Swarm 2.1 Reference Guide

Swarm Development Group

Mar 1, 2000

Contents

Chapter 1

Overview

The following are all the packages contained in this document:

Chapter 2

Package swarm.defobj

Interfaces in package:

Classes in package:

2.1 swarm.defobj.Archiver

public interface *Archiver*
implements CreateS,Create,DropS,Drop

High level abstract serialization interface.. High level abstract serialization interface.

2.1.1 Methods

public Object ***getObject****(java.lang.String key)*

Create the object with 'key' using the Archiver's own Zone

public Object ***getWithZone$key****(swarm.defobj.Zone aZone, java.lang.String key)*

Create the object with 'key' in the specified Zone

public void ***putDeep$object****(java.lang.String key, java.lang.Object object)*

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

public void ***putShallow$object****(java.lang.String key, java.lang.Object object)*

As per -putDeep, but only make a shallow version

public void ***registerClient****(java.lang.Object client)*

public void ***sync****()*

Ensure that that all registered the requested backend

public void ***unregisterClient****(java.lang.Object client)*

2.2 swarm.defobj.ArchiverC

public interface *ArchiverC*
implements CreateS,CreateC,DropS,DropC

High level abstract serialization interface.. High level abstract serialization interface.

2.2.1 Methods

public Object ***setDefaultAppPath()***

Specify that the Archiver to use the default application path

public Object ***setDefaultPath()***

Specify that the Archiver instance use the default system path

public Object ***setInhibitLoadFlag****(boolean inhibitLoadFlag)*

Make the Archiver ignore any file found in the specified path

public Object ***setPath****(java.lang.String path)*

Set the physical path for the Archiver to read/write

public Object ***setSystemArchiverFlag****(boolean systemArchiverFlag)*

Make the Archiver expect application metadata, such as 'mode' information

2.3 swarm.defobj.ArchiverS

public interface *ArchiverS*
implements CreateS,DropS

High level abstract serialization interface.. High level abstract serialization interface.

2.4 swarm.defobj.BehaviorPhase

public interface *BehaviorPhase*
implements CreatedClassS,CreatedClass

Created class which implements a phase of object behavior.. Created class which implements a phase of object behavior.

2.4.1 Methods

*public Object **getNextPhase()***

2.5 swarm.defobj.BehaviorPhaseC

public interface *BehaviorPhaseC*
implements CreatedClassS,CreatedClassC

Created class which implements a phase of object behavior.. Created class which implements a phase of object behavior.

2.5.1 Methods

public void ***setNextPhase****(java.lang.Object aClass)*

2.6 swarm.defobj.BehaviorPhaseS

public interface *BehaviorPhaseS*
implements CreatedClassS

Created class which implements a phase of object behavior.. Created class which implements a phase of object behavior.

2.7 swarm.defobj.Copy

public interface *Copy*

Copy all state defined as part of object.. 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.

2.7.1 Methods

public Object ***copy****(swarm.defobj.Zone aZone)*

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

2.8 swarm.defobj.CopyC

public interface *CopyC*

Copy all state defined as part of object.. 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.

2.9 swarm.defobj.CopyS

public interface *CopyS*

Copy all state defined as part of object.. 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.

2.10 swarm.defobj.Create

public interface *Create*
implements DefinedObjectS,DefinedObject,CustomizeS,Customize

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

2.11 swarm.defobj.CreateC

public interface *CreateC*
implements DefinedObjectS,DefinedObjectC,CustomizeS,CustomizeC

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

2.11.1 Methods

public Object ***create****(swarm.defobj.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.

public Object ***createBegin****(swarm.defobj.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.

public Object ***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.

2.12 swarm.defobj.CreatedClass

public interface *CreatedClass*
implements CreateS,Create,DefinedClassS,DefinedClass

Class with variables and/or methods defined at runtime.. Class with variables and/or methods defined at runtime.

2.12.1 Methods

public Object ***getDefiningClass()***

2.13 swarm.defobj.CreatedClassC

public interface *CreatedClassC*
implements CreateS,CreateC,DefinedClassS,DefinedClassC

Class with variables and/or methods defined at runtime.. Class with variables and/or methods defined at runtime.

2.13.1 Methods

public Object ***hdf5InCreate****(java.lang.Object hdf5Obj)*

public Object ***hdf5OutShallow****(java.lang.Object hdf5Obj)*

public Object ***lispInCreate****(java.lang.Object expr)*

public Object ***lispOutShallow****(java.lang.Object stream)*

public Object ***setClass****(java.lang.Object aClass)*

public Object ***setDefiningClass****(java.lang.Object aClass)*

public Object ***setName****(java.lang.String name)*

public Object ***setSuperclass****(java.lang.Object aClass)*

public Object ***updateArchiver****(java.lang.Object archiver)*

2.14 swarm.defobj.CreatedClassS

public interface *CreatedClassS*
implements CreateS,DefinedClassS

Class with variables and/or methods defined at runtime.. Class with variables and/or methods defined at runtime.

2.15 swarm.defobj.CreateS

public interface *CreateS*
implements DefinedObjectS,CustomizeS

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

2.16 swarm.defobj.Customize

public interface *Customize*

Create-phase customization.. 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.

2.17 swarm.defobj.CustomizeC

public interface *CustomizeC*

Create-phase customization.. 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.

2.17.1 Methods

public Object ***customizeBegin****(java.lang.Object 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.

public Object ***customizeCopy****(java.lang.Object aZone)*

The customizeCopy: message creates a new copy of the interim object returned by customizeBegin: which may be used for further customizations that do not affect the customization already in progress. It may be used to branch off a path of a customization in progress to create an alternate final customization. customizeCopy may be used only on an interim object returned by customizeBegin: and not yet finalized by customizeEnd. The new version of the interim object being customized may be allocated in the same or different zone as the original version, using the zone argument required by customizeCopy:

public Object ***customizeEnd()***

Returns the new, customized version of the original type.

2.18 swarm.defobj.CustomizeS

public interface *CustomizeS*

Create-phase customization.. 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.

2.19 swarm.defobj.DefinedClass

---

public interface *DefinedClass*
implements DefinedObjectS,DefinedObject,GetNameS,GetName

Class which implements an interface of a type.. Class which implements an interface of a type.

2.19.1 Methods

public Object ***getSuperclass****()*

public Object ***getTypeImplemented****()*

public boolean ***isSubclass****(java.lang.Object aClass)*

public void ***setTypeImplemented****(java.lang.Object aType)*

2.20 swarm.defobj.DefinedClassC

public interface *DefinedClassC*
implements DefinedObjectS,DefinedObjectC,GetNameS,GetNameC

Class which implements an interface of a type.. Class which implements an interface of a type.

2.21 swarm.defobj.DefinedClassS

public interface *DefinedClassS*
implements DefinedObjectS,GetNameS

Class which implements an interface of a type.. Class which implements an interface of a type.

2.22 swarm.defobj.DefinedObject

public interface *DefinedObject*

Object with defined type and implementation.. 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.

2.22.1 Methods

public int ***compare****(java.lang.Object anObject)*

A local implementation of an Object method.

public void ***describe****(java.lang.Object 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.

public void ***describeID****(java.lang.Object outputCharStream)*

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

public String ***getDisplayName****()*

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

public String ***getTypeName****()*

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

public Zone ***getZone****()*

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

public Object ***perform****(swarm.Selector aSel)*

A local implementation of an Object method.

public Object ***perform$with****(swarm.Selector aSel, java.lang.Object anObject1)*

A local implementation of an Object method.

public Object ***perform$with$with****(swarm.Selector aSel, java.lang.Object anObject1, java.lang.Object anObj2)*

A local implementation of an Object method.

public Object ***perform$with$with$with****(swarm.Selector aSel, java.lang.Object anObject1, java.lang.Object anObj2, java.lang.Object anObj3)*

Perform a selector with three object arguments.

public boolean ***respondsTo****(swarm.Selector 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.

public void ***setDisplayName****(java.lang.String displayName)*

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

public void ***xfprint****()*

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

public void ***xfprintid****()*

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

public void ***xprint****()*

Like describe:, but output goes to standard output.

public void ***xprintid()***

Like describeID:, but output goes to standard output.

2.23 swarm.defobj.DefinedObjectC

public interface *DefinedObjectC*

Object with defined type and implementation.. 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.

2.24 swarm.defobj.DefinedObjectS

public interface *DefinedObjectS*

Object with defined type and implementation.. 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.

2.25 swarm.defobj.Drop

public interface *Drop*

Deallocate an object allocated within a zone.. 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.

2.25.1 Methods

*public void **drop()***

Immediate effects of the drop message depends on the subtype of Zone used to provide storage for the object. For some zone types, the drop message immediately deallocates storage for the object and makes the freed storage available for other use. Subsequent use could include the allocation of a new object at precisely the same location, resulting in a new object id identical to a previously dropped one. The Drop type may be inherited by any type that provides drop support for its instances. In addition to freeing the storage and invalidating the object, a drop message may release other resources acquired or held within the object. Not every object which can be created can also be dropped, and some objects can be dropped which are not directly creatable. Some objects may be created as a side effect of other operations and still be droppable, and some objects may be created with links to other objects and not droppable on their own. A type independently inherits Create or Drop types, or both, to indicate its support of these standard interfaces to define the endpoints of an object lifecycle.

2.26 swarm.defobj.DropC

public interface *DropC*

Deallocate an object allocated within a zone.. 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.

2.27 swarm.defobj.DropS

public interface *DropS*

Deallocate an object allocated within a zone.. 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.

2.28 swarm.defobj.Error

public interface *Error*
implements WarningS,Warning

A condition which prevents further execution.. A condition which prevents further execution.

2.29 swarm.defobj.ErrorC

public interface *ErrorC*
implements WarningS,WarningC
A condition which prevents further execution.. A condition which prevents further execution.

2.29.1 Implementing class: swarm.defobj.ErrorCImpl

java.lang.Object

swarm.PhaseCImpl

public *ErrorCImpl*
extends PhaseCImpl
implements WarningS,WarningC,ErrorS,ErrorC

2.29.2 Implementing class: swarm.defobj.ErrorImpl

java.lang.Object

public *ErrorImpl*
extends Object
implements WarningS,Warning,ErrorS,Error

2.30 swarm.defobj.ErrorS

public interface *ErrorS*
implements WarningS

A condition which prevents further execution.. A condition which prevents further execution.

2.31 swarm.defobj.EventType

public interface *EventType*
implements SymbolS,Symbol

A report of some condition detected during program execution.. A report of some condition detected during program execution.

2.31.1 Methods

*public void **raiseEvent()***

Raise an event noting the event symbol type.

2.32 swarm.defobj.EventTypeC

public interface *EventTypeC*
implements SymbolS,SymbolC

A report of some condition detected during program execution.. A report of some condition detected during program execution.

2.33 swarm.defobj.EventTypeS

public interface *EventTypeS*
implements SymbolS

A report of some condition detected during program execution.. A report of some condition detected during program execution.

2.34 swarm.defobj.FArguments

public interface *FArguments*
implements CreateS,Create,DropS,Drop

A language independent interface to dynamic call argument construction.. A language independent interface to dynamic call argument construction.

2.34.1 Methods

public boolean ***getJavaFlag****()*

2.35 swarm.defobj.FArgumentsC

public interface *FArgumentsC*
implements CreateS,CreateC,DropS,DropC

A language independent interface to dynamic call argument construction.. A language independent interface to dynamic call argument construction.

2.35.1 Methods

public Object ***addBoolean****(boolean value)*

public Object ***addChar****(char value)*

public Object ***addDouble****(double value)*

public Object ***addFloat****(float value)*

public Object ***addInt****(int value)*

public Object ***addJavaObject****(java.lang.Object obj)*

public Object ***addLong****(int value)*

public Object ***addLongLong****(long value)*

public Object ***addObject****(java.lang.Object obj)*

public Object ***addShort****(short value)*

public Object ***addString****(java.lang.String value)*

public Object ***addUnsigned****(int value)*

public Object ***addUnsignedChar****(byte value)*

public Object ***addUnsignedLong****(int value)* *public Object* ***addUnsignedLongLong****(long value)*

public Object ***addUnsignedShort****(short value)*

public Object ***create$setSelector$setJavaFlag****(swarm.defobj.Zone aZone, swarm.Selector aSel, boolean javaFlag)*

public Object ***createBegin****(swarm.defobj.Zone aZone)*

public Object ***createEnd****()*

public Object ***setBooleanReturnType****()*

public Object ***setJavaFlag****(boolean javaFlag)*

public Object ***setJavaSignature****(java.lang.String javaSignature)*

public Object ***setObjCReturnType****(char type)*

public Object ***setSelector****(swarm.Selector aSel)*

2.35.2 Implementing class: swarm.defobj.FArgumentsCImpl

java.lang.Object

swarm.PhaseCImpl

---

public *FArgumentsCImpl*
extends PhaseCImpl
implements CreateS,CreateC,DropS,DropC,FArgumentsS,FArgumentsC

2.35.3 Implementing class: swarm.defobj.FArgumentsImpl

java.lang.Object

---

public *FArgumentsImpl*
extends Object
implements CreateS,Create,DropS,Drop,FArgumentsS,FArguments

2.36 swarm.defobj.FArgumentsS

public interface *FArgumentsS*
implements CreateS,DropS

A language independent interface to dynamic call argument construction.. A language independent interface to dynamic call argument construction.

2.37 swarm.defobj.FCall

public interface *FCall*
implements CreateS,Create,DropS,Drop

A language independent interface to dynamic calls.. A language independent interface to dynamic calls.

2.37.1 Methods

public Object ***getArguments()***

public void ***performCall()***

2.38 swarm.defobj.FCallC

public interface *FCallC*
implements CreateS,CreateC,DropS,DropC
A language independent interface to dynamic calls.. A language independent interface to dynamic calls.

2.38.1 Methods

*public Object **create$target$selector$arguments**(swarm.defobj.Zone aZone, java.lang.Object obj, swarm.Selector aSel, swarm.defobj.FArguments fa)*

*public Object **createBegin**(swarm.defobj.Zone aZone)*

*public Object **createEnd**()*

*public Object **setArguments**(java.lang.Object args)*

*public Object **setJavaMethod$inClass**(java.lang.String methodName, java.lang.String className)*

*public Object **setJavaMethod$inObject**(java.lang.String methodName, java.lang.Object obj)*

*public Object **setMethod$inObject**(swarm.Selector method, java.lang.Object object)*

2.38.2 Implementing class: swarm.defobj.FCallCImpl

java.lang.Object

swarm.PhaseCImpl

public *FCallCImpl*
extends PhaseCImpl
implements CreateS,CreateC,DropS,DropC,FCallS,FCallC

2.38.3 Implementing class: swarm.defobj.FCallImpl

java.lang.Object

---

public *FCallImpl*
extends Object
implements CreateS,Create,DropS,Drop,FCallS,FCall

2.39 swarm.defobj.FCallS

public interface *FCallS*
implements CreateS,DropS

A language independent interface to dynamic calls.. A language independent interface to dynamic calls.

2.40 swarm.defobj.GetName

public interface *GetName*

Get name which identifies object in its context of use.. Get name which identifies object in its context of use.

2.40.1 Methods

*public String **getName()***

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

2.41 swarm.defobj.GetNameC

public interface *GetNameC*

Get name which identifies object in its context of use.. Get name which identifies object in its context of use.

2.42 swarm.defobj.GetNameS

public interface *GetNameS*

Get name which identifies object in its context of use.. Get name which identifies object in its context of use.

2.43 swarm.defobj.GetOwner

public interface *GetOwner*

Get object on which existence of object depends.. 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.

2.43.1 Methods

public Object ***getOwner()***

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

2.44 swarm.defobj.GetOwnerC

public interface *GetOwnerC*

Get object on which existence of object depends.. 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.

2.45 swarm.defobj.GetOwnerS

public interface *GetOwnerS*

Get object on which existence of object depends.. 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.

2.46 swarm.defobj.HDF5

public interface *HDF5*
implements CreateS,Create,DropS,Drop
HDF5 interface. HDF5 interface

2.46.1 Methods

public void ***assignIvar****(java.lang.Object obj)*

public boolean ***checkDatasetName****(java.lang.String datasetName)*

public boolean ***checkName****(java.lang.String name)*

public void ***drop****()*

public String ***getAttribute****(java.lang.String attributeName)*

public Object ***getCompoundType****()*

public int ***getCount****()*

public int ***getDatasetDimension****(int dimNumber)*

public boolean ***getDatasetFlag****()*

public int ***getDatasetRank****()*

public String ***getName****()*

public boolean ***getWriteFlag****()*

public void ***nameRecord$name****(int recordNumber, java.lang.String recordName)*

public void ***numberRecord****(int recordNumber)* *public void* ***selectRecord****(int recordNumber)*

public void ***shallowLoadObject****(java.lang.Object obj)*

public void ***shallowStoreObject****(java.lang.Object obj)*

public void ***storeAttribute$value****(java.lang.String attributeName, java.lang.String valueString)*

public void ***storeComponentTypeName****(java.lang.String typeName)*

public void ***storeTypeName****(java.lang.String typeName)*

public void ***writeLevels****()*

public void ***writeRowNames****()*

2.47 swarm.defobj.HDF5Archiver

public interface *HDF5Archiver*
implements ArchiverS,Archiver

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

2.48 swarm.defobj.HDF5ArchiverC

---

public interface *HDF5ArchiverC*
implements ArchiverS,ArchiverC

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

2.48.1 Methods

public Object ***create$setPath****(swarm.defobj.Zone aZone, java.lang.String path)*

Convenience method to create an HDF5Archiver from a specified path

2.48.2 Implementing class: swarm.defobj.HDF5ArchiverCImpl

java.lang.Object

swarm.PhaseCImpl

---

public *HDF5ArchiverCImpl*
extends PhaseCImpl
implements ArchiverS,ArchiverC,HDF5ArchiverS,HDF5ArchiverC

2.48.3 Implementing class: swarm.defobj.HDF5ArchiverImpl

java.lang.Object

---

public *HDF5ArchiverImpl*
extends Object
implements ArchiverS,Archiver,HDF5ArchiverS,HDF5Archiver

2.49 swarm.defobj.HDF5ArchiverS

public interface *HDF5ArchiverS*
implements ArchiverS

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

2.50 swarm.defobj.HDF5C

public interface *HDF5C*
implements CreateS,CreateC,DropS,DropC
HDF5 interface. HDF5 interface

2.50.1 Methods

public Object ***createBegin****(swarm.defobj.Zone aZone)*

public Object ***createEnd****()*

public Object ***setCompoundType****(java.lang.Object compoundType)*

public Object ***setCount****(int count)*

public Object ***setDatasetFlag****(boolean datasetFlag)*

public Object ***setParent****(java.lang.Object parent)*

public Object ***setWriteFlag****(boolean writeFlag)*

2.50.2 Implementing class: swarm.defobj.HDF5CImpl

java.lang.Object

swarm.PhaseCImpl

public *HDF5CImpl*
extends PhaseCImpl
implements CreateS,CreateC,DropS,DropC,HDF5S,HDF5C

2.50.3 Implementing class: swarm.defobj.HDF5Impl

java.lang.Object

public *HDF5Impl*
extends Object
implements CreateS,Create,DropS,Drop,HDF5S,HDF5

2.51 swarm.defobj.HDF5S

---

public interface *HDF5S*
implements CreateS,DropS

HDF5 interface. HDF5 interface

2.51.1 Methods

public Object ***setBaseTypeObject****(java.lang.Object baseTypeObject)*

public Object ***setName****(java.lang.String 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.

2.52 swarm.defobj.LispArchiver

---

public interface *LispArchiver*
implements ArchiverS,Archiver

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

2.53 swarm.defobj.LispArchiverC

public interface *LispArchiverC*
implements ArchiverS,ArchiverC

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

2.53.1 Methods

public Object ***create$setPath****(swarm.defobj.Zone aZone, java.lang.String path)*

Convenience method to create LispArchiver from a specified path

2.53.2 Implementing class: swarm.defobj.LispArchiverCImpl

java.lang.Object

swarm.PhaseCImpl

public *LispArchiverCImpl*
extends PhaseCImpl
implements ArchiverS,ArchiverC,LispArchiverS,LispArchiverC

2.53.3 Implementing class: swarm.defobj.LispArchiverImpl

java.lang.Object

public *LispArchiverImpl*
extends Object
implements ArchiverS,Archiver,LispArchiverS,LispArchiver

2.54 swarm.defobj.LispArchiverS

public interface *LispArchiverS*
implements ArchiverS

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

2.55 swarm.defobj.Serialization

public interface *Serialization*

Object serialization protocol.. Object serialization protocol.

2.55.1 Methods

public Object ***hdf5OutDeep****(swarm.defobj.HDF5 hdf5obj)*

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

public Object ***hdf5OutShallow****(swarm.defobj.HDF5 hdf5obj)*

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

public Object ***lispOutDeep****(java.lang.Object stream)*

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

public Object ***lispOutShallow****(java.lang.Object stream)*

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

public Object ***lispOutVars$deep****(java.lang.Object stream, boolean deepFlag)*

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

public Object ***updateArchiver****(java.lang.Object archiver)*

2.56 swarm.defobj.SerializationC

public interface *SerializationC*

Object serialization protocol.. Object serialization protocol.

2.56.1 Methods

public Object ***hdf5InCreate****(swarm.defobj.HDF5 hdf5Obj)*

Process HDF5 object to set create-time parameters.

public Object ***lispInCreate****(java.lang.Object expr)*

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

2.57 swarm.defobj.SerializationS

public interface *SerializationS*

Object serialization protocol.. Object serialization protocol.

2.57.1 Methods

*public Object **hdf5In**(swarm.defobj.HDF5 hdf5Obj)*

Load instance variables from an HDF5 object.

*public Object **lispIn**(java.lang.Object expr)*

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

2.58 swarm.defobj.SetInitialValue

public interface *SetInitialValue*

Create using initial value from an existing object.. 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.

2.59 swarm.defobj.SetInitialValueC

public interface *SetInitialValueC*

Create using initial value from an existing object.. 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.

2.59.1 Methods

public void ***setInitialValue****(java.lang.Object initialValue)*

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

2.60 swarm.defobj.SetInitialValueS

public interface *SetInitialValueS*

Create using initial value from an existing object.. 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.

2.61 swarm.defobj.Symbol

---

public interface *Symbol*
implements CreateS,Create,GetNameS,GetName

Object defined as a distinct global id constant.. 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.

2.62 swarm.defobj.SymbolC

public interface *SymbolC*
implements CreateS,CreateC,GetNameS,GetNameC

Object defined as a distinct global id constant.. 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.

2.62.1 Methods

*public Object **create$setName**(swarm.defobj.Zone aZone, java.lang.String name)*

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

2.62.2 Implementing class: swarm.defobj.SymbolCImpl

java.lang.Object

swarm.PhaseCImpl

public *SymbolCImpl*
extends PhaseCImpl
implements CreateS,CreateC,GetNameS,GetNameC,SymbolS,SymbolC

2.62.3 Implementing class: swarm.defobj.SymbolImpl

java.lang.Object

public *SymbolImpl*

extends Object
implements CreateS,Create,GetNameS,GetName,SymbolS,Symbol

2.63 swarm.defobj.SymbolS

public interface *SymbolS*
implements CreateS,GetNameS

Object defined as a distinct global id constant.. 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.

2.64 swarm.defobj.Warning

---

public interface *Warning*
implements EventTypeS,EventType

A condition of possible concern to a program developer.. A condition of possible concern to a program developer.

2.64.1 Methods

*public String **getMessageString**()*

Return the message associated with this warning.

*public void **setMessageString**(java.lang.String messageString)*

Associate a message string with this warning.

2.65 swarm.defobj.WarningC

public interface *WarningC*
implements EventTypeS,EventTypeC

A condition of possible concern to a program developer.. A condition of possible concern to a program developer.

2.65.1 Implementing class: swarm.defobj.WarningCImpl

java.lang.Object

swarm.PhaseCImpl

public *WarningCImpl*
extends PhaseCImpl
implements EventTypeS,EventTypeC,WarningS,WarningC

2.65.2 Implementing class: swarm.defobj.WarningImpl

java.lang.Object

public *WarningImpl*
extends Object
implements EventTypeS,EventType,WarningS,Warning

2.66 swarm.defobj.WarningS

public interface *WarningS*
implements EventTypeS

A condition of possible concern to a program developer.. A condition of possible concern to a program developer.

2.67 swarm.defobj.Zone

public interface *Zone*
implements CreateS,Create,DropS,Drop

Modular unit of storage allocation.. 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 DefinedObject type returns the zone which provides these base pages. Since a new zone always requires that an existing zone be identified, no new zones could be created unless there were some zones that already existed. Two such zones are predefined as part of the defobj library: globalZone and scratchZone.

2.67.1 Methods

public Object ***allocIVars****(java.lang.Class aClass)*

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

public Object ***allocIVarsComponent****(java.lang.Class aClass)*

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

public Object ***copyIVars****(java.lang.Object 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.

public Object ***copyIVarsComponent****(java.lang.Object anObject)*

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

public void ***describeForEach****(java.lang.Object outputCharStream)*

Generate debug description for each member of the zone population.

public void ***describeForEachID****(java.lang.Object outputCharStream)*

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

public void ***freeIVars****(java.lang.Object 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.

public void ***freeIVarsComponent****(java.lang.Object anObject)*

Frees the instance variable storage for an object.

public 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.

public int ***getPageSize()***

public List ***getPopulation()***

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

public Object ***getReclaimPolicy()***

public boolean ***getStackedSubzones()***

2.68 swarm.defobj.ZoneC

public interface *ZoneC*
implements CreateS,CreateC,DropS,DropC

Modular unit of storage allocation.. 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 DefinedObject type returns the zone which provides these base pages. Since a new zone always requires that an existing zone be identified, no new zones could be created unless there were some zones that already existed. Two such zones are predefined as part of the defobj library: globalZone and scratchZone.

2.68.1 Methods

public void ***setPageSize****(int pageSize)*

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

2.68.2 Implementing class: swarm.defobj.ZoneCImpl

java.lang.Object

swarm.PhaseCImpl

---

public *ZoneCImpl*
extends PhaseCImpl
implements CreateS,CreateC,DropS,DropC,ZoneS,ZoneC

2.68.3 Implementing class: swarm.defobj.ZoneImpl

java.lang.Object

---

public *ZoneImpl*
extends Object
implements CreateS,Create,DropS,Drop,ZoneS,Zone

2.69 swarm.defobj.ZoneS

public interface *ZoneS*
implements CreateS,DropS

Modular unit of storage allocation.. 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 DefinedObject type returns the zone which provides these base pages. Since a new zone always requires that an existing zone be identified, no new zones could be created unless there were some zones that already existed. Two such zones are predefined as part of the defobj library: globalZone and scratchZone.

