tuya Matter Device Smart App SDK Instructions

Contents

tuya Matter Device Smart App SDK

Discover device

Matter devices can be discovered in the following two ways:

- Scan QR code or enter setup code A standard pairing method for Matter devices, including Tuya-enabled Matter devices and third-party Matter devices.
- Pair by auto discovery Tuya's proprietary method, currently exclusive to Tuya-enabled Matter devices.

The SDK provides two ways to discover Matter devices:

Method 1: Scan QR code or enter setup code

This is the standard method of pairing Matter devices. You can pass the content of the Matter pairing QR code scanned by users or the Matter setup code entered by users to the SDK for parsing. Then, return the parsed object ThingSmartMatterSetupPayload and proceed to the steps as illustrated in the pairing flowchart.

API description

```
- (nullable ThingSmartMatterSetupPayload *)parseSetupCode:(NSString
*)matterCode;
```

Parameters

Parameter	Description
matterCode	The QR code or setup code on the device outer packaging.

Example

Objective-C:

```
- (void)checkCode{
    ThingSmartMatterSetupPayload *payload = [[ThingSmartMatterActiva
tor sharedInstance]parseSetupCode:@""];
    if(payload){
        //Use the payload to build `ThingSmartConnectDeviceBuilder`
object
    }else{
        //Code is invalid;
    }
}
```

Swift:

```
func checkCode() {
    var payload = ThingSmartMatterActivator.sharedInstance().parseSe
tupCode("code")
    if payload != nil{
         // Use the payload to build `ThingSmartConnectDeviceBuilder
` object
    }else{
        //Code is invalid;
    }
}
```

Method 2: Pair by auto-discovery

Before pairing by auto-discovery, the app must be granted access to Bluetooth, and the mobile phone is connected to a Wi-Fi network.

After users start this pairing method, nearby Matter devices that support this method can be automatically found.

API description

```
- (void)startDiscovery;
```

Example

Objective-C:

```
[[ThingSmartMatterDiscoveryActivator sharedInstance] startDiscovery]
;
```

Swift:

```
ThingSmartMatterDiscoveryActivator.sharedInstance().startDis  covery()
```

Device scanning callback

The callback to invoked when a nearby Matter device that supports auto-discovery is automatically found. The callback is invoked once each time a Matter device is discovered.

API description

```
- (void)discoveryedDevice:(ThingSmartMatterDiscoveryModel *)model;
```

Parameters

Parameter	Description
model	The data model of pairing by auto discovery.

Example

Objective-C:

```
- (void)discoveryedDevice:(ThingSmartMatterDiscoveryModel *)model{
    NSLog(@"Discovery a Matter Device!");
}
```

Swift:

```
func discoveryedDevice(_ model: ThingSmartMatterDiscoveryModel) {
    print("Discovery a Matter Device!");
}
```

Define data model of pairing by auto discovery

Generates a data model and returns it to the business layer when a nearby Matter device that supports auto discovery is automatically found.

Parameters

Parameter	Description
payload	A section of the setup code generated by the program, without the pairing key.
iconUrl	The URL of the device icon.
deviceName	The name of the device.
productId	The product ID of the device.
isThingDevice	Indicates whether this is a Tuya-enabled device.
deviceType	Device types

Stop auto-discovery

Stops auto-discovery when the auto-discovery page is closed to avoid unnecessary resource usage.

API description

```
- (void)stopDiscovery;
```

Example

Objective-C:

```
[[ThingSmartMatterDiscoveryActivator sharedInstance] stopDiscovery];
```

Swift:

```
ThingSmartMatterDiscoveryActivator.sharedInstance().stopDisc  overy()
```

Start pairing process

Continues the steps as illustrated in the pairing flowchart, based on the payload property object included in the discovered model object ThingSmartMatterDiscoveryModel. The design of the Matter protocol determines that only one device can be paired at a time during the pairing process. Therefore, in your implementation of pairing, finish pairing a device before starting paring the next device.

