User Manual

# Ayla Mobile Getting Started

## *Developers' Guide for iOS and Android platforms*

Version: 5

Date Released: July 26, 2026

Document Number: AY006GSB6-5

Ayla Networks

## Table of Contents

## Table of Figures



## Table of Tables

# Introduction

The Ayla Mobile SDK provides Ayla customers a powerful set of tools for building mobile applications integrated with the Ayla Cloud Service. Designed for both iOS and Android platforms, the Ayla Mobile SDK lets app developer's focus on the app details instead of data management or network communication, leading to a faster time to market and a more solid application.

## 1.1. Audience

This document is directed towards mobile application developers who are writing an Ayla-connected application. The Ayla Mobile SDK was redesigned in version 5.0 to handle most of the mundane or complicated tasks required by all Ayla-connected mobile applications, as well as provide a clean and easy-to-use set of APIs to interact with the Cloud Service or directly with local IoT devices.

## 1.2. Related Documentation

OEM Dashboard User Guide (AY006UDB3)

Developer Portal User Guide (AY006UDP3)

AURA app and Ayla SDK (Both are available on GitHub for registered OEM's who have signed the Ayla software license agreement.)

## 1.3. Customer Support

Support is available at http://support.aylanetworks.com. The support site provides access to the knowledge base. You can also create a support ticket for additional help.

## 2. Overview

The Ayla Mobile SDK provides your application with the ability to create, update, and manage the items listed below.

- Sign up a new user
- Sign in an existing user
- Request account updates (password reset, profile changes, etc.)
- Manage login sessions, keeping authorization updated automatically
- Manage devices, keeping the list of registered devices and their states up to date
- Configure push, email or SMS notifications for device states
- Configure schedules to update devices at a preset time or interval
- Notify the application when any devices change state
- Control and configure devices
- Discover and configure new devices
- Manage contacts for notifications

The Mobile SDK is part of the mobile applications that are available to mobile developers. Figure 1 shows where the SDK fits with the other mobile content.

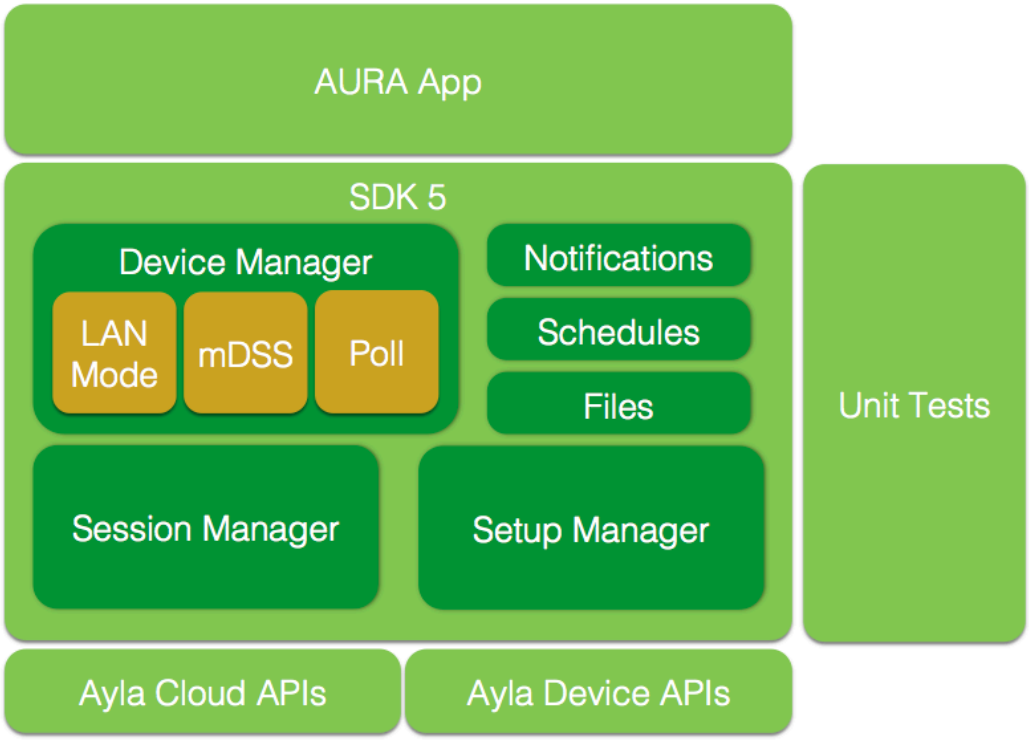


**Figure 1 Mobile SKD Contents**

The AURA app is a sample application that is available in GitHub. AURA is not a part of the SDK but is useful when using the SDK. More information about AURA is in on page 9.

