HomeKit

Session 231

Keith Rauenbuehler, HomeKit Engineering
HomeKit overview
Building an accessory
Creating a HomeKit app
HomeKit overview
Building an accessory
Creating a HomeKit app
Ellsworth St

Front door unlocked.
Living Room blinds open.
4 lights on.

and 2 More:

Favorite Scenes

- I'm Home
- Good Morning
- Good Night

Favorite Accessories

- Living Room Thermostat
  Heating to 72°
- Living Room Shades
  Open
- Hallway Light
  70%
- Front Door
  Unlocked
- Dining Room Light
  70%
- Garage Door
  Closed
- Living Room Smoke Detec...

Living Room Thermostat
Heating to 72°
Living Room Shades
Open
Hallway Light
70%
Front Door
Unlocked
Dining Room Light
70%
Garage Door
Closed
Living Room Smoke Detec...

I'm Home
Good Morning
Good Night

Living Room Thermostat
Heating to 72°
Living Room Shades
Open
Hallway Light
70%
Front Door
Unlocked
Dining Room Light
70%
Garage Door
Closed
Living Room Smoke Detec...

I'm home
Good morning
Ellsworth St

Favorite Scenes

- I'm home
- Good morning
- Good night

Favorite Accessories

- 68°F Living Room Thermostat
- Living Room Shades Closed
- Hallway Light
- Front Door Locked
- Dining Room Light 70%
- Garage Door Closed
Quick controls
Siri voice control
Remote Access
Remote Access
Remote Access
Automations

Choose when you want this automation to occur.

- **People Arrive**
  - Ex. "I Arrive Home"

- **People Leave**
  - Ex. "Everyone Has Left Home"

- **A Time of Day Occurs**
  - Ex. "At 8:00 AM" or "At Sunset"

- **An Accessory is Controlled**
  - Ex. "Light Turns On" or "Door Opened"

- **A Sensor Detects Something**
  - Ex. "Motion Detected" or "Smoke Detected"
Accessory Setup

Use the HomeKit Setup Code

Look for the code on the accessory or its packaging and position it in the frame.

OR

Hold iPhone Near the Accessory

You can hold iPhone near this label if it appears on your accessory.

Don’t Have a Code or Can’t Scan?

Learn About Home Accessories
Accessory Setup

Seamless network joining
Accessory Setup

Seamless network joining

QR code scanning
Accessory Setup

Seamless network joining
QR code scanning
NFC
Security and Privacy
Security and Privacy

Encrypted communication
Security and Privacy

Encrypted communication

Perfect forward secrecy
Security and Privacy

- Encrypted communication
- Perfect forward secrecy
- All data is private
HomeKit Accessory Protocol
HomeKit Accessory Protocol

Communication channel to accessory
HomeKit Accessory Protocol

Communication channel to accessory

Secure messaging
HomeKit Accessory Protocol

Communication channel to accessory

Secure messaging

IP and Bluetooth LE transport
HomeKit Accessory Protocol

Communication channel to accessory

Secure messaging

IP and Bluetooth LE transport
HomeKit Accessory Protocol

Communication channel to accessory

Secure messaging

IP and Bluetooth LE transport
HomeKit Accessory Protocol

Communication channel to accessory
Secure messaging
IP and Bluetooth LE transport
Accessory categories
Accessory Categories

- Lights
- Switches
- Outlets
- Thermostats
- Windows
- Fans
- Blinds
- Air Conditioners
- Humidifiers
- Air Purifiers
- Sensors
- Security
- Locks
- Cameras
- Doorbells
- Garage Doors
- Bridges
- Water Valves
Accessory Categories

- Lights
- Switches
- Outlets
- Thermostats
- Windows
- Fans
- Blinds
- Air Conditioners
- Humidifiers
- Air Purifiers
- Sensors
- Security
- Locks
- Cameras
- Doorbells
- Garage Doors
- Bridges
- Water Valves
- Speakers
HomeKit overview
Building an accessory
Creating a HomeKit app
HomeKit Accessory Protocol Specification
HomeKit Accessory Protocol Specification

Become MFi Licensee
http://developer.apple.com/mfi
HomeKit Accessory Protocol Specification

Become MFi Licensee
http://developer.apple.com/mfi

Available to all developers
http://developer.apple.com/homekit
Resources for MFi Partners
Resources for MFi Partners

HomeKit Accessory Development Kit
Resources for MFi Partners

HomeKit Accessory Development Kit

Certification tools
Resources for MFi Partners

HomeKit Accessory Development Kit
Certification tools
Authentication
HomeKit Accessory Development Kit

- Accessory Logic
- Protocol
- Crypto
- WAC
- Bonjour
- Platform Logic
HomeKit Accessory Development Kit

Accessory Logic

HomeKit Accessory Development Kit

Platform Logic
HomeKit Accessory Development Kit

Accessory Logic

HomeKit Accessory Development Kit

Platform Logic
HomeKit Accessory Development Kit

Why use the ADK?
HomeKit Accessory Development Kit

Why use the ADK?

Easier to get started
HomeKit Accessory Development Kit

Why use the ADK?

Easier to get started

Faster to integrate
HomeKit Accessory Development Kit

Why use the ADK?

Easier to get started
Faster to integrate
Reliable and secure
HomeKit Accessory Development Kit

Why use the ADK?

