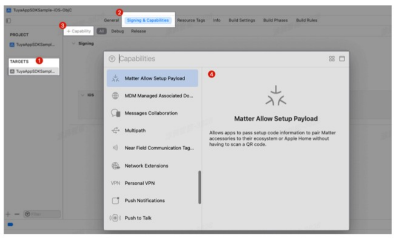


Matter Allow Setup Payload

Allows apps to pass setup code information to pair Matter accessories to their ecosystem or Apple Home without having to scan a QR code.

tuya Prepare for Integration with Matter Device User Guide

Prepare for Integration with Matter Device
Version: 20230719
Online Version

Contents

Prepare for Integration with Matter Device

Before the integration of a Matter device into your project, you must configure the project. For the pairing process implemented by Tuya, Matter devices are classified into Tuya-enabled Matter devices and third-party Matter devices. For a thirdparty Matter device, add an extension target to your Xcode project.

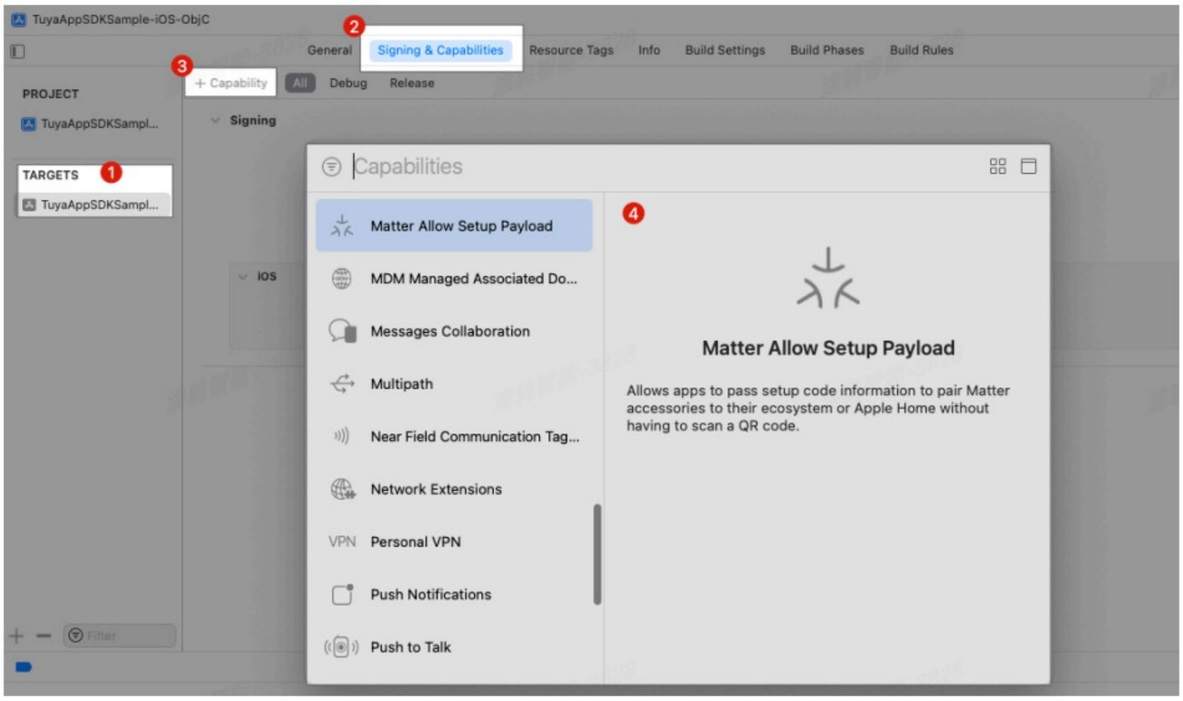
Prerequisites

- Before you start, the steps in Fast Integration with Smart Life App SDK must be finished.
- If you require UI BizBundles, the version of Smart Life App SDK must be the same as that of the UI BizBundles to ensure stable pairing and control of devices.

Configure and integrate with main project

2.1 Configure entitlements

1. Open the project settings in Xcode.
2. Choose the target of the main project> Signing& Capabilities, and click + Capability.

