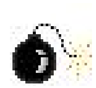
Simtools Library

Documentation and Implementation Status



Compatibility

- `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.`

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, the ControlPanel and GUISwarm which support GUI interactions, but also UName which simply generates unique names for objects.

Documentation

The *`objectbase`* library is defined and documented following the basic structure of the `Library Interface Conventions`

`(defobj/library.html)`
. The following standard sections of documentation are available:

- `Interface Reference`
`(simtools/Interface.html)`

- Usage Guide
(simtools/UsageGuide.html)
- Advanced Usage Notes
- Interface Design Notes
- Implementation Notes

Marcus G. Daniels <mgd@santafe.edu>
(mailto:mgd@santafe.edu)

Last modified: 1998-01-08

Simtools Library: 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.

Marcus G. Daniels <mgd@santafe.edu>
(mailto:mgd@santafe.edu)

Last modified: 1998-01-08

Simtools Library: Interface Reference

The *simtools* library uses the module definition conventions
(../defobj/module.html)
established by the *defobj* library
(../defobj.html)
. It also depends on standard supertypes and classes defined by this library.

Dependencies

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

```
#import <objectbase.h>
#import <random.h> // for global RNGs...
#import <tkobjc.h> // some methods return graphics stuff...
```

The *objectbase* library interface is included to provide the basic object support. The *random*library is included to provide support for default generators and distributions, the use of which is deprecated, but are included for backwards compatibility.

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. `uniformIntRand`, `probeDisplayManager`, and so on).

void initSwarm(int argc, char ** argv)
: Initializes the Swarm libraries. This should be the call in any Swarm code you write. `argc` and `argv` are the input parameters to `main()`.

ProbeDisplayManager * probeDisplayManager
: A global instance of the ProbeDisplayManager object. You can use this object to create new probe displays, and should schedule an update message to it in your display schedule so as to update the probe displays it has created. See the documentation on probes
(../probing.html)
for more details.

int swarmGUIMode
: A flag that indicates whether graphics are enabled. Set automatically based on whether the switch `-batchmode` was present as one of the command line arguments starting the application (e.g. `heatbugs -batchmode`). If it is false, then

globalTkInterp (from libtkobjc) is set to be a normal Tcl interpreter instead of a Tcl/Tk/BLT interpreter.

Object Types (with links to reference pages)

As simtools is a collection of tools that are only loosely related to each other - the class hierarchy is virtually flat. Most of the classes are standalone.

Contents:

- GUISwarm
(GUISwarm.html)
- ControlPanel
(ControlPanel.html)
- ActionCache
(ActionCache.html)
- ProbeDisplay
(ProbeDisplay.html)
- CompleteProbeDisplay
(CompleteProbeDisplay.html)
- ProbeDisplayManager
(ProbeDisplayManager.html)
- InFile
(InFile.html)
- OutFile
(OutFile.html)
- AppendFile
(AppendFile.html)
- ObjectLoader
(ObjectLoader.html)
- ObjectSaver
(ObjectSaver.html)
- QSort
(QSort.html)
- NSelect

(NSelect.html)

- UName
(UName.html)
- ActiveGraph
(ActiveGraph.html)
- ActiveOutFile
(ActiveOutFile.html)

Marcus G. Daniels <mgd@santafe.edu>
(mailto:mgd@santafe.edu)

Last modified: 1998-01-10

Name

GUISwarm

Synopsis

```
//
// GUISwarm -- a version of the Swarm class which is graphics aware
//

@protocol GUISwarm <SwarmProcess>
@end
```

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.

- 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).

