: 1];
  displaySchedule = [displaySchedule createEnd];
  [displaySchedule at: 0 createActionTo: aTurtle message: M(print)];

  // create a schedule to terminate all activity at timestep 32

  terminateSchedule = [Schedule create: obsSwarm];
  [terminateSchedule at: 8 createActionCall: stopFunction];
  [terminateSchedule at: 11 createActionCall: terminateFunction];

  obsSwarmActivity = [obsSwarm activateIn: nil];
  [obsSwarm activate: terminateSchedule];
  [obsSwarm activate: displaySchedule];

  [obsSwarm activate: simSwarm];
  [simSwarm activate: simSched1];
  [simSwarm activate: simSched2];

  // run the top-level swarm to completion

  while ( [obsSwarmActivity next] != Completed );

  [obsSwarmActivity drop];
  [_activity_zone xprint];

  [aTurtle drop];
#if 0
  [simSched1 drop];
  [simSched2 drop];
#endif
  [simSwarm drop];
#if 0
  [displaySchedule drop];
  [terminateSchedule drop];
#endif

  [[obsSwarm getInternalZone] xprint];
  [[obsSwarm getInternalZone] xfprint];
  [obsSwarm drop];
```

```
  [aZone xprint];
  [aZone drop];
  return 0;
}
```

Example A-23 grid9.m

```
#import "grid.h"
#import <activity.h>
#import <activity/SwarmProcess.h>
#import <objectbase/Swarm.h>
#import <objectbase.h>
#include <misc.h>

static void testTerminate( timeval_t terminateTime )
{
  if ( getCurrentTime() >= terminateTime )
    [getTopLevelActivity() terminate];
}

@interface MySwarm: Swarm
{
  int  myVar;
}
- (void) setMyVar: (int)value;
- (void) printMyVar;
@end

@implementation MySwarm

- (void) setMyVar: (int)value
{
  myVar = value;
}

- (void) printMyVar
{
  printf( "value is %d\n", myVar );
}

@end

int main(void)
{
  id   aZone, aTurtle;
  id   simSched1, action1, action2, action3, moveGroup;
```

```
  id   simSched2, displaySchedule, terminateSchedule;
  id   displaySwarm, simSwarm;
  id   <Activity>swarmActivity;

  // initialize

  initModule( grid );
  initModule( activity );

  aZone = [Zone create: globalZone];
  displaySwarm = [Swarm create: aZone];
  simSwarm = [MySwarm create: displaySwarm];
  _activity_zone = [Zone create: globalZone];

  // create a turtle to turn and move during simulation

  aTurtle = [GridTurtle create: simSwarm];

  // create a pair of simulation schedules (repeat intervals of 2 and 4)

  simSched1 = [Schedule createBegin: simSwarm];
  [simSched1 setRepeatInterval: 2];
  simSched1 = [simSched1 createEnd];
  action1 = [simSched1 at: 0 createActionTo: aTurtle mes-
sage: M(move:):(id)10];
  moveGroup = [ActionGroup create: simSwarm];
  action2 = [moveGroup createActionTo: aTurtle message: M(move:) : (id)-5];
  [simSched1 at: 0 createAction: moveGroup];
  [simSched1 at: 0 createActionCall: (func_t)testTerminate : (id)20];

  simSched2 = [Schedule createBegin: simSwarm];
  [simSched2 setRepeatInterval: 4];
  simSched2 = [simSched2 createEnd];
  [simSched2 at: 0 createActionTo: aTurtle message: M(turn:) : (id)1];
  action3 = [simSched2 at: 0 createAction-
Call: (func_t)testTerminate : (id)10];

  // create a display schedule to print turtle at every time step

  displaySchedule = [Schedule createBegin: displaySwarm];
  [displaySchedule setRepeatInterval: 1];
  displaySchedule = [displaySchedule createEnd];
  [displaySchedule createActionTo: aTurtle message: M(print)];

  // create a schedule to terminate after a while
```

```
  terminateSchedule = [Schedule create: displaySwarm];
  [terminateSchedule at: 10
    createActionTo: moveGroup message: M(remove:) : action2];
  [terminateSchedule at: 4
    createActionTo: simSched2 message: M(remove:) : action3];
  [terminateSchedule at: 16 createAction-
Call: (func_t)testTerminate : (id)16];

  // activate top-level swarm

  swarmActivity = [displaySwarm activateIn: nil];

  // manipulate custom swarm and then activate it

  [simSwarm setMyVar: 99];
  [simSwarm printMyVar];

  [simSwarm activateIn: displaySwarm];
  [[[simSwarm getSwarmActivity] getSwarm] printMyVar];

  // activate simulation schedules in simulation swarm

  [simSched1 activateIn: simSwarm];
  [simSched2 activateIn: simSwarm];

  // activate display schedules to mix in with simulation swarm

  [displaySchedule activateIn: displaySwarm];
  [terminateSchedule activateIn: displaySwarm];

  // run the top-level swarm to completion

  while ( [swarmActivity next] != Completed );

  [swarmActivity drop];
  [_activity_zone xprint];
  return 0;
}
```

Example A-24 mousetraps.m

```
#import "Mousetrap.h"
#import <objectbase.h>

#include <misc.h>
```

```
int gridSize              = 40;
int printTriggerEvents    = 1;
int printSameTickTriggers = 0;


// simulation parameters (these need to be a power of 2)

int triggerLikelihood    = 256;
int numberOutputTriggers = 2;
int maxTriggerDistance   = 4;
int maxTriggerTime       = 16;

// data from the mousetrap world

unsigned numTriggered = 0;

id grid, worldSchedule;

static void testTerminate( void )
{
  if ( ! [worldSchedule getCount] ) [getTopLevelActivity() terminate];
}

int main(int argc, char ** argv)
{
  id   aMousetrap, worldSwarm;
  id   displaySchedule, observerSwarm, observerActivity, status;
  int  x, y, stepCount;

  initModule( activity );

  observerSwarm = [Swarm create: globalZone];
  worldSwarm = [Swarm create: observerSwarm];
  _activity_zone = [Zone create: globalZone];

  // put a mousetrap on every grid point

  grid = [Grid2D create: worldSwarm setGridSize: gridSize];
  for ( y = 0; y < gridSize; y ++ ) {
    for ( x = 0; x < gridSize; x++ ) {
      aMousetrap = [Mousetrap create: worldSwarm setXCoord: x setYCoord: y];
      [grid atX: x atY: y put: aMousetrap];
    }
  }
```

```
  // create the mousetrap world schedule and schedule its first action

  worldSchedule = [Schedule createBegin: worldSwarm];
  [worldSchedule setAutoDrop: 1];
  worldSchedule = [worldSchedule createEnd];

  aMousetrap = [grid atX: gridSize/2 atY: gridSize/2];
  [worldSchedule at: 0 createActionTo: aMousetrap message: M(trigger)];

  // create display schedule and schedule the display events

  displaySchedule = [Schedule createBegin: observerSwarm];
  [displaySchedule setRepeatInterval: 10];
  displaySchedule = [displaySchedule createEnd];

  [displaySchedule at: 0 createActionTo:
    [grid atX: (gridSize + gridSize/2)/(gridSize/2) atY: gridSize/2]
    message: M(printTriggered)];
  [displaySchedule at: 0 createActionCall: (func_t)testTerminate];

  // create a swarm to combine the display schedule and the world swarm

  observerActivity = [observerSwarm activateIn: nil];
  [displaySchedule activateIn: observerSwarm];

  [worldSwarm activateIn: observerSwarm];
  [worldSchedule activateIn: worldSwarm];

  // [observerActivity run];

  status = [observerActivity stepUntil: 12];
  printf( "returned from stepUntil: 12 with status %s\n", [status getName] );
  status = [observerActivity stepUntil: 2];
  printf( "returned from stepUntil: 2 with status %s\n", [status getName] );

  for ( stepCount = 0; [observerActivity step] == Stopped; stepCount++ );
  printf( "number of steps: %d\n", stepCount );
  printf( "total number of mousetraps triggered: %d\n", numTriggered );
  return 0;
}
```

Example A-25 mousetraps2.m

```
#import "Mousetrap.h"
#import <objectbase.h>
#include <misc.h>
```

```
int gridSize              = 40;
int printTriggerEvents    = 1;
int printSameTickTriggers = 0;


// simulation parameters (these need to be a power of 2)

int triggerLikelihood    = 256;
int numberOutputTriggers = 2;
int maxTriggerDistance   = 4;
int maxTriggerTime       = 16;

// data from the mousetrap world

unsigned numTriggered = 0;

id grid, worldSchedule;

static void testTerminate( void )
{
  int  remainingCount;

  remainingCount = [worldSchedule getCount];
  if ( ! [worldSchedule getCount] ) [getTopLevelActivity() terminate];
}

int main(int argc, char ** argv)
{
  id   aZone, aMousetrap, subschedType, subgroup;
  id   displaySchedule, observerSwarm, observerActivity;
  id   worldSwarm, worldActivity;
  int  x, y, stepCount;

  initModule( activity );

  aZone = globalZone;

  // put a mousetrap on every grid point

  grid = [Grid2D create: aZone setGridSize: gridSize];
  for ( y = 0; y < gridSize; y ++ ) {
    for ( x = 0; x < gridSize; x++ ) {
      aMousetrap = [Mousetrap create: aZone setXCoord: x setYCoord: y];
      [grid atX: x atY: y put: aMousetrap];
    }
```

```
  }

  // create the mousetrap world schedule and schedule its first action

  subschedType = [ConcurrentSchedule customizeBegin: aZone];
  [subschedType setAutoDrop: 1];
  subschedType = [subschedType customizeEnd];

  worldSchedule = [Schedule createBegin: aZone];
  [worldSchedule setAutoDrop: 1];
  [worldSchedule setConcurrentGroupType: subschedType];
  worldSchedule = [worldSchedule createEnd];

  aMousetrap = [grid atX: gridSize/2 atY: gridSize/2];
  subgroup = [worldSchedule insertGroup: (id)0];
  [subgroup at: 0 createActionTo: aMousetrap message: M(trigger)];

  // create display schedule and schedule the display events

  displaySchedule = [Schedule createBegin: aZone];
  [displaySchedule setRepeatInterval: 1];
  displaySchedule = [displaySchedule createEnd];

  [displaySchedule at: 0 createActionTo:
    [grid atX: (gridSize + gridSize/2)/(gridSize/2) atY: gridSize/2]
    message: M(printTriggered)];
  [displaySchedule at: 0 createActionCall: (func_t)testTerminate];

  // create a swarm to combine the display schedule and the world swarm

  observerSwarm = [Swarm create: aZone];
  observerActivity = [observerSwarm activateIn: nil];
  [displaySchedule activateIn: observerSwarm];

  worldSwarm = [Swarm create: aZone];
  worldActivity = [worldSwarm activateIn: observerSwarm];
  subgroup = [worldSchedule activateIn: worldSwarm];

  // [observerActivity run];
  for ( stepCount = 0; [observerActivity step] == Stopped; stepCount++ );
  printf( "number of steps: %d\n", stepCount );
  printf( "total number of mousetraps triggered: %d\n", numTriggered );
  return 0;
}
```

Example A-26 strtest.m

```
#import <collections.h>

#define __USE_FIXED_PROTOTYPES__   // for gcc headers
#include <stdio.h>

int main( void )
{
  id  aZone, s1, s2, s3, t1, FooBarString, s4;

  initModule( collections );

  aZone = [Zone create: globalZone];

  t1 = [String customizeBegin: aZone];
  [t1 setC: "foobar"];
  FooBarString = [t1 customizeEnd];
  printf( "FooBarString id is: %0#8x\n", (unsigned int)FooBarString );
  [FooBarString drop];
  [aZone xprint];
  if ( _obj_debug ) exit(0);

  s1 = [String create: aZone setC: "abcdef"];
  printf( "string is: %s\n", [s1 getC] );
  [s1 setC: "foo"];
  printf( "string is: %s\n", [s1 getC] );
  [s1 catC: "bar"];
  printf( "string is: %s\n", [s1 getC] );
  [s1 drop];

  s2 = [String create: aZone];
  [s2 setC: ""];
  printf( "string is: %s\n", [s2 getC] );
  [s2 catC: "foo"];
  printf( "string is: %s\n", [s2 getC] );
  [s2 drop];

  s3 = [String create: aZone];
  printf( "string is: %s\n", [s3 getC] );
  [s3 catC: "foo"];
  printf( "string is: %s\n", [s3 getC] );
  s4 = [s3 perform: @selector( copy: ) with: aZone];
  [s4 catC: "FOO"];
  printf( "string is: %s\n", [s4 getC] );
  [s3 drop];
```

