Home Information Management

Version: 20250827

Online Version

Contents

After login into the app, call `ThingSmartHomeManager` to get a list of homes. Then, initialize an object of the `ThingSmartHome` class to get the details of a specific home. This enables device control for the home.

1. Functional description

Class name (protocol name)	Description
ThingSmartHomeManager	Query a list of homes, sort homes, and add homes.
ThingSmartHomeManagerDelegate	The callback that is executed when MQTT connections are created or homes are added and removed.

`ThingSmartHome` must be initialized with the correct value of `homeId` to implement information management for a single home. An incorrect device ID might cause failed initialization. In this case, `nil` will be returned. Information management for a single home includes the capabilities to manage devices, groups, members, rooms, and other resources of the home.

Class name (protocol name)	Description
ThingSmartHome	The home management class.
ThingSmartHomeDelegate	The callback that is executed when home information is changed.

Before retrieving all devices and groups in a home, it is necessary to initialize the `home` object and query the details of the home using the method `getHomeDataWithSuccess:failure:` . After this step, the `home` instance object will have data in properties such as `homeModel` , `roomList` , `deviceList` , `groupList` , `sharedDeviceList` , and `sharedGroupList` .

2. Create a home

API description

```
- (void)addHomeWithName:(NSString *)homeName
                geoName:(NSString *)geoName
                  rooms:(NSArray <NSString *>*)rooms
               latitude:(double)latitude
              longitude:(double)longitude
                success:(ThingSuccessLongLong)success
                failure:(ThingFailureError)failure;
```

Parameters

Parameter	Description
homeName	The name of a home.
geoName	The address of the home.
rooms	A list of room names for the home.
latitude	The latitude of the home.
longitude	The longitude of the home.
success	The success callback.
failure	The failure callback.

Example

ObjC:

```
- (void)addHome {
    [self.homeManager addHomeWithName:@"you_home_name"
                              geoName:@"city_name"
                                rooms:@[@"room_name"]
                             latitude:lat
                            longitude:lon
                              success:^(double homeId) {

        // The value of `homeId` for the home.
        NSLog(@"add home success");
    } failure:^(NSError *error) {
        NSLog(@"add home failure: %@", error);
    }];
}
```

Swift:

```
 func addHome() {
    homeManager.addHome(withName: "you_home_name",
                              geoName: "city_name",
                                rooms: ["room_name"],
                             latitude: lat,
                            longitude: lon,
                              success: { (homeId) in
        // The value of `homeId` for the home.
```

```
            print("add home success")
        }) { (error) in
            if let e = error {
                print("add home failure: \(e)")
            }
        }
    }
```

3. Query a list of homes

Returns a simple list of homes. To get home details, initialize the `home` object of `ThingSmartHome` and call the API method `getHomeDataWithSuccess:failure:` .

API description

```
// Returns a list of homes.
- (void)getHomeListWithSuccess:(void(^)(NSArray <ThingSmartHomeModel *>
*homes))success
                       failure:(ThingFailureError)failure;
```

Parameters

Parameter	Description
success	The success callback.
failure	The failure callback.

Example

ObjC:

```
- (void)getHomeList {

    [self.homeManager getHomeListWithSuccess:^(NSArray<ThingSmartHomeModel *>
*homes) {
        // A list of homes.
    } failure:^(NSError *error) {
        NSLog(@"get home list failure: %@", error);
    }];
}
```

Swift:

```
let homeManager: ThingSmartHomeManager = ThingSmartHomeManager()

func getHomeList() {
    homeManager.getHomeList(success: { (homes) in
        // A list of homes.
    }) { (error) in
        if let e = error {
            print("get home list failure: \(e)")
        }
    }
}
```

4. Modify home information

API description

```
- (void)updateHomeInfoWithName:(NSString *)homeName
                       geoName:(NSString *)geoName
                      latitude:(double)latitude
                     longitude:(double)longitude
                       success:(ThingSuccessHandler)success
                       failure:(ThingFailureError)failure;
```

Parameters

Parameter	Description
homeName	The name of a home.
geoName	The name of the home address.
latitude	The latitude of the home.
longitude	The longitude of the home.
success	The success callback.
failure	The failure callback.

