- 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: 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

(NSelect.html)

- UName
(UName.html)

- ActiveGraph
(ActiveGraph.html)

- ActiveOutFile
(ActiveOutFile.html)

(../simtools.html)

(../simtools.html)

(UsageGuide.html)

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

Last modified: 1998-01-10

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

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

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`

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

- 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

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

- 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

- 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

@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

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

+(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

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

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.

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>

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