Undo Architecture

Cocoa > Data Management

2002-11-12

Contents

Introduction to Undo Architecture

This topic describes how to record operations with NSUndoManager, so the user can reverse an operation's effect.

Organization of This Document

- "Undo Manager" (page 7) describes the object which records operations for undo and redo.
- "Registering Undo Operations" (page 9) describes how to add operations to the undo stack.
- "Performing Undo and Redo" (page 11) describes how to do an undo and a redo.
- "Cleaning the Undo Stack" (page 13) describes how to remove operations from the undo stack.
- "Setting Action Names" (page 15) describes how to provide custom names for the Undo and Redo menu items.
- "Using Undo in Applications" (page 17) describes the Application Kit's additions to undo.
- "Using Undo Notifications" (page 19) describes synchronizing using notifications.

Undo Manager

NSUndoManager is a general-purpose recorder of operations for undo and redo. You register an undo operation by specifying the object that is changing (or the owner of that object), along with a method to invoke to revert its state, and the arguments for that method. NSUndoManager groups all operations within a single cycle of the run loop, so that performing an undo reverts all changes that occurred during the loop. Also, when performing undo an NSUndoManager saves the operations reverted so that you can redo the undos.

NSUndoManager is implemented as a class of the Foundation framework because executables other than applications might want to revert changes to their states. For example, you might have an interactive command-line tool with undo and redo commands; or there could be Distributed Object implementations that can revert operations "over the wire." However, users typically see undo and redo as application features. The Application Kit implements undo and redo in its NSTextView object and makes it easy to implement it in objects along the responder chain. For more on the role of the Application Kit in undo and redo, see "Using Undo in Applications" (page 17).

Undo Operations and Groups

An undo operation is a method for reverting a change to an object, along with the arguments needed to revert the change (for example, its state before the change). Undo operations are typically collected in undo groups, which represent whole revertible actions, and are stored on a stack. Redo operations and groups are simply undo operations stored on a separate stack (described below). When an NSUndoManager performs undo or redo, it is actually undoing or redoing an entire group of operations. For example, a user could change the type face and the font size of some text. An application might package both attribute-setting operations as a group, so when the user chooses Undo, both type face and font size are reverted. To undo a single operation, it must still be packaged in a group.

NSUndoManager normally creates undo groups automatically during the run loop. The first time it is asked to record an undo operation in the run loop, it creates a new group. Then, at the end of the loop, it closes the group. You can create additional, nested undo groups within these default groups using the `beginUndoGrouping` and `enableUndoRegistration` methods. You can also turn off the default grouping behavior using `setGroupsByEvent:`.

The Undo and Redo Stacks

Undo groups are stored on a stack, with the oldest groups at the bottom and the newest at the top. The undo stack is unlimited by default, but you can restrict it to a maximum number of groups using the `setLevelsOfUndo:` method. When the stack exceeds the maximum, the oldest undo groups are dropped from the bottom.

Initially, both stacks are empty. Recording undo operations adds to the undo stack, but the redo stack remains empty until undo is performed. Performing undo causes the reverting operations in the latest group to be applied to their objects. Since these operations cause changes to the objects' states, the objects presumably register new operations with the NSUndoManager, this time in the reverse direction from the original operations. Since the NSUndoManager is in the process of performing undo, it records these operations as redo operations on the redo stack. Consecutive undos add to the redo stack. Subsequent redo operations pull the operations off the redo stack, apply them to the objects, and push them back onto the undo stack.

The redo stack's contents last as long as undo and redo are performed successively. However, because applying a new change to an object invalidates the previous changes, as soon as a new undo operation is registered, the redo stack is cleared. This prevents redo from returning objects to an inappropriate prior state. You can check for the ability to undo and redo with the `canUndo` and `canRedo` methods.

Registering Undo Operations

To add an undo operation to the undo stack, you must register it with the object that performs the undo operation. NSUndoManager supports two ways to register undo operations: one based on a simple selector with a single object argument, and one based on a general NSInvocation (which allows any number and type of arguments). In the first type of operation, when an object changes, the object itself (or another object acting on its behalf) registers the change with the undo manager, passing an argument that holds the attributes of the object prior to the change. (This argument is frequently an NSDictionary object, but it can be any object.) Performing the undo then involves resetting the object with these attributes. Invocation-based undo is useful for undoing specific state-changing methods, such as a `setFont:color:` method.