Example

ObjC:

```
- (void)updateHomeInfo {
    self.home = [ThingSmartHome homeWithHomeId:homeId];
    [self.home updateHomeInfoWithName:@"new_home_name" geoName:@"city_name"
latitude:lat longitude:lon success:^{
        NSLog(@"update home info success");
    } failure:^(NSError *error) {
        NSLog(@"update home info failure: %@", error);
    }];
}
```

Swift:

```
func updateHomeInfo() {
    home?.updateInfo(withName: "new_home_name", geoName: "city_name", latitude:
lat, longitude: lon, success: {
        print("update home info success")
    }, failure: { (error) in
        if let e = error {
            print("update home info failure: \(e)")
        }
    })
}
```

5. Delete a home

API description

```
- (void)dismissHomeWithSuccess:(ThingSuccessHandler)success
                       failure:(ThingFailureError)failure;
```

Parameters

Parameter	Description
success	The success callback.
failure	The failure callback.

Example

ObjC:

```
- (void)dismissHome {

    [self.home dismissHomeWithSuccess:^() {
        NSLog(@"dismiss home success");
    } failure:^(NSError *error) {
        NSLog(@"dismiss home failure: %@", error);
    }];
}
```

Swift:

```
func dismissHome() {
    home?.dismiss(success: {
        print("dismiss home success")
    }, failure: { (error) in
        if let e = error {
            print("dismiss home failure: \(e)")
        }
    })
}
```

6. Query home details

Returns the details of a specific home. This way, the `home` instance object can have data respecting the properties `homeModel`, `roomList`, `deviceList`, `groupList`, `sharedDeviceList`, and `sharedGroupList`.

API description

```
- (void)getHomeDataWithSuccess:(void (^)(ThingSmartHomeModel *homeModel))success
                       failure:(ThingFailureError)failure;
```

Parameters

Parameter	Description
success	The success callback.
failure	The failure callback.

Example

ObjC:

```
- (void)getHomeDataInfo {
    self.home = [ThingSmartHome homeWithHomeId:homeId];
        [self.home getHomeDataWithSuccess:^(ThingSmartHomeModel *homeModel) {
        // The home information indicated by `homeModel`.
        NSLog(@"get home data success");
    } failure:^(NSError *error) {
        NSLog(@"get home data failure: %@", error);
    }];
}
```

Swift:

```
func getHomeDataInfo() {
    home?.getDataWithSuccess({ (homeModel) in
        print("get home data success")
    }, failure: { (error) in
        if let e = error {
            print("get home data failure: \(e)")
        }
    })
}
```

7. Home cache data

After each API request updates the home data, the home data will be cached. The next time the app is launched, it will automatically load the current user's home cache data. Once the cache loading is completed, you can obtain the cached `ThingSmartHome` data and perform subsequent operations. The `ThingSmartHomeManager` provides a method to wait for the cache loading to complete.

API description

```
- (void)waitLoadCacheComplete:(void (^)(BOOL complete))block;
```

Parameters

Parameter	Description
block	Callback after cache loading is completed. The value of `complete` is `YES` when the loading is successful.

Example

Objc:

```
- (void)getCacheHome {
    [self.homeManager waitLoadCacheComplete:^(BOOL complete) {
       NSLog(@"load home cache complete");
    }];
}
```

Swift:

```
func getCacheHome() {
    homeManager?.waitLoadCacheComplete { complete in
        print("load home cache complete")
    }
}
```

8. Sort devices and groups in a home

API description

```
- (void)sortDeviceOrGroupWithOrderList:(NSArray<NSDictionary *> *)orderList
                               success:(ThingSuccessHandler)success
                               failure:(ThingFailureError)failure;
```

Parameters

Parameter	Description
orderList	The list of sorted devices or groups.
success	The success callback.
failure	The failure callback.

