NSMenuItem

Adopted By: NSMenuItem

Conforms To: NSCoding, NSCopying
NSObject (NSObject)

Declared In: AppKit/NSMenuItem.h

Protocol Description

The NSMenuItem protocol declares methods that are used to manipulate command items in menus. With some implementations of the OpenStep specification (including NeXT's OPENSTEP), you cannot replace the NSMenuItem class with a different class which conforms to the NSMenuItem protocol. You may, however, subclass the NSMenuItem class if necessary.

See the NSMenu and NSMenuItem class specifications for more information on menus.

Keyboard Equivalents

An object conforming to the NSMenuItem protocol can be assigned a keyboard eqivalent, so that when the user types it the menu item's action is sent. The keyboard eqivalent is defined in two parts. First is the basic key equivalent, which must be a Unicode character that can be generated by a single key press without modifier keys (Shift excepted). It is also possible to use a sequence of Unicode characters so long as the user's key mapping is able to generate the sequence with a single key press. The basic key equivalent is set using **setKeyEquivalent:** and returned by **keyEquivalent**. The second part defines the modifier keys that must also be pressed. This is set using **setKeyEquivalentModifierMask:** and returned by **keyEquivalentModifierMask**. The modifier mask by default includes NSCommandKeyMask, and may also include the masks for the Shift, Alternate, or other modifier keys. Specifying keyboard equivalents in two parts allows you to define a modified keyboard equivalent without having to know which character is generated by the basic key plus the modifier. For example, you can define the keyboard equivalent Command-Alt-f without having to know which character is generated by typing Alt-f.

Certain methods in the NSMenuItem protocol can override assigned keyboard equivalents with those the user has specified in the defaults system. The **setUsesUserKeyEquivalents:** protocol method turns this behavior on or off, and **usesUserKeyEquivalents** returns its status. To determine the user-defined key equivalent for an NSMenuItem object, invoke the **userKeyEquivalent** instance method. If user-defined key equivalents are active and an NSMenuItem object has a user-defined key equivalent, its **keyEquivalent** method returns the user-defined key equivalent and not the one set using **setKeyEquivalent:**.

Mnemonics

On certain platforms, currently including Microsoft Windows, an object conforming to the NSMenuItem protocol can also be assigned a mnemonic. Mnemonics can be assigned on other platforms as well, however, they won't have any effect. Mnemonics are represented by an underlined character in the title of a menu item. The mnemonic can be any character that can be generated by a single key press without modifier keys (Shift excepted). When the menu is active, the user can type the underlined character in the menu item in order to activate that menu item. On Microsoft Windows a user activates the menu by pressing the Alternate key. A particular mnemonic character should only be used once within the set of menu items contained either in the same menu as the menu item or in the application's main menu.

Method Types

Setting the target and action	– setTarget: – target – setAction: – action
Setting the title	– setTitle: – title
Setting the tag	– setTag: – tag
Enabling a menu item	– setEnabled: – isEnabled
Checking for a submenu	– hasSubmenu
Managing key equivalents	– setKeyEquivalent: – keyEquivalent – setKeyEquivalentModifierMask: – keyEquivalentModifierMask
Managing mnemonics	– setMnemonicLocation: – mnemonicLocation – setTitleWithMnemonic: – mnemonic
Managing user key equivalents	+ setUsesUserKeyEquivalents: + usesUserKeyEquivalents – userKeyEquivalent
Representing an object	– setRepresentedObject: – representedObject

Class Methods

setUsesUserKeyEquivalents:

+ (void)**setUsesUserKeyEquivalents:**(BOOL)*flag*

If *flag* is YES, menu items conform to user preferences for key equivalents; otherwise, the key equivalents originally assigned to the menu items are used.

See also: + usesUserKeyEquivalents, – userKeyEquivalent

usesUserKeyEquivalents

+ (BOOL)**usesUserKeyEquivalents**

Returns YES if menu items conform to user preferences for key equivalents; otherwise, returns NO.

See also: + setUsesUserKeyEquivalents:, – userKeyEquivalent

Instance Methods

action

– (SEL)**action**

Returns the receiver's action method.

See also: – target, – setAction:

hasSubmenu

– (BOOL)**hasSubmenu**

Returns YES if the receiver has a submenu, NO if it doesn't.

See also: – setSubmenu:forItem:(NSMenu)

isEnabled

– (BOOL)**isEnabled**

Returns YES if the receiver is enabled, NO if not.

See also: – setEnabled:

keyEquivalent

– (NSString *)**keyEquivalent**

Returns the receiver's unmodified keyboard equivalent, or the empty string if one hasn't been defined. Use **keyEquivalentModifierMask** to determine the modifier mask for the key equivalent.