```
  [s4 drop];
  [aZone xprint];

  t1 = [String customizeBegin: aZone];
  [t1 setC: "foobar"];
  FooBarString = [t1 customizeEnd];
  printf( "FooBarString id is: %0#8x\n", (unsigned int)FooBarString );
  [FooBarString drop];
  [aZone xprint];
  if ( _obj_debug ) exit(0);

  s4 = [FooBarString create: aZone];
  printf( "string is: %s  id is: %0#8x\n", [s4 getC], (unsigned int)s4 );
  [s4 catC: "foo"];
  printf( "string is: %s\n", [s4 getC] );

  s1 = [OutputStream create: scratchZone setFileStream: _obj_xdebug];
  [s1 catC: "output string\n"];
  [s1 drop];

  s1 = [OutputStream createBegin: scratchZone];
  [s1 setFileStream: _obj_xdebug];
  s1 = [s1 createEnd];
  [s1 catC: "output string\n"];
  [s1 drop];

  [s4 xprint];
  [s4 setC: "long output string that just keeps going and go-
ing and going\n"];
  [s4 xprint];
  [s4 drop];
  [FooBarString xprint];
  [FooBarString drop];
  [aZone xprint];
  [aZone drop];
  return 0;
}
```

Appendix B. Random Library Appendix

B.1. Randomness for neophytes

Why randomness? While much interesting work can be done with completely deterministic models, sooner or later you will want to simulate some real-life stochastic phenomenon which occurs in a manner resembling an identifiable statistical distribution, for example (to take the canonical simulation example) the time intervals between customers arriving to join a queue in front of a bank teller. Or perhaps you just want to add some controlled unpredictability to the behavior of your agents.

For these purposes, we need random generators.

A "random generator" is a device or algorithm that puts out "random" numbers. Without going into a long philosophical discussion about the true meaning of randomness, this means that the device or program puts out numbers that are such that even knowing the whole past output of the device or program does not help us predict the next number to come out.

A device that provides truly random numbers in this sense can be built; it would have to rely on such natural processes as counting cosmic particles hitting the Earth, or digitizing pictures of a Lava lamp. Few computer applications actually make use of such truly random devices. Most current cryptography applications that need randomness rely on collecting various ‘random’ (i.e. unpredictable to an outsider) data from your computer, such as the time between your keystrokes etc.

But this is not really what we want for simulation experiments, because we do want, on occasion, to rerun simulations exactly as they ran before, and that‘s not possible with a truly random generator. So what we use instead are ‘pseudo-random’ generators, which are algorithms that provide streams of numbers with certain statistical properties. We can make them repeat their output streams by initializing them with a given starting ‘seed’ (an integer value).

Such a generator provides output from a set of possible integers, e.g. all integers between 0 and 2^{32}-1. It delivers a stream of output which visits all possible integers before it starts repeating itself. The length of the cycle depends, among other things, on the number of bits used to hold the internal state of the generator (its ‘memory’). The larger the state, the longer the cycle possible. If the cycle is longer than the desired set of output integers, the generator may run through the set of such integers several different ways before exhausting its ‘meta-cycle’.

(It should be obvious that the output of such a generator is not truly random, since you can perfectly predict the next output given the state and the algorithm employed. A computer executing a pseudorandom algorithm is, after all, a perfectly deterministic little finite state machine!)

Other things being equal, we’d like a generator that has a single cycle that is as long as possible, that runs fast and uses little memory. (These wishes are of course in conflict with each other!) And the generator should perform ‘acceptably’ in a statistical sense. Readers who wish to pursue what this might mean are referred to the Bibliography *(see page 443)*.

This library implements a large number of different algoritms, some old ones included mostly for historical interest, and newer better ones drawn from recent literature. These generators have been subjected to various statistical tests, and the results of these tests are described in Advanced Usage Guide *(see page 426)*.

B.1.1. Using the Swarm default random generator

Every Swarm simulation has a predefined random generator of type MT19937 allocated (in random/random.m); its name is ‘randomGenerator’. This generator can be accessed from anywhere (any object or agent) in your simulation; its name is a global variable.

You can draw random numbers from this generator to your heart’s content and not worry about any statistical problems. The generator has a period close to 2^{19937} (1×10^{6001}), so there is no danger of running a simulation long enough for the generator to repeat itself. [At one microsecond per call, it would take about 3.2×10^{5987} years to exhaust this generator. For comparison, the age of the Universe is ‘only’ 2×10^{10} years!]

You obtain ‘random’ numbers of type unsigned integer this way:

```
unsigned int myUnsigned;
myUnsigned = [randomGenerator getUnsignedSample];
```

The values returned will be uniformly distributed in the range [0,4294967295] = $[0,2^{32}\text{-}1]$.

Or, if you need floating-point values instead, you can say

```
double myDouble;
myDouble = [randomGenerator getDoubleSample];
```

The returned values will be uniformly distributed in the range [0.0,1.0), i.e. they may be equal to 0.0 but never 1.0.

B.1.2. A note on starting seeds

Whenever a random generator is created, its state has to be initialized. What the state is initialized to determines where in its cycle the generator will start. To make life easy for the user, the Swarm generators can be initialized to a predictable and repeatable state using a starting ‘seed’, an integer between 1 and 2^{32}-1 (0 is not allowed.) Every time you initialize a given generator with a particular seed, you should get the same sequence of numbers from it.

You initialize a generator with a specific seed this way:

```
unsigned int randomSeed;
```

```
randomSeed = 4532657;
[randomGenerator setStateFromSeed: randomSeed];
```

You may do this any time during a simulation, not just at the start.

If you start your simulation in the normal way (with **`./mysim`** or **`./mysim -b`**), your generator will be started with the same default starting seed every time. This means it will produce the exact same sequence of numbers for each run, which makes replication easy.

However, if you want to run the same simulation several times but with different starting seeds, so that each run uses a different sequence of random numbers, this is easily accomplished by specifying `-s` or `-varyseed` on the command line.

B.1.3. Creating your own generators

Should you want to create your own random generators, e.g. to give each agent its own source of randomness, you do the following:

```
#import <random.h>
id <SimpleRandomGenerator> myGenerator;
unsigned int mySeed;
```

Then, to create a generator and start it with a specific seed value:

```
mySeed = 123776;
myGenerator = [RWC8gen create: [self getZone]
setStateFromSeed: mySeed];
```

Or, to create a generator and start it with the default system seed:

```
myGenerator = [PSWBgen createWithDefaults: [self getZone]];
```

The default system seed is normally a specific, static value, which is the same for each run, unless you run with the `-varyseed` command line argument, in which case it is a different seed value for each run.

In either case, you can (re-)set the seed at any time during a run:

```
mySeed = 345;
[myGenerator setStateFromSeed: mySeed];
```

(All the generators except two conform to the ‘SimpleRandomGenerator’ protocol. The two ‘split’ generators that do not, C2LCGX and C4LCGX, are described in the Generator Usage Guide.)

Finally, you can of course create your new generator in whatever memory zone you choose, not just the creating object’s own zone:

```
myGenerator = [SWB3gen createWithDefaults: globalZone];
```

(‘globalZone’ is the only predefined zone; other zones you would need to create yourself. See the Defobj Library *(see page 21)*to read more about zones.)

For a more detailed description of methods available from generator objects, see Usage Guide for Generators *(see page 416)*.

B.1.4. Using the Swarm default random distributions

A ‘distribution’ is an object which takes as its input a stream of (uniform) numbers from a random generator, and delivers as it output a stream of numbers that conforms to the desired statistical distribution. Most distribution objects tailor their output on the basis of parameters that you can set, for example Mean and Variance for the Normal distribution.

Each Swarm simulation comes with 3 pre-defined uniform distribution objects (defined in random/random.m):

```
id <UniformIntegerDist>  uniformIntRand;
id <UniformUnsignedDist> uniformUnsRand;
id <UniformDoubleDist>   uniformDblRand;
```

These distribution objects have all been set to draw their random numbers from the predefined random generator ‘randomGenerator’ (discussed above). These objects can be accessed from anywhere in your program.

You may draw (pseudo)random numbers uniformly distributed over a specified range this way:

```
int imin=-10, imax=10;
myInteger  = [uniformIntRand getIntegerWithMin:  imin withMax: imax];

unsigned int umin=600, umax=900;
myUnsigned = [uniformUnsRand getUnsignedWithMin: umin withMax: umax];

double dmin=0.375, dmax=0.665;
myDouble   = [uniformDblRand getDoubleWithMin:   dmin withMax: dmax];
```

Should the ‘min’ value that you specify be greater than the ‘max’ value, the distribution will switch them for you. If they are equal, then that value will be the result returned. (Note that if you only need uniform floating-point values between 0.0 and 1.0, you don’t need a distribution – any random generator will give you that. See above.)

B.1.5. Creating your own distribution objects

Each distribution object must be assigned a random generator to draw from when it is created. You may create a new generator for each distribution. Or, you may connect multiple distribution objects to one generator, so that they end up drawing output from the generator in an interleaved fashion. (This is what has been done with the predefined distributions.)

If you create distribution objects using the +createWithDefaults method Usage Guide for Distributions *(see page 423)*, each distribution object is assigned its own, newly created, private random generator. Each distribution class uses a different class of default random generator, just to keep things as statistically independent as possible.

HOWEVER: note that if you do not use the `-varyseed` command line switch, two different distribution objects of the same class, created with the +createWithDefaults method, will end up with generators of the same class that use the SAME starting seed, and so their output will be the exact same sequences. Their output will then be perfectly correlated, rather than statistically independent which is what we normally want. Beware!

To create your own distribution object, for example a Normal distribution, you would do this:

```
#import <random.h>
id <NormalDist> myNormalDist;
```

To create this distribution object and connect it to the predefined MT19937 generator:

```
myNormalDist = [NormalDist create: [self getZone]
setRandomGenerator: randomGenerator];
```

Or, if you want it to have its own private generator:

```
myNormalDist = [NormalDist createWithDefaults: [self getZone]];
```

Note that in this case, if you want to set the generator's seed you can do it as follows:

```
[[myNormalDist getGenerator] setStateFromSeed: 9874321];
```

Each distribution has its own set of key parameters. You may deal with these parameters in three different ways:

1. you assign a set of default parameter values to the object on creation, and draw from the distribution using those parameters. For example:

   ```
   #import <random.h>
   id <NormalDist> myNormalDist;
   double sample;

   myNormalDist = [NormalDist create: [self getZone]
   setGenerator: randomGenerator];
   [myNormalDist setMean: 0.0 setVariance: 2.1];
   ```

```
sample = [myNormalDist getDoubleSample];
```

2. you may refrain from assigning default parameters, in which case you must specify the (possibly different) desired parameters on each call.

