Introduction to Notifications

Session 707

Kritarth Jain iOS Notifications Engineer
Julien Barlerin iOS Frameworks QA Engineer
Agenda
Agenda

Notifications Overview
Agenda

Notifications Overview

User Notifications Framework
Agenda

Notifications Overview

User Notifications Framework

- Registration
- Content
- Scheduling
- Management
- Actions
Agenda

Notifications Overview

User Notifications Framework
• Registration
• Content
• Scheduling
• Management
• Actions

Service Extensions
## Agenda

There’s a lot more…

| Advanced Notifications | Pacific Heights | Wednesday 10:00AM |
User Notifications
User Notifications
User Notifications

Visual alert
User Notifications

Visual alert

Sound and vibration
User Notifications

Visual alert
Sound and vibration
App icon badging
User Notifications
User Notifications
User Notifications
User Notifications
Types of Notifications
Types of Notifications

Local Notifications
Remote Notifications
Local Notifications
Local Notifications

Application on device
Local Notifications

Application on device

Schedule
Local Notifications

Application on device

Examples

• Task reminder alerts
• Calendar alerts
• Location-based triggers
Remote Notifications
Remote Notifications

Server-side application component
Remote Notifications

Server-side Application
Remote Notifications

Apple Push Notification Service (APNs)
Remote Notifications

Apple Push Notification Service (APNs)
Remote Notifications

Apple Push Notification Service (APNs)

Examples
- News alerts
- Instant messaging
- Sports updates
Remote Notifications

Components

- Server-side Application
- APNs
- Notification Payload
Remote Notifications

Components

User-facing
Silent update

Server-side Application

Notification Payload

APNs
Remote Notifications

User-facing

- Server-side Application
- Notification Payload
- APNs
- Mobile Application
Remote Notifications

User-facing

Displayed to the user on device

Server-side Application

Notification Payload

APNs

Application
Remote Notifications

Silent update
Remote Notifications
Silent update

Background App Refresh

Server-side Application

Notification Payload

APNs

{content-available:1}
Existing API Overview
Existing API Overview
Existing API Overview

UIApplication

• Registration
• Scheduling
Existing API Overview

UIApplication
- Registration
- Scheduling

Different callbacks for local and remote notifications
Existing API Overview

UIApplication
  • Registration
  • Scheduling
Different callbacks for local and remote notifications
Limited control after notifications are scheduled
Existing API Overview

UIApplication

• Registration
• Scheduling

Different callbacks for local and remote notifications

Limited control after notifications are scheduled

Different support across multiple platforms
User Notifications Framework
User Notifications Framework
Overview
User Notifications Framework

Overview

Familiar API with feature parity
User Notifications Framework

Overview

Familiar API with feature parity

Expanded content
User Notifications Framework

Overview

Familiar API with feature parity
Expanded content
Same code path for local and remote notification handling
User Notifications Framework

Overview

Familiar API with feature parity
Expanded content
Same code path for local and remote notification handling
Simplified delegate methods
User Notifications Framework

Overview

Familiar API with feature parity
Expanded content
Same code path for local and remote notification handling
Simplified delegate methods
Better notification management
User Notifications Framework

Overview

Familiar API with feature parity
Expanded content
Same code path for local and remote notification handling
Simplified delegate methods
Better notification management
In-app presentation option
User Notifications Framework

Overview

Familiar API with feature parity
Expanded content
Same code path for local and remote notification handling
Simplified delegate methods
Better notification management
In-app presentation option
Schedule and handle notifications in extensions
User Notifications Framework

Overview

Familiar API with feature parity
Expanded content
Same code path for local and remote notification handling
Simplified delegate methods
Better notification management
In-app presentation option
Schedule and handle notifications in extensions
Notification Extensions!
User Notifications Framework

Multi-Platform Support
User Notifications Framework

Multi-Platform Support

Single Notifications API across platforms

iOS  watchOS  tvOS
User Notifications Framework

iOS
User Notifications Framework

iOS

Full support to schedule and manage notifications
User Notifications Framework

watchOS
User Notifications Framework

watchOS

Existing support for forwarded notifications
User Notifications Framework
watchOS

Existing support for forwarded notifications
Local Notifications on the watch
User Notifications Framework

Existing support for forwarded notifications

Local Notifications on the watch

Examples

• “Workout goal met”
• “Timers on the watch”
User Notifications Framework

Existing support for forwarded notifications

Local Notifications on the watch

Examples

• “Workout goal met”

• “Timers on the watch”
User Notifications Framework
tvOS
User Notifications Framework
tvOS

Support to badge app icons
User Notifications Framework

tvOS

Support to badge app icons

• Examples
  • “3 unwatched episodes”
  • “Pending user turn in game”
Notification Delivery
Notification Delivery
Notification Delivery
Notification Delivery
Notification Delivery