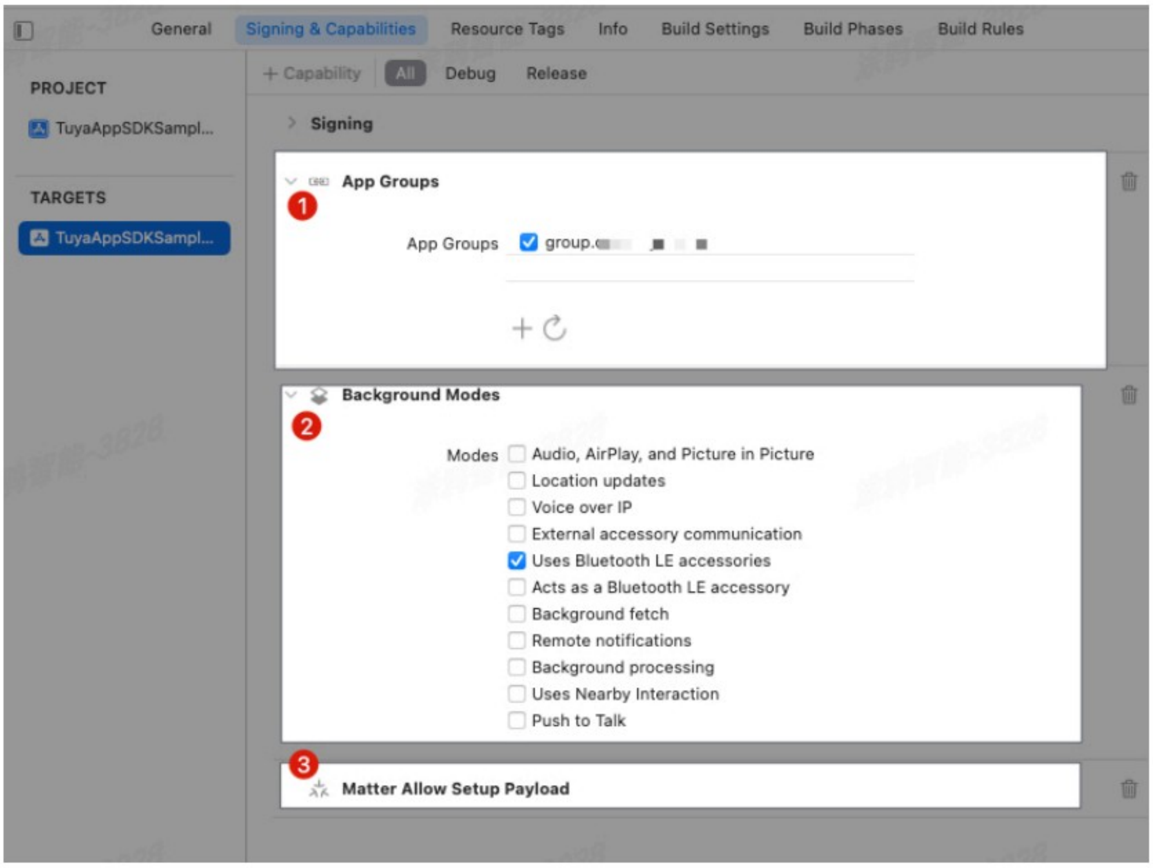


Matter Allow Setup Payload

Allows apps to pass setup code information to pair Matter accessories to their ecosystem or Apple Home without having to scan a QR code.

3. Add the entitlement Matter Allow Setup Payload that is used to handle parsing of Matter QR codes.
4. Add the App Groups entitlement that is used to share data with Matter Extension Target. Set App Group ID to the same value as Matter Extension Target.
5. Add the Background Modes entitlement and select Uses Bluetooth LE accessories that is used to ensure stable communication over Bluetooth.

   The following figure shows these steps.



App Groups

App Groups ✓ group.[redacted]