The Ayla Mobile SDK is divided into several components to help with account creation and management, device setup and the maintenance of device state while the application is running.

The Session Manager is created after the user signs in to the Ayla Cloud service and is responsible for keeping that session up-to-date by refreshing the authorization as needed. It also provides notifications to the application when the login state has changed, such as the user signing out.

The Device Manager is also created after the use signs in to the Ayla Cloud service and is responsible for maintaining the list of OEM devices associated with the user's account. The Device Manager is the most commonly used component of the Ayla Mobile SDK, as most parts of the mobile application deal directly with Device objects and their states.

The SDK also contains methods to help configure new devices and register them to the user's account. It includes classes such as *AylaSetup* and *AylaRegistration* to assist with these tasks.

Included with the SDK distribution are unit tests, which may be used as examples of SDK API usage as well as verifying the functionality of API methods.

Distributed separately from the SDK is *Aura*, a developer oriented test application written by Ayla to provide examples of SDK usage in a real-world system. It is strongly recommended that developers download and build the *Aura* application for the platform under development. *Aura* is a valuable reference to just about any type of interaction with the Ayla SDK. Using Aura along with an Ayla Design Kit, (also known as an Evaluation Board or EVB), provides a known good solution. Having a working solution for Wi-Fi Setup, Registration, and other Ayla Platform features can be of great benefit when developing a new IoT solution, which consists of new hardware/firmware on a device, a new app running on a phone/tablet, and incrementally changing device templates on the Ayla cloud.

# 3. Getting Started: Prerequisites

The following prerequisites will assist your mobile development effort.

## 3.1. Aura

The Ayla Mobile SDK includes a sample application, *Aura*, designed for developers working on Ayla-connected apps. The source code and screens in *Aura* will be referenced throughout this document, so it is recommended that it is downloaded, built and run before continuing to the SDK of sections of this document.

The source code for *Aura* may be found in the following repositories:

https://github.com/AylaNetworks/Android_Aura_Public

https://github.com/AylaNetworks/iOS_Aura_Public

Instructions for building and installing Aura can be found in the README.md file at the root of each project.

Once Aura has been built and installed on a mobile device, either sign in using an existing Ayla account, or create a new Ayla account through the app or at https://developer.aylanetworks.com/ .

A new account will not have any devices registered to it, so will appear sparse after the first sign-in.

## 3.2. Ayla Design Kit

In order for Aura to be useful, at least one device should be registered to the account. Aura was designed to be compatible with the Ayla Design Kit, though it will often work with other devices with limited functionality. It is recommended to obtain an Ayla Design Kit to use during development as a reference while developing Ayla mobile applications. Contact Ayla Customer Support at http://support.aylanetworks.comto obtain an Ayla Design Kit.

## 3.3. Ayla OEM Dashboard

The mobile stet up that in includes using the OEM Dashboard are described in this section.

### 3.3.1. Ayla Regions

Ayla provides Cloud Services throughout the world. There are three locations where Ayla has deployed cloud services: the United States of America, Europe,

and China. Each IoT device is bound to one region. As a mobile developer, your application also needs to be bound to the same region/location. The first step in creating that binding is by creating an App ID and App Secret using the OEM Dashboard. Ayla OEM Dashboard

The Ayla Cloud Service provides a web interface to configure various aspects of the Ayla OEM account for your company. Ayla customers may sign in to the OEM Dashboard to create application IDs, necessary for mobile apps, as well as configure various parameters for device templates and other OEM-specific information.

There are two services provided by Ayla for customers: the Development Services and the Field Services. The Development Services are used for devices and applications under active development and is only accessible to Ayla customers. The Field Services are used for final products released to the public: the OEM's end customer.

Note that there is an OEM Dashboard for each Ayla region and service. The Ayla OEM Dashboard can be found here:

### 3.3.2. OEM Dashboards Per Location/Region and Service

- US Development Service: https://dashboardinternal.aylanetworks.com
- US Field Service: https://dashboardfield.aylanetworks.com
- CN Development Service: https://dashboard.ayla.com.cn
- CN Field Service: https://dashboardinternal.ayla.com.cn
- EU Development Service: Use US Development OEM Dashboard
- EU Field Service: https://dashboard-field-eu.aylanetworks.com

Use the OEM Dashboard for OEM scoped activities like creating appId, APNS certificate association, email template CRUD, etc. Don't use an OEM Dashboard to change your test device as the user scope mismatch will cause permission errors. Use the app you are developing or the Developer Site for individual device changes and testing. Here are the Developer Sites for each region.