Registration
Registration
Registration

User Authorization

• Banners
• Sound alerts
• Badging
Registration

User Authorization

• Banners
• Sound alerts
• Badging

Needed for local and remote notifications
Registration

User Authorization

• Banners
• Sound alerts
• Badging

Needed for local and remote notifications

```swift
UNUserNotificationCenter.current().requestAuthorization([.alert, .sound, .badge])
{ (granted, error) in // ... }
```
Registration
Notification settings
Registration

Notification settings

Configurable in Settings per app
Registration

Notification settings

Configurable in Settings per app
Access to user-defined settings

UNUserNotificationCenter.current().getNotificationSettings { (settings) in // ... }
Token Registration
Token Registration

Remote Notifications
Remote Notifications
Existing API

```swift
UIApplication.shared().registerForRemoteNotifications()
```
Token Registration

Remote Notifications
Existing API

```
UIApplication.shared().registerForRemoteNotifications()
```

Need network connection to talk to APNs
Token Registration

Remote Notifications

Existing API

```swift
UIApplication.shared().registerForRemoteNotifications()
```

Need network connection to talk to APNs
Token Registration

Remote Notifications
Existing API

```
UIApplication.shared().registerForRemoteNotifications()
```

Need network connection to talk to APNs
Token must be included in remote payload
Notification Delivery

Content

Registration
Content
Content
Content
Content

Subtitle
Title
Body

NEW
let content = UNMutableNotificationContent()
content.title = "Introduction to Notifications"
content.subtitle = "Session 707"
content.body = "Woah! These new notifications look amazing! Don't you agree?"
content.badge = 1
Remote Notification

Content

{
    "aps" : {
        "alert" : {
            "title" : "Introduction to Notifications",
            "subtitle" : "Session 707",
            "body" : "Woah! These new notifications look amazing! Don’t you agree?"
        },
        "badge" : 1
    }
}
Content
Media Attachments

NEW
## Content

### Media Attachments

More on Media Attachments

| Advanced Notifications | Pacific Heights | Wednesday 10:00AM |
Notification Delivery

Trigger
Triggers
Triggers

Push
Triggers

Push

Remote Notifications
Triggers

Time Interval
Triggers

Time Interval

Examples

• “In 2 minutes from now”

UNTimeIntervalNotificationTrigger(timeInterval: 120, repeats: false)
Triggers

Time Interval

Examples

• “In 2 minutes from now”

UNTimeIntervalNotificationTrigger(timeInterval: 120, repeats: false)

• “Repeat every hour starting now”

UNTimeIntervalNotificationTrigger(timeInterval: 3600, repeats: true)
Triggers
Calendar

Wednesday
15
Calendar
Triggers
Calendar

Examples
• “8:00am tomorrow morning”
• “Repeat every Monday at 6:00pm”

let dateComponents = DateComponents()
// Configure dateComponents
UNCalendarNotificationTrigger(dateMatching: dateComponents,
    repeats: false)
Triggers
Location

Location
Triggers
Location

Examples
• “When leaving home”
• “When arriving in proximity of grocery store”

```swift
let region = CLRegion()
// Configure region
UNLocationNotificationTrigger(region: region, repeats: false);
```
Notification Delivery

- Registration
- Content
- Trigger
Schedule

Local Notifications
Local Notifications

Schedule

UNUserNotificationCenter

add(_ request:)
Local Notifications

Schedule

UNUserNotificationCenter

add(_: request:)

Trigger
Schedule
Remote Notifications
Schedule
Remote Notifications

Server-side Application

Notification Payload

APNs
Schedule
Remote Notifications

Server-side Application

Notification Payload

APNs
Delivery Summary
Delivery Summary
Delivery Summary

Registration
Delivery Summary

- Registration
- Content
Delivery Summary
import UserNotifications

UNUserNotificationCenter.current().requestAuthorization([.alert, .sound, .badge])
    { (granted, error) in // ... }

let content = UNMutableNotificationContent()
content.title = "Introduction to Notifications"
content.body = "Let's talk about notifications!"

let trigger = UNTimeIntervalNotificationTrigger(timeInterval: 5, repeats: false)

let requestIdentifier = "sampleRequest"
let request = UNNotificationRequest(identifier: requestIdentifier,
                                      content: content,
                                      trigger: trigger)

UNUserNotificationCenter.current().add(request) { (error) in // ... }
import UserNotifications

UNUserNotificationCenter.current().requestAuthorization([.alert, .sound, .badge])
{ (granted, error) in // ... }

let content = UNMutableNotificationContent()
content.title = "Introduction to Notifications"
content.body = "Let's talk about notifications!"

let trigger = UNTimeIntervalNotificationTrigger(timeInterval: 5, repeats: false)

let requestIdentifier = "sampleRequest"
let request = UNNotificationRequest(identifier: requestIdentifier,
content: content,
trigger: trigger)

UNUserNotificationCenter.current().add(request) { (error) in // ... }
import UserNotifications

UNUserNotificationCenter.current().requestAuthorization([.alert, .sound, .badge])
    { (granted, error) in // ... }

let content = UNMutableNotificationContent()
content.title = "Introduction to Notifications"
content.body = "Let's talk about notifications!"