Build pairing parameters

Get SetupPayload

For more information about how to get the object of ThingSmartMatterSetupPayload, see Discover device.

Get a token

Before the wired pairing process, the SDK must get a pairing token from the cloud in the networked state. The token is valid for 10 minutes and expires immediately after the device is paired. A new token must be obtained if the device needs to be paired again.

API description

```
- (void)getTokenWithHomeId:(long long)homeId
                   success:(ThingSuccessString)success
                   failure:(ThingFailureError)failure;
```

Parameters

Parameter	Description
homeId	The ID of the home with which the device is bound.
success	The success callback. A pairing token is returned.
failure	The failure callback. An error message is returned.

Example

Objective-C:

```objc
- (void)getToken {
    [[ThingSmartActivator new] getTokenWithHomeId:homeId success:^(N
SString *token) {
            NSLog(@"getToken success: %@", token);
    } failure:^(NSError *error) {
            NSLog(@"getToken failure: %@", error.localizedDescriptio
n);
    }];
}
```

Swift:

```swift
func getToken() {
    ThingSmartActivator().getTokenWithHomeId(homeId, success: { (tok
en) in
        print("getToken success: \(token)")
    }, failure: { (error) in
        if let e = error {
            print("getToken failure: \(e)")
        }
    })
}
```

Assemble pairing parameters into object

Provides required parameters for pairing. This is the last preparation step before pairing can be started.

API description

```objc
- (instancetype)initWithPayload:(ThingSmartMatterSetupPayload *)payl
oad spaceId:(long long)spaceId token:(NSString *)token;
```

Parameters

Parameter	Description
payload	The object of `ThingSmartMatterSetupPayload` that is returned after users **scan a QR code, enter a setup code**, or **perform auto discovery**.
spaceId	The home ID. This is the `homeId` parameter of the current home in most cases.
token	The pairing token that is returned by `getTokenWithHomeId`.

Example

Objective-C:

```
- (ThingSmartConnectDeviceBuilder *)connectBuilder{
    ThingSmartConnectDeviceBuilder *builder = [[ThingSmartConnectDev
iceBuilder alloc]initWithPayload:payload spaceId:homeId token:token]
;
    return builder;
}
```

Swift:

```
func connectBuilder() -> ThingSmartConnectDeviceBuilder?{
    let builder = ThingSmartConnectDeviceBuilder(payload: payload, s
paceId: homeId, token: token)
    return builder
}
```

Pairing process

Connect to device

After the Matter device is connected, the PASE (Passcode-Authenticated Session Establishment) session is established, and the PASE session success callback is invoked.

API description

```
- (void)connectDeviceWithConnectDeviceBuilder:(ThingSmartConne  ctDevi
ceBuilder *)builder timeout:(NSTimeInterval)timeout;
```

Parameters

Parameter	Description
builder	The model object of `ThingSmartConnectDeviceBuilder` into which required pairing parameters are assembled.
timeout	The timeout period.

Example

Objective-C:

```
- (void)startActivator{
    // ThingSmartConnectDeviceBuilder *builder = [self connectBuilde
r];
    [[ThingSmartMatterActivator sharedInstance]connectDeviceWithConn
ectDeviceBuilder:builder timeout:300];
```

Swift:

```
func startActivator(){
    // guard let builder = connectBuilder() else {return}
    ThingSmartMatterActivator.sharedInstance().connectDevice(wit  h: b
uilder, timeout: 300)
}
```

Device connection callback

The callback to invoked after an attempt to connect to a device by connect. The callback is intended to notify the business layer of the pairing mode and device type.

API description

```
- (void)matterDeviceDiscoveryed:(BOOL)isThingDevice deviceType:(Thin
gSmartMatterDeviceType)deviceType;
```

Parameters

Parameter	Description
isThingDevice	Indicates whether this is a Tuya-enabled device.
deviceType	The type of device.

