What's New in Core Spotlight

Search on macOS and iOS

Session 231

John Hörnkvist, Spotlight
Lyn Fong, Spotlight
CoreSpotlight on macOS
CoreSpotlight on macOS
Drag and Drop
CoreSpotlight on macOS
Drag and Drop
Quick Look Previews
CoreSpotlight on macOS
Drag and Drop
Quick Look Previews
Ranking
CoreSpotlight on macOS
Drag and Drop
Quick Look Previews
Ranking
Indexing and Metadata
CoreSpotlight on macOS
Drag and Drop
Quick Look Previews
Ranking
Indexing and Metadata
Search
CoreSpotlight on macOS
CoreSpotlight on macOS

Same API as on iOS
CoreSpotlight on macOS

Same API as on iOS

Used by Notes, Safari, and CoreData
CoreSpotlight on macOS

Same API as on iOS

Used by Notes, Safari, and CoreData

Great for databases, shoeboxes
CoreSpotlight on macOS

Same API as on iOS

Used by Notes, Safari, and CoreData

Great for databases, shoeboxes

Not for "documents"
CoreSpotlight on macOS

Same API as on iOS

Used by Notes, Safari, and CoreData

Great for databases, shoeboxes

Not for “documents”

No sharing between users
Drag and Drop
Drag and Drop

Promise drag types when indexed
Drag and Drop

Promise drag types when indexed

App extension fulfills the promise
Your App
Indexing

CoreSpotlight
Your App
Indexing

CoreSpotlight

Item 6  Item 5  Item 4  Item 3  Item 2  Item 1
Drag Types
Drag Types

Uniform Type Identifiers
• Great info on developer.apple.com
Drag Types

Uniform Type Identifiers
• Great info on developer.apple.com

Declare your own types for your data
Drag Types

Uniform Type Identifiers
• Great info on developer.apple.com

Declare your own types for your data

Use well known types for your promises
extension CSSearchableItemAttributeSet {

    // The string value of type identifier can only be used by one providerTypeIdentifier array.
    // An array of types identifiers that owner can provide a NSData representation for.
    open var providerDataTypeIdentifiers: [String]?

    // An array of types identifiers that owner can provided a NSURL to file representation.
    open var providerFileTypeIdentifiers: [String]?

    // An array of types identifiers that owner can provided a NSURL to inplace file representation.
    open var providerInPlaceFileTypeIdentifiers: [String]?
}

extension CSSearchableItemAttributeSet {

    // The string value of type identifier can only be used by one providerTypeIdentifier array.
    // An array of types identifiers that owner can provide a NSData representation for.
    open var providerDataTypeIdentifiers: [String]?

    // An array of types identifiers that owner can provided a NSURL to file representation.
    open var providerFileTypeIdentifiers: [String]?

    // An array of types identifiers that owner can provided a NSURL to inplace file representation.
    open var providerInPlaceFileTypeIdentifiers: [String]?
}

extension CSSearchableItemAttributeSet {

    // The string value of type identifier can only be used by one providerTypeIdentifier array.
    // An array of types identifiers that owner can provide a NSData representation for.
    open var providerDataTypeIdentifiers: [String]? 

    // An array of types identifiers that owner can provided a NSURL to file representation.
    open var providerFileTypeIdentifiers: [String]?

    // An array of types identifiers that owner can provided a NSURL to inplace file representation.
    open var providerInPlaceFileTypeIdentifiers: [String]?
}

extension CSSearchableItemAttributeSet {

    // The string value of type identifier can only be used by one providerTypeIdentifier array.
    // An array of types identifiers that owner can provide a NSData representation for.
    open var providerDataTypeIdentifiers: [String]?

    // An array of types identifiers that owner can provided a NSURL to file representation.
    open var providerFileTypeIdentifiers: [String]?

    // An array of types identifiers that owner can provided a NSURL to inplace file representation.
    open var providerInPlaceFileTypeIdentifiers: [String]?  
}

// Setting up for drag and drop

let attrs : CSSearchableItemAttributeSet = CSSearchableItemAttributeSet(itemContentType: kMyType as String)

attrs.providerFileTypeIdentifiers = [kUTTypeImage as String]
attrs.providerDataTypeIdentifiers = [kUTTypeUTF8PlainText as String]
// Setting up for drag and drop

let attrs : CSSearchableItemAttributeSet = CSSearchableItemAttributeSet(itemContentType: kMyType as String)

attrs.providerFileTypeIdentifiers = [kUTTypeImage as String]
attrs.providerDataTypeIdentifiers = [kUTTypeUTF8PlainText as String]
/ Setting up for drag and drop

let attrs : CSSearchableItemAttributeSet = CSSearchableItemAttributeSet(itemContentType: kMyType as String)

attrs.providerFileTypeIdentifiers = [kUTTypeImage as String]
attrs.providerDataTypeIdentifiers = [kUTTypeUTF8PlainText as String]
// The developer may provided a NSData representation if type was specified in
providerDataTypeIdentifiers property.

    optional public func data(for searchableIndex: CSSearchableIndex, itemIdentifier: String,
typeIdentifier: String) throws -> Data

// The developer may provided a NSURL to file representation representation if type was
specified from providerDataTypeIdentifiers or providerInPlaceFileTypeIdentifiers property.

    optional public func fileURL(for searchableIndex: CSSearchableIndex, itemIdentifier: String,
typeIdentifier: String, inPlace: Bool) throws -> URL
// The developer may provided a NSData representation if type was specified in providerDataTypeIdentifiers property.

    optional public func data(for searchableIndex: CSSearchableIndex, itemIdentifier: String, typeIdentifier: String) throws -> Data

// The developer may provided a NSURL to file representation representation if type was specified from providerDataTypeIdentifiers or providerInPlaceFileTypeInfoIdentifiers property.

    optional public func fileURL(for searchableIndex: CSSearchableIndex, itemIdentifier: String, typeIdentifier: String, inPlace: Bool) throws -> URL
// The developer may provided a NSData representation if type was specified in
// providerDataTypeIdentifiers property.

    optional public func data(for searchableIndex: CSSearchableIndex, itemIdentifier: String,
        typeIdentifier: String) throws -> Data

// The developer may provided a NSURL to file representation representation if type was
// specified from providerDataTypeIdentifiers or providerInPlaceFileTypeIdentifiers property.

    optional public func fileURL(for searchableIndex: CSSearchableIndex, itemIdentifier: String,
        typeIdentifier: String, inPlace: Bool) throws -> URL
override func data(for searchableIndex: CSSearchableIndex, itemIdentifier: String,
typeIdentifier: String) throws -> Data {

