DOEventLoop

Inherits From: Object

Conforms To: DOMachMessageHandling

Declared In: remote/DOEventLoop.h

Class Description

The DOEventLoop class is provided with Portable Distributed Objects systems so that PDO applications can have an object that handles the concept of a “run loop” in much the way that NEXTSTEP’s Application object does. You can register objects with a DOEventLoop such that they’ll be notified when data is available to be read from a file descriptor, when a Mach message is waiting on a port, or when a specified amount of time has passed. Such objects are called event “handlers.”

Normally, you’ll set up a few handlers on an event loop and then run it (with the **run** method). The handler objects may in turn add more handlers for new file descriptors, ports, and timeout events.

There are three new protocols and two new data types used with the DOEventLoop class. Since they’re all quite small, they’re described at the end of this class specification.

Instance Variables

None declared in this class.

Adopted Protocols

DOEventHandling – machMessageReceived:handlerData:

Method Types

Managing event handlers	– addConnection: – addFileDescriptor:handler:handlerData: – removeFileDescriptor: – addPort:handler:handlerData: – removePort: – addTimeoutEntry:handler:handlerData: – removeTimeoutEntry:
Running a loop	– run – stop

Instance Methods

addConnection:

– **addConnection:**(NXConnection *)*aConnection*

This methods adds *aConnection* to the event loop, allowing it to handle remote messages.

See also: – **run**

addFileDescriptor:handler:handlerData:

– **addFileDescriptor:**(int)*fd* **handler:**(id <DOFileDescriptorHandling>)*anObject*
handlerData:(void *)*data*

This method registers *anObject* as the handler for incoming data on the UNIX file descriptor *fd*. The handler data, *data*, may be anything needed by *anObject*.

When data is ready to be read from the file descriptor, *anObject* will receive a **dataOnFileDescriptor:handlerData:** message. Handler objects aren't notified until the DOEventLoop has received a **run** message.

Returns **self** if *fd* is successfully added, **nil** otherwise.

See also: – **removeFileDescriptor:**,
– **dataOnFileDescriptor:handlerData:** (DOFileDescriptorHandling protocol)

addPort:handler:handlerData:

– **addPort:**(port_t)*port* **handler:**(id <DOMachMessageHandling>)*anObject*
handlerData:(void *)*data*

This method registers *anObject* as the handler for incoming Mach messages on the Mach port *port*. The handler data, *data*, may be anything needed by *anObject*.

When a Mach message is waiting on *port*, *anObject* will receive a **machMessageReceived:handlerData:** message. Handler objects aren't notified until the DOEventLoop has received a **run** message.

Returns **self** if successful, or **nil** if an error occurs.

See also: – **removePort:**,
– **machMessageReceived:handlerData:** (DOMachMessageHandling protocol)

addTimeoutEntry:handler:handlerData:

– (DOTimeoutReceipt)**addTimeoutEntry:**(DOTimeInterval)*timeout*
handler:(id <DOTimeoutHandling>)*anObject* **handlerData:**(void *)*data*

This method registers *anObject* as the handler for a timeout event *timeout* milliseconds from the time this method is invoked. The handler data, *data*, may be anything needed by *anObject*.

When the timeout occurs, *anObject* will receive a **timeoutOccurred:handlerData:** message. Timing doesn't begin and handler objects aren't notified until the DOEventLoop has received a **run** message.

Returns a receipt that can later be used to identify the timeout event created by this method; for example, in a **removeTimeoutEntry:** message.

See also: – **removeTimeoutEntry:**,
– **timeoutOccurred:handlerData:** (DOTimeoutHandling protocol)

removeFileDescriptor:

– **removeFileDescriptor:**(int)*fd*

Deregisters the handler object for the file descriptor *fd*. Returns **self** if *fd* was registered, **nil** if it wasn't.

See also: – **addFileDescriptor:handler:handlerData:**

removePort:

– **removePort:**(port_t)*port*

Deregisters the handler object for the Mach port *port*. Returns **self** if *port* was registered, **nil** if it wasn't.

See also: – **addPort:handler:handlerData:**

removeTimeoutEntry:

– **removeTimeoutEntry:**(DOTimeoutReceipt)*receipt*

Deregisters the handler object for the timeout event identified by *receipt*. Returns **self** if *receipt* identified a registered timeout, **nil** if it wasn't.

See also: – **addTimeoutEntry:handler:handlerData:**

run

– **run**

Runs the event loop. The receiver waits for data or timeouts, and notifies the appropriate handlers. This method returns **self** if the event loop receives a **stop** message, and otherwise returns **nil** if a serious error occurs, such as the program deallocating a port and not removing the handler for it.

If you send **run** to an event loop with no connections or handlers, your application will be caught in an infinite loop. You should never send **run** to an event loop that's already running.

See also: – **stop**

stop

– (void)**stop**

Stops a running event loop. The loop's **run** method will then return **self**. You should never send **stop** to an event loop that isn't running.

See also: – **run**

DOFileDescriptorHandling Protocol

dataOnFileDescriptor:handlerData:

– (void)**dataOnFileDescriptor:**(int)*fd* **handlerData:**(void *)*data*

An event handler object receives this message when there's data available to read on the file descriptor *fd*. The handler should do whatever it needs to process the event. *data* is the data originally passed in by the invocation of **addFileDescriptor:handler:handlerData:** that registered the handler object.

See also: – **addFileDescriptor:handler:handlerData:** (DOEventLoop class)

DOMachMessageHandling Protocol

machMessageReceived:handlerData:

– (void)**machMessageReceived:**(msg_header_t *)*msg* **handlerData:**(void *)*data*

An event handler object receives this message when there's a Mach message waiting to be read. The handler should do whatever it needs to process the event. *data* is the data originally passed in by the invocation of **addPort:handler:handlerData:** that registered the handler object.

See also: – **addPort:handler:handlerData:** (DOEventLoop class)

DOTimeoutHandling Protocol

timeoutOccurred:handlerData:

– (void)**timeoutOccurred:**(DOTimoutReceipt)*receipt* **handlerData:**(void *)*data*

An event handler object receives this message when a timeout previously requested occurs. *receipt* identifies the particular timeout event. The handler should do whatever it needs to process the event. *data* is the data originally passed in by the invocation of **addTimeoutEntry:handler:handlerData:** that registered the handler object.

See also: – **addTimeoutEntry:handler:handlerData:** (DOEventLoop class)

Data Types

typedef unsigned long **DOTimeInterval**;
typedef unsigned long **DOTimeoutReceipt**;

These two data types are used to specify the time in milliseconds before a timeout occurs, and to identify a particular timeout event. Their use is explained in the timeout-handling methods above.