Using and Extending the Xcode Source Editor

Session 414

Mike Swingler Xcode Infrastructure and Editors
Chris Hanson Xcode Infrastructure and Editors
Overview

Using

• New features in Xcode 8
• Other not so new, but useful features
Overview

Using

• New features in Xcode 8
• Other not so new, but useful features

Extending

• How to add your own features
• …and share them!
Demo

New features in the Xcode 8 source editor
Xcode Source Editor Extensions
Enhancing Xcode
Extending Xcode
What you can do
Extending Xcode

What you can do

Add commands to the source editor
Extending Xcode

What you can do

Add commands to the source editor
Edit text
Extending Xcode

What you can do

- Add commands to the source editor
- Edit text
- Change selections
Extending Xcode

What you can do

Add commands to the source editor
Edit text
Change selections
One extension, several commands
Extending Xcode
How it works

Xcode Extensions are Application Extensions
Extending Xcode

How it works

Xcode Extensions are Application Extensions

• Each runs in its own process
Extending Xcode

How it works

Xcode Extensions are Application Extensions

- Each runs in its own process
- Sandboxed and uses entitlements
Extending Xcode

How it works

Xcode Extensions are Application Extensions

• Each runs in its own process
• Sandboxed and uses entitlements
• Gets access to text at invocation
Stability,
Stability, Security,
Stability, Security, Speed
Delivering Xcode Extensions

Getting them into users’ hands

An Xcode Extension is embedded in an application
Delivering Xcode Extensions

Getting them into users’ hands

An Xcode Extension is embedded in an application

• Your App is a great place to put your extension’s preferences
Delivering Xcode Extensions

Getting them into users’ hands

An Xcode Extension is embedded in an application

• Your App is a great place to put your extension’s preferences
• Any other UI you want to provide—no UI in extensions
Delivering Xcode Extensions
Getting them into users’ hands

An Xcode Extension is embedded in an application

• Your App is a great place to put your extension’s preferences
• Any other UI you want to provide—no UI in extensions
• Distribute via the Mac App Store
Delivering Xcode Extensions

Getting them into users’ hands

An Xcode Extension is embedded in an application

• Your App is a great place to put your extension’s preferences
• Any other UI you want to provide—no UI in extensions
• Distribute via the Mac App Store
• Distribute on your own via Developer ID
The Xcode Extension Lifecycle

Startup
The Xcode Extension Lifecycle

Startup

Xcode automatically finds and starts extensions
The Xcode Extension Lifecycle

Startup

Xcode automatically finds and starts extensions

• Extensions are kept alive while the user is likely to need them
The Xcode Extension Lifecycle

Startup

Xcode automatically finds and starts extensions

- Extensions are kept alive while the user is likely to need them

Extensions sent `extensionDidFinishLaunching`
The Xcode Extension Lifecycle

Startup

Xcode automatically finds and starts extensions

• Extensions are kept alive while the user is likely to need them

Extensions sent `extensionDidFinishLaunching`

• Do any needed startup as fast as possible
The Xcode Extension Lifecycle

Startup

Xcode automatically finds and starts extensions

- Extensions are kept alive while the user is likely to need them

Extensions sent `extensionDidFinishLaunching`

- Do any needed startup as fast as possible
- Asynchronous with Xcode and other extensions
The Xcode Extension Lifecycle

Providing commands

Xcode asks each extension for its commands, which can come from:
The Xcode Extension Lifecycle

Providing commands

Xcode asks each extension for its commands, which can come from:

- Your extension’s Info.plist

```xml
<key>NSExtensionAttributes</key>
<dict>
  <key>XCSOURCEEDITORCOMMANDDEFINITIONS</key>
  <array>
    <dict>
      <key>XCSourceEditorCommandClassName</key>
      <string>ChrisFormat.WrapText</string>
      <key>XCSourceEditorCommandIdentifier</key>
      <string>com.example.ChrisFormat.WrapText</string>
      <key>XCSourceEditorCommandName</key>
      <string>Wrap Text</string>
    </dict>
  </array>
  <key>XCSourceEditorExtensionPrincipalClass</key>
  <string>ChrisFormat.ChrisFormatExtension</string>
</dict>
```
The Xcode Extension Lifecycle

Providing commands

Xcode asks each extension for its commands, which can come from:

• Your extension’s Info.plist
• Your extension’s `commandDefinitions` property, overriding the Info.plist

```swift
var commandDefinitions = {
    return [ [
        .classNameKey: "ChrisFormat.WrapText",
        .identifierKey: "com.example.ChrisFormat.WrapText",
        .nameKey: "Wrap Text"
    ] ]
}
```
The Xcode Extension Lifecycle
Where commands live
The Xcode Extension Lifecycle
Where commands live

Each extension gets a submenu of the Editor menu for its commands
Each extension gets a submenu of the Editor menu for its commands