Example

Objective-C:

```
- (void)matterDeviceDiscoveryed:(BOOL)isThingDevice deviceType:(Thin
gSmartMatterDeviceType)deviceType{
    NSLog(@"Discoveryed Device");
}
```

Swift:

```
func matterDeviceDiscoveryed(_ isThingDevice: Bool, deviceType: Thin
gSmartMatterDeviceType) {
    print("Discoveryed Device")
}
```

Matter pairing option callback

The callback to invoked when the optimal pairing option is selected based on the device advertising data. Then, the SDK returns the callback to the business layer to implement the process of displaying the respective page. The following types of URLs are supported:

- Pairing option implemented by Tuya (ThingMatterRoutingTypeThing)
- Sharing and pairing option (ThingMatterRoutingTypeSupport)
- MatterSupport option for third-party Matter devices (ThingMatterRoutingTypeShare)

::: important

To enable pairing with the MatterSupport option, follow the instructions in Prepare for Integration with Matter Device and configure Matter Extension Target. Otherwise, pairing will fail. :::

API description

```
- (void)matterRoutingComplete:(ThingMatterRoutingType)routingT  ype
```

Parameters

Parameter	Description
routingType	The pairing option.

Example

Objective-C:

```
- (void)matterRoutingComplete:(ThingMatterRoutingType)routingT  ype{
    NSLog(@"Routing Complete");
}
```

Swift:

```
func matterRoutingComplete(_ routingType: ThingMatterRoutingType) {
    print("Routing Complete");
}
```

PASE session establishment callback

After the Matter device is connected, the PASE session is established, and the PASE session success callback is invoked.

API description

```
- (void)matterCommissioningSessionEstablishmentComplete:(ThingSmartMatterDeviceModel *)deviceModel;
```

Parameters

Parameter	Description
deviceModel	The Matter device model that returns certain data during the PASE session.

Example

Objective-C:

```
- (void)matterCommissioningSessionEstablishmentComplete:(ThingSmartMatterDeviceModel *)deviceModel {
    NSLog(@"Establishment Complete!");
}
```

Swift:

```
func matterCommissioningSessionEstablishmentComplete(_ deviceModel: ThingSmartMatterDeviceModel) {
    print("Establishment Complete!");
}
```

Assemble parameters for commissioning Tuya-enabled combo device

To establish a Certificate Authenticated Session Establishment (CASE) session with a Tuya-enabled combo device, you must provide and assemble the required parameters. Then, the device can be commissioned. Only Tuya-enabled Matter devices require you to assemble the parameters.

API description

```
- (instancetype)initWithSSid:(NSString *)ssid password:(NSString *)password;
```

Parameters

Parameter	Description
ssid	The name of the target Wi-Fi network.
password	The password of the target Wi-Fi network.

Example

Objective-C:

```
- (ThingSmartMatterCommissionModel *)commissionBuilder{
    ThingSmartMatterCommissionModel *builder = [[ThingSmartMatterCom
missionModel alloc]initWithSSid:@"ssid" password:@"password"];
    return builder;
}
```

Swift:

```
func commissionBuilder() -> ThingSmartMatterCommissionModel{
    let commissionBuilder = ThingSmartMatterCommissionModel(sSid: "s
sid", password: "password")
    return commissionBuilder
}
```

Assemble parameters for commissioning Thread sub-device

Pairing the sub-device relies on the OTBR network provided by the Matter gateway. Therefore, you must set the gateway ID before a PASE session is established. In a PASE session, the SDK scans for available Thread networks nearby and selects the optimal gateway ID with the strongest signal strength. In the callback –(void)matterCommissioningSessionEstablishmentComplete:(ThingSmartMatterDeviceModel *)deviceModel;, ThingSmartMatterDeviceModel provides a list of available gateway IDs indicated by gatewayId. If this parameter is empty or users want to specify a gateway, use another gateway ID for the current home. Only Tuya-enabled Matter devices require you to assemble the parameters.

