Article

Supporting Writing Tools via the pasteboard

Adopt a simplified version of the Writing Tools experience in a custom view using the pasteboard and macOS services.

Overview

Writing Tools works automatically with `NSTextView` in AppKit, but you can also use it with any custom subclass of `NSView` that has text. To use Writing Tools with a custom view, you have to write some additional code to transfer text between your view and the Writing Tools feature. You also have to provide a contextual menu for your view, which AppKit modifies to allow people to launch the Writing Tools UI.

You can add Writing Tools support to both read-only and editable views. When your view is read-only, Writing Tools leaves the updated text on the pasteboard instead of incorporating it back into your view. For editable views, your code replaces the view's current text selection with the rewritten content that Writing Tools provides.

> **Note**
>
> If you display text using `NSTextView`, you don't need to do additional work to support Writing Tools, but you can customize your app's behavior when the feature is active. For information about how to customize your app's behavior, see Customizing Writing Tools behavior for AppKit views.

Add a contextual menu to your custom view

People launch the Writing Tools UI through interactions with your view's contextual menu. If your view doesn't currently have a contextual menu, add one and populate it with app-specific commands. For example, you might add commands to cut, copy, or paste text. When you display the menu, the system automatically adds Writing Tools–specific commands to it, based on the level of support your app provides.

Construct a contextual menu for your view and save it in the view's menu property. When this property contains a menu, `NSView` displays it as a contextual menu automatically when appropriate events occur. For example, the view displays it in response to a right mouse-down event. The following code shows you how to create a menu, add some menu items to it, and save it to the view's menu property:

```
self.menu = NSMenu()
self.menu?.addItem(NSMenuItem(title: "Cut",
                         action: #selector(cut(_:)),
                         keyEquivalent: ""))
self.menu?.addItem(NSMenuItem(title: "Copy",
                         action: #selector(copy(_:)),
                         keyEquivalent: ""))
self.menu?.addItem(NSMenuItem(title: "Paste",
                         action: #selector(paste(_:)),
                         keyEquivalent: ""))
```

If you want to customize your view's contextual menu before you display it, implement the `menu(for:)` method to return the menu you want for appropriate events. You can also display a contextual menu programmatically using the menu's `popUpContextMenu(_:with:for:)` method.

Provide a requester object to coordinate interactions

The AppKit support for Writing Tools in custom views relies on the same infrastructure that apps use to offer services from the Services menu. Before displaying your view's contextual menu, AppKit calls the view's `validRequestor(forSendType:returnType:)` method to determine if your view supports specific types of text data. If you do, you return an `NSServicesMenuRequestor` object from that method to enable the Writing Tools feature for your view. AppKit then automatically adds commands to your view's contextual menu to let someone launch the feature's UI.

When determining whether your view supports Writing Tools, AppKit calls your view's `validRequestor(forSendType:returnType:)` method multiple times with different pasteboard types. Writing Tools can work with rich text or plain text content, and AppKit asks for a requestor object for the `rtf` pasteboard type first. If you don't provide an object for that type, AppKit asks for a requestor object for the `string` pasteboard type. If your view is editable, return a requestor object when both method parameters contain the supported pasteboard type. For a read-only view, return a requestor object when the `sendType` parameter contains the supported type and the `returnType` parameter is `nil`.

The following code shows an implementation of the `validRequestor(forSendType:returnType:)` method in a read-only view that displays a plain text string. Because the view is read-only, the implementation returns the view object only when the `returnType` parameter is `nil`. If you don't perform that check, AppKit assumes your view is editable and tries to integrate the rewritten text back into your view.

```
override func validRequestor(forSendType
          sendType: NSPasteboard.PasteboardType?,
          returnType: NSPasteboard.PasteboardType?) -> Any? {
    if sendType == .string && returnType == nil {
        return self
    }
    return super.validRequestor(forSendType: sendType,
                                returnType: returnType)
}
```

AppKit uses your requestor object to fetch your view's selected text, and to deliver the rewritten text back to you. Adopt the `NSServicesMenuRequestor` protocol in the object you use to manage the text selection for your view. You can adopt this protocol in your view, or in a separate object you use to manage the backing store for that view.

Write your view's selected text to the pasteboard

The requestor object you provide to AppKit is responsible for writing your custom view's text selection to the pasteboard. When someone engages a Writing Tools feature for your view, AppKit calls your requestor object's `writeSelection(to:types:)` method to get the current text. Write text to the pasteboard in the format that your view's `validRequestor(forSendType:returnType:)` method indicates you support. For example, if you provided a requestor object for the `rtf` type, write data to the pasteboard in the RTF format. You don't need to write multiple data formats to the pasteboard to ensure interoperability with other apps. For Writing Tools operations, AppKit provides you with a dedicated pasteboard that's private to the operation.

For plain text, write the text directly to the pasteboard using `setString(_:forType:)` method of `NSPasteboard`. For rich text, make sure your view manages its content using an attributed string. You can generate a `Data` object in the RTF format directly from the contents of an `NSAttributedString` object. If you store your view's text using a Swift `AttributedString` type instead, create an `NSAttributedString` from your Swift string first. Set the data on the pasteboard and return true to pass the data along to the Writing Tools system, as shown in the following example:

```
func writeSelection(to pboard: NSPasteboard,
                    types: [NSPasteboard.PasteboardType]) -> Bool {
    pboard.setString(text, forType: .string)
    return true
}
```

Read updated text from the pasteboard

If your view contains editable text, implement the `readSelection(from:)` method in your requestor object to integrate any rewritten content back into your view. AppKit calls the `readSelection(from:)` method after the person accepts any Writing Tools changes. In your implementation, read the updated text from the provided pasteboard and replace your view's current text selection with the new text.

> Note
>
> If you indicate that your view is read-only in your `validRequestor(forSendType:returnType:)` method, AppKit doesn't call your requestor object's `readSelection(from:)` method. Instead, AppKit places the rewritten text on the general pasteboard so that the person can paste it elsewhere.

Writing Tools calls your view's `readSelection(from:)` method first to handle text insertions and replacements. If you implement that method, insert the provided text, replacing any previous text as needed. If you don't implement that method in your view, Writing Tools calls the `insertText(_:replacementRange:)` method instead.

Customize the anchor placement for the Writing Tools popover

When AppKit displays the Writing Tools UI, it anchors the popover to your entire view. To change the anchor point, you need to supply a new anchor point for your view. For example, you might want to anchor the Writing Tools UI to the rectangle that contains the current text selection instead.

To anchor the Writing Tools UI to a custom part of your view's content, adopt the `NSViewContentSelectionInfo` protocol in your view and set the `selectionAnchorRect` property to a rectangle in your view's coordinate space. For example, for an editable view with an active text selection, set it to the rectangle of the selected text. AppKit places the Writing Tools popover relative to the custom rectangle you provide.

See Also

Writing Tools for custom views

Adding Writing Tools support to a custom AppKit view

Integrate Writing Tools support, including support for inline replacement animations, to your custom text views on macOS.

`class NSWritingToolsCoordinator`

An object that manages interactions between Writing Tools and your custom text view.

`protocol Delegate`

An interface that you use to manage interactions between Writing Tools and your custom text view.

`class Context`

A data object that you use to share your custom view's text with Writing Tools.

`class AnimationParameters`

An object you use to configure additional tasks or animations to run alongside the Writing Tools animations.

{} Enhancing your custom text engine with Writing Tools

Add Writing Tools support to your custom text engine to enhance the text editing experience.