    // Request indexed data for the requested picture
    var data = Data(bytes:[84,69,88,84,32,68,65,84,65])

    if var picture = Datastore.sharedDatastore.picture(identifier:itemIdentifier) {
        if typeIdentifier.isEqual(kUTTypeUTF8PlainText as String) {
            data = (picture.asciiImage?.data(using:String.Encoding.utf8))!
        }
    }

    return data
}
override func data(for searchableIndex: CSSearchableIndex, itemIdentifier: String, typeIdentifier: String) throws -> Data {
    // Request indexed data for the requested picture
    var data = Data(bytes:[84,69,88,84,32,68,65,84,65])
    if var picture = Datastore.sharedDatastore.picture(identifier:itemIdentifier) {
        if typeIdentifier.isEqual(kUTTypeUTF8PlainText as String) {
            data = (picture.asciiImage?.data(using:String.Encoding.utf8))!
        }
    }

    return data
}
override func data(for searchableIndex: CSSearchableIndex, itemIdentifier: String, typeIdentifier: String) throws -> Data {

    // Request indexed data for the requested picture
    var data = Data(bytes: [84, 69, 88, 84, 32, 68, 65, 84, 65])

    if var picture = Datastore.sharedDatastore.picture(identifier: itemIdentifier) {
        if typeIdentifier.isEqual(kUTTypeUTF8PlainText as String) {
            data = (picture.asciiImage?.data(using: String.Encoding.utf8))!
        }
    }

    return data
}
override func data(for searchableIndex: CSSearchableIndex, itemIdentifier: String, typeIdentifier: String) throws -> Data
{

    // Request indexed data for the requested picture
    var data = Data(bytes:[84,69,88,84,32,68,65,84,65])

    if var picture = Datastore.sharedDatastore.picture(identifier:itemIdentifier) {
        if typeIdentifier.isEqual(kUTTypeUTF8PlainText as String) {
            data = (picture.asciiImage?.data(using:String.Encoding.utf8))!
        }
    }

    return data
}
override func fileURL(for searchableIndex: CSSearchableIndex, itemIdentifier: String, typeIdentifier: String, inPlace: Bool) throws -> URL {

    // Request indexed URL for the requested picture
    var url = URL(string:"file://")!

    if let picture = Datastore.sharedDatastore.picture(identifier:itemIdentifier) {
        if typeIdentifier.isEqual(kUTTypeImage as String) {
            url = picture.thumbnailURL!
        }
    }

    return url
}
}
override func fileURL(for searchableIndex: CSSearchableIndex, itemIdentifier: String, typeIdentifier: String, inPlace: Bool) throws -> URL {
    // Request indexed URL for the requested picture
    var url = URL(string:"file:///")!

    if let picture = Datastore.sharedDatastore.picture(identifier:itemIdentifier) {
        if typeIdentifier.isEqual(kUTTypeImage as String) {
            url = picture.thumbnailURL!
        }
    }

    return url
}
Drag and Drop

Summary

Declare drag types at indexing time

CoreSpotlight extension is critical
• It fulfills your promises

Make it fast!

iOS and macOS
Quick Look Previews
For Core Spotlight

Lyn Fong, Spotlight
Previewing Your Core Spotlight Items on iOS

Content is previewed when you peek and pop on Spotlight results.

Spotlight provides a default preview.

Create a Quick Look Preview extension to customize your preview.
Previewing Your Core Spotlight Items on iOS

Content is previewed when you peek and pop on Spotlight results.

Spotlight provides a default preview.

Create a Quick Look Preview extension to customize your preview.
Previewing Your Core Spotlight Items on iOS
Default

Rachel the Rose
This rose regrets not auditioning for a role in a fairy tale-based live action film.
Previewing Your Core Spotlight Items on iOS
Quick Look Preview extension
Previewing Your Core Spotlight Items on iOS

Quick Look Preview extension
Previewing Your Core Spotlight Items on iOS

Choose a template for your new target:

- iOS
- watchOS
- tvOS
- macOS
- Cross-platform

Application Extension

Quick Look
Preview Extension

Cancel  Previous  Next
Previewing Your Core Spotlight Items on iOS

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSExtension</td>
<td>Dictionary</td>
<td>(3 items)</td>
</tr>
<tr>
<td>NSExtensionAttributes</td>
<td>Dictionary</td>
<td>(2 items)</td>
</tr>
<tr>
<td>QLSupportedContentTypes</td>
<td>Array</td>
<td>(0 items)</td>
</tr>
<tr>
<td>QLSupportsSearchableItems</td>
<td>Boolean</td>
<td>YES</td>
</tr>
<tr>
<td>NSExtensionMainStoryboard</td>
<td>String</td>
<td>MainInterface</td>
</tr>
<tr>
<td>NSExtensionPointIdentifier</td>
<td>String</td>
<td>com.apple.quicklook.preview</td>
</tr>
</tbody>
</table>
// Quick Look Core Spotlight Preview API

func preparePreviewOfSearchableItem(identifier: String, queryString: String?,
        completionHandler handler: @escaping QLPreviewItemLoadingBlock) {

    // retrieve the searched for content from the identifier
    let content = findContent(identifier: identifier)

    // setup your view based on the content retrieved
    setupViewForContent(content: content)

    // make sure you call the completion handler once you're done
    handler(nil)
}
// Quick Look Core Spotlight Preview API

func preparePreviewOfSearchableItem(identifier: String, queryString: String?, completionHandler handler: @escaping QLPreviewItemLoadingBlock) {

    // retrieve the searched for content from the identifier
    let content = findContent(identifier: identifier)

    // setup your view based on the content retrieved
    setupViewForContent(content: content)

    // make sure you call the completion handler once you're done
    handler(nil)
}
func preparePreviewOfSearchableItem(identifier: String, queryString: String?,
completionHandler handler: @escaping QLPreviewItemLoadingBlock) {

    //retrieve the searched for content from the identifier
    let content = findContent(identifier: identifier)

    //setup your view based on the content retrieved
    setupViewForContent(content: content)

    //make sure you call the completion handler once you're done
    handler(nil)
}
func preparePreviewOfSearchableItem(identifier: String, queryString: String?, completionHandler handler: @escaping QLPreviewItemLoadingBlock) {

    // retrieve the searched for content from the identifier
    let content = findContent(identifier: identifier)

    // setup your view based on the content retrieved
    setupViewForContent(content: content)

    // make sure you call the completion handler once you're done
    handler(nil)
}
Previewing Your Core Spotlight Items on iOS
Debugging

Not your typical extension workflow

Pick any host app

Launch from Spotlight

Xcode will attach when the extension is launched in Spotlight
Previewing Your Core Spotlight Items on iOS

Debugging

Not your typical extension workflow

Pick any host app

Launch from Spotlight

Xcode will attach when the extension is launched in Spotlight
Previewing Your Core Spotlight Items on iOS

Debugging

Not your typical extension workflow

Pick any host app

Launch from Spotlight

Xcode will attach when the extension is launched in Spotlight
Demo
Core Spotlight Previews on iOS
Previewing Your Core Spotlight Items on iOS

Final tips

Be fast!

Call the completion handler as soon as possible to avoid

Be memory efficient in an extension

No background work after calling the completion handler
Previewing Your Core Spotlight Items on MacOS

Content is previewed when you select a result in Spotlight.

Spotlight provides no preview.