API description

```
- (instancetype)initWithGatewayId:(NSString *)gwId;
```

Parameters

Parameters

Parameter	Description
gwId	The device ID of the gateway.

Example

Objective-C:

```
- (ThingSmartMatterCommissionModel *)commissionBuilder{
    ThingSmartMatterCommissionModel *builder = [[ThingSmartMatterCom
missionModel alloc] initWithGatewayId:@"gwId"];
    return builder;
}
```

Swift:

```
func commissionBuilder() -> ThingSmartMatterCommissionModel{
    let commissionBuilder = ThingSmartMatterCommissionModel(gatewayId:"gwId")
    return commissionBuilder
}
```

Commission Matter device

After a PASE session, certain key information such as the type of device can be obtained. When required parameters have been assembled for a CASE session, to activate the device in the cloud, make this API request to establish a CASE session. Only Tuya-enabled Matter devices require you to make this API request. Third-party devices do not support this API request.

API description

```
- (void)commissionDevice:(nonnull ThingSmartMatterDeviceModel *)deviceModel commissionModel:(ThingSmartMatterCommissionModel *) commissionModel;
```

Parameters

Parameter	Description
deviceModel	The Matter device model returned in the callback `matterCommissioningSessionEstablishmentComplete` to invoke after a PASE session is finished.
commissionModel	The model returned after you assemble CASE session parameters.

Example

Objective-C:

```
- (void)continueActivator{
    ThingSmartMatterCommissionModel *builder = [self commissionBuilder];
    [[ThingSmartMatterActivator sharedInstance]commissionDevice:self.matterDeviceModel commissionModel:builder];
}
```

Swift:

```
func continueActivator(){
    let builder = commissionBuilder()
    ThingSmartMatterActivator.sharedInstance().commissionDevice(  self.matterDeviceModel, commissionModel: builder)
}
```

Device attestation callback

The callback to invoke when an attempt is made to pair a device that is not Mattercertified. After this callback is invoked, the pairing process will be suspended until you choose to continue or give up pairing.

API description

```
- (void)matterDeviceAttestation:(void *)device error:(NSError * _Non
null)error;
```

Parameters

Parameter	Description
device	The pointer to the device object address in the pairing option.
error	The error message that is returned when device attestation failed.

Example

Objective-C:

Example

Objective-C:

```
- (void)matterDeviceAttestation:(void *)device error:(NSError *)erro
r{
    NSLog(@"Device Attestation!");
}
```

Swift:

```
func matterDeviceAttestation(_ device: UnsafeMutableRawPointer, erro
r: Error) {
    print("Device Attestation!")
}
```

Trust unattested device and continue pairing

Continues pairing if the device certificate is regarded to be trustable.

API description

```
- (void)continueCommissioningDevice:(void *)device
            ignoreAttestationFailure:(BOOL)ignoreAttestationFailure
                               error:(NSError * __autoreleasing *)err
or;
```

Parameters

Parameter	Description
device	The pointer to the device object address included in the `Attestation` callback method. The callback parameters must be returned as **pass-through parameters**, without additional operations.
ignoreAttestationFailure	Specifies whether to ignore the attestation failure.
error	The error message that might be returned in this call.

Example

Objective-C:

```objc
- (void)matterDeviceAttestation:(void *)device error:(NSError *)erro
r{
    UIAlertController * alertController = [UIAlertController
        alertControllerWithTitle:@"Device Attestation"
                         message:@"Device Attestation failed for dev
ice under commissioning. Do you wish to continue pairing?"
                  preferredStyle:UIAlertControllerStyleAlert];
    [alertController addAction:[UIAlertAction actionWithTitle:@"No"
                                                        style:UIAler
tActionStyleDefault
                                                      handler:^(UIAl
ertAction * action) {
                                                          NSError *
err;
        [[ThingSmartMatterActivator sharedInstance]continueCommissio
ningDevice:device ignoreAttestationFailure:NO error:&err];
                                                      }]];
    [alertController addAction:[UIAlertAction actionWithTitle:@"Cont
inue"
                                                        style:UIAler
tActionStyleDefault
                                                      handler:^(UIAl
ertAction * action) {
                                                          NSError *
err;
        [[ThingSmartMatterActivator sharedInstance]continueCommissio
ningDevice:device ignoreAttestationFailure:YES error:&err];
                                                      }]];
    [self presentViewController:alertController animated:YES complet
ion:nil];
}
```