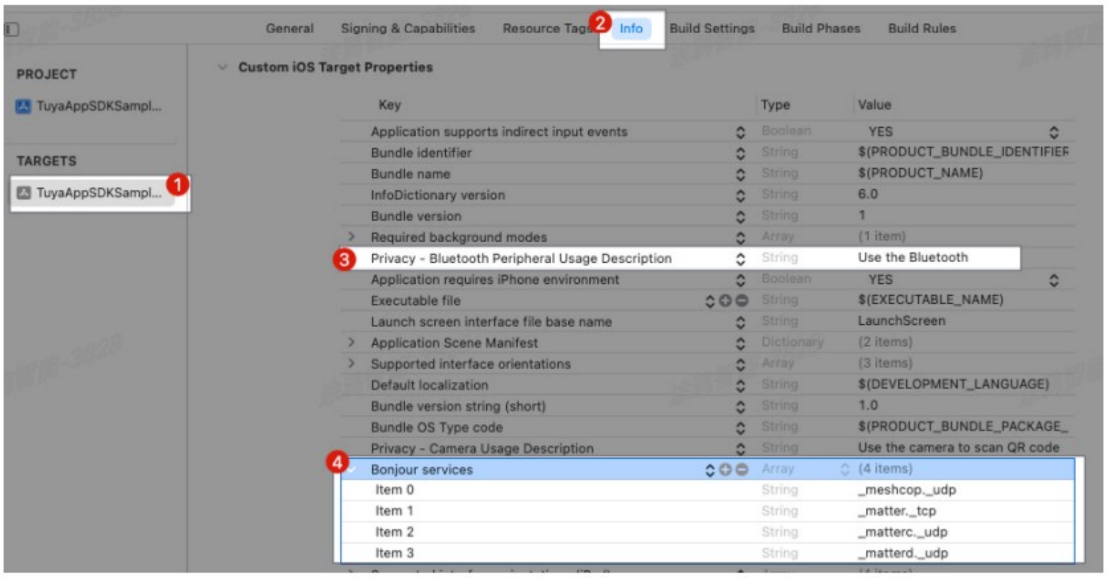
Background Modes

Modes
- [ ] Audio, AirPlay, and Picture in Picture
- [ ] Location updates
- [ ] Voice over IP
- [ ] External accessory communication
- [x] Uses Bluetooth LE accessories
- [ ] Acts as a Bluetooth LE accessory
- [ ] Background fetch
- [ ] Remote notifications
- [ ] Background processing
- [ ] Uses Nearby Interaction
- [ ] Push to Talk

Matter Allow Setup Payload

2.2 Configure Info.plist

1. Open the project settings in Xcode.
2. Choose the target of the main project> Info> Custom iOS Target Prop- erties.



Custom iOS Target Properties

Key	Type	Value
Application supports indirect input events	Boolean	YES
Bundle identifier	String	$(PRODUCT_BUNDLE_IDENTIFIEF
Bundle name	String	$(PRODUCT_NAME)
InfoDictionary version	String	6.0
Bundle version	String	1
Required background modes	Array	(1 item)
Privacy - Bluetooth Peripheral Usage Description	String	Use the Bluetooth
Application requires iPhone environment	Boolean	YES
Executable file	String	$(EXECUTABLE_NAME)
Launch screen interface file base name	String	LaunchScreen
Application Scene Manifest	Dictionary	(2 items)
Supported interface orientations	Array	(3 items)
Default localization	String	$(DEVELOPMENT_LANGUAGE)
Bundle version string (short)	String	1.0
Bundle OS Type code	String	$(PRODUCT_BUNDLE_PACKAGE_
Privacy - Camera Usage Description	String	Use the camera to scan QR code
Bonjour services	Array	(4 items)
Item 0	String	_meshcop._udp
Item 1	String	_matter._tcp
Item 2	String	_matterc._udp
Item 3	String	_matterd._udp

- **Privacy – Bluetooth Peripheral Usage Description**

  Add the entitlement NSBluetoothPeripheralUsageDescription that is used for privacy and permission purposes and to provide a user-facing description of the reason for requesting access to Bluetooth peripherals.
- **Bonjour services**

  Matter has a strong dependency on communication on a LAN. In light of this, you must configure Bonjour services in the file Info.plist of the main project. Example:

```
<key>NSBonjourServices</key>
    <array>
        <string>_meshcop._udp</string>
        <string>_matter._tcp</string>
        <string>_matterc._udp</string>
        <string>_matterd._udp</string>
    </array>
```

2.3 Use CocoaPods for fast integration

1. We recommend that you update CocoaPods to the latest version.
2. Add the following code block to the Podfile:

```
platform :ios, '9.0'
target 'Your_Project_Name' do
    pod "ThingSmartMatterKit"
end
```

3. In the root directory of your project, run pod update.

   For more information about CocoaPods, see CocoaPods Guides.

2.4 Initialize module

Follow the instructions in Fast Integration with Smart Life App SDK for iOS and initialize the SDK. Then, initialize the Matter module.

```
// Initialize the SDK.
[[TuyaSmartSDK sharedInstance] startWithAppKey:<#your_app_key#> secr
etKey:<#your_secret_key#>];
// Initialize the Matter module.
[ThingSmartMatterActivatorConfig setMatterConfigKey:<#YOUR_APP_GROUP
_ID#>];
```

API description

```
+ (void)setMatterConfigKey:(NSString *)configKey;
```

Parameters

Parameter	Description
configKey	The App Group ID of the project.