Chapter 3

Package swarm.collections

Interfaces in package:

Classes in package:

- 3.38.1 MapCImpl (page 158)
- 3.38.2 MapImpl (page 158)
- 3.40.1 MapIndexCImpl (page 160)
- 3.40.2 MapIndexImpl (page 160)
- 3.53.2 PermutationCImpl (page 173)
- 3.53.3 PermutationImpl (page 173)
- 3.55.2 PermutationItemCImpl (page 175)
- 3.55.3 PermutationItemImpl (page 175)
- 3.59.2 PermutedIndexCImpl (page 179)
- 3.59.3 PermutedIndexImpl (page 179)
- 3.62.2 StringCImpl (page 182)
- 3.62.3 StringImpl (page 182)

3.1 swarm.collections.Array

public interface *Array*
implements CollectionS,Collection,DefaultMemberS,DefaultMember,MemberBlockS,MemberBlock

Collection supporting access only by relative position.. 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.

3.2 swarm.collections.ArrayC

public interface *ArrayC*
implements CollectionS,CollectionC,DefaultMemberS,DefaultMemberC,MemberBlockS,MemberBlockC

Collection supporting access only by relative position.. 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.

3.2.1 Methods

public Object ***create$setCount****(swarm.defobj.Zone aZone, int count)*

3.2.2 Implementing class: swarm.collections.ArrayCImpl

java.lang.Object

swarm.PhaseCImpl

---

public *ArrayCImpl*
extends PhaseCImpl
implements CollectionS,CollectionC,DefaultMemberS,DefaultMemberC,MemberBlockS,MemberBlockC,ArrayS,ArrayC

3.2.3 Implementing class: swarm.collections.ArrayImpl

java.lang.Object

---

public *ArrayImpl*
extends Object
implements CollectionS,Collection,DefaultMemberS,DefaultMember,MemberBlockS,MemberBlock,ArrayS,Array

3.3 swarm.collections.ArrayS

public interface *ArrayS*
implements CollectionS,DefaultMemberS,MemberBlockS

Collection supporting access only by relative position.. 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.

3.3.1 Methods

public Object ***setCount****(int count)*

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

3.4 swarm.collections.Collection

public interface *Collection*
implements CreateS,Create,SetInitialValueS,SetInitialValue,CopyS,Copy,DropS,Drop,OffsetsS,Offsets,ForEachS,ForEach

A generic collection interface.. 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.

3.4.1 Methods

public boolean ***allSameClass()***

Returns YES if all members are of the same class.

public Index ***begin****(swarm.defobj.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.

public PermutedIndex ***beginPermuted****(swarm.defobj.Zone aZone)*

public boolean ***contains****(java.lang.Object 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.

public Object ***copy****(swarm.defobj.Zone aZone)*

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

public void ***deleteAll()***

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

public int ***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.

public boolean ***getReplaceOnly()***

public Object ***remove****(java.lang.Object 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.

public 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.

3.5 swarm.collections.CollectionC

public interface *CollectionC*
implements CreateS,CreateC,SetInitialValueS,SetInitialValueC,CopyS,CopyC,DropS,DropC,OffsetsS,OffsetsC,ForEachS,ForEachC

A generic collection interface.. 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.

3.5.1 Methods

public void ***setIndexFromMemberLoc****(int byteOffset)*

public void ***setReplaceOnly****(boolean replaceOnly)*

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

3.6 swarm.collections.CollectionS

public interface *CollectionS*
implements CreateS,SetInitialValueS,CopyS,DropS,OffsetsS,ForEachS

A generic collection interface.. 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.

3.7 swarm.collections.CompareFunction

public interface *CompareFunction*

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

3.8 swarm.collections.CompareFunctionC

public interface *CompareFunctionC*

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

3.8.1 Methods

public Object ***setCompareCStrings()***

public Object ***setCompareIDs()***

public Object ***setCompareIntegers()***

public Object ***setCompareUnsignedIntegers()***

3.9 swarm.collections.CompareFunctionS

---

public interface *CompareFunctionS*

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

3.10 swarm.collections.DefaultMember

public interface *DefaultMember*

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

3.10.1 Methods

public Object ***getDefaultMember()***

3.11 swarm.collections.DefaultMemberC

public interface *DefaultMemberC*

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

3.12 swarm.collections.DefaultMemberS

public interface *DefaultMemberS*

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

3.12.1 Methods

public void ***setDefaultMember****(java.lang.Object memberValue)*

3.13 swarm.collections.ForEach

public interface *ForEach*

Messages for performing the same message on objects in a collection.. 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.

3.13.1 Methods

public void ***describeForEach****(java.lang.Object outputCharStream)*

public void ***describeForEachID****(java.lang.Object outputCharStream)*

public void ***forEach****(swarm.Selector aSelector)*

public void ***forEach****(swarm.Selector aSelector, java.lang.Object arg1)*

public void ***forEach****(swarm.Selector aSelector, java.lang.Object arg1, java.lang.Object arg2)*

public void ***forEach****(swarm.Selector aSelector, java.lang.Object arg1, java.lang.Object arg2, java.lang.Object arg3)*

3.14 swarm.collections.ForEachC

public interface *ForEachC*

Messages for performing the same message on objects in a collection.. 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.

3.15 swarm.collections.ForEachKey

public interface *ForEachKey*

Exactly the same as the ForEach protocol, but only for KeyedCollections.. Works identically to the ForEach protocol, but loops through the keys in a KeyedCollection, rather than the members.

3.15.1 Methods

*public void **forEachKey**(swarm.Selector aSelector)*

*public void **forEachKey**(swarm.Selector aSelector, java.lang.Object arg1)*

*public void **forEachKey**(swarm.Selector aSelector, java.lang.Object arg1, java.lang.Object arg2)*

*public void **forEachKey**(swarm.Selector aSelector, java.lang.Object arg1, java.lang.Object arg2, java.lang.Object arg3)*

3.16 swarm.collections.ForEachKeyC

public interface *ForEachKeyC*

Exactly the same as the ForEach protocol, but only for KeyedCollections.. Works identically to the ForEach protocol, but loops through the keys in a KeyedCollection, rather than the members.

3.17 swarm.collections.ForEachKeyS

public interface *ForEachKeyS*

Exactly the same as the ForEach protocol, but only for KeyedCollections.. Works identically to the ForEach protocol, but loops through the keys in a KeyedCollection, rather than the members.

3.18 swarm.collections.ForEachS

public interface *ForEachS*

Messages for performing the same message on objects in a collection.. 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.

3.19 swarm.collections.Index

public interface *Index*
implements DefinedObjectS,DefinedObject,DropS,Drop

Reference into the enumeration sequence for a collection.. 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.

3.19.1 Methods

public int ***compare****(java.lang.Object 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.

public Object ***findNext****(java.lang.Object 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.

public Object ***findPrev****(java.lang.Object 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.

public Object ***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.

public Object ***getCollection****()*

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

public 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.

public 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.

public Object ***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.

public Object ***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.

public Object ***put****(java.lang.Object 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.

public Object ***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.

public void ***setLoc****(swarm.defobj.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.

public Object ***setOffset****(int 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.

3.20 swarm.collections.IndexC

public interface *IndexC*
implements DefinedObjectS,DefinedObjectC,DropS,DropC

Reference into the enumeration sequence for a collection.. 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.

3.20.1 Implementing class: swarm.collections.IndexCImpl

java.lang.Object

swarm.PhaseCImpl

---

public *IndexCImpl*
extends PhaseCImpl
implements DefinedObjectS,DefinedObjectC,DropS,DropC,IndexS,IndexC

3.20.2 Implementing class: swarm.collections.IndexImpl

java.lang.Object

---

public *IndexImpl*
extends Object
implements DefinedObjectS,DefinedObject,DropS,Drop,IndexS,Index

3.21 swarm.collections.IndexS

public interface *IndexS*
implements DefinedObjectS,DropS

Reference into the enumeration sequence for a collection.. 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.

3.22 swarm.collections.KeyedCollection

public interface *KeyedCollection*
implements CollectionS,Collection,ForEachKeyS,ForEachKey

Member identity definition shared by Set and Map types.. 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.

3.22.1 Methods

public Object ***at****(java.lang.Object 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.

public boolean ***containsKey****(java.lang.Object aKey)*

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

public Object ***createIndex$fromMember****(swarm.defobj.Zone aZone, java.lang.Object anObject)*

public Object ***removeKey****(java.lang.Object aKey)*

The removeKey: message removes a member matching a key value from the collection, and returns the member just removed. It returns nil if there is no key value in the collection which matches. If more than one entry was present for the key value, it removes and returns the first member in the internal collection created for duplicate members.

3.23 swarm.collections.KeyedCollectionC

public interface *KeyedCollectionC*
implements CollectionS,CollectionC,ForEachKeyS,ForEachKeyC

Member identity definition shared by Set and Map types.. 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.

3.24 swarm.collections.KeyedCollectionIndex

---

public interface *KeyedCollectionIndex*
implements IndexS,Index

Index behavior shared by Set and Map types.. 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.

3.25 swarm.collections.KeyedCollectionIndexC

public interface *KeyedCollectionIndexC*
implements IndexS,IndexC

Index behavior shared by Set and Map types.. 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.

3.26 swarm.collections.KeyedCollectionIndexS

---

public interface *KeyedCollectionIndexS*
implements IndexS

Index behavior shared by Set and Map types.. 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.

3.27 swarm.collections.KeyedCollectionS

---

public interface *KeyedCollectionS*
implements CollectionS,ForEachKeyS

Member identity definition shared by Set and Map types.. 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.

3.28 swarm.collections.List

public interface *List*
implements CollectionS,Collection

Collection of members in an externally assigned linear sequence.. 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.

3.28.1 Methods

public void ***addFirst****(java.lang.Object anObject)*

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

public void ***addLast****(java.lang.Object anObject)*

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

public ListIndex ***listBegin****(swarm.defobj.Zone aZone)*

Returns a ListIndex, the special index for the List type

public Object ***removeFirst****()*

Removes the first member from the list and returns it.

public Object ***removeLast****()*

Removes the last member from the list and returns it.

3.29 swarm.collections.ListC

public interface *ListC*
implements CollectionS,CollectionC

Collection of members in an externally assigned linear sequence.. 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.

3.29.1 Implementing class: swarm.collections.ListCImpl

java.lang.Object

swarm.PhaseCImpl

public *ListCImpl*
extends PhaseCImpl
implements CollectionS,CollectionC,ListS,ListC

3.29.2 Implementing class: swarm.collections.ListImpl

java.lang.Object

public *ListImpl*
extends Object
implements CollectionS,Collection,ListS,List

3.30 swarm.collections.ListIndex

public interface *ListIndex*
implements IndexS,Index

Index with insertion capability at any point in list.. 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.

3.30.1 Methods

*public void **addAfter**(java.lang.Object anObject)*

Add a member after the index.

*public void **addBefore**(java.lang.Object anObject)*

Add a member before the index.

3.31 swarm.collections.ListIndexC

---

public interface *ListIndexC*
implements IndexS,IndexC

Index with insertion capability at any point in list.. 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.

3.31.1 Implementing class: swarm.collections.ListIndexCImpl

java.lang.Object

swarm.PhaseCImpl

---

public *ListIndexCImpl*
extends PhaseCImpl
implements IndexS,IndexC,ListIndexS,ListIndexC

3.31.2 Implementing class: swarm.collections.ListIndexImpl

java.lang.Object

---

public *ListIndexImpl*
extends Object
implements IndexS,Index,ListIndexS,ListIndex

3.32 swarm.collections.ListIndexS

public interface *ListIndexS*
implements IndexS

Index with insertion capability at any point in list.. 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.

3.33 swarm.collections.ListS

public interface *ListS*
implements CollectionS

Collection of members in an externally assigned linear sequence.. 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.

3.34 swarm.collections.ListShuffler

public interface *ListShuffler*
implements CreateS,Create,DropS,Drop

A class to randomize the order of a given Swarm List. 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.

3.34.1 Methods

public Object ***shufflePartialList$Num****(java.lang.Object list, int 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).

public Object ***shuffleWholeList****(java.lang.Object list)*

the shuffleWholeList method randomizes the whole list.

3.35 swarm.collections.ListShufflerC

public interface *ListShufflerC*
implements CreateS,CreateC,DropS,DropC

A class to randomize the order of a given Swarm List. 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.

3.35.1 Methods

*public Object **create$setUniformRandom**(swarm.defobj.Zone aZone, java.lang.Object dist)*

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

*public Object **createEnd**()*

*public Object **setUniformRandom**(java.lang.Object dist)*

the setUniformRandom: method connects the supplied uniform distribution to the Shuffler (run after createBegin:).

3.35.2 Implementing class: swarm.collections.ListShufflerCImpl

java.lang.Object

swarm.PhaseCImpl

public *ListShufflerCImpl*
extends PhaseCImpl
implements CreateS,CreateC,DropS,DropC,ListShufflerS,ListShufflerC

3.35.3 Implementing class: swarm.collections.ListShufflerImpl

java.lang.Object

public *ListShufflerImpl*
extends Object
implements CreateS,Create,DropS,Drop,ListShufflerS,ListShuffler

3.36 swarm.collections.ListShufflerS

public interface *ListShufflerS*
implements CreateS,DropS

A class to randomize the order of a given Swarm List. 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.

3.37 swarm.collections.Map

public interface *Map*
implements KeyedCollectionS,KeyedCollection,CompareFunctionS,CompareFunction

Collection of associations from key objects to member objects.. 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.

3.37.1 Methods

public boolean ***at$insert****(java.lang.Object aKey, java.lang.Object anObject)*

at:insert: inserts an entry into a Map containing the key and member values given as its arguments. It returns true if the key was not previously contained in the collection. An attempt to insert a duplicate key is simply rejected and false is returned.

public Object ***at$replace****(java.lang.Object aKey, java.lang.Object 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.

public MapIndex ***mapBegin****(swarm.defobj.Zone aZone)*

Returns a MapIndex, the special index for the Map type

3.38 swarm.collections.MapC

public interface *MapC*
implements KeyedCollectionS,KeyedCollectionC,CompareFunctionS,CompareFunctionC

Collection of associations from key objects to member objects.. 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.

3.38.1 Implementing class: swarm.collections.MapCImpl

java.lang.Object

swarm.PhaseCImpl

public *MapCImpl*
extends PhaseCImpl
implements KeyedCollectionS,KeyedCollectionC,CompareFunctionS,CompareFunctionC,MapS,MapC

3.38.2 Implementing class: swarm.collections.MapImpl

java.lang.Object

public *MapImpl*
extends Object
implements KeyedCollectionS,KeyedCollection,CompareFunctionS,CompareFunction,MapS,Map

3.39 swarm.collections.MapIndex

public interface *MapIndex*
implements KeyedCollectionIndexS,KeyedCollectionIndex

The index behavior for a Map.. The index behavior for a Map.

3.39.1 Methods

public Object ***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.

public Object ***setKey****(java.lang.Object 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.

3.40 swarm.collections.MapIndexC

---

public interface *MapIndexC*
implements KeyedCollectionIndexS,KeyedCollectionIndexC
The index behavior for a Map.. The index behavior for a Map.

3.40.1 Implementing class: swarm.collections.MapIndexCImpl

java.lang.Object

swarm.PhaseCImpl

---

public *MapIndexCImpl*
extends PhaseCImpl
implements KeyedCollectionIndexS,KeyedCollectionIndexC,MapIndexS,MapIndexC

3.40.2 Implementing class: swarm.collections.MapIndexImpl

java.lang.Object

---

public *MapIndexImpl*
extends Object
implements KeyedCollectionIndexS,KeyedCollectionIndex,MapIndexS,MapIndex

3.41 swarm.collections.MapIndexS

public interface *MapIndexS*
implements KeyedCollectionIndexS

The index behavior for a Map.. The index behavior for a Map.

3.42 swarm.collections.MapS

public interface *MapS*
implements KeyedCollectionS,CompareFunctionS

Collection of associations from key objects to member objects.. 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.

3.43 swarm.collections.MemberBlock

public interface *MemberBlock*

A way to wrap an existing C array for access as an object collection.. 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.

3.44 swarm.collections.MemberBlockC

public interface *MemberBlockC*

A way to wrap an existing C array for access as an object collection.. 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.

3.45 swarm.collections.MemberBlockS

public interface *MemberBlockS*

A way to wrap an existing C array for access as an object collection.. 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.

3.46 swarm.collections.MemberSlot

public interface *MemberSlot*

Allocation in member/key for fast setMember:/setKey:. 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.

3.47 swarm.collections.MemberSlotC

public interface *MemberSlotC*

Allocation in member/key for fast setMember:/setKey:. 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.

3.48 swarm.collections.MemberSlotS

public interface *MemberSlotS*

Allocation in member/key for fast setMember:/setKey:. 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.

3.49 swarm.collections.Offsets

public interface *Offsets*

Methods for accessing collection members by position.. 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.

3.49.1 Methods

public Object ***atOffset****(int offset)*

Returns the member at a particular member offset.

public Object ***atOffset$put****(int offset, java.lang.Object 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.

public Object ***getFirst****()*

Equivalent to [aCollection atOffset: 0].

public Object ***getLast****()*

Equivalent to [aCollection atOffset: [aCollection getCount] - 1].

3.50 swarm.collections.OffsetsC

public interface *OffsetsC*

Methods for accessing collection members by position.. 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.

3.51 swarm.collections.OffsetsS

public interface *OffsetsS*

Methods for accessing collection members by position.. 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.

3.52 swarm.collections.Permutation

public interface *Permutation*
implements CollectionS,Collection,CreateS,Create,ArrayS,Array

A class that represents a permutation of elements of a collection. 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.

3.53 swarm.collections.PermutationC

---

public interface *PermutationC*
implements CollectionS,CollectionC,CreateS,CreateC,ArrayS,ArrayC

A class that represents a permutation of elements of a collection. 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.

3.53.1 Methods

*public Object **createBegin**(swarm.defobj.Zone aZone)*

*public Object **setCollection**(java.lang.Object collection)*

*public Object **setUniformRandom**(java.lang.Object rnd)*

3.53.2 Implementing class: swarm.collections.PermutationCImpl

java.lang.Object

swarm.PhaseCImpl

---

public *PermutationCImpl*
extends PhaseCImpl
implements CollectionS,CollectionC,CreateS,CreateC,ArrayS,ArrayC,PermutationS,PermutationC

3.53.3 Implementing class: swarm.collections.PermutationImpl

java.lang.Object

---

public *PermutationImpl*
extends Object
implements CollectionS,Collection,CreateS,Create,ArrayS,Array,PermutationS,Permutation

3.54 swarm.collections.PermutationItem

public interface *PermutationItem*
implements CreateS,Create

An element of a Permutation. An element of a Permutation

3.54.1 Methods

*public Object **getItem()***

*public int **getPosition()***

3.55 swarm.collections.PermutationItemC

---

public interface *PermutationItemC*
implements CreateS,CreateC
An element of a Permutation. An element of a Permutation

3.55.1 Methods

*public Object **setItem**(java.lang.Object item)*

*public Object **setPosition**(int position)*

3.55.2 Implementing class: swarm.collections.PermutationItemCImpl

java.lang.Object

swarm.PhaseCImpl

---

public *PermutationItemCImpl*
extends PhaseCImpl
implements CreateS,CreateC,PermutationItemS,PermutationItemC

3.55.3 Implementing class: swarm.collections.PermutationItemImpl

java.lang.Object

---

public *PermutationItemImpl*
extends Object
implements CreateS,Create,PermutationItemS,PermutationItem

3.56 swarm.collections.PermutationItemS

public interface *PermutationItemS*
implements CreateS

An element of a Permutation. An element of a Permutation

3.57 swarm.collections.PermutationS

public interface *PermutationS*
implements CollectionS,CreateS,ArrayS

A class that represents a permutation of elements of a collection. 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.

3.58 swarm.collections.PermutedIndex

---

public interface *PermutedIndex*
implements IndexS,Index

General PermutedIndex class. . 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.

3.58.1 Methods

public Object ***findNext****(java.lang.Object anObject)*

public Object ***findPrev****(java.lang.Object anObject)*

public Object ***get****()*

public Symbol ***getLoc****()*

public int ***getOffset****()*

public Object ***next****()*

public Object ***prev****()*

public Object ***reshuffle****()*

public void ***setLoc****(swarm.defobj.Symbol locSymbol)*

public Object ***setOffset****(int offset)*

3.59 swarm.collections.PermutedIndexC

public interface *PermutedIndexC*
implements IndexS,IndexC

General PermutedIndex class. . 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.

3.59.1 Methods

public Object ***createBegin****(swarm.defobj.Zone aZone)*

public Object ***createEnd****()*

public Object ***setCollection****(java.lang.Object aCollection)*

public Object ***setUniformRandom****(java.lang.Object rnd)*

3.59.2 Implementing class: swarm.collections.PermutedIndexCImpl

java.lang.Object

swarm.PhaseCImpl

public *PermutedIndexCImpl*
extends PhaseCImpl
implements IndexS,IndexC,PermutedIndexS,PermutedIndexC

3.59.3 Implementing class: swarm.collections.PermutedIndexImpl

java.lang.Object

public *PermutedIndexImpl*
extends Object
implements IndexS,Index,PermutedIndexS,PermutedIndex

3.60 swarm.collections.PermutedIndexS

public interface *PermutedIndexS*
implements IndexS

General PermutedIndex class. . 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.

3.61 swarm.collections.String

public interface *String*
implements CreateS,Create,DropS,Drop,CopyS,Copy

Character string object (later to support collection behavior).. 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.

3.61.1 Methods

*public void **catC**(java.lang.String cstring)*

*public int **compare**(java.lang.Object aString)*

*public String **getC**()*

*public int **getCount**()*

3.62 swarm.collections.StringC

public interface *StringC*
implements CreateS,CreateC,DropS,DropC,CopyS,CopyC

Character string object (later to support collection behavior).. 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.

3.62.1 Methods

public Object ***create$setC****(swarm.defobj.Zone aZone, java.lang.String cstring)*

3.62.2 Implementing class: swarm.collections.StringCImpl

java.lang.Object

swarm.PhaseCImpl

public *StringCImpl*
extends PhaseCImpl
implements CreateS,CreateC,DropS,DropC,CopyS,CopyC,StringS,StringC

3.62.3 Implementing class: swarm.collections.StringImpl

java.lang.Object

public *StringImpl*
extends Object
implements CreateS,Create,DropS,Drop,CopyS,Copy,StringS,String

3.63 swarm.collections.StringS

public interface *StringS*
implements CreateS,DropS,CopyS

Character string object (later to support collection behavior).. 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.

3.63.1 Methods

public void ***setC****(java.lang.String cstring)*

Chapter 4

Package swarm.activity

Interfaces in package:

Classes in package:

- 4.44.2 ActivityImpl (page 236)
- 4.53.1 ConcurrentGroupCImpl (page 245)
- 4.53.2 ConcurrentGroupImpl (page 245)
- 4.59.1 ConcurrentScheduleCImpl (page 251)
- 4.59.2 ConcurrentScheduleImpl (page 251)
- 4.65.2 FActionCImpl (page 257)
- 4.78.1 FActionForEachHeterogeneousCImpl (page 270)
- 4.78.2 FActionForEachHeterogeneousImpl (page 270)
- 4.81.1 FActionForEachHomogeneousCImpl (page 273)
- 4.81.2 FActionForEachHomogeneousImpl (page 273)
- 4.83.1 FActionImpl (page 275)
- 4.86.1 ForEachActivityCImpl (page 278)
- 4.86.2 ForEachActivityImpl (page 278)
- 4.99.1 ScheduleActivityCImpl (page 292)
- 4.99.2 ScheduleActivityImpl (page 292)
- 4.101.2 ScheduleCImpl (page 294)
- 4.101.3 ScheduleImpl (page 295)
- 4.107.1 SwarmActivityCImpl (page 301)
- 4.107.2 SwarmActivityImpl (page 301)

4.1 swarm.activity.Action

public interface *Action*
implements CreateS,Create,DropS,Drop,GetOwnerS,GetOwner

An action type that has been customized for direct execution by an action interpreter.. 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.

4.1.1 Methods

*public void **_performAction_**(swarm.activity.Activity activity)*

4.2 swarm.activity.ActionArgs

public interface *ActionArgs*

Supertype of ActionCall, ActionTo, and ActionForEach.. 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.

4.2.1 Methods

public Object ***getArg1()***

public Object ***getArg2()***

public Object ***getArg3()***

public int ***getNArgs()***

4.3 swarm.activity.ActionArgsC

public interface *ActionArgsC*

Supertype of ActionCall, ActionTo, and ActionForEach.. 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.

4.3.1 Methods

public void ***setArg1****(java.lang.Object arg1)*

public void ***setArg2****(java.lang.Object arg2)*

public void ***setArg3****(java.lang.Object arg3)*

4.4 swarm.activity.ActionArgsS

public interface *ActionArgsS*

Supertype of ActionCall, ActionTo, and ActionForEach.. 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.

4.5 swarm.activity.ActionC

public interface *ActionC*
implements CreateS,CreateC,DropS,DropC,GetOwnerS,GetOwnerC

An action type that has been customized for direct execution by an action interpreter.. 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.

4.6 swarm.activity.ActionCall

public interface *ActionCall*
implements ActionS,Action,ActionArgsS,ActionArgs

An action defined by calling a C function.. An action defined by calling a C function.

4.7 swarm.activity.ActionCallC

public interface *ActionCallC*
implements ActionS,ActionC,ActionArgsS,ActionArgsC

An action defined by calling a C function.. An action defined by calling a C function.

4.7.1 Implementing class: swarm.activity.ActionCallCImpl

java.lang.Object

swarm.PhaseCImpl

public *ActionCallCImpl*
extends PhaseCImpl
implements ActionS,ActionC,ActionArgsS,ActionArgsC,ActionCallS,ActionCallC

4.7.2 Implementing class: swarm.activity.ActionCallImpl

java.lang.Object

public *ActionCallImpl*
extends Object
implements ActionS,Action,ActionArgsS,ActionArgs,ActionCallS,ActionCall

4.8 swarm.activity.ActionCallS

public interface *ActionCallS*
implements ActionS,ActionArgsS

An action defined by calling a C function.. An action defined by calling a C function.

4.9 swarm.activity.ActionChanged

public interface *ActionChanged*
implements ActionS,Action

An action generated when actions changes from single to concurrent.. An action generated when actions changes from single to concurrent.

4.10 swarm.activity.ActionChangedC

---

public interface *ActionChangedC*
implements ActionS,ActionC

An action generated when actions changes from single to concurrent.. An action generated when actions changes from single to concurrent.

4.10.1 Implementing class: swarm.activity.ActionChangedCImpl

java.lang.Object

swarm.PhaseCImpl

---

public *ActionChangedCImpl*
extends PhaseCImpl
implements ActionS,ActionC,ActionChangedS,ActionChangedC

4.10.2 Implementing class: swarm.activity.ActionChangedImpl

java.lang.Object

---

public *ActionChangedImpl*
extends Object
implements ActionS,Action,ActionChangedS,ActionChanged

4.11 swarm.activity.ActionChangedS

public interface *ActionChangedS*
implements ActionS

An action generated when actions changes from single to concurrent.. An action generated when actions changes from single to concurrent.