### 3.3.3. Developer Portal Sites Per Location/Region

- US Development Service: https://developer.aylanetworks.com
- CN Development Service: https://developer.ayla.com.cn
- EU Development Service: Use US Development Site portal

There is no Developer Site portals on any of Field Services as active development on the Field Service is not allowed.

Refer to the OEM Dashboard User Guide (AY006UDB3) for information.

#### 3.3.4. App ID and Secret

Each mobile application requires an application ID and secret to identify the application and OEM within the Ayla network. One app ID / secret pair may be used for both iOS and Android versions of the same application.

OEMs may create app IDs and secrets in the OEM Dashboard by selecting “OEM Profile” in the left pane, then choosing the “Apps” tab in the right pane and clicking “Create New”. Choose a name for your application and a prefix (a string that is pre-pended to the unique ID and secret) and click “Create”.

Copy the generated app ID and secret, as these will be required to configure the Ayla SDK within each mobile application.

### 3.4. Email templates

The Ayla Cloud Service sends users an email in response to different events or requests. The format of these emails may be specified by each manufacturer and provided as an email template on the Ayla OEM Dashboard.

Email templates may include substitution arguments such as those shown in Table 1.

**Table 1 Subsection arguments**

| Argument | Definition |
|---|---|
| [[logo_image]] | The OEM’s logo |
| [[android_app_link_image]] | Image for the link to launch the Android application |
| [[ios_app_link_image]] | Image for the link to launch the iOS application |
| [[property_name]] | Name of the property |
| [[property_value]] | Value of the property |
| [[property_update_time]] | Time the property was updated |
| [[device_product_name]] | Name of the device |

| | |
|---|---|
| [[device_dsn]] | Serial number of the device |
| [[user_name]] | Name of the user |
| [[user_message]] | Message defined when the trigger app was created |
| [[trigger_app_email_body]] | Placeholder for HTML included in the trigger app |
| [[user_confirmation_token]] | Confirmation token for user sign-up |
| [[user_password_reset_token]] | Confirmation token for password reset |

Manufacturers should set up email templates for:

- Password reset confirmation
- New user confirmation
- Property notifications (may be several depending on the nature of the manufacturer's devices)

Details of email templates can be found in the *Customizing Notification Messages* document from the Ayla Customer Support site.

# 4. Creating an App

Once the prerequisites have been taken care of, you're ready to start work on the app. Each platform has slightly different steps to integrating the SDK into your application. The following instructions assume that a skeleton application has already been created using the platform's tools. Always defer to the README.md file in the root of each project for the latest build information.

## 4.1. Android

1. Build the SDK either in Android Studio or on the command line with gradle assembleDebug
2. In your application workspace, open the Project Structure dialog from the File menu in Android Studio
3. Select the '+' in the top-left to add a new module

4. Choose “Import Gradle Project” and navigate to the root of the Android_AylaSDK directory

### 4.2. iOS

1. Build the SDK by following the instructions in the README.md file at the root of the project
2. If you don’t already have one, create a workspace in Xcode containing both your application’s project and the Ayla SDK project
3. Add *pod 'iOS_AylaSDK'* to your build target in Podfile.
4. Execute pod install from the workspace shell
5. Open the .xcworkspace file to launch Xcode and import the project

### 4.3. Android Application Permissions

Android requires applications specify a set of permissions that are needed to be granted by the user to use the application. The Ayla SDK has a set of permissions that are required to be granted in order to function. Table 2 lists the required permissions.

**Table 2 Android required permissions**

| Permission | Reason |
|---|---|
| INTERNET | Internet access is required to communicate with the Ayla Cloud Service. The SDK cannot function without this permission. |
| ACCESS_NETWORK_STATE | The network state must be readable by the SDK in order to communicate with the Cloud Service and devices on the local network |
| ACCESS_WIFI_STATE | The WiFi state must be readable by the SDK in order to set up WiFi devices and handle connectivity changes on the mobile device |
| CHANGE_WIFI_MULTICAST_STATE | The WiFi multicast state must be modified by the SDK in order to discover devices on the local network |

| | |
|---|---|
| CHANGE_WIFI_STATE | The WiFi state must be configurable by the SDK in order