Marcus G. Daniels <mgd@santafe.edu>
(mailto:mgd@santafe.edu)

Last modified: 1998-01-08

Name

ControlPanel - class to control the top level SwarmProcess

Synopsis

```
// ControlPanel --
// an class used to control the top-level SwarmProcess. It
// accepts manipulations both by the Swarm it's controlling and
// the ActionCache.
//

@protocol ControlPanel <SwarmObject>
CREATING
- createEnd;
USING
- getState;
- setState: s;
- waitForControlEvent;
- startInActivity: activityID;
- setStateRunning;
- setStateStopped;
- setStateStepping;
- setStateQuit;
- setStateNextTime;

// Deprecated methods
- doTkEvents;
- getPanel;
@end

// State Symbols for the ControlPanel.
extern id ControlStateRunning, ControlStateStopped;
extern id ControlStateStepping, ControlStateNextTime, ControlStateQuit;
```

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.

- getState

Get the current button state of the controlpanel. Is one of ControlStateRunning, ControlStateStopped, ControlStateStepping, ControlStateNextTime, or ControlStateQuit.

- setStateRunning, - setStateStopped, - setStateStepping, - setStateQuit, - setStateNextTime

  Sets the state to the indicated value, and also sends the appropriate stop or terminate message to the running activity tree. 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).

Marcus G. Daniels <mgd@santafe.edu>
(mailto:mgd@santafe.edu)

Last modified: 1998-01-08

Name

ActionCache - a class to manage threads and Swarms

Synopsis

```
//
// ActionCache -
//
// 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.
//

@protocol ActionCache <SwarmObject>
CREATING
- setControlPanel: cp;
- setScheduleContext: context;
- setWindowGeometryRecordName: (const char *)name;
- createEnd;

USING
- insertAction: actionHolder;
- deliverActions;
- sendActionOfType: (id <Symbol>)type toExecute: (const char *)cmd;
- sendStartAction;
- sendStopAction;
- sendStepAction;
- sendNextAction;
- sendQuitAction;
- verifyActions;

- createProcCtrl;
- getPanel;
- doTkEvents; //should change to pollGUI or something
@end

// Type Symbols for ActionCache
extern id <Symbol> Control, Probing, Spatial;
// Error Symbols for ActionCache
extern id <Symbol> InvalidActionType, ActionTypeNotImplemented;
```

Description

- setWindowGeometryRecordName: (const char *)name
  Set the name to be used for archiving the geometry of the control panel widget. There is no default; the position of the control panel will not be preserved unless you use this method.

Marcus G. Daniels <mgd@santafe.edu>
(mailto:mgd@santafe.edu)

Last modified: 1998-01-08

Name

ProbeDisplay - a class to display ProbeMaps

Synopsis

```
//
// ProbeDisplay
//
// A class which generates a GUI to a ProbeMap of probes applied to a
// given target object.
//

@protocol ProbeDisplay <SwarmObject>
CREATING
- setProbedObject: anObject;
- setProbeMap: (ProbeMap *)probeMap;
- setWindowGeometryRecordName: (const char *)name;
USING
- getProbedObject;
- getProbeMap;
- update; // implemented in SimpleProbeDisplay...
@end
```

Description

Create Phase Protocol

- setProbedObject: anObject

  This method *must* be called.

- setProbeMap: (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.

- setWindowGeometryRecordName: (const char *)name

In order to record the window position of a ProbeDisplay, it is necessary to give it a name. This method is used to give an instance ProbeDisplay a name, which will used by the Archiver when recording its geometry information.

Regular Protocol

- getProbedObject

- getProbeMap

- 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.

Marcus G. Daniels <mgd@santafe.edu>
(mailto:mgd@santafe.edu)

Last modified: 1998-01-08

Name

CompleteProbeDisplay

Synopsis

```
//
// CompleteProbeDisplay --
// 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).
//

@protocol CompleteProbeDisplay <SwarmObject>
CREATING
- setProbedObject: anObject;
- setWindowGeometryRecordName: (const char *)name;
USING
- getProbedObject;
- update;
- getMarkedForDropFlag;
@end
```

Description

Marcus G. Daniels <mgd@santafe.edu>
(mailto:mgd@santafe.edu)

