==========

SktSocketManager

INHERITS FROM	Object
DECLARED IN	SktSocketManager.h

CLASS DESCRIPTION

The SktSocketManager class, together with the SktSocket and SktSocketUser classes, provides an application with the ability to act as a server for Berkeley UNIX stream socket connections. An SktSocket object handles input/output on the socket itself, while an instance of a subclass of SktSocketUser processes the input and provides output to the SktSocket. The SktSocketManager class coordinates the efforts of each SktSocket.

SktSocketManager requires an **update** message to act. The **update** message is best sent as a regular interval message, from within a timed entry, or within a loop.

During an **update** cycle**,** an SktSocketManager has all SktSocket objects perform input and output, and handles requests for connections

by creating a new SktSocket/SktSocketUser pair.

The **update** method makes use of the *select()* UNIX system call to determine if there's any data waiting to be read or written. Several methods have been provided to allow precise control over the behavior of this function, but you don't need to know about *select()* to get started using the socket classes.

These classes are intended to make sockets easy to use. However, several methods have been provided to access fairly low-level details of sockets which may be useful in some situations. For more information on sockets, see: the related UNIX man pages; ªAn Introductory 4.3BSD Interprocess Communication Tutorialº (reprinted in UNIX Programmer's Supplementary Documents Volume 1, PS1:7); or, ªAn Advanced 4.3BSD Interprocess Communication Tutorialº (reprinted in UNIX Programmer's Supplementary Documents Volume 1, PS1:8).

See also: SktSocket, SktSocketUser

INSTANCE VARIABLES

Inherited from Object	Class	isa;
Declared in SktSocketManager		
	FILE	*logfile;
	id	userClass;
	char	*hostaddress;
	char	*hostname;
	int	servicePort;
	int	numAvailFds;
	int	maxSocketFd;
	int	serviceSocketFd;
	List	*openSockets;
	BOOL	doesLog;
	NXZone	*zone;
	struct timeval	selectTimeout;
	BOOL	timeoutIndefinite;

	int selectSignalMask;
logfile	The file to which all diagnostic output is printed.
userClass	Factory for new SocketUsers.
hostaddr	The primary Internet address of the machine the SktSocketManager's application is running on, in dot notation.
hostname	The name of the machine the SktSocketManager's application is running on.
servicePort	The Internet port used to connect to the SktSocketManager's application (for example, via TELNET).
numAvailFds	The number of file descriptors available for SktSocket assignment.
maxSocketFd	The highest valued file descriptor currently assigned.
serviceSocketFd	The file descriptor of SktSocketManager's socket.
openSockets	A List of currently active SktSocket objects.
doesLog	YES if non-error messages are written to the log file.
zone	The zone that the SktSocketManager was allocated from.
selectTimeout	The longest time that the SktSocketManager will wait for input or output during an **update**.
timeoutIndefinite	YES if **selectTimeout** is ignored, and **update** will block indefinitely until there is input or

output, or until a signal occurs.

selectSignalMask — A mask of signals that are blocked during the *select()* system call in **update**.

METHOD TYPES

Initializing and freeing a SktSocketManager

-€initPort:logfile:fdCapacity: userClass:
-€setSocketOptions:
-€free
-€setDoesLog:
-€doesLog

Advanced initialization (you should know about *select()*)

-€setTimeoutSeconds: microseconds:
-€getTimeoutSeconds: microseconds:
-€setTimeoutIndefinite
-€isTimeoutIndefinite
-€setSignalMask:
-€signalMask

Accessing SktSocketManager attributes

-€setFdCapacity:
-€adjustFdCapacity:
-€fdCapacity
-€numAvailFds
-€logfile
-€hostaddress
-€hostname
-€servicePort
-€getInetAddresses

Synchronization and management -€update

	-€closeSocket: -€closeAllSockets
Utility methods	-€announceString: -€log:

INSTANCE METHODS

adjustFdCapacity:

-€(int)**adjustFdCapacity:**(int)*byAmount*

Adjusts the number of available file descriptors by *byAmount*. If this would result in the total number of descriptors reserved being greater than the process's dtablesize (see *getdtablesize(2)* in the UNIX manual pages), no change is made. If it would result in less than zero descriptors being reserved, the number of available descriptors is set to zero. Returns the new capacity of the SktSocketManager.

This method is useful for reserving file descriptors in your process for things other than socket connections.

See also: -€**initPort:logfile:capacity:userClass:**, -€**fdCapacity:**, -€**numAvailFds**, -€**setFdCapacity:**

announceString:

-€**announceString:**(const char *)*announcement*

Has every active SktSocket queue *announcement* as output. Returns **self**.