```
#import <random.h>
id <NormalDist> myNormalDist;
double sample;

myNormalDist = [NormalDist create: [self getZone]
setGenerator: randomGenerator];
sample = [myNormalDist getSampleWithMean: 0.0 withVariance: 1.3];
```

3. You can (re-)set the default parameters any time, and you may call for a variate with specified parameter values even if different default parameters have been set. But note well: doing so does *not* reset the default parameters. Thus if you set the default parameters:

```
[myNormalDist setMean: 0.0 setVariance: 2.1];
sample = [myNormalDist getDoubleSample]; // from N[0.0,2.1]
```

then call for a sample with specific (different) parameters:

```
sample = [myNormalDist getSampleWithMean: 1.0 withVariance: 3.6];
```

and then call again for a sample without specifying parameters:

```
sample = [myNormalDist getDoubleSample];
```

on that last call your will again get a sample from the object's default settings, in this case a sample from N[0.0,2.1].

For a more detailed description of the methods available from distribution objects, see the Usage Guide for Distributions *(see page 423)*.

B.1.6. Some caveats

1. DO NOT use generators with bad statistical properties. See Advanced Usage Guide *(see page 426)* for a discussion of the generators implemented in this library.
2. DO NOT use generators whose maximum cycle length is too short for the intended application; you don't want your generators to start repeating themselves. Be especially aware of this if you use the PMMLCGgen class of generator; these have good properties but a fairly short cycle. See Advanced Usage Guide *(see page 426)* to read more about how to select a generator.
3. AVOID having generators in your simulation run in 'lock-step', producing output that is statistically correlated. This may happen if you have several generators of the same class, all started with the same default seed.
4. Be aware that even the best generators can have unexpected correlations with particular implementations of some models. As a result, in some cases using a "better" random number

generator can result in worse (less correct) model behavior than one could obtain when using a "bad" generator. If you suspect your model may have this kind of problem, you probably should re-run some experiments using a different underlying generator, to make sure the results are (statistically at least) the same. (For examples of this, see the references [Ferrenberg et al 1992] and [Nature 1994].)

B.2. Usage Guide

B.2.1. Usage Guide for Generators

A note on 'simple' vs. 'split' generators

Most of the generators supplied are of the 'simple' variety. Think of these as supplying a single, long stream of pseudorandom numbers. The description in the introduction of how to obtain generator output assumed a simple generator.

There are also two generators, C2LCGXgen and C4LCGXgen, which are of the 'split' variety. Think of them as consisting of a number of virtual generators, each supplying an independent stream of numbers which we can divide up into a number of segments of a given length. To obtain output from such a generator, we need to specify what virtual generator to draw from:

```
myUnsigned = [mySplitGenerator getUnsignedSample: virtualgen];
```

Read more about how to manage split generators below.

I shall first discuss 'simple' generators, and then discuss how 'split' generators differ from 'simple' ones. In the following text, wherever I use `PSWBgen` as an example you may substitute any other generator.

Note: any name that starts with `my` is meant to designate a variable of the appropriate type which you have defined in your own program.

Note: on defining variables that hold (pointers to) generators: it is now a convention in the rest of Swarm that if you want to specify what type of object a pointer should point to, you say:

```
id <protocolname> varname;
varname = [classname create: aZone];
```

instead of

```
classname *varname;
varname = [classname create: aZone];
```

Although it is usually the case that the `protocolname = the classname`, in some cases it is not. And publishing the protocols allows the programmers to keep unpublished what should remain internal private class methods.

The generators are different from other Swarm objects, in that they all perform the same function; they are drop-in replacements for each other. The 'split' generators (`C2LCGXgen`, `C4LCGXgen`) all conform to the same protocol, `<SplitRandomGenerator>`. The 'simple' (non-split) generators all conform to the same protocol, `<SimpleRandomGenerator>`.

Thus, when defining generators in your own program, you should say

```
id <SimpleRandomGenerator> varname;
varname = [classname create: aZone];
```

(Though see below for the different create methods available.)

For backward compatibility, protocols `<LCG1gen>`, `<TT800gen>` etc. are still defined, *but their use is deprecated and they may disappear later.*

B.2.1.1. Simple generators

You create a generator in one of 3 ways:

B.2.1.1.1. the lazy way

```
   id <SimpleRandomGenerator> myGenerator;
   myGenerator = [ PSWBgen createWithDefaults: [self getZone] ];
```

This allocates the object and initializes it with STARTSEED, which equals NEXTSEED if `-varyseed` was not specified, or RANDOMSEED if it was. (These macros are defined in the file 'randomdefs.h' in the source directory.)

B.2.1.1.2. using a single seed value

```
   id <SimpleRandomGenerator> myGenerator;
   myGenerator = [ PSWBgen create: [self getZone]
setStateFromSeed: mySeed ];
```

This allocates the object and initializes it with your seed value. If the object actually requires a vector of seed values to fill the state, this method generates the rest of the values needed using an inline PMMLCG generator.

You can find out later what seed value was used to initialize the generator:

```
   myUnsigned = [ myGenerator getInitialSeed ];
```

And you can find out what the largest valid seed value is by calling

```
   myUnsigned = [ myGenerator getMaxSeedValue ];
```

(In the current version of the library, the largest valid seed value is 2^{32}-1 for all the generators. The seed may not be 0.)

You may reset the generator's state at any time using this method:

```
[ myGenerator setStateFromSeed: mySeedValue ];
```

This will also reset to 0 the `currentCount` variable.

Alternatively, you may use the new -reset method [myGenerator reset], which resets the generator its state at startup, or its state at the point when -setStateFromSeed(s) was last used. Counters are zeroed.

B.2.1.1.3. using a vector of seed values

Assume we have defined a fixed array at compile time:

```
unsigned  int mySeedVector [vectorLength];
```

Then we can do this:

```
   id <SimpleRandomGenerator> myGenerator;
   myGenerator = [ PSWBgen create: [self getZone]
setStateFromSeeds: mySeedVector ];
```

You can find out how many seed values are required by asking

```
myUnsigned = [ myGenerator lengthOfSeedVector ];
```

(Obviously, you must first successfully have created the object to do this, for example using createWithDefaults! Or, see data in Random Library: Generator Data *(see page 432)*)

And we allocate the seed vector dynamically this way:

```
unsigned int *mySeedVector;
mySeedVector = [[self getZone] alloc: [ myGenerator lengthOfSeedVector]];
```

You can find out what vector of seed values was used to initialize the object:

```
unsigned int *myVector;
myVector = [ myGenerator getInitialSeeds ];
```

And you can find out the largest seed values that are allowed for the particular generator:

```
unsigned int *myVectorToo;
myVectorToo = [ myGenerator getMaxSeedValues ];
```

(These values vary from generator to generator, and they may not be the same for all elements of the vector for a given generator. Valid seeds never take the value 0.)

NOTE: in the above two calls, the variable myVector is set to point to an array internal to the generator. If you want to preserve the array's values outside the generator, you need to allocate space in your program either statically or dynamically, and use a for-loop to copy data from myVector[i] to myAllocatedVector[i].

You may reset the generator's state at any time by using the method

```
[ myGenerator setStateFromSeeds: (unsigned *) mySeedVector ];
```

This will also reset to 0 the currentCount variable.

NOTE: if you set a generator's state from a vector of seeds, the call:

```
myUnsignedValue = [ myGenerator getInitialSeed ];
```

will return a value of 0 (an invalid seed). On the other hand, if you initialize the generator with a single seed value, the call

```
mySeedVector = [ myGenerator getInitialSeeds ];
```

will return the seed vector that would produce identical output to what you obtained using the single seed.

B.2.1.1.4. antithetic values

You can make the generator serve up antithetic values by setting:

```
[ myGenerator setAntithetic: YES ];
```

If thus set, this makes -getUnsignedSample return (unsignedMax-x) instead of x, and the floating point methods return (1.0 - y) instead of y. The default for this parameter is that it is not set.

You can ascertain if this flag is set by calling

```
myBooleanValue = [ myGenerator getAntithetic ];
```

B.2.1.1.5. generator output

You obtain successive pseudorandom numbers from a generator by calling:

```
myUnsignedValue = [ myGenerator getUnsignedSample ];
```

The largest value that may be returned can be found by asking

```
myUnsignedValue = [ myGenerator getUnsignedMax ];
```

(The smallest value returned is always 0.)

If you would rather have floating point output in the range [0.0,1.0), you call one of these:

```
// Using 1 unsigned value to fill the mantissa:
myFloatValue  = [ myGenerator getFloatSample ];
myDoubleValue = [ myGenerator getThinDoubleSample ];

// Using 2 unsigned values to fill the mantissa:
myDoubleValue     = [ myGenerator getDoubleSample ];
myLongDoubleValue = [ myGenerator getLongDoubleSample ];
```

NOTE that the last method is not portable, because the size of a long double varies and hence the precision varies between architectures.

Finally, you can obtain a count of how many variates have been generated:

```
   myLongLongInt = [ myGenerator getCurrentCount ];
```

(currentCount is an unsigned long long int, which counts up to 2^{64}.)

B.2.1.2. Split generators

A split generator is a generator for which we are able to split the output stream into arbitrary non-overlapping segments, which we can access directly and easily. Such segments are statistically independent streams of (pseudo)random numbers.

We configure a split generator as consisting of a number (A) of "virtual generators", each of which has access to a number (2^v) of segments of length 2^w. The parameters A,v,w are specified when the generator is created. For example, for the C4LCGXgen generator, the default creation values are A=128, v=31, w=41. The only limitation is that A*(2^v)*(2^w) must not exceed the generator's cycle length, which is 2^{60} for C2LCGXgen and 2^{120} for C4LCGXgen.

We specify the configuration (A,v,w) at create time this way:

```
   id <SplitRandomGenerator> myGenerator;
   myGenerator = [ C4LCGXgen create: [self getZone]
setA: 64 setv: 20 setw: 76
setStateFromSeed: mySeedValue ];

   id <SplitRandomGenerator> myGenerator;
   myGenerator = [ C4LCGXgen create: [self getZone]
setA: 32 setv: 25 setw: 60
setStateFromSeeds: (unsigned *) mySeedVector ];
```

(In both cases, the only limitation is that A * 2^v * 2^w must be less than the generator's period, 2^{60} for C2LCGX and 2^{120} for C4LCGX.)

For obtaining output, we need to specify which of the A 'virtual' generators we want to draw from:

```
   myUnsignedValue   = [ myGenerator getUnsignedSample: 12 ];
   myFloatValue      = [ myGenerator getFloatSample: myVirtualGenerator ];
   myDoubleValue     = [ myGenerator getThinDoubleSample: someUnsigned-
Value ];
   myDoubleValue     = [ myGenerator getDoubleSample: 32 ];
   myLongDoubleValue = [ myGenerator getLongDoubleSample: 0 ];
```

Virtual generators are numbered from 0 to (A-1).

Obtaining the current count of variates generated likewise:

```
   myLongLongInt = [ myGenerator getCurrentCount:   myVirtualGenerator ];
   myLongLongInt = [ myGenerator getCurrentSegment: myVirtualGenerator ];
```

The latter call indicates what segment number the specified virtual generator is currently drawing numbers from.

Other than these methods, the methods discussed above under ’simple’ generators are the same for ’split’ generators.

In *addition* to this, ’split’ generators have the following methods to manage the virtual generators:

```
// Place all virtual generators at the start of the first segment:
[ myGenerator initAll ]; // done automatically at creation

// Place all virtual generators back to the start of the current segment:
[ myGenerator restartAll ];

// Place all virtual generators at the start of the next segment:
[ myGenerator advanceAll ];

// Place all virtual generators at the start of the indicated segment:
[ myGenerator jumpAllToSegment: myLongLongIntValue ];
```

You may also address individual virtual generators:

```
[ myGenerator initGenerator:    myVgen ];
[ myGenerator restartGenerator: myVgen ];
[ myGenerator advanceGenerator: myVgen ];
[ myGenerator jumpGenerator:    myVgen    toSegment: myLongLongIntValue ];
```

InternalState methods common to simple and split generators:

```
// Print (most of) the object’s state data to a stream:
[ myNormalDist describe: myStream ];
```

The stream myStream may be created thus:

```
id myStream = [ OutStream create: [self getZone] setFileStream: stdout ]; or
id myStream = [ OutStream create: [self getZone] setFileStream: stderr ];

   // Get the (class) name of the object:
   myString = [ myNormalDist getName ];

   // Get the object’s ’magic number’, used by putStateInto / setStateFrom:
   myUnsigned = [ myNormalDist getMagic ];
```

B.2.1.3. Saving and Resetting State

You may save, and later restore, the internal state of a generator using these methods:

```
   // Get the size of the memory buffer needed by putStateInto / setState-
From:
   myUnsigned = [ myGenerator getStateSize ];
```

```
// Extract the generator's state data into your memory buffer:
[ myGenerator putStateInto: myBuffer ];

// Set the generator's state from data in a memory buffer:
[ myGenerator setStateFrom: myBuffer ];
```

To illustrate, assume the following data definitions:

```
FILE * myFile;
const char * myFileName = "MyGenFile.bin"; // or whatever
int stateSizeG;
id stateBufG;
int status;
```

The following code shows how to save an object's state to disk: (You should add your own code to deal with disk file errors, either aborting or printing out error messages.)

```
// Ask how big a buffer we need:
stateSizeG = [ myGenerator getStateSize ];

// Allocate memory for the buffer:
stateBufG  = [[self getZone] alloc: stateSizeG];

// Ask the generator to put state data into the buffer:
[ myGenerator putStateInto: (void *) stateBufG ];

// Open a disk file for output:
myFile = fopen(myFileName, "w");
if (myFile == NULL) { }; // error on open: disk full, or no permissions

// Write the state buffer to disk in binary form:
status = fwrite(stateBufG, stateSizeG, 1, myFile);
if (status < 1) { }; // error on write: disk full?