let trigger = UNTimeIntervalNotificationTrigger(timeInterval: 5, repeats: false)

let requestIdentifier = "sampleRequest"
let request = UNNotificationRequest(identifier: requestIdentifier,
    content: content,
    trigger: trigger)

UNUserNotificationCenter.current().add(request) { (error) in // ... }
// Notification Delivery Summary

import UserNotifications

UNUserNotificationCenter.current().requestAuthorization([.alert, .sound, .badge])
    { (granted, error) in // ... }  

let content = UNMutableNotificationContent()
content.title = "Introduction to Notifications"
content.body = "Let's talk about notifications!"

let trigger = UNTimeIntervalNotificationTrigger(timeInterval: 5, repeats: false)

let requestIdentifier = "sampleRequest"
let request = UNNotificationRequest(identifier: requestIdentifier,
                                    content: content,
                                    trigger: trigger)

UNUserNotificationCenter.current().add(request) { (error) in // ... }
// Notification Delivery Summary

import UserNotifications

UNUserNotificationCenter.current().requestAuthorization(
    [.alert, .sound, .badge])
    { (granted, error) in // ... }

let content = UNMutableNotificationContent()
content.title = "Introduction to Notifications"
content.body = "Let's talk about notifications!

let trigger = UNTimeIntervalNotificationTrigger(timeInterval: 5, repeats: false)

let requestIdentifier = "sampleRequest"
let request = UNNotificationRequest(identifier: requestIdentifier,
                                    content: content,
                                    trigger: trigger)

UNUserNotificationCenter.current().add(request) { (error) in // ... }
import UserNotifications

UNUserNotificationCenter.current().requestAuthorization([.alert, .sound, .badge])
    { (granted, error) in // ... }

let content = UNMutableNotificationContent()
content.title = "Introduction to Notifications"
content.body = "Let's talk about notifications!"

let trigger = UNTimeIntervalNotificationTrigger(timeInterval: 5, repeats: false)

let requestIdentifier = "sampleRequest"
let request = UNNotificationRequest(identifier: requestIdentifier,
                                      content: content,
                                      trigger: trigger)

UNUserNotificationCenter.current().add(request) { (error) in // ... }
Notification Handling
Notification Handling
Notification Handling

Application in foreground
Notification Handling

Application in foreground

```swift
protocol UNUserNotificationCenterDelegate : NSObjectProtocol

func userNotificationCenter(_ center: UNUserNotificationCenter,
                           willPresent notification: UNNotification,
                           withCompletionHandler completionHandler:
                                (UNNotificationPresentationOptions) -> Void)
```
Notification Handling

Application in foreground

```swift
protocol UNUserNotificationCenterDelegate : NSObjectProtocol

func userNotificationCenter(_ center: UNUserNotificationCenter,
willPresent notification: UNNotification,
withCompletionHandler completionHandler: 
(UNNotificationPresentationOptions) -> Void)
```
Notification Handling

Application in foreground

```swift
protocol UNUserNotificationCenterDelegate: NSObjectProtocol {
    func userNotificationCenter(_ center: UNUserNotificationCenter, willPresent notification: UNNotification, withCompletionHandler completionHandler: @escaping (UNNotificationPresentationOptions) -> Void)
}
```
Notification Handling

Application in foreground
In-app presentation

```swift
protocol UNUserNotificationCenterDelegate : NSObjectProtocol

func userNotificationCenter(_ center: UNUserNotificationCenter, willPresent notification: UNNotification, withCompletionHandler completionHandler: (UNNotificationPresentationOptions) -> Void)
```
Notification Handling

Application in foreground

In-app presentation
Notification Handling

Application in foreground
In-app presentation

```swift
func userNotificationCenter(_ center: UNUserNotificationCenter,
willPresent notification: UNNotification,
withCompletionHandler handlerBlock:
(UNNotificationPresentationOptions) -> Void) {

    // Roll banner and sound alert
    handlerBlock([.alert, .sound])
}
```
Notification Handling

Application in foreground
In-app presentation

```swift
func userNotificationCenter(_ center: UNUserNotificationCenter,
    willPresent notification: UNNotification,
    withCompletionHandler handlerBlock: (UNNotificationPresentationOptions) -> Void) {

    // Roll banner and sound alert
    handlerBlock([.alert, .sound])
}
```
Notification Management
Notification Management

Overview
Notification Management

Overview

Access

• Pending Notifications
• Delivered Notifications
Notification Management

Overview

Access

• Pending Notifications
• Delivered Notifications

Remove Notifications
Notification Management

Overview

Access
• Pending Notifications
• Delivered Notifications

Remove Notifications

Update and promote Notifications
Notification Management
Request Identifier
Notification Management

Request Identifier

Local Notifications

• Set on Notification Request
Notification Management

Request Identifier

Local Notifications
• Set on Notification Request

Remote Notifications
• New field on the HTTP/2 request header: `apns-collapse-id`
Notification Management
Example
Notification Management