Easier to get started
Faster to integrate
Reliable and secure
Stay in sync with iOS
HomeKit Accessory Development Kit

1 Week

Binary (0 Days) → Integrate (1-2 Days) → Application (3-4 Days) → Prototype
HomeKit Accessory Development Kit

Develop → Integrate → Certify → Ship

12 Months

6 months
HomeKit Accessory Development Kit

3 Months

Develop → Integrate → Certify → Ship

1 month
Accessory Authentication
Accessory Authentication

For MFi partners
Accessory Authentication

For MFi partners

Consumer trusted
Accessory Authentication

For MFi partners

Consumer trusted

Works with Apple HomeKit
Accessory Authentication

For MFi partners

Consumer trusted

Quality certified

Works with Apple HomeKit
Accessory Authentication

For MFi partners
Consumer trusted
Quality certified
Hardware and software based

Works with Apple HomeKit
Authentication
Authentication

Hardware
Authentication

Hardware

Software

11001110001111101110000011000000110001110
00111111000001000111000111111001100100011
1000111001001110100011000000011110001110001
11010011100011001101000111010001110001111001
100001101100000000000111000011110000110010
0000100010000001110110010000111110001111001
1101101100111011000111110001111110011111101
100111111111111111111111111111111111111111
HomeKit Accessory Tester
HomeKit Accessory Tester

Available for MFi licensees
HomeKit Accessory Tester

Available for MFi licensees

Application interface
HomeKit Accessory Tester

Available for MFi licensees

Application interface

Great for testing your accessory
HomeKit Certification Assistant
HomeKit Certification Assistant

Available for MFi licensees
HomeKit Certification Assistant

Available for MFi licensees

Self-certification tool
HomeKit Certification Assistant

Available for MFi licensees

Self-certification tool

Automated test execution
HomeKit Accessory Simulator
HomeKit Accessory Simulator

Simulate any HomeKit accessory
HomeKit Accessory Simulator

Simulate any HomeKit accessory

Great for testing your app
HomeKit Accessory Simulator

Simulate any HomeKit accessory

Great for testing your app

No physical accessories required
HomeKit Accessory Simulator

Simulate any HomeKit accessory

Great for testing your app

No physical accessories required

Available via Hardware IO Tools for Xcode
Making Your App Unique
Making Your App Unique

Custom functionality
Making Your App Unique

Custom functionality

Region-specific use cases
Making Your App Unique

Custom functionality

Region-specific use cases

Custom services
Home Object Hierarchy
Home Object Hierarchy

HMHome
Home Object Hierarchy

HMHome

HMZone
Home Object Hierarchy

HMHome
HMZone
HMRoom
Home Object Hierarchy

HMHome
HMZone
HMRoom
HMAccessory
Home Management
HMHome, HMHomeManager
Home Management
HMHome, HMHomeManager
extension HMHome {
    var currentUser: HMUser { get }

    func manageUsers(completionHandler: @escaping (Error?) -> Void)
    func homeAccessControl(for user: HMUser) -> HMHomeAccessControl
}
extension HMHome {
    var currentUser: HMUser { get }

    func manageUsers(completionHandler: @escaping (Error?) -> Void)
    func homeAccessControl(for user: HMUser) -> HMHomeAccessControl
}
extension HMHome {
    var currentUser: HMUser { get }

    func manageUsers(completionHandler: @escaping (Error?) -> Void)

    func homeAccessControl(for user: HMUser) -> HMHomeAccessControl
}

extension HMHome {
    var currentUser: HMUser { get }

    func manageUsers(completionHandler: @escaping (Error?) -> Void)
    func homeAccessControl(for user: HMUser) -> HMHomeAccessControl
}

class HMHomeAccessControl : NSObject {
    var administrator: Bool { get }
}
// Create Home & Manage Users
import HomeKit

class MyClass: NSObject, HMHomeDelegate {
    let homeManager = HMHomeManager()
    var myHome: HMHome?

    func configureHome(withName name: String) {
        homeManager.addHome(withName: name) { home, error in
            guard let home = home else { /* Handle error */ return }

            self.myHome = home
            home.delegate = self
            home.manageUsers { error in
                guard error == nil else { /* Handle error */ return }
            }
        }
    }
}
import HomeKit

class MyClass: NSObject, HMHomeDelegate {
    let homeManager = HMHomeManager()
    var myHome: HMHome?

    func configureHome(withName name: String) {
        homeManager.addHome(withName: name) { home, error in
            guard let home = home else { /* Handle error */ return }
            self.myHome = home
            home.delegate = self
            home.manageUsers { error in
                guard error == nil else { /* Handle error */ return }
            }
        }
    }
}
import HomeKit

class MyClass: NSObject, HMHomeDelegate {
    let homeManager = HMHomeManager()
    var myHome: HMHome?

    func configureHome(withName name: String) {
        homeManager.addHome(withName: name) { home, error in
            guard let home = home else { /* Handle error */ return }

            self.myHome = home
            home.delegate = self
            home.manageUsers { error in
                guard error == nil else { /* Handle error */ return }
            }
        }
    }
}
// Create Home & Manage Users

import HomeKit

class MyClass: NSObject, HMHomeDelegate {
    let homeManager = HMHomeManager()
    var myHome: HMHome?

    func configureHome(withName name: String) {
        homeManager.addHome(withName: name) { home, error in
            guard let home = home else { /* Handle error */ return }

            self.myHome = home
            home.delegate = self
            home.manageUsers { error in
                guard error == nil else { /* Handle error */ return }
            }
        }
    }
}
// Create Home & Manage Users

import HomeKit

class MyClass: NSObject, HMHomeDelegate {
    let homeManager = HMHomeManager()
    var myHome: HMHome?