See also: -€**queueOutput:** (SktSocket)

fdCapacity

-€(int)**capacity**

Returns the number of file descriptors, both available and in use, reserved by the SktSocketManager.

See also: -€**adjustFdCapacity:**, -€**initPort:logfile:capacity:userClass:**, -€**numAvailFds**, -€**setFdCapacity:**

closeAllSockets

-€**closeAllSockets**

Sends **self** a **closeSocket:** message for each open SktSocket. Returns **self**.

See also: -€**closeSocket:**

closeSocket:

-€**closeSocket:***socketObj*

Removes *socketObj* from the list of open SktSockets and sends it a **free** message. Returns **self**.

See also: -€**closeAllSockets**

doesLog:

-€(BOOL)**doesLog**

Returns YES if non-error messages (notices of new connections, closing of connections, etc.) as well as error messages will be written to the log file. The default is to write such messages.

See also: -€**setDoesLog:**

getInetAddresses

-€(char **)**getInetAddresses**

Returns a null-terminated array of character strings containing the

Internet addresses (in dot notation) of the machine the application is running on. This array is created on demand, and it and its contents may be freed by the sender of the message when it is no longer needed (you must free only the top-level pointer). If the addresses can't be retrieved, this method returns NULL.

See also: -€**hostaddress**, -€**hostname**

getTimeoutSeconds:microseconds:

-€**getTimeoutSeconds:**(long int *)*secs* **microseconds:**(long int *)*usecs*

Returns by reference the components of the timeout value used by *select()* in **update**. **update** will not wait longer than this for input or output to process. The default values for these components are 0 and 0; that is, **update** will simply poll for input or output. Returns **self**.

See also: -€**setTimeoutSeconds:microseconds:**, -€**setTimeoutIndefinite**, -€**isTimeoutIndefinite**, -€**update**, select(2)

hostaddress

-€(const char *)**hostaddress**

Returns the Internet address of the machine the SktSocketManager's application is running on, in dot notation.

See also: -€**hostname**, -€**getInetAddresses**

hostname

-€(const char *)**hostname**

Returns the hostname of the machine the SktSocketManager's application is running on.

See also: -€**hostaddr**, -€**inetAddresses**

initPort:logfile:fdCapacity:userClass:

-€initPort:(int)*portNum* **logfile:**(FILE *)*file* **fdCapacity:**(int)*cap* **userClass:***aClass*

Attempts to create a socket bound to *portNum*. This socket is used to handle requests for new connections. In addition, *file* is recorded as the file for diagnostic output, c*ap* file descriptors are assumed available by the SktSocketManager for creation of new SktSocket objects and *aClass* is recorded as the class object which will allocate SktSocketUser objects for newly created Sockets. A new SktSocketUser is created by sending **alloc** and **init** messages to the SktSocketUser subclass represented by *aClass.*

This method also invokes **setSocketOptions:**, so that you may set file descriptor flags in subclasses. If **setSocketOptions:** returns **nil**, the initialization is cancelled and returns **nil**. An SktSocketManager doesn't set any options for itself.

Returns **self** if successful, or **nil** on any of these conditions: *portNum* is in use or reserved (ports equal to or below 1024 are reserved for use by the super user), *cap* is negative or greater than the process's dtablesize (see *getdtablesize*(2) in the UNIX manual pages), *aClass* isn't SktSocketUser or a subclass thereof, **setSocketOptions:** returns **nil**, the socket couldn't be bound, or the List of open SktSockets couldn't be created. An error message is logged detailing the particular error condition.

This method is the designated initializer for SktSocketManager objects.

See also: **-€setSocketOptions:**, **-€adjustFdCapacity:**, **-€fdCapacity:**, **-€numAvailFds**, **-€servicePort**, **-€setFdCapacity:**, fcntl(2), setsockopt(2)

isTimeoutIndefinite

-€(BOOL)**isTimeoutIndefinite**

Returns YES if **update** waits indefinitely for input or output to process. The default behavior is to poll, and not wait at all. Returns **self**.

See also: -€**setTimeoutIndefinite**, -€**setTimeoutSeconds:microseconds:**, -€**getTimeoutSeconds:microseconds:**, -€**update**, select(2)

log:

-€**log:**(const char *)*format, ...*

Prints message to the log file, in the same manner as fprintf(). Returns **self**.

numAvailFds

-€(int)**numAvailFds**

Returns the number of file descriptors the SktSocketManager currently has reserved and unassigned to sockets.