// Close the file
status = fclose(myFile);
if (status) { }; // error on close ?

// Free the memory allocated to the buffer:
[[self getZone] free: stateBufG];

// Or, for test purposes, just zero the buffer data instead:
// memset(stateBufG, 0, stateSizeG);
```

This code shows how to set an object's state from a disk file:

```
// Ask how big a buffer we need:
```

```
stateSizeG = [ myGenerator getStateSize ];

// Allocate memory for the buffer:
stateBufG  = [[self getZone] alloc: stateSizeG];

// Open a disk file for input:
myFile = fopen(myFileName, "r");
if (myFile == NULL) { }; // error on open: file not found

// Read state data into the memory buffer:
status = fread(stateBufG, stateSizeG, 1, myFile);
if (status < 1) { }; // error on read

// Close the file:
status = fclose(myFile);
if (status) { }; // error on close

// Ask the generator set its state from the buffer data:
[ myGenerator setStateFrom: (void *) stateBufG ];

// Free the memory allocated to the buffer:
[[self getZone] free: stateBufG];
```

B.2.2. Usage Guide for Distributions

Where I use `NormalDist` in examples below, substitute any other distribution and its parameters as needed.

NOTE: any name that starts with `my` is meant to designate a variable of the appropriate type which you have defined in your own program.

B.2.2.1. Creating distributions

You create a distribution in one of several ways:

B.2.2.1.1. the lazy way:

```
id <NormalDist> myNormalDist;
myNormalDist = [ NormalDist createWithDefaults: [self getZone]];
```

This method will create a distribution object with no default statistical parameters set, as well as a fresh generator object connected to it. The generator object is initialized with STARTSEED (see the discussion above). Different distribution classes use different generators for this purpose.

B.2.2.1.2. Without default parameters, using a simple generator

```
   id <NormalDist> myNormalDist;
   myNormalDist = [ NormalDist create: [self getZone]
setGenerator: mySimpleGenerator ];
```

`myGenerator` must of course first have been set to point to a random generator of the 'simple' type. Note that you cannot assign a different generator to a distribution after it has been created.

You can create the generator at the same time as the distribution:

```
   id <NormalDist> myNormalDist;
   myNormalDist = [ NormalDist create: [self getZone]
setGenerator: [TT800gen create: [self getZone]
setStateFromSeed: 34453]         ];
```

B.2.2.1.3. Without default parameters, using a split generator

```
   id <NormalDist> myNormalDist;
   myNormalDist = [ NormalDist create: [self getZone]
setGenerator: mySplitGenerator
setVirtualGenerator: 7 ];
```

or perhaps

```
   id <NormalDist> myNormalDist;
   myNormalDist = [ NormalDist create: [self getZone]
setGenerator: [C4LCGXgen createWithDefaults: [self getZone]]
setVirtualGenerator: 99 ];
```

A split generator can be thought of as comprising a set of virtual generators (streams of random numbers), and a distribution object must be 'connected' to one of these streams. You cannot re-assign the generator or the virtual generator after a distribution object has been created.

In all these cases, when we want to obtain a random variate from this distribution object we need to specify the statistical parameters:

```
   myDouble = [ myNormalDist getSampleWithMean: 3.3 withVariance: 1.7];
```

You can use different parameters for every call. (And you can use this call even if default parameters have been set.)

B.2.2.1.4. With default parameters, using a simple generator

```
   id <NormalDist> myNormalDist;
   myNormalDist = [ NormalDist create: [self getZone]
setGenerator: mySimpleGenerator
setMean: 7.6 setVariance: 1.2 ];
```

B.2.2.1.5. With default parameters, using a split generator

```
   id <NormalDist> myNormalDist;
   myNormalDist = [ NormalDist create: [self getZone]
setGenerator: mySplitGenerator
setVirtualGenerator: 33
setMean: 3.2 setVariance: 2.1 ];
```

In these cases, we do not need to specify parameters to get a random number:

```
   myDouble = [ myNormalDist getDoubleSample ];
```

However, you *are* allowed to specify parameters even if default parameters have been set.

(Of course, different distributions have different parameters: RandomBitDist has none, the Uniform objects have minimum and maximum limit values, NormalDist and LogNormalDist use Mean and Variance, ExponentialDist only Mean, and GammaDist used alpha and beta. See the individual distribution protocols or the file `random/distributions.h` for the specific methods available.)

B.2.2.1.6. You may reset the default parameters this way, as often as you like

```
   [ myNormalDist setMean: 3.3 setVariance: 2.2 ];
```

B.2.2.1.7. You can obtain the current values of parameters

```
   // Default parameters:
   myDouble1 = [ myNormalDist getMean ];
   myDouble2 = [ myNormalDist getVariance ];
   myDouble3 = [ myNormalDist getStdDev ];

   // Get a pointer to the generator object:
   myOtherGenerator = [ myNormalDist getGenerator ];

   // Get the number of the virtual generator (if a split generator is used):
   myUnsignedValue  = [ myNormalDist getVirtualGenerator];

   // Find out if default parameters have been set:
   myBoolean        = [ myNormalDist getOptionsInitialized ];

   // Find out how many variates the object has delivered so far:
   // (The counter is an unsigned long long int, which goes up to 2^64.)
   myLongLongInt    = [ myNormalDist getCurrentCount ];
```

B.2.2.1.8. You can reset the variate counter and other state variables this way

```
   [ myNormalDist reset ];
```

This is most likely done in conjunction with resetting the connected generator, using

```
[ myGenerator setStateFromSeed: mySeedValue ]
```

or simply

```
[ myGenerator reset ];
```

B.2.2.1.9. Finally, we have the InternalState protocol methods

```
// Print (most of) the object's state data to a stream:
[ myNormalDist describe: myStream ];
```

The stream myStream may be created thus:

```
id myStream = [ OutStream create: [self getZone] setFileStream: stdout ]; or
id myStream = [ OutStream create: [self getZone] setFileStream: stderr ];
```

```
// Get the (class) name of the object:
myString = [ myNormalDist getName ];