In most applications a single instance of NSUndoManager belongs to an object that contains or manages other objects. This is particularly the case with document-based applications, where each NSDocument object is responsible for all undo and redo operations for a document. An object such as this is often called the NSUndoManager's client. Each client object has its own NSUndoManager. The client claims exclusive right to alter its undoable objects so that it can record undo operations for all changes. In the specific case of documents, this scheme keeps each pair of undo and redo stacks separate so that when an undo is performed, it applies to the focal document in the application (typically the one displayed in the key window). It also relieves the individual objects in a document from having to know the identity of their NSUndoManager or from having to track changes to themselves.

However, an object that is changed can have its own NSUndoManager and perform its own undo and redo operations. For example, you could have a custom view that displays images dragged into it; with each successful drag operation, it registers a new undo group. If the view is then selected (that is, made first responder) and the Undo command applied, the previously displayed image would be redisplayed.

Simple Undo

To record a simple undo operation, you need only invoke `registerUndoWithTarget:selector:object:`, giving the object to be sent the undo operation selector, the selector to invoke, and an argument to pass with that message. The target object is usually not the actual object whose state is changing; instead, it is the client object, a document or container that holds many undoable objects. The argument is an object that captures the state of the object before the change is made. Here is an Objective-C method from the Draw example application (`DrawDocument.m`):

```
- (void)setGridVisible:(NSNumber *)flag {
    BOOL flagValue = [flag boolValue];
    if (gvFlags.showGrid != flagValue) {
        NSNumber *currentValue = [NSNumber numberWithBool:gvFlags.showGrid];
        gvFlags.showGrid = flagValue;
        if (flagValue)
            [graphicView resetGUP];
        [graphicView cache:[graphicView bounds]];
        [undoManager registerUndoWithTarget:self
                selector:@selector(setGridVisible:)
                object:currentValue];
        [undoManager setActionName:GRID_OP];
    }
```

If the user chooses Undo, `setGridVisible:` is invoked with the previous value.

Invocation-Based Undo

For other changes involving specific methods or arguments that are not objects, you can use invocation-based undo, which records an actual message to revert the target object's state. As with simple undo, you record a message that reverts the object to its state before the change. However, in this case you do so by sending the message directly to the NSUndoManager, after preparing it with a special message to note the target, as in this Objective-C example:

```
[[myUndoManager prepareWithInvocationTarget:drawObject]
    setFont:[drawObject font] color:[drawObject color]];
[drawObject setFont:newFont color:newColor];
```

The `prepareWithInvocationTarget:` method records the argument as the target of the undo operation about to be established. Following this, you send the message that reverts the target's state—in this case, `setFont:color:`. Because NSUndoManager does not respond to this method, `forwardInvocation:` is invoked, which NSUndoManager implements to record the NSInvocation containing the target, selector, and all arguments. Performing undo thus results in *`drawObject`* being sent a `setFont:color:` message with the old values.

Performing Undo and Redo

Performing undo and redo is usually as simple as sending `undo` and `redo` messages to the NSUndoManager. `undo` closes the last open undo group and then applies all the undo operations in that group (recording any undo operations as redo operations instead). `redo` likewise applies all the redo operations on the top redo group.

`undo` is intended for undoing top-level groups, and should not be used for nested undo groups. If any unclosed, nested undo groups are on the stack when `undo` is invoked, it raises an exception. To undo nested groups, you must explicitly close the group with an `enableUndoRegistration` message, then use `undoNestedGroup` to undo it. Note also that if you turn off automatic grouping by event with `setGroupsByEvent:`, you must explicitly close the current undo group with `enableUndoRegistration` before invoking either undo method.

Cleaning the Undo Stack

NSUndoManager does not retain the targets of undo operations, for several reasons. Foremost is that the client—the object performing undo operations—typically owns the NSUndoManager; thus for the NSUndoManager to retain the target would create cycles. The NSUndoManager does contain references to the targets of undo operations, however, which it uses to send undo messages when undo is performed. If a target object has been deallocated, and an undo message is sent to it, errors result.

To remedy this, the client must take care to clear undo operations for targets that are being deallocated. This typically occurs in one of three ways:

- The client is the exclusive owner of the NSUndoManager and the target of all undo operations. In this case the client can simply release the NSUndoManager in its `dealloc` method or deconstructor.
- The client shares the NSUndoManager with other clients. To handle this the client should send `removeAllActionsWithTarget:` (argument of `self`) to the NSUndoManager before releasing it in its `dealloc` method.
- The client registers objects other than itself for undo operations. Here either the client must watch for the other objects being deallocated in order to send `removeAllActionsWithTarget:`, or the other objects must do so themselves when deallocated (which requires that they have a reference to the NSUndoManager). This is likely to be needed with invocation-based undo.