See also: -€**adjustCapacity:**, -€**capacity:**, -€**initPort:controller:capacity:userClass:**, -€**setCapacity:**

servicePort

-€(int)**servicePort**

Returns the Internet port used to connect to the SktSocketManager (for example, via TELNET).

See also: -€**initPort:controller:capacity:userClass:**

setDoesLog:

-€**setDoesLog:**(BOOL)*flag*

If *flag* is YES, then non-error messages (notices of new connections, closing of connections, etc.) as well as error messages will be written to the log file. The default is to write such messages. Returns **self**.

See also: -€**doesLog**

setFdCapacity:

-€setFdCapacity:(int)*cap*

Changes the number of file descriptors reserved by the SktSocketManager (both currently allocated and free) to *cap*. If *cap* is greater than the process's dtablesize (see getdtablesize(2) in the UNIX manual pages), no change is made. If *cap* is less than zero, then zero descriptors are reserved. Returns **self**.

This method is useful for reserving file descriptors in your process for things other than socket connections.

See also: **-€fdCapacity:**, **-€adjustFdCapacity:**, **-€initPort:logfile:fdCapacity:userClass:**, **-€numAvailFds**

setSignalMask:

-€setSignalMask:(int)*mask*

Records *mask* as the signal mask to use before a call to *select()*, so that *select()* isn't affected by the signals specified in *mask*. Returns **self**.

To create a signal mask, use the marco *signalMask()*, defined in <signal.h>. For example, to get the signal mask for SIGHUP and SIGINT, write

```
int mask = signalMask(SIGHUP) || signalMask(SIGINT);
```

You can build up a mask covering many signals by logically combining these mask values.

See also: **-€signalMask**, **-€update**, sigsetmask(2)

setSocketOptions:

-€setSocketOptions:(int)*fd*

Does nothing and returns **self**. You can override this method in subclasses to set any options that you like on the service socket's file

descriptor. Be sure to send **setSocketOptions:** to **super** in your own method. If an operation fails so that you don't want the SktSocketManager to be initialized, this method should return **nil**.

See also: **-€initPort:logfile:fdCapacity:userClass:**, fcntl(2), setsockopt(2)

setTimeoutIndefinite

-€setTimeoutIndefinite

Causes **update** to wait indefinitely for input or output to process. The default behavior is to poll, and not wait at all. Returns **self**.

See also: **-€isTimeoutIndefinite**, **-€setTimeoutSeconds:microseconds:**, **-€getTimeoutSeconds:microseconds:**, **-€update**, select(2)

setTimeoutSeconds:microseconds:

-€setTimeoutSeconds:(long int)*secs* **microseconds:**(long int)*usecs*

Records *secs* and *usecs* as the components of the timeout value used by *select()* in **update**. **update** will not wait longer than this for input or output to process. The default values for these components are 0 and 0; that is, **update** will simply poll for input or output, not waiting at all. Returns **self**.

See also: **-€getTimeoutSeconds:microseconds:**, **-€setTimeoutIndefinite**, **-€isTimeoutIndefinite**, **-€update**, select(2)

signalMask

-€(int)signalMask

Returns the signal mask used before a call to *select()*, so that *select()* is not affected by the signals specified in *mask*.

See also: **-€setSignalMask**, **-€update**, sigsetmask(2)

update

-€update

Performs SktSocketManager's control cycle. First, all Sockets with exceptions pending are closed via **closeSocket:.** All remaining Sockets are then sent a **flushOutput** message. Any Sockets with input pending are sent a **getInput** message. Last, if there is a new connection pending, then for each new connection, a new SktSocket object is created, an object of class **userClass** is allocated and initialized with the new SktSocket, and the SktSocket is added to the list of open SktSockets**.** Returns **self**.

update makes use of the timeout specified by the last invocation of **setTimeoutSeconds:microseconds:** or **setTimeoutIndefinite**, as well as any mask set by **setSignalMask:**. Before *select()* is called, the signal mask is set, and after it returns, the previous mask is restored. *select()* uses the timeout specified by **selectTimeout**, or blocks indefinitely is **timeoutIndefinite** is YES.

See also: **-€closeSocket**, **-€flushOutput** (SktSocket), **-€getInput** (SktSocket), **-€setTimeoutSeconds:microseconds:**, **-€getTimeoutSeconds:microseconds:**, **-€setTimeoutIndefinite**, **-€isTimeoutIndefinite**, **-€setSignalMask:**, **-€signalMask**, select(2)