Last modified: 1998-01-08

Name

ProbeDisplayManager

Synopsis

```
//
// ProbeDisplayManager --
// A singleton class whose instance is used to manage all the
// ProbeDisplays created by the user during a GUI run of the
// simulation.
//

@protocol ProbeDisplayManager <SwarmObject>
USING
- createProbeDisplayFor: anObject;
- createProbeDisplayFor: anObject setWindowGeometryRecord-
Name: (const char *)name;

- createCompleteProbeDisplayFor: anObject;
- createCompleteProbeDisplayFor: anObject setWindowGeometryRecord-
Name: (const char *)name;

- addProbeDisplay: probeDisplay;
- removeProbeDisplayFor: anObject;
- removeProbeDisplay: probeDisplay;
- setDropImmediatelyFlag: (BOOL)dropImmediateFlag;
- update;
@end
```

Description

- addProbeDisplay: pd

- removeProbeDisplay: pd

  These methods are used to specify which Probe Displays should be managed by the ProbeDisplayManager.

- update

  This method will recursively send an update message to all the Probe Displays managed by the ProbeDisplayManager.

- createProbeDisplayFor: anObject
- createDefaultProbeDisplayFor: anObject
- createCompleteProbeDisplayFor: anObject
- createProbeDisplayFor: anObject setWindowGeometryRecordName: (const char *)name
- createCompleteProbeDisplayFor: anObject setWindowGeometryRecordName: (const char *)name;
- dropProbeDisplaysFor: anObject

  These methods allow a one-step creation/removal of ProbeDisplays onto a given object (dropProbeDisplaysFor: will actually *drop* the appropriate ProbeDisplays). The setWindowGeometryRecordName: creation variety can be used to set names for archiving geometry information.

Marcus G. Daniels <mgd@santafe.edu>
(mailto:mgd@santafe.edu)

Last modified: 1997-12-19

Name

InFile - class to perform file input

Synopsis

```
//
// InFile --
// A class which was intended to support file input. There have been
// justified requests from our userbase to re-design this interface.
//

@protocol InFile <SwarmObject>
CREATING
+ create: aZone withName: (const char *)theName;
USING
- (int)getWord: (char *)aWord;
- (int)getLine: (char *)aLine;
- (int)getInt: (int *)anInt;
- (int)getUnsigned: (unsigned *)anUnsigned;
- (int)getLong: (long *)aLong;
- (int)getUnsignedLong: (unsigned long *)anUnsLong;
- (int)getDouble: (double *)aDouble;
- (int)getFloat: (float *)aFloat;
- (int)getChar: (char *)aChar;
- (int)unGetChar: (char)aChar;
- (int)skipLine;
@end
```

Description

This class is 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.

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

This is the create method for InFiles, where theName is, of course the name of the file to open.

`- (int)getWord: (char *)aWord`

We use the term word instead of string because the returned value will not contain spaces, tabs and newlines -> thus getWord...

- (int)getLine: (char *)aLine
  This routine loads the argument string with the characters up to, but not including a newline character.

- (int)getInt: (int *)anInt;

- (int)getUnsigned: (unsigned *)anUnsigned;

- (int)getLong: (long *)aLong;

- (int)getUnsignedLong: (unsigned long *)anUnsLong;

- (int)getDouble: (double *)aDouble;

- (int)getFloat: (float *)aFloat;

- (int)getChar: (char *)aChar;
  These routines take a pointer to the required type, and read into it an instance of that type from the open file. In case of failure these message returns 0.

- (int)unGetChar: (char) aChar

- (int)skipLine
  These routines are used by ObjectLoader, but are documented in case the user wishes to define his/her own file format which may require backtracking or skipping lines.

- (void)drop
  This method must be called when the user wants to close the file. Only by dropping the InFile object will the file truly be closed!