Create a Quick Look Preview extension to provide a preview.
Previewing Your Core Spotlight Items on MacOS

Default

Search bar with text: rachel the rose

Images section:
- Rachel the Rose
- Show all in Finder
Previewing Your Core Spotlight Items on MacOS

Quick Look Preview extension
Previewing Your Core Spotlight Items on MacOS
Quick Look Preview extension
Previewing Your Core Spotlight Items on MacOS

Debugging

Not your typical extension workflow

Spotlight vanishes if Xcode has focus

Use the Quick Look Simulator instead
Demo
Core Spotlight Previews on MacOS
Previewing Your Core Spotlight Items on MacOS
More information and tips

Be fast and memory efficient!

No first responder in the extension

Preview is not meant to be interactive

Supports only Core Spotlight items
Ranking

Machine-learning-based ranker
Ranking

Machine-learning-based ranker

Personalized and adaptive
Ranking

Machine-learning-based ranker

Personalized and adaptive

Runs on device
Ranking

Machine-learning-based ranker

Personalized and adaptive

Runs on device

Private
Ranking

Machine-learning-based ranker

Personalized and adaptive

Runs on device

Private
New attributes to let you inform our ranking
New attributes to let you inform our ranking

// (1-100 , 100 being better)
open var rankingHint: NSNumber?
New attributes to let you inform our ranking

// (1-100, 100 being better)
open var rankingHint: NSNumber?

// Boolean attribute, set to true if the user created the item
open var userCreated: NSNumber?
New attributes to let you inform our ranking

// (1-100, 100 being better)
open var rankingHint: NSNumber?

// Boolean attribute, set to true if the user created the item
open var userCreated: NSNumber?

// Boolean attribute, set to true if the user purchased the item
open var userOwned: NSNumber?
New attributes to let you inform our ranking

// (1-100, 100 being better)
open var rankingHint: NSNumber?

// Boolean attribute, set to true if the user created the item
open var userCreated: NSNumber?

// Boolean attribute, set to true if the user purchased the item
open var userOwned: NSNumber?

// Boolean attribute, set to true if the user selected/favorited/collected the item
open var userCurated: NSNumber?
Ranking Tips and Tricks
Match quality and usage information is critical for ranking
Ranking Tips and Tricks

Match quality and usage information is critical for ranking.

Use NSUserActivity to provide usage information from your app.
Ranking Tips and Tricks

Match quality and usage information is critical for ranking

Use NSUserActivity to provide usage information from your app

Provide rich metadata for ranking

• Title
• Description
• Dates
• Keywords
CoreSpotlight Refresher
CoreSpotlight Refresher

Indexing CSSearchableItem
CoreSpotlight Refresher

Indexing CSSearchableItem

Indexing NSUserActivity
CoreSpotlight Refresher

Indexing CSSearchableItem

Indexing NSUserActivity

Deleting indexed items
CoreSpotlight Refresher
Indexing CSSearchableItem
CoreSpotlight Refresher
Indexing CSSearchableItem

Items for all that your app has to offer
CoreSpotlight Refresher
Indexing CSSearchableItem

Items for all that your app has to offer

```swift
let attributes = CSSearchableItemAttributeSet(itemContentType: kUTTypeImage as String)
attributes.displayName = name

let item = CSSearchableItem(uniqueIdentifier: identifier,
                             domainIdentifier: "mydomain", attributeSet: attributes)

let index = CSSearchableIndex.default()
index.indexSearchableItems(items, completionHandler: handler)
```
Items for all that your app has to offer

```swift
let attributes = CSSearchableItemAttributeSet(itemContentType: kUTTypeImage as String)
attributes.displayName = name

let item = CSSearchableItem(uniqueIdentifier: identifier,
domainIdentifier: "mydomain", attributeSet: attributes)

let index = CSSearchableIndex.default()
index.indexSearchableItems(items, completionHandler: handler)
```
CoreSpotlight Refresher
Indexing CSSearchableItem

Items for all that your app has to offer

```swift
let attributes = CSSearchableItemAttributeSet(itemContentType: kUTTypeImage as String)
attributes.displayName = name

let item = CSSearchableItem(uniqueIdentifier: identifier,
                            domainIdentifier: "mydomain", attributeSet: attributes)

let index = CSSearchableIndex.default()
index.indexSearchableItems(items, completionHandler: handler)
```
CoreSpotlight Refresher
Indexing CSSearchableItem

Items for all that your app has to offer

```swift
let attributes = CSSearchableItemAttributeSet(itemContentType: kUTTypeImage as String)
attributes.displayName = name
let item = CSSearchableItem(uniqueIdentifier: identifier,
                             domainIdentifier: "mydomain", attributeSet: attributes)
let index = CSSearchableIndex.default()
index.indexSearchableItems(items, completionHandler: handler)
```
CoreSpotlight Refresher
Indexing NSUserActivity
NSUserActivity can be used to index content and navigation points in your app.
CoreSpotlight Refresher
Indexing NSUserActivity

NSUserActivity can be used to index content and navigation points in your app
• NSUserActivity reflects what the user did
CoreSpotlight Refresher

Indexing NSUserActivity

NSUserActivity can be used to index content and navigation points in your app

- NSUserActivity reflects what the user did
- CSSearchableItem reflects what your app has
CoreSpotlight Refresher
Indexing NSUserActivity

NSUserActivity can be used to index content and navigation points in your app
• NSUserActivity reflects what the user did
• CSSearchableItem reflects what your app has

Relate NSUserActivities to CSSearchableItems to help ranking
CoreSpotlight Refresher
Indexing NSUserActivity

NSUserActivity can be used to index content and navigation points in your app
• NSUserActivity reflects what the user did
• CSSearchableItem reflects what your app has

Relate NSUserActivities to CSSearchableItems to help ranking

```swift
let attributes = CSSearchableItemAttributeSet(itemContentType: kUTTypeImage)
attributes.displayName = "Private content!"
attributes.relatedUniqueIdentifier = "myIdentifier"

let userActivity = NSUserActivity(activityType: "myActivityType");
userActivity.eligibleForSearch = true
userActivity.contentAttributeSet = attributes
```
CoreSpotlight Refresher

Indexing NSUserActivity

NSUserActivity can be used to index content and navigation points in your app
- NSUserActivity reflects what the user did
- CSSearchableItem reflects what your app has

Relate NSUserActivities to CSSearchableItems to help ranking

```swift
let attributes = CSSearchableItemAttributeSet(itemContentType: kUTTypeImage)
attributes.displayName = "Private content!"
attributes.relatedUniqueIdentifier = "myIdentifier"

let userActivity = NSUserActivity(activityType: "myActivityType")
userActivity.eligibleForSearch = true
userActivity.contentAttributeSet = attributes
```
NSUserActivity can be used to index content and navigation points in your app

- NSUserActivity reflects what the user did
- CSSearchableItem reflects what your app has