    func configureHome(withName name: String) {
        homeManager.addHome(withName: name) { home, error in
            guard let home = home else { /* Handle error */ return }

            self.myHome = home
            home.delegate = self
            home.manageUsers { error in
                guard error == nil else { /* Handle error */ return }
            }
        }
    }
}
Rooms and Zones
HMRoom, HMZone

class HMZone : NSObject {
    var name: String { get }
    var uniqueIdentifier: UUID { get }
    var rooms: [HMRoom] { get }
}

class HMRoom : NSObject {
    var name: String { get }
    var uniqueIdentifier: UUID { get }
    var accessories: [HMAccessory] { get }
}
Rooms and Zones
HMRoom, HMZone

class HMZone : NSObject {
    var name: String { get }
    var uniqueIdentifier: UUID { get }
    var rooms: [HMRoom] { get }
}

class HMRoom : NSObject {
    var name: String { get }
    var uniqueIdentifier: UUID { get }
    var accessories: [HMAccessory] { get }
}
class HMZone: NSObject {
    var name: String { get }
    var uniqueIdentifier: UUID { get }
    var rooms: [HMRoom] { get }
}

class HMRoom: NSObject {
    var name: String { get }
    var uniqueIdentifier: UUID { get }
    var accessories: [HMAccessory] { get }
}
class HMZone: NSObject {
    var name: String { get }
    var uniqueIdentifier: UUID { get }
    var rooms: [HMRoom] { get }
}

class HMRoom: NSObject {
    var name: String { get }
    var uniqueIdentifier: UUID { get }
    var accessories: [HMAccessory] { get }
}
extension HMHome {
    var rooms: [HMRoom] { get }
    func addRoom(withName name: String, completionHandler: @escaping (HMRoom?, Error?) -> Void)
    func removeRoom(_ room: HMRoom, completionHandler: @escaping (Error?) -> Void)
}

extension HMHome {
    var zones: [HMZone] { get }
    func addZone(withName name: String, completionHandler: @escaping (HMZone?, Error?) -> Void)
    func removeZone(_ zone: HMZone, completionHandler: @escaping (Error?) -> Void)
}
extension HMHome {

  var rooms: [HMRoom] { get }

  func addRoom(withName name: String, completionHandler: @escaping (HMRoom?, Error?) -> Void)
  func removeRoom(_ room: HMRoom, completionHandler: @escaping (Error?) -> Void)
}

extension HMHome {

  var zones: [HMZone] { get }

  func addZone(withName name: String, completionHandler: @escaping (HMZone?, Error?) -> Void)
  func removeZone(_ zone: HMZone, completionHandler: @escaping (Error?) -> Void)
}
Rooms and Zones
HMHome integration

extension HMHome {
    var rooms: [HMRoom] { get }
    func addRoom(withName name: String, completionHandler: @escaping (HMRoom?, Error?) -> Void)
    func removeRoom(_ room: HMRoom, completionHandler: @escaping (Error?) -> Void)
}

extension HMHome {
    var zones: [HMZone] { get }
    func addZone(withName name: String, completionHandler: @escaping (HMZone?, Error?) -> Void)
    func removeZone(_ zone: HMZone, completionHandler: @escaping (Error?) -> Void)
}
// Create Room & Zone

func addRoom(withName roomName: String, zoneName: String, home: HMHome) {
    home.addRoom(withName: roomName) { room, error in
        guard let room = room else { /* Handle error */ return }

        home.addZone(withName: zoneName) { zone, error in
            guard let zone = zone else { /* Handle error */ return }

            zone.addRoom(room) { error in
                guard error == nil else { /* Handle error */ return }
            }
        }
    }
}
func addRoom(withName roomName: String, zoneName: String, home: HMHome) {
    home.addRoom(withName: roomName) { room, error in
        guard let room = room else { /* Handle error */ return }
        home.addZone(withName: zoneName) { zone, error in
            guard let zone = zone else { /* Handle error */ return }
            zone.addRoom(room) { error in
                guard error == nil else { /* Handle error */ return }
            }
        }
    }
}
// Create Room & Zone

func addRoom(withName roomName: String, zoneName: String, home: HMHome) {
    home.addRoom(withName: roomName) { room, error in
        guard let room = room else { /* Handle error */ return }

        home.addZone(withName: zoneName) { zone, error in
            guard let zone = zone else { /* Handle error */ return }

            zone.addRoom(room) { error in
                guard error == nil else { /* Handle error */ return }
            }
        }
    }
}
}
func addRoom(withName roomName: String, zoneName: String, home: HMHome) {
    home.addRoom(withName: roomName) { room, error in
        guard let room = room else { /* Handle error */ return }
    }

    home.addZone(withName: zoneName) { zone, error in
        guard let zone = zone else { /* Handle error */ return }
    }

    zone.addRoom(room) { error in
        guard error == nil else { /* Handle error */ return }
    }
}
Accessories, Services, and Characteristics
Accessories, Services, and Characteristics
Accessories, Services, and Characteristics
Accessories, Services, and Characteristics
Accessories, Services, and Characteristics
Accessories, Services, and Characteristics

- Fan
- Light
Accessories, Services, and Characteristics

- Power
- Brightness
- Saturation
- Light
- Fan
Accessories, Services, and Characteristics

- Power
- Brightness
- Saturation

Light

- Power
- Rotation Speed
- Rotation Direction

Fan
Accessories, Services, and Characteristics

HMAccessory

- Power
- Brightness
- Saturation
- Rotation Speed
- Rotation Direction

- Light
- Fan
Accessories, Services, and Characteristics

HMAccessory

HMService

- **Power**
- **Brightness**
- **Saturation**

- **Light**
- **Fan**

- **Power**
- **Rotation Speed**
- **Rotation Direction**
Accessories, Services, and Characteristics