Example

Objective-C:

```
[ThingSmartMatterActivatorConfig setMatterConfigKey:<#YOUR_APP_GROUP
_ID#>];
```

Swift:

```
ThingSmartMatterActivatorConfig.setMatterKey("YOUR_APP_GROUP  _ID")
```

Configure and integrate with Extension Target

3.1 Things to note

- Xcode 14.1 or later is required.
- iOS 16.1 or later is required.
- Create Matter Extension Target and use the default code file to perform the steps described in the following sections.

- Matter Extension Target supports Swift projects only.

3.2 Configure entitlements

1. Open the project settings in Xcode.
2. Choose the target of the main project> Signing& Capabilities, and click + Capability.
3. Add the App Groups entitlement that is used to share data with the target of the main project. Set App Group ID to the same value as the target of the main project.

3.3 Use CocoaPods for fast integration

1. We recommend that you update CocoaPods to the latest version.
2. Add the following code block to the Podfile:
```
platform :ios, '9.0'
target 'Your_Matter_Extension_Target_Name' do
    pod "ThingSmartMatterExtensionKit"
end
```
3. In the root directory of your project, run pod update.
   For more information about CocoaPods, see **CocoaPods Guides**.

3.4 Initialize module

Open the file RequestHandler.swift in the ExtensionTarget project and rewrite the init method.
```
override init()
```
API description

```
+ (void)setMatterConfigKey:(NSString *)configKey;
```
Parameters

Parameter	Description
configKey	The App Group ID of the project.

Example

Swift:
```
override init() {
  super.init()
  ThingMatterExtensionSupport.shared.setMatterConfigKey(config  Key: <
#YOUR_APP_GROUP_ID#>)
}
```
3.5 Use Thing Smart Matter Extension Kit

After the Matter Extension Target project is generated, the file RequestHandler.swift appears in the Extension project. Use the API methods provided by the system and call ThingSmartMatterExtensionKit as shown in the following example.

1. Import ThingSmartMatterExtensionKit into the project.
```
import ThingSmartMatterExtensionKit
```
2. Make API requests with the methods that are automatically generated by the system, as shown in the following code block.

The callback methods in Request Handler.swift are automatically generated by the system and cannot be modified.

```swift
    override func validateDeviceCredential(_ deviceCredential: MatterAddDeviceExtensionRequestHandler.DeviceCredential) async throws {
        ThingMatterExtensionSupport.shared.validateDeviceCredential(deviceCredential)
    }
    override func selectWi-FiNetwork(from wifiScanResults: [MatterAddDeviceExtensionRequestHandler.Wi-FiScanResult]) async throws -> MatterAddDeviceExtensionRequestHandler.Wi-FiNetworkAssociation {
        // Use this function to select a Wi-Fi network for the device if your ecosystem has special requirements.
        // Or, return `.defaultSystemNetwork` to use the iOS device's current network.
        return ThingMatterExtensionSupport.shared.selectWi-FiNetwork(from: wifiScanResults)
    }
    override func selectThreadNetwork(from threadScanResults: [MatterAddDeviceExtensionRequestHandler.ThreadScanResult]) async throws -> MatterAddDeviceExtensionRequestHandler.ThreadNetworkAssociation {
        // Use this function to select a Thread network for the device if your ecosystem has special requirements.
        // Or, return `.defaultSystemNetwork` to use the default Thread network.
        return ThingMatterExtensionSupport.shared.selectThreadNetwork(from: threadScanResults)
    }
    override func commissionDevice(in home: MatterAddDeviceRequest.Home?, onboardingPayload: String, commissioningID: UUID) async throws {
        // Use this function to commission the device with your Matter stack.
        ThingMatterExtensionSupport.shared.commissionDevice(in: home, onboardingPayload: onboardingPayload, commissioningID: commissioningID)
    }
    override func rooms(in home: MatterAddDeviceRequest.Home?) async -> [MatterAddDeviceRequest.Room] {
        // Use this function to return the rooms your ecosystem manages.
        // If your ecosystem manages multiple homes, ensure you are returning rooms that belong to the provided home.
```

Configure Matter capabilities

Due to the characteristics of a Matter device, all its capabilities are implemented based on Matter fabrics. Before a Matter device can be paired, controlled, or managed, you must make API requests to configure basic information about the Matter device. This configuration must be finished before any Matter services are implemented.
A fabric is a group of networked devices (also known as nodes) that share the same security domain. This enables secure communications among these nodes within the fabric. Nodes in the same fabric share the same Certificate Authority's (CA) top-level certificate (Root of Trust) and within the context of the CA, a unique 64-bit identifier named Fabric ID.

