AP Mode

Version: 20240520

Online Version

Contents

This topic describes the **access point (AP)** or hotspot mode to pair devices. It is a connection capability for pairing over Wi-Fi. After a mobile phone is connected to the Wi-Fi hotspot of a target device, the device is paired and communicates with the mobile phone over Wi-Fi. With a high success rate and good reliability, this mode adapts to 2.4 GHz and 5 GHz dual-band routers. However, users need to manually switch between the Wi-Fi bands connected to the mobile phone.

1 Pairing methods

Smart Life App SDK supports the following AP pairing processes:

- Legacy AP pairing: Get the pairing token from the cloud and implement a connection between the device and the cloud.

  Prerequisite: suitable for all devices.

- New AP pairing: Users connect the device with the app first. Then, access the device on the app and scan for available Wi-Fi networks. Only compatible Wi-Fi networks are discovered. This way, the unsupported 5 GHz Wi-Fi networks are automatically filtered out. Users do not need to manually handle them and can enjoy a higher success rate of pairing.

  Prerequisite:

  - The TuyaOS version of the firmware built into the target device must be v3.6.1 or later.
  - Log in to the Apple Developer platform and enable **Hotspot**.
  - Integrate the `ThingSmartHotspotCredentialKit` library into your project.

2 Legacy pairing process

1. **Before pairing**
   - Guide the user to reset the device to Wi-Fi AP mode, typically identified by a slow-blinking Wi-Fi indicator.
   - Guide the user to connect their phone to a Wi-Fi network, usually a 2.4 GHz Wi-Fi network.
2. **Get pairing token and Wi-Fi network credentials**
   - The app gets the pairing token by calling the method in the SDK.
   - The app gets the Wi-Fi network credentials (SSID and password) input by the user. The SSID can also be obtained through an API call.
3. **Connect to AP**
   Guide the user to connect their phone to the AP emitted by the device.
4. **Start pairing**
   The app calls the pairing method in the SDK to set the Wi-Fi network credentials (SSID and password) and the pairing token.
5. **Finish pairing**
   The device automatically turns off the AP. The app receives a callback from the SDK and finishes the pairing process.

2.1 Get the token

Before the AP 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 generated if the device needs to be paired again.

API description

```
1 - (void)getTokenWithHomeId:(long long)homeId
2                    success:(ThingSuccessString)success
3                    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:

```
- (void)getToken {
        ThingSmartActivator *apActivator = [[ThingSmartActivator all
oc] init];
    [apActivator getTokenWithHomeId:homeId success:^(NSString *token
) {
        NSLog(@"getToken success: %@", token);
        // TODO: startConfigWiFi
    } failure:^(NSError *error) {
        NSLog(@"getToken failure: %@", error.localizedDescription);
    }];
}
```

Swift:

```
func getToken() {
    let ezActivator = ThingSmartActivator()
    ezActivator.getTokenWithHomeId(homeId, success: { token in
        print("getToken success: \(token)")
        // TODO: startConfigWiFi
    }, failure: { error in
        print("getToken failure: \(error.localizedDescription)")
    })
}
```

2.2 Callback of pairing delegate

```
- (void)activator:(ThingSmartActivator *)activator didReceiveDevice:
(ThingSmartDeviceModel *)deviceModel error:(NSError *)error;
```

Parameters

Parameter	Description
activator	The instance of the `ThingSmartActivator` object for pairing.
deviceModel	The paired device model that is returned if the request is successful. `nil` is returned if the request failed.
error	The error message that is returned if the request failed. `nil` is returned if the request is successful.

2.3 Start pairing

API description

```
- (void)startConfigWiFi:(ThingActivatorMode)mode
                   ssid:(NSString *)ssid
               password:(NSString *)password
                  token:(NSString *)token
                timeout:(NSTimeInterval)timeout;
```

Parameters

Parameter	Description
mode	The pairing mode.
ssid	The name of the target Wi-Fi network.
password	The password of the target Wi-Fi network.
token	The pairing token.
timeout	The timeout period.

Example

Objective-C:

```
- (void)startConfigWiFi:(NSString *)ssid password:(NSString *)passwo
rd token:(NSString *)token {
    // Implements the delegate method of `ThingSmartActivator`.
    self.apActivator.delegate = self;
    // Starts AP pairing, in which `mode` is set to `ThingActivatorM
odeEZ`.
    [self.apActivator startConfigWiFi:ThingActivatorModeAP ssid:ssid
 password:password token:token timeout:100];
}
- (ThingSmartActivator *)apActivator {
    if (!_apActivator) {
        _apActivator = [[ThingSmartActivator alloc] init];
    }
    return _apActivator;
}
#pragma mark - ThingSmartActivatorDelegate
- (void)activator:(ThingSmartActivator *)activator didReceiveDevice:
(ThingSmartDeviceModel *)deviceModel error:(NSError *)error {
    if (!error && deviceModel) {
        // The device is paired.
    }
    if (error) {
        // Failed to pair the device.
    }
}
// Added to v4.0.0: The callback for the intermediate process is tri
ggered in the task of pairing a certain security device.
- (void)activator:(ThingSmartActivator *)activator didPassWIFIToSecu
rityLevelDeviceWithUUID:(NSString *)uuid {
        // Then, you can guide users to connect to the specified Wi-
Fi network.
    // This way, the mobile is connected to the Wi-Fi network with t
he same SSID as specified by `- startConfigWiFi:password:token:`.
      // You can get the same SSID and call `- continueConfigSecurit
yLevelDevice` to continue with the pairing task.
//    UIAlertController *vc = [UIAlertController alertControllerWith
Title:@"SecurityLevelDevice" message:@"continue pair? (Please check
you phone connected the same Wi-Fi as you Inputed)" preferredStyle:U
IAlertControllerStyleAlert];
//    [vc addAction:[UIAlertAction actionWithTitle:@"cancel" style:U
IAlertActionStyleCancel handler:nil]];
//    [vc addAction:[UIAlertAction actionWithTitle:@"continue" style
:UIAlertActionStyleDestructive handler:^(UIAlertAction * _Nonnull ac
tion) {
//
//        NSString *wifi = [self getCurrentWiFi];
//        if ([wifi isEqualToString:self.inputSSID]) {
//            [self.apActivator continueConfigSecurityLevelDevice];
//        }
//    }]];
//    [self presentViewController:vc animated:YES completion:nil];
```

```
}
```

Swift:

```
func startConfigWiFi(withSsid ssid: String, password: String, token:
 String) {
    // Implements the delegate method of `ThingSmartActivator`.
    apActivator.delegate = self
    // Starts pairing.
    apActivator.startConfigWiFi(ThingActivatorModeAP, ssid: ssid, pa
ssword: password, token: token, timeout: 100)
}
lazy var apActivator: ThingSmartActivator = {
    let activator = ThingSmartActivator()
    return activator
}()
#pragma mark - ThingSmartActivatorDelegate
func activator(_ activator: ThingSmartActivator!, didReceiveDevice d
eviceModel: ThingSmartDeviceModel!, error: Error!) {
    if deviceModel != nil && error == nil {
        // The device is paired.
    }
    if let e = error {
        // Failed to pair the device.
        print("\(e)")
    }
}
// Added to v4.0.0: The callback for the intermediate process is tri
ggered in the task of pairing a certain security device.
func activator(_ activator: ThingSmartActivator!, didPassWIFIToSecur
ityLevelDeviceWithUUID uuid: String!) {
        // Then, you can guide users to connect to the specified Wi-
Fi network.
    // This way, the mobile is connected to the Wi-Fi network with t
he same SSID as specified by `- startConfigWiFi:password:token:`.
      // You can get the same SSID and call `- continueConfigSecurit
yLevelDevice` to continue with the pairing task.
//        let alert = UIAlertController(title: "SecurityLevelDevice"
, message: "continue pair? (Please check you phone connected the sam
e Wi-Fi as you Inputed)", preferredStyle: .alert);
//        alert.addAction(UIAlertAction(title: "cancel", style: .can
cel))
//        alert.addAction(UIAlertAction(title: "continue", style: .d
estructive, handler: { _ in
//            let wifi = getCurrentWiFi()
//            if wifi == inputSSID {
//                apActivator.continueConfigSecurityLevelDevice()
//            }
//        }))
//        present(alert, animated: true)
}
```

The AP mode is similar to the EZ mode, except that the first parameter of `[self.apActivator startConfigWiFi:ssid:password:token:timeout:]` is set to `ThingActivatorModeAP`

for the AP mode.

`ssid` and `password` respectively specify the hotspot name and password of the router rather than that of the device.

2.4 Stop pairing

After the pairing process is started, the app continuously broadcasts the pairing data until a device is paired or the process times out. To allow users to cancel or complete pairing during the process, you must call the API method `[ThingSmartActivator stopConfigWiFi]` to stop pairing.

API description

```
- (void)stopConfigWiFi;
```

Example

Objective-C:

```
- (void)stopConfigWifi {
    self.apActivator.delegate = nil;
    [self.apActivator stopConfigWiFi];
}
```

Swift:

```
func stopConfigWifi() {
    apActivator.delegate = nil
    apActivator.stopConfigWiFi()
}
```

3 New pairing process

To improve the user experience and success rate of pairing, Tuya provides a new pairing process that is available to the new device firmware version only.

1. **Before pairing**
   Guide the user to reset the device to Wi-Fi AP mode, typically identified by a slow-blinking Wi-Fi indicator.
2. **Get pairing token and security configuration**
   The app gets the pairing token and security configuration by calling the method in the SDK.
3. **Connect to AP**
   Guide the user to connect their phone to the AP emitted by the device.
4. **Get the list of discovered Wi-Fi networks**
   The app gets the list of available Wi-Fi networks by calling the method in the SDK.
5. **Display available Wi-Fi networks and get Wi-Fi network credentials**
   The app displays the list of available Wi-Fi networks and guides the user to select one and enter the password.
6. **Start pairing**
   The app calls the pairing method in the SDK to set the Wi-Fi network credentials, pairing token, and security configuration.
7. **Restart pairing (if error occurs)**
   When the user enters an incorrect password, the app guides the user to enter the password again to restart the pairing process.
8. **Finish pairing**
   The device automatically turns off the AP. The app receives a callback from the SDK and finishes the pairing process.

3.1 Get device security configurations

API description