Example

ObjC:

```
// orderList: [@{@"bizId": @"XXX", @"bizType": @"XXX"},@{@"bizId":
@"XXX",@"bizType": @"XXX"}] `bizId` is the device ID or the group ID. `bizType`
of the device = @"6" `bizType` of the group = @"5"
- (void)sortDeviceOrGroupWithOrderList:(NSArray<NSDictionary *> *)orderList {
    [self.home sortDeviceOrGroupWithOrderList:orderList success:^ {
        NSLog(@"sort device or group success");
    } failure:^(NSError *error) {
        NSLog(@"sort device or group failure: %@", error);
    }];
}
```

Swift:

```
func sortDeviceOrGroup(withOrderList orderList: [[AnyHashable : Any]]?) {
    home.sortDeviceOrGroup(withOrderList: orderList, success: {
        print("sort device or group success")
    }, failure: { error in
        if let error = error {
            print("sort device or group failure: \(error)")
        }
    })
}
```

9. Callbacks of home list changes

After you implement the delegate protocol `ThingSmartHomeManagerDelegate` , you can process the callbacks of home list changes.

9.1. Add a home

API description

```
- (void)homeManager:(ThingSmartHomeManager *)manager didAddHome:
(ThingSmartHomeModel *)home;
```

Parameters

Parameter	Description
manager	The instance of the home management class.
home	The added home model.

9.2. Delete a home listener

API description

```
- (void)homeManager:(ThingSmartHomeManager *)manager didRemoveHome:(long
long)homeId;
```

Parameters

Parameter	Description
manager	The instance of the home management class.
homeId	The ID of the deleted home.

9.3. Connect to an MQTT server

An MQTT persistent connection is closed after the program enters the background, and restarted after the program enters the foreground. Therefore, the current home details must be queried again using the delegate to keep the current home data up to date.

API description

```
- (void)serviceConnectedSuccess;
```

Example

ObjC:

```
#pragma mark - ThingSmartHomeManagerDelegate

// A home is added.
- (void)homeManager:(ThingSmartHomeManager *)manager didAddHome:(ThingSmartHomeModel *)home {

}

// A home is deleted.
- (void)homeManager:(ThingSmartHomeManager *)manager didRemoveHome:(long long)homeId {

}

// An MQTT connection is built.
- (void)serviceConnectedSuccess {
    // Returns the details of the current home from the cloud and refreshes the UI.
}
```

Swift:

```
extension ViewController: ThingSmartHomeManagerDelegate {

    // A home is added.
    func homeManager(_ manager: ThingSmartHomeManager!, didAddHome home: ThingSmartHomeModel!) {

    }

    // A home is deleted.
    func homeManager(_ manager: ThingSmartHomeManager!, didRemoveHome homeId: Int64) {

    }

    // An MQTT connection is built.
    func serviceConnectedSuccess() {
        // Returns the details of the current home from the cloud and refreshes the UI.
    }
}
```

10. Callbacks of home information changes

After you implement the delegate protocol `ThingSmartHomeDelegate`, you can process the callbacks of information changes for a single home.

Example

ObjC:

```
- (void)initHome {
    self.home = [ThingSmartHome homeWithHomeId:homeId];
    self.home.delegate = self;
}

#pragma mark - ThingSmartHomeDelegate

// Home information such as a home name is changed.
- (void)homeDidUpdateInfo:(ThingSmartHome *)home {
    [self reload];
}

// The list of shared devices is updated.
- (void)homeDidUpdateSharedInfo:(ThingSmartHome *)home {
    [self reload];
}

// A room is added to the home.
- (void)home:(ThingSmartHome *)home didAddRoom:(ThingSmartRoomModel *)room {
    [self reload];
}

// A room is removed from the home.
- (void)home:(ThingSmartHome *)home didRemoveRoom:(long long)roomId {
    [self reload];
}

// Room information such as a room name is changed.
- (void)home:(ThingSmartHome *)home roomInfoUpdate:(ThingSmartRoomModel *)room {
    [self reload];
}

// The mappings between rooms and devices or groups are updated.
- (void)home:(ThingSmartHome *)home roomRelationUpdate:(ThingSmartRoomModel
*)room {
    [self reload];
}

// A device is added.
- (void)home:(ThingSmartHome *)home didAddDeivice:(ThingSmartDeviceModel
*)device {
    [self reload];
}

// A device is removed.
- (void)home:(ThingSmartHome *)home didRemoveDeivice:(NSString *)devId {
    [self reload];
}