Example

Notifications scheduled for a sports game
• Start of game
// Pending Notification Removal

let gameStartIdentifier = "game1.start.identifier"
let gameStartRequest = UNNotificationRequest(identifier: gameStartIdentifier,
                                          content: content,
                                          trigger: startTrigger)

UNUserNotificationCenter.current().add(gameStartRequest) { (error) in // ... } // Game was cancelled

UNUserNotificationCenter.current().removePendingNotificationRequests(withIdentifiers: [gameStartIdentifier])
let gameStartIdentifier = "game1.start.identifier"

let gameStartRequest = UNNotificationRequest(identifier: gameStartIdentifier,
                                            content: content,
                                            trigger: startTrigger)

UNUserNotificationCenter.current().add(gameStartRequest) { (error) in // ... }

// Game was cancelled

UNUserNotificationCenter.current().removePendingNotificationRequests(withIdentifiers: [gameStartIdentifier])
// Pending Notification Removal

let gameStartIdentifier = "game1.start.identifier"

let gameStartRequest = UNNotificationRequest(identifier: gameStartIdentifier,
                                         content: content,
                                         trigger: startTrigger)

UNUserNotificationCenter.current().add(gameStartRequest) { (error) in // ... }

// Game was cancelled

UNUserNotificationCenter.current()
    .removePendingNotificationRequests(withIdentifiers: [gameStartIdentifier])
// Pending Notification Removal

let gameStartIdentifier = "game1.start.identifier"

let gameStartRequest = UNNotificationRequest(identifier: gameStartIdentifier,
                                          content: content,
                                          trigger: startTrigger)

UNUserNotificationCenter.current().add(gameStartRequest) { (error) in // ... }

// Game was cancelled

UNUserNotificationCenter.current() .removePendingNotificationRequests(withIdentifiers: [gameStartIdentifier])
// Pending Notification Removal

let gameStartIdentifier = "game1.start.identifier"
let gameStartRequest = UNNotificationRequest(identifier: gameStartIdentifier,
                                           content: content,
                                           trigger: startTrigger)

UNUserNotificationCenter.current().add(gameStartRequest) { (error) in // ... }  

// Game was cancelled

UNUserNotificationCenter.current().removePendingNotificationRequests(withIdentifiers: [gameStartIdentifier])
// Pending Notification Update

let gameStartIdentifier = "game1.start.identifier"
let gameStartRequest = UNNotificationRequest(identifier: gameStartIdentifier, 
                                           content: content,
                                           trigger: startTrigger)

UNUserNotificationCenter.current().add(gameStartRequest) { (error) in // ... }

// Game start time was updated

let updatedGameStartRequest = UNNotificationRequest(identifier: gameStartIdentifier, 
                                                      content: content, 
                                                      trigger: newStartTrigger)

UNUserNotificationCenter.current().add(updatedGameStartRequest) { (error) in // ... }
// Pending Notification Update

let gameStartIdentifier = "game1.start.identifier"
let gameStartRequest = UNNotificationRequest(identifier: gameStartIdentifier,
                                           content: content,
                                           trigger: startTrigger)

UNUserNotificationCenter.current().add(gameStartRequest) { (error) in // ... }

// Game start time was updated

let updatedGameStartRequest = UNNotificationRequest(identifier: gameStartIdentifier,
                                                     content: content,
                                                     trigger: newStartTrigger)

UNUserNotificationCenter.current().add(updatedGameStartRequest) { (error) in // ... }
// Pending Notification Update

let gameStartIdentifier = "game1.start.identifier"

let gameStartRequest = UNNotificationRequest(identifier: gameStartIdentifier,
                                           content: content,
                                           trigger: startTrigger)

UNUserNotificationCenter.current().add(gameStartRequest) { (error) in // ... }

// Game start time was updated

let updatedGameStartRequest = UNNotificationRequest(identifier: gameStartIdentifier,
                                                      content: content,
                                                      trigger: newStartTrigger)

UNUserNotificationCenter.current().add(updatedGameStartRequest) { (error) in // ... }
let gameStartIdentifier = "game1.start.identifier"
let gameStartRequest = UNNotificationRequest(identifier: gameStartIdentifier,
                                           content: content,
                                           trigger: startTrigger)

UNUserNotificationCenter.current().add(gameStartRequest) { (error) in // ... }

// Game start time was updated

let updatedGameStartRequest = UNNotificationRequest(identifier: gameStartIdentifier,
                                                       content: content,
                                                       trigger: newStartTrigger)

UNUserNotificationCenter.current().add(updatedGameStartRequest) { (error) in // ... }
// Pending Notification Update

let gameStartIdentifier = "game1.start.identifier"

let gameStartRequest = UNNotificationRequest(identifier: gameStartIdentifier,
                                            content: content,
                                            trigger: startTrigger)

UNUserNotificationCenter.current().add(gameStartRequest) { (error) in // ... }

// Game start time was updated

let updatedGameStartRequest = UNNotificationRequest(identifier: gameStartIdentifier,
                                                    content: content,
                                                    trigger: newStartTrigger)

UNUserNotificationCenter.current().add(updatedGameStartRequest) { (error) in // ... }
Notification Management