// Get the object's 'magic number', used by putStateInto / setStateFrom:
myUnsigned = [ myNormalDist getMagic ];
```

B.2.2.2. Saving And Restoring State

You may save, and later restore, the internal state of a distribution object using InternalState methods.

- See the Generator Usage Guide, which describes how to do this. The code for saving/restoring distributions would be similar.
- Note that saving the state of a distribution object will NOT automatically save the state of the attached generator; you are responsible for doing so. (Since it is possible, even encouraged, to use a single generator to feed several distribution objects, this is the only sane way of doing it.)

B.3. Advanced Usage Guide

This section amplifies the Usage Guide with material on how to choose a random generator for your simulation, what default generators have been implemented for the distribution objects, and information on the set of test programs used.

B.3.1. Choosing a Generator

B.3.1.1. Choosing A Generator

In this version of Random for Swarm there are a total of 36 different generators implemented. If you are a serious simulationist you need to select which one(s) to use for your model.

Here are some factors to consider:

a. We want a generator to have as good statistical properties as possible. We measure this by subjecting the generators to various tests. I have subjected the implemented generators to two sets of tests, Diehard and ENT. (Look in directory /testR6 of the test package.) (The Random Library: Generator Statistical Tests *(see page 428)* summarizes the test results. The highlights are:

   i. The LCG and SCG generators are of *very* poor quality (they fail many of the tests), and should never be used. [They are likely to disappear in the next release.]

   ii. The lagged-Fibonacci generators (ACG, SWBi, PSWB) all fail the Diehard 'Birthday Spacings Test', and are therefore only conditionally recommended for use.

   iii. The other 32-bit generators pass all tests, and I therefore have no reason not to recommend them all for use.

   iv. The 31-bit generators all fail certain tests because one bit has a constant value. Beyond that they all seem to be ok.

b. We want a generator to have a long enough period for our purpose, and in general the longer the period the better. (However, note that to generate 2^{64} random numbers from a fast generator like MT19937 would take 2.1 million years on a 486/66, so in practice any generator with a period close to 2^{60} or larger will be acceptable.) The PMMLCG generators, although they are of acceptable quality otherwise, have a period less than 2^{31} which we can exhaust in under 3 hours. So use those only for quick 'toy' applications.

c. We want a generator to execute efficiently, the faster the better. The Random Library: Generator Data *(see page 432)* indicates the execution times for the generators as implemented.

d. We want a generator to take as little resources (memory) as possible. The Random Library: Generator Data *(see page 432)* indicates the size of each generator's state in bits.

e. Other things being equal, we want our generator to have a single full period rather than a number of shorter periods, since we may not know whether a particular starting seed will put us into a long or a short subcycle. This disqualifies SWB, and possibly the new MWC and RWC generators (Marsaglia does not say how many periods these generators have, but only gives the period length.)

B.3.1.2. Strategy For Using Random Generators

There are 3 possible strategies for using random generators:

a. Use one single generator for your whole simulation, and have all 'users' of randomness (distributions and other objects) call this single generator in an interleaved fashion. For this purpose, generators such as MT19937, TT800, TT775, TT443 (and possibly PSWB and SWB) seem particularly well suited since they have immense periods. The code in `random/random.m` shows how to connect several distribution objects to a single random generator. The generator `randomGenerator` supplied there is of class MT19937.

b. Use a single generator of long period, but divide this long period up into a number of non-overlapping (hence statistically independent) segments, and let each ‘user’ of randomness draw their random numbers from separate segments. Doing this requires that we be able to quickly access these separate segments. The ‘split’ generators C2LCGX and C4LCGX implement such a scheme. You can specify at creation how you want the period of these generators subdivided. (See the source code for details.)

c. Use a separate random generator for each ‘user’ of randomness. In this case, you need to make sure that two or more generators of the same type are not started with the same seed, since in this case their output will be highly correlated. Provide your own seeds, or use RANDOMSEED or NEXTSEED.

The distribution objects, if created with the ‘createWithDefault: aZone’ method, will create for themselves a fresh generator, with each class of distribution using a different class of generator. All the generators in this case are initialized with NEXTSEED (or RANDOMSEED, if you start the program with `-varyseed`).

B.3.1.3. Generator Quality

These tables shows the results of testing the generator objects statistically:

Table B-1. Random Library: Generator Statistical Tests

Generator	timing uS (unsigned)	bits state	period (s) # length	tests failed (x)
				`abcdefghijklmnopqrstuvwx`
	30-bit output			
SCG	3.328	1650	1 2^{55}	`x.xx2xxxxxxx..x..xx.xx..`
	32-bit output			
LCG1	2.564	32	1 2^{32}	`x...xxxxxxx.........x...`
LCG2	2.564	32	1 2^{32}	`x...xxxxxxx.........x...`
LCG3	2.564	32	1 2^{32}	`x...xxxxxxx.........x...`
ACG	2.702	1760	1 2^{55}	`x.......................`
SWB1	3.285	1185	64 2^{1178}	`x.......................`

SWB2	3.285	769	1536 2^{757}	x.......................
SWB3	3.285	673	192 2^{664}	x.......................
PSWB	3.452	1377	1 2^{1376}	x.......................
MWCA	3.399	64	?? 2^{59}	....
MWCB	3.420	64	?? 2^{59}	....
MT19937	3.698	19968	1 2^{19937}	....
TT800	4.654	800	1 2^{800}	....
C3MWC	6.387	192	?? 2^{118}	....
RWC2	7.445	96	?? 2^{92}	....
RWC8	14.649	288	?? 2^{250}	....
	31-bit output			
C2TAUS1	4.078	62	1 2^{60}	1..-1x111x1.........x...
C2TAUS2	4.078	62	1 2^{60}	1..-1x111x1.........x...
C2TAUS3	4.078	62	1 2^{60}	1..-1x111x1.........x...
TT775	4.654	775	1 2^{775}	1..-1x111x1.........x...
TT403	4.670	403	1 2^{403}	1..-1x111x1.........x...
PMMLCG1	4.715	31	1 2^{31}	1..-1x111x1.........x...
PMMLCG2	4.715	31	1 2^{31}	1..-1x111x1.........x...
PMMLCG3	4.715	31	1 2^{31}	1..-1x111x1.........x...
PMMLCG4	4.715	31	1 2^{31}	1..-1x111x1.........x...
PMMLCG5	4.715	31	1 2^{31}	1..-1x111x1.........x...
PMMLCG6	4.715	31	1 2^{31}	1..-1x111x1.........x...
PMMLCG7	4.715	31	1 2^{31}	1..-1x111x1.........x...
PMMLCG8	4.715	31	1 2^{31}	1..-1x111x1.........x...
PMMLCG9	4.715	31	1 2^{31}	1..-1x111x1.........x...
C2LCGX	7.029	62	1 2^{60} (Split)	1..-1x111x1.........x...

MRG5	9.674	155	1 2^{155}	`1..-1x111x1.........x...`
MRG6	10.449	186	1 2^{186}	`1..-1x111x1.........x...`
MRG7	10.913	217	1 2^{217}	`1..-1x111x1.........x...`
C4LCGX	11.688	124	1 2^{210} (Split)	`1..-1x111x1.........x...`
C2MRG3	13.459	186	1 2^{185}	`1..-1x111x1.........x...`

Test code	**Test suite**	**Explanation of test codes**
a	Diehard	Birthday Spacings Test
b	Diehard	Overlapping 5-permutation Test
c	Diehard	Binary Rank Test (31x31)
d	Diehard	Binary Rank Test (32x32)
e	Diehard	Binary Rank Test (6x8)
f	Diehard	Bitstream Test (overlapping 20-tuples)
g	Diehard	OPSO (Overlapping Pairs, Sparse Occupancy)
h	Diehard	OQSO (Overlapping Quadruples, Sparse Occupancy)
i	Diehard	DNA Test
j	Diehard	Count-the-1's Test (integers)
k	Diehard	Count-the-1's Test (specific bytes)
l	Diehard	Parking Lot Test
m	Diehard	Minimum Distance Test
n	Diehard	3D Spheres Test
o	Diehard	Squeeze Test
p	Diehard	Overlapping Sums Test
q	Diehard	Runs Test

r	Diehard	Craps Test
s	ENT	Entropy Test
t	ENT	Compression test
u	ENT	Chi-Square Test
v	ENT	Arithmetic Mean
w	ENT	Monte Carlo value for Pi
x	ENT	Serial Correlation Coefficient

Indication	Explanation of indications
.	The generator passed this test
1	The generator passed this test, except for the lowest bit
2	The generator passed this test, except for the 2 lowest bits
x	The generator failed this test completely
-	This test cannot be passed by this generator (too few bits)

Notes.

- For 31-bit generators, it is normal to fail 1 part of these tests:a,e,g,h,i,k. This is because we left-shift the output of these generators by 1 bit, so that the lowest bit is always 0.
- For 31-bit generators, it is normal to fail test d (32x32 rank test).
- All 31-bit generators also fail tests f, j, and u (and most others don't.) This is *likely* due to the constant low bit.

Choosing A Generator.

- Clearly unacceptable generators: LCG1, LCG2, LCG3, SCG.
- Conditionally recommended generators: ACG, SWB1, SWB2, SWB3, PSWB.
- Use with caution: the PMMLCGx generators (due to their very short period 2^{31}).
- All other generators are recommended at this time.

B.3.1.4. More generator data

And this table gives further data about the generators:

Table B-2. Random Library: Generator Data

Generator	Seeds	Modulus	Cycles (length)	Bits	Speed	State
	(a) Simple Short Generators					
LCG1	1	2^{32}	1 m	32	1.442	32
LCG2	1	2^{32}	1 m	32	1.442	32
LCG3	1	2^{32}	1 m	32	1.442	32
PMMLCG1	1	2^{31}-1	1 m-1	31	0.784	31
PMMLCG2	1	2^{31}-1	1 m-1	31	0.784	31
PMMLCG3	1	2^{31}-1	1 m-1	31	0.784	31
PMMLCG4	1	2^{31}-1	1 m-1	31	0.784	31
PMMLCG5	1	2^{31}-105	1 m-1	31	0.784	31
PMMLCG6	1	2^{31}-225	1 m-1	31	0.784	31
PMMLCG7	1	2^{31}-325	1 m-1	31	0.784	31
PMMLCG8	1	2^{31}-85	1 m-1	31	0.784	31
PMMLCG9	1	2^{31}-249	1 m-1	31	0.784	31
	(b) Simple Long Generators					
ACG	55	2^{32}	1 2^{55}	32	1.369	760
SCG	55	10^{9}	1 2^{55}	30	1.111	650
SWB1	37+c	2^{32}	64 2^{1178}	32	1.126	185
SWB2	24+c	2^{32}	1536 2^{757}	32	1.126	769
SWB3	21+c	2^{32}	192 2^{664}	32	1.126	673
PSWB	43+c	2^{32}-5	1 2^{1376}	32	1.070	1377
TT403	13	2^{31}	1 2^{403}	31	0.792	403

TT775	25	2^{31}	1 2^{775}	31	0.795	775
TT800	25	2^{32}	1 2^{800}	32	0.795	800
MT19937	624	2^{32}	1 2^{19937}	32	1.000	19937
MRG5	5	2^{31}-1	1 2^{155}	31	0.382	155
MRG6	6	2^{31}-1	1 2^{186}	31	0.354	186
MRG7	7	2^{31}-1	1 2^{217}	31	0.339	217
MWCA	2	2^{32}	? 2^{59}	32	1.088	64
MWCB	2	2^{32}	? 2^{59}	32	1.088	64
RWC2	3	2^{32}	? 2^{92}	32	0.497	96
RWC8	18s	2^{32}	? 2^{250}	32	0.252	288
	(c) Long Generators with Splitting Facilities: (none)					
	(d) Combined Generators					
C2TAUS1 (M)	2	2^{31}-1	1 2^{60}	31	0.907	62
C2TAUS2 (M)	2	2^{31}-1	1 2^{60}	31	0.907	62
C2TAUS3 (M)	2	2^{31}-1	1 2^{60}	31	0.907	62
C2MRG3 (M)	6	2^{31}-1	1 2^{185}	31	0.275	186
C3MWC (M)	6	2^{32}	? 2^{118}	32	0.570	192
	(e) Combined Generators with Splitting Facilities					
C2LCG (M,S)	2	2^{31}-85	1 2^{60}	31	0.526	62
C4LCG (M,S)	4	2^{31}-1	1 2^{120}	31	0.316	124

Generator	**Calculating cycle lengths**
LCGi	cycle = m = 2^{32}

PMMLCGi	cycle = m-1 < 2^{31}
ACG	cycle = 2^r
SCG	cycle = 2^r
SWBi	cycle = (2^{32r} - 2^{32s}) / #cycles
PSWB	cycle = m^r - m^s
TGFSRi	cycle = 2^{w*N} - 1
MRGi	cycle = m^i - 1
C2MRG3	cycle = ($m1^3$ - 1)*($m2^3$ - 1)/2
C2TAUSi	cycle = Mask1 * Mask2
C2LCG	cycle = (m1 * m2) / 2
C4LCG	cycle = (m1 * m2 * m3 * m4) / 8
MWCA, MWCB	cycle = ? (not specified by Marsaglia)
C3MWC	cycle = ? " "
RWC2	cycle = ? " "
RWC8	cycle = ? " "

B.3.2. Default Generators for the Distributions

B.3.2.1. Random Library: Default Generators

When distributions are created using the `createWithDefaults: aZone` method, they create their own generator and initialize it with NEXTSEED (or with RANDOMSEED, if you started the program with the `-varyseed` switch).

The generators used are as follows:

Table B-3. Random Library: Default Generators

Distribution		Generator
RandomBitDist	uses	C2TAUS1gen
BernoulliDist	uses	C2TAUS2gen

UniformIntegerDist	uses	TT403gen
UniformUnsignedDist	uses	TT775gen
UniformDoubleDist	uses	TT800gen
NormalDist	uses	MWCAgen
LogNormalDist	uses	MWCBgen
ExponentialDist	uses	C2TAUS3gen
GammaDist	uses	PSWBgen

These generators were chosen on the basis of quality and execution speed.

B.3.2.2. Utility Generator And Distributions

There are 4 default random objects defined in `random/random.m`. These are:

```
id <MT19937gen>           randomGenerator;
id <UniformIntegerDist>    uniformIntRand;
id <UniformUnsignedDist>  uniformUnsRand;
id <UniformDoubleDist>     uniformDblRand;
```

These objects may be called from anywhere in your program. Note (a): the generator is initialized with NEXTSEED or RANDOMSEED depending on the use of the `-varyseed` command line option. Note (b): the distribution objects are created *without* default statistical parameters.

B.3.3. Random Library Test Programs

1. In a separate tar file *(ftp://ftp.santafe.edu/pub/swarm/RandomTests-0.81.tgz)* , available at the SFI ftp site, there are a set of programs which exercise aspects of the generator objects' functionality. The following (very utilitarian) programs are available:

   - **testR0.** a program which exercises every generator and distribution, verifying correct operation and comparing output to that obtained on the author's system.
   - **testR1.** a program which prints out diagnostic output for code in random.m.
   - **testR2.** a program which asks each distribution and generator to describe itself using the Swarm xprint method.
   - **testR3.** a program which asks each distribution and generator to describe itself using the objects' 'describe' method.

- **testR4.** a program that performs timing tests on each generator and distribution, computing the time it takes to call each object 10,000,000 times.
- **testR6.** a program that generates a binary file containing 2.5M variates from a specified generator, for purposes of statistical testing (e.g. with ENT or Diehard.)
- **testR7.** a program which records, for each generator, the value of unsignedMax, the number of output bits, and the value of lengthOfSeedVector.
- **testR9.** a program which records, for each generator and distribution, the object's 'magic number' and the buffer size needed for getState/setState.

2. Statistical testing: the generators have been put through the tests in the `/testR6/ENT` and `/testR6/Diehard` directories. The raw data can be found there. The results are summarized in the test log files found in the same test distribution. The distribution objects have not been tested statistically yet.

The code here represents an effort to implement several efficient, reasonably safe generators. The algorithms come from reading the literature (Bibliography *(see page 443)*). These algorithms have been implemented as accurately as possible and run through some simple tests. Some generators, included here for historical reasons only, are known to have bad statistical properties, and their use is deprecated. See Advanced Usage Guide *(see page 426)*for information on the quality of the included generators.

While the objects in this library are believed to function correctly, the prudent and paranoid modeller would do well testing them in some domain-specific way. One easy way to do this is to run an experiment twice: once with one class of generator (say, PMMLCG), and once with another (say, SWB). If the results differ radically, then you can suspect one of the generators. If they don't, well, the generators still might be faulty!

The generators supplied with this release have been subjected to statistical testing using George Marsaglia's *Diehard* tests as well as John Walker's entropy tests (ENT). The results of these tests are summarized in Random Library: Generator Statistical Tests *(see page 428)*. Other data on the properties of the generators are found there as well. These data support a discussion of how to choose one or more generators for your simulation.

The ENT test is included in the tarball of test programs mentioned above. The Diehard tests are copyright and hence are not, but they can be downloaded from the web at: ftp://ftp.csis.hku.hk/pub/random *(ftp://ftp.csis.hku.hk/pub/random)* .

The distribution objects have not been statistically tested.

B.4. Implementation details

This section provides implementation details for the current version of the random library.

B.4.1. 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 190)* 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`**).

B.4.2. 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 Random Library: Generator Statistical Tests *(see page 428)*, 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").

B.4.3. 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.

B.4.4. 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>`

B.5. Resources for random number generation

This section lists some of the source reference material used in programming the random generators and distributions. The following is a list of source articles or books from which the generator and distribution objects were implemented

B.5.1. Generators

- **LCG.** a golden oldie; see [Knuth 1981] or [Numerical Recipes].
- **PMMLCG.** see [Park & Miller 1988], [L'Ecuyer & Cote 1991] and [L'Ecuyer & Andres 1997].
- **ACG.** a golden oldie; see [Knuth 1981] or [Numerical Recipes].
- **SCG.** a golden oldie; see [Knuth 1981] or [Numerical Recipes].
- **SWB.** see [Marsaglia & Zaman 1991]
- **PSWB.** see [Marsaglia & Zaman 1991]
- **MRG.** see [L'Ecuyer et al 1993]
- **C2MRG3.** see [L'Ecuyer 1996(a)] and [L'Ecuyer 1999(a)]
- **C2TAUS.** see [Tezuka & L'Ecuyer 1991], [L'Ecuyer 1996(b)] and [L'Ecuyer 1999(b)]. Also [Tausworthe 1965].
- **TGFSR (TT800, TT775, TT403).** see [Matsumoto & Kurita 1992] and [Matsumoto & Kurita 1994].
- **MT19937.** see [Matsumoto & Nishimura 1998].
- **MWCA, MWCB, C3MWC, RWC2, RWC8 ("Mother").** See [Marsaglia 1994(a)] and [Marsaglia 1994(b)].
- **C2LCGX.** See [L'Ecuyer & Cote 1991]
- **C4LCGX.** See [L'Ecuyer & Andres 1997]

B.5.2. Distributions

- **RandomBitDist.** Code contributed by Nelson Minar. *(mailto://nelson@media.mit.edu)*
- **BernoulliDist.** Code contributed by Barry McMullin *(mailto://mcmullin@santafe.edu)* .
- **UniformIntegerDist.** Code contributed by Nelson Minar *(mailto://nelson@media.mit.edu)* .
- **UniformUnsignedDist.** Code contributed by Nelson Minar *(mailto://nelson@media.mit.edu)* .
- **UniformDoubleDist.** Code contributed by Nelson Minar *(mailto://nelson@media.mit.edu)* .
- **NormalDist.** See [Numerical Recipes].
- **LogNormalDist.** See [Numerical Recipes].
- **ExponentialDist.** See [Russell 1992].

- **GammaDist.** See [Watkins 1994].

B.5.3. Useful Web Sites

- Pierre L'Ecuyer maintains a personal web site *(http://www.iro.umontreal.ca/~lecuyer)* . He has many of his own papers there, as well as further links. He has papers both on generating random numbers and on testing random number generators.
- George Marsaglia has a personal web site *(http://stat.fsu.edu/~geo)* . His battery of tests for random generators, called Diehard, is not available there, but rather from Hong Kong *(ftp://ftp.csis.hku.hk/pub/random)* . A GUI version of Diehard has been under development for some time, but is not yet ready.
- The pLab *(http://random.mat.sbg.ac.at)* project in Salzburg, Austria, also has much useful information.

Bibliography

[Ferrenberg et al 1992]ăAlan M. Ferrenberg, D. P. Landau, and Y. Joanna Wong, *"Monte Carlo Simulations: Hidden Errors from "Good" Random Number Generators": Physical Review Letters*, no. 23, vol. 69, December, 1992.

[Knuth 1981]ăDonald Knuth, *The Art of Computer Programming*, 2nd ed., vol. II, *Seminumerical Algorithms*, Addison-Wesley, Reading, 1981.

[L'Ecuyer et al 1993]ăPierre L'Ecuyer, F. Blouin, and R. Couture, *"A Search for Good Multiple Recursive Random Generators.": ACM TOMACS*, vol. 3, pp. 87-98, 1993.

[L'Ecuyer 1996(a)]ăPierre L'Ecuyer, *"Combined Multiple Recursive Generators.": Operations Research*, no. 5, vol. 44, pp. 816-822, 1997.

[L'Ecuyer 1996(b)]ăPierre L'Ecuyer, *"Maximally Equidistributed Combined Tausworthe Generators": Mathematics and Computation*, no. 65, vol. 213, pp. 203-213, 1996.

[L'Ecuyer 1999(a)]ăPierre L'Ecuyer, *"Good Parameter Sets for Combined Multiple Recursive Random Number Generators": Operations Research*, no. 1, vol. 47, pp. 159-164, 1999.

[L'Ecuyer 1999(b)]ăPierre L'Ecuyer, *"Tables of Maximally-Equidistributed Combined LFSR Generators": Mathematics and Computation*, no. 225, vol. 68, pp. 261-269, 1999.

[L'Ecuyer & Andres 1997]ăPierre L'Ecuyer and Terry H. Andres, *"A Random Number Generator Based on the Combination of Four LCGs.": Mathematics and Computers in Simulation*, vol. 44, pp. 99-107, 1997.

[L'Ecuyer & Cote 1991]ăPierre L'Ecuyer and Serge Cote, *"Implementing a Random Number Package with Splitting Facilities": ACM TOMACS*, no. 1, vol. 17, pp. 98-111, March, 1991.

[Marsaglia 1994(a)]ăGeorge Marsaglia, *"The Mother of All Random Generators": Posted by Bob Wheeler to sci.stat.consult (news://sci.stat.consult) and sci.math.num-analysis (news://sci.math.num-analysis) on behalf of George Marsaglia on October 28, 1994. The code is available at ftp.taygeta.com (ftp://ftp.taygeta.com/pub/c/mother.c)* .

[Marsaglia 1994(b)]ăGeorge Marsaglia, *"Multiply-With-Carry Generators": File "mwc1.ps" on Marsaglia's Diehard CD-ROM, available online at ftp.csis.hku.hk (ftp://ftp.csis.hku.hk/pub/random)* .

[Marsaglia & Zaman 1991]ăGeorge Marsaglia and Arif Zaman, *"A New Class of Random Number Generators.": Annals of Applied Probability*, no. 3, vol. 3, pp. 462-480.

[Matsumoto & Kurita 1992]ăM. Matsumoto and Y. Kurita, *"Twisted GFSR generators": ACM TOMACS*, vol. 2, 1992, pp. 179-194.

[Matsumoto & Kurita 1994]ăMakoto Matsumoto and Yoshiharu Kurita, *"Twisted GFSR Generators II": ACM TOMACS (Amended by K. Matsumoto, 8 July 1996).*, no. 3, vol. 4, pp. 254-266.

[Matsumoto & Nishimura 1998]ăM. Matsumoto and T. Nishimura, *"Mersenne Twister: A 623-dimensionally Equidistributed Uniform Pseudorandomnumber Generator": ACM TOMACS*, no. 1, vol. 8, pp. 3-30, 17 December 1998.

[Nature 1994]ă*"News and Views": Nature*, vol. 372, December, 1994.

[Numerical Recipes]ăW. H. Press, S. A Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C*, 2nd ed., Cambridge University Press, 1992.

[Park & Miller 1988]ăStephen K. Park and Keith W. Miller, *"Random Number Generators: Good Ones Are Hard to Find.": CACM*, no. 10, vol. 31, pp. 1192-1201, October 1988.

[Tausworthe 1965]ăR.C. Tausworthe, *"Random Numbers Generated by Linear Recurrence modulo 2": Math. Comput.*, vol. 19, pp. 201-209, 1965.

[Tezuka & L'Ecuyer 1991]ăShu Tezuka and Pierre L'Ecuyer, *"Efficient and Portable Combined Tausworthe Random Number Generators.": ACM TOMACS*, no. 2, vol. 1, pp. 99-112.

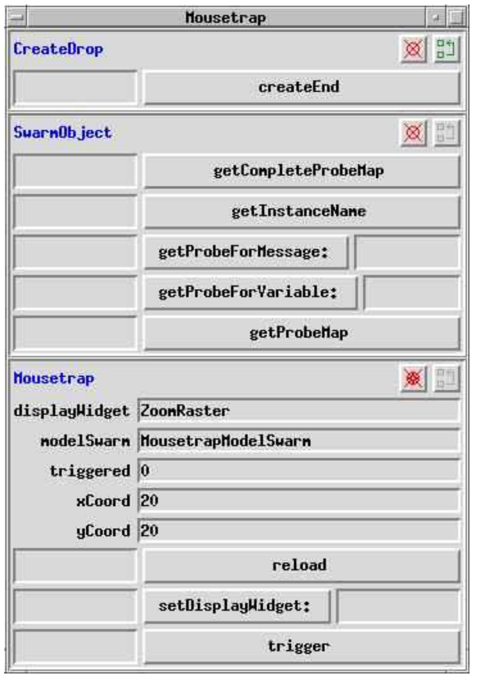
[Russell 1992]ăEdward C. Russell, *"Building Simulation Models With SIMSCRIPT II.5"*, CACI Products Company, La Jolla, 1992, *The company's web site is: http://www.caciasl.com (http://www.caciasl.com)* .

[Watkins 1994]ăKevin Watkins, *Discrete Event Simulation in C*, McGraw-Hill, 1994.

Appendix C. Probing and Displaying the Contents of a SwarmObject

C.1. Overview

Figure C-1. Complete probe map



C.2. The Framework

Probes, ProbeMaps and ProbeDisplays allow the user to dynamically interact with the objects in their simulation. By *interaction* we mean reading/setting instance variables as well as dynamically generating method calls. The main point being that these interactions are not hardwired into the program code, but occur due to user-generated requests, mainly through the provided GUI.

The key to this capability is the Probe: in general a probe takes an object and either extracts the value of a specific variable, or calls a specific method. For this purpose we provide two subclasses: `VarProbe` and `MessageProbe`.

There are two main uses for probes: they can be fed into data-collection objects and serve as interfaces to the objects about which data is being collected (thus keeping the data-collection objects as general as possible) - the Averager class, for example, directly subclasses MessageProbe. Or, they can be used in order to generate a GUI to the individual objects in the simulation (the more common usage).

In order to generate a graphical version of a given probe, the programmer must place it inside a `ProbeDisplay`, which will automagically generate a window with the appropriate interface. Since, more often than not, the programmer will want to generate windows with more than one variable, we have designed the ProbeDisplay to deal with `ProbeMaps` rather than individual probes. Thus, by generating a ProbeMap containing exactly the right probes, a user is able to customise the window generated by the ProbeDisplay.

In order to facilitate the creation of all these different objects, the Swarm kernel provides some alternative methods for Probe/Map/Display generation:

- **Direct Generation.** which leaves the user in charge of all the details.
- **Object Generation.** (`Probes/ProbeMaps`) - where we use method calls (defined in SwarmObject) on a given target object to generate the Probe/ProbeMap for that object. This allows the object to be aware that it is being probed and to control directly the contents of the Probe/ProbeMap being requested...
- **Library Generation.** (`Probes/ProbeMaps`) - where the programmer "checks out" a unique, shared copy of a Probe/ProbeMap from the `probeLibrary` object (of class `ProbeLibrary`) provided by the kernel. By shared we mean that a similar request made at a different point in the code, will return a reference to the very same probe instance. *Note that, by default, Object Generation is equivalent to Library Generation since the default behaviour of SwarmObjects, when requested to create a ProbeMap, is to return a reference to the probeLibrary's unique copy!!!*
- **ProbeDisplayManager Generation.** (`ProbeDisplays`) - where the programmer generates a ProbeDisplay directly, by requesting it from the `probeDisplayManager` object (of class `ProbeDisplay`) provided by the kernel (in graphics mode). The probeDisplayManager will create the ProbeDisplay based on a ProbeMap given to it by the probed object. Note: since the probeDisplayManager is only created in graphics mode, it is documented in the simtools area of the documentation. Here we emphasise simply that in order to generate a ProbeDisplay onto an object the programmer need only write the following line of code:

  ```
  [probeDisplayManager createProbeDisplayFor: anObject] ;
  ```

And that the contents of the returned ProbeDisplay will be based on a ProbeMap obtained from the probed object itself.

C.3. Probe support in objectbase

C.3.1. Overview

Probe support resident in the *Objectbase Library (see page 156)* library is documented with that library. Extra description, above and beyond the bare interface description for precision probing and SwarmObject is below:

C.3.2. Precision setting for floating-point numbers with probes

C.3.2.1. Global setting of precision

There are two types of global precision setting via ProbeLibrary *(see page 179)*:

- **-setDisplayPrecision: (int) nSigFigsSaved.** Sets the number of significant figures for floating point (and double-floating) numbers diplayed on a GUI widget. Currently this is only implemented for VarProbes. The display uses the %*g sprintf-style formatting, which can vary slightly from implementation to implementation. If you set the number of significant figures to 3, then a float of value of 0.6344346 is displayed as 0.634 on the GUI widget.Note that this in no way affects the underlying stored value of the floating point number.
- **-setSavedPrecision: (int) nSigFigsSaved.** Sets the global default for the saving of floats through ObjectSaver. All objects with floats and doubles as instance variables are saved with the precision specified by this method. This is *independent* of the displayed precision of the same instance variable on a GUI widget.

To actually initialise these defaults: in the top level swarm, you should add the calls to the global ProbeLibrary instance (which is actually created by the initSwarm call in main) during the - createBegin method (this sets the precision in the global instance, *before* any probes are checked out of the instance. If neither method is called on probeLibrary, then the precision defaults to six significant figures in both cases.

Example C-1 Global setting precision in `HeatbugObserverSwarm.m`

```
+ createBegin: aZone
{
  HeatbugObserverSwarm * obj;
  id <ProbeMap> probeMap;

  [...]

  probeMap = [EmptyProbeMap createBegin: aZone];
  [probeMap setProbedClass: [self class]];
  probeMap = [probeMap createEnd];
```

```
    // set the display defaults
  [probeLibrary setDisplayPrecision: 3];
    // typically saved precision would be higher than displayed precision
  // for statistical and data analysis purposes
  [probeLibrary setSavedPrecision: 10];

  // Add in a bunch of variables, one per simulation parameters
  [probeMap addProbe: [probeLibrary getProbeForVariable: "displayFrequency"
    inClass: [self class]]];
  [...]