See also: – userKeyEquivalent, – mnemonic, – setKeyEquivalent:

keyEquivalentModifierMask

– (unsigned int)**keyEquivalentModifierMask**

Returns the receiver's keyboard equivalent modifier mask.

See also: – setKeyEquivalentModifierMask:

mnemonic

– (NSString *)mnemonic

Returns the character in the menu item title that appears underlined for use as a mnemonic. If there is no mnemonic character, returns an empty string.

See also: – **setTitleWithMnemonic:**

mnemonicLocation

– (unsigned)**mnemonicLocation**

Returns the position of the underlined character in the menu item title used as a mnemonic. The position is the zero based index of that character in the title string. If the receiver has no mnemonic character, returns NSNotFound.

See also: – **setMnemonicLocation:**

representedObject

– (id)**representedObject**

Returns the object that the receiving menu item represents. For example, you might have a menu list the names of views that are swapped into the same panel. The represented objects would be the appropriate NSView objects. The user would then be able to swtich back and forth between the different views that are displayed by selecting the various menu items.

See also: – **tag**, – **setRepresentedObject:**

setAction:

– (void)**setAction:**(SEL)*aSelector*

Sets the receiver's action method to *aSelector*.

See also: – setTarget:, – action

setEnabled:

– (void)**setEnabled:**(BOOL)*flag*

Sets whether the receiver is enabled based on *flag*. If a menu item is disabled, it's keyboard equivalent and mnemonic are also disabled. See the NSMenuActionResponder informal protocol specification for cautions regarding this method.

See also: – isEnabled

setKeyEquivalent:

– (void)**setKeyEquivalent:**(NSString *)*aString*

Sets the receiver's unmodified key equivalent to *aString*. Use **setKeyEquivalentModifierMask:** to set the appropriate mask for the modifier keys for the key equivalent.

See also: – setMnemonicLocation:, – keyEquivalent

setKeyEquivalentModifierMask:

– (void)**setKeyEquivalentModifierMask:**(unsigned int)*mask*

Sets the receiver's keyboard equivalent modifiers (indicating modifiers such as the Shift or Alternate keys) to those in *mask*. *mask* is an integer bit field containing any of these modifier key masks, combined using the C bitwise OR operator:

NSShiftKeyMask
NSAlternateKeyMask
NSCommandKeyMask

On Mach, you should always set NSCommandKeyMask in *mask*; on Microsoft Windows, this is not required.

NSShiftKeyMask is relevant only for function keys; that is, for key events whose modifier flags include NSFunctionKeyMask. For all other key events NSShiftKeyMask is ignored and characters typed while the Shift key is pressed are interpreted as the shifted versions of those characters; for example, Command-Shift-'c' is interpreted as Command-'C'.

See the NSEvent class specification for more information about modifier mask values.

See also: – keyEquivalentModifierMask

setMnemonicLocation:

– (void)**setMnemonicLocation:**(unsigned)*location*

Sets the character of the menu item title at *location* that is to be underlined. *location* must be between 0 and 254. This character identifies the access key on Windows by which users can access the menu item.

See also: – **mnemonicLocation**

setRepresentedObject:

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

Sets the object represented by the receiver to *anObject*. By setting a represented object for a menu item you make an association between the menu item and that object. The represented object functions as a more specific form of tag that allows you to associate any object, not just an **int**, with the items in a menu.

For example, an NSView object might be associated with a menu item—when the user selects the menu item, the represented object is fetched and displayed in a panel. Several menu items might control the display of multiple views in the same panel.

See also: – **setTag:**, – **representedObject**

setTag:

– (void)**setTag:**(unsigned int)*anInt*

Sets the receiver's tag to *anInt*.

See also: – **setRepresentedObject:**, – **tag**

setTarget:

– (void)**setTarget:**(id)*anObject*

Sets the receiver's target to *anObject*.

See also: – setAction:, – target

setTitle:

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

Sets the receiver's title to *aString*.

See also: – title

setTitleWithMnemonic:

– (void)**setTitleWithMnemonic:**(NSString *)*aString*

Sets the title of a menu item with a character underlined to denote an access key (Windows only). Use an ampersand character to mark the character (the one following the ampersand) to be underlined. For example, the following message causes the 'c' in 'Receive' to be underlined:

```
[aMenuItem setTitleWithMnemonic:NSLocalizedString(@"Re&ceive")];
```

See also: – **mnemonic**, – **setMnemonicLocation:**

tag

– (unsigned int)**tag**

Returns the receiver's tag.

See also: – **representedObject**, – **setTag:**

target

– (id)**target**

Returns the receiver's target.

See also: – action, – setTarget:

title

– (NSString *)**title**

Returns the receiver's title.

See also: – setTitle:

userKeyEquivalent

– (NSString *)**userKeyEquivalent**

Returns the user-assigned key equivalent for the receiver.

See also: – keyEquivalent