In a more general sense, it sometimes makes sense to clear all undo and redo operations. Some applications might want to do this when saving a document, for example. To this end, NSUndoManager defines the `removeAllActions` method, which clears both stacks.

Setting Action Names

NSUndoManager provides the `setActionName:` method to qualify the Undo and Redo command titles in the Edit menu. You pass the string you want appended to “Undo” and “Redo” in the menu items when the current undo group is at the top of the undo and redo stacks. Take, as an example, a graphics application that allows users to add circles, fill them with a color, and delete them. With this method, you could set the name of each action to “Delete”, “Fill”, and “Add Circle”. Upon successive undos, the Undo menu item shows up as “Undo Delete,” “Undo Fill,” and “Undo Add Circle.”

NSUndoManager automatically localizes the “Undo” and “Redo” portion of the command titles, but merely appends the action name to them. You should localize the action names yourself. If you want to further customize how these titles are localized, override `undoMenuTitleForUndoActionName:` and `redoMenuTitleForUndoActionName:`.

Using Undo in Applications

The Application Kit supplements the behavior of NSUndoManager in several ways. It offers default undo and redo behavior in text. It includes APIs for managing the action names that appear with "Undo" and "Redo" in an application's menu. And it establishes a framework for the distribution and selection of NSUndoManagers in an application.

Undo and the Responder Chain

As stated earlier, an application can have one or more clients: objects that register and perform undo operations in their local contexts. Each of these objects has its own NSUndoManager and the associated undo and redo stacks. One example of this scenario involves custom views, each a client of an NSUndoManager. For example, you could have a window with two custom views; each view can display text in changeable attributes (such as font, color, and size) and users can undo (or redo) each change to any attribute in either of the views. NSResponder and NSWindow define methods to help you control the context of undo operations within the view hierarchy.

NSResponder declares the `undoManager` method for most objects that inherit from it (namely, windows and views). When the first responder of an application receives an `undo` or `redo` message, NSResponder goes up the responder chain looking for a next responder that returns an NSUndoManager object from `undoManager` . Any returned NSUndoManager object is used for the undo or redo operation. If the `undoManager` message wends its way up the responder chain to the window, the NSWindow object queries its delegate with `windowWillReturnUndoManager:` to see if the delegate has an NSUndoManager. If the delegate does not implement this method, the NSWindow creates an NSUndoManager for the window and all its views.

Document-based applications often make their NSDocument objects the delegates of their windows and have them respond to the `windowWillReturnUndoManager:` message by returning the NSUndoManager used for the document. These applications can also make each NSWindowController the delegate of its window—the window controller implements `windowWillReturnUndoManager:` to get the NSUndoManager from its document and return it. Here are examples for Objective-C and Java:

```
return [[self document] undoManager]; // Objective-C
```

```
return this.document().undoManager(); // Java
```

NSTextView

NSTextViews provide undo and redo behavior by default. For your application to take advantage of this feature, however, it must send `setAllowsUndo:` with an argument of `YES` to the text view. If you want a text view to have its own NSUndoManager (and not the window's), have the text view's delegate implement `undoManagerForTextView:`, to return the NSUndoManager.

The default undo and redo behavior applies to text fields and text in cells as long as the field or cell is the first responder (that is, the focus of keyboard actions). Once the insertion point leaves the field or cell, prior operations cannot be undone.

Using Undo Notifications

An NSUndoManager regularly posts checkpoint notifications to synchronize the inclusion of undo operations in undo groups. Objects sometimes delay performing changes, for various reasons. This means they may also delay registering undo operations for those changes. Because NSUndoManager collects individual operations into groups, it must be sure to synchronize its client with the creation of these groups so that operations are entered into the proper undo groups. To this end, whenever an NSUndoManager opens or closes a new undo group (except when it opens a top-level group), it posts an NSUndoManagerCheckpointNotification so observers can apply their pending undo operations to the group in effect. The NSUndoManager's client should register itself as an observer for this notification and record undo operations for all pending changes upon receiving it.

NSUndoManager also posts a number of other notifications at specific intervals: when a group is created, when a group is closed, and just before and just after both undo and redo operations.

Document Revision History

This table describes the changes to *Undo Architecture*.

Date	Notes
2002-11-12	Revision history was added to existing topic. It will be used to record changes to the content of the topic.