NSMenu

Inherits From:	NSObject
Conforms To:	NSCoding NSObject (NSObject)
Declared In:	AppKit/NSMenu.h

Class Description

This class defines an object that manages an application's menus. An NSMenu object displays a list of items that a user can choose from. When an item is clicked, it may either issue a command directly or bring up another menu (a *submenu*) that offers further choices. An NSMenu object's choices are implemented with NSMenuItem objects. Each menu item can be configured either to send its action message to a target or to open a submenu.

It's typically more convenient to use Interface Builder to construct your application's menus—see Interface Builder's Help for more information about using this application. NSMenu and NSMenuItem provide you with additional flexibility to dynamically contruct or modify your application's menus at run time.

Exactly one NSMenu created by the application is designated as the main menu for the application (with NSApplication's **setMainMenu:** method). Depending on the user interface of the host system, the main menu displays itself as a free standing window with a title bar and a list of menu items, as a menu bar with no title, or in some other form. The form a menu takes in the user interface may limit which methods in the class's interface actually have an effect; for example, on Microsoft Windows submenus can't be detached. In all cases, however, methods return values that reflect their actual state or ability (**isTornOff** always returns NO on Microsoft Windows, for example).

A free standing main menu is displayed on top of all other windows whenever the application is active, and can be moved by the user by dragging its title bar. When a submenu is opened, it appears attached to the right of its supermenu with a title bar, allowing the user to drag it away from its supermenu so that it remains on the screen. A detached submenu displays a close button to allow the user to dismiss it (the main menu, of course, never displays a close button). If the user moves a menu window while a submenu is attached, the submenu follows its supermenu. If a menu window lies partly off-screen, when the user tracks the mouse pointer to the edge of the screen, by holding down the mouse button and dragging the mouse pointer, the menu temporarily shifts onto the screen (along with any attached super- or submenus), allowing the user to access all of its items.

Where the main menu appears as a menu bar, it doesn't display a title, nor do its submenus. The submenus of a menu bar are typical drop-down menus, and submenus of these appear to the right or left, depending on the available screen space. Submenus of a menu bar typically can't be detached by the user.

NSMenu supports the assignment of keyboard equivalents to its menu items. On Microsoft Windows, the class also supports the assignment of mnemonics to menu items. Any menu item, except those that open submenus, can have a key equivalent, but whether they should depends on the host system's user interface guidelines. Unlike keyboard equivalents, mnemonics only function when their menu is active, and they can be assigned to menu items which open submenus.

See the NSMenuItem protocol, NSMenuItem class, and NSMenuActionResponder protocol specifications for more information.

Method Types

Controlling allocation zones	+ menuZone + setMenuZone:
Creating an NSMenu	– initWithTitle:
Setting up menu commands	– addItemWithTitle:action:keyEquivalent: – insertItemWithTitle:action:keyEquivalent:atIndex: – removeItem: – itemArray
Finding menu items	– itemWithTag: – itemWithTitle:
Managing submenus	– setSubmenu:forItem: – submenuAction: – attachedMenu – isAttached – isTornOff – locationForSubmenu: – supermenu
Enabling and disabling menu items	– autoenablesItems – setAutoEnablesItems: – update
Handling keyboard equivalents	– performKeyEquivalent:
Updating menu layout	– menuChangedMessagesEnabled – setMenuChangedMessagesEnabled: – sizeToFit
Displaying context-sensitive help	– helpRequested:

Class Methods

menuZone

+ (NSZone *)**menuZone**

Returns the zone from which NSMenus should be allocated, creating it if necessary.

setMenuZone:

+ (void)**setMenuZone:**(NSZone *)*zone*

Sets the zone from which NSMenus should be allocated to *zone*.

Instance Methods

addItemWithTitle:action:keyEquivalent:

– (id <NSMenuItem>)**addItemWithTitle:**(NSString *)*aString*
action:(SEL)*aSelector*
keyEquivalent:(NSString *)*keyEquiv*

Adds a new item with title *aString*, action *aSelector*, and key equivalent *keyEquiv* to the end of the menu. Returns the new menu item.

attachedMenu

– (NSMenu *)**attachedMenu**

Returns the menu currently attached to the receiver or **nil** if there's no such object.

autoenablesItems

– (BOOL)**autoenablesItems**

Returns whether the receiver automatically enables and disables its menu items based on the NSMenuActionResponder informal protocol. By default NSMenus do autoenable their menu items. See that protocol specification for more information.

See also: – setAutoenablesItems:

helpRequested:

– (void)**helpRequested:**(NSEvent *)*event*

Overridden by subclasses to implement specialized context-sensitive help behavior by causing the Help manager to display the help associated with the receiver. Never invoke this method directly.