Swift:

```swift
func matterDeviceAttestation(_ device: UnsafeMutableRawPointer, erro
r: Error) {
    let alertController = UIAlertController(title: "Device Attestati
on", message: "Device Attestation failed for device under commission
ing. Do you wish to continue pairing?", preferredStyle: .alert)
    alertController.addAction(UIAlertAction(title: "NO", style: .def
ault, handler: { _ in
        ThingSmartMatterActivator.sharedInstance().continueCommissio
ningDevice(device, ignoreAttestationFailure: false, error: nil)
    }))
    alertController.addAction(UIAlertAction(title: "Continue", style
: .default, handler: { _ in
        ThingSmartMatterActivator.sharedInstance().continueCommissio
ningDevice(device, ignoreAttestationFailure: true, error: nil)
    }))
    self.present(alertController, animated: true)
}
```

Pairing success callback

The callback to invoke when a Matter device is paired and activated in the cloud.

API description

```
- (void)matterDeviceActivatorSuccess:(ThingSmartMatterDeviceMo  del *)
matterDevice;
```

Parameters

Parameter	Description
matterDevice	The complete data model of the Matter device.

Example

Objective-C:

```
- (void)matterDeviceActivatorSuccess:(ThingSmartMatterDeviceMo  del *)
matterDevice{
    NSLog(@"Activator Success!");
}
```

Swift:

```
func matterDeviceActivatorSuccess(_ matterDevice: ThingSmartMatterDe
viceModel) {
    print("Activator Success!")
}
```

Pairing failure callback

The callback to invoke when a Matter device failed to be paired.

API description

```
- (void)matterDevice:(ThingSmartMatterSetupPayload *)payload activat
orFailed:(NSError *)error;
```

Parameters

Parameter	Description
payload	The data model of the setup code for failed pairing of the Matter device.
error	The error message. For more information about the description of a specific error code, see the definition in `ThingSmartMatterKitErrors`.

Example

Objective-C:

```
- (void)matterDevice:(ThingSmartMatterSetupPayload *)payload activat
orFailed:(NSError *)error{
    NSLog(@"Activator Failed");
}
```

Swift:

```
func matterDevice(_ payload: ThingSmartMatterSetupPayload, activator
Failed error: Error) {
    print("Activator Failed")
}
```

MatterSupport pairing success callback

The MatterSupport process is designed for a third-party Wi-Fi or Thread device. If this process is used for pairing, MatterSupport is regarded as a part of the whole pairing process. When MatterSupport is finished, invoke this callback to continue the pairing process implemented by Tuya.

API description

```
- (void)matterSupportComplete:(NSString *)gatewayName;
```

Parameters

Parameter	Description
gatewayName	The Tuya-enabled gateway ID that is assigned to the OTBR network used by `MatterSupport`. An empty value indicates that HomePod or Apple TV is used by `MatterSupport` to implement pairing.

Example

Objective-C:

```
- (void)matterSupportComplete:(NSString *)gatewayName{
    NSLog(@"Matte Support Completed");
}
```

Swift:

```
func matterSupportComplete(_ gatewayName: String) {
    print("Matte Support Completed")
}
```

Error codes

For more information about the description of a specific error code, see the declaration and definition in ThingSmartMatterKitErrors.h.

Documents / Resources

	tuya Matter Device Smart App SDK [pdf] Instructions Matter Device Smart App SDK, Device Smart App SDK, Smart App SDK, App SDK, SDK

References

- User Manual