Example

Notifications scheduled for a sports game

• Score Updates
// Delivered Notification Removal

let gameScoreIdentifier = "game1.score.identifier"

let gameScoreRequest = UNNotificationRequest(identifier: gameScoreIdentifier,
                                           content: scoreContent,
                                           trigger: trigger)

UNUserNotificationCenter.current().add(gameScoreRequest) { (error) in // ... }

// Wrong game score was published

UNUserNotificationCenter.current()
  .removeDeliveredNotifications(withIdentifiers: [gameScoreIdentifier])
// Delivered Notification Removal

let gameScoreIdentifier = "game1.score.identifier"
let gameScoreRequest = UNNotificationRequest(identifier: gameScoreIdentifier, 
content: scoreContent, 
trigger: trigger)

UNUserNotificationCenter.current().add(gameScoreRequest) { (error) in // ... }

// Wrong game score was published

UNUserNotificationCenter.current().removeDeliveredNotifications(withIdentifiers: [gameScoreIdentifier])
// Delivered Notification Removal

let gameScoreIdentifier = "game1.score.identifier"

let gameScoreRequest = UNNotificationRequest(identifier: gameScoreIdentifier,
                                          content: scoreContent,
                                          trigger: trigger)

UNUserNotificationCenter.current().add(gameScoreRequest) { (error) in // ... }

// Wrong game score was published

UNUserNotificationCenter.current().removeDeliveredNotifications(withIdentifiers: [gameScoreIdentifier])
// Delivered Notification Removal

let gameScoreIdentifier = "game1.score.identifier"
let gameScoreRequest = UNNotificationRequest(identifier: gameScoreIdentifier,
                                          content: scoreContent,
                                          trigger: trigger)

UNUserNotificationCenter.current().add(gameScoreRequest) { (error) in // ... }

// Wrong game score was published

UNUserNotificationCenter.current().removeDeliveredNotifications(withIdentifiers: [gameScoreIdentifier])
// Delivered Notification Removal

let gameScoreIdentifier = "game1.score.identifier"
let gameScoreRequest = UNNotificationRequest(identifier: gameScoreIdentifier,
                                          content: scoreContent,
                                          trigger: trigger)

UNUserNotificationCenter.current().add(gameScoreRequest) { (error) in // ... }

// Wrong game score was published

UNUserNotificationCenter.current().removeDeliveredNotifications(withIdentifiers: [gameScoreIdentifier])
// Delivered Notification Update

let gameScoreIdentifier = "game1.score.identifier"
let gameScoreRequest = UNNotificationRequest(identifier: gameScoreIdentifier,
                                           content: scoreContent,
                                           trigger: trigger)

UNUserNotificationCenter.current().add(gameScoreRequest) { (error) in // ... }

// Score game was updated

let updateGameScoreRequest = UNNotificationRequest(identifier: gameScoreIdentifier,
                                                    content: newScoreContent,
                                                    trigger: newTrigger)

UNUserNotificationCenter.current().add(updateGameScoreRequest) { (error) in // ... }
let gameScoreIdentifier = "game1.score.identifier"
let gameScoreRequest = UNNotificationRequest(identifier: gameScoreIdentifier,
                                           content: scoreContent,
                                           trigger: trigger)

UNUserNotificationCenter.current().add(gameScoreRequest) { (error) in // ... }

// Score game was updated

let updateGameScoreRequest = UNNotificationRequest(identifier: gameScoreIdentifier,
                                                      content: newScoreContent,
                                                      trigger: newTrigger)

UNUserNotificationCenter.current().add(updateGameScoreRequest) { (error) in // ... }
// Delivered Notification Update

let gameScoreIdentifier = "game1.score.identifier"
let gameScoreRequest = UNNotificationRequest(identifier: gameScoreIdentifier,
                                              content: scoreContent,
                                              trigger: trigger)

UNUserNotificationCenter.current().add(gameScoreRequest) { (error) in // ... }

// Score game was updated

let updateGameScoreRequest = UNNotificationRequest(identifier: gameScoreIdentifier,
                                                     content: newScoreContent,
                                                     trigger: newTrigger)

UNUserNotificationCenter.current().add(updateGameScoreRequest) { (error) in // ... }
// Delivered Notification Update

let gameScoreIdentifier = "game1.score.identifier"
let gameScoreRequest = UNNotificationRequest(identifier: gameScoreIdentifier,
                                             content: scoreContent,
                                             trigger: trigger)

UNUserNotificationCenter.current().add(gameScoreRequest) { (error) in // ... }

// Score game was updated

let updateGameScoreRequest = UNNotificationRequest(identifier: gameScoreIdentifier,
                                                     content: newScoreContent,
                                                     trigger: newTrigger)

UNUserNotificationCenter.current().add(updateGameScoreRequest) { (error) in // ... }
// Delivered Notification Update

let gameScoreIdentifier = "game1.score.identifier"

let gameScoreRequest = UNNotificationRequest(identifier: gameScoreIdentifier,
                                           content: scoreContent,
                                           trigger: trigger)