HMAccessory

HMServices
- Power
- Brightness
- Saturation

HMCharacteristics
- Power
- Rotation Speed
- Rotation Direction
class HMAccessory : NSObject {
    var name: String { get }
    var category: HMAccessoryCategory { get }
    var services: [HMService] { get }
    var model: String? { get }
    var manufacturer: String? { get }
    var firmwareVersion: String? { get }
}
class HMAccessory : NSObject {
    var name: String { get }
    var category: HMAccessoryCategory { get }
    var services: [HMService] { get }
    var model: String? { get }
    var manufacturer: String? { get }
    var firmwareVersion: String? { get }
}
class HMAccessory : NSObject {
    var name: String { get }
    var category: HMAccessoryCategory { get }
    var services: [HMService] { get }
    var model: String? { get }
    var manufacturer: String? { get }
    var firmwareVersion: String? { get }
}
class HMAccessory : NSObject {
    var name: String { get }
    var category: HMAccessoryCategory { get }
    var services: [HMService] { get }
    var model: String? { get }
    var manufacturer: String? { get }
    var firmwareVersion: String? { get }
}

class HMService: NSObject {
    var serviceType: String { get }
    var name: String { get }
    var characteristics: [HMCharacteristic] { get }
    var isUserInteractive: Bool { get }

    func updateName(_ name: String, completionHandler: @escaping (Error?) -> Void)
}
class HMService : NSObject {
    var serviceType: String { get }
    var name: String { get }
    var characteristics: [HMCharacteristic] { get }
    var isUserInteractive: Bool { get }

    func updateName(_ name: String, completionHandler: @escaping (Error?) -> Void)
}
class HMService : NSObject {
    var serviceType: String { get }
    var name: String { get }
    var characteristics: [HMCharacteristic] { get }
    var isUserInteractive: Bool { get }

    func updateName(_ name: String, completionHandler: @escaping (Error?) -> Void)
}
Services
Naming best practices
Services

Naming best practices

Used by Home app and Siri
Services
Naming best practices

Used by Home app and Siri

Choose good default names
Services
Naming best practices

Used by Home app and Siri

Choose good default names
Services

Naming best practices

Used by Home app and Siri

Choose good default names

No special characters or numbers
Services
Naming best practices

Used by Home app and Siri
Choose good default names
No special characters or numbers
Service Types
Predefined service types

public let HMServiceTypeLightbulb: String
public let HMServiceTypeSwitch: String
public let HMServiceTypeThermostat: String
public let HMServiceTypeGarageDoorOpener: String
public let HMServiceTypeFan: String
public let HMServiceTypeOutlet: String
public let HMServiceTypeContactSensor: String
...
...
public let HMServiceTypeMotionSensor: String
public let HMServiceTypeSmokeSensor: String
public let HMServiceTypeWindowCovering: String
public let HMServiceTypeValve: String
extension HMHome {
    var accessories: [HMAccessory] { get }

    func addAndSetupAccessories(completionHandler: @escaping (Error?) -> Void)
    func addAndSetupAccessories(with payload: HMAccessorySetupPayload, completionHandler: @escaping ([HMAccessory]?, Error?) -> Void)
}
Accessories
HMHome integration

```swift
extension HMHome {
    var accessories: [HMAccessory] { get }

    func addAndSetupAccessories(completionHandler: @escaping (Error?) -> Void)
    func addAndSetupAccessories(with payload: HMAccessorySetupPayload, completionHandler: @escaping ([HMAccessory]?, Error?) -> Void)
}
```
extension HMHome {
    var accessories: [HMAccessory] { get }

    func addAndSetupAccessories(completionHandler: @escaping (Error?) -> Void)
    func addAndSetupAccessories(with payload: HMAccessorySetupPayload, completionHandler: @escaping ([HMAccessory]?, Error?) -> Void)
}
// Add an accessory

func addAccessory(to home: HMHome) {
    home.addAndSetupAccessories { error in
        guard error == nil else { /* Handle error */ return }
    }
}
// Add an accessory

func addAccessory(to home: HMHome) {
    home.addAndSetupAccessories { error in
        guard error == nil else { /* Handle error */ return }
    }
}
Scenes
Scenes

Integrate accessories together
Scenes

Integrate accessories together

Predefined scenes with suggested actions
Scenes

Integrate accessories together

Predefined scenes with suggested actions

User customizable
Scenes

Integrate accessories together

Predefined scenes with suggested actions

User customizable

Built-in Siri support
Scenes
HMActionSet
Scenes

HMActionSet

Arrive Home
Scenes
HMActionSet

- Arrive Home
- Unlock doors
- Raise shades
- Open garage
- Open gates
- Switch on
- Temp up
- Light on
Scenes
HMActionSet

- Arrive Home
  - Open gates
  - Open garage
  - Raise shades
  - Switch on
  - Temp up
  - Unlock doors
  - Light on

Unlock doors
Raise shades
Temp up
Switch on
Open garage
Open gates
Light on
Arrive Home
Scenes
Predefined action set types
Scenes
Predefined action set types

HMActionSetTypeWakeUp
Scenes
Predefined action set types

HMActionSetTypeWakeUp

HMActionSetTypeSleep
Scenes
Predefined action set types

HMActionSetTypeWakeUp
HMActionSetTypeSleep
HMActionSetTypeHomeDeparture
Scenes
Predefined action set types

HMAActionSetTypeWakeUp
HMAActionSetTypeSleep
HMAActionSetTypeHomeDeparture
HMAActionSetTypeHomeArrival
Scenes
Predefined action set types

HMAActionSetTypeWakeUp
HMAActionSetTypeSleep
HMAActionSetTypeHomeDeparture
HMAActionSetTypeHomeArrival
HMAActionSetTypeUserDefined
Scenes
HMHome integration

```swift
extension HMHome {
    var actionSets: [HMActionSet] { get }
    func addActionSet(withName: String, completionHandler: @escaping (HMActionSet?, Error?) -> Void)
    func removeActionSet(_: HMActionSet, completionHandler: @escaping (Error?) -> Void)
    func executeActionSet(_: HMActionSet, completionHandler: @escaping (Error?) -> Void)
    func builtinActionSet(ofType actionSetType: String) -> HMActionSet?
}
```
extension HMHome {
    var actionSets: [HMActionSet] { get }

    func addActionSet(withName: String, completionHandler: @escaping (HMActionSet?, Error?) -> Void)

    func removeActionSet(_: HMActionSet, completionHandler: @escaping (Error?) -> Void)

    func executeActionSet(_: HMActionSet, completionHandler: @escaping (Error?) -> Void)

    func builtinActionSet(ofType actionSetType: String) -> HMActionSet?
}
extension HMHome {

    var actionSets: [HMActionSet] { get }

    func addActionSet(withName: String, completionHandler: @escaping (HMActionSet?, Error?) -> Void)
    func removeActionSet(_: HMActionSet, completionHandler: @escaping (Error?) -> Void)
    func executeActionSet(_: HMActionSet, completionHandler: @escaping (Error?) -> Void)
    func builtinActionSet(ofType actionSetType: String) -> HMActionSet?

