NSInvocation

Inherits From:	NSObject
Conforms To:	NSCoding NSObject (NSObject)
Declared In:	Foundation/NSInvocation.h

Class Description

An NSInvocation is an Objective-C message rendered static, an action turned into an object. NSInvocation objects are used to store and forward messages between objects and between applications, primarily by NSTimers and the distributed objects system. An NSInvocation contains all the elements of an Objective-C message: a target, a selector, arguments, and the return value. Each of these elements can be set directly, and the return value is set automatically when the NSInvocation is dispatched.

An NSInvocation can be repeatedly dispatched to different targets; its arguments can be modified between dispatch for varying results; even its selector can be changed to another with the same method signature (argument and return types). This makes it useful for repeating messages with many arguments and variations; rather than retyping a slightly different expression for each message, you modify the NSInvocation as needed each time before dispatching it to a new target.

Creating an NSInvocation requires several steps. Consider this method of the custom class MyCalendar:

– (BOOL)**updateAppointmentsForDate:**(NSDate *)*aDate*

updateAppointmentsForDate: takes an NSDate as its only argument, and returns YES or NO depending on whether the appointments could be updated without conflicts. The following code fragment sets up an NSInvocation for it:

```
SEL theSelector;
NSMethodSignature *aSignature;
NSInvocation *anInvocation;

theSelector = @selector(updateAppointmentsForDate:);
aSignature = [MyCalendar methodSignatureForSelector:theSelector]
anInvocation = [NSInvocation invocationWithMethodSignature:aSignature];
[anInvocation setSelector:theSelector];
```

The first two lines get the NSMethodSignature for the **updateAppointmentsForDate:** method. The last two lines actually create the NSInvocation and set its selector. Note that the selector can be set to any selector matching the signature of **updateAppointmentsForDate:**. Any of these methods can be used with **anInvocation**:

– (BOOL)**clearAppointmentsForDate:**(NSDate *)*aDate*
– (BOOL)**isAvailableOnDate:**(NSDate *)*aDate*
– (BOOL)**setMeetingTime:**(NSDate *)*aDate*

Before being dispatched, **anInvocation** must have its target and arguments set:

```
MyCalendar *userDatebook;    /* Assume this exists. */
NSDate *todaysDate;          /* Assume this exists. */

[anInvocation setTarget:userDatebook];
[anInvocation setArgument:&todaysDate atIndex:3];
```

setArgument:atIndex: sets the specified argument to the value supplied. Every method has two hidden arguments, the target and selector, so the first argument that needs to be set is actually at index 3. In this case, **todaysDate** will be the NSDate argument to **updateAppointmentsForDate:**.

To dispatch the NSInvocation, send an **invoke** or **invokeWithTarget:** message. **invoke** only produces a result if the NSInvocation has a target set. Once dispatched, the NSInvocation contains the return value of the message, which **getReturnValue:** produces:

```
BOOL result;

[anInvocation invoke];
[anInvocation getReturnValue:&result];
```

Saving NSInvocations for Later Use

Because an NSInvocation doesn't always need to retain its arguments, by default it doesn't do so. This can cause object arguments as well as the target to become invalid if they're automatically released. If you plan to cache an NSInvocation or dispatch it repeatedly during the execution of your application, you should send it a **retainArguments** message. This method retains the target and all object arguments, and copies C strings so that they're not lost because another object frees them.

Using NSInvocations with NSTimers

Suppose the NSInvocation created above is being used in a time-management application that allows multiple users to set appointments for others, such as group meetings. This application might allow a user's calendar to be automatically updated every few minutes, so that the user always knows what his schedule looks like. Such automatic updating can be accomplished by setting up NSTimer objects with NSInvocations.

Given the NSInvocation above, this is as simple as invoking one NSTimer method:

```
[NSTimer scheduledTimerWithInterval:600
    invocation:anInvocation
    repeats:YES];
```

This line of code sets up an NSTimer to dispatch **anInvocation** every 10 minutes (600 seconds). Note that an NSTimer always instructs its NSInvocation to retain its arguments; thus, you don't need to send **retainArguments** yourself. See the NSTimer class specification for more information on timers.

Adopted Protocols

NSCoding	– encodeWithCoder: – initWithCoder:

Method Types

Creating instances	+ invocationWithMethodSignature:
Accessing message elements	– setArgument:atIndex: – getArgument:atIndex: – setReturnValue: – getReturnValue: – setSelector: – selector – setTarget: – target
Managing arguments	– argumentsRetained – retainArguments
Dispatching an invocation	– invoke – invokeWithTarget:
Getting the method signature	– methodSignature

Class Methods

invocationWithMethodSignature:

+ (NSInvocation *)**invocationWithMethodSignature:**(NSMethodSignature *)*signature*

Returns an NSInvocation object able to construct messages using selectors described by *signature*. The new object must have its selector set with **setSelector:** and its arguments set with **setArgument:atIndex:** before it can be invoked.

nstance Methods

argumentsRetained

– (BOOL)**argumentsRetained**

Returns YES if the NSInvocation has retained its arguments, NO otherwise.