Relate NSUserActivities to CSSearchableItems to help ranking

```swift
let attributes = CSSearchableItemAttributeSet(itemContentType: kUTTypeImage)
attributes.displayName = "Private content!"
attributes.relatedUniqueIdentifier = "myIdentifier"

let userActivity = NSUserActivity(activityType: "myActivityType");
userActivity.eligibleForSearch = true
userActivity.contentAttributeSet = attributes
```
CoreSpotlight Refresher
Indexing NSUserActivity

NSUserActivity can be used to index content and navigation points in your app
• NSUserActivity reflects what the user did
• CSSearchableItem reflects what your app has

Relate NSUserActivities to CSSearchableItems to help ranking

```swift
let attributes = CSSearchableItemAttributeSet(itemContentType: kUTTypeImage)
attributes.displayName = "Private content!"
attributes.relatedUniqueIdentifier = "myIdentifier"

let userActivity = NSUserActivity(activityType: "myActivityType")
userActivity.eligibleForSearch = true
userActivity.contentAttributeSet = attributes
```
CoreSpotlight Refresher
Indexing NSUserActivity

NSUserActivity can be used to index content and navigation points in your app
- NSUserActivity reflects what the user did
- CSSearchableItem reflects what your app has

Relate NSUserActivities to CSSearchableItems to help ranking

```swift
let attributes = CSSearchableItemAttributeSet(itemContentType: kUTTypeImage)
attributes.displayName = "Private content!"
attributes.relatedUniqueIdentifier = "myIdentifier"

let userActivity = NSUserActivity(activityType: "myActivityType");
userActivity.eligibleForSearch = true
userActivity.contentAttributeSet = attributes
```
CoreSpotlight Refresher

Deleting items
CoreSpotlight Refresher
Deleting items

Clear items deleted by the user
CoreSpotlight Refresher
Deleting items

Clear items deleted by the user
Dispose of stale content
CoreSpotlight Refresher
Deleting items

Clear items deleted by the user
Dispose of stale content

```swift
let index = CSSearchableIndex.default()

index.deleteSearchableItems(withIdentifiers: ["hello"], completionHandler: handler)

index.deleteSearchableItems(withDomainIdentifiers: ["Greetings"], completionHandler: handler)

index.deleteAllSearchableItems(completionHandler: handler)
```
CoreSpotlight Refresher
Deleting items

Clear items deleted by the user
Dispose of stale content

```swift
let index = CSSearchableIndex.default()

index.deleteSearchableItems(withIdentifiers: ["hello"], completionHandler: handler)

index.deleteSearchableItems(withDomainIdentifiers: ["Greetings"], completionHandler: handler)

index.deleteAllSearchableItems(completionHandler: handler)
```
CoreSpotlight Refresher
Deleting items

Clear items deleted by the user

Dispose of stale content

```swift
let index = CSSearchableIndex.default()

index.deleteSearchableItems(withIdentifiers: ["hello"], completionHandler: handler)

index.deleteSearchableItems(withDomainIdentifiers: ["Greetings"], completionHandler: handler)

index.deleteAllSearchableItems(completionHandler: handler)
```
CoreSpotlight Indexing
Getting it right

Registering as an index delegate
Creating a CoreSpotlight extension
Use client state
Performance considerations
Index Delegate

Responsibilities
Index Delegate
Responsibilities

Full reindexing
Index Delegate
Responsibilities

Full reindexing
Selective reindexing
Index Delegate

Responsibilities

- Full reindexing
- Selective reindexing
- Reacting to index throttling
Index Delegate
Responsibilities

Full reindexing
Selective reindexing
Reacting to index throttling
Drag and drop
Index Delegate

Responsibilities

Full reindexing

Selective reindexing

Reacting to index throttling

Drag and drop

//Register as the index delegate
CSSearchableIndex.default().indexDelegate = self
public protocol CSSearchableIndexDelegate : NSObjectProtocol {

// Indexing
    public func searchableIndex(_ searchableIndex: CSSearchableIndex, reindexAllSearchableItemsWithAcknowledgementHandler acknowledgementHandler: @escaping () -> Swift.Void)
    public func searchableIndex(_ searchableIndex: CSSearchableIndex, reindexSearchableItemsWithIdentifiers identifiers: [String], acknowledgementHandler: @escaping () -> Swift.Void)

    optional public func searchableIndexDidThrottle(_ searchableIndex: CSSearchableIndex)
    optional public func searchableIndexDidFinishThrottle(_ searchableIndex: CSSearchableIndex)

// Drag and drop
    optional public func data(for searchableIndex: CSSearchableIndex, itemIdentifier: String, typeIdentifier: String) throws -> Data
    optional public func fileURL(for searchableIndex: CSSearchableIndex, itemIdentifier: String, typeIdentifier: String, inPlace: Bool) throws -> URL
}
public protocol CSSearchableIndexDelegate : NSObjectProtocol {

    // Indexing
    public func searchableIndex(_ searchableIndex: CSSearchableIndex,
    reindexAllSearchableItemsWithAcknowledgementHandler acknowledgementHandler: @escaping () -> Swift.Void)
    public func searchableIndex(_ searchableIndex: CSSearchableIndex,
    reindexSearchableItemsWithIdentifiers identifiers: [String], acknowledgementHandler: @escaping () -> Swift.Void)

    optional public func searchableIndexDidThrottle(_ searchableIndex: CSSearchableIndex)
    optional public func searchableIndexDidFinishThrottle(_ searchableIndex: CSSearchableIndex)

    // Drag and drop
    optional public func data(for searchableIndex: CSSearchableIndex, itemIdentifier: String, typeIdentifier: String) throws -> Data
    optional public func fileURL(for searchableIndex: CSSearchableIndex, itemIdentifier: String, typeIdentifier: String, inPlace: Bool) throws -> URL
}
public protocol CSSearchableIndexDelegate : NSObjectProtocol {

    // Indexing
    public func searchableIndex(_ searchableIndex: CSSearchableIndex, reindexAllSearchableItemsWithAcknowledgementHandler acknowledgementHandler: @escaping () -> Swift.Void)
    public func searchableIndex(_ searchableIndex: CSSearchableIndex, reindexSearchableItemsWithIdentifiers identifiers: [String], acknowledgementHandler: @escaping () -> Swift.Void)

    optional public func searchableIndexDidThrottle(_ searchableIndex: CSSearchableIndex)
    optional public func searchableIndexDidFinishThrottle(_ searchableIndex: CSSearchableIndex)

    // Drag and drop
    optional public func data(for searchableIndex: CSSearchableIndex, itemIdentifier: String, typeIdentifier: String) throws -> Data
    optional public func fileURL(for searchableIndex: CSSearchableIndex, itemIdentifier: String, typeIdentifier: String, inPlace: Bool) throws -> URL
}

public protocol CSSearchableIndexDelegate : NSObjectProtocol {

    // Indexing
    public func searchableIndex(_ searchableIndex: CSSearchableIndex,
    reindexAllSearchableItemsWithAcknowledgementHandler acknowledgementHandler: @escaping () -> Swift.Void)
    public func searchableIndex(_ searchableIndex: CSSearchableIndex,
    reindexSearchableItemsWithIdentifiers identifiers: [String], acknowledgementHandler: @escaping () -> Swift.Void)

    optional public func searchableIndexDidThrottle(_ searchableIndex: CSSearchableIndex)
    optional public func searchableIndexDidFinishThrottle(_ searchableIndex: CSSearchableIndex)