}
extension HMHome {
    var actionSets: [HMActionSet] { get }
    func addActionSet(withName: String, completionHandler: @escaping (HMActionSet?, Error?) -> Void)
    func removeActionSet(_ actionSet: HMActionSet, completionHandler: @escaping (Error?) -> Void)
    func executeActionSet(_ actionSet: HMActionSet, completionHandler: @escaping (Error?) -> Void)
    func builtinActionSet(ofType actionSetType: String) -> HMActionSet?
}
func createActionSet(withName name: String, actions: [HMAction], home: HMHome) {
    home.addActionSet(withName: name) { actionSet, error in
        guard let actionSet = actionSet else { /* Handle error */ return }

        for action in actions {
            actionSet.addAction(action) { error in
                guard error == nil else { /* Handle error */ return }
            }
        }
    }
}

func executeActionSet(_ actionSet: HMActionSet, home: HMHome) {
    home.executeActionSet(actionSet) { error in
        guard error == nil else { /* Handle error */ return }
    }
}
```swift
func createActionSet(withName name: String, actions: [HMAction], home: HMHome) {
    home.addActionSet(withName: name) { actionSet, error in
        guard let actionSet = actionSet else { /* Handle error */ return }

        for action in actions {
            actionSet.addAction(action) { error in
                guard error == nil else { /* Handle error */ return }
            }
        }
    }
}

func executeActionSet(_ actionSet: HMActionSet, home: HMHome) {
    home.executeActionSet(actionSet) { error in
        guard error == nil else { /* Handle error */ return }
    }
}
```
```swift
func createActionSet(withName name: String, actions: [HMAction], home: HMHome) {
    home.addActionSet(withName: name) { actionSet, error in
        guard let actionSet = actionSet else { /* Handle error */ return }
        for action in actions {
            actionSet.addAction(action) { error in
                guard error == nil else { /* Handle error */ return }
            }
        }
    }
}

func executeActionSet(_ actionSet: HMActionSet, home: HMHome) {
    home.executeActionSet(actionSet) { error in
        guard error == nil else { /* Handle error */ return }
    }
}
```
func createActionSet(withName name: String, actions: [HMAction], home: HMHome) {
    home.addActionSet(withName: name) { actionSet, error in
        guard let actionSet = actionSet else { /* Handle error */ return }
        for action in actions {
            actionSet.addAction(action) { error in
                guard error == nil else { /* Handle error */ return }
            }
        }
    }
}

func executeActionSet(_ actionSet: HMActionSet, home: HMHome) {
    home.executeActionSet(actionSet) { error in
        guard error == nil else { /* Handle error */ return }
    }
}
func createActionSet(withName name: String, actions: [HMAction], home: HMHome) {
    home.addActionSet(withName: name) { actionSet, error in
        guard let actionSet = actionSet else { /* Handle error */ return }

        for action in actions {
            actionSet.addAction(action) { error in
                guard error == nil else { /* Handle error */ return }
            }
        }
    }
}

func executeActionSet(_ actionSet: HMActionSet, home: HMHome) {
    home.executeActionSet(actionSet) { error in
        guard error == nil else { /* Handle error */ return }
    }
}
Event Triggers
Event Triggers

Location

Set your lights to turn on as soon as you pull up to the house.
Event Triggers

Location
Set your lights to turn on as soon as you pull up to the house.

Time
Have your home turn up the heat at 6:00 a.m., before you get out of bed.
Event Triggers

**Location**
Set your lights to turn on as soon as you pull up to the house.

**Time**
Have your home turn up the heat at 6:00 a.m., before you get out of bed.

**Action**
Use a motion sensor in the doorway to turn your kitchen lights on when you walk in.
Event Triggers

**Location**
Set your lights to turn on as soon as you pull up to the house.

**Time**
Have your home turn up the heat at 6:00 a.m., before you get out of bed.

**Action**
Use a motion sensor in the doorway to turn your kitchen lights on when you walk in.

**Occupancy**
Trigger a scene based on who’s home, like automatically turning the lights off as you leave if nobody else is there.
Event Triggers

HMTrigger

class HMTrigger : NSObject {
    var name: String { get }
    var isEnabled: Bool { get }
    var actionSets: [HMActionSet] { get }
    func updateName(_ name: String, completionHandler: @escaping (Error?) -> Void)
    func addActionSet(_ actionSet: HMActionSet, completionHandler: @escaping (Error?) -> Void)
    func enable(_ enable: Bool, completionHandler: @escaping (Error?) -> Void)
}
class HMTrigger: NSObject {
    var name: String { get }
    var isEnabled: Bool { get }
    var actionSets: [HMActionSet] { get }

    func updateName(_ name: String, completionHandler: @escaping (Error?) -> Void)
    func addActionSet(_ actionSet: HMActionSet, completionHandler: @escaping (Error?) -> Void)
    func enable(_ enable: Bool, completionHandler: @escaping (Error?) -> Void)
}
class HMTrigger : NSObject {
    var name: String { get }
    var isEnabled: Bool { get }
    var actionSets: [HMActionSet] { get }

    func updateName(_ name: String, completionHandler: @escaping (Error?) -> Void)
    func addActionSet(_ actionSet: HMActionSet, completionHandler: @escaping (Error?) -> Void)
    func enable(_ enable: Bool, completionHandler: @escaping (Error?) -> Void)
}
Event Triggers