Marcus G. Daniels <mgd@santafe.edu>
(mailto:mgd@santafe.edu)

Last modified: 1998-01-08

Name

OutFile - class to perform file output

Synopsis

```
//
// OutFile --
// A class which was intended to support file output.
//

@protocol OutFile <SwarmObject>
CREATING
+ create: aZone withName: (const char *)theName;
USING
- putString: (const char *)aString;
- putInt: (int) anInt ;
- putUnsigned: (unsigned) anUnsigned;
- putLong: (long) aLong ;
- putUnsignedLong: (unsigned long) anUnsLong ;
- putDouble: (double) aDouble ;
- putFloat: (float) aFloat ;
- putChar: (char) aChar ;
- putTab ;
- putNewLine ;
@end
```

Description

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.

`+ create: aZone withName: (const char *)theName`
: This is the create method for OutFiles, where theName is, of course the name of the file to open.

`- putString: (const char *)aString`

`- putInt: (int) anInt`

- putUnsigned: (unsigned) anUnsigned;
- putLong: (long) aLong;
- putUnsignedLong: (unsigned long)anUnsLong;
- putDouble: (double)aDouble
- putFloat: (float)aFloat
- putChar: (char)aChar
- putTab: (char)aChar
- putNewLine

    These routines take an instance of the appropriate type and write it into the open file. The return value is simply self since no error detection is performed.

- (void)drop

    This method must be called when the user wants to close the file. Only by dropping the OutFile object will the file truly be closed!

Marcus G. Daniels <mgd@santafe.edu>
(mailto:mgd@santafe.edu)

Last modified: 1998-01-08

Name

AppendFile - class for appended file output

Synopsis

```
//
// AppendFile --
// A class to support appended file output.
//
@protocol AppendFile <SwarmObject>CREATING
+ create: aZone withName: (const char *)theName;
USING
- putString: (const char *)aString;
- putInt: (int)anInt;
- putUnsigned: (unsigned)anUnsigned;
- putLong: (long)aLong;
- putUnsignedLong: (unsigned long)anUnsLong;
- putDouble: (double)aDouble;
- putFloat: (float)aFloat;
- putChar: (char)aChar;
- putTab;
- putNewLine;
@end
```

Description

This class subclasses from OutFile, the only difference in functionality, being that it opens a given file in *Append*

mode, rather than *Overwrite*mode, which OutFile does.

+ create: aZone withName: (const char *)theName
  This is the create method for AppendFiles, where theName is, of course the name of the file to open.

- putString: (const char *)aString

- putInt: (int)anInt

- putUnsigned: (unsigned)anUnsigned;

- putLong: (long)aLong;
- putUnsignedLong: (unsigned long)anUnsLong;
- putDouble: (double)aDouble
- putFloat: (float)aFloat
- putChar: (char)aChar
- putTab: (char)aChar
- putNewLine

  These routines take an instance of the appropriate type and write it into the open file. The return value is simply self since no error detection is performed.
- (void)drop

  This method *must* be called when the user wants to close the file. Only by dropping the AppendFile object will the file truly be closed!

Marcus G. Daniels <mgd@santafe.edu>
(mailto:mgd@santafe.edu)

Last modified: 1997-12-19

Name

ObjectLoader - class to load objects IVars from a file

Synopsis

```
//
// ObjectLoader --
// An attempt to design some form of standard for object loading.
//

@protocol ObjectLoader <SwarmObject>
USING
+ load: anObject from: aFileObject;
+ load: anObject fromFileNamed: (const char *)aFileName;

- setFileObject: aFileObject;
- loadObject: anObject;
// In case the same class is being loaded multiple times...
- updateCache: exampleTarget;
@end
```

Description

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:

```
# A Comment

@begin

aVarName aValue
aVarName aValue # A comment
aVarName aValue
#
# A Comment
#
aVarName aValue
```

```
@end
```

Note: A single file can be used to initialize multiple objects by simply inserting more than one @begin @end block.