4.11.1 Implementing class: swarm.activity.ActionCImpl

java.lang.Object

swarm.PhaseCImpl

public *ActionCImpl*
extends PhaseCImpl
implements CreateS,CreateC,DropS,DropC,GetOwnerS,GetOwnerC,ActionS,ActionC

4.12 swarm.activity.ActionCreating

public interface *ActionCreating*
implements FActionCreatingS,FActionCreating,ActionCreatingCallS,ActionCreatingCall,ActionCreatingToS,ActionCreatingTo,ActionCreatingForEachS,ActionCreatingForEach,FActionCreatingForEachHeterogeneousS,FActionCreatingForEachHeterogeneous,FActionCreatingForEachHomogeneousS,FActionCreatingForEachHomogeneous

Protocol shared by ActionGroup and Schedule.. 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.)

4.12.1 Methods

public Object ***createAction****(java.lang.Object anActionType)*

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

4.13 swarm.activity.ActionCreatingC

public interface *ActionCreatingC*
implements FActionCreatingS,FActionCreatingC,ActionCreatingCallS,ActionCreatingCallC,ActionCreatingToS,ActionCreatingToC,ActionCreatingForEachS,ActionCreatingForEachC,FActionCreatingForEachHeterogeneousS,FActionCreatingForEachHeterogeneousC,FActionCreatingForEachHomogeneousS,FActionCreatingForEachHomogeneousC

Protocol shared by ActionGroup and Schedule.. 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.)

4.14 swarm.activity.ActionCreatingCall

public interface *ActionCreatingCall*

An action that calls a C function.. 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.

4.15 swarm.activity.ActionCreatingCallC

public interface *ActionCreatingCallC*

An action that calls a C function.. 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.

4.16 swarm.activity.ActionCreatingCallS

public interface *ActionCreatingCallS*

An action that calls a C function.. 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.

4.17 swarm.activity.ActionCreatingForEach

public interface *ActionCreatingForEach*

Send a message to every item in target, which is assumed to be a collection.. 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.

4.17.1 Methods

public ActionForEach ***createActionForEach$message****(java.lang.Object target, swarm.Selector aSel)*

public ActionForEach ***createActionForEach$message****(java.lang.Object target, swarm.Selector aSel, java.lang.Object arg1)*

public ActionForEach ***createActionForEach$message****(java.lang.Object target, swarm.Selector aSel, java.lang.Object arg1, java.lang.Object arg2)*

public ActionForEach ***createActionForEach$message****(java.lang.Object target, swarm.Selector aSel, java.lang.Object arg1, java.lang.Object arg2, java.lang.Object arg3)*

4.18 swarm.activity.ActionCreatingForEachC

public interface *ActionCreatingForEachC*

Send a message to every item in target, which is assumed to be a collection.. 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.

4.19 swarm.activity.ActionCreatingForEachS

public interface *ActionCreatingForEachS*

Send a message to every item in target, which is assumed to be a collection.. 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.

4.20 swarm.activity.ActionCreatingS

---

public interface *ActionCreatingS*
implements FActionCreatingS,ActionCreatingCallS,ActionCreatingToS,ActionCreatingForEachS,FActionCreatingForEachHeterogeneousS,FActionCreatingForEachHomogeneousS

Protocol shared by ActionGroup and Schedule.. 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.)

4.21 swarm.activity.ActionCreatingTo

public interface *ActionCreatingTo*

An action that sends a message to an object.. 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.

4.21.1 Methods

public ActionTo ***createActionTo$message****(java.lang.Object target, swarm.Selector aSel)*

public ActionTo ***createActionTo$message****(java.lang.Object target, swarm.Selector aSel, java.lang.Object arg1)*

public ActionTo ***createActionTo$message****(java.lang.Object target, swarm.Selector aSel, java.lang.Object arg1, java.lang.Object arg2)*

public ActionTo ***createActionTo$message****(java.lang.Object target, swarm.Selector aSel, java.lang.Object arg1, java.lang.Object arg2, java.lang.Object arg3)*

4.22 swarm.activity.ActionCreatingToC

public interface *ActionCreatingToC*

An action that sends a message to an object.. 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.

4.23 swarm.activity.ActionCreatingToS

public interface *ActionCreatingToS*

An action that sends a message to an object.. 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.

4.24 swarm.activity.ActionForEach

---

public interface *ActionForEach*
implements ActionToS,ActionTo,DefaultOrderS,DefaultOrder

An action defined by sending a message to every member of a collection.. An action defined by sending a message to every member of a collection.

4.25 swarm.activity.ActionForEachC

public interface *ActionForEachC*
implements ActionToS,ActionToC,DefaultOrderS,DefaultOrderC
An action defined by sending a message to every member of a collection..
An action defined by sending a message to every member of a collection.

4.25.1 Implementing class: swarm.activity.ActionForEachCImpl

java.lang.Object

swarm.PhaseCImpl

public *ActionForEachCImpl*
extends PhaseCImpl
implements ActionToS,ActionToC,DefaultOrderS,DefaultOrderC,ActionForEachS,ActionForEachC

4.25.2 Implementing class: swarm.activity.ActionForEachImpl

java.lang.Object

public *ActionForEachImpl*
extends Object
implements ActionToS,ActionTo,DefaultOrderS,DefaultOrder,ActionForEachS,ActionForEach

4.26 swarm.activity.ActionForEachS

public interface *ActionForEachS*
implements ActionToS,DefaultOrderS

An action defined by sending a message to every member of a collection.. An action defined by sending a message to every member of a collection.

4.27 swarm.activity.ActionGroup

public interface *ActionGroup*
implements CompoundActionS,CompoundAction,ActionCreatingS,ActionCreating,DefaultOrderS,DefaultOrder

A collection of actions under total or partial order constraints.. 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.

4.28 swarm.activity.ActionGroupC

public interface *ActionGroupC*
implements CompoundActionS,CompoundActionC,ActionCreatingS,ActionCreatingC,DefaultOrderS,DefaultOrderC

A collection of actions under total or partial order constraints.. 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.

4.28.1 Implementing class: swarm.activity.ActionGroupCImpl

java.lang.Object

swarm.PhaseCImpl

public *ActionGroupCImpl*
extends PhaseCImpl
implements CompoundActionS,CompoundActionC,ActionCreatingS,ActionCreatingC,DefaultOrderS,DefaultOrderC,ActionGroupS,ActionGroupC

4.28.2 Implementing class: swarm.activity.ActionGroupImpl

java.lang.Object

public *ActionGroupImpl*
extends Object
implements CompoundActionS,CompoundAction,ActionCreatingS,ActionCreating,DefaultOrderS,DefaultOrder,ActionGroupS,ActionGroup

4.29 swarm.activity.ActionGroupS

---

public interface *ActionGroupS*
implements CompoundActionS,ActionCreatingS,DefaultOrderS

A collection of actions under total or partial order constraints.. 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.

4.29.1 Implementing class: swarm.activity.ActionImpl

java.lang.Object

---

public *ActionImpl*
extends Object
implements CreateS,Create,DropS,Drop,GetOwnerS,GetOwner,ActionS,Action

4.30 swarm.activity.ActionS

public interface *ActionS*
implements CreateS,DropS,GetOwnerS

An action type that has been customized for direct execution by an action interpreter.. 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.

4.31 swarm.activity.ActionTarget

public interface *ActionTarget*

Messages common to actions that are sent to an object.. Messages common to actions that are sent to an object.

4.31.1 Methods

*public Object **getTarget()***

4.32 swarm.activity.ActionTargetC

public interface *ActionTargetC*

Messages common to actions that are sent to an object.. Messages common to actions that are sent to an object.

4.32.1 Methods

*public void **setTarget**(java.lang.Object target)*

4.33 swarm.activity.ActionTargetS

public interface *ActionTargetS*

Messages common to actions that are sent to an object.. Messages common to actions that are sent to an object.

4.34 swarm.activity.ActionTo

public interface *ActionTo*
implements ActionS,Action,ActionTargetS,ActionTarget,ActionArgsS,ActionArgs

An action defined by sending an Objective C message.. An action defined by sending an Objective C message.

4.35 swarm.activity.ActionToC

public interface *ActionToC*
implements ActionS,ActionC,ActionTargetS,ActionTargetC,ActionArgsS,ActionArgsC

An action defined by sending an Objective C message.. An action defined by sending an Objective C message.

4.35.1 Methods

*public void **setMessageSelector**(swarm.Selector aSel)*

4.35.2 Implementing class: swarm.activity.ActionToCImpl

java.lang.Object

swarm.PhaseCImpl

public *ActionToCImpl*
extends PhaseCImpl
implements ActionS,ActionC,ActionTargetS,ActionTargetC,ActionArgsS,ActionArgsC,ActionToS,ActionToC

4.35.3 Implementing class: swarm.activity.ActionToImpl

java.lang.Object

public *ActionToImpl*
extends Object
implements ActionS,Action,ActionTargetS,ActionTarget,ActionArgsS,ActionArgs,ActionToS,ActionTo

4.36 swarm.activity.ActionToS

public interface *ActionToS*
implements ActionS,ActionTargetS,ActionArgsS

An action defined by sending an Objective C message.. An action defined by sending an Objective C message.

4.37 swarm.activity.ActionType

public interface *ActionType*

Specification of an executable process.. 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. ..)

4.37.1 Methods

public Activity ***activateIn****(swarm.objectbase.Swarm swarmContext)*

The activateIn: message is used to initialize a process for executing the actions of an ActionType. This process is controlled by an object called an Activity. The activateIn message initializes an activity to run under the execution context passed as the swarmContext argument, and return the activity object just created. If the execution context is nil, an activity is returned that allows complete execution control by the caller. Otherwise, the execution context must be either an instance of SwarmProcess or SwarmActivity. (These objects are always maintained in one-to-one association with each other, either one of the pair is equivalent to the other as a swarmContext argument.) If a top-level activity is created (swarmContext is nil), the created activity may be processed using activity processing commands such as run, step, etc. If an activity is created to run under a swarm context, the swarm itself has responsibility for advancing the subactivity according to its requirements for synchronization and control among all its activities. Activating a plan for execution under a swarm turns over control to the swarm to execute the subactivity as a more-or-less autonomous activity.

4.38 swarm.activity.ActionTypeC

public interface *ActionTypeC*

Specification of an executable process.. 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. ..)

4.39 swarm.activity.ActionTypeS

public interface *ActionTypeS*

Specification of an executable process.. 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. ..)

4.40 swarm.activity.ActivationOrder

---

public interface *ActivationOrder*
implements ActionGroupS,ActionGroup

Default type used as concurrent group of a swarm.. Concurrent group to order merge by activation order within swarm.

4.40.1 Methods

public void ***addLast****(java.lang.Object mergeAction)*

Method to sort concurrent merge actions in the order of swarm activation.

public Object ***remove****(java.lang.Object mergeAction)*

Method to remove concurrent merge action from sorted group

4.41 swarm.activity.ActivationOrderC

public interface *ActivationOrderC*
implements ActionGroupS,ActionGroupC
Default type used as concurrent group of a swarm.. Concurrent group to order merge by activation order within swarm.

4.41.1 Implementing class: swarm.activity.ActivationOrderCImpl

java.lang.Object

swarm.PhaseCImpl

public *ActivationOrderCImpl*
extends PhaseCImpl
implements ActionGroupS,ActionGroupC,ActivationOrderS,ActivationOrderC

4.41.2 Implementing class: swarm.activity.ActivationOrderImpl

java.lang.Object

public *ActivationOrderImpl*
extends Object
implements ActionGroupS,ActionGroup,ActivationOrderS,ActivationOrder

4.42 swarm.activity.ActivationOrderS

public interface *ActivationOrderS*
implements ActionGroupS

Default type used as concurrent group of a swarm.. Concurrent group to order merge by activation order within swarm.

4.43 swarm.activity.Activity

public interface *Activity*
implements DefinedObjectS,DefinedObject,DropS,Drop

A level of processing by the interpreter of an action type.. 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.

4.43.1 Methods

public Action ***getAction()***

Get action containing parameter bindings for the local activity.

public Object ***getActionType()***

Get action type of action being performed by activity.

public Activity ***getControllingActivity()***

Return activity that issued current run request on top-level activity.

public Activity ***getCurrentSubactivity()***

Get running subactivity or next subactivity to run.

public 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.)

public Activity ***getOwnerActivity()***

Return activity under which this activity is running.

public ScheduleActivity ***getScheduleActivity()***

Return most immediately containing Schedule activity.

public boolean ***getSerialMode()***

Return indicator for serial execution mode.

public 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.

public Object ***getSubactivities()***

Return set of subactivities pending to be run.

public SwarmActivity ***getSwarmActivity()***

Return most immediately containing Swarm activity.

public Activity ***getTopLevelActivity()***

Return top of activity tree running the local activity.

public 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.

public Symbol ***run****()*

The run message continue processing of an activity from any state in which its execution has been suspended. An error is raised if the activity is already running. run returns either a Stopped or Completed status that the activity has when it is no longer eligible to be run further.

public void ***setOwnerActivity****(java.lang.Object ownerActivity)*

Change owner from one swarm activity to another.

public Object ***setSerialMode****(boolean serialMode)*

Set serial execution mode.

public Symbol ***step****()*

The step message executes a single action within a tree of activities.

public Object ***stop****()*

The stop message causes the a currently running tree of activities to stop further processing and return a Stopped status.

public 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.

4.44 swarm.activity.ActivityC

public interface *ActivityC*
implements DefinedObjectS,DefinedObjectC,DropS,DropC

A level of processing by the interpreter of an action type.. 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.

4.44.1 Implementing class: swarm.activity.ActivityCImpl

java.lang.Object

swarm.PhaseCImpl

public *ActivityCImpl*
extends PhaseCImpl
implements DefinedObjectS,DefinedObjectC,DropS,DropC,ActivityS,ActivityC

4.44.2 Implementing class: swarm.activity.ActivityImpl

java.lang.Object

---

public *ActivityImpl*
extends Object
implements DefinedObjectS,DefinedObject,DropS,Drop,ActivityS,Activity

4.45 swarm.activity.ActivityS

public interface *ActivityS*
implements DefinedObjectS,DropS

A level of processing by the interpreter of an action type.. 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.

4.46 swarm.activity.AutoDrop

public interface *AutoDrop*

Specify that an action is dropped after being processed.. 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.

4.46.1 Methods

public boolean ***getAutoDrop()***

4.47 swarm.activity.AutoDropC

public interface *AutoDropC*

Specify that an action is dropped after being processed.. 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.

4.47.1 Methods

public Object ***setAutoDrop****(boolean autoDrop)*

4.48 swarm.activity.AutoDropS

public interface *AutoDropS*

Specify that an action is dropped after being processed.. 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.

4.49 swarm.activity.CompoundAction

---

public interface *CompoundAction*
implements ActionTypeS,ActionType,CollectionS,Collection,AutoDropS,AutoDrop,DefaultOrderS,DefaultOrder

A collection of actions to be performed in any order consistent with a set of ordering constraints.. 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.

4.50 swarm.activity.CompoundActionC

public interface *CompoundActionC*
implements ActionTypeS,ActionTypeC,CollectionS,CollectionC,AutoDropS,AutoDropC,DefaultOrderS,DefaultOrderC

A collection of actions to be performed in any order consistent with a set of ordering constraints.. 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.

4.51 swarm.activity.CompoundActionS

public interface *CompoundActionS*
implements ActionTypeS,CollectionS,AutoDropS,DefaultOrderS

A collection of actions to be performed in any order consistent with a set of ordering constraints.. 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.

4.52 swarm.activity.ConcurrentGroup

public interface *ConcurrentGroup*
implements ActionGroupS,ActionGroup

Default type used as concurrent group of a schedule.. Default type used as concurrent group of a schedule.

4.53 swarm.activity.ConcurrentGroupC

public interface *ConcurrentGroupC*
implements ActionGroupS,ActionGroupC

Default type used as concurrent group of a schedule.. Default type used as concurrent group of a schedule.

4.53.1 Implementing class: swarm.activity.ConcurrentGroupCImpl

java.lang.Object

swarm.PhaseCImpl

public *ConcurrentGroupCImpl*
extends PhaseCImpl
implements ActionGroupS,ActionGroupC,ConcurrentGroupS,ConcurrentGroupC

4.53.2 Implementing class: swarm.activity.ConcurrentGroupImpl

java.lang.Object

public *ConcurrentGroupImpl*
extends Object
implements ActionGroupS,ActionGroup,ConcurrentGroupS,ConcurrentGroup

4.54 swarm.activity.ConcurrentGroupS

---

public interface *ConcurrentGroupS*
implements ActionGroupS

Default type used as concurrent group of a schedule.. Default type used as concurrent group of a schedule.

4.55 swarm.activity.ConcurrentGroupType

public interface *ConcurrentGroupType*

Handle actions scheduled at same time value.. 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.)

4.55.1 Methods

public Object ***getConcurrentGroupType****()*

4.56 swarm.activity.ConcurrentGroupTypeC

public interface *ConcurrentGroupTypeC*

Handle actions scheduled at same time value.. 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.)

4.57 swarm.activity.ConcurrentGroupTypeS

public interface *ConcurrentGroupTypeS*

Handle actions scheduled at same time value.. 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.)

4.57.1 Methods

public void ***setConcurrentGroupType****(java.lang.Object groupType)*

4.58 swarm.activity.ConcurrentSchedule

public interface *ConcurrentSchedule*
implements ActionGroupS,ActionGroup,ScheduleS,Schedule

Time-based map usable for concurrent group.. Time-based map usable for concurrent group.

4.59 swarm.activity.ConcurrentScheduleC

public interface *ConcurrentScheduleC*
implements ActionGroupS,ActionGroupC,ScheduleS,ScheduleC
Time-based map usable for concurrent group.. Time-based map usable for concurrent group.

4.59.1 Implementing class: swarm.activity.ConcurrentScheduleCImpl

java.lang.Object

swarm.PhaseCImpl

public *ConcurrentScheduleCImpl*
extends PhaseCImpl
implements ActionGroupS,ActionGroupC,ScheduleS,ScheduleC,ConcurrentScheduleS,ConcurrentScheduleC

4.59.2 Implementing class: swarm.activity.ConcurrentScheduleImpl

java.lang.Object

public *ConcurrentScheduleImpl*
extends Object
implements ActionGroupS,ActionGroup,ScheduleS,Schedule,ConcurrentScheduleS,ConcurrentSchedule

4.60 swarm.activity.ConcurrentScheduleS

public interface *ConcurrentScheduleS*
implements ActionGroupS,ScheduleS

Time-based map usable for concurrent group.. Time-based map usable for concurrent group.

4.61 swarm.activity.DefaultOrder

public interface *DefaultOrder*

The DefaultOrder option indicates the ordering to be assumed among actions of the plan when no other explicit ordering has been assigned.. 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.

4.61.1 Methods

public Symbol ***getDefaultOrder()***

4.62 swarm.activity.DefaultOrderC

public interface *DefaultOrderC*

The DefaultOrder option indicates the ordering to be assumed among actions of the plan when no other explicit ordering has been assigned.. 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.

4.63 swarm.activity.DefaultOrderS

public interface *DefaultOrderS*

The DefaultOrder option indicates the ordering to be assumed among actions of the plan when no other explicit ordering has been assigned.. 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.

4.63.1 Methods

public Object ***setDefaultOrder****(swarm.defobj.Symbol aSymbol)*

4.64 swarm.activity.FAction

---

public interface *FAction*
implements ActionS,Action

An action defined by sending a FCall.. An action defined by sending a FCall.

4.65 swarm.activity.FActionC

---

public interface *FActionC*
implements ActionS,ActionC

An action defined by sending a FCall.. An action defined by sending a FCall.

4.65.1 Methods

*public Object **setCall**(java.lang.Object fcall)*

4.65.2 Implementing class: swarm.activity.FActionCImpl

java.lang.Object

swarm.PhaseCImpl

---

public *FActionCImpl*
extends PhaseCImpl
implements ActionS,ActionC,FActionS,FActionC

4.66 swarm.activity.FActionCreating

---

public interface *FActionCreating*

An action that calls a FCall.. The createFAction: message creates an action that runs a FCall closure.

4.66.1 Methods

public FAction ***createFAction****(swarm.defobj.FCall call)*

4.67 swarm.activity.FActionCreatingC

---

public interface *FActionCreatingC*

An action that calls a FCall.. The createFAction: message creates an action that runs a FCall closure.

4.68 swarm.activity.FActionCreatingForEachHeterogeneous

public interface *FActionCreatingForEachHeterogeneous*

Invoke a FCall for every item in the target collection, which can include objects of various types.. Invoke a FCall for every item in the target collection, which can include objects of various types.

4.68.1 Methods

public FActionForEachHeterogeneous ***createFActionForEachHeterogeneous$call****(java.lang.Object target, swarm.defobj.FCall call)*

4.69 swarm.activity.FActionCreatingForEachHeterogeneousC

public interface *FActionCreatingForEachHeterogeneousC*

Invoke a FCall for every item in the target collection, which can include objects of various types.. Invoke a FCall for every item in the target collection, which can include objects of various types.

4.70 swarm.activity.FActionCreatingForEachHeterogeneousS

public interface *FActionCreatingForEachHeterogeneousS*

Invoke a FCall for every item in the target collection, which can include objects of various types.. Invoke a FCall for every item in the target collection, which can include objects of various types.

4.71 swarm.activity.FActionCreatingForEachHomogeneous

public interface *FActionCreatingForEachHomogeneous*

Invoke a FCall for every item in the target collection. All members must be of the same type.. Invoke a FCall for every item in the target collection. All members must be of the same type.

4.71.1 Methods

public FActionForEachHomogeneous
createFActionForEachHomogeneous$call*(java.lang.Object target, swarm.defobj.FCall call)*

4.72 swarm.activity.FActionCreatingForEachHomogeneousC

public interface *FActionCreatingForEachHomogeneousC*

Invoke a FCall for every item in the target collection. All members must be of the same type.. Invoke a FCall for every item in the target collection. All members must be of the same type.

4.73 swarm.activity.FActionCreatingForEachHomogeneousS

public interface *FActionCreatingForEachHomogeneousS*

Invoke a FCall for every item in the target collection. All members must be of the same type.. Invoke a FCall for every item in the target collection. All members must be of the same type.

4.74 swarm.activity.FActionCreatingS

public interface *FActionCreatingS*

An action that calls a FCall.. The createFAction: message creates an action that runs a FCall closure.

4.75 swarm.activity.FActionForEach

---

public interface *FActionForEach*
implements FActionS,FAction,ActionTargetS,ActionTarget,DefaultOrderS,DefaultOrder

Base protocol for FActionForEach{Homogeneous,Heterogeneous}.. Base protocol for FActionForEach{Homogeneous,Heterogeneous}.

4.76 swarm.activity.FActionForEachC

public interface *FActionForEachC*
implements FActionS,FActionC,ActionTargetS,ActionTargetC,DefaultOrderS,DefaultOrderC

Base protocol for FActionForEach{Homogeneous,Heterogeneous}.. Base protocol for FActionForEach{Homogeneous,Heterogeneous}.

4.77 swarm.activity.FActionForEachHeterogeneous

public interface *FActionForEachHeterogeneous*
implements FActionForEachS,FActionForEach

An action defined by applying a FAction to every member of a collection.. An action defined by applying a FAction to every member of a collection.

4.78 swarm.activity.FActionForEachHeterogeneousC

public interface *FActionForEachHeterogeneousC*
implements FActionForEachS,FActionForEachC

An action defined by applying a FAction to every member of a collection..
An action defined by applying a FAction to every member of a collection.

4.78.1 Implementing class: swarm.activity.FActionForEachHeterogeneousCImpl

java.lang.Object

swarm.PhaseCImpl

public *FActionForEachHeterogeneousCImpl*
extends PhaseCImpl
implements FActionForEachS,FActionForEachC,FActionForEachHeterogeneousS,FActionForEachHeterogeneousC

4.78.2 Implementing class: swarm.activity.FActionForEachHeterogeneousImpl

java.lang.Object

public *FActionForEachHeterogeneousImpl*
extends Object
implements FActionForEachS,FActionForEach,FActionForEachHeterogeneousS,FActionForEachHeterogeneous

4.79 swarm.activity.FActionForEachHeterogeneousS

public interface *FActionForEachHeterogeneousS*
implements FActionForEachS

An action defined by applying a FAction to every member of a collection.. An action defined by applying a FAction to every member of a collection.

4.80 swarm.activity.FActionForEachHomogeneous

public interface *FActionForEachHomogeneous*
implements FActionForEachS,FActionForEach

An action defined by applying a FAction to every member of a collection All members of the collection must be of the same type.. An action defined by applying a FAction to every member of a collection. All members of the collection must be of the same type.

4.81 swarm.activity.FActionForEachHomogeneousC

public interface *FActionForEachHomogeneousC*
implements FActionForEachS,FActionForEachC

An action defined by applying a FAction to every member of a collection All members of the collection must be of the same type.. An action defined by applying a FAction to every member of a collection. All members of the collection must be of the same type.

4.81.1 Implementing class: swarm.activity.FActionForEachHomogeneousCImpl

java.lang.Object

swarm.PhaseCImpl

public *FActionForEachHomogeneousCImpl*
extends PhaseCImpl
implements FActionForEachS,FActionForEachC,FActionForEachHomogeneousS,FActionForEachHomogeneousC

4.81.2 Implementing class: swarm.activity.FActionForEachHomogeneousImpl

java.lang.Object

public *FActionForEachHomogeneousImpl*
extends Object
implements FActionForEachS,FActionForEach,FActionForEachHomogeneousS,FActionForEachHomogeneous

4.82 swarm.activity.FActionForEachHomogeneousS

public interface *FActionForEachHomogeneousS*
implements FActionForEachS

An action defined by applying a FAction to every member of a collection All members of the collection must be of the same type.. An action defined by applying a FAction to every member of a collection. All members of the collection must be of the same type.

4.83 swarm.activity.FActionForEachS

public interface *FActionForEachS*
implements FActionS,ActionTargetS,DefaultOrderS

Base protocol for FActionForEach{Homogeneous,Heterogeneous}.. Base protocol for FActionForEach{Homogeneous,Heterogeneous}.

4.83.1 Implementing class: swarm.activity.FActionImpl

java.lang.Object

public *FActionImpl*
extends Object
implements ActionS,Action,FActionS,FAction

4.84 swarm.activity.FActionS

public interface *FActionS*
implements ActionS
An action defined by sending a FCall.. An action defined by sending a FCall.

4.84.1 Methods

*public Object **setAutoDrop**(boolean autoDrop)*

4.85 swarm.activity.ForEachActivity

---

public interface *ForEachActivity*
implements ActivityS,Activity

State of execution within a ForEach action.. State of execution within a ForEach action.