```
- (void)getDeviceSecurityConfigs:(ThingSuccessDict)success
                         failure:(ThingFailureError)failure;
```

Example

Objective-C:

```
self.activator = [[ThingSmartActivator alloc] init];
self.activator.delegate = self;
[self.activator getDeviceSecurityConfigs:^(NSDictionary *dict) {
  // Security configurations are obtained.
} failure:^(NSError *error) {
  // Failed to get security configurations.
}];
```

Swift:

```
activator.getDeviceSecurityConfigs { res in
  // Security configurations are obtained.
} failure: { error in
  // Failed to get security configurations.
}
```

3.2 Query Wi-Fi networks discovered by device

Before this call, the SDK checks the device status first. If the device has a status exception, for example, an unknown state, the system will not query the Wi-Fi networks discovered by the device.

API description

```
- (void)connectDeviceAndQueryWifiListWithTimeout:(NSTimeInterv  al)tim
eout;
```

Parameters

Parameter	Description
timeout	This parameter is optional. If the value is larger than 0, the countdown for timeout is enabled. Unit: seconds.

Example

Objective-C:

```
[self.activator connectDeviceAndQueryWifiListWithTimeout:120];
```

Swift:

```
activator.connectDeviceAndQueryWifiList(withTimeout: 120)
```

3.3 Start pairing

You can display the returned list of Wi-Fi networks for users to choose from.

API description

```
- (void)resumeAPPlusWithSSID:(NSString *)ssid
                    password:(NSString *)password
                       token:(NSString *)token
                     timeout:(NSTimeInterval)timeout;
```

Parameters

Parameter	Description
ssid	The name of the Wi-Fi network to which a paired device is connected.
password	The password of the Wi-Fi network to which a paired device is connected.
token	The pairing token.
timeout	The timeout value of a pairing task. This parameter is required. Unit: seconds.

Example

Objective-C:

```
[self.activator resumeAPPlusWithSSID:@"SSID" password:@"password" to
ken:@"token" timeout:120];
```

Swift:

```
activator.resumeAPPlus(withSSID: "SSID", password: "password", token
: "token", timeout: 120)
```

3.4 Restart pairing

Asks users to provide the correct Wi-Fi name and password, and restarts pairing if the entered password is incorrect.

During the pairing process, the SDK automatically connects to the device within a certain period. After the device is connected, the SDK gets and reports the device status. This feature is supported by the latest firmware version only.

API description

```
- (int)resumeConfigWiFi:(ThingSmartPairingResumeConfigWiFiPara  m*)par
am error:(NSError**)error;
```

Parameters

`ThingSmartPairingResumeConfigWiFiParam`

Parameter	Description
ThingActivatorMode	The pairing mode. Set the value to `ThingActivatorModeAPPlus`.
ssid	The SSID of the router.
password	The password of the router.

Example

Objective-C:

```
ThingSmartPairingResumeConfigWiFiParam *param = [ThingSmartPairingRe
sumeConfigWiFiParam new];
param.ssid = activatorParam.apActivatorParams.ssid;
param.password = activatorParam.apActivatorParams.pwd;
param.mode = ThingActivatorModeAPPlus;
[self.activator resumeConfigWiFi:param error:nil];
```

Swift:

```
let param = ThingSmartPairingResumeConfigWiFiParam()
param.mode = .apPlus
param.ssid = "ssid"
param.password = "password"
activator.resumeConfigWiFi(param, error: nil)
```

3.5 Stop pairing

You need to manually call this API method when the pairing task is stopped or resources are destroyed.

Example

Objective-C:

```
[self.activator stopConfigWiFi];
```

Swift:

```
activator.stopConfigWiFi()
```

3.6 Error codes

The following error codes occur while processing device connection. The error messages can be obtained from the device.

Error code	Description
207201	Failed to create a connection channel with the device.
207206	An unknown state is returned during the query of device status.
207207	The device does not support the feature of getting Wi-Fi networks discovered.
207209	The pairing information received by the device is incorrect.
207210	The device failed to find a router after receiving the pairing information.
207211	The password in the pairing information received by the device is incorrect.
207212	The device failed to connect to a router after receiving the pairing information.
207213	Failed to get a Dynamic Host Configuration Protocol (DHCP)-assigned IP address after receiving the pairing information.
207214	An error has occurred while activating the device in the cloud.
207215	Failed to connect the device to the cloud.
207216	The request to activate the device failed.
207218	The request to activate the device in the cloud failed.
207219	Failed to connect to `iot-dns` in the cloud during device activation.
207220	The pairing task timed out.

Error code	Description
207222	Failed to get the Wi-Fi networks of the device.