UNUserNotificationCenter.current().add(gameScoreRequest) { (error) in // ... }

// Score game was updated

let updateGameScoreRequest = UNNotificationRequest(identifier: gameScoreIdentifier,
                                                content: newScoreContent,
                                                trigger: newTrigger)

UNUserNotificationCenter.current().add(updateGameScoreRequest) { (error) in // ... }
Notification Management

Example

- John Appleseed
  Yay! we are winning

- Score Update
  Team A takes the lead!
  Team A: 1 Team B: 0
Notification Management

Example

Missed

- John Appleseed
  Yay! we are winning

Score Update
Team A takes the lead!
Team A: 1 Team B: 0
Notification Actions

Julien Barlerin iOS Frameworks QA Engineer
Default Action
Default Action
Default Action
Default Action

Welcome to WWDC!
Actionable Notifications
Actionable Notifications

Buttons with customizable title
Actionable Notifications

Buttons with customizable title

Text input
Actionable Notifications

Buttons with customizable title
Text input
Background or foreground
Actionable Notifications

Buttons with customizable title
Text input
Background or foreground
iOS and watchOS
Actionable Notifications

iOS
Actionable Notifications

iOS
Actionable Notifications

watchOS
Actionable Notifications

watchOS
Actionable Notifications

watchOS
Actionable Notifications

watchOS

Quick Interaction Techniques for watchOS

Presidio

Wednesday 11:00AM
Actionable Notifications
Registration
let action = UNNotificationAction(identifier: "reply", title: "Reply", options: [])

let category = UNNotificationCategory(identifier: "message", actions: [action], minimalActions: [action], intentIdentifiers: [], options: [])

UNUserNotificationCenter.current().setNotificationCategories([category])
Actionable Notifications
Registration

```swift
let action = UNNotificationAction(identifier: "reply", title: "Reply", options: [])

let category = UNNotificationCategory(identifier: "message", actions: [action],
    minimalActions: [action], intentIdentifiers: [], options: [])

UNUserNotificationCenter.current().setNotificationCategories([category])
```
let action = UNNotificationAction(identifier: "reply", title: "Reply", options: [])

let category = UNNotificationCategory(identifier: "message", actions: [action], minimalActions: [action], intentIdentifiers: [], options: [])

UNUserNotificationCenter.current().setNotificationCategories([[category]]).
let action = UNNotificationAction(identifier:"reply",title:"Reply",options:[])

let category = UNNotificationCategory(identifier: "message", actions: [action],
                                             minimalActions: [action], intentIdentifiers: [], options: [])

UNUserNotificationCenter.current().setNotificationCategories([category])

Introducing SiriKit
let action = UNNotificationAction(identifier: "reply", title: "Reply", options: [])

let category = UNNotificationCategory(identifier: "message", actions: [action], minimalActions: [action], intentIdentifiers: [], options: [])

UNUserNotificationCenter.current().setNotificationCategories([category])
Actionable Notifications

Presentation
Actionable Notifications
Presentation

Remote Notifications

```json
{
  aps: {
    alert: "Welcome to WWDC!",
    category: "message"
  }
}
```
Actionable Notifications

Presentation

Remote Notifications

```
{
  aps: {
    alert: "Welcome to WWDC!",
    category: "message"
  }
}
```

Local Notifications

```
content.categoryIdentifier = "message"
```
Dismiss Action
Dismiss Action
Dismiss Action

Welcome to WWDC!
Dismiss Action
Category option
Dismiss Action
Category option

customDismissAction: `UNNotificationCategoryOptions`
Dismiss Action

Category option

customDismissAction: `UNNotificationCategoryOptions`

```swift
let category = UNNotificationCategory(identifier: "message", actions: [action],
    minimalActions: [action], intentIdentifiers: [], options: [])
```
Dismiss Action

Category option

```swift
let category = UNNotificationCategory(identifier: "message", actions: [action],
    minimalActions: [action], intentIdentifiers: [], options: [.customDismissAction])
```
Notification Actions

Summary
Notification Actions

Summary

Default action
Notification Actions

Summary

Default action

Actionable Notifications
Notification Actions

Summary

Default action

Actionable Notifications

Dismiss action
Notification Actions
Response handling
Notification Actions

Response handling

```swift
protocol UNUserNotificationCenterDelegate : NSObjectProtocol

func userNotificationCenter(_ center: UNUserNotificationCenter,
   didReceive response: UNNotificationResponse,
   withCompletionHandler completionHandler: () -> Void)
```
Notification Actions

Response handling

```swift
protocol UNUserNotificationCenterDelegate : NSObjectProtocol

func userNotificationCenter(_ center: UNUserNotificationCenter,
    didReceive response: UNNotificationResponse,
    withCompletionHandler completionHandler: () -> Void)
```
Notification Actions