See also: – **retainArguments**

getArgument:atIndex:

– (void)**getArgument:**(void *)*buffer* **atIndex:**(int)*index*

Copies the argument stored at *index* into the storage pointed to by *buffer*. Indices 0 and 1 indicate the hidden arguments **self** and **_cmd**, respectively; these values can be retrieved directly with the **target** and **selector** methods. Use indices 2 and greater for the arguments normally passed in a message.

buffer must be large enough to accommodate the argument value. This method raises NSInvalidArgumentException if *index* is greater than the actual number of arguments for the selector.

See also: – **setArgument:atIndex:** – **numberOfArguments** (NSMethodSignature)

getReturnValue:

– (void)**getReturnValue:**(void *)*buffer*

Copies the invocation's return value into the storage pointed to by *buffer*, which should be large enough to accommodate the value. Use NSMethodSignature's **methodReturnLength** method to determine the size needed for **buffer**:

```
unsigned int length = [[myInvocation methodSignature]
    methodReturnLength];
buffer = (void *)malloc(length);
```

If the NSInvocation has never been invoked the result of this method is undefined.

See also: – **setReturnValue:**, – **methodReturnType** (NSMethodSignature)

invoke

– (void)**invoke**

Uses **invokeWithTarget:** to send the NSInvocation's message with arguments to its target. The NSInvocation's target, selector, and argument values must be set before this method is invoked.

See also: – **getReturnValue:**, – **setSelector:**, – **setTarget:**, – **setArgument:atIndex:**

invokeWithTarget:

– (void)**invokeWithTarget:**(id)*anObject*

Sends the NSInvocation's message with arguments to *anObject* and sets the return value. Doesn't set the NSInvocation's target. The NSInvocation's selector and argument values must be set before this method is invoked.

See also: – **getReturnValue:**, – **invoke**, – **setSelector:**, – **setTarget:**, – **setArgument:atIndex:**

methodSignature

– (NSMethodSignature *)**methodSignature**

Returns the invocation's method signature.

retainArguments

– (void)**retainArguments**

If the NSInvocation hasn't already done so, retains the NSInvocation's target and all object arguments, and copies all C string arguments. Before this method is invoked, **argumentsRetained** returns NO; after, it returns YES.

For efficiency, newly created NSInvocations don't retain or copy their arguments, nor do they retain their targets or copy C strings. You should instruct an NSInvocation to retain its arguments if you intend to cache it, since the arguments may otherwise be released before the NSInvocation is invoked. NSTimers always instruct their NSInvocations to retain their arguments, for example, since there's usually a delay before an NSTimer fires.

selector

– (SEL)**selector**

Returns the NSInvocation's selector, or 0 if it hasn't been set.

See also: – **setSelector:**

setArgument:atIndex:

– (void)**setArgument:**(void *)*buffer* **atIndex:**(int)*index*

Copies the contents of *buffer* as the argument at *index*. Indices 0 and 1 indicate the hidden arguments **self** and **_cmd**, respectively; these values should be set directly with the **setTarget:** and **setSelector:** methods.

Use indices 2 and greater for the arguments normally passed in a message. The number of bytes copied is determined by the argument size.

This method raises NSInvalidArgumentException if the value of *index* is greater than the actual number of arguments for the selector.

See also: – **getArgument:atIndex:**, – **numberOfArguments** (NSMethodSignature)

setReturnValue:

– (void)**setReturnValue:**(void *)*buffer*

Copies the contents of *buffer* as the NSInvocation's return value. This is normally set when you send an **invoke** or **invokeWithTarget:** message.

See also: – **getReturnValue:**, – **methodReturnLength** (NSMethodSignature),
– **methodReturnType** (NSMethodSignature)

setSelector:

– (void)**setSelector:**(SEL)*selector*

Sets the NSInvocation's selector to *selector.*

See also: – **selector**

setTarget:

– (void)**setTarget:**(id)*anObject*

Sets the NSInvocation's target to *anObject*. The target is the receiver of the message sent by **invoke**.

See also: – **target**, – **invokeWithTarget:**

target

– (id)**target**

Returns the NSInvocation's target, or **nil** if the NSInvocation has no target.

See also: – **setTarget:**