HMTrigger

class HMTrigger : NSObject {
    var name: String { get }
    var isEnabled: Bool { get }
    var actionSets: [HMActionSet] { get }
    func updateName(_ name: String, completionHandler: @escaping (Error?) -> Void)
    func addActionSet(_ actionSet: HMActionSet, completionHandler: @escaping (Error?) -> Void)
    func enable(_ enable: Bool, completionHandler: @escaping (Error?) -> Void)
}
Event Triggers

HMTrigger

class HMTrigger : NSObject {
    var name: String { get }
    var isEnabled: Bool { get }
    var actionSets: [HMActionSet] { get }

    func updateName(_ name: String, completionHandler: @escaping (Error?) -> Void)
    func addActionSet(_ actionSet: HMActionSet, completionHandler: @escaping (Error?) -> Void)
    func enable(_ enable: Bool, completionHandler: @escaping (Error?) -> Void)
}
Event Triggers
HMHome integration
Event Triggers
HMHome integration
Event Triggers
HMHome integration
Event Triggers
HMHome integration
Event Triggers

HMHome integration
// Create 7PM Trigger

func create7PMTrigger(withName name: String, actionSet: HMActionSet, home: HMHome) {
    var components = DateComponents()
    components.hour = 19

    let timeEvent = HMCalendarEvent(fire: components)
    let trigger = HMEventTrigger(name: name, events: [timeEvent], predicate: nil)

    home.addTrigger(trigger) { error in
        guard error == nil else { /* Handle error */ return }

        trigger.addActionSet(actionSet) { error in
            guard error == nil else { /* Handle error */ return }
        }
    }
}
// Create 7PM Trigger

```swift
func create7PMTrigger(withName name: String, actionSet: HMActionSet, home: HMHome) {
    var components = DateComponents()
    components.hour = 19

    let timeEvent = HMCalendarEvent(fire: components)
    let trigger = HMEventTrigger(name: name, events: [timeEvent], predicate: nil)

    home.addTrigger(trigger) { error in
        guard error == nil else { /* Handle error */ return }
    }

    trigger.addActionSet(actionSet) { error in
        guard error == nil else { /* Handle error */ return }
    }
}
```
// Create 7PM Trigger

func create7PMTrigger(withName name: String, actionSet: HMActionSet, home: HMHome) {
  var components = DateComponents()
  components.hour = 19

  let timeEvent = HMCalendarEvent(fire: components)
  let trigger = HMEventTrigger(name: name, events: [timeEvent], predicate: nil)

  home.addTrigger(trigger) { error in
    guard error == nil else { /* Handle error */ return }

    trigger.addActionSet(actionSet) { error in
      guard error == nil else { /* Handle error */ return }
    }
  }
}
Create 7PM Trigger

```swift
func create7PMTrigger(withName name: String, actionSet: HMActionSet, home: HMHome) {
    var components = DateComponents()
    components.hour = 19

    let timeEvent = HMCalendarEvent(fire: components)
    let trigger = HMEventTrigger(name: name, events: [timeEvent], predicate: nil)

    home.addTrigger(trigger) { error in
        guard error == nil else { /* Handle error */ return }
    }

    trigger.addActionSet(actionSet) { error in
        guard error == nil else { /* Handle error */ return }
    }
}
```
// Create 7PM Trigger

func create7PMTrigger(withName name: String, actionSet: HMActionSet, home: HMHome) {
    var components = DateComponents()
    components.hour = 19

    let timeEvent = HMCalendarEvent(fire: components)
    let trigger = HMEventTrigger(name: name, events: [timeEvent], predicate: nil)

    home.addTrigger(trigger) { error in
        guard error == nil else { /* Handle error */ return }
    }

    trigger.addActionSet(actionSet) { error in
        guard error == nil else { /* Handle error */ return }
    }
}
// Create 7PM Trigger

func create7PMTrigger(withName name: String, actionSet: HMActionSet, home: HMHome) {
    var components = DateComponents()
    components.hour = 19

    let timeEvent = HMCalendarEvent(fire: components)
    let trigger = HMEventTrigger(name: name, events: [timeEvent], predicate: nil)

    home.addTrigger(trigger) { error in
        guard error == nil else { /* Handle error */ return }
    }

    trigger.addActionSet(actionSet) { error in
        guard error == nil else { /* Handle error */ return }
    }
}
}
Significant Time Events

HMSignificantTimeEvent
func createSunsetTrigger(withName name: String, actionSet: HMActionSet, home: HMHome) {
    var offset = DateComponents()
    offset.minute = -30

    let sunsetEvent = HMSignificantTimeEvent(significantEvent: .sunset, offset: offset)
    let trigger = HMEventTrigger(name: name, events: [sunsetEvent], predicate: nil)

    home.addTrigger(trigger) { error in
        guard error == nil else { /* Handle error */ return }

        trigger.addActionSet(actionSet) { error in
            guard error == nil else { /* Handle error */ return }
        }
    }
}
// Create Sunset Trigger

func createSunsetTrigger(withName name: String, actionSet: HMActionSet, home: HMHome) {
    var offset = DateComponents()
    offset.minute = -30

    let sunsetEvent = HMSignificantTimeEvent(significantEvent: .sunset, offset: offset)
    let trigger = HMEventTrigger(name: name, events: [sunsetEvent], predicate: nil)

    home.addTrigger(trigger) { error in
        guard error == nil else { /* Handle error */ return }
    }

    trigger.addActionSet(actionSet) { error in
        guard error == nil else { /* Handle error */ return }
    }
}
// Create Sunset Trigger

func createSunsetTrigger(withName name: String, actionSet: HMActionSet, home: HMHome) {
    var offset = DateComponents()
    offset.minute = -30