- Extensions listed in Finder sort order
The Xcode Extension Lifecycle

Where commands live

Each extension gets a submenu of the Editor menu for its commands

- Extensions listed in Finder sort order
- Commands are in the order the extension provides
The Xcode Extension Lifecycle

Invoking commands

User chooses a command
The Xcode Extension Lifecycle

Invoking commands

User chooses a command

• Selecting menu item
The Xcode Extension Lifecycle

Invoking commands

User chooses a command

• Selecting menu item
• Pressing keyboard equivalent
The Xcode Extension Lifecycle

Invoking commands

User chooses a command

• Selecting menu item
• Pressing keyboard equivalent

Your command is sent an invocation and a callback
The Xcode Extension Lifecycle

Invoking commands

User chooses a command

• Selecting menu item
• Pressing keyboard equivalent

Your command is sent an invocation and a callback

• The invocation contains a text buffer and metadata to operate on
The Xcode Extension Lifecycle

Invoking commands

User chooses a command
- Selecting menu item
- Pressing keyboard equivalent

Your command is sent an invocation and a callback
- The invocation contains a text buffer and metadata to operate on
- The command uses the callback to tell Xcode it’s done
// Commands

public protocol XCSourceEditorCommand : NSObjectProtocol {

    public func perform(with invocation: XCSourceEditorCommandInvocation,
                         completionHandler: (NSError?) -> Void) -> Void

}
public protocol XCSourceEditorCommand : NSObjectProtocol {

    public func perform(with invocation: XCSourceEditorCommandInvocation,
                         completionHandler: (NSError?) -> Void) -> Void

}

public class XCSourceEditorCommandInvocation : NSObject {

    public let commandIdentifier: String

    public var cancellationHandler: () -> Void

    public let buffer: XCSourceTextBuffer

}
// Commands

public protocol XCSourceEditorCommand : NSObjectProtocol {

    public func perform(with invocation: XCSourceEditorCommandInvocation,
                         completionHandler: (NSError?) -> Void) -> Void

}

public class XCSourceEditorCommandInvocation : NSObject {

    public let commandIdentifier: String

    public var cancellationHandler: () -> Void

    public let buffer: XCSourceTextBuffer

}

// Commands

public protocol XCSourceEditorCommand : NSObjectProtocol {

    public func perform(with invocation: XCSourceEditorCommandInvocation,
                         completionHandler: (NSError?) -> Void) -> Void

}

public class XCSourceEditorCommandInvocation : NSObject {

    public let commandIdentifier: String

    public var cancellationHandler: () -> Void

    public let buffer: XCSourceTextBuffer

}
// Commands

public protocol XCSourceEditorCommand : NSObjectProtocol {

    public func perform(with invocation: XCSourceEditorCommandInvocation,
                         completionHandler: (NSError?) -> Void) -> Void

}

public class XCSourceEditorCommandInvocation : NSObject {

    public let commandIdentifier: String

    public var cancellationHandler: () -> Void

    public let buffer: XCSourceTextBuffer

}
public class XCSourceTextBuffer : NSObject {

    public let contentUTI: String

    public let tabWidth: Int
    public let indentationWidth: Int
    public let usesTabsForIndentation: Bool

    public var completeBuffer: String

    public let lines: NSMutableArray<String>
    public let selections: NSMutableArray<XCSourceTextRange>
}

// Text Buffer
public class XCSourceTextBuffer : NSObject {

    public let contentUTI: String

    public let tabWidth: Int
    public let indentationWidth: Int
    public let usesTabsForIndentation: Bool

    public var completeBuffer: String

    public let lines: NSMutableArray<String>
    public let selections: NSMutableArray<XCSourceTextRange>

}
// Text Buffer

public class XCSourceTextBuffer : NSObject {

    public let contentUTI: String

    public let tabWidth: Int
    public let indentationWidth: Int
    public let usesTabsForIndentation: Bool

    public var completeBuffer: String

    public let lines: NSMutableArray<String>
    public let selections: NSMutableArray<XCSourceTextRange>

}
public class XCSourceTextBuffer : NSObject {

    public let contentUTI: String

    public let tabWidth: Int
    public let indentationWidth: Int
    public let usesTabsForIndentation: Bool

    var usesTabsForIndentation: Bool { get }

    Declaration
    Whether tabs are used for indentation, or just spaces. When tabs are used for indentation, indented text is effectively padded to the indentation width using space characters, and then every tab width space characters is replaced with a tab character.

    For example, say an XCSourceTextBuffer instance has a tabWith of 8, an indentationWidth of 4, and its usesTabsForIndentation is true. The first indentation level will be represented by four space characters, the second by a tab character, the third by a tab followed by four space characters, the fourth by two tab characters, and so on.