Response handling

```swift
protocol UNUserNotificationCenterDelegate : NSObjectProtocol

func userNotificationCenter(_ center: UNUserNotificationCenter,
  didReceive response: UNNotificationResponse,
  withCompletionHandler completionHandler: () -> Void)
```
Notification Actions
Response handling

protocol UNUserNotificationCenterDelegate : NSObjectProtocol

func userNotificationCenter(_ center: UNUserNotificationCenter,
didReceive response: UNNotificationResponse,
withCompletionHandler completionHandler: () -> Void)
Notification Actions
Response handling

```swift
protocol UNUserNotificationCenterDelegate : NSObjectProtocol

func userNotificationCenter(_ center: UNUserNotificationCenter,
    didReceive response: UNNotificationResponse,
    withCompletionHandler completionHandler: () -> Void)
```

Response

actionIdentifier
Notification Actions

Response handling

```swift
protocol UNUserNotificationCenterDelegate : NSObjectProtocol

func userNotificationCenter(_ center: UNUserNotificationCenter,
    didReceive response: UNNotificationResponse,
    withCompletionHandler completionHandler: () -> Void)
```

Response

- actionIdentifier
- userText
Notification Actions
Response handling

```swift
protocol UNUserNotificationCenterDelegate : NSObjectProtocol

func userNotificationCenter(_ center: UNUserNotificationCenter,
                           didReceive response: UNNotificationResponse,
                           withCompletionHandler completionHandler: () -> Void)
```

Response handling involves actions such as responding to notifications. The `UNUserNotificationCenterDelegate` protocol is used to handle responses to notifications. The `userNotificationCenter(_:didReceive:withCompletionHandler:)` function is called when a notification is received, allowing you to take specific actions based on the notification's response.

The diagram shows the interaction between the `Response` and `Notification` with the `userText` and `actionIdentifier` properties indicating the details of the notification response.
Notification Actions

Response handling

```swift
protocol UNUserNotificationCenterDelegate : NSObjectProtocol

func userNotificationCenter(_ center: UNUserNotificationCenter, 
didReceive response: UNNotificationResponse, 
withCompletionHandler completionHandler: () -> Void)
```
Notification Actions
Response handling

```swift
protocol UNUserNotificationCenterDelegate : NSObjectProtocol

func userNotificationCenter(_ center: UNUserNotificationCenter,
    didReceive response: UNNotificationResponse,
    withCompletionHandler completionHandler: () -> Void)
```
Remote Notifications
Remote Notifications

Server-side Application → APNs → iPhone
Service Extension
Service Extension

Basics

Non UI iOS Extension
Service Extension

Basics

Non UI iOS Extension

Augment or Replace the content of visible Remote Notifications
Service Extension

Basics

Non UI iOS Extension
Augment or Replace the content of visible Remote Notifications
Service Extension

Basics

Non UI iOS Extension

Augment or Replace the content of visible Remote Notifications
Service Extension
Details
Service Extension

Details

Short execution time
Service Extension

Details

Short execution time

Fallback
Service Extension
Potential uses
Service Extension

Potential uses

End-to-end encryption
Service Extension

Potential uses

End-to-end encryption
Add Attachments

Advanced Notifications

Pacific Heights

Wednesday 10:00AM
Service Extension
How to implement it
Service Extension

Choose a template for your new target:

<table>
<thead>
<tr>
<th>iOS</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>Intents Extension</td>
<td>Intents UI Extension</td>
<td>Messages Extension</td>
<td>Notification Content</td>
</tr>
<tr>
<td>Framework &amp; Library</td>
<td>Photo Editing Extension</td>
<td>Share Extension</td>
<td>Shared Links Extension</td>
<td></td>
</tr>
<tr>
<td>Test</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>watchOS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Application</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Framework &amp; Library</td>
<td>Photo Editing Extension</td>
<td>Share Extension</td>
<td>Shared Links Extension</td>
<td></td>
</tr>
<tr>
<td>Test</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tvOS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Application</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Framework &amp; Library</td>
<td>Photo Editing Extension</td>
<td>Share Extension</td>
<td>Shared Links Extension</td>
<td></td>
</tr>
<tr>
<td>Test</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OS X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Application</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Framework &amp; Library</td>
<td>Photo Editing Extension</td>
<td>Share Extension</td>
<td>Shared Links Extension</td>
<td></td>
</tr>
<tr>
<td>Test</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Notification Service Extension**

This template builds a Notification Service application extension.
// Service Extension

import UserNotifications

class NotificationService: UNNotificationServiceExtension {

    override func didReceive(request: UNNotificationRequest, withContentHandler contentHandler:(UNNotificationContent) -> Void) {
        // Modify the notification content
    }

    override func serviceExtensionTimeWillExpire() {
        // Called before the extension will be terminated by the system
    }
}

}
// Service Extension

import UserNotifications

class NotificationService: UNNotificationServiceExtension {

    override func didReceive(request: UNNotificationRequest, withContentHandler contentHandler:(UNNotificationContent) -> Void) {
        // Modify the notification content
    }

    override func serviceExtensionTimeWillExpire() {
        // Called before the extension will be terminated by the system
    }