    // Drag and drop
    optional public func data(for searchableIndex: CSSearchableIndex, itemIdentifier: String,
    typeIdentifier: String) throws -> Data
    optional public func fileURL(for searchableIndex: CSSearchableIndex, itemIdentifier: String,
    typeIdentifier: String, inPlace: Bool) throws -> URL

}
// Called when everything needs to be indexed

    func searchableIndex(_: CSSearchableIndex,
    reindexAllSearchableItemsWithAcknowledgementHandler acknowledgementHandler: @escaping () -> Void) {
        let group = DispatchGroup()
        //get all items, index asynchronously
        //...
        //call the acknowledgement handle when indexing has completed
        group.notify(queue: dataStore.queue) {
            acknowledgementHandler()
        }
    }
// Called when select items needs to be indexed

func searchableIndex(_: CSSearchableIndex, reindexSearchableItemsWithIdentifiers
    identifiers: [String], acknowledgementHandler: @escaping () -> Void)
{
    let group = DispatchGroup()
    // look up requested items, and index them asynchronously
    // ...
    // call the acknowledgement handle when indexing has completed
    group.notify(queue:dataStore.queue) {
        acknowledgementHandler()
    }
}

CoreSpotlight Extension

Catching up in the background
CoreSpotlight Extension
Catching up in the background

Provide a CoreSpotlight extension
CoreSpotlight Extension
Catching up in the background

Provide a CoreSpotlight extension
• The extension can index when your app isn’t running
• Same interface as the index delegate
CoreSpotlight Extension
Catching up in the background

Provide a CoreSpotlight extension
• The extension can index when your app isn’t running
• Same interface as the index delegate

```swift
//reindex all searchable items for you app
public func searchableIndex(_: searchableIndex: CSSearchableIndex,
reindexAllSearchableItemsWithAcknowledgementHandler
    acknowledgementHandler: @escaping () -> Swift.Void)

//reindex select items for you app
public func searchableIndex(_: searchableIndex: CSSearchableIndex,
reindexSearchableItemsWithIdentifiers identifiers: [String],
    acknowledgementHandler: @escaping () -> Swift.Void)
```
Using Client State
Using Client State

Makes it easy to keep your data store and Spotlight in sync
Using Client State

Makes it easy to keep your data store and Spotlight in sync

An opaque token stored in Spotlight’s index

• You own it
• You decide what it means
Using Client State

Makes it easy to keep your data store and Spotlight in sync

An opaque token stored in Spotlight’s index

• You own it
• You decide what it means

Often a sequence number
Using Client State

Makes it easy to keep your data store and Spotlight in sync

An opaque token stored in Spotlight’s index
- You own it
- You decide what it means

Often a sequence number

Great with journals or database annotations
Batching and Client State

Indexing

Your App

Indexing

CoreSpotlight Journal

Client State 2  Item 4  Item 3  Client State 1  Item 2  Item 1
Batching and Client State

Disaster
Batching and Client State

Recovery

Your App Recovering

Client State 2

CoreSpotlight Journal

Client State 2  Item 4  Item 3  Client State 1  Item 2  Item 1
Batching and Client State

Recovery

Your App

Indexing

Client State 3

Item 7 Item 6 Item 5

CoreSpotlight Journal

Client State 2 Item 4 Item 3 Client State 1 Item 2 Item 1
Setting Client State
Setting Client State

Index state uses a named index instance
Index state uses a named index instance

- Create multiple instances if you have more than one data source
Setting Client State

Index state uses a named index instance
• Create multiple instances if you have more than one data source

```swift
let index = CSSearchableIndex(name: "myname")
index.beginBatch()
index.indexSearchableItems(items, completionHandler: nil);
let stateString = String(offset + items.count)
index.endBatch(withClientState: stateString.data(using:NSUTF8StringEncoding)!, completionHandler: handler)
```
Index state uses a named index instance

- Create multiple instances if you have more than one data source

```swift
let index = CSSearchableIndex(name: "myname")

index.beginBatch()

index.indexSearchableItems(items, completionHandler: nil);

let stateString = String(offset + items.count)

index.endBatch(withClientState: stateString.data(using:NSUTF8StringEncoding)!, completionHandler: handler)
```
**Setting Client State**

Index state uses a named index instance

- Create multiple instances if you have more than one data source

```swift
let index = CSSearchableIndex(name: "myname")
index.beginBatch()
index.indexSearchableItems(items, completionHandler: nil);
let stateString = String(offset + items.count)
index.endBatch(withClientState: stateString.data(using:NSUTF8StringEncoding)!, completionHandler: handler)
```
Index state uses a named index instance

- Create multiple instances if you have more than one data source

```swift
let index = CSSearchableIndex(name: "myname")

index.beginBatch()

index.indexSearchableItems(items, completionHandler: nil);

let stateString = String(offset + items.count)

index.endBatch(withClientState: stateString.data(using:NSUTF8StringEncoding)!, completionHandler: handler)
```
Index state uses a named index instance

- Create multiple instances if you have more than one data source

```swift
let index = CSSearchableIndex(name: "myname")

index.beginBatch()

index.indexSearchableItems(items, completionHandler: nil);

let stateString = String(offset + items.count)

index.endBatch(withClientState: stateString.data(using:NSUTF8StringEncoding)!, completionHandler: handler)
```
Setting Client State

Index state uses a named index instance
• Create multiple instances if you have more than one data source

```swift
let index = CSSearchableIndex(name: "myname")

index.beginBatch()

index.indexSearchableItems(items, completionHandler: nil);

let stateString = String(offset + items.count)

index.endBatch(withClientState: stateString.data(using:NSUTF8StringEncoding)!,
               completionHandler: handler)
```
Checking Client State
Checking Client State

Use client state to resume interrupted indexing
Checking Client State

Use client state to resume interrupted indexing

Fetch and check client state when starting
Use client state to resume interrupted indexing

Fetch and check client state when starting

```swift
let index = CSSearchableIndex(name: "myname")

index.fetchLastClientState(completionHandler: { (data, error) in
    if error != nil {
        // deal with the error!
    } else if (data != expectedData) {
        doIndex(index: index, data: data)
    }
})
```
Checking Client State

Use client state to resume interrupted indexing

Fetch and check client state when starting

```swift
let index = CSSearchableIndex(name: "myname")

index.fetchLastClientState(completionHandler: { (data, error) in
    if error != nil {
        // deal with the error!
    } else if (data != expectedData) {
        doIndex(index:index, data:data)
    }
})
```
Use client state to resume interrupted indexing