+ load: anObject fromFileNamed: (const char *)aFileName This convenience class method will load anObject from the file named aFileName (the class will open the file, initialize the object with its contents and then close the file).
+ load: anObject from: aFileObject This convenience class method will load anObject from the previously opened aFileObject without returning an actual instance of the ObjectLoader class (also, the FileObject remains open after the method has been called).
- setFileObject: aFileObject If the user bothers to generates an actual instance of the ObjectLoader class, s/he can set the source fileObject which the instance should use by sending it this message.
- loadObject: anObject This is the message the user must send an instance of the ObjectLoader class in order to initialize the target object from the requested file.
- updateCache This method should be called if an ObjectLoader instance is going to initialize a large number of objects from the same class.
- setTemplateProbeMap: aProbeMap This 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.

Marcus G. Daniels <mgd@santafe.edu>
(mailto:mgd@santafe.edu)

Last modified: 1998-01-08

Name

ObjectSaver - class to save object's IVars to a file

Synopsis

```
//
// ObjectSaver --
// An attempt to design some form of standard for object saving.
// The ProbeMap argument is used to specify what variables get saved.
//

@protocol ObjectSaver <SwarmObject>
USING
+ save: anObject to: aFileObject;
+ save: anObject to: aFileObject withTemplate: aProbeMap;
+ save: anObject toFileNamed: (const char *)aFileName;
+ save: anObject toFileNamed: (const char *)aFileName withTemplate: (Probe-
Map *)aProbeMap;

- setFileObject: aFileObject;
- setTemplateProbeMap: aProbeMap;
- saveObject: anObject;
@end
```

Description

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).

+ save: anObject to: aFileObject

+ save: anObject to: aFileObject withTemplate: aProbeMap

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

+ save: anObject toFileNamed: (const char *)aFileName withTemplate: aProbeMap

As in the case of ObjectLoader we provide a set of convience class methods which will save the entire target object (or the subset of variables specified in a template) without actually returning an instance of ObjectSaver to the user.

- setFileObject: aFileObject
  If the user bothers to generates an actual instance of the ObjectSaver class, s/he can set the target fileObject which the instance should use by sending it this message.
- saveObject: anObject
  This is the message the user must send an instance of the ObjectSaver class in order to save the state of the target object into the requested file.
- setTemplateProbeMap: aProbeMap
  This 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.

Saving floats and doubles: The desired number of significant figures for all floats and doubles can be set in the global instance of the ProbeLibrary. However, if notset, it will default to a value of six (6) significant figures, which is identical to previous versions that did not have this feature. See the documentation on probes (../probing.html)
for how to set this precision.

Marcus G. Daniels <mgd@santafe.edu>
(mailto:mgd@santafe.edu)

Last modified: 1998-01-08

Name

QSort - class to sort a collection

Synopsis

```
//
// QSort --
// A class (not to be instantiated) wrapper around the C sort routine.
//

@protocol QSort <SwarmObject>
USING
+ sortObjectsIn: aCollection;
+ (void)sortObjectsIn: aCollection using: (SEL)aSelector;
+ (void)sortNumbersIn: aCollection;
+ (void)sortNumbersIn: aCollection us-
ing: (int(*)(const void*,const void*)) comp_fun;
+ (void)reverseOrderOf: aCollection;
@end
```

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.

    +(void) sortObjectsIn:  aCollection
         Given a collection, this method will sort the objects with the "compare" function
         for the object.  If the objects don't provide a compare function, it uses
         the default from the defobj library.
    +(void)sortObjectsIn:  aCollection using:  (SEL)aSelector
         This method allows the user to specify a given message selector to use to
         "compare" the objects.
    +(void) sortNumbersIn:  aCollection;
         This method is intended specifically for integer comparisons, again using
         a default "compare" function.  The default assumes numbers should be monotonically
         increasing.
```

+(void) sortNumbersIn: aCollection using: (int(*)(const void*,const void*))comp_fun

As above, except that a specific function is given.

+(void) reverseOrderOf: aCollection

Reverses the current order of a collection. To make a "reversed" (i.e. decending) sort, simply call one of the appropriate "sort..." methods on a collection, then call this function on the same collection.

Marcus G. Daniels <mgd@santafe.edu>
(mailto:mgd@santafe.edu)

Last modified: 1998-01-08

Name

NSelect - class to select exactly N elements at random from a collection

Synopsis