    let sunsetEvent = HMSignificantTimeEvent(significantEvent: .sunset, offset: offset)
    let trigger = HMEventTrigger(name: name, events: [sunsetEvent], predicate: nil)

    home.addTrigger(trigger) { error in
        guard error == nil else { /* Handle error */ return }
    }

    trigger.addActionSet(actionSet) { error in
        guard error == nil else { /* Handle error */ return }
    }
}
// Create Sunset Trigger

func createSunsetTrigger(withName name: String, actionSet: HMActionSet, home: HMHome) {
    var offset = DateComponents()
    offset.minute = -30

    let sunsetEvent = HMSignificantTimeEvent(significantEvent: .sunset, offset: offset)
    let trigger = HMEventTrigger(name: name, events: [sunsetEvent], predicate: nil)

    home.addTrigger(trigger) { error in
        guard error == nil else { /* Handle error */ return }
    }

    trigger.addActionSet(actionSet) { error in
        guard error == nil else { /* Handle error */ return }
    }
}
// Create Sunset Trigger

func createSunsetTrigger(withName name: String, actionSet: HMActionSet, home: HMHome) {
    var offset = DateComponents()
    offset.minute = -30

    let sunsetEvent = HMSignificantTimeEvent(significantEvent: .sunset, offset: offset)
    let trigger = HMEventTrigger(name: name, events: [sunsetEvent], predicate: nil)
    home.addTrigger(trigger) { error in
        guard error == nil else { /* Handle error */ return }
    }
    trigger.addActionSet(actionSet) { error in
        guard error == nil else { /* Handle error */ return }
    }
}
Event Trigger Conditions
Event Trigger Conditions

Conditionalize automated controls
Event Trigger Conditions

Conditionalize automated controls
• Time
Event Trigger Conditions

Conditionalize automated controls

- Time
- Presence
Event Trigger Conditions

Conditionalize automated controls
- Time
- Presence
- Accessory conditions
Conditionalize automated controls

- Time
- Presence
- Accessory conditions
- Significant events
// Create Sunset Presence Trigger

func createSunsetPresenceTrigger(withName name: String, actionSet: HMActionSet, home: HMHome) {
    var offset = DateComponents()
    offset.minute = -30
    let sunsetEvent = HMSignificantTimeEvent(significantEvent: .sunset, offset: offset)

    let presence = HMPresenceEvent(presenceEventType: .atHome, presenceUserType: .homeUsers)
    let predicate = HMEventTrigger.predicateForEvaluatingTrigger(withPresence: presence)
    let trigger = HMEventTrigger(name: name, events: [sunsetEvent], predicate: predicate)

    home.addTrigger(trigger) { error in
    guard error == nil else { /* Handle error */ return }

    trigger.addActionSet(actionSet) { error in
    guard error == nil else { /* Handle error */ return }
    }
    }
}
// Create Sunset Presence Trigger
func createSunsetPresenceTrigger(withName name: String, actionSet: HMActionSet, home: HMHome) {
    var offset = DateComponents()
    offset.minute = -30
    let sunsetEvent = HMSignificantTimeEvent(significantEvent: .sunset, offset: offset)

    let presence = HMPresenceEvent(presenceEventType: .atHome, presenceUserType: .homeUsers)
    let predicate = HMEventTrigger.predicateForEvaluatingTrigger(withPresence: presence)
    let trigger = HMEventTrigger(name: name, events: [sunsetEvent], predicate: predicate)

    home.addTrigger(trigger) { error in
        guard error == nil else { /* Handle error */ return }
    }

    trigger.addActionSet(actionSet) { error in
        guard error == nil else { /* Handle error */ return }
    }
}
}
func createSunsetPresenceTrigger(withName name: String, actionSet: HMActionSet, home: HMHome) {
    var offset = DateComponents()
    offset.minute = -30
    let sunsetEvent = HMSignificantTimeEvent(significantEvent: .sunset, offset: offset)

    let presence = HMPresenceEvent(presenceEventType: .atHome, presenceUserType: .homeUsers)
    let predicate = HMEventTrigger.predicateForEvaluatingTrigger(withPresence: presence)
    let trigger = HMEventTrigger(name: name, events: [sunsetEvent], predicate: predicate)

    home.addTrigger(trigger) { error in
        guard error == nil else { /* Handle error */ return }
    }

    trigger.addActionSet(actionSet) { error in
        guard error == nil else { /* Handle error */ return }
    }
}
// Create Sunset Presence Trigger

func createSunsetPresenceTrigger(withName name: String, actionSet: HMActionSet, home: HMHome) {
    var offset = DateComponents()
    offset.minute = -30
    let sunsetEvent = HMSignificantTimeEvent(significantEvent: .sunset, offset: offset)
    let presence = HMPresenceEvent(presenceEventType: .atHome, presenceUserType: .homeUsers)
    let predicate = HMEventTrigger.predicateForEvaluatingTrigger(withPresence: presence)
    let trigger = HMEventTrigger(name: name, events: [sunsetEvent], predicate: predicate)

    // Add trigger to the home
    home.addTrigger(trigger) { error in
        guard error == nil else { /* Handle error */ return }
    }

    // Add action set to the trigger
    trigger.addActionSet(actionSet) { error in
        guard error == nil else { /* Handle error */ return }
    }
}