Fetch and check client state when starting

```swift
let index = CSSearchableIndex(name: "myname")

index.fetchLastClientState(completionHandler: { (data, error) in
    if error != nil {
        // deal with the error!
    } else if (data != expectedData) {
        doIndex(index: index, data: data)
    }
})
```
Checking Client State

Use client state to resume interrupted indexing

Fetch and check client state when starting

```swift
let index = CSSearchableIndex(name: "myname")

index.fetchLastClientState(completionHandler: { (data, error) in
    if error != nil {
        // deal with the error!
    } else if (data != expectedData) {
        doIndex(index:index, data: data)
    }
})
```
Use client state to resume interrupted indexing

Fetch and check client state when starting

```swift
let index = CSSearchableIndex(name: "myname")

index.fetchLastClientState(completionHandler: { (data, error) in
    if error != nil {
        // deal with the error!
    } else if (data != expectedData) {
        doIndex(index: index, data: data)
    }
})
```
Checking Client State

Use client state to resume interrupted indexing

Fetch and check client state when starting

```swift
let index = CSSearchableIndex(name: "myname")

index.fetchLastClientState(completionHandler: { (data, error) in
    if error != nil {
        // deal with the error!
    } else if (data != expectedData) {
        doIndex(index:index, data:data)
    }
})
```
Indexing and Performance
Indexing and Performance

Indexing is background work
Indexing and Performance

Indexing is background work

Minimize overhead
Indexing and Performance

Indexing is background work

Minimize overhead

Optimize storage and database access
Indexing and Performance

Indexing is background work

Minimize overhead

Optimize storage and database access

Use batching
Indexing and Performance

Indexing is background work

Minimize overhead

Optimize storage and database access

Use batching
  • Size batches for available memory
Indexing and Performance

Indexing is background work

Minimize overhead

Optimize storage and database access

Use batching

• Size batches for available memory

Don’t block the main thread
Indexing and Performance

Indexing is background work

Minimize overhead

Optimize storage and database access

Use batching

• Size batches for available memory

Don’t block the main thread

Index on a background queue
The Square Cousins
These chairs appreciate their tables predictable shape.
7/24/13

Bob the Bench
This bench waits patiently for his clients.

Sammy the Shrub
A small plant dreams of growing big someday.
Cupertino

The Square Family
These chairs wish their table wasn’t such a square.

Ted the Tree Tag
A tree tag hangs happily from its tree, observing the world below.

Seth the Shrub
This frondy shrub wonders if Sammy the shrub can come out to play soon.

Rachel the Rose
This rose regrets not auditioning for a role in a fairytale based live action film.

Fred the Fire Hydrant
The Square Cousins
These chairs appreciate their tables predictable shape.
7/24/13

Bob the Bench
This bench waits patiently for his clients.
⭐⭐⭐⭐⭐ 10234 reviews

Sammy the Shrub
A small plant dreams of growing big someday.
Cupertino

The Square Family
These chairs wish their table wasn’t such a square.

Ted the Tree Tag
A tree tag hangs happily from its tree, observing the world below.

Seth the Shrub
This frondy shrub wonders if Sammy the shrub can come out to play soon.

Rachel the Rose
This rose regrets not auditioning for a role in a fairy tale based live action film.

Fred the Fire Hydrant
The Square Cousins
These chairs appreciate their tables predictable shape.
7/24/13

Bob the Bench
This bench waits patiently for his clients.
4 stars 10,234 reviews

Sammy the Shrub
A small plant dreams of growing big someday.
Cupertino

The Square Family
These chairs wish their table wasn't such a square.

Ted the Tree Tag
A tree tag hangs happily from its tree, observing the world below.

Seth the Shrub
This frondy shrub wonders if Sammy the shrub can come out to play soon.

Rachel the Rose
This rose regrets not auditioning for a role in a fairy tale based live action film.

Fred the Fire Hydrant
locationName

The Square Cousins
These chairs appreciate their tables predictable shape.
7/24/13

Bob the Bench
This bench waits patiently for his clients.
5☆☆☆☆ 10,234 reviews

Sammy the Shrub
A small plant dreams of growing big someday.
Cupertino

The Square Family
These chairs wish their table wasn't such a square.

Ted the Tree Tag
A tree tag hangs happily from its tree, observing the world below.

Seth the Shrub
This frondy shrub wonders if Sammy the shrub can come out to play soon.

Rachel the Rose
This rose regrets not auditioning for a role in a fairy tale based live action film.

Fred the Fire Hydrant
Set a descriptive title
Metadata for Display

Set a descriptive title

Set a good looking, informative thumbnail
Metadata for Display

Set a descriptive title

Set a good looking, informative thumbnail

Set the right content type for your content
Metadata for Display

Set a descriptive title

Set a good looking, informative thumbnail

Set the right content type for your content

Use attributes to fill out the UI

- contentDescription
- rating, ratingDescription
- completionDate, dueDate, startDate, endDate, allDay
- fileSize, pageCount
The Square Cousins
These chairs appreciate their tables predictable shape.
7/24/13

Bob the Bench
This bench waits patiently for his clients.
5⭐️ 10234 reviews

Sammy the Shrub
A small plant dreams of growing big someday.
Cupertino

The Square Family
These chairs wish their table wasn't such a square.

Ted the Tree Tag
A tree tag hangs happily from its tree, observing the world below.

Seth the Shrub
This frondy shrub wonders if Sammy the shrub can come out to play soon.

Rachel the Rose
This rose regrets not auditioning for a role in a fairy tale based live action film.

Fred the Fire Hydrant
Metadata for User Experience
Metadata for User Experience

Make it easy to get your content

• Set attributes the user can understand
• Keyword stuffing confuses the user and leads to poor ranking
Make it easy to get your content

• Set attributes the user can understand
• Keyword stuffing confuses the user and leads to poor ranking

Set contact identifiers to support contact search
Metadata for User Experience

Make it easy to get your content
• Set attributes the user can understand
• Keyword stuffing confuses the user and leads to poor ranking

Set contact identifiers to support contact search

Set metadata for drag and drop
Metadata for User Experience

Make it easy to get your content

• Set attributes the user can understand

• Keyword stuffing confuses the user and leads to poor ranking

Set contact identifiers to support contact search

Set metadata for drag and drop

Provide quick actions for navigation and calls
func application(application: NSApplication, continueUserActivity uA: NSUserActivity, restorationHandler: ([AnyObject]?) -> Void) -> Bool {
    if uA.activityType == CSSearchableItemActionType {
        if let i = uA.userInfo?[CSSearchableItemActivityIdentifier] as? String {
            // show the found item
        }
        return true
    }
    if userActivity.activityType == CSQueryContinuationActionType {
        if let searchQuery = userActivity.userInfo?[CSSearchQueryString] as? String {
            // run the search
        }
        return true
    }
    return false
}
func application(application: NSApplication, continueUserActivity uA: NSUserActivity,
    restorationHandler: ([AnyObject]?) -> Void) -> Bool {
    if uA.activityType == CSSearchableItemActionType {
        if let i = uA.userInfo?[CSSearchableItemActivityIdentifier] as? String {
            // show the found item
        }
        return true
    }
    if userActivity.activityType == CSQueryContinuationActionType {
        if let searchQuery = userActivity.userInfo?[CSSearchQueryString] as? String {
            // run the search
        }
        return true
    }
    return false
}