4.85.1 Methods

public Object ***getCurrentMember()***

4.86 swarm.activity.ForEachActivityC

---

public interface *ForEachActivityC*
implements ActivityS,ActivityC
State of execution within a ForEach action.. State of execution within a ForEach action.

4.86.1 Implementing class: swarm.activity.ForEachActivityCImpl

java.lang.Object

swarm.PhaseCImpl

---

public *ForEachActivityCImpl*
extends PhaseCImpl
implements ActivityS,ActivityC,ForEachActivityS,ForEachActivityC

4.86.2 Implementing class: swarm.activity.ForEachActivityImpl

java.lang.Object

---

public *ForEachActivityImpl*
extends Object
implements ActivityS,Activity,ForEachActivityS,ForEachActivity

4.87 swarm.activity.ForEachActivityS

public interface *ForEachActivityS*
implements ActivityS

State of execution within a ForEach action.. State of execution within a ForEach action.

4.88 swarm.activity.GetSubactivityAction

public interface *GetSubactivityAction*

Declare an internal method for getCurrentAction().. Declare an internal method for getCurrentAction().

4.88.1 Methods

public Object ***_getSubactivityAction_****()*

4.89 swarm.activity.GetSubactivityActionC

public interface *GetSubactivityActionC*

Declare an internal method for getCurrentAction().. Declare an internal method for getCurrentAction().

4.90 swarm.activity.GetSubactivityActionS

public interface *GetSubactivityActionS*

Declare an internal method for getCurrentAction().. Declare an internal method for getCurrentAction().

4.91 swarm.activity.RelativeTime

public interface *RelativeTime*

Specifies that time is relative to when the schedule started.. 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.

4.91.1 Methods

public boolean ***getRelativeTime****()*

4.92 swarm.activity.RelativeTimeC

public interface *RelativeTimeC*

Specifies that time is relative to when the schedule started.. 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.

4.93 swarm.activity.RelativeTimeS

---

public interface *RelativeTimeS*

Specifies that time is relative to when the schedule started.. 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.

4.93.1 Methods

*public Object **setRelativeTime**(boolean relativeTime)*

4.94 swarm.activity.RepeatInterval

public interface *RepeatInterval*

Reschedule actions after a period of time.. 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.)

4.94.1 Methods

public int ***getRepeatInterval***()

4.95 swarm.activity.RepeatIntervalC

---

public interface *RepeatIntervalC*

Reschedule actions after a period of time.. 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.)

4.96 swarm.activity.RepeatIntervalS

public interface *RepeatIntervalS*

Reschedule actions after a period of time.. 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.)

4.96.1 Methods

public Object ***setRepeatInterval****(int repeatInterval)*

4.97 swarm.activity.Schedule

public interface *Schedule*
implements MapS,Map,CompoundActionS,CompoundAction,ActionCreatingS,ActionCreating,RelativeTimeS,RelativeTime,RepeatIntervalS,RepeatInterval,ConcurrentGroupTypeS,ConcurrentGroupType,SingletonGroupsS,SingletonGroups

A collection of actions ordered by time values.. 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.)

4.97.1 Methods

*public Object **at$createAction**(int tVal, java.lang.Object anActionType)*

*public ActionForEach **at$createActionForEach$message**(int tVal, java.lang.Object target, swarm.Selector aSel)*

*public ActionForEach **at$createActionForEach$message**(int tVal, java.lang.Object target, swarm.Selector aSel, java.lang.Object arg1)*

*public ActionForEach **at$createActionForEach$message**(int tVal, java.lang.Object target, swarm.Selector aSel, java.lang.Object arg1, java.lang.Object arg2)*

*public ActionForEach **at$createActionForEach$message**(int tVal, java.lang.Object target, swarm.Selector aSel, java.lang.Object arg1, java.lang.Object arg2, java.lang.Object arg3)* *public ActionTo* ***at$createActionTo$message****(int tVal, java.lang.Object target, swarm.Selector aSel)*

public ActionTo ***at$createActionTo$message****(int tVal, java.lang.Object target, swarm.Selector aSel, java.lang.Object arg1)*

public ActionTo ***at$createActionTo$message****(int tVal, java.lang.Object target, swarm.Selector aSel, java.lang.Object arg1, java.lang.Object arg2)*

public ActionTo ***at$createActionTo$message****(int tVal, java.lang.Object target, swarm.Selector aSel, java.lang.Object arg1, java.lang.Object arg2, java.lang.Object arg3)*

public FAction ***at$createFAction****(int tVal, swarm.defobj.FCall call)*

public FActionForEachHeterogeneous ***at$createFActionForEachHeterogeneous$call****(int tVal, java.lang.Object target, swarm.defobj.FCall call)*

public FActionForEachHomogeneous ***at$createFActionForEachHomogeneous$call****(int tVal, java.lang.Object target, swarm.defobj.FCall call)*

public ActionGroup ***insertGroup****(int tVal)*

public Object ***remove****(java.lang.Object anAction)*

Remove action from either schedule or concurrent group.

public Object ***setKeepEmptyFlag****(boolean keepEmptyFlag)*

Indicate whether an empty schedule should be dropped and ignored or or kept and attended to (default is YES).

4.98 swarm.activity.ScheduleActivity

public interface *ScheduleActivity*
implements ActivityS,Activity

State of execution within a Schedule.. State of execution within a Schedule.

4.98.1 Methods

*public int **getCurrentTime**()*

Get current time of activity (pending time if holding).

*public Object **stepUntil**(int tVal)*

Advance activity until requested time has been reached.

4.99 swarm.activity.ScheduleActivityC

---

public interface *ScheduleActivityC*
implements ActivityS,ActivityC

State of execution within a Schedule.. State of execution within a Schedule.

4.99.1 Implementing class: swarm.activity.ScheduleActivityCImpl

java.lang.Object

swarm.PhaseCImpl

---

public *ScheduleActivityCImpl*
extends PhaseCImpl
implements ActivityS,ActivityC,ScheduleActivityS,ScheduleActivityC

4.99.2 Implementing class: swarm.activity.ScheduleActivityImpl

java.lang.Object

---

public *ScheduleActivityImpl*
extends Object
implements ActivityS,Activity,ScheduleActivityS,ScheduleActivity

4.100 swarm.activity.ScheduleActivityS

public interface *ScheduleActivityS*
implements ActivityS

State of execution within a Schedule.. State of execution within a Schedule.

4.101 swarm.activity.ScheduleC

public interface *ScheduleC*
implements MapS,MapC,CompoundActionS,CompoundActionC,ActionCreatingS,ActionCreatingC,RelativeTimeS,RelativeTimeC,RepeatIntervalS,RepeatIntervalC,ConcurrentGroupTypeS,ConcurrentGroupTypeC,SingletonGroupsS,SingletonGroupsC

A collection of actions ordered by time values.. 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.)

4.101.1 Methods

public Object ***create$setAutoDrop****(swarm.defobj.Zone aZone, boolean autoDrop)*

Convenience method for creating an AutoDrop Schedule

public Object ***create$setRepeatInterval****(swarm.defobj.Zone aZone, int rptInterval)*

Convenience method for creating a repeating Schedule

4.101.2 Implementing class: swarm.activity.ScheduleCImpl

java.lang.Object

swarm.PhaseCImpl

public *ScheduleCImpl*
extends PhaseCImpl
implements MapS,MapC,CompoundActionS,CompoundActionC,ActionCreatingS,ActionCreatingC,RelativeTimeS,RelativeTimeC,RepeatIntervalS,RepeatIntervalC,ConcurrentGroupTypeS,ConcurrentGroupTypeC,SingletonGroupsS,SingletonGroupsC,ScheduleS,ScheduleC

4.101.3 Implementing class: swarm.activity.ScheduleImpl

java.lang.Object

---

public *ScheduleImpl*
extends Object
implements MapS,Map,CompoundActionS,CompoundAction,ActionCreatingS,ActionCreating,RelativeTimeS,RelativeTime,RepeatIntervalS,RepeatInterval,ConcurrentGroupTypeS,ConcurrentGroupType,SingletonGroupsS,SingletonGroups,ScheduleS,Schedule

4.102 swarm.activity.ScheduleS

public interface *ScheduleS*
implements MapS,CompoundActionS,ActionCreatingS,RelativeTimeS,RepeatIntervalS,ConcurrentGroupTypeS,SingletonGroupsS

A collection of actions ordered by time values.. 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.)

4.103 swarm.activity.SingletonGroups

---

public interface *SingletonGroups*

Indicates that an action group should be created for every time value which is present.. 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.

4.103.1 Methods

public boolean ***getSingletonGroups()***

4.104 swarm.activity.SingletonGroupsC

public interface *SingletonGroupsC*

Indicates that an action group should be created for every time value which is present.. 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.

4.105 swarm.activity.SingletonGroupsS

public interface *SingletonGroupsS*

Indicates that an action group should be created for every time value which is present.. 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.

4.105.1 Methods

public void ***setSingletonGroups****(boolean singletonGroups)*

4.106 swarm.activity.SwarmActivity

public interface *SwarmActivity*
implements ScheduleActivityS,ScheduleActivity

A collection of started subactivities.. A collection of started subactivities.

4.106.1 Methods

*public Object **getSwarm**()*

Return swarm object containing this swarm activity, if any.

*public Schedule **getSynchronizationSchedule**()*

4.107 swarm.activity.SwarmActivityC

public interface *SwarmActivityC*
implements ScheduleActivityS,ScheduleActivityC
A collection of started subactivities.. A collection of started subactivities.

4.107.1 Implementing class: swarm.activity.SwarmActivityCImpl

java.lang.Object

swarm.PhaseCImpl

public *SwarmActivityCImpl*
extends PhaseCImpl
implements ScheduleActivityS,ScheduleActivityC,SwarmActivityS,SwarmActivityC

4.107.2 Implementing class: swarm.activity.SwarmActivityImpl

java.lang.Object

public *SwarmActivityImpl*
extends Object
implements ScheduleActivityS,ScheduleActivity,SwarmActivityS,SwarmActivity

4.108 swarm.activity.SwarmActivityS

public interface *SwarmActivityS*
implements ScheduleActivityS

A collection of started subactivities.. A collection of started subactivities.

4.109 swarm.activity.SwarmProcess

public interface *SwarmProcess*
implements ActionTypeS,ActionType,ZoneS,Zone,SynchronizationTypeS,SynchronizationType

An object that holds a collection of concurrent subprocesses.. 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.

4.109.1 Methods

public SwarmActivity ***getActivity()***

getActivity returns the activity which is currently running of subactivities within the swarm. This activity is the same as the value returned by activateIn: when the swarm was first activated. It returns nil if the swarm has not yet been activated.

public Object ***getInternalZone()***

getInternalZone returns a Zone object that is used by the swarm to hold its internal objects. Even though the swarm itself inherits from Zone and can be used as a Zone for nearly all purposes, this message is also provided so that the zone itself can be obtained independent of all zone behavior.

4.110 swarm.activity.SwarmProcessC

public interface *SwarmProcessC*
implements ActionTypeS,ActionTypeC,ZoneS,ZoneC,SynchronizationTypeS,SynchronizationTypeC

An object that holds a collection of concurrent subprocesses.. 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.

4.110.1 Methods

public Object ***setInternalZoneType****(java.lang.Object aZoneType)*

The InternalZoneType option sets the type of zone which is created by the swarm to hold objects created within the swarm. If set to nil, no internal zone is created within the swarm, and all use of the swarm as if it were a zone raises an error. The default of this option is standard Zone type. (.. Since there is no other Zone type yet, there's no reason to set this option yet except to turn off the internal zone. ..)

4.111 swarm.activity.SwarmProcessS

---

public interface *SwarmProcessS*
implements ActionTypeS,ZoneS,SynchronizationTypeS

An object that holds a collection of concurrent subprocesses.. 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.

4.112 swarm.activity.SynchronizationType

public interface *SynchronizationType*

Synchronization type sets the type of schedule which is used internally by the swarm to synchronize subschedules. . 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.

4.112.1 Methods

public Object ***getSynchronizationType***()

4.113 swarm.activity.SynchronizationTypeC

public interface *SynchronizationTypeC*

Synchronization type sets the type of schedule which is used internally by the swarm to synchronize subschedules. . 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.

4.113.1 Methods

public void ***setSynchronizationType****(java.lang.Object aScheduleType)*

4.114 swarm.activity.SynchronizationTypeS

public interface *SynchronizationTypeS*

Synchronization type sets the type of schedule which is used internally by the swarm to synchronize subschedules. . 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.

Chapter 5

Package swarm.objectbase

Interfaces in package:

Classes in package:

5.1 swarm.objectbase.ActivityControl

public interface *ActivityControl*
implements SwarmObjectS,SwarmObject

A class that provides an object interface to an activity.. 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.

5.1.1 Methods

public Object ***attachToActivity****(java.lang.Object 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.

public Object ***getStatus****()*

The getStatus method returns the status of the activity.

public Object ***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.

public Object ***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.

public Object ***step****()*

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

public Object ***stepUntil****(int 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.

public Object ***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.

public Object ***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.

5.2 swarm.objectbase.ActivityControlC

---

public interface *ActivityControlC*
implements SwarmObjectS,SwarmObjectC

A class that provides an object interface to an activity.. 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.

5.2.1 Implementing class: swarm.objectbase.ActivityControlCImpl

java.lang.Object

swarm.PhaseCImpl

---

public *ActivityControlCImpl*
extends PhaseCImpl
implements SwarmObjectS,SwarmObjectC,ActivityControlS,ActivityControlC

5.2.2 Implementing class: swarm.objectbase.ActivityControlImpl

java.lang.Object

---

public *ActivityControlImpl*
extends Object
implements SwarmObjectS,SwarmObject,ActivityControlS,ActivityControl

5.3 swarm.objectbase.ActivityControlS

public interface *ActivityControlS*
implements SwarmObjectS

A class that provides an object interface to an activity.. 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.

5.4 swarm.objectbase.CompleteProbeMap

---

public interface *CompleteProbeMap*
implements ProbeMapS,ProbeMap

A subclass of ProbeMap whose initial state contains the VarProbes and MessageProbes of the requested target class but also those of all its subclasses.. Upon creation, this subclass of the ProbeMap will contain all the variables and all the messages of a given class (including the inherited ones).

5.5 swarm.objectbase.CompleteProbeMapC

public interface *CompleteProbeMapC*
implements ProbeMapS,ProbeMapC

A subclass of ProbeMap whose initial state contains the VarProbes and MessageProbes of the requested target class but also those of all its subclasses.. Upon creation, this subclass of the ProbeMap will contain all the variables and all the messages of a given class (including the inherited ones).

5.5.1 Implementing class: swarm.objectbase.CompleteProbeMapCImpl

java.lang.Object

swarm.PhaseCImpl

public *CompleteProbeMapCImpl*
extends PhaseCImpl
implements ProbeMapS,ProbeMapC,CompleteProbeMapS,CompleteProbeMapC

5.5.2 Implementing class: swarm.objectbase.CompleteProbeMapImpl

java.lang.Object

public *CompleteProbeMapImpl*
extends Object
implements ProbeMapS,ProbeMap,CompleteProbeMapS,CompleteProbeMap

5.6 swarm.objectbase.CompleteProbeMapS

public interface *CompleteProbeMapS*
implements ProbeMapS

A subclass of ProbeMap whose initial state contains the VarProbes and MessageProbes of the requested target class but also those of all its subclasses.. Upon creation, this subclass of the ProbeMap will contain all the variables and all the messages of a given class (including the inherited ones).

5.7 swarm.objectbase.CompleteVarMap

public interface *CompleteVarMap*
implements ProbeMapS,ProbeMap

A subclass of ProbeMap, whose initial state contains no MessageProbes.. 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.

5.8 swarm.objectbase.CompleteVarMapC

public interface *CompleteVarMapC*
implements ProbeMapS,ProbeMapC

A subclass of ProbeMap, whose initial state contains no MessageProbes.. 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.

5.8.1 Implementing class: swarm.objectbase.CompleteVarMapCImpl

java.lang.Object

swarm.PhaseCImpl

public *CompleteVarMapCImpl*
extends PhaseCImpl
implements ProbeMapS,ProbeMapC,CompleteVarMapS,CompleteVarMapC

5.8.2 Implementing class: swarm.objectbase.CompleteVarMapImpl

java.lang.Object

public *CompleteVarMapImpl*
extends Object
implements ProbeMapS,ProbeMap,CompleteVarMapS,CompleteVarMap

5.9 swarm.objectbase.CompleteVarMapS

public interface *CompleteVarMapS*
implements ProbeMapS

A subclass of ProbeMap, whose initial state contains no MessageProbes.. 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.

5.10 swarm.objectbase.CustomProbeMap

public interface *CustomProbeMap*
implements ProbeMapS,ProbeMap

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.. 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).

5.11 swarm.objectbase.CustomProbeMapC

---

public interface *CustomProbeMapC*
implements ProbeMapS,ProbeMapC

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.. 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).

5.11.1 Implementing class: swarm.objectbase.CustomProbeMapCImpl

java.lang.Object

swarm.PhaseCImpl

---

public *CustomProbeMapCImpl*
extends PhaseCImpl
implements ProbeMapS,ProbeMapC,CustomProbeMapS,CustomProbeMapC

5.11.2 Implementing class: swarm.objectbase.CustomProbeMapImpl

java.lang.Object

---

public *CustomProbeMapImpl*
extends Object
implements ProbeMapS,ProbeMap,CustomProbeMapS,CustomProbeMap

5.12 swarm.objectbase.CustomProbeMapS

public interface *CustomProbeMapS*
implements ProbeMapS

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.. 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).

5.13 swarm.objectbase.DefaultProbeMap

public interface *DefaultProbeMap*
implements ProbeMapS,ProbeMap

A subclass of ProbeMap, whose initial state contains all the VarProbes of the requested target class and also those of all its superclasses.. A subclass of ProbeMap, whose initial state contains all the VarProbes of the requested target class and also those of all its superclasses.

5.14 swarm.objectbase.DefaultProbeMapC

public interface *DefaultProbeMapC*
implements ProbeMapS,ProbeMapC

A subclass of ProbeMap, whose initial state contains all the VarProbes of the requested target class and also those of all its superclasses.. A subclass of ProbeMap, whose initial state contains all the VarProbes of the requested target class and also those of all its superclasses.

5.14.1 Implementing class: swarm.objectbase.DefaultProbeMapCImpl

java.lang.Object

swarm.PhaseCImpl

public *DefaultProbeMapCImpl*
extends PhaseCImpl
implements ProbeMapS,ProbeMapC,DefaultProbeMapS,DefaultProbeMapC

5.14.2 Implementing class: swarm.objectbase.DefaultProbeMapImpl

java.lang.Object

public *DefaultProbeMapImpl*
extends Object
implements ProbeMapS,ProbeMap,DefaultProbeMapS,DefaultProbeMap

5.15 swarm.objectbase.DefaultProbeMapS

public interface *DefaultProbeMapS*
implements ProbeMapS

A subclass of ProbeMap, whose initial state contains all the VarProbes of the requested target class and also those of all its superclasses.. A subclass of ProbeMap, whose initial state contains all the VarProbes of the requested target class and also those of all its superclasses.

5.16 swarm.objectbase.EmptyProbeMap

---

public interface *EmptyProbeMap*
implements CustomProbeMapS,CustomProbeMap

A CustomProbeMap to be used for building up ProbeMaps from scratch..
A CustomProbeMap to be used for building up ProbeMaps from scratch.

5.17 swarm.objectbase.EmptyProbeMapC

public interface *EmptyProbeMapC*
implements CustomProbeMapS,CustomProbeMapC

A CustomProbeMap to be used for building up ProbeMaps from scratch.. A CustomProbeMap to be used for building up ProbeMaps from scratch.

5.17.1 Methods

*public Object **create$forClass**(swarm.defobj.Zone aZone, java.lang.Class aClass)*

Convenience method for creating an EmptyProbeMap

5.17.2 Implementing class: swarm.objectbase.EmptyProbeMapCImpl

java.lang.Object

swarm.PhaseCImpl

public *EmptyProbeMapCImpl*
extends PhaseCImpl
implements CustomProbeMapS,CustomProbeMapC,EmptyProbeMapS,EmptyProbeMapC

5.17.3 Implementing class: swarm.objectbase.EmptyProbeMapImpl

java.lang.Object

public *EmptyProbeMapImpl*
extends Object
implements CustomProbeMapS,CustomProbeMap,EmptyProbeMapS,EmptyProbeMap

5.18 swarm.objectbase.EmptyProbeMapS

---

public interface *EmptyProbeMapS*
implements CustomProbeMapS

A CustomProbeMap to be used for building up ProbeMaps from scratch..
A CustomProbeMap to be used for building up ProbeMaps from scratch.

5.19 swarm.objectbase.MessageProbe

---

public interface *MessageProbe*
implements ProbeS,Probe

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.. 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.

5.19.1 Methods

*public double **doubleDynamicCallOn**(java.lang.Object 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.

*public int **getArgCount**()*

*public String **getArgName**(int which)*

The getArgName: method returns a string representation of the argument key with the given index.

*public boolean **getHideResult**()*

The getHideResult method returns 1 if the result field is "hidden".

*public String **getProbedMessage**()*

The getProbedMessage method returns the string matching the identifier of the message being probed.

*public boolean **isArgumentId**(int which)*

The isArgumentId: method returns 1 if a given argument of the message is of type object, and returns 0 otherwise.

*public boolean **isResultId**()*

The isResultId method returns 1 if the return value of the message is of type object, and returns 0 otherwise.

*public int **longDynamicCallOn**(java.lang.Object 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.

public Object ***objectDynamicCallOn****(java.lang.Object 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.

public Object ***setArg$ToString****(int which, java.lang.String what)*

The setArg:ToString: method sets the nth argument of the message. The argument must be provided in string form.

public String ***stringDynamicCallOn****(java.lang.Object 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 *.

5.20 swarm.objectbase.MessageProbeC

---

public interface *MessageProbeC*
implements ProbeS,ProbeC

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.. 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.

5.20.1 Methods

public Object ***create$setProbedSelector****(swarm.defobj.Zone aZone, swarm.Selector aSel)*

Convenience factory method for common case.

public Object ***setProbedSelector****(swarm.Selector aSel)*

The setProbedSelector: method sets the message to be probed given the selector.

5.20.2 Implementing class: swarm.objectbase.MessageProbeCImpl

java.lang.Object

swarm.PhaseCImpl

---

public *MessageProbeCImpl*
extends PhaseCImpl
implements ProbeS,ProbeC,MessageProbeS,MessageProbeC

5.20.3 Implementing class: swarm.objectbase.MessageProbeImpl

java.lang.Object

---

public *MessageProbeImpl*
extends Object
implements ProbeS,Probe,MessageProbeS,MessageProbe

5.21 swarm.objectbase.MessageProbeS

public interface *MessageProbeS*
implements ProbeS

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.. 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.

5.21.1 Methods

public Object ***setHideResult****(boolean 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.

5.22 swarm.objectbase.Probe

public interface *Probe*
implements SwarmObjectS,SwarmObject

An abstract superclass of both VarProbe and MessageProbe.. 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.

5.22.1 Methods

*public Object **clone**(swarm.defobj.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.

*public Class **getProbedClass**()*

The getProbedClass method returns the class of the object the probe points at as a Class pointer.

*public String **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.

5.23 swarm.objectbase.ProbeC

public interface *ProbeC*
implements SwarmObjectS,SwarmObjectC

An abstract superclass of both VarProbe and MessageProbe.. 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.

5.23.1 Methods

public Object ***setProbedClass****(java.lang.Class aClass)*

The setProbedClass: method sets the class of the object the probe points at and must be called at create time.

5.24 swarm.objectbase.ProbeLibrary

public interface *ProbeLibrary*
implements CreateS,Create,DropS,Drop

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.. 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.

5.24.1 Methods

public ProbeMap ***getCompleteProbeMapFor****(java.lang.Class aClass)*

The getCompleteProbeMapFor: method returns a ProbeMap containing Probes for all the instance variables and messages of the given Class (including inherited variables and messages). The current implementation of ProbeLibrary does not cache CompleteProbeMaps.

public ProbeMap ***getCompleteVarMapFor****(java.lang.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.

public int ***getDisplayPrecision****()*

The getDisplayPrecision method gets the current display precision set in the ProbeLibrary instance.

public MessageProbe ***getProbeForMessage$inClass****(java.lang.String aMessage, java.lang.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.

public VarProbe ***getProbeForVariable$inClass****(java.lang.String aVar, java.lang.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.

public ProbeMap ***getProbeMapFor****(java.lang.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.

public int ***getSavedPrecision****()*

The getSavedPrecision method gets the current saved precision set in the ProbeLibrary instance.

public boolean ***isProbeMapDefinedFor****(java.lang.Class aClass)*

The isProbeMapDefinedFor: method returns True if there is a non-nil value in the ProbeLibrary for that class and False otherwise.

public Object ***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.

public Object ***setProbeMap$For****(java.lang.Object aMap, java.lang.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.

public Object ***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.

5.25 swarm.objectbase.ProbeLibraryC

public interface *ProbeLibraryC*
implements CreateS,CreateC,DropS,DropC

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.. 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.

5.25.1 Implementing class: swarm.objectbase.ProbeLibraryCImpl

java.lang.Object

swarm.PhaseCImpl

public *ProbeLibraryCImpl*
extends PhaseCImpl
implements CreateS,CreateC,DropS,DropC,ProbeLibraryS,ProbeLibraryC

5.25.2 Implementing class: swarm.objectbase.ProbeLibraryImpl

java.lang.Object

public *ProbeLibraryImpl*
extends Object
implements CreateS,Create,DropS,Drop,ProbeLibraryS,ProbeLibrary

5.26 swarm.objectbase.ProbeLibraryS

public interface *ProbeLibraryS*
implements CreateS,DropS

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.. 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.

5.27 swarm.objectbase.ProbeMap

public interface *ProbeMap*
implements SwarmObjectS,SwarmObject

A container class for Probes used to specify the contents of a ProbeDisplay.. 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.

5.27.1 Methods

public Object ***addProbe****(java.lang.Object 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.

public Object ***addProbeMap****(java.lang.Object 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.

public Object ***begin****(swarm.defobj.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.

public Object ***clone****(swarm.defobj.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.

public Object ***dropProbeForMessage****(java.lang.String 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.

public Object ***dropProbeForVariable****(java.lang.String 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.

public Object ***dropProbeMap****(java.lang.Object 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.

public int ***getNumEntries****()*

The getNumEntries method returns the number of probes in the ProbeMap.

public Class ***getProbedClass****()*

The getProbedClass method returns the class of the object that the set of probes that constitute the probe map points at.

public MessageProbe ***getProbeForMessage****(java.lang.String aMessage)*

The getProbeForMessage: method returns the Probe corresponding to the specified message name.

public VarProbe ***getProbeForVariable****(java.lang.String aVariable)*

The getProbeForVariable: method returns the Probe corresponding to the given variable name.