// Create Sunset Presence Trigger

func createSunsetPresenceTrigger(withName name: String, actionSet: HMActionSet, home: HMHome) {
    var offset = DateComponents()
    offset.minute = -30
    let sunsetEvent = HMSignificantTimeEvent(significantEvent: .sunset, offset: offset)
    let presence = HMPresenceEvent(presenceEventType: .atHome, presenceUserType: .homeUsers)
    let predicate = HMEventTrigger.predicateForEvaluatingTrigger(withPresence: presence)
    let trigger = HMEventTrigger(name: name, events: [sunsetEvent], predicate: predicate)

    home.addTrigger(trigger) { error in
        guard error == nil else { /* Handle error */ return }
    }

    trigger.addActionSet(actionSet) { error in
        guard error == nil else { /* Handle error */ return }
    }
}
}
// Create Sunset Presence Trigger
func createSunsetPresenceTrigger(withName name: String, actionSet: HMActionSet, home: HMHome) {
    var offset = DateComponents()
    offset.minute = -30
    let sunsetEvent = HMSignificantTimeEvent(significantEvent: .sunset, offset: offset)

    let presence = HMPresenceEvent(presenceEvent: .atHome, presenceUserType: .homeUsers)
    let predicate = HMEventTrigger.predicateForEvaluatingTrigger(withPresence: presence)
    let trigger = HMEventTrigger(name: name, events: [sunsetEvent], predicate: predicate)

    home.addTrigger(trigger) { error in
        guard error == nil else { /* Handle error */ return }
        trigger.addActionSet(actionSet) { error in
            guard error == nil else { /* Handle error */ return }
        }
    }
}
Camera Accessories
Camera Accessories

Show live streams
Camera Accessories

Show live streams
Display still images
Camera Accessories

Show live streams
Display still images
Control the camera settings
Camera Accessories

Show live streams
Display still images
Control the camera settings
Control the speaker and microphone
import HomeKit

class MyCameraClass: UIViewController {
    var liveStreamView: HMCameraView?

    func startCameraStream(for accessory: HMAccessory) {
        // Ensure this is a camera accessory
        guard let cameraStreamControl = accessory.cameraProfiles?.first?.streamControl else {
            return
        }

        cameraStreamControl.delegate = self
        cameraStreamControl.startStream()

        let liveStreamView = HMCameraView()
        self.view.addSubview(liveStreamView)
        self.liveStreamView = liveStreamView
    }
}
import HomeKit

class MyCameraClass: UIViewController {
    var liveStreamView: HMCameraView?

    func startCameraStream(for accessory: HMAccessory) {
        // Ensure this is a camera accessory
        guard let cameraStreamControl = accessory.cameraProfiles?.first?.streamControl else {
            return
        }

        cameraStreamControl.delegate = self
        cameraStreamControl.startStream()

        let liveStreamView = HMCameraView()
        self.view.addSubview(liveStreamView)
        self.liveStreamView = liveStreamView
    }
}
import HomeKit

class MyCameraClass: UIViewController {
    var liveStreamView: HMCameraView?

    func startCameraStream(for accessory: HMAccessory) {
        // Ensure this is a camera accessory
        guard let cameraStreamControl = accessory.cameraProfiles?.first?.streamControl else
        { return }

        cameraStreamControl.delegate = self
        cameraStreamControl.startStream()

        let liveStreamView = HMCameraView()
        self.view.addSubview(liveStreamView)
        self.liveStreamView = liveStreamView
    }
}
import HomeKit

class MyCameraClass: UIViewController {
    var liveStreamView: HMCameraView?

    func startCameraStream(for accessory: HMAccessory) {
        // Ensure this is a camera accessory
        guard let cameraStreamControl = accessory.cameraProfiles?.first?.streamControl else
        { return }

        cameraStreamControl.delegate = self
        cameraStreamControl.startStream()

        let liveStreamView = HMCameraView()
        self.view.addSubview(liveStreamView)
        self.liveStreamView = liveStreamView
    }
}
import HomeKit

class MyCameraClass: UIViewController {
    var liveStreamView: HMCameraView?

    func startCameraStream(for accessory: HMAccessory) {
        // Ensure this is a camera accessory
        guard let cameraStreamControl = accessory.cameraProfiles?.first?.streamControl else {
            return
        }

        cameraStreamControl.delegate = self
        cameraStreamControl.startStream()

        let liveStreamView = HMCameraView()
        self.view.addSubview(liveStreamView)
        self.liveStreamView = liveStreamView
    }
}
import HomeKit

class MyCameraClass: UIViewController {
    var liveStreamView: HMCameraView?

    func startCameraStream(for accessory: HMAccessory) {
        // Ensure this is a camera accessory
        guard let cameraStreamControl = accessory.cameraProfiles?.first?.streamControl else {
            return
        }

        cameraStreamControl.delegate = self
        cameraStreamControl.startStream()

        let liveStreamView = HMCameraView()
        self.view.addSubview(liveStreamView)
        self.liveStreamView = liveStreamView
    }
}
extension MyCameraClass: HMCameraStreamControlDelegate {
    func cameraStreamControlDidStartStream(_ cameraStreamControl: HMCameraStreamControl) {
        liveStreamView?.cameraSource = cameraStreamControl.cameraStream
    }
}
extension MyCameraClass: HMCameraStreamControlDelegate {
    func cameraStreamControlDidStartStream(_ cameraStreamControl: HMCameraStreamControl) {
        liveStreamView?.cameraSource = cameraStreamControl.cameraStream
    }
}
extension MyCameraClass: HMCameraStreamControlDelegate {
    func cameraStreamControlDidStartStream(_ cameraStreamControl: HMCameraStreamControl) {
        liveStreamView?.cameraSource = cameraStreamControl.cameraStream
    }
}
extension MyCameraClass: HMCameraStreamControlDelegate {
    func cameraStreamControlDidStartStream(_ cameraStreamControl: HMCameraStreamControl) {
        liveStreamView?.cameraSource = cameraStreamControl.cameraStream
    }
}
Summary
Summary

Join the MFi program
Summary

Join the MFi program

Take advantage of the ADK
Summary

Join the MFi program
Take advantage of the ADK
Create a great HomeKit accessory
Summary

Join the MFi program
Take advantage of the ADK
Create a great HomeKit accessory
Build a HomeKit app
More Information

https://developer.apple.com/wwdc18/231

HomeKit Lab

Technology Lab 12

Friday 12:00PM