4.1 Sequence diagram

The following figure shows the sequence in which basic Matter information is configured.

```
sequenceDiagram
participant user as User
participant app as App
participant sdk as SDK
user ->> app: Initially load home or switch between homes
app ->> sdk: Load information about home
sdk -->> app: Return information about home
app ->> sdk: Request loading list of devices
sdk -->> app: Return list of devices
rect rgb(206, 235, 252)
  note over app, sdk: Handle Matter capabilities
  app ->> sdk: Prepare information about fabric
  sdk --> app: Fabric information loaded
  app ->> sdk: Prepare information about devices
  sdk -->> app: Device information prepared
end
user ->> app: Use Matter capabilities
```

4.2 Prepare information about fabric

Information about Matter is bound with homes. Therefore, at the end of the operation of switching between homes or initially loading a home, call – loadFabricWithSpaceId to get information about the fabric that is bound with the home.

API description

```
@interface ThingSmartMatterManager : NSObject
+ (instancetype)sharedInstance;
- (void)loadFabricWithSpaceId:(long long)spaceId
                  success:(ThingSuccessHandler)success
                        failure:(ThingFailureError)failure;
@end
```

Parameters

Parameter	Description
spaceId	The value of HomeID for the current home.
success	The success callback.
failure	The failure callback.

Example

Objective-C:

```objc
- (void)loadMatterCurrentHomeFabric:(long long)homeId {
    // Called at the end of the operation of switching between homes or initially loading a home.
    [[ThingSmartMatterManager sharedInstance] loadFabricWithSpaceId:homeId success:^{
        NSLog(@"load fabric success");
    } failure:^(NSError *error) {
        NSLog(@"load fabric fail");
    }];
}
```

Swift:

```swift
func loadMatterCurrentHomeFabric(homeId: Int64) {
    // Called at the end of the operation of switching between homes or initially loading a home.
    ThingSmartMatterManager.sharedInstance().loadFabric(withSpaceId: homeId) {
        print("load fabric success")
    } failure: { error in
        print("load fabric fail")
    }
}
```

4.3 Prepare information about devices

Different from other types of Tuya-enabled devices, certain information about Matter devices must be prepared in advance. For this purpose, at the end of loading home information and fabric information, call –getDevicesFabricNodesWithdevIds :callback: to handle Matter devices for the specified home.

API description

```objc
@interface ThingSmartMatterShareManager : NSObject
+ (instancetype)sharedInstance;
- (void)getDevicesFabricNodesWithdevIds:(NSArray <NSString *>*)devIds callback:(void(^)(NSArray <ThingSmartMatterDeviceNodeModel *>*result))callback;
@end
```

Parameters

Parameter	Description
devIds	The list of device IDs.

Example

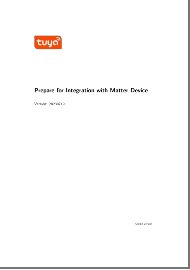
Objective-C:

```objc
- (void)loadMatterDeviceInfo {
    // Called at the end of `- loadFabric` and loading devices in the home.
    [[ThingSmartMatterShareManager sharedInstance] getDevicesFabricNodesWithdevIds:deviceIdList callback:^(NSArray<ThingSmartMatterDeviceNodeModel *> *result) {
        NSLog(@"load matter device node succes");
    }];
}
```

Swift:

```swift
func loadMatterDeviceInfo(){
  // Called at the end of `- loadFabric` and loading devices in the home.
  ThingSmartMatterShareManager.sharedInstance().getDevicesFabricNodesWithdevIds(deviceIdList) { modelList in
      print("load matter device info success")
  }
}
```

Documents / Resources

 Prepare for Integration with Matter Device Version: 20230719 Online Version	tuya Prepare for Integration with Matter Device [pdf] User Guide Prepare for Integration with Matter Device, Integration with Matter Device, Matter Device

References

- Fast Integration with Smart Life App SDK for iOS-IoT App SDK-Tuya Developer
- What is UI BizBundle SDK for iOS?-IoT App SDK-Tuya Developer
- Prepare for Integration with Matter Device-IoT App SDK-Tuya Developer
- CocoaPods Guides - Home
- User Manual

Manuals+, Privacy Policy

This website is an independent publication and is neither affiliated with nor endorsed by any of the trademark owners. The "Bluetooth®" word mark and logos are registered trademarks owned by Bluetooth SIG, Inc. The "Wi-Fi®" word mark and logos are registered trademarks owned by the Wi-Fi Alliance. Any use of these marks on this website does not imply any affiliation with or endorsement.