5.28 swarm.objectbase.ProbeMapC

---

public interface *ProbeMapC*
implements SwarmObjectS,SwarmObjectC

A container class for Probes used to specify the contents of a ProbeDisplay.. 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.

5.28.1 Methods

public Object ***setProbedClass****(java.lang.Class aClass)*

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.

5.28.2 Implementing class: swarm.objectbase.ProbeMapCImpl

java.lang.Object

swarm.PhaseCImpl

---

public *ProbeMapCImpl*
extends PhaseCImpl
implements SwarmObjectS,SwarmObjectC,ProbeMapS,ProbeMapC

5.28.3 Implementing class: swarm.objectbase.ProbeMapImpl

java.lang.Object

---

public *ProbeMapImpl*
extends Object
implements SwarmObjectS,SwarmObject,ProbeMapS,ProbeMap

5.29 swarm.objectbase.ProbeMapS

---

public interface *ProbeMapS*
implements SwarmObjectS

A container class for Probes used to specify the contents of a ProbeDisplay.. 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.

5.30 swarm.objectbase.ProbeS

public interface *ProbeS*
implements SwarmObjectS

An abstract superclass of both VarProbe and MessageProbe.. 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.

5.30.1 Methods

public Object ***setSafety()***

The setSafety method turns on the option of checking the compatibility of the class of the object before any actions are performed on the object.

public 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.

5.31 swarm.objectbase.Swarm

public interface *Swarm*
implements SwarmProcessS,SwarmProcess
A temporal container.. A Swarm is a community of agents sharing a common timescale as well as common memory pool.

5.31.1 Methods

public Activity ***activateIn****(swarm.objectbase.Swarm swarmContext)*

Override this to activate any actions you built in buildActions. Note, you must activate yourself first before you can activate actions inside you.

public Object ***buildActions****()*

Override this to let your Swarm build its actions.

public Object ***buildObjects****()*

Override this to let your Swarm create the objects that it contains.

public ProbeMap ***getCompleteProbeMap****()*

Needed to support probing of Swarms.

public VarProbe ***getProbeForVariable****(java.lang.String aVariable)*

Needed to support probing of Swarms.

public ProbeMap ***getProbeMap****()*

Needed to support probing of Swarms.

5.32 swarm.objectbase.SwarmC

---

public interface *SwarmC*
implements SwarmProcessS,SwarmProcessC

A temporal container.. A Swarm is a community of agents sharing a common timescale as well as common memory pool.

5.32.1 Implementing class: swarm.objectbase.SwarmCImpl

java.lang.Object

swarm.PhaseCImpl

---

public *SwarmCImpl*
extends PhaseCImpl
implements SwarmProcessS,SwarmProcessC,SwarmS,SwarmC

5.32.2 Implementing class: swarm.objectbase.SwarmImpl

java.lang.Object

---

public *SwarmImpl*
extends Object
implements SwarmProcessS,SwarmProcess,SwarmS,Swarm

5.33 swarm.objectbase.SwarmObject

---

public interface *SwarmObject*
implements CreateS,Create,DropS,Drop

A superclass of most objects in a Swarm simulation that provides support for probing.. 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.

5.33.1 Methods

public ProbeMap ***getCompleteProbeMap()***

The getCompleteProbeMap method returns a newly created CompleteProbeMap for an object.

public MessageProbe ***getProbeForMessage****(java.lang.String aMessage)*

The getProbeForMessage: method returns the MessageProbe indexed in the ProbeMap by the string aMessage.

public VarProbe ***getProbeForVariable****(java.lang.String aVariable)*

The getProbeForVariable: method returns the VarProbe indexed in the ProbeMap by the string aVariable.

public 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.

5.34 swarm.objectbase.SwarmObjectC

---

public interface *SwarmObjectC*
implements CreateS,CreateC,DropS,DropC

A superclass of most objects in a Swarm simulation that provides support for probing.. 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.

5.34.1 Implementing class: swarm.objectbase.SwarmObjectCImpl

java.lang.Object

swarm.PhaseCImpl

---

public *SwarmObjectCImpl*
extends PhaseCImpl
implements CreateS,CreateC,DropS,DropC,SwarmObjectS,SwarmObjectC

5.34.2 Implementing class: swarm.objectbase.SwarmObjectImpl

java.lang.Object

---

public *SwarmObjectImpl*
extends Object
implements CreateS,Create,DropS,Drop,SwarmObjectS,SwarmObject

5.35 swarm.objectbase.SwarmObjectS

public interface *SwarmObjectS*
implements CreateS,DropS

A superclass of most objects in a Swarm simulation that provides support for probing.. 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.

5.36 swarm.objectbase.SwarmS

public interface *SwarmS*
implements SwarmProcessS

A temporal container.. A Swarm is a community of agents sharing a common timescale as well as common memory pool.

5.37 swarm.objectbase.VarProbe

public interface *VarProbe*
implements ProbeS,Probe

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.. 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.

5.37.1 Methods

public String ***getBaseType****()*

In the case of arrays, returns the base type.

public boolean ***getInteractiveFlag****()*

The getInteractiveFlag method returns the interactivity state of the VarProbe.

public String ***getProbedVariable****()*

The getProbedVariable method returns a string matching the identifier of variable being probed.

public int ***getRank****()*

Returns rank of array, or 0 for scalar objects.

public double ***probeAsDouble****(java.lang.Object anObject)*

The probeAsDouble: method returns a pointer to the probed variable as a double.

public int ***probeAsInt****(java.lang.Object anObject)*

The probeAsInt: method returns a pointer to the probed variable as an integer.

public Object ***probeObject****(java.lang.Object anObject)*

A field probed with probeAsObject: must be an object.

public boolean ***setData$ToString****(java.lang.Object anObject, java.lang.String 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'").

5.38 swarm.objectbase.VarProbeC

public interface *VarProbeC*
implements ProbeS,ProbeC

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.. 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.

5.38.1 Methods

public Object ***createEnd****()*

public Object ***setProbedVariable****(java.lang.String 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.

5.38.2 Implementing class: swarm.objectbase.VarProbeCImpl

java.lang.Object

swarm.PhaseCImpl

public *VarProbeCImpl*
extends PhaseCImpl
implements ProbeS,ProbeC,VarProbeS,VarProbeC

5.38.3 Implementing class: swarm.objectbase.VarProbeImpl

java.lang.Object

public *VarProbeImpl*
extends Object
implements ProbeS,Probe,VarProbeS,VarProbe

5.39 swarm.objectbase.VarProbeS

---

public interface *VarProbeS*
implements ProbeS

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.. 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.

5.39.1 Methods

public Object ***setFloatFormat****(java.lang.String format)*

The setFloatFormat: method sets the floating-point format of a GUI display widget when given a sprintf-style formatting string.

public Object ***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.

public Object ***setStringReturnType****(java.lang.Object 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.

Chapter 6

Package swarm.random

Interfaces in package:

Classes in package:

6.1 swarm.random.BasicRandomGenerator

public interface *BasicRandomGenerator*
implements SwarmObjectS,SwarmObject,InternalStateS,InternalState,CommonGeneratorS,CommonGenerator

The common functionality of simple and split generators.. This protocol covers methods common to simple and split generators.

6.2 swarm.random.BasicRandomGeneratorC

public interface *BasicRandomGeneratorC*
implements SwarmObjectS,SwarmObjectC,InternalStateS,InternalStateC,CommonGeneratorS,CommonGeneratorC

The common functionality of simple and split generators.. This protocol covers methods common to simple and split generators.

6.3 swarm.random.BasicRandomGeneratorS

public interface *BasicRandomGeneratorS*
implements SwarmObjectS,InternalStateS,CommonGeneratorS

The common functionality of simple and split generators.. This protocol covers methods common to simple and split generators.

6.4 swarm.random.BernoulliDist

public interface *BernoulliDist*
implements BooleanDistributionS,BooleanDistribution

Bernoulli Distribution . A distribution returning YES with a given probability.

6.4.1 Methods

public double ***getProbability()***

The getProbability method returns the probability of returning YES.

public boolean ***getSampleWithProbability****(double p)*

The getSampleWithProbability: returns a sample YES or NO value.

6.5 swarm.random.BernoulliDistC

---

public interface *BernoulliDistC*
implements BooleanDistributionS,BooleanDistributionC
Bernoulli Distribution . A distribution returning YES with a given probability.

6.5.1 Methods

public Object ***create$setGenerator$setProbability****(swarm.defobj.Zone aZone, java.lang.Object generator, double p)*

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

public Object ***create$setGenerator$setVirtualGenerator$setProbability****(swarm.defobj.Zone aZone, java.lang.Object generator, int vGen, double p)*

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

6.5.2 Implementing class: swarm.random.BernoulliDistCImpl

java.lang.Object

swarm.PhaseCImpl

---

public *BernoulliDistCImpl*
extends PhaseCImpl
implements BooleanDistributionS,BooleanDistributionC,BernoulliDistS,BernoulliDistC

6.5.3 Implementing class: swarm.random.BernoulliDistImpl

java.lang.Object

---

public *BernoulliDistImpl*
extends Object
implements BooleanDistributionS,BooleanDistribution,BernoulliDistS,BernoulliDist

6.6 swarm.random.BernoulliDistS

public interface *BernoulliDistS*
implements BooleanDistributionS

Bernoulli Distribution . A distribution returning YES with a given probability.

6.6.1 Methods

public Object ***setProbability****(double p)*

The setProbability: method sets the probability of returning YES.

6.7 swarm.random.BooleanDistribution

public interface *BooleanDistribution*
implements ProbabilityDistributionS,ProbabilityDistribution

Boolean Distribution. A probability distribution that returns YES/NO sample values.

6.7.1 Methods

public boolean ***getBooleanSample()***

The getBooleanSample method returns a YES or NO sample value.

public int ***getIntegerSample()***

6.8 swarm.random.BooleanDistributionC

public interface *BooleanDistributionC*
implements ProbabilityDistributionS,ProbabilityDistributionC

Boolean Distribution. A probability distribution that returns YES/NO sample values.

6.9 swarm.random.BooleanDistributionS

public interface *BooleanDistributionS*
implements ProbabilityDistributionS

Boolean Distribution. A probability distribution that returns YES/NO sample values.

6.10 swarm.random.C2TAUS1gen

---

public interface *C2TAUS1gen*
implements C2TAUSgenS,C2TAUSgen

Combined Tausworthe generator 1. 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.

6.11 swarm.random.C2TAUS1genC

public interface *C2TAUS1genC*
implements C2TAUSgenS,C2TAUSgenC

Combined Tausworthe generator 1. 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.

6.11.1 Implementing class: swarm.random.C2TAUS1genCImpl

java.lang.Object

swarm.PhaseCImpl

public *C2TAUS1genCImpl*
extends PhaseCImpl
implements C2TAUSgenS,C2TAUSgenC,C2TAUS1genS,C2TAUS1genC

6.11.2 Implementing class: swarm.random.C2TAUS1genImpl

java.lang.Object

public *C2TAUS1genImpl*
extends Object
implements C2TAUSgenS,C2TAUSgen,C2TAUS1genS,C2TAUS1gen

6.12 swarm.random.C2TAUS1genS

public interface *C2TAUS1genS*
implements C2TAUSgenS

Combined Tausworthe generator 1. 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.

6.13 swarm.random.C2TAUS2gen

public interface *C2TAUS2gen*
implements C2TAUSgenS,C2TAUSgen

Combined Tausworthe generator 2. 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.

6.14 swarm.random.C2TAUS2genC

---

public interface *C2TAUS2genC*
implements C2TAUSgenS,C2TAUSgenC

Combined Tausworthe generator 2. 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.

6.14.1 Implementing class: swarm.random.C2TAUS2genCImpl

java.lang.Object

swarm.PhaseCImpl

---

public *C2TAUS2genCImpl*
extends PhaseCImpl
implements C2TAUSgenS,C2TAUSgenC,C2TAUS2genS,C2TAUS2genC

6.14.2 Implementing class: swarm.random.C2TAUS2genImpl

java.lang.Object

---

public *C2TAUS2genImpl*
extends Object
implements C2TAUSgenS,C2TAUSgen,C2TAUS2genS,C2TAUS2gen

6.15 swarm.random.C2TAUS2genS

public interface *C2TAUS2genS*
implements C2TAUSgenS

Combined Tausworthe generator 2. 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.

6.16 swarm.random.C2TAUS3gen

public interface *C2TAUS3gen*
implements C2TAUSgenS,C2TAUSgen

Combined Tausworthe generator 3. 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.

6.17 swarm.random.C2TAUS3genC

---

public interface *C2TAUS3genC*
implements C2TAUSgenS,C2TAUSgenC

Combined Tausworthe generator 3. 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.

6.17.1 Implementing class: swarm.random.C2TAUS3genCImpl

java.lang.Object

swarm.PhaseCImpl

---

public *C2TAUS3genCImpl*
extends PhaseCImpl
implements C2TAUSgenS,C2TAUSgenC,C2TAUS3genS,C2TAUS3genC

6.17.2 Implementing class: swarm.random.C2TAUS3genImpl

java.lang.Object

---

public *C2TAUS3genImpl*
extends Object
implements C2TAUSgenS,C2TAUSgen,C2TAUS3genS,C2TAUS3gen

6.18 swarm.random.C2TAUS3genS

public interface *C2TAUS3genS*
implements C2TAUSgenS

Combined Tausworthe generator 3. 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.

6.19 swarm.random.C2TAUSgen

---

public interface *C2TAUSgen*
implements SimpleRandomGeneratorS,SimpleRandomGenerator

Combined Tausworthe generator . This generator is based on 2 component generators of periods 2^31-1 and 2^29-1.

6.20 swarm.random.C2TAUSgenC

public interface *C2TAUSgenC*
implements SimpleRandomGeneratorS,SimpleRandomGeneratorC

Combined Tausworthe generator . This generator is based on 2 component generators of periods 2^31-1 and 2^29-1.

6.21 swarm.random.C2TAUSgenS

public interface *C2TAUSgenS*
implements SimpleRandomGeneratorS

Combined Tausworthe generator . This generator is based on 2 component generators of periods 2^31-1 and 2^29-1.

6.22 swarm.random.CommonGenerator

public interface *CommonGenerator*

Internal.

6.22.1 Methods

public boolean ***getAntithetic()***

The getAntithetic method returns the current values of the parameter.

public int ***getInitialSeed()***

The getInitialSeed method returns the value of the generator's starting seed.

public int ***getMaxSeedValue()***

The getMaxSeedValue method returns the upper limit on the seed value that can be supplied.

public int ***getUnsignedMax()***

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

public int ***lengthOfSeedVector()***

The lengthOfSeedVector method returns the number of seeds required if you wish to set the state directly.

public Object ***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.

6.23 swarm.random.CommonGeneratorC

public interface *CommonGeneratorC*

Internal.

6.23.1 Methods

public Object ***createWithDefaults****(swarm.defobj.Zone aZone)*

6.24 swarm.random.CommonGeneratorS

public interface *CommonGeneratorS*

Internal.

6.24.1 Methods

public Object ***setAntithetic****(boolean antiT)*

The setAntithetic method turns on or off antithetic output (default=off). Antithetic output is (unsignedMax - u) or (1.0 - d).

public Object ***setStateFromSeed****(int seed)*

The setStateFromSeed method initializes the seed dependent part of the state from a single seed value.

6.25 swarm.random.DoubleDistribution

public interface *DoubleDistribution*
implements ProbabilityDistributionS,ProbabilityDistribution

Double Distribution . A probability distribution that returns an approximation of continuous values as represented by double-precision floating point values.

6.25.1 Methods

public double ***getDoubleSample()***

The getDoubleSample method returns a double-precision floating point value.

6.26 swarm.random.DoubleDistributionC

public interface *DoubleDistributionC*
implements ProbabilityDistributionS,ProbabilityDistributionC

Double Distribution . A probability distribution that returns an approximation of continuous values as represented by double-precision floating point values.

6.27 swarm.random.DoubleDistributionS

public interface *DoubleDistributionS*
implements ProbabilityDistributionS

Double Distribution . A probability distribution that returns an approximation of continuous values as represented by double-precision floating point values.

6.28 swarm.random.ExponentialDist

public interface *ExponentialDist*
implements DoubleDistributionS,DoubleDistribution

Exponential distribuiton . A well-known continuous probability distribution returning doubles.

6.28.1 Methods

public double ***getMean()***

The getMean method returns the mean of the distribution.

public double ***getSampleWithMean****(double mean)*

The getSampleWithMean: method returns a sample value from a distribution with the specified mean.

6.29 swarm.random.ExponentialDistC

public interface *ExponentialDistC*
implements DoubleDistributionS,DoubleDistributionC

Exponential distribuiton . A well-known continuous probability distribution returning doubles.

6.29.1 Methods

*public Object **create$setGenerator$setMean**(swarm.defobj.Zone aZone, java.lang.Object generator, double mean)*

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

*public Object **create$setGenerator$setVirtualGenerator$setMean**(swarm.defobj.Zone aZone, java.lang.Object generator, int vGen, double mean)*

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

6.29.2 Implementing class: swarm.random.ExponentialDistCImpl

java.lang.Object

swarm.PhaseCImpl

public *ExponentialDistCImpl*
extends PhaseCImpl
implements DoubleDistributionS,DoubleDistributionC,ExponentialDistS,ExponentialDistC

6.29.3 Implementing class: swarm.random.ExponentialDistImpl

java.lang.Object

public *ExponentialDistImpl*
extends Object
implements DoubleDistributionS,DoubleDistribution,ExponentialDistS,ExponentialDist

6.30 swarm.random.ExponentialDistS

public interface *ExponentialDistS*
implements DoubleDistributionS

Exponential distribuiton . A well-known continuous probability distribution returning doubles.

6.30.1 Methods

*public Object **setMean**(double mean)*

The setMean: method sets the mean of the distribution.

6.31 swarm.random.GammaDist

public interface *GammaDist*
implements DoubleDistributionS,DoubleDistribution

Gamma distribution. A well-known continuous probability distribution returning doubles

6.31.1 Methods

public double ***getAlpha()***

The getAlpha method returns the alpha value.

public double ***getBeta()***

The getBeta method returns the beta value.

public double ***getSampleWithAlpha$withBeta****(double alpha, double beta)*

The getSampleWithAlpha:withBeta: method returns a sample value from a Gamma distribution with the specified alpha and beta values.

6.32 swarm.random.GammaDistC

public interface *GammaDistC*
implements DoubleDistributionS,DoubleDistributionC

Gamma distribution. A well-known continuous probability distribution returning doubles

6.32.1 Methods

public Object ***create$setGenerator$setAlpha$setBeta****(swarm.defobj.Zone aZone, java.lang.Object generator, double alpha, double beta)*

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

public Object ***create$setGenerator$setVirtualGenerator$setAlpha$setBeta****(swarm.defobj.Zone aZone, java.lang.Object generator, int vGen, double alpha, double beta)*

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

6.32.2 Implementing class: swarm.random.GammaDistCImpl

java.lang.Object

swarm.PhaseCImpl

public *GammaDistCImpl*
extends PhaseCImpl
implements DoubleDistributionS,DoubleDistributionC,GammaDistS,GammaDistC

6.32.3 Implementing class: swarm.random.GammaDistImpl

java.lang.Object

public *GammaDistImpl*
extends Object
implements DoubleDistributionS,DoubleDistribution,GammaDistS,GammaDist

6.33 swarm.random.GammaDistS

public interface *GammaDistS*
implements DoubleDistributionS

Gamma distribution. A well-known continuous probability distribution returning doubles

6.33.1 Methods

public Object ***setAlpha$setBeta****(double alpha, double beta)*

The setAlpha:setBeta: method sets the alpha and beta values for the gamma distribution.

6.34 swarm.random.IntegerDistribution

public interface *IntegerDistribution*
implements ProbabilityDistributionS,ProbabilityDistribution

Integer Distribution . A probability distribution that returns integer sample values.

6.34.1 Methods

public int ***getIntegerSample()***

The getIntegerSample method returns an integer sample value.

6.35 swarm.random.IntegerDistributionC

public interface *IntegerDistributionC*
implements ProbabilityDistributionS,ProbabilityDistributionC

Integer Distribution . A probability distribution that returns integer sample values.

6.36 swarm.random.IntegerDistributionS

---

public interface *IntegerDistributionS*
implements ProbabilityDistributionS

Integer Distribution . A probability distribution that returns integer sample values.

6.37 swarm.random.InternalState

public interface *InternalState*

Archiving routines for internal generator and distribution state.. 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.

6.37.1 Methods

*public void **describe**(java.lang.Object outStream)*

*public int **getMagic**()*

*public String **getName**()*

*public int **getStateSize**()*

Specifies the minimum buffer size needed.

6.38 swarm.random.InternalStateC

public interface *InternalStateC*

Archiving routines for internal generator and distribution state.. 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.

6.39 swarm.random.InternalStateS

public interface *InternalStateS*

Archiving routines for internal generator and distribution state.. 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.

6.40 swarm.random.LogNormalDist

public interface *LogNormalDist*
implements NormalS,Normal

Log-Normal distribution. A well-known continuous probability distribution returning doubles.

6.41 swarm.random.LogNormalDistC

---

public interface *LogNormalDistC*
implements NormalS,NormalC
Log-Normal distribution. A well-known continuous probability distribution returning doubles.

6.41.1 Implementing class: swarm.random.LogNormalDistCImpl

java.lang.Object

swarm.PhaseCImpl

---

public *LogNormalDistCImpl*
extends PhaseCImpl
implements NormalS,NormalC,LogNormalDistS,LogNormalDistC

6.41.2 Implementing class: swarm.random.LogNormalDistImpl

java.lang.Object

---

public *LogNormalDistImpl*
extends Object
implements NormalS,Normal,LogNormalDistS,LogNormalDist

6.42 swarm.random.LogNormalDistS

public interface *LogNormalDistS*
implements NormalS

Log-Normal distribution. A well-known continuous probability distribution returning doubles.

6.43 swarm.random.MT19937gen

public interface *MT19937gen*
implements SimpleRandomGeneratorS,SimpleRandomGenerator

'Mersenne Twister' Twisted GFSR generator. This generator has a single cycle of length 2^19937-1.

6.44 swarm.random.MT19937genC

public interface *MT19937genC*
implements SimpleRandomGeneratorS,SimpleRandomGeneratorC
'Mersenne Twister' Twisted GFSR generator. This generator has a single cycle of length 2^19937-1.

6.44.1 Implementing class: swarm.random.MT19937genCImpl

java.lang.Object

swarm.PhaseCImpl

public *MT19937genCImpl*
extends PhaseCImpl
implements SimpleRandomGeneratorS,SimpleRandomGeneratorC,MT19937genS,MT19937genC

6.44.2 Implementing class: swarm.random.MT19937genImpl

java.lang.Object

public *MT19937genImpl*
extends Object
implements SimpleRandomGeneratorS,SimpleRandomGenerator,MT19937genS,MT19937gen

6.45 swarm.random.MT19937genS

public interface *MT19937genS*
implements SimpleRandomGeneratorS

'Mersenne Twister' Twisted GFSR generator. This generator has a single cycle of length 2^19937-1.

6.46 swarm.random.Normal

public interface *Normal*
implements DoubleDistributionS,DoubleDistribution

Internal.

6.46.1 Methods

public double ***getMean****()*

The getMean method returns the mean of the distribution.

public double ***getSampleWithMean$withStdDev****(double mean, double sdev)*

The getSampleWithMean:withStdDev: method returns a sample value drawn from a distribution with the specified mean and standard deviation.

public double ***getSampleWithMean$withVariance****(double mean, double variance)*

The getSampleWithMean:withVariance: method returns a sample value drawn from a distribution with the specified mean and variance.

public double ***getStdDev****()*

The getStdDev method returns the standard deviation of the distribution.

public double ***getVariance****()*

The getVariance method returns the variance of the distribution.

6.47 swarm.random.NormalC

public interface *NormalC*
implements DoubleDistributionS,DoubleDistributionC

Internal.

6.47.1 Methods

public Object
create$setGenerator$setMean$setStdDev*(swarm.defobj.Zone aZone, java.lang.Object generator, double mean, double sdev)*

Use this create message if the generator to be attached is a Simple one and you wish to specify the standard deviation:

public Object
create$setGenerator$setMean$setVariance*(swarm.defobj.Zone aZone, java.lang.Object generator, double mean, double variance)*

Use this create message if the generator to be attached is a Simple one and you wish to specify the variance:

public Object ***create$setGenerator$setVirtualGenerator$setMean$setStdDev****(swarm.defobj.Zone aZone, java.lang.Object generator, int vGen, double mean, double sdev)*

Use this create message if the generator to be attached is a Split one and you wish to specify the standard deviation:

public Object ***create$setGenerator$setVirtualGenerator$setMean$setVariance****(swarm.defobj.Zone aZone, java.lang.Object generator, int vGen, double mean, double variance)*

Use this create message if the generator to be attached is a Split one and you wish to specify the variance:

6.48 swarm.random.NormalDist

---

public interface *NormalDist*
implements NormalS,Normal

Normal (Gaussian) distribution. A well-known continuous probability distribution returning doubles.

6.49 swarm.random.NormalDistC

---

public interface *NormalDistC*
implements NormalS,NormalC
Normal (Gaussian) distribution. A well-known continuous probability distribution returning doubles.

6.49.1 Implementing class: swarm.random.NormalDistCImpl

java.lang.Object

swarm.PhaseCImpl

---

public *NormalDistCImpl*
extends PhaseCImpl
implements NormalS,NormalC,NormalDistS,NormalDistC

6.49.2 Implementing class: swarm.random.NormalDistImpl

java.lang.Object

---

public *NormalDistImpl*
extends Object
implements NormalS,Normal,NormalDistS,NormalDist

6.50 swarm.random.NormalDistS

public interface *NormalDistS*
implements NormalS

Normal (Gaussian) distribution. A well-known continuous probability distribution returning doubles.

6.51 swarm.random.NormalS

public interface *NormalS*
implements DoubleDistributionS
Internal.

6.51.1 Methods

public Object ***setMean$setStdDev****(double mean, double sdev)*

The setMean:setStdDev: method sets the mean and the standard deviation of the distribution.

public Object ***setMean$setVariance****(double mean, double variance)*

The setMean:setVariance: method sets the mean and the variance of the distribution.

6.52 swarm.random.PMMLCG1gen

public interface *PMMLCG1gen*
implements PMMLCGgenS,PMMLCGgen

Prime Modulus Multiplicative Linear Congruential Generator 1. With parameters a = 16,807 and m = 2,147,483,647, this generator has a single full cycle of length (m-1).

6.53 swarm.random.PMMLCG1genC

public interface *PMMLCG1genC*
implements PMMLCGgenS,PMMLCGgenC

Prime Modulus Multiplicative Linear Congruential Generator 1. With parameters a = 16,807 and m = 2,147,483,647, this generator has a single full cycle of length (m-1).