See also: – **showContextHelpForObject:locationHint:** (NSHelpManager)

initWithTitle:

– (id)**initWithTitle:**(NSString *)*aTitle*

Initializes and returns a new menu using *aTitle* for its title. This method is the designated initializer for the class. Returns **self**.

insertItemWithTitle:action:keyEquivalent:atIndex:

– (id <NSMenuItem>)**insertItemWithTitle:**(NSString *)*aString*
action:(SEL)*aSelector*
keyEquivalent:(NSString *)*keyEquiv*
atIndex:(unsigned int)*index*

Adds a new item at *index* having the title *aString*, action *aSelector*, and key equivalent *keyEquiv*. Returns the new menu item.

isAttached

– (BOOL)**isAttached**

Returns YES if the receiver is currently attached to another menu, NO otherwise. This method always returns NO on Microsoft Windows.

isTornOff

– (BOOL)**isTornOff**

Returns NO if the receiver is off-screen or attached to another menu (or if it’s the main menu), YES otherwise. This method always returns NO on Microsoft Windows.

itemArray

– (NSArray *)**itemArray**

Returns the receiver's menu items.

ItemWithTag:

– (id <NSMenuItem>)**itemWithTag:**(int)*aTag*

Returns the first menu item in the receiver that has *aTag* as its tag.

itemWithTitle:

– (id <NSMenuItem>)**itemWithTitle:**(NSString *)*aString*

Returns the first menu item in the receiver that has *aString* as its title.

locationForSubmenu:

– (NSPoint)**locationForSubmenu:**(NSMenu *)*aSubmenu*

On Mach, returns the screen coordinates where *aSubmenu* will be displayed when it's opened as a submenu of the receiver (regardless of its current location). On Microsoft Windows, the coordinates that are reutrned are not meaningful.

menuChangedMessagesEnabled

– (BOOL)**menuChangedMessagesEnabled**

Returns YES if messages are being sent to the application's windows upon each change to the menu, NO otherwise.

See also: – **setMenuChangedMessagesEnabled:**

performKeyEquivalent:

– (BOOL)**performKeyEquivalent:**(NSEvent *)*theEvent*

Searches for a menu item in the receiver, or on Microsoft Windows in any of its submenus as well, whose key equivalent exactly matches the character, or character sequence, of the keyboard event *theEvent* and whose modifier flags match the key-equivalent modifier mask in *theEvent*, and causes that item to send its action message.

removeItem:

– (void)**removeItem:**(id <NSMenuItem>)*anItem*

Removes *anItem* from the receiver.

setAutoenablesItems:

– (void)**setAutoenablesItems:**(BOOL)*flag*

Controls whether the receiver automatically enables and disables its menu items based on the NSMenuActionResponder informal protocol. If *flag* is YES, menu items are automatically enabled and disabled. If *flag* is NO, menu items are not automatically enabled or disabled. See the NSMenuActionResponder protocol specification for more information.

See also: – autoenablesItems

setMenuChangedMessagesEnabled:

– (void)**setMenuChangedMessagesEnabled:**(BOOL)*flag*

Controls whether the receiver sends messages to the application's windows upon each menu change. To avoid the "flickering" effect of many successive menu changes, invoke this method with NO as *flag*, make changes to the menu, and invoke the method again with YES as *flag*. This has the effect of batching changes and having them applied all at once.

See also: – **menuChangedMessagesEnabled**

setSubmenu:forItem:

– (void)**setSubmenu:**(NSMenu *)*aMenu* **forItem:**(id <NSMenuItem>)*anItem*

Makes *aMenu* a submenu controlled by *anItem*, automatically setting *anItem*'s action to **submenuAction:**.

setTitle:

– (NSString *)**setTitle:**(NSString *)*aString*

Sets the receiver's title to *aString*.

See also: – title

sizeToFit

– (void)**sizeToFit**

Resizes the receiver to exactly fit its items. On Microsoft Windows, this method has no effect.

submenuAction:

– (void)**submenuAction:**(id)*sender*

This is the action method assigned to menu items that open submenus. Never invoke this method directly.

supermenu

– (NSMenu *)**supermenu**

Returns the receiver's supermenu or **nil** if it has none.

title

– (NSString *)**title**

Returns the receiver's title.

See also: – setTitle:

update

– (void)**update**

Enables or disables the receiver's menu items based on the NSMenuActionResponder informal protocol and sizes the menu to fit its current menu items if necessary. See the NSMenuActionResponder protocol specification for more information.