```

```
// Device information such as a device name or online status is changed.
- (void)home:(ThingSmartHome *)home deviceInfoUpdate:(ThingSmartDeviceModel *)device {
    [self reload];
}

// Device data points (DPs) are updated for the home.
- (void)home:(ThingSmartHome *)home device:(ThingSmartDeviceModel *)device dpsUpdate:(NSDictionary *)dps {
    [self reload];
}

// A group is added.
- (void)home:(ThingSmartHome *)home didAddGroup:(ThingSmartGroupModel *)group {
    [self reload];
}

// A group is removed.
- (void)home:(ThingSmartHome *)home didRemoveGroup:(NSString *)groupId {
    [self reload];
}

// Group information such as a group name is changed.
- (void)home:(ThingSmartHome *)home groupInfoUpdate:(ThingSmartGroupModel *)group {
    [self reload];
}

// Group DPs are updated for the home.
- (void)home:(ThingSmartHome *)home group:(ThingSmartGroupModel *)group dpsUpdate:(NSDictionary *)dps {
    [self reload];
}

// Device alerts are updated for the home.
- (void)home:(ThingSmartHome *)home device:(ThingSmartDeviceModel *)device warningInfoUpdate:(NSDictionary *)warningInfo {
    //...
}


// Device update status is changed for the home.
- (void)home:(ThingSmartHome *)home device:(ThingSmartDeviceModel *)device upgradeStatus:(ThingSmartDeviceUpgradeStatus)upgradeStatus {
    //...
}
```

Swift:

```
var home: ThingSmartHome?

extension ViewController: ThingSmartHomeDelegate {

  func initHome() {
      home = ThingSmartHome(homeId: homeId)
      home?.delegate = self
  }

```

```
    // Home information such as a home name is changed.
    func homeDidUpdateInfo(_ home: ThingSmartHome!) {
//        reload()
    }

    // The list of shared devices is updated.
    func homeDidUpdateSharedInfo(_ home: ThingSmartHome!) {

    }

    // A room is added to the home.
    func home(_ home: ThingSmartHome!, didAddRoom room: ThingSmartRoomModel!) {
        //...
    }

    // A room is removed from the home.
    func home(_ home: ThingSmartHome!, didRemoveRoom roomId: int32!) {
        //...
    }

    // Room information such as a room name is changed.
    func home(_ home: ThingSmartHome!, roomInfoUpdate room: ThingSmartRoomModel!)
{
//        reload()/
    }

    // The mappings between rooms and devices or groups are updated.
    func home(_ home: ThingSmartHome!, roomRelationUpdate room:
ThingSmartRoomModel!) {

    }

    // A device is added.
    func home(_ home: ThingSmartHome!, didAddDeivice device:
ThingSmartDeviceModel!) {

    }

    // A device is removed.
    func home(_ home: ThingSmartHome!, didRemoveDeivice devId: String!) {

    }

    // Device information such as a device name is changed.
    func home(_ home: ThingSmartHome!, deviceInfoUpdate device:
ThingSmartDeviceModel!) {

    }

    // Device DPs are updated for the home.
    func home(_ home: ThingSmartHome!, device: ThingSmartDeviceModel!, dpsUpdate
dps: [AnyHashable : Any]!) {
        //...
    }

    // A group is added.
    func home(_ home: ThingSmartHome!, didAddGroup group: ThingSmartGroupModel!) {

    }

    // A group is removed.
    func home(_ home: ThingSmartHome!, didRemoveGroup groupId: String!) {