6.53.1 Implementing class: swarm.random.PMMLCG1genCImpl

java.lang.Object

swarm.PhaseCImpl

public *PMMLCG1genCImpl*
extends PhaseCImpl
implements PMMLCGgenS,PMMLCGgenC,PMMLCG1genS,PMMLCG1genC

6.53.2 Implementing class: swarm.random.PMMLCG1genImpl

java.lang.Object

public *PMMLCG1genImpl*
extends Object
implements PMMLCGgenS,PMMLCGgen,PMMLCG1genS,PMMLCG1gen

6.54 swarm.random.PMMLCG1genS

public interface *PMMLCG1genS*
implements PMMLCGgenS

Prime Modulus Multiplicative Linear Congruential Generator 1. With parameters a = 16,807 and m = 2,147,483,647, this generator has a single full cycle of length (m-1).

6.55 swarm.random.PMMLCGgen

public interface *PMMLCGgen*
implements SimpleRandomGeneratorS,SimpleRandomGenerator

Prime Modulus Multiplicative Linear Congruential Generator. These generator have single full cycle of length (m-1).

6.56 swarm.random.PMMLCGgenC

public interface *PMMLCGgenC*
implements SimpleRandomGeneratorS,SimpleRandomGeneratorC

Prime Modulus Multiplicative Linear Congruential Generator. These generator have single full cycle of length (m-1).

6.57 swarm.random.PMMLCGgenS

public interface *PMMLCGgenS*
implements SimpleRandomGeneratorS

Prime Modulus Multiplicative Linear Congruential Generator. These generator have single full cycle of length (m-1).

6.58 swarm.random.ProbabilityDistribution

---

public interface *ProbabilityDistribution*
implements SwarmObjectS,SwarmObject,InternalStateS,InternalState

Probability Distribution. 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.

6.58.1 Methods

public long ***getCurrentCount()***

The getCurrentCount method returns the count of variates generated.

public Object ***getGenerator()***

The getGenerator method returns the id of the generator.

public boolean ***getOptionsInitialized()***

The getOptionsInitialized returns the value of the parameter.

public int ***getVirtualGenerator()***

The getVirtualGenerator returns the number of the virtual generator used.

6.59 swarm.random.ProbabilityDistributionC

public interface *ProbabilityDistributionC*
implements SwarmObjectS,SwarmObjectC,InternalStateS,InternalStateC

Probability Distribution. 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.

6.59.1 Methods

public Object ***create$setGenerator****(swarm.defobj.Zone aZone, java.lang.Object generator)*

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

public Object
create$setGenerator$setVirtualGenerator*(swarm.defobj.Zone aZone, java.lang.Object generator, int vGen)*

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

public Object ***createWithDefaults****(swarm.defobj.Zone aZone)*

The createWithDefaults method creates a distribution object with a default set of seeds and parameters, and its own private generator.

6.60 swarm.random.ProbabilityDistributionS

public interface *ProbabilityDistributionS*
implements SwarmObjectS,InternalStateS

Probability Distribution. 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.

6.60.1 Methods

public Object ***reset()***

The reset method resets the currentCount and other state data.

public Object ***setGenerator****(java.lang.Object generator)*

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

public Object ***setGenerator$setVirtualGenerator****(java.lang.Object generator, int vGen)*

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

6.61 swarm.random.PSWBgen

public interface *PSWBgen*
implements SimpleRandomGeneratorS,SimpleRandomGenerator

Subtract-with-borrow Congruential Generator with prime modulus. PSWB is an improvement on SWB in that the use of a prime modulus guarantees a single full cycle. It's slower, of course.

6.62 swarm.random.PSWBgenC

---

public interface *PSWBgenC*
implements SimpleRandomGeneratorS,SimpleRandomGeneratorC

Subtract-with-borrow Congruential Generator with prime modulus. PSWB is an improvement on SWB in that the use of a prime modulus guarantees a single full cycle. It's slower, of course.

6.62.1 Implementing class: swarm.random.PSWBgenCImpl

java.lang.Object

swarm.PhaseCImpl

---

public *PSWBgenCImpl*
extends PhaseCImpl
implements SimpleRandomGeneratorS,SimpleRandomGeneratorC,PSWBgenS,PSWBgenC

6.62.2 Implementing class: swarm.random.PSWBgenImpl

java.lang.Object

---

public *PSWBgenImpl*
extends Object
implements SimpleRandomGeneratorS,SimpleRandomGenerator,PSWBgenS,PSWBgen

6.63 swarm.random.PSWBgenS

public interface *PSWBgenS*
implements SimpleRandomGeneratorS

Subtract-with-borrow Congruential Generator with prime modulus. PSWB is an improvement on SWB in that the use of a prime modulus guarantees a single full cycle. It's slower, of course.

6.64 swarm.random.RandomBitDist

public interface *RandomBitDist*
implements BooleanDistributionS,BooleanDistribution

Random Bit Distribution . A generator that returns uniformly distributed single bit values (i.e. fair coin tosses).

6.64.1 Methods

public boolean ***getCoinToss()***

The getCoinToss method returns a YES or NO value.

6.65 swarm.random.RandomBitDistC

---

public interface *RandomBitDistC*
implements BooleanDistributionS,BooleanDistributionC
Random Bit Distribution . A generator that returns uniformly distributed single bit values (i.e. fair coin tosses).

6.65.1 Implementing class: swarm.random.RandomBitDistCImpl

java.lang.Object

swarm.PhaseCImpl

---

public *RandomBitDistCImpl*
extends PhaseCImpl
implements BooleanDistributionS,BooleanDistributionC,RandomBitDistS,RandomBitDistC

6.65.2 Implementing class: swarm.random.RandomBitDistImpl

java.lang.Object

---

public *RandomBitDistImpl*
extends Object
implements BooleanDistributionS,BooleanDistribution,RandomBitDistS,RandomBitDist

6.66 swarm.random.RandomBitDistS

public interface *RandomBitDistS*
implements BooleanDistributionS

Random Bit Distribution . A generator that returns uniformly distributed single bit values (i.e. fair coin tosses).

6.67 swarm.random.SimpleGenerator

public interface *SimpleGenerator*

Internal.

6.67.1 Methods

public long ***getCurrentCount()***

The getCurrentCount method returns the count of variates generated.

public double ***getDoubleSample()***

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

public float ***getFloatSample()***

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

public double ***getThinDoubleSample()***

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

public int ***getUnsignedSample()***

The getUnsignedSample method returns a random unsigned integer uniformly distributed over [0,unsignedMax].

6.68 swarm.random.SimpleGeneratorC

public interface *SimpleGeneratorC*

Internal.

6.68.1 Methods

public Object ***create$setStateFromSeed****(swarm.defobj.Zone aZone, int seed)*

6.69 swarm.random.SimpleGeneratorS

public interface *SimpleGeneratorS*

Internal.

6.70 swarm.random.SimpleRandomGenerator

public interface *SimpleRandomGenerator*
implements BasicRandomGeneratorS,BasicRandomGenerator,SimpleGeneratorS,SimpleGenerator

A Simple (non-split) generator.. This protocol covers all implemented non-split generators.

6.71 swarm.random.SimpleRandomGeneratorC

public interface *SimpleRandomGeneratorC*
implements BasicRandomGeneratorS,BasicRandomGeneratorC,SimpleGeneratorS,SimpleGeneratorC

A Simple (non-split) generator.. This protocol covers all implemented non-split generators.

6.72 swarm.random.SimpleRandomGeneratorS

public interface *SimpleRandomGeneratorS*
implements BasicRandomGeneratorS,SimpleGeneratorS

A Simple (non-split) generator.. This protocol covers all implemented non-split generators.

6.73 swarm.random.SplitGenerator

public interface *SplitGenerator*

Internal.

6.73.1 Methods

public Object ***advanceAll()***

The advanceAll method resets the state of all the virtual generators to the start of their next segment.

public Object ***advanceGenerator****(int vGen)*

The advanceGenerator method resets the state of a virtual generator to the start of the next segment.

public long ***getCurrentCount****(int vGen)*

The getCurrentCount method returns the current count of the specified virtual generator (i.e. the number of variates delivered).

public long ***getCurrentSegment****(int vGen)*

The getCurrentSegment method returns the number of the current segment of the specified virtual generator.

public double ***getDoubleSample****(int 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.

public float ***getFloatSample****(int 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.

public int ***getNumGenerators()***

The getNumGenerators method returns the current number of virtual generators (A).

public int ***getNumSegments()***

The getNumSegments method returns log2(the current number of segments) = v.

public int ***getSegmentLength()***

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

public double ***getThinDoubleSample****(int 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.

public int ***getUnsignedSample****(int vGen)*

The getUnsignedSample method returns a random unsigned integer uniformly distributed over the interval [0,unsignedMax] from virtual generator (data stream) vGen.

public Object ***jumpAllToSegment****(long seg)*

The jumpAlltoSegment: method resets the state of all the virtual generators to the start of the specified segment.

public Object ***jumpGenerator$toSegment****(int vGen, long seg)*

The jumpGenerator:toSegment: method resets the state of a virtual generator to the start of the specified segment.

public Object ***restartAll()***

The restartAll method resets the state of all the virtual generators to the start of their current segment.

public Object ***restartGenerator****(int vGen)*

The restartGenerator method resets the state of a virtual generator to the start of the current segment.

6.74 swarm.random.SplitGeneratorC

---

public interface *SplitGeneratorC*

Internal.

6.74.1 Methods

public Object
create$setA$setV$setW$setStateFromSeed*(swarm.defobj.Zone aZone, int A, int v, int w, int seed)*

6.75 swarm.random.SplitGeneratorS

public interface *SplitGeneratorS*

Internal.

6.75.1 Methods

public Object ***initAll()***

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

public Object ***initGenerator****(int vGen)*

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

6.76 swarm.random.UniformDoubleDist

public interface *UniformDoubleDist*
implements DoubleDistributionS,DoubleDistribution

Uniform Double Distribution. 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).

6.76.1 Methods

public double ***getDoubleMax****()*

The getDoubleMax method returns the maximum floating point value in the specified range.

public double ***getDoubleMin****()*

The getDoubleMin method returns the minimum floating point value in the specified range.

public double ***getDoubleWithMin$withMax****(double minValue, double maxValue)*

The getDoubleWithMin:withMax: method returns a floating point value within the range [min, max).

6.77 swarm.random.UniformDoubleDistC

---

public interface *UniformDoubleDistC*
implements DoubleDistributionS,DoubleDistributionC

Uniform Double Distribution. 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).

6.77.1 Methods

public Object
create$setGenerator$setDoubleMin$setMax*(swarm.defobj.Zone aZone, java.lang.Object generator, double minValue, double maxValue)*

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

public Object ***create$setGenerator$setVirtualGenerator$setDoubleMin$setMax****(swarm.defobj.Zone aZone, java.lang.Object generator, int vGen, double minValue, double maxValue)*

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

6.77.2 Implementing class: swarm.random.UniformDoubleDistCImpl

java.lang.Object

swarm.PhaseCImpl

---

public *UniformDoubleDistCImpl*
extends PhaseCImpl
implements DoubleDistributionS,DoubleDistributionC,UniformDoubleDistS,UniformDoubleDistC

6.77.3 Implementing class: swarm.random.UniformDoubleDistImpl

java.lang.Object

---

public *UniformDoubleDistImpl*
extends Object
implements DoubleDistributionS,DoubleDistribution,UniformDoubleDistS,UniformDoubleDist

6.78 swarm.random.UniformDoubleDistS

public interface *UniformDoubleDistS*
implements DoubleDistributionS

Uniform Double Distribution. 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).

6.78.1 Methods

public Object ***setDoubleMin$setMax****(double minValue, double maxValue)*

The setDoubleMin:setMax method sets the minimum and maximum floating point values of the distribution.

6.79 swarm.random.UniformIntegerDist

public interface *UniformIntegerDist*
implements IntegerDistributionS,IntegerDistribution

Uniform Integer Distribution. 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).

6.79.1 Methods

*public int **getIntegerMax**()*

The getIntegerMax method returns the maximum integer value.

*public int **getIntegerMin**()*

The getIntegerMin method returns the minimum integer value.

*public int **getIntegerWithMin$withMax**(int minValue, int maxValue)*

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

6.80 swarm.random.UniformIntegerDistC

public interface *UniformIntegerDistC*
implements IntegerDistributionS,IntegerDistributionC

Uniform Integer Distribution. 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).

6.80.1 Methods

public Object ***create$setGenerator$setIntegerMin$setMax****(swarm.defobj.Zone aZone, java.lang.Object generator, int minValue, int maxValue)*

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

public Object ***create$setGenerator$setVirtualGenerator$setIntegerMin$setMax****(swarm.defobj.Zone aZone, java.lang.Object generator, int vGen, int minValue, int maxValue)*

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

6.80.2 Implementing class: swarm.random.UniformIntegerDistCImpl

java.lang.Object

swarm.PhaseCImpl

public *UniformIntegerDistCImpl*
extends PhaseCImpl
implements IntegerDistributionS,IntegerDistributionC,UniformIntegerDistS,UniformIntegerDistC

6.80.3 Implementing class: swarm.random.UniformIntegerDistImpl

java.lang.Object

public *UniformIntegerDistImpl*
extends Object
implements IntegerDistributionS,IntegerDistribution,UniformIntegerDistS,UniformIntegerDist

6.81 swarm.random.UniformIntegerDistS

public interface *UniformIntegerDistS*
implements IntegerDistributionS

Uniform Integer Distribution. 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).

6.81.1 Methods

public Object ***setIntegerMin$setMax****(int minValue, int maxValue)*

The setIntegerMin:setMax: method sets the minimum and maximum integer values to be returned

6.82 swarm.random.UniformUnsignedDist

public interface *UniformUnsignedDist*
implements UnsignedDistributionS,UnsignedDistribution

Uniform Unsigned Distribution. 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).

6.82.1 Methods

public int ***getUnsignedMax()***

The getUnsignedMax method returns the maximum unsigned value.

public int ***getUnsignedMin()***

The getUnsignedMin method returns the minimum unsigned value.

public int ***getUnsignedWithMin$withMax****(int minVal, int maxVal)*

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

6.83 swarm.random.UniformUnsignedDistC

public interface *UniformUnsignedDistC*
implements UnsignedDistributionS,UnsignedDistributionC

Uniform Unsigned Distribution. 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).

6.83.1 Methods

public Object ***create$setGenerator$setUnsignedMin$setMax****(swarm.defobj.Zone aZone, java.lang.Object generator, int minValue, int maxValue)*

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

public Object ***create$setGenerator$setVirtualGenerator$setUnsignedMin$setMax****(swarm.defobj.Zone aZone, java.lang.Object generator, int vGen, int minValue, int maxValue)*

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

6.83.2 Implementing class: swarm.random.UniformUnsignedDistCImpl

java.lang.Object

swarm.PhaseCImpl

public *UniformUnsignedDistCImpl*
extends PhaseCImpl
implements UnsignedDistributionS,UnsignedDistributionC,UniformUnsignedDistS,UniformUnsignedDistC

6.83.3 Implementing class: swarm.random.UniformUnsignedDistImpl

java.lang.Object

public *UniformUnsignedDistImpl*
extends Object
implements UnsignedDistributionS,UnsignedDistribution,UniformUnsignedDistS,UniformUnsignedDist

6.84 swarm.random.UniformUnsignedDistS

public interface *UniformUnsignedDistS*
implements UnsignedDistributionS

Uniform Unsigned Distribution. 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).

6.84.1 Methods

public Object ***setUnsignedMin$setMax****(int minValue, int maxValue)*

The setUnsignedMin:setMax: method sets the minimum and maximum unsigned values to be returned

6.85 swarm.random.UnsignedDistribution

public interface *UnsignedDistribution*
implements ProbabilityDistributionS,ProbabilityDistribution

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

6.85.1 Methods

public int ***getUnsignedSample****()*

The getUnsignedSample method returns a non-negative integer sample value.

6.86 swarm.random.UnsignedDistributionC

public interface *UnsignedDistributionC*
implements ProbabilityDistributionS,ProbabilityDistributionC

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

6.87 swarm.random.UnsignedDistributionS

public interface *UnsignedDistributionS*
implements ProbabilityDistributionS

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

Chapter 7

Package swarm.simtools

Interfaces in package:

Classes in package:

7.1 swarm.simtools.NSelect

public interface *NSelect*
implements SwarmObjectS,SwarmObject

A class to select exactly N elements at random from a collection.. NSelect selects exactly N elements from a collection without repetition. A target collection must be provided.

7.1.1 Methods

public void ***select$from$into****(int n, java.lang.Object aCollection, java.lang.Object bCollection)*

The select:from:into: method selects exactly N elements from a collection without repetition into another collection. The selection algorithm is from Knuth.

7.2 swarm.simtools.NSelectC

---

public interface *NSelectC*
implements SwarmObjectS,SwarmObjectC

A class to select exactly N elements at random from a collection.. NSelect selects exactly N elements from a collection without repetition. A target collection must be provided.

7.2.1 Implementing class: swarm.simtools.NSelectCImpl

java.lang.Object

swarm.PhaseCImpl

---

public *NSelectCImpl*
extends PhaseCImpl
implements SwarmObjectS,SwarmObjectC,NSelectS,NSelectC

7.2.2 Implementing class: swarm.simtools.NSelectImpl

java.lang.Object

---

public *NSelectImpl*
extends Object
implements SwarmObjectS,SwarmObject,NSelectS,NSelect

7.3 swarm.simtools.NSelectS

public interface *NSelectS*
implements SwarmObjectS

A class to select exactly N elements at random from a collection.. NSelect selects exactly N elements from a collection without repetition. A target collection must be provided.

7.4 swarm.simtools.QSort

---

public interface *QSort*
implements SwarmObjectS,SwarmObject

A class to sort a collection.. 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.

7.4.1 Methods

public void ***reverseOrderOf****(java.lang.Object 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.

public void ***sortNumbersIn****(java.lang.Object 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.

public void ***sortObjectsIn****(java.lang.Object 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.

public void ***sortObjectsIn$using****(java.lang.Object aCollection, swarm.Selector aSelector)*

The sortObjectsIn:using: method will sort the objects in the collection with the specified comparison function for the object.

7.5 swarm.simtools.QSortC

---

public interface *QSortC*
implements SwarmObjectS,SwarmObjectC

A class to sort a collection.. 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.

7.5.1 Implementing class: swarm.simtools.QSortCImpl

java.lang.Object

swarm.PhaseCImpl

---

public *QSortCImpl*
extends PhaseCImpl
implements SwarmObjectS,SwarmObjectC,QSortS,QSortC

7.5.2 Implementing class: swarm.simtools.QSortImpl

java.lang.Object

---

public *QSortImpl*
extends Object
implements SwarmObjectS,SwarmObject,QSortS,QSort

7.6 swarm.simtools.QSortS

---

public interface *QSortS*
implements SwarmObjectS

A class to sort a collection.. 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.

7.7 swarm.simtools.UName

public interface *UName*
implements SwarmObjectS,SwarmObject

A class used to generate unique names (e.g. "critter1", "critter2" etc.). 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!

7.7.1 Methods

public String ***getNewName()***

The getNewName method generates a new name as a character string.

public Object ***getNewNameObject()***

The getNewNameObject generates a new name as a String Object.

public Object ***resetCounter()***

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

7.8 swarm.simtools.UNameC

---

public interface *UNameC*
implements SwarmObjectS,SwarmObjectC

A class used to generate unique names (e.g. "critter1", "critter2" etc.). 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!

7.8.1 Methods

public Object ***create$setBaseName****(swarm.defobj.Zone aZone, java.lang.String 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.

public Object ***create$setBaseNameObject****(swarm.defobj.Zone aZone, java.lang.Object 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.

public Object ***setBaseName****(java.lang.String aString)*

The setBaseName: method is used to set the base name given a const char *.

public Object ***setBaseNameObject****(java.lang.Object aStringObject)*

The setBaseNameObject: method is used to set the base name given an object of class String.

7.8.2 Implementing class: swarm.simtools.UNameCImpl

java.lang.Object

swarm.PhaseCImpl

---

public *UNameCImpl*
extends PhaseCImpl
implements SwarmObjectS,SwarmObjectC,UNameS,UNameC

7.8.3 Implementing class: swarm.simtools.UNameImpl

java.lang.Object

---

public *UNameImpl*
extends Object
implements SwarmObjectS,SwarmObject,UNameS,UName

7.9 swarm.simtools.UNameS

public interface *UNameS*
implements SwarmObjectS

A class used to generate unique names (e.g. "critter1", "critter2" etc.). 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!

Chapter 8

Package swarm.gui

Interfaces in package:

Classes in package:

- 8.8.1 RasterCImpl (page 471)
- 8.8.2 RasterImpl (page 471)
- 8.14.1 ZoomRasterCImpl (page 479)
- 8.14.2 ZoomRasterImpl (page 479)

8.1 swarm.gui.ArchivedGeometryWidget

public interface *ArchivedGeometryWidget*
implements WidgetS,Widget

Base class for widgets that archive geometry.. Subclasses of this class inherit the ability to archive their window geometry. This class also provides an interface to destroy notification.

8.1.1 Methods

*public void **drop()***

*public Object **updateArchiver**(java.lang.Object archiver)*

8.2 swarm.gui.ArchivedGeometryWidgetC

public interface *ArchivedGeometryWidgetC*
implements WidgetS,WidgetC

Base class for widgets that archive geometry.. Subclasses of this class inherit the ability to archive their window geometry. This class also provides an interface to destroy notification.

8.2.1 Methods

public Object
create$setWindowGeometryRecordName*(swarm.defobj.Zone aZone, java.lang.String name)*

public Object ***loadWindowGeometryRecord****()*

public Object ***registerAndLoad****()*

public Object ***setSaveSizeFlag****(boolean saveSizeFlag)*

Determines whether or not size is saved in addition to position.

public Object ***setWindowGeometryRecordName****(java.lang.String recordName)*

Called to set a name for archiving.

8.3 swarm.gui.ArchivedGeometryWidgetS

public interface *ArchivedGeometryWidgetS*
implements WidgetS

Base class for widgets that archive geometry.. Subclasses of this class inherit the ability to archive their window geometry. This class also provides an interface to destroy notification.

8.4 swarm.gui.Colormap

public interface *Colormap*
implements CreateS,Create

An class for creating a color palette for use with a Raster.. 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.

8.4.1 Methods

public boolean ***setColor$ToGrey****(byte c, double g)*

Add a color of a certain level of grey.

public boolean ***setColor$ToName****(byte c, java.lang.String colorName)*

Add color index 'c' looking up the color name in the color database.

public boolean ***setColor$ToRed$Green$Blue****(byte c, double r, double g, double b)*

Add color index 'c' to the color map, using a certain percent of red, green, and blue.

public Object ***unsetColor****(byte c)*

Remove color at index 'c' from the color map.

8.5 swarm.gui.ColormapC

public interface *ColormapC*
implements CreateS,CreateC

An class for creating a color palette for use with a Raster.. 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.

8.5.1 Implementing class: swarm.gui.ColormapCImpl

java.lang.Object

swarm.PhaseCImpl

public *ColormapCImpl*
extends PhaseCImpl
implements CreateS,CreateC,ColormapS,ColormapC

8.5.2 Implementing class: swarm.gui.ColormapImpl

java.lang.Object

public *ColormapImpl*
extends Object
implements CreateS,Create,ColormapS,Colormap

8.6 swarm.gui.ColormapS

---

public interface *ColormapS*
implements CreateS

An class for creating a color palette for use with a Raster.. 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.

8.7 swarm.gui.Raster

public interface *Raster*
implements ArchivedGeometryWidgetS,ArchivedGeometryWidget

A two dimension color display class.. 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.

8.7.1 Methods

*public Object **draw$X$Y**(java.lang.Object drawer, int x, int y)*

Draw an object at a given position.

*public Object **drawPointXYColor**(int x, int y, byte c)*

Draw a point at the given coordinates with the given color.

*public Object **drawSelf**()*

Draw the raster to the display.

*public Object **ellipseX0$Y0$X1$Y1$Width$Color**(int x0, int y0, int x1, int y1, int penWidth, byte c)*

Draw an ellipse of given geometry, pen width, and color.

*public Object **erase**()*

Erase the raster.

*public Object **fillRectangleX0$Y0$X1$Y1$Color**(int x0, int y0, int x1, int y1, byte color)*

Fill a rectangle of given geometry and color.

*public Object **lineX0$Y0$X1$Y1$Width$Color**(int x0, int y0, int x1, int y1, int penWidth, byte c)*

Draw a line of given geometry, pen width, and color.

*public Object **rectangleX0$Y0$X1$Y1$Width$Color**(int x0, int y0, int x1, int y1, int penWidth, byte c)*

Draw a rectangle of given geometry, pen width, and color.

public Object ***setButton$Client$Message****(int n, java.lang.Object c, swarm.Selector sel)*

Configure at mouse button to send a message to a given client object.

public Object ***setColormap****(swarm.gui.Colormap c)*

Set the palette for this raster.

public Object ***setWidth$Height****(int newWidth, int newHeight)*

Set the size of the raster.

8.8 swarm.gui.RasterC

public interface *RasterC*
implements ArchivedGeometryWidgetS,ArchivedGeometryWidgetC

A two dimension color display class.. 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.

8.8.1 Implementing class: swarm.gui.RasterCImpl

java.lang.Object

swarm.PhaseCImpl

public *RasterCImpl*
extends PhaseCImpl
implements ArchivedGeometryWidgetS,ArchivedGeometryWidgetC,RasterS,RasterC

8.8.2 Implementing class: swarm.gui.RasterImpl

java.lang.Object

public *RasterImpl*
extends Object
implements ArchivedGeometryWidgetS,ArchivedGeometryWidget,RasterS,Raster

8.9 swarm.gui.RasterS

public interface *RasterS*
implements ArchivedGeometryWidgetS

A two dimension color display class.. 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.

8.10 swarm.gui.Widget

public interface *Widget*
implements CreateS,Create,DropS,Drop

Widget base class.. 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.

8.10.1 Methods

*public Object **disableDestroyNotification**()*

Prevent calling the destroy notification method.

*public void **drop**()*

public Object
***enableDestroyNotification$notificationMethod**(java.lang.Object notificationTarget, swarm.Selector destroyNotificationMethod)*

Call a method if we are destroyed.

*public boolean **getDestroyedFlag**()*

*public int **getHeight**()*

Get the height of the widget.

*public Object **getParent**()*

Get the containing window of the widget.

*public Object **getTopLevel**()*

Get top level frame

*public String **getWidgetName**()*

Get the widget name.