  // Now install our custom probeMap into the probeLibrary.
  [probeLibrary setProbeMap: probeMap For: [self class]];
  return obj;
}
```

C.3.2.2. Setting precision for individual probes

The formatting for an individual probe can be set directly, using a sprintf-style formatting string. Typically, `customProbeMaps` are created in the `+ createBegin` factory method for a Swarm or a SwarmObject. To set the formatting for a floating point probe, the method from `VarProbe` *(see page 185)* is used:

- **-setFormatFloat: (const char *)floatFormat.** is applied to the instance of the VarProbe "checked-out" of the global probe library instance. The sprintf-formatting string can "over-ride" the "%g" format set by the global precision (as above) (Typically "%g" chooses between the "%f" and "%e", depending on the size of the exponent - which is implementation-dependent - this method allows you to explicitly set the type of display).

In the following example, it is desired that the number of significant figures for the floating point variable `randomMoveProbability` is three (3). Currently (Swarm 1.0.5) this is only works for `VarProbes` and not `MessageProbes`, as yet.

Example C-2 Setting precision for individual probes in `HeatbugModelSwarm.m`:

```
+ createBegin: aZone
{
  HeatbugModelSwarm * obj;
  id <ProbeMap> probeMap;
  id floatProbe;

  [...]
    // the -setFloatFormat is applied to the probe which is
  // "returned" from the call to probeLibrary
  floatProbe = [[probeLibrary getProbeForVariable: "randomMoveProbability"
    inClass: [self class]]
```

```
                                  setFloatFormat: ".3f"];

  // now we have the probe - put it back into the customMap
  [probeMap addProbe: floatProbe];

  [...]
  return obj;
}
```

Or, more compactly:

```
+ createBegin: aZone
{
  HeatbugModelSwarm *obj;
  id <ProbeMap> probeMap;

  [...]

  [probeMap addProbe: [[probeLibrary getProbeForVari-
able: "randomMoveProbability"
    inClass: [self class]]
                           setFloatFormat: "%.3f"]];
  [...]
  return obj;
}
```

C.4. SwarmObject Support for Probing

As mentioned earlier, all SwarmObjects can generate Probes/ProbeMaps. By default, they achieve this by "checking out" the appropriate references from the probeLibrary. SwarmObject *(see page 184)* from Objectbase Library *(see page 156)* library.

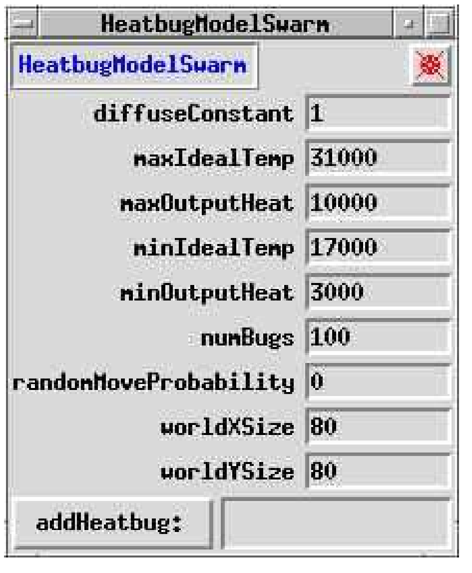
- **`- (const char*)getInstanceName`.** If you want to put a specific title on the ProbeDisplay window associated with your object, simply override this method. By default it returns the class name of the object. This is why the default ProbeDisplay window title is always the class name of the probed object. `Note`: SwarmObjects do not actually have an instance variable called "instanceName" - so if you want to store a name for your object, you will need to provide your own storage at the subclass level and rewrite `getInstanceName` to refer to this variable.
- **`- getProbeForVariable: (const char *)aVariable`.**
- **`- getProbeForMessage: (const char *)aMessage`.**
- **`- getProbeMap`.**
- **`- getCompleteProbeMap`.**

C.5. Probe support resident in the *Simtoolsgui Library (see page 260)* library:

C.5.1. ProbeDisplay

The `ProbeDisplay` class is meant to serve a dual role, represented by this figure and the previous figure:

Figure C-2. Customized probe map



On the one hand if an object to be probed is specified *without* any particular `ProbeMap` being specified, then the ProbeDisplay generated will provide a window of class DefaultProbeDisplay, which displays all the variables resident in that class structure.

On the other hand, if a `ProbeMap` *is* specified then the `ProbeDisplay` follows exactly the specification as represented by the contents of the ProbeMap (above). When used in this manner, ProbeDisplays can generate tailored interfaces to objects (so for example, we have purposefully hidden certain instance variables in the HeatbugModelSwarm class, and have shown only one of the methods which the class understands).

C.5.2. Graphical Interface Usage

Common to both the standard ProbeDisplay and the CompleteProbeDisplay:

- The different fields in the ProbeDisplay can be updated by typing in new values and pressing `Return`. However, certain fields (containing pointers or ids, for example) cannot be modified and will generate a beep if such a modification is attempted.
- If an instance variable/argument slot is defined to hold an object, then that object can be drag&dropped into another variable/argument slot by clicking on it with the `first mouse button` (a small rectangle with the name of the object will appear - simply drag it to another object-typed variable/argument slot and release the mouse button).
- Also, if an instance variable/argument slot is defined to hold an object, then that object can be inspected by clicking the entry for that variable/argument slot with the `third mouse button` (a ProbeDisplay for that object will be generated).

Available only on the standard ProbeDisplay:

- Note that the sunken label at the top of the ProbeDisplay is also active. By clicking on it with the `first mouse button` you get a drag&drop'able representation of *self*. By clicking on it with the `third mouse button` you get a CompleteProbeDisplay to *self*.

Available only on the ProbeDisplay:

- The green "superclass" button can be used to display the succesive superclasses of the object being probed.
- The red "hide" button can be used to hide classes which are irrelevant thus reducing clutter.
- The hide button on the lowest class in the hierarchy has a special meaning since clicking on it dismisses the entire ProbeDisplay.

ProbeDisplay *(see page 273)* from Simtoolsgui Library *(see page 260)* library.

C.6. ProbeDisplayManager

The normal Swarm simulation will probably only ever contain one instance of this class, namely the probeDisplayManager. This object is used for automatic generation of ProbeDisplays: given an object it will attempt to get a ProbeMap from the object. If the object does not respond to the method -getProbeMap (i.e. it does not inherit from SwarmObject), the probeDisplayManager will query the probeLibrary for a ProbeMap. If there is not a *ProbeMap* installed in the `probeLibrary` for the class of that object, it will creat a *DefaultProbeMap*. It will then create a ProbeDisplay to the target object, and from then on, until the Probe Display is removed it will update the ProbeDisplay when receiving an update message.

ProbeDisplayManager *(see page 274)* from Simtoolsgui Library *(see page 260)* library.

Appendix D. Objective C and Swarm

D.1. Objective C Web Resources

The main technical skill required of a Swarm user is the ability to write Objective C code. Other recommended skills include the ability to use tools such as gdb and emacs. For the record, if you know C and have some form of experience in either C++ or Smalltalk, then learning Objective C should take no more than a day or so.

- Objective C references *(http://www.santafe.edu/projects/swarm/ObjC/objective-c.html)* List of Objective C references on the Web. Includes a 10 minute overview to Objective C that is most of what you need to know about Objective C. Originally contributed by Nelson Minar
- *Object-Oriented Programming and the Objective-C Language (http://developer.apple.com/techpubs/macosxserver/ObjectiveC/index.html)* An excellent (online!) book on Objective C. This is for generic NeXT Objective C: Swarm uses GNU Objective C in addition to our own Defobj Library *(see page 21)* extensions.

D.2. Objective C - Swarm Style

The Swarm system provides a few extensions to the syntax/style of Objective-C. This note describes those features which are important to know as a beginner and those which can be ignored...

D.2.1. Non-Conventional Techniques, And The Libraries In Which They're Used

When building some of the internals of the Swarm system, we found that there was a need to add a certain amount of machinery to the conventional Objective C language. In particular, we found that the system used for object creation needed to be expanded. The main goal of this document is to explain very briefly what those changes are, and why the user need never know about them...

Even within the Swarm libraries, only the most fundamental ones require the usage of some of the magic described below. Essentially, only Defobj, Collections and Activity use the features described below. Nevertheless, there are some minor conventions, which as a consequence, must be used throughout the code.

D.2.2. Zones in Principle

Perhaps the most visible change from normal Objective C is the use of Zones within which objects are allocated. We have requiered that objects be allocated within specific Zones for two main reasons:

- **Probing.** Zones will be used to facilitate the process of Probing.
- **Garbage Collection.** In later versions of Swarm, the Zones will be used to coordinate garbage collection.
- **Parallelism.** In Swarm 2.0 we hope to use the Zone system to facilitate the management of distributed memory.

D.2.3. Zones in Practice

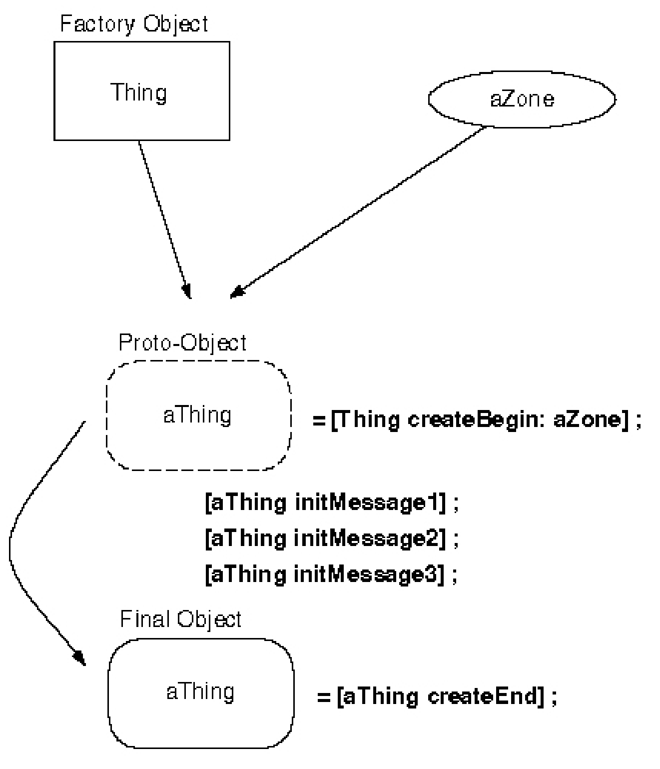
As a beginning user, when required to provide a Zone, you can simply use the globalZone variable which is global and is already initialised to a valid zone. But, this is discouraged because the things in that Zone are expected to be resident throughout the entire execution of the program and can lead to inefficient use of memory. Also, you should always attempt to allocate memory from Zones rather than using malloc. And when allocating a temporary object that won't be needed past the current scope, a scratchZone is provided.

D.2.4. The Create Phase in Principle

One of the more surprising mechanisms used in Swarm is the Create Phase protocol. The idea behind the protocol is that when you create an object, you are not really getting the final object - only a "proto-object". This "proto-object" is then sent a sequence of "create phase" messages which are meant to provide hints to the system about the way in which the object is going to be used.

So, for example, when creating a List object you may declare that you will only access the list from either end, never from an arbitrary location. By doing so you allow the "proto-object" to provide you with a tailored implementation which attempts to meet your specific usage patterns. This sort of approach is crucial for the performance-critical libraries in Swarm (such as the Activity library). Here is a schematic of what sometimes occurs in these libraries:

Figure D-1. Schematic of proto-object creation



D.2.5. The Create Phase in Practice

Since this form of object creation is still quite rare in the object oriented community, we do not expect the users to write code which actually implements this sort of technique. However, we do advise the users to try and split those messages which are supposed to be sent only once in the lifetime of an object (just after the object is created) from the ones that are sent multiple times.

For example, if your object has a variable which will never change and furthermore should be set at the very early stages of its lifetime (say, Color), then you should declare the message which sets this variable before the createEnd message:

```
-setColor: (int) aColor ; // createPhase message -createEnd ;
```

```
-(int) getColor ; // normal message
```

The only other requirement is that whenever you create any object in Swarm, you should always *re-assign* the object, when createEnd is called.