```

```
    }

    // Group information such as a group name is changed.
    func home(_ home: ThingSmartHome!, groupInfoUpdate group:
ThingSmartGroupModel!) {

    }

    // Group DPs are updated for the home.
    func home(_ home: ThingSmartHome!, group: ThingSmartGroupModel!, dpsUpdate
dps: [AnyHashable : Any]!) {
                //...
    }

    // Device alerts are updated for the home.
    func home(_ home: ThingSmartHome!, device: ThingSmartDeviceModel!,
warningInfoUpdate warningInfo: [AnyHashable : Any]!) {
            //...
    }

    // Device update status is changed for the home.
    func home(_ home: ThingSmartHome!, device: ThingSmartDeviceModel!,
upgradeStatus status ThingSmartDeviceUpgradeStatus) {
            //....
    }

}
```

11. Query weather of the home location

11.1. Query the weather overview for a home

API description

Returns the weather overview of the city where the home is located. Weather data includes the city name, weather conditions, such as sunny, cloudy, or rainy, and weather icons.

```
- (void)getHomeWeatherSketchWithSuccess:(void(^)(ThingSmartWeatherSketchModel
*))success
                                failure:(ThingFailureError)failure;
```

If no method is found, add the following line:

```
#import <ThingSmartDeviceKit/ThingSmartHome+Weather.h>
```

Parameters

Parameter	Description
success	The success callback.
failure	The failure callback.

Response parameters of `ThingSmartWeatherSketchModel`

Parameter	Description
condition	The weather conditions, such as sunny, cloudy, and rainy.
iconUrl	The highlighted URL of a weather icon.
inIconUrl	The URL of a weather icon.
temp	The temperature.

Example

ObjC:

```
- (void)getHomeWeatherSketch {
    [self.home getHomeWeatherSketchWithSuccess:^(ThingSmartWeatherSketchModel
*weatherSketchModel) {
        NSLog(@"success get weather summary model: %@",weatherSketchModel);
    } failure:^(NSError *error) {
        NSLog(@"failure with error: %@", error);
    }];
}
```

Swift:

```
func getHomeWeatherSketch() {
    home.getWeatherSketch(success: { (weatherSketchModel) in
        print("success get weather summary model: \(weatherSketchModel)");
    }) { (e) in
        print("failure with error: \(e)")
    };
}
```

11.2. Query the weather details for a home

API description

Returns the weather details for the city where the home is located. Multiple types of weather data are returned, such as the temperature, humidity, ultraviolet (UV) index, and air quality.

- `optionModel` can be `nil`. If so, the response parameters follow the settings of the last successful request. If only a single unit is changed, the remaining parameters still follow the settings of the last successful request.
- The weather service and returned weather details might be different depending on the served area. For example, if the current home account is registered in China, the information about the wind speed and air pressure is not returned.

```
- (void)getHomeWeatherDetailWithOption:(ThingSmartWeatherOptionModel *)optionModel
                               success:(void(^)(NSArray<ThingSmartWeatherModel *> *))success
                               failure:(ThingFailureError)failure;
```

Parameters

Parameter	Description
optionModel	The unit settings of the weather details.
success	The success callback.
failure	The failure callback.

Parameters of `ThingSmartWeatherOptionModel` in the request

Parameter	Description
pressureUnit	The unit of air pressure.
windspeedUnit	The unit of wind speed.
temperatureUnit	The unit of temperature.

Parameter	Description
limit	The number of request parameters. By default, all parameters are returned.

Response parameters of `ThingSmartWeatherModel`

Parameter	Description
icon	The URL of a weather details icon.
name	The name of a weather parameter.
unit	The unit of a parameter.
value	The value of a parameter.

Example

ObjC:

```
- (void)getHomeWeatherDetail {
    [self.home getHomeWeatherDetailWithOption:optionModel
                                      success:^(NSArray<ThingSmartWeatherModel *>
*weatherModels) {
          NSLog(@"success get weather model: %@",weatherModels);
                                    } failure:^(NSError *error) {
          NSLog(@"failure with error: %@", error);
    }];
}
```

Swift:

```
func getHomeWeatherDetail() {
    let optionModel = ThingSmartWeatherOptionModel()
    // do some optionModel config
    home.getWeatherDetail(withOption: optionModel, success:
{ (weatherSketchModel) in
        print("success get weather summary model: \(weatherSketchModel)");
    }) { (error) in
        print("failure with error: \(error)")
    }
}
```