*public int **getWidth**()*

Get the widget the widget.

public String ***getWindowGeometry****()*

public int ***getX****()*

Get the X position of the widget.

public int ***getY****()*

Get the Y position of the widget.

public Object ***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.

public Object ***packBeforeAndFillLeft$expand****(java.lang.Object widget, boolean expandFlag)*

public Object ***packFill****()*

public Object ***packFillLeft****(boolean expandFlag)*

public Object ***packForgetAndExpand****()*

public Object ***packToRight****(java.lang.Object widget)*

public Object ***setActiveFlag****(boolean activeFlag)*

Enable or disable the widget.

public Object ***setHeight****(int height)*

Set the height of the widget.

public Object ***setWidth****(int width)*

Set the width of the widget.

public Object ***setWidth$Height****(int width, int height)*

Set the width and height of the widget.

public Object ***setWindowGeometry****(java.lang.String s)*

public Object ***setWindowTitle****(java.lang.String title)*
Set the title on the widget.

public Object ***setX$Y****(int x, int y)*
Set the position of the widget.

8.11 swarm.gui.WidgetC

public interface *WidgetC*
implements CreateS,CreateC,DropS,DropC

Widget base class.. 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.

8.11.1 Methods

public Object ***createParent****(java.lang.Object 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

public String ***makeWidgetNameFor****(java.lang.Object widget)*

Compute the widget name for a component widget.

public Object ***setParent****(java.lang.Object parent)*

Set the containing window of the widget.

public Object ***setWidgetNameFromParent****(java.lang.Object parent)*

Set the widget name using the parent as context.

public Object ***setWidgetNameFromParentName****(java.lang.String parentWidgetName)*

Set the widget name using a hypothetical parent name.

8.12 swarm.gui.WidgetS

---

public interface *WidgetS*
implements CreateS,DropS

Widget base class.. 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.

8.13 swarm.gui.ZoomRaster

public interface *ZoomRaster*
implements RasterS,Raster

A zoomable Raster.. ZoomRaster is a subclass of Raster that implements a zoomable image. It handles translation between logical coordinates and screen coordinates.

8.13.1 Methods

public Object ***decreaseZoom()***

Make the raster smaller.

public int ***getZoomFactor()***

Get the current zoom factor.

public Object ***handleConfigureWidth$Height****(int newWidth, int newHeight)*

Reconfigures the ZoomRaster when the window is resized.

public Object ***increaseZoom()***

Make the raster bigger.

public Object ***setZoomFactor****(int z)*

Set the zoom factor.

8.14 swarm.gui.ZoomRasterC

public interface *ZoomRasterC*
implements RasterS,RasterC

A zoomable Raster.. ZoomRaster is a subclass of Raster that implements a zoomable image. It handles translation between logical coordinates and screen coordinates.

8.14.1 Implementing class: swarm.gui.ZoomRasterCImpl

java.lang.Object

swarm.PhaseCImpl

public *ZoomRasterCImpl*
extends PhaseCImpl
implements RasterS,RasterC,ZoomRasterS,ZoomRasterC

8.14.2 Implementing class: swarm.gui.ZoomRasterImpl

java.lang.Object

public *ZoomRasterImpl*
extends Object
implements RasterS,Raster,ZoomRasterS,ZoomRaster

8.15 swarm.gui.ZoomRasterS

public interface *ZoomRasterS*
implements RasterS

A zoomable Raster.. ZoomRaster is a subclass of Raster that implements a zoomable image. It handles translation between logical coordinates and screen coordinates.

Chapter 9

Package swarm.simtoolsgui

Interfaces in package:

Classes in package:

- 9.25.1 ProbeDisplayManagerCImpl (page 510)
- 9.25.2 ProbeDisplayManagerImpl (page 510)
- 9.29.2 SimpleProbeDisplayCImpl (page 514)
- 9.29.3 SimpleProbeDisplayImpl (page 514)

9.1 swarm.simtoolsgui.ActionCache

---

public interface *ActionCache*

A class to manage threads and Swarms.. 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.

9.1.1 Methods

public Object ***deliverActions****()*

public Object ***doTkEvents****()*

public void ***drop****()*

public Object ***getPanel****()*

public Object ***insertAction****(java.lang.Object actionHolder)*

public Object ***sendActionOfType$toExecute****(swarm.defobj.Symbol type, java.lang.String cmd)*

public Object ***sendNextAction****()*

public Object ***sendQuitAction****()*

public Object ***sendStartAction****()*

public Object ***sendStepAction****()*

public Object ***sendStopAction****()*

public Object ***setScheduleContext****(swarm.objectbase.Swarm context)*

public Object ***verifyActions****()*

public Object ***waitForControlEvent***()

9.2 swarm.simtoolsgui.ActionCacheC

---

public interface *ActionCacheC*

A class to manage threads and Swarms.. 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.

9.2.1 Methods

public Object ***createEnd***()

public Object ***createProcCtrl***()

public Object ***setControlPanel***(*swarm.simtoolsgui.ControlPanel cp*)

9.2.2 Implementing class: swarm.simtoolsgui.ActionCacheCImpl

java.lang.Object

swarm.PhaseCImpl

---

public *ActionCacheCImpl*
extends PhaseCImpl
implements ActionCacheS,ActionCacheC

9.2.3 Implementing class: swarm.simtoolsgui.ActionCacheImpl

java.lang.Object

---

public *ActionCacheImpl*
extends Object
implements ActionCacheS,ActionCache

9.3 swarm.simtoolsgui.ActionCacheS

public interface *ActionCacheS*

A class to manage threads and Swarms.. 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.

9.4 swarm.simtoolsgui.CommonProbeDisplay

public interface *CommonProbeDisplay*

A protocol underlying ProbeDisplay and CompleteProbeDisplay. This protocol provides the common interface to all kinds of ProbeDisplays.

9.4.1 Methods

public boolean ***getMarkedForDropFlag****()*

public Object ***getTopLevel****()*

public Object ***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.

9.5 swarm.simtoolsgui.CommonProbeDisplayC

public interface *CommonProbeDisplayC*

A protocol underlying ProbeDisplay and CompleteProbeDisplay. This protocol provides the common interface to all kinds of ProbeDisplays.

9.6 swarm.simtoolsgui.CommonProbeDisplayS

public interface *CommonProbeDisplayS*

A protocol underlying ProbeDisplay and CompleteProbeDisplay. This protocol provides the common interface to all kinds of ProbeDisplays.

9.7 swarm.simtoolsgui.CompleteProbeDisplay

public interface *CompleteProbeDisplay*
implements SingleProbeDisplayS,SingleProbeDisplay

A class that generates a complete ProbeMap for an object.. 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)...

9.8 swarm.simtoolsgui.CompleteProbeDisplayC

public interface *CompleteProbeDisplayC*
implements SingleProbeDisplayS,SingleProbeDisplayC

A class that generates a complete ProbeMap for an object.. 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)...

9.8.1 Implementing class: swarm.simtoolsgui.CompleteProbeDisplayCImpl

java.lang.Object

swarm.PhaseCImpl

public *CompleteProbeDisplayCImpl*
extends PhaseCImpl
implements SingleProbeDisplayS,SingleProbeDisplayC,CompleteProbeDisplayS,CompleteProbeDisplayC

9.8.2 Implementing class: swarm.simtoolsgui.CompleteProbeDisplayImpl

java.lang.Object

public *CompleteProbeDisplayImpl*
extends Object
implements SingleProbeDisplayS,SingleProbeDisplay,CompleteProbeDisplayS,CompleteProbeDisplay

9.9 swarm.simtoolsgui.CompleteProbeDisplayS

public interface *CompleteProbeDisplayS*
implements SingleProbeDisplayS

A class that generates a complete ProbeMap for an object.. 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)...

9.10 swarm.simtoolsgui.ControlPanel

public interface *ControlPanel*
implements SwarmObjectS,SwarmObject

Class to control the top level SwarmProcess. 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.

9.10.1 Methods

public Symbol ***getState()***

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

public Object ***setState****(swarm.defobj.Symbol s)*

public Object ***setStateNextTime****()*

Stop the running activity, and then set state to 'ControlStateNextTime'.

public Object ***setStateQuit****()*

Terminate activities, and set state to 'ControlStateQuit'.

public Object ***setStateRunning****()*

Sets the state to 'running'.

public Object ***setStateSave****()*

Saves the objects that are registered for archiving.

public Object ***setStateStepping****()*

Stop the running activity, and then set state to 'ControlStateStepping'.

public Object ***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).

public Object ***startInActivity****(swarm.activity.SwarmActivity activityID)*

9.11 swarm.simtoolsgui.ControlPanelC

---

public interface *ControlPanelC*
implements SwarmObjectS,SwarmObjectC

Class to control the top level SwarmProcess. 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.

9.11.1 Methods

public Object ***createEnd***()

9.11.2 Implementing class: swarm.simtoolsgui.ControlPanelCImpl

java.lang.Object

swarm.PhaseCImpl

---

public *ControlPanelCImpl*
extends PhaseCImpl
implements SwarmObjectS,SwarmObjectC,ControlPanelS,ControlPanelC

9.11.3 Implementing class: swarm.simtoolsgui.ControlPanelImpl

java.lang.Object

---

public *ControlPanelImpl*
extends Object
implements SwarmObjectS,SwarmObject,ControlPanelS,ControlPanel

9.12 swarm.simtoolsgui.ControlPanelS

public interface *ControlPanelS*
implements SwarmObjectS

Class to control the top level SwarmProcess. 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.

9.13 swarm.simtoolsgui.GUIComposite

public interface *GUIComposite*

Base class for objects that use several GUI components.. Base class for objects that use several GUI components.

9.13.1 Methods

public Object ***disableDestroyNotification****()*

public Object
enableDestroyNotification$notificationMethod*(java.lang.Object notificationTarget, swarm.Selector notificationMethod)*

9.14 swarm.simtoolsgui.GUICompositeC

public interface *GUICompositeC*

Base class for objects that use several GUI components.. Base class for objects that use several GUI components.

9.15 swarm.simtoolsgui.GUICompositeS

public interface *GUICompositeS*

Base class for objects that use several GUI components.. Base class for objects that use several GUI components.

9.16 swarm.simtoolsgui.GUISwarm

public interface *GUISwarm*
implements SwarmS,Swarm

A version of the Swarm class which is graphics aware. . 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.

9.16.1 Methods

public void ***drop()***

public ActionCache ***getActionCache()***

public ControlPanel ***getControlPanel()***

public Object ***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).

9.17 swarm.simtoolsgui.GUISwarmC

public interface *GUISwarmC*
implements SwarmS,SwarmC

A version of the Swarm class which is graphics aware. . 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.

9.17.1 Implementing class: swarm.simtoolsgui.GUISwarmCImpl

java.lang.Object

swarm.PhaseCImpl

public *GUISwarmCImpl*
extends PhaseCImpl
implements SwarmS,SwarmC,GUISwarmS,GUISwarmC

9.17.2 Implementing class: swarm.simtoolsgui.GUISwarmImpl

java.lang.Object

public *GUISwarmImpl*
extends Object
implements SwarmS,Swarm,GUISwarmS,GUISwarm

9.18 swarm.simtoolsgui.GUISwarmS

public interface *GUISwarmS*
implements SwarmS

A version of the Swarm class which is graphics aware. . 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.

9.19 swarm.simtoolsgui.MultiVarProbeDisplay

public interface *MultiVarProbeDisplay*
implements CommonProbeDisplayS,CommonProbeDisplay

A display for displaying a ProbeMap across a number of objects.. 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.

9.19.1 Methods

public void ***drop()***

9.20 swarm.simtoolsgui.MultiVarProbeDisplayC

public interface *MultiVarProbeDisplayC*
implements CommonProbeDisplayS,CommonProbeDisplayC

A display for displaying a ProbeMap across a number of objects.. 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.

9.20.1 Methods

public Object ***createBegin****(swarm.defobj.Zone aZone)*

public Object ***createEnd****()*

public Object ***setObjectList****(swarm.collections.List objectList)*

Sets the list of objects to display.

public Object ***setObjectNameSelector****(swarm.Selector objectNameSelector)*

Sets the selector to send for labeling the object.

public Object ***setProbeMap****(swarm.objectbase.ProbeMap probeMap)*

Sets the probe map (i.e. list of fields) to display.

9.20.2 Implementing class: swarm.simtoolsgui.MultiVarProbeDisplayCImpl

java.lang.Object

swarm.PhaseCImpl

public *MultiVarProbeDisplayCImpl*
extends PhaseCImpl
implements CommonProbeDisplayS,CommonProbeDisplayC,MultiVarProbeDisplayS,MultiVarProbeDisplayC

9.20.3 Implementing class: swarm.simtoolsgui.MultiVarProbeDisplayImpl

java.lang.Object

public *MultiVarProbeDisplayImpl*
extends Object
implements CommonProbeDisplayS,CommonProbeDisplay,MultiVarProbeDisplayS,MultiVarProbeDisplay

9.21 swarm.simtoolsgui.MultiVarProbeDisplayS

public interface *MultiVarProbeDisplayS*
implements CommonProbeDisplayS

A display for displaying a ProbeMap across a number of objects.. 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.

9.22 swarm.simtoolsgui.ProbeDisplay

public interface *ProbeDisplay*
implements SingleProbeDisplayS,SingleProbeDisplay

A class to display ProbeMaps. A class which generates a GUI to a ProbeMap of probes applied to a given target object.

9.22.1 Methods

public ProbeMap ***getProbeMap****()*

Gets the probedMap.

9.23 swarm.simtoolsgui.ProbeDisplayC

public interface *ProbeDisplayC*
implements SingleProbeDisplayS,SingleProbeDisplayC

A class to display ProbeMaps. A class which generates a GUI to a ProbeMap of probes applied to a given target object.

9.23.1 Methods

public Object ***createEnd****()*

public Object ***setProbeMap****(swarm.objectbase.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.

9.23.2 Implementing class: swarm.simtoolsgui.ProbeDisplayCImpl

java.lang.Object

swarm.PhaseCImpl

public *ProbeDisplayCImpl*
extends PhaseCImpl
implements SingleProbeDisplayS,SingleProbeDisplayC,ProbeDisplayS,ProbeDisplayC

9.23.3 Implementing class: swarm.simtoolsgui.ProbeDisplayImpl

java.lang.Object

public *ProbeDisplayImpl*
extends Object
implements SingleProbeDisplayS,SingleProbeDisplay,ProbeDisplayS,ProbeDisplay

9.24 swarm.simtoolsgui.ProbeDisplayManager

public interface *ProbeDisplayManager*
implements SwarmObjectS,SwarmObject

The ProbeDisplay manager.. A (singleton) class whose instance is used to manage all the ProbeDisplays created by the user during a GUI run of the simulation.

9.24.1 Methods

public Object ***addProbeDisplay****(java.lang.Object probeDisplay)*

Add a probe display to be managed by the ProbeDisplayManager.

public CompleteProbeDisplay ***createArchivedCompleteProbeDisplayFor$variableName****(java.lang.Object anObject, java.lang.String variableName)*

public ProbeDisplay ***createArchivedDefaultProbeDisplayFor$variableName****(java.lang.Object anObject, java.lang.String variableName)*

public ProbeDisplay ***createArchivedProbeDisplayFor$variableName****(java.lang.Object anObject, java.lang.String variableName)*

public CompleteProbeDisplay ***createCompleteProbeDisplayFor****(java.lang.Object anObject)*

public ProbeDisplay ***createDefaultProbeDisplayFor****(java.lang.Object anObject)*

public ProbeDisplay ***createProbeDisplayFor****(java.lang.Object anObject)*

public Object ***dropProbeDisplaysFor****(java.lang.Object anObject)*

Remove and drop probe displays associated with a given object.

public boolean ***getDropImmediatelyFlag****()*

public Object ***removeProbeDisplay****(java.lang.Object probeDisplay)*

Remove a probe display from management by the ProbeDisplayManager.

public Object ***setDropImmediatelyFlag****(boolean dropImmediateFlag)*

public Object ***update****()*

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

9.25 swarm.simtoolsgui.ProbeDisplayManagerC

public interface *ProbeDisplayManagerC*
implements SwarmObjectS,SwarmObjectC

The ProbeDisplay manager.. A (singleton) class whose instance is used to manage all the ProbeDisplays created by the user during a GUI run of the simulation.

9.25.1 Implementing class: swarm.simtoolsgui.ProbeDisplayManagerCImpl

java.lang.Object

swarm.PhaseCImpl

public *ProbeDisplayManagerCImpl*
extends PhaseCImpl
implements SwarmObjectS,SwarmObjectC,ProbeDisplayManagerS,ProbeDisplayManagerC

9.25.2 Implementing class: swarm.simtoolsgui.ProbeDisplayManagerImpl

java.lang.Object

public *ProbeDisplayManagerImpl*
extends Object
implements SwarmObjectS,SwarmObject,ProbeDisplayManagerS,ProbeDisplayManager

9.26 swarm.simtoolsgui.ProbeDisplayManagerS

public interface *ProbeDisplayManagerS*
implements SwarmObjectS

The ProbeDisplay manager.. A (singleton) class whose instance is used to manage all the ProbeDisplays created by the user during a GUI run of the simulation.

9.27 swarm.simtoolsgui.ProbeDisplayS

public interface *ProbeDisplayS*
implements SingleProbeDisplayS

A class to display ProbeMaps. A class which generates a GUI to a ProbeMap of probes applied to a given target object.

9.28 swarm.simtoolsgui.SimpleProbeDisplay

public interface *SimpleProbeDisplay*
implements SingleProbeDisplayS,SingleProbeDisplay
.

9.28.1 Methods

public ProbeMap ***getProbeMap***()

9.29 swarm.simtoolsgui.SimpleProbeDisplayC

public interface *SimpleProbeDisplayC*
implements SingleProbeDisplayS,SingleProbeDisplayC
.

9.29.1 Methods

*public Object **createEnd**()*

*public Object **setProbeMap**(java.lang.Object probeMap)*

9.29.2 Implementing class: swarm.simtoolsgui.SimpleProbeDisplayCImpl

java.lang.Object

swarm.PhaseCImpl

public *SimpleProbeDisplayCImpl*
extends PhaseCImpl
implements SingleProbeDisplayS,SingleProbeDisplayC,SimpleProbeDisplayS,SimpleProbeDisplayC

9.29.3 Implementing class: swarm.simtoolsgui.SimpleProbeDisplayImpl

java.lang.Object

public *SimpleProbeDisplayImpl*
extends Object
implements SingleProbeDisplayS,SingleProbeDisplay,SimpleProbeDisplayS,SimpleProbeDisplay

9.30 swarm.simtoolsgui.SimpleProbeDisplayS

public interface *SimpleProbeDisplayS*
implements SingleProbeDisplayS
.

9.31 swarm.simtoolsgui.SingleProbeDisplay

public interface *SingleProbeDisplay*
implements CommonProbeDisplayS,CommonProbeDisplay

An abstract protocol underlying single-object probe displays.. This protocol is common to CompleteProbeDisplay and ProbeDisplay.

9.31.1 Methods

public Object ***getProbedObject()***

Gets the probed object.

9.32 swarm.simtoolsgui.SingleProbeDisplayC

public interface *SingleProbeDisplayC*
implements CommonProbeDisplayS,CommonProbeDisplayC

An abstract protocol underlying single-object probe displays.. This protocol is common to CompleteProbeDisplay and ProbeDisplay.

9.32.1 Methods

*public Object **createEnd**()*

*public Object **setProbedObject**(java.lang.Object anObject)*

This method must be called.

9.33 swarm.simtoolsgui.SingleProbeDisplayS

public interface *SingleProbeDisplayS*
implements CommonProbeDisplayS

An abstract protocol underlying single-object probe displays.. This protocol is common to CompleteProbeDisplay and ProbeDisplay.

Chapter 10

Package swarm.space

Interfaces in package:

Classes in package:

10.1 swarm.space.Ca2d

public interface *Ca2d*
implements DblBuffer2dS,DblBuffer2d

Defines abstract protocol for cellular automata.. Inherits from DblBuffer2d, defines abstract protocol for cellular automata.

10.1.1 Methods

*public Object **stepRule**()*

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

10.2 swarm.space.Ca2dC

public interface *Ca2dC*
implements DblBuffer2dS,DblBuffer2dC

Defines abstract protocol for cellular automata.. Inherits from DblBuffer2d, defines abstract protocol for cellular automata.

10.2.1 Methods

public Object ***createEnd()***

Check that numStates has been set.

public Object ***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.

public Object ***setNumStates****(int n)*

Record the number of states the CA understands.

10.3 swarm.space.Ca2dS

public interface *Ca2dS*
implements DblBuffer2dS

Defines abstract protocol for cellular automata.. Inherits from DblBuffer2d, defines abstract protocol for cellular automata.

10.4 swarm.space.ConwayLife2d

public interface *ConwayLife2d*
implements Ca2dS,Ca2d

Classic 2d Conway's Life CA.. Classic 2d Conway's Life CA.

10.4.1 Methods

*public Object **stepRule**()*

Run Conway's Life rule (simpleminded version).

10.5 swarm.space.ConwayLife2dC

---

public interface *ConwayLife2dC*
implements Ca2dS,Ca2dC
Classic 2d Conway's Life CA.. Classic 2d Conway's Life CA.

10.5.1 Methods

public Object ***createBegin****(swarm.defobj.Zone aZone)*

Set number of states to 2.

public Object ***initializeLattice****()*

Initialize lattice to random 1/3 in state 1.

10.5.2 Implementing class: swarm.space.ConwayLife2dCImpl

java.lang.Object

swarm.PhaseCImpl

---

public *ConwayLife2dCImpl*
extends PhaseCImpl
implements Ca2dS,Ca2dC,ConwayLife2dS,ConwayLife2dC

10.5.3 Implementing class: swarm.space.ConwayLife2dImpl

java.lang.Object

---

public *ConwayLife2dImpl*
extends Object
implements Ca2dS,Ca2d,ConwayLife2dS,ConwayLife2d

10.6 swarm.space.ConwayLife2dS

public interface *ConwayLife2dS*
implements Ca2dS
Classic 2d Conway's Life CA.. Classic 2d Conway's Life CA.

10.7 swarm.space.DblBuffer2d

public interface *DblBuffer2d*
implements Discrete2dS,Discrete2d

A double buffered space.. 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.

10.7.1 Methods

public Object ***putObjectatXY****(java.lang.Object anObject, int x, int y)*

Overridden so writes happen to newLattice.

public Object ***putValueatXY****(int v, int x, int y)*

Overridden so writes happen to newLattice.

public Object ***updateLattice****()*

Copy newLattice to lattice, in effect updating the lattice.

10.8 swarm.space.DblBuffer2dC

public interface *DblBuffer2dC*
implements Discrete2dS,Discrete2dC

A double buffered space.. 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.

10.8.1 Methods

public Object ***create$setSizeX$Y****(swarm.defobj.Zone aZone, int x, int y)*

Convenience constructor for DblBuffer lattice

public Object ***createEnd****()*

Rewrites the method from Discrete2d. Allocate two lattices, makes the offsets.

10.8.2 Implementing class: swarm.space.DblBuffer2dCImpl

java.lang.Object

swarm.PhaseCImpl

public *DblBuffer2dCImpl*
extends PhaseCImpl
implements Discrete2dS,Discrete2dC,DblBuffer2dS,DblBuffer2dC

10.8.3 Implementing class: swarm.space.DblBuffer2dImpl

java.lang.Object

public *DblBuffer2dImpl*
extends Object
implements Discrete2dS,Discrete2d,DblBuffer2dS,DblBuffer2d

10.9 swarm.space.DblBuffer2dS

public interface *DblBuffer2dS*
implements Discrete2dS

A double buffered space.. 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.

10.10 swarm.space.Diffuse2d

public interface *Diffuse2d*
implements Ca2dS,Ca2d

2d difussion with evaporation.. Discrete 2nd order approximation to 2d diffusion with evaporation. Math is done in integers on the range [0,0x7fff].

10.10.1 Methods

public Object ***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.

10.11 swarm.space.Diffuse2dC

---

public interface *Diffuse2dC*
implements Ca2dS,Ca2dC
2d difussion with evaporation.. Discrete 2nd order approximation to 2d diffusion with evaporation. Math is done in integers on the range [0,0x7fff].

10.11.1 Methods

public Object ***create$setSizeX$Y$setDiffusionConstant$setEvaporationRate****(swarm.defobj.Zone aZone, int x, int y, double d, double e)*

Convenience constructor for Diffuse2d

public Object ***createBegin****(swarm.defobj.Zone aZone)*

Set diffusion constant and evaporation rate to 1.0, numStates to 0x7fff.

public Object ***initializeLattice****()*

Initialize world to 0.

10.11.2 Implementing class: swarm.space.Diffuse2dCImpl

java.lang.Object

swarm.PhaseCImpl

---

public *Diffuse2dCImpl*
extends PhaseCImpl
implements Ca2dS,Ca2dC,Diffuse2dS,Diffuse2dC

10.11.3 Implementing class: swarm.space.Diffuse2dImpl

java.lang.Object

---

public *Diffuse2dImpl*
extends Object
implements Ca2dS,Ca2d,Diffuse2dS,Diffuse2d

10.12 swarm.space.Diffuse2dS

public interface *Diffuse2dS*
implements Ca2dS

2d difussion with evaporation.. Discrete 2nd order approximation to 2d diffusion with evaporation. Math is done in integers on the range [0,0x7fff].

10.12.1 Methods

public Object ***setDiffusionConstant****(double d)*

Set the diffusion constant. Values over 1.0 might not be valid.

public Object ***setEvaporationRate****(double e)*

Set the evaporation rate. Values over 1.0 don't make much sense

10.13 swarm.space.Discrete2d

public interface *Discrete2d*
implements SwarmObjectS,SwarmObject

Root class of all 2d discrete spaces.. 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.

10.13.1 Methods

public Object ***copyDiscrete2d$toDiscrete2d****(swarm.space.Discrete2d a, swarm.space.Discrete2d b)*

This method copies the data in one Discrete2d object to another Discrete2d object. It assumes that both objects already exist.

public Object ***fastFillWithObject****(java.lang.Object anObj)*

Directly fills the lattice with an object.

public Object ***fastFillWithValue****(int aValue)*

Directly fills the lattice with a value.

public Object ***fillWithObject****(java.lang.Object anObj)*

Fills the space using putObject.

public Object ***fillWithValue****(int aValue)*

Fills the space using putValue.

public Object ***getObjectAtX$Y****(int x, int y)*

Return the pointer stored at (x,y).

public int ***getSizeX****()*

Get the size of the lattice in the X dimension.

public int ***getSizeY****()*

Get the size of the lattice in the Y dimension.

public int ***getValueAtX$Y****(int x, int y)*

Return the integer stored at (x,y).

public Object ***putObjectatXY****(java.lang.Object anObject, int x, int y)*

Put the given pointer to (x,y) overwriting whatever was there.

public Object ***putValueatXY****(int v, int x, int y)*

Put the given integer to (x,y) overwriting whatever was there.

public int ***setDiscrete2d$toFile****(swarm.space.Discrete2d a, java.lang.String filename)*

This method reads a PGM formatted file and pipes the data into a Discrete2d object.