}
// Service Extension

import UserNotifications

class NotificationService: UNNotificationServiceExtension {

    override func didReceive(request: UNNotificationRequest, withContentHandler contentHandler:(UNNotificationContent) -> Void) {
        // Modify the notification content
    }

    override func serviceExtensionTimeWillExpire() {
        // Called before the extension will be terminated by the system
    }
}

// Service Extension

import UserNotifications

class NotificationService: UNNotificationServiceExtension {

    override func didReceive(request: UNNotificationRequest, withContentHandler contentHandler:(UNNotificationContent) -> Void) {
        // Modify the notification content
    }

    override func serviceExtensionTimeWillExpire() {
        // Called before the extension will be terminated by the system
    }
}

Service Extension

Example payload

{
    aps: {
        alert: "New Message Available",
        mutable-content: 1
    },
    encrypted-content: "#myencryptedcontent"
}
Service Extension

Example payload

```json
{
    aps: {
        alert: "New Message Available",
        mutable-content: 1,
    },
    encrypted-content: "#myencryptedcontent"
}
```
Service Extension

Example payload

```json
{
    aps: {
        alert: "New Message Available",
        mutable-content: 1
    },
    encrypted-content: "#myencryptedcontent"
}
```
// Decrypt Remote Notification Payload in Service Extension and Update Notification Content

override func didReceive(request: UNNotificationRequest, withContentHandler contentHandler: (UNNotificationContent) -> Void) {

    // Decrypt the payload
    let decryptedBody = decrypt(request.content.userInfo["encrypted-content"]) {

    let newContent = UNMutableNotificationContent()

    // Modify the notification content
    newContent.body = decryptedBody

    // Call content handler with updated content
    contentHandler(newContent)

    }

}
override func didReceive(request: UNNotificationRequest, withContentHandler contentHandler: (UNNotificationContent) -> Void) {

    // Decrypt the payload
    let decryptedBody = decrypt(request.content.userInfo["encrypted-content"])

    let newContent = UNMutableNotificationContent()

    // Modify the notification content
    newContent.body = decryptedBody

    // Call content handler with updated content
    contentHandler(newContent)
}
// Decrypt Remote Notification Payload in Service Extension and Update Notification Content

override func didReceive(request: UNNotificationRequest, withContentHandler contentHandler: (UNNotificationContent) -> Void) {

    // Decrypt the payload
    let decryptedBody = decrypt(request.content.userInfo["encrypted-content"])

    let newContent = UNMutableNotificationContent()

    // Modify the notification content
    newContent.body = decryptedBody

    // Call content handler with updated content
    contentHandler(newContent)
}
// Decrypt Remote Notification Payload in Service Extension and Update Notification Content

override func didReceive(request: UNNotificationRequest, withContentHandler contentHandler: (UNNotificationContent) -> Void) {

    // Decrypt the payload
    let decryptedBody = decrypt(request.content.userInfo["encrypted-content"])

    let newContent = UNMutableNotificationContent()

    // Modify the notification content
    newContent.body = decryptedBody

    // Call content handler with updated content
    contentHandler(newContent)
}
// Decrypt Remote Notification Payload in Service Extension and Update Notification Content

override func didReceive(request: UNNotificationRequest, withContentHandler contentHandler: (UNNotificationContent) -> Void) {

    // Decrypt the payload
    let decryptedBody = decrypt(request.content.userInfo["encrypted-content"])

    let newContent = UNMutableNotificationContent()

    // Modify the notification content
    newContent.body = decryptedBody

    // Call content handler with updated content
    contentHandler(newContent)
}
Summary
Summary

Notifications Overview
Summary

Notifications Overview

User Notifications Framework
Summary

Notifications Overview

User Notifications Framework

• Registration
Summary

Notifications Overview

User Notifications Framework

• Registration

• Content
Summary

Notifications Overview

User Notifications Framework

• Registration
• Content
• Scheduling
Summary

Notifications Overview

User Notifications Framework

• Registration
• Content
• Scheduling
• Management
Summary

Notifications Overview

User Notifications Framework

- Registration
- Content
- Scheduling
- Management
- Actions
Summary

Notifications Overview

User Notifications Framework

• Registration
• Content
• Scheduling
• Management
• Actions

Service Extensions
More Information

## Related Sessions

<table>
<thead>
<tr>
<th>Session</th>
<th>Location</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Advanced Notifications</td>
<td>Pacific Heights</td>
<td>Wednesday 10:00AM</td>
</tr>
<tr>
<td>Quick Interaction Techniques for watchOS</td>
<td>Presidio</td>
<td>Wednesday 11:00AM</td>
</tr>
<tr>
<td>Lab Name</td>
<td>Additional Lab</td>
<td>Date/Time</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>Notifications Lab</td>
<td>Frameworks Lab C</td>
<td>Wednesday 11:00AM</td>
</tr>
<tr>
<td>Notifications Lab</td>
<td>Games, Graphics and</td>
<td>Friday 09:00AM</td>
</tr>
<tr>
<td></td>
<td>Media Lab B</td>
<td></td>
</tr>
</tbody>
</table>