    Declared In  XcodeKit

}
public class XCSourceTextBuffer : NSObject {

    public let contentUTI: String

    public let tabWidth: Int
    public let indentationWidth: Int
    public let usesTabsForIndentation: Bool

    public var completeBuffer: String

    public let lines: NSMutableArray<String>
    public let selections: NSMutableArray<XCSourceTextRange>

}
public class XCSourceTextBuffer : NSObject {

    public let contentUTI: String

    public let tabWidth: Int
    public let indentationWidth: Int
    public let usesTabsForIndentation: Bool

    public var completeBuffer: String

    public let lines: NSMutableArray<String>
    public let selections: NSMutableArray<XCSourceTextRange>

}

// Text Buffer
// Text Buffer

public class XCSourceTextBuffer : NSObject {

    public let contentUTI: String

    public let tabWidth: Int
    public let indentationWidth: Int
    public let usesTabsForIndentation: Bool

    public var completeBuffer: String

    public let lines: NSMutableArray<String>
    public let selections: NSMutableArray<XCSourceTextRange>

}
public class XCSourceTextBuffer : NSObject {

    public let contentUTI: String
    public let tabWidth: Int
    public let indentationWidth: Int
    public let usesTabsForIndentation: Bool

    public var completeBuffer: String

    public let lines: NSMutableArray<String>
    public let selections: NSMutableArray<XCSourceTextRange>

}
// Positions and Ranges

public class XCSourceTextRange : NSObject, NSCopying {

    public var start: XCSourceTextPosition
    public var end: XCSourceTextPosition

}

public struct XCSourceTextPosition {

    public var line: Int
    public var column: Int

}
public class XCSourceTextRange : NSObject, NSCopying {

    public var start: XCSourceTextPosition
    public var end: XCSourceTextPosition

}

public struct XCSourceTextPosition {

    public var line: Int
    public var column: Int

}
// Positions and Ranges

public class XCSourceTextRange : NSObject, NSCopying {

    public var start: XCSourceTextPosition
    public var end: XCSourceTextPosition

}

public struct XCSourceTextPosition {

    public var line: Int
    public var column: Int

}
Demo

Creating an Xcode source editor extension
Speed

Text editing is “user-synchronous”
Speed

Text editing is “user-synchronous”

Users will invoke your command via typing
Speed

Text editing is “user-synchronous”

Users will invoke your command via typing

User changes to a document are prevented while a command is running
Speed

Text editing is “user-synchronous”

Users will invoke your command via typing

User changes to a document are prevented while a command is running

The user can cancel your command
Speed

Text editing is “user-synchronous”

Users will invoke your command via typing

User changes to a document are prevented while a command is running

The user can cancel your command

• A command that takes a while gets a cancellation banner
Speed

Text editing is “user-synchronous”

Users will invoke your command via typing

User changes to a document are prevented while a command is running

The user can cancel your command

• A command that takes a while gets a cancellation banner
Speed

How Xcode helps
Speed

How Xcode helps

Keeps your extension alive for fast invocation
Speed

How Xcode helps

Keeps your extension alive for fast invocation
Optimizes data transfer for performance
Speed
How Xcode helps

Keeps your extension alive for fast invocation
Optimizes data transfer for performance
Cancellation is immediate for the user
Speed
How you can help Xcode
Speed

How you can help Xcode

Start up quickly
Speed
How you can help Xcode

Start up quickly
Use GCD and follow standard asynchronous patterns
Speed
How you can help Xcode

Start up quickly
Use GCD and follow standard asynchronous patterns
Don’t replace the whole buffer if you don’t have to
Speed
How you can help Xcode

Start up quickly
Use GCD and follow standard asynchronous patterns
Don’t replace the whole buffer if you don’t have to
Handle cancellation quickly
Summary

New features in the source editor
• Documentation comments
• Color and image literals, with code complete

Recent features
• Fuzzy code completion

Xcode source editor extensions
• How they work
• How to make them
More Information

# Related Sessions

<table>
<thead>
<tr>
<th>Session</th>
<th>Location</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimizing App Startup Time</td>
<td>Mission</td>
<td>Wednesday 10:00AM</td>
</tr>
<tr>
<td>Introduction to Xcode</td>
<td>Nob Hill</td>
<td>Thursday 1:40PM</td>
</tr>
<tr>
<td>Creating Extensions for iOS and OS X, Part 1</td>
<td></td>
<td>WWDC 2014</td>
</tr>
<tr>
<td>Creating Extensions for iOS and OS X, Part 2</td>
<td></td>
<td>WWDC 2014</td>
</tr>
<tr>
<td>App Extension Best Practices</td>
<td></td>
<td>WWDC 2015</td>
</tr>
<tr>
<td>Xcode Open Hours</td>
<td>Developer Tools Lab B</td>
<td>Friday 9:00AM</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>Xcode Open Hours</td>
<td>Developer Tools Lab B</td>
<td>Friday 12:00PM</td>
</tr>
<tr>
<td>Xcode Open Hours</td>
<td>Developer Tools Lab B</td>
<td>Friday 3:00PM</td>
</tr>
</tbody>
</table>