10.14 swarm.space.Discrete2dC

---

public interface *Discrete2dC*
implements SwarmObjectS,SwarmObjectC

Root class of all 2d discrete spaces.. 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.

10.14.1 Methods

public Object ***create$setSizeX$Y****(swarm.defobj.Zone aZone, int x, int y)*

Convenience constructor for Discrete2d lattice

public Object ***createEnd****()*

Create the lattice, precompute the offsets based on Y coordinate.

public Object ***makeOffsets****()*

Given an array size, compute the offsets array that caches the multiplication by ysize. See the discrete2dSiteAt macro.

public Object ***setSizeX$Y****(int x, int y)*

Set the world size.

10.14.2 Implementing class: swarm.space.Discrete2dCImpl

java.lang.Object

swarm.PhaseCImpl

---

public *Discrete2dCImpl*
extends PhaseCImpl
implements SwarmObjectS,SwarmObjectC,Discrete2dS,Discrete2dC

10.14.3 Implementing class: swarm.space.Discrete2dImpl

java.lang.Object

---

public *Discrete2dImpl*
extends Object
implements SwarmObjectS,SwarmObject,Discrete2dS,Discrete2d

10.15 swarm.space.Discrete2dS

public interface *Discrete2dS*
implements SwarmObjectS

Root class of all 2d discrete spaces.. 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.

10.16 swarm.space.Grid2d

public interface *Grid2d*
implements Discrete2dS,Discrete2d

A 2d container class for agents.. 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.

10.16.1 Methods

*public Object **putObjectatXY**(java.lang.Object anObject, int x, int 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.

*public Object **setOverwriteWarnings**(boolean 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.

10.17 swarm.space.Grid2dC

---

public interface *Grid2dC*
implements Discrete2dS,Discrete2dC

A 2d container class for agents.. 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.

10.17.1 Methods

*public Object **create$setSizeX$Y**(swarm.defobj.Zone aZone, int x, int y)*

Convenience constructor for Grid2d

10.17.2 Implementing class: swarm.space.Grid2dCImpl

java.lang.Object

swarm.PhaseCImpl

---

public *Grid2dCImpl*
extends PhaseCImpl
implements Discrete2dS,Discrete2dC,Grid2dS,Grid2dC

10.17.3 Implementing class: swarm.space.Grid2dImpl

java.lang.Object

---

public *Grid2dImpl*
extends Object
implements Discrete2dS,Discrete2d,Grid2dS,Grid2d

10.18 swarm.space.Grid2dS

public interface *Grid2dS*
implements Discrete2dS

A 2d container class for agents.. 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.

10.19 swarm.space.Int2dFiler

public interface *Int2dFiler*
implements SwarmObjectS,SwarmObject

Saves the state of a Discrete2d object [DEPRECATED].. 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.

10.19.1 Methods

public Object ***fileTo****(java.lang.String aFileName)*

When the Int2dFiler receives this message, it opens a file called fileName, stores the state of a pre-specified space into it, and then closes the file.

public Object ***setBackground****(int aValue)*

This message is optional. It is used when the target Discrete2d contains objects. If a particular location in the space has no resident object, the argument of this message is the value which gets writtent to the file. The default background value is 0.

public Object ***setDiscrete2dToFile****(swarm.space.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).

public Object ***setValueMessage****(swarm.Selector 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.

10.20 swarm.space.Int2dFilerC

public interface *Int2dFilerC*
implements SwarmObjectS,SwarmObjectC

Saves the state of a Discrete2d object [DEPRECATED].. 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.

10.20.1 Implementing class: swarm.space.Int2dFilerCImpl

java.lang.Object

swarm.PhaseCImpl

public *Int2dFilerCImpl*
extends PhaseCImpl
implements SwarmObjectS,SwarmObjectC,Int2dFilerS,Int2dFilerC

10.20.2 Implementing class: swarm.space.Int2dFilerImpl

java.lang.Object

public *Int2dFilerImpl*
extends Object
implements SwarmObjectS,SwarmObject,Int2dFilerS,Int2dFiler

10.21 swarm.space.Int2dFilerS

public interface *Int2dFilerS*
implements SwarmObjectS

Saves the state of a Discrete2d object [DEPRECATED].. 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.

10.22 swarm.space.Object2dDisplay

---

public interface *Object2dDisplay*
implements SwarmObjectS,SwarmObject

Object2dDisplay displays 2d arrays of objects.. 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.

10.22.1 Methods

public Object ***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.

public Object ***makeProbeAtX$Y****(int x, int y)*

Make a probe for an object at a specific point.

public Object ***setObjectCollection****(java.lang.Object 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.

10.23 swarm.space.Object2dDisplayC

public interface *Object2dDisplayC*
implements SwarmObjectS,SwarmObjectC

Object2dDisplay displays 2d arrays of objects.. 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.

10.23.1 Methods

public Object ***create$setDisplayWidget$setDiscrete2dToDisplay$setDisplayMessage****(swarm.defobj.Zone aZone, swarm.gui.Raster r, swarm.space.Discrete2d c, swarm.Selector s)*

Convenience constructor for Object2dDisplay

public Object ***setDiscrete2dToDisplay****(swarm.space.Discrete2d c)*

Set the 2d array to draw.

public Object ***setDisplayMessage****(swarm.Selector s)*

Set the message to be sent to each object in the grid to make it draw itself.

public Object ***setDisplayWidget****(swarm.gui.Raster r)*

Set the display widget to use for drawing.

10.23.2 Implementing class: swarm.space.Object2dDisplayCImpl

java.lang.Object

swarm.PhaseCImpl

public *Object2dDisplayCImpl*
extends PhaseCImpl
implements SwarmObjectS,SwarmObjectC,Object2dDisplayS,Object2dDisplayC

10.23.3 Implementing class: swarm.space.Object2dDisplayImpl

java.lang.Object

---

public *Object2dDisplayImpl*
extends Object
implements SwarmObjectS,SwarmObject,Object2dDisplayS,Object2dDisplay

10.24 swarm.space.Object2dDisplayS

public interface *Object2dDisplayS*
implements SwarmObjectS

Object2dDisplay displays 2d arrays of objects.. 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.

10.25 swarm.space.Value2dDisplay

public interface *Value2dDisplay*
implements SwarmObjectS,SwarmObject

Value2dDisplay displays 2d arrays of values.. 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.

10.25.1 Methods

public Object ***display()***

Draw the array on the given widget. Note that you still have to tell the widget to draw itself afterwards. The code for display uses the fast macro access in Discrete2d on the cached return value from getLattice. It also caches the drawPointX:Y: method lookup on the display widget - this is a nice trick that you might want to look at.

public Object ***setDisplayMappingM$C****(int m, int c)*

Linear transform of states to colours for drawing. color = state / m + c If not set, assume m == 1 and c == 0.

10.26 swarm.space.Value2dDisplayC

public interface *Value2dDisplayC*
implements SwarmObjectS,SwarmObjectC

Value2dDisplay displays 2d arrays of values.. 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.

10.26.1 Methods

public Object ***create$setDisplayWidget$colormap$setDiscrete2dToDisplay****(swarm.defobj.Zone aZone, swarm.gui.Raster r, swarm.gui.Colormap c, swarm.space.Discrete2d d)*

Convenience constructor for Value2dDisplay

public Object ***createEnd****()*

public Object ***setDiscrete2dToDisplay****(swarm.space.Discrete2d c)*

Set which array to draw.

public Object ***setDisplayWidget$colormap****(swarm.gui.Raster r, swarm.gui.Colormap c)*

Set the display widget and the colourmap to use to draw the value array.

10.26.2 Implementing class: swarm.space.Value2dDisplayCImpl

java.lang.Object

swarm.PhaseCImpl

public *Value2dDisplayCImpl*
extends PhaseCImpl
implements SwarmObjectS,SwarmObjectC,Value2dDisplayS,Value2dDisplayC

10.26.3 Implementing class: swarm.space.Value2dDisplayImpl

java.lang.Object

public *Value2dDisplayImpl*
extends Object
implements SwarmObjectS,SwarmObject,Value2dDisplayS,Value2dDisplay

10.27 swarm.space.Value2dDisplayS

public interface *Value2dDisplayS*
implements SwarmObjectS

Value2dDisplay displays 2d arrays of values.. 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.

Chapter 11

Package swarm.analysis

Interfaces in package:

Classes in package:

11.1 swarm.analysis.ActiveGraph

---

public interface *ActiveGraph*
implements MessageProbeS,MessageProbe

Provides a continuous data feed between Swarm and the GUI.. 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.

11.1.1 Methods

public Object ***setDataFeed****(java.lang.Object d)*

Sets the object that will be probed for data.

public Object ***setElement****(java.lang.Object ge)*

Sets the graph element used to draw on.

public Object ***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.

11.2 swarm.analysis.ActiveGraphC

public interface *ActiveGraphC*
implements MessageProbeS,MessageProbeC

Provides a continuous data feed between Swarm and the GUI.. 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.

11.2.1 Implementing class: swarm.analysis.ActiveGraphCImpl

java.lang.Object

swarm.PhaseCImpl

public *ActiveGraphCImpl*
extends PhaseCImpl
implements MessageProbeS,MessageProbeC,ActiveGraphS,ActiveGraphC

11.2.2 Implementing class: swarm.analysis.ActiveGraphImpl

java.lang.Object

public *ActiveGraphImpl*
extends Object
implements MessageProbeS,MessageProbe,ActiveGraphS,ActiveGraph

11.3 swarm.analysis.ActiveGraphS

public interface *ActiveGraphS*
implements MessageProbeS

Provides a continuous data feed between Swarm and the GUI.. 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.

11.4 swarm.analysis.ActiveOutFile

public interface *ActiveOutFile*
implements MessageProbeS,MessageProbe

An object that actively updates its file stream when updated.. 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.

11.4.1 Methods

public Object ***setDataFeed****(java.lang.Object d)*

The setDataFeed: method sets the object that will be probed for data.

public Object ***setFileObject****(java.lang.Object aFileObj)*

The setFileObject: method sets the file object to which the data will be sent.

public Object ***step****()*

The step method fires the probe, reads the value from the object, and sends the value to the file.

11.5 swarm.analysis.ActiveOutFileC

public interface *ActiveOutFileC*
implements MessageProbeS,MessageProbeC

An object that actively updates its file stream when updated.. 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.

11.5.1 Implementing class: swarm.analysis.ActiveOutFileCImpl

java.lang.Object

swarm.PhaseCImpl

public *ActiveOutFileCImpl*
extends PhaseCImpl
implements MessageProbeS,MessageProbeC,ActiveOutFileS,ActiveOutFileC

11.5.2 Implementing class: swarm.analysis.ActiveOutFileImpl

java.lang.Object

public *ActiveOutFileImpl*
extends Object
implements MessageProbeS,MessageProbe,ActiveOutFileS,ActiveOutFile

11.6 swarm.analysis.ActiveOutFileS

public interface *ActiveOutFileS*
implements MessageProbeS

An object that actively updates its file stream when updated.. 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.

11.7 swarm.analysis.Averager

public interface *Averager*
implements MessageProbeS,MessageProbe

Averages together data, gives the data to whomever asks.. Averager objects read a value (via a MessageProbe) from a collection (typically a list) of objects and collect statistics over them.

11.7.1 Methods

public 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.

public int ***getCount()***

The getCount method returns the number of values the averager collects.

public 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.

public 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.

public Object ***update()***

The update method runs through the collection calling the selector on each object.

11.8 swarm.analysis.AveragerC

public interface *AveragerC*
implements MessageProbeS,MessageProbeC

Averages together data, gives the data to whomever asks.. Averager objects read a value (via a MessageProbe) from a collection (typically a list) of objects and collect statistics over them.

11.8.1 Methods

public Object ***createEnd****()*

public Object ***setCollection****(java.lang.Object aCollection)*

Sets the collection of objects that will be probed.

11.8.2 Implementing class: swarm.analysis.AveragerCImpl

java.lang.Object

swarm.PhaseCImpl

public *AveragerCImpl*
extends PhaseCImpl
implements MessageProbeS,MessageProbeC,AveragerS,AveragerC

11.8.3 Implementing class: swarm.analysis.AveragerImpl

java.lang.Object

public *AveragerImpl*
extends Object
implements MessageProbeS,MessageProbe,AveragerS,Averager

11.9 swarm.analysis.AveragerS

public interface *AveragerS*
implements MessageProbeS

Averages together data, gives the data to whomever asks.. Averager objects read a value (via a MessageProbe) from a collection (typically a list) of objects and collect statistics over them.

11.10 swarm.analysis.Entropy

public interface *Entropy*
implements MessageProbeS,MessageProbe

Computes entropy via a MessageProbe.. Entropy objects read probabilities (via a MessageProbe) from a collection of objects and calculate the entropy of the underlying distribution.

11.10.1 Methods

public double ***getEntropy()***

The getEntropy method returns the calculated Entropy. The entropy value is read out of the object, not computed everytime it is requested.

public Object ***update()***

The update method polls the collection and updates the entropy. This method should be scheduled prior to collecting the data using getEntropy.

11.11 swarm.analysis.EntropyC

public interface *EntropyC*
implements MessageProbeS,MessageProbeC

Computes entropy via a MessageProbe.. Entropy objects read probabilities (via a MessageProbe) from a collection of objects and calculate the entropy of the underlying distribution.

11.11.1 Methods

public Object ***createEnd***()

public Object ***setCollection***(*java.lang.Object aCollection*)

The setCollection method sets the collection of objects that will be probed.

11.11.2 Implementing class: swarm.analysis.EntropyCImpl

java.lang.Object

swarm.PhaseCImpl

public *EntropyCImpl*
extends PhaseCImpl
implements MessageProbeS,MessageProbeC,EntropyS,EntropyC

11.11.3 Implementing class: swarm.analysis.EntropyImpl

java.lang.Object

public *EntropyImpl*
extends Object
implements MessageProbeS,MessageProbe,EntropyS,Entropy

11.12 swarm.analysis.EntropyS

public interface *EntropyS*
implements MessageProbeS

Computes entropy via a MessageProbe.. Entropy objects read probabilities (via a MessageProbe) from a collection of objects and calculate the entropy of the underlying distribution.

11.13 swarm.analysis.EZAverageSequence

---

public interface *EZAverageSequence*
implements SwarmObjectS,SwarmObject

Protocol for an EZAverageSequence. A sequence generated using an EZGraph instance returns an object of this type

11.14 swarm.analysis.EZAverageSequenceC

---

public interface *EZAverageSequenceC*
implements SwarmObjectS,SwarmObjectC

Protocol for an EZAverageSequence. A sequence generated using an EZGraph instance returns an object of this type

11.14.1 Implementing class: swarm.analysis.EZAverageSequenceCImpl

java.lang.Object

swarm.PhaseCImpl

---

public *EZAverageSequenceCImpl*
extends PhaseCImpl
implements SwarmObjectS,SwarmObjectC,EZAverageSequenceS,EZAverageSequenceC

11.14.2 Implementing class: swarm.analysis.EZAverageSequenceImpl

java.lang.Object

---

public *EZAverageSequenceImpl*
extends Object
implements SwarmObjectS,SwarmObject,EZAverageSequenceS,EZAverageSequence

11.15 swarm.analysis.EZAverageSequenceS

public interface *EZAverageSequenceS*
implements SwarmObjectS

Protocol for an EZAverageSequence. A sequence generated using an EZGraph instance returns an object of this type

11.16 swarm.analysis.EZBin

public interface *EZBin*
implements SwarmObjectS,SwarmObject,GUICompositeS,GUIComposite

An easy to use histogram interface.. 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.

11.16.1 Methods

public 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.

public int ***getBinColorCount()***

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

public int ***getBinCount()***

The getBinCount method gets the number of bins in the histogram.

public int ***getCount()***

The getCount method gets the number of entries which landed within the bounds of the histogram.

public String ***getFileName()***

Return the filename string.

public double ***getLowerBound()***

The getLowerBound method gets the lower bound on the histogram range.

public double ***getMax()***

The getMax method gets the maximum value in the dataset.

public double ***getMin()***

The getMin method gets the minimum value in the dataset.

public int ***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.

public 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.

public String ***getTitle***()

Return the title string.

public double ***getUpperBound***()

The getUpperBound method gets the upper bound on the histogram range.

public Object ***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.

public Object ***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.

public Object ***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.

public Object ***reset***()

The reset method resets the histogram.

public Object ***setPrecision****(int precision)*

Sets the number of significant figures shown for major-tick labels.

public Object ***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.

11.17 swarm.analysis.EZBinC

public interface *EZBinC*
implements SwarmObjectS,SwarmObjectC,GUICompositeS,GUICompositeC

An easy to use histogram interface.. 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.

11.17.1 Methods

public Object ***createBegin****(swarm.defobj.Zone aZone)*

public Object ***createEnd****()*

public Object ***setAxisLabelsX$Y****(java.lang.String xl, java.lang.String 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.)

public Object ***setBinCount****(int theBinCount)*

The setBinCount method sets the number of bins the histogram will have.

public Object ***setCollection****(java.lang.Object aCollection)*

The setCollection method sets the collection of target objects which will be requested to generate the dataset for the histogram.

public Object ***setFileName****(java.lang.String 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.

public Object ***setFileOutput****(boolean 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.

public Object ***setGraphics****(boolean 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.

public Object ***setLowerBound****(double theMin)*

The setLowerBound method sets the inclusive lower bound on the histogram range.

public Object ***setMonoColorBars****(boolean mcb)*

The setMonoColorBars method specifies whether all bars should be shown in a single color (blue). The default is differently colored bars.

public Object ***setProbedSelector****(swarm.Selector 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.)

public Object ***setTitle****(java.lang.String 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.)

public Object ***setUpperBound****(double theMax)*

The setUpperBound method sets the non-inclusive upper bound on the histogram range.

11.17.2 Implementing class: swarm.analysis.EZBinCImpl

java.lang.Object

swarm.PhaseCImpl

---

public *EZBinCImpl*
extends PhaseCImpl
implements SwarmObjectS,SwarmObjectC,GUICompositeS,GUICompositeC,EZBinS,EZBinC

11.17.3 Implementing class: swarm.analysis.EZBinImpl

java.lang.Object

---

public *EZBinImpl*
extends Object
implements SwarmObjectS,SwarmObject,GUICompositeS,GUIComposite,EZBinS,EZBin

11.18 swarm.analysis.EZBinS

public interface *EZBinS*
implements SwarmObjectS,GUICompositeS

An easy to use histogram interface.. 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.

11.19 swarm.analysis.EZDistribution

public interface *EZDistribution*
implements EZBinS,EZBin

An EZBin that treats data as a distribution.. 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.

11.19.1 Methods

public double ***getEntropy()***

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

public Object ***output()***

The output method causes the graphical display to be updated with the information extracted by the previous call to update. When file I/O is enabled (the state of setFileOutput is set to 1), the probability associated with each bin is sent to the output file. When the graphical display is enabled (the state of setGraphics is set to 1), the histogram will be drawn.

public Object ***update()***

The update method polls the bins and updates the entropy of the distribution as well as the probabilities associated with the individual bins.

11.20 swarm.analysis.EZDistributionC

public interface *EZDistributionC*
implements EZBinS,EZBinC

An EZBin that treats data as a distribution.. 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.

11.20.1 Methods

public Object ***createEnd()***

11.20.2 Implementing class: swarm.analysis.EZDistributionCImpl

java.lang.Object

swarm.PhaseCImpl

public *EZDistributionCImpl*
extends PhaseCImpl
implements EZBinS,EZBinC,EZDistributionS,EZDistributionC

11.20.3 Implementing class: swarm.analysis.EZDistributionImpl

java.lang.Object

public *EZDistributionImpl*
extends Object
implements EZBinS,EZBin,EZDistributionS,EZDistribution

11.21 swarm.analysis.EZDistributionS

---

public interface *EZDistributionS*
implements EZBinS

An EZBin that treats data as a distribution.. 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.

11.22 swarm.analysis.EZGraph

public interface *EZGraph*
implements SwarmObjectS,SwarmObject,GUICompositeS,GUIComposite

A class for easily create graphs.. 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.

11.22.1 Methods

public EZAverageSequence
createAverageSequence$withFeedFrom$andSelector*(java.lang.String aName, java.lang.Object aCollection, swarm.Selector 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.

public EZAverageSequence
createCountSequence$withFeedFrom$andSelector*(java.lang.String aName, java.lang.Object aCollection, swarm.Selector 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.

public EZAverageSequence
createMaxSequence$withFeedFrom$andSelector*(java.lang.String aName, java.lang.Object aCollection, swarm.Selector 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.

public EZAverageSequence
createMinSequence$withFeedFrom$andSelector*(java.lang.String aName, java.lang.Object aCollection, swarm.Selector 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.

public EZSequence ***createSequence$withFeedFrom$andSelector****(java.lang.String aName, java.lang.Object anObj, swarm.Selector 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.

public EZAverageSequence ***createTotalSequence$withFeedFrom$andSelector****(java.lang.String aName, java.lang.Object aCollection, swarm.Selector 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.

public Object ***dropSequence****(java.lang.Object 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.

public String ***getFileName()***

Return the file name prefix string.

public String ***getTitle()***

Return the title string.

public Object ***outputGraph()***

the outputGraph method updates the graph with the data obtained from the last call to -update. If setGraphics==0, nothing is done.

public Object ***outputToFile()***

the outputToFile method sends to the disk file data obtained from the last call to -update. If setFileOutput==0, nothing is done.

public Object ***setRangesXMin$Max****(double xmin, double xmax)*

Fix the range of X values on the graph between some range.

public Object ***setRangesYMin$Max****(double ymin, double ymax)*

Fix the range of Y values on the graph between some range.

public Object ***setScaleModeX$Y****(boolean xs, boolean ys)*

Whether to autoscale every timestep or instead to jump scale.

public Object ***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.

public Object ***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.

11.23 swarm.analysis.EZGraphC

public interface *EZGraphC*
implements SwarmObjectS,SwarmObjectC,GUICompositeS,GUICompositeC

A class for easily create graphs.. 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.

11.23.1 Methods

public Object ***create$setFileName****(swarm.defobj.Zone aZone, java.lang.String aFileName)*

Convenience method for creating a non-graphical EZGraph, in this case, the filename is explicitly set by the user

public Object ***create$setFileOutput****(swarm.defobj.Zone aZone, boolean fileOutputFlag)*

Convenience method for creating a non-graphical EZGraph, the filename is generated from the sequence name

public Object ***create$setTitle$setAxisLabelsXYsetWindowGeometryRecordName****(swarm.defobj.Zone aZone, java.lang.String aTitle, java.lang.String xl, java.lang.String yl, java.lang.String windowGeometryRecordName)*

Convenience method for creating 'graphical' EZGraph instances

public Object ***createEnd****()*

public Object ***setAxisLabelsX$Y****(java.lang.String xl, java.lang.String 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.)

public Object ***setFileName****(java.lang.String 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.

public Object ***setFileOutput****(boolean 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.

public Object ***setGraphics****(boolean 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.

public Object ***setTitle****(java.lang.String 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.)

11.23.2 Implementing class: swarm.analysis.EZGraphCImpl

java.lang.Object

swarm.PhaseCImpl

---

public *EZGraphCImpl*
extends PhaseCImpl
implements SwarmObjectS,SwarmObjectC,GUICompositeS,GUICompositeC,EZGraphS,EZGraphC

11.23.3 Implementing class: swarm.analysis.EZGraphImpl

java.lang.Object

---

public *EZGraphImpl*
extends Object
implements SwarmObjectS,SwarmObject,GUICompositeS,GUIComposite,EZGraphS,EZGraph

11.24 swarm.analysis.EZGraphS

---

public interface *EZGraphS*
implements SwarmObjectS,GUICompositeS

A class for easily create graphs.. 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.

11.25 swarm.analysis.EZSequence

---

public interface *EZSequence*
implements SwarmObjectS,SwarmObject

Protocol for an EZSequence. A sequence generated using an EZGraph instance returns an object of this type

11.26 swarm.analysis.EZSequenceC

public interface *EZSequenceC*
implements SwarmObjectS,SwarmObjectC

Protocol for an EZSequence. A sequence generated using an EZGraph instance returns an object of this type

11.26.1 Implementing class: swarm.analysis.EZSequenceCImpl

java.lang.Object

swarm.PhaseCImpl

public *EZSequenceCImpl*
extends PhaseCImpl
implements SwarmObjectS,SwarmObjectC,EZSequenceS,EZSequenceC

11.26.2 Implementing class: swarm.analysis.EZSequenceImpl

java.lang.Object

public *EZSequenceImpl*
extends Object
implements SwarmObjectS,SwarmObject,EZSequenceS,EZSequence

11.27 swarm.analysis.EZSequenceS

---

public interface *EZSequenceS*
implements SwarmObjectS

Protocol for an EZSequence. A sequence generated using an EZGraph instance returns an object of this type

11.28 swarm.analysis.FunctionGraph

---

public interface *FunctionGraph*
implements SwarmObjectS,SwarmObject

A widget for drawing a function over a range of one variable.. 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.

11.28.1 Methods

*public Object **graph**()*

Draw the graph with the current contents of the graph element.

11.29 swarm.analysis.FunctionGraphC

public interface *FunctionGraphC*
implements SwarmObjectS,SwarmObjectC

A widget for drawing a function over a range of one variable.. 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.

11.29.1 Methods

public Object ***createBegin****(swarm.defobj.Zone aZone)*

public Object ***createEnd****()*

public Object ***setArithmeticWarn****(boolean state)*

If true, raise a warning if the function method failed to compute a value.

public Object ***setDataFeed****(java.lang.Object feed)*

Set the target to send the function method.

public Object ***setElement****(java.lang.Object graphElement)*

Set the GraphElement to use for plotting.

public Object ***setFunctionSelector****(swarm.Selector aSel)*

Set the function method.

public Object ***setResetFrequency****(int freq)*

Set the frequency at which to clear the graph element.

public Object ***setXMinMaxResolution****(double minx, double maxx, int steps)*

Set the range and resolution of X values at which to compute values.

public Object ***setXMinMaxStepSize****(double minx, double maxx, double size)*

Set the range and step size of X values at which to compute values.

11.29.2 Implementing class: swarm.analysis.FunctionGraphCImpl

java.lang.Object

swarm.PhaseCImpl

---

public *FunctionGraphCImpl*
extends PhaseCImpl
implements SwarmObjectS,SwarmObjectC,FunctionGraphS,FunctionGraphC

11.29.3 Implementing class: swarm.analysis.FunctionGraphImpl

java.lang.Object

---

public *FunctionGraphImpl*
extends Object
implements SwarmObjectS,SwarmObject,FunctionGraphS,FunctionGraph

11.30 swarm.analysis.FunctionGraphS

public interface *FunctionGraphS*
implements SwarmObjectS

A widget for drawing a function over a range of one variable.. 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.