Restoring with NSUserActivity
func application(application: NSApplication, continueUserActivity uA: NSUserActivity, 
restorationHandler: ([AnyObject]?) -> Void) -> Bool {
    if uA.activityType == CSSearchableItemActionType {
        if let i = uA.userInfo?[CSSearchableItemActivityIdentifier] as? String {
            // show the found item
        }
        return true
    }
    if userActivity.activityType == CSQueryContinuationActionType {
        if let searchQuery = userActivity.userInfo?[CSSearchQueryString] as? String {
            // run the search
        }
        return true
    }
    return false
}
func application(application: NSApplication, continueUserActivity uA: NSUserActivity, restorationHandler: ([AnyObject]?) -> Void) -> Bool {
    if uA.activityType == CSSearchableItemActionType {
        if let i = uA.userInfo?[CSSearchableItemActivityIdentifier] as? String {
            // show the found item
        }
        return true
    }
    if userActivity.activityType == CSQueryContinuationActionType {
        if let searchQuery = userActivity.userInfo?[CSSearchQueryString] as? String {
            // run the search
        }
        return true
    }
    return false
}

Restoring with NSUserActivity
func application(application: NSApplication, continueUserActivity uA: NSUserActivity,
    restorationHandler: ([AnyObject]?) -> Void) -> Bool {
    if uA.activityType == CSSearchableItemActionType {
        if let i = uA.userInfo?[CSSearchableItemActivityIdentifier] as? String {
            // show the found item
            return true
        }
    }
    return true
}

if userActivity.activityType == CSQueryContinuationActionType {
    if let searchQuery = userActivity.userInfo?[CSSearchQueryQueryString] as? String {
        // run the search
        return true
    }
    return false
}
Searching with CoreSpotlight
Searching with CoreSpotlight

Search the data you’ve already given to Spotlight
Searching with CoreSpotlight

Search the data you’ve already given to Spotlight

Same search engine that powers Spotlight, Mail, Notes, and more
Searching with CoreSpotlight

Search the data you’ve already given to Spotlight

Same search engine that powers Spotlight, Mail, Notes, and more

Consistent behavior with Spotlight and system apps
Searching with CoreSpotlight

Search the data you’ve already given to Spotlight

Same search engine that powers Spotlight, Mail, Notes, and more

Consistent behavior with Spotlight and system apps

Great for all your content on the device
Searching with CoreSpotlight

Search the data you’ve already given to Spotlight

Same search engine that powers Spotlight, Mail, Notes, and more

Consistent behavior with Spotlight and system apps

Great for all your content on the device

Available on macOS and iOS
Using the Query Language
Using the Query Language

pageCount > 10
Using the Query Language

```plaintext
pageCount > 10
InRange(pageCount, 10, 20)
```
Using the Query Language

pageCount > 10

InRange(pageCount, 10, 20)

height > 1024 && width > 1024
Using the Query Language

```plaintext
pageCount > 10

InRange(pageCount,10,20)

height > 1024 && width > 1024

authors = "Johnny Appleseed"cwd || authors = "Jane Appleseed"cwd
```
Using the Query Language

pageCount > 10

InRange(pageCount, 10, 20)

height > 1024 && width > 1024

authors = "Johnny Appleseed"cwd || authors = "Jane Appleseed"cwd

authors = "Äppelfrö"cw
Using the Query Language

```
pageCount > 10

InRange(pageCount, 10, 20)

height > 1024 && width > 1024

authors = "Johnny Appleseed"cwd || authors = "Jane Appleseed"cwd

authors = "Äppelfrö"cw

authorEmailAddresses = "john.appleseed@apple.com"
```
Using the Query Language

- pageCount > 10
- InRange(pageCount,10,20)
- height > 1024 && width > 1024
- authors = "Johnny Appleseed"cwd || authors = "Jane Appleseed"cwd
- authors = "Äppelfrö"cw
- authorEmailAddresses = "john.appleseed@apple.com"
- ** = "some text the user typed*"cdwt
Using the Query Language

```plaintext
pageCount > 10

InRange(pageCount, 10, 20)

height > 1024 && width > 1024

authors = "Johnny Appleseed" || authors = "Jane Appleseed"

authors = "Äppelfrö"

authorEmailAddresses = "john.appleseed@apple.com"

** = "some text the user typed"

textContent = "phrase match" && * = "blue"
```
<table>
<thead>
<tr>
<th>Feature</th>
<th>Token</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equality</td>
<td>==</td>
<td>keywords=&quot;search&quot;</td>
</tr>
<tr>
<td>Not Equal</td>
<td>!=</td>
<td>keywords!=&quot;search&quot;</td>
</tr>
<tr>
<td>Greater than</td>
<td>&gt;, &gt;=</td>
<td>pageCount &gt; 10</td>
</tr>
<tr>
<td>Less than</td>
<td>&lt;, &lt;=</td>
<td>pageCount &lt; 10</td>
</tr>
<tr>
<td>Range search</td>
<td>InRange</td>
<td>InRange(pageCount, 5, 10)</td>
</tr>
<tr>
<td>AND</td>
<td>&amp;&amp;</td>
<td>fileSize &gt; 100 &amp;&amp; pageCount &gt; 10</td>
</tr>
<tr>
<td>OR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NOT</td>
<td>!</td>
<td>!(fileSize &gt; 100</td>
</tr>
<tr>
<td>Field wildcard</td>
<td>*</td>
<td>* = &quot;search&quot;</td>
</tr>
<tr>
<td>Field or content wildcard</td>
<td>**</td>
<td>** = &quot;search&quot;</td>
</tr>
<tr>
<td>Feature</td>
<td>Syntax</td>
<td>Performance</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Exact match</td>
<td>&quot;search&quot;</td>
<td>Fastest</td>
</tr>
<tr>
<td>Partial</td>
<td>&quot;search*&quot;</td>
<td>Fast</td>
</tr>
<tr>
<td>Prefix match</td>
<td>&quot;search**&quot;</td>
<td>Fast</td>
</tr>
<tr>
<td>Phrase</td>
<td>&quot;johnny appleseed&quot;</td>
<td>Slower</td>
</tr>
<tr>
<td>Suffix</td>
<td>&quot;*rch&quot;</td>
<td>Slow</td>
</tr>
<tr>
<td>Infix</td>
<td>&quot;<em>arc</em>&quot;</td>
<td>Slow</td>
</tr>
<tr>
<td>Infix phrase</td>
<td>&quot;<em>johnny</em> <em>appleseed</em>&quot;</td>
<td>Slowest</td>
</tr>
</tbody>
</table>
## String Matching