```
//
// NSelect --
// A class (not to be instantiated) wrapper around a Knuth algorithm
// for the selection of exactly N elements form a collection without
// repetition.
//

@protocol NSelect <SwarmObject>
USING
+ (void)select: (int)n from: aCollection into: bCollection;
@end
```

Description

NSelect selects exactly N elements from a collection without repetition. A target collection must be provided.

Marcus G. Daniels <mgd@santafe.edu>
(mailto:mgd@santafe.edu)

Last modified: 1998-01-08

Name

UName - class to generate unique names for objects/agents

Synopsis

```
//
// UName --
// A class used to generate unique names (e.g. "critter1", "critter2" etc).
//

@protocol UName <SwarmObject>
CREATING
+ create: aZone setBaseName: (const char *)aString;
+ create: aZone setBaseNameObject: aStringObject;

- setBaseName: (const char *)aString;
- setBaseNameObject: aStringObject;
USING
- (const char *)getNewName;
- getNewNameObject;
@end
```

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`

`(../collections/String.html)`
. 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!

+ create: aZone setBaseName: (const char *)aString

```
+ create: aZone setBaseNameObject: aStringObject

- setBaseName: (const char *)aString

- setBaseNameObject: aStringObject
      These messages should be used to set or reset the base name.
      Note: these methods will automatically reset the counter.

- (const char *)getNewName

- getNewNameObject
      Generates a new name in the requested format.

- resetCounter
      Resets the counter used as a suffix in the unique names generated.
```

Marcus G. Daniels <mgd@santafe.edu>
(mailto:mgd@santafe.edu)

Last modified: 1997-12-19

Name

ActiveGraph - provides continous data feed between Swarm and the GUI

Synopsis

```
//
// ActiveGraph --
// This class provides the continuous data feed between Swarm and the GUI.
//

@protocol ActiveGraph <MessageProbe>
USING
- setElement: ge;
- setDataFeed: d;
- step;
@end
```

Description

An active graph object is the glue between a MessageProbe (for reading data) and a BLTGraph 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.

- setElement: get
  Sets the graph element used to draw on
- setDataFeed: d
  Sets the object that will be probed for data.
- setProbedSelector: (SEL)s
  Sets the message sent to the object for graphing (inherited from MessageProbe).
- 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.

Marcus G. Daniels <mgd@santafe.edu>
(mailto:mgd@santafe.edu)

Last modified: 1998-01-08

Name

ActiveGraph - provides continous data feed between Swarm and a file

---

Synopsis

```
//
// ActiveOutFile --
// Provides the continuous data feed between Swarm and a File.
//

@protocol ActiveOutFile <MessageProbe>
USING
- setFileObject: aFileObj;
- setDataFeed: d;
- step;
@end
```

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. 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.

```
- setFileObject: anOutFile
     Sets the file object to which the data will be sent.
- setDataFeed: d
     Sets the object that will be probed for data.
- setProbedSelector: (SEL) s
     Sets the message sent to the target object (inherited from MessageProbe).
- step
     Fires the probe, reads the value from the object, and sends it to the file.
     The X value is implicitly the current simulation time.  Y is the value read.
```

Marcus G. Daniels <mgd@santafe.edu>

(mailto:mgd@santafe.edu)

Last modified: 1998-01-08

Simtools Library: Usage Guide

Overview

The *simtools* library contains a number of semi-related tools that are useful for writing simulations.

(Interface.html)

(../simtools.html)

(../simtools.html)

Marcus G. Daniels <mgd@santafe.edu>
(mailto:mgd@santafe.edu)

Last modified: 1998-01-10