```
anObject = [anObject createEnd] ;
```

This is because certain proto-objects in the swarm library will not return themselves at createEnd time. In fact, they may return a completely different class of object, which happens to satisfy the needs of the user. We therefore suggest that even when createEnding your own classes you should stick to this standard creation format.

In some classes, you will see that some "convenience methods" for object creation have been provided. These are basically parametrized constructors that the author of the class felt would be handy. However, the objects that result from the use of these "convenience methods" should not differ in any way from an object created via a sequential set of separate methods called during the create phase (as long as each of the appropriate parameters are set). It is important to retain the "null constructor" for a couple of reasons: 1) it allows the interface for a class to *grow* without breaking legacy code and 2) it facilitates the use of development and design tools in handling objects. JavaBeans is one example where this particular constraint (the requirement of a null constructor) is in use and necessary, today. Of course, we don't have design and development tools like that for Swarm, yet...

D.2.6. Collections and Defobj

Two last pieces of advice:

1. When using the Collections library, if you want to create your own versions of the collections, don't do it by subclassing the collections themselves. Instead, create a different class of object which contains inside it, an instance of the appropriate class (this is called delegation in the OOP community). You should do this because the actual subclassing of objects which actively use the createPhase stuff (by actively I mean ones that don't return self on a createEnd), is not well supported.
2. Defobj is the most basic library in Swarm. It provides the foundation on which the system builds schedules, collections and swarmObjects. It is therefore quite involved and you should not feel disheartened if the concepts embodied within it seem unfamiliar, or if the code seems a bit cryptic.

You should now be ready to use Objective C in a more or less Swarm compliant way. Theoretically, all you need to do is follow these standards. In particular, you should be able to code your simulations without needing to know anything else about the language extensions (you still need to learn about the libraries and tools provided by Swarm, but their interface is written in straight Objective C, so there should be no language barrier).

D.3. Debugging Tips for Swarm

Debugging software is an art, perhaps even more so than writing the software itself. All software we write will have bugs: it is important to know how to diagnose a bug when it happens and how to write code defensively so you have less bugs in the first place.

D.3.1. Finding bugs

Bugs come in two categories: those that crash your program and those that don't. Bugs that crash your program are friendlier because they're obvious. Bugs that you don't notice are much more dangerous: one nagging question in every programmer's mind should be "is the program doing what I think it is doing?"

gdb. By far the most useful tool for finding bugs is a good debugger, a shell you can run a program under and set breakpoints, inspect the values of variables, etc. The best free debugger for Unix is probably gdb, available from the GNU ftp site *(ftp://prep.ai.mit.edu/pub/gnu/)* . gdb seems unfriendly and confusing at first, but it is definitely worth your time to learn it.

The most important gdb commands are `help`: browse online help, `where`: show me a stack trace, where did we crash?, `list`: show me the source code where we are, `break`, set a breakpoint, and `print`: show me the value of of some expression. If your program is crashing on you, run it under gdb and look at the stack trace. If it looks to be buggy but you don't know where, start setting breakpoints and see when things go awry.

gdb and Objective C. Unfortunately, at this time gdb does not directly support Objective C. There are some workarounds *(http://www.cons.org/cracauer/objc-hint-gdb.html)* that make debugging Objective C programs possible. They are based on the knowledge that Objective C is little more than a glorified syntax for structs (objects) and strange function names (methods).

defobj: xprint(), xprintid(), xfprint(), xfprintid(), xexec(), and xfexec(). Swarm also has a few functions defined that can be used to make debugging easier. In particular, the function xprint(object) prints out the class of an object, and xexec(object, "message") calls the message specified on the object. These can be invoked under gdb as `call xprint(aHeatbug)`. Note that you can't pass arguments to a message, nor can you see the return value. There are also methods xfprint(collection) and xfexec(collection, "message") that print or exec foreach member of a collection.

D.3.2. Preventing Bugs

Defensive programming can help prevent a good number of bugs. When writing code, try to test it incrementally: make small changes whose effects you think you can predict and then test them. Don't outsmart yourself with cleverness: write code correctly first, then go in and hack it up if you need it to be more efficient. Put in sanity checks for conditions that shouldn't go wrong in normal usage, but might if you make a mistake.

-Wall. Swarm currently compiles all code with "gcc -Wall", which tells gcc to emit warnings for a lot of things that it wouldn't normally complain about. Warnings are not (necessarily) errors - warnings will be generated for legal code if gcc thinks that what you're doing could easily be a mistake. You might find this frustrating at first, but it helps catch a lot of common errors, including forgetting to include a prototype or forgetting to return a value from a function. Passing -Wall is good discipline.

nil_method. Objects in Objective C are essentially pointers to structs. So what happens if you send an object to the pointer 0x0, "nil" in Objective C parlance? Unfortunately, most implementations of Objective C, including gcc, define methods to nil as having no effect. The code:

```
aHeatbug = [Heatbug create: aZone];
aHeatbug = 0; // oops! bug.
[aHeatbug setIdealTemperature: idealTemp];
```

will *not* generate any errors.

The reason this is unfortunate is that it's a common bug to trash a pointer accidentally, set it to 0. It would be nice if your program then crashed when you tried to send a message to that mangled object: instead, the message send will fail silently and the program will continue to execute. This can make it hard to find bugs.

There are two ways to make messages sent to nil crash your program. The simplest is to put a breakpoint on `nil_method` under gdb: nil_method is invoked every time a message is sent to nil. Alternately, you can make a copy of the libobjc runtime source and edit nil_method to do whatever you want. The source code in `src/defobj` contains a file `objc.patch` that patches the runtime from gcc 2.7.2.

D.4. Emacs and Objective C

Emacs is the world's best editor, infinitely configurable and powerful and free. If you're not used to a Unix text editor and want to write programs, emacs is probably best to learn. The information here will show you how to set up emacs to use an Objective C specific mode for editing source code and how to get emacs to colour your source code for you nicely. Emacs can also help you find compilation errors, run a debugger, and even act like a class browser. Documentation for those things are not here, see the Emacs documentation itself.

objc-mode. One of the best things about emacs is that one can load different "major modes" to edit types of files. Major modes making editing files easier by providing structure to your editing. In particular, there are programming language modes that do nice things like indentation for you.

Starting with emacs-19.30, the default C mode is cc-mode.el, a nice rewrite of the original c-mode.el. Luckily for us, cc-mode.el supports Objective C directly. The basic function to invoke it is `objc-mode`. The following bit of code in your .emacs will cause emacs to automatically enter objc-mode on all files that end in `.m` or `.h`:

```
(setq auto-mode-alist
(append '(("\\.h$" . objc-mode)
("\\.m$" . objc-mode))))
```

Highlighting code. Emacs running under X has the ability to colour your source code according to syntax. There are two packages to do this: `font-lock` and `hilit19`.

font-lock. emacs-20 and later also has a font-lock mode for Objective-C that supports the method syntax unique to Objective-C. We recommend the use of font-lock, over hilit19, now that the identification of Objective C syntax in emacs is better supported, as font-lock supports 'lazy-fontification' (i.e. as you type fontification), a feature that hilit19 never offered. See the manual provided with emacs for how to turn on this feature to highlight .h and .m files in 'objc-mode' automatically.

Appendix E. Library Interface Conventions

E.1. Overview

Some of the basic Swarm libraries are implemented using object definition conventions established by the Defobj Library *(see page 21)*. 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.

E.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 21)* 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.

E.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 21)* are divided into separate interfaces that define distinct phases of an object's life cycle.

E.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 21)* 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.

E.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.

E.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).

E.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 F. GNU Library General Public Licence

Terms

These are the license terms for the distribution of the Swarm libraries.

F.1. Text of Licence

```
  GNU LIBRARY GENERAL PUBLIC LICENSE
       Version 2, June 1991

 Copyright (C) 1991 Free Software Foundation, Inc.
                    675 Mass Ave, Cambridge, MA 02139, USA
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.

[This is the first released version of the library GPL.  It is
 numbered 2 because it goes with version 2 of the ordinary GPL.]

   Preamble

  The licenses for most software are designed to take away your
freedom to share and change it.  By contrast, the GNU General Public
Licenses are intended to guarantee your freedom to share and change
free software–to make sure the software is free for all its users.

  This license, the Library General Public License, applies to some
specially designated Free Software Foundation software, and to any
other libraries whose authors decide to use it.  You can use it for
your libraries, too.

  When we speak of free software, we are referring to freedom, not
price.  Our General Public Licenses are designed to make sure that you
have the freedom to distribute copies of free software (and charge for
this service if you wish), that you receive source code or can get it
if you want it, that you can change the software or use pieces of it
```

in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library, or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link a program with the library, you must provide complete object files to the recipients so that they can relink them with the library, after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

Our method of protecting your rights has two steps: (1) copyright the library, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the library.

Also, for each distributor's protection, we want to make certain that everyone understands that there is no warranty for this free library. If the library is modified by someone else and passed on, we want its recipients to know that what they have is not the original version, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that companies distributing free software will individually obtain patent licenses, thus in effect transforming the program into proprietary software. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License, which was designed for utility programs. This license, the GNU Library General Public License, applies to certain designated libraries. This license is quite different from the ordinary one; be sure to read it in full, and don't assume that anything in it is the same as in the ordinary license.

The reason we have a separate public license for some libraries is that they blur the distinction we usually make between modifying or adding to a program and simply using it. Linking a program with a library, without changing the library, is in some sense simply using the library, and is analogous to running a utility program or application program. However, in a textual and legal sense, the linked executable is a combined work, a derivative of the original library, and the ordinary General Public License treats it as such.

Because of this blurred distinction, using the ordinary General Public License for libraries did not effectively promote software sharing, because most developers did not use the libraries. We concluded that weaker conditions might promote sharing better.

However, unrestricted linking of non-free programs would deprive the users of those programs of all benefit from the free status of the libraries themselves. This Library General Public License is intended to permit developers of non-free programs to use free libraries, while preserving your freedom as a user of such programs to change the free libraries that are incorporated in them. (We have not seen how to achieve this as regards changes in header files, but we have achieved it as regards changes in the actual functions of the Library.) The hope is that this will lead to faster development of free libraries.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, while the latter only works together with the library.

Note that it is possible for a library to be covered by the ordinary General Public License rather than by this special one.

GNU LIBRARY GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Library General Public License (also called "this License"). Each licensee is

addressed as "you".

A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) The modified work must itself be a software library.

b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.

c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.

d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

(For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or

linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also compile or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

b) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

c) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

d) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.

b) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further

restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus

excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Library General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU

FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

Appendix: How to Apply These Terms to Your New Libraries

If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the ordinary General Public License).

To apply these terms, attach the following notices to the library. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one's name and a brief idea of what it does.>
Copyright (C) <year> <name of author>

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Library General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Library General Public License for more details.

You should have received a copy of the GNU Library General Public License along with this library; if not, write to the Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

Also add information on how to contact you by electronic and paper mail.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the library, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the
library ‘Frob’ (a library for tweaking knobs) written by James Random Hacker.

<signature of Ty Coon>, 1 April 1990
Ty Coon, President of Vice

That’s all there is to it!

Appendix G. GNU General Public Licence

Terms

These are the license terms for the distribution of the Swarm example applications and the Swarm documentation.

G.1. Text of Licence

```
 GNU GENERAL PUBLIC LICENSE
            Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.
59 Temple Place - Suite 330, Boston, MA 02111-1307, USA

Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.
```

Preamble
========

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software–to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or

translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   a. You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

   b. You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

   c. If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

   a. Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   b. Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c. Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions

are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

   If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

   It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

ONE LINE TO GIVE THE PROGRAM'S NAME AND A BRIEF IDEA OF WHAT IT DOES.
Copyright (C) 19YY NAME OF AUTHOR

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) 19YY NAME OF AUTHOR
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details
type ‘show w’.
This is free software, and you are welcome to redistribute it
under certain conditions; type ‘show c’ for details.

The hypothetical commands ‘show w’ and ‘show c’ should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than ‘show w’ and ‘show c’; they could even be mouse-clicks or menu items–whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program
‘Gnomovision’ (which makes passes at compilers) written by James Hacker.

SIGNATURE OF TY COON, 1 April 1989

Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

Protocol Index

Method Index