<table>
<thead>
<tr>
<th>Feature</th>
<th>Flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case insensitive</td>
<td>‘c’</td>
</tr>
<tr>
<td>Diacritics insensitive</td>
<td>‘d’</td>
</tr>
<tr>
<td>(ö = o, å = a, …)</td>
<td></td>
</tr>
<tr>
<td>Word matching</td>
<td>‘w’</td>
</tr>
<tr>
<td>(Inc = “Apple Inc”, String = NSString)</td>
<td></td>
</tr>
<tr>
<td>Tokenized</td>
<td>‘t’</td>
</tr>
<tr>
<td>(Apple Inc = Inc, … Apple)</td>
<td></td>
</tr>
</tbody>
</table>
func search(userQuery :String) {
    query.cancel();
    let escapedString = escapedUserQuery(userQuery)
    let queryString = "**="" + escapedString + "\"cwdt"
    let newQuery = CSSearchQuery(queryString: queryString, attributes: ["displayName"])

    newQuery.foundItemsHandler = {
        (items : [CSSearchableItem]) -> Void in
        /* process received items */
    }

    newQuery.completionHandler = { (err) -> Void in
        /* finish processing */
        updateDisplay()
    }

    newQuery.start()
    query=newQuery
}
Searching with CoreSpotlight

```swift
func search(userQuery :String) {
    query.cancel();
    let escapedString = escapedUserQuery(userQuery)
    let queryString = "**="" + escapedString + ""cwdt"
    let newQuery = CSSearchQuery(queryString: queryString, attributes: ["displayName"])

    newQuery.foundItemsHandler = {
        (items : [CSSearchableItem]) -> Void in
        /* process received items */
    }

    newQuery.completionHandler = { (err) -> Void in
        /* finish processing */
        updateDisplay()
    }

    newQuery.start()
    query=newQuery
}
```
func search(userQuery :String) {
    query.cancel();
    let escapedString = escapedUserQuery(userQuery)
    let queryString = "**=" + escapedString + "\cwdt"
    let newQuery = CSSearchQuery(queryString: queryString, attributes: ["displayName"])

    newQuery.foundItemsHandler = { (items : [CSSearchableItem]) -> Void in
        /* process received items */
    }

    newQuery.completionHandler = { (err) -> Void in
        /* finish processing */
        updateDisplay()
    }

    newQuery.start()
    query=newQuery
}
Searching with CoreSpotlight

```swift
func search(userQuery :String) {  
    query.cancel();
    let escapedString = escapedUserQuery(userQuery)
    let queryString = "**="" + escapedString + ""cwdt"
    let newQuery = CSSearchQuery(queryString: queryString, attributes: ["displayName"])

    newQuery.foundItemsHandler = {  
        (items : [CSSearchableItem]) -> Void in
        /* process received items */
    }

    newQuery.completionHandler = {  (err) -> Void in
        /* finish processing */
        updateDisplay()
    }

    newQuery.start()
    query=newQuery
}
```
func search(userQuery : String) {
    query.cancel();
    let escapedString = escapedUserQuery(userQuery)
    let queryString = "**="" + escapedString + "\"cwdt"
    let newQuery = CSSearchQuery(queryString: queryString, attributes: ["displayName"])

    newQuery.foundItemsHandler = {
        (items : [CSSearchableItem]) -> Void in
        /* process received items */
    }

    newQuery.completionHandler = {
        (err) -> Void in
        /* finish processing */
        updateDisplay()
    }

    newQuery.start()
    query = newQuery
}
func search(userQuery: String) {
    query.cancel();
    let escapedString = escapedUserQuery(userQuery)
    let queryString = "**=" + escapedString + "\"cwdt"
    let newQuery = CSSearchQuery(queryString: queryString, attributes: ["displayName"])

    newQuery.foundItemsHandler = {
        (items : [CSSearchableItem]) -> Void in
        /* process received items */
    }

    newQuery.completionHandler = { (err) -> Void in
        /* finish processing */
        updateDisplay()
    }

    newQuery.start()
    query = newQuery
}
func search(userQuery : String) {
    query.cancel();
    let escapedString = escapedUserQuery(userQuery)
    let queryString = "**="" + escapedString + "\"cwdt"
    let newQuery = CSSearchQuery(queryString: queryString, attributes: ["displayName"])

    newQuery.foundItemsHandler = {
        (items : [CSSearchableItem]) -> Void in
        /* process received items */
    }

    newQuery.completionHandler = {
        (err) -> Void in
        /* finish processing */
        updateDisplay()
    }

    newQuery.start()
    query=newQuery
}

func search(userQuery :String) {
    query.cancel();
    let escapedString = escapedUserQuery(userQuery)
    let queryString = "**="" + escapedString + "\cwdt"
    let newQuery = CSSearchQuery(queryString: queryString, attributes: ["displayName"])

    newQuery.foundItemsHandler = { (items : [CSSearchableItem]) -> Void in
        /* process received items */
    }

    newQuery.completionHandler = { (err) -> Void in
        /* finish processing */
        updateDisplay()
    }

    newQuery.start()
    query=newQuery
}
Summary
CoreSpotlight is available on macOS
Summary

CoreSpotlight is available on macOS
Support Previews and Drag and Drop
Summary

CoreSpotlight is available on macOS

Support Previews and Drag and Drop

Provide rich metadata for search, display, and ranking
Summary

CoreSpotlight is available on macOS

Support Previews and Drag and Drop

Provide rich metadata for search, display, and ranking

Use NSUserActivity indexing to provide usage information
Summary

CoreSpotlight is available on macOS
Support Previews and Drag and Drop
Provide rich metadata for search, display, and ranking
Use NSUserActivity indexing to provide usage information
Keep the index accurate and up to date
Summary

CoreSpotlight is available on macOS

Support Previews and Drag and Drop

Provide rich metadata for search, display, and ranking

Use NSUserActivity indexing to provide usage information

Keep the index accurate and up to date
• Implement an indexing extension
Summary

CoreSpotlight is available on macOS
Support Previews and Drag and Drop
Provide rich metadata for search, display, and ranking
Use NSUserActivity indexing to provide usage information
Keep the index accurate and up to date
  • Implement an indexing extension
  • Use batching and client state for indexing
More Information

## Related Sessions

<table>
<thead>
<tr>
<th>Session</th>
<th>WWDC 2017</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introducing Drag and Drop</td>
<td></td>
</tr>
<tr>
<td>Privacy and Your Apps</td>
<td></td>
</tr>
<tr>
<td>What's New in CoreData</td>
<td></td>
</tr>
<tr>
<td>Mastering Drag and Drop</td>
<td></td>
</tr>
<tr>
<td>Building Great Document-based Apps in iOS 11</td>
<td></td>
</tr>
<tr>
<td>Lab</td>
<td>Location</td>
</tr>
<tr>
<td>-----------------------------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>Core Data Lab</td>
<td>Technology Lab H</td>
</tr>
<tr>
<td>Core Spotlight and Search Lab</td>
<td>Technology Lab H</td>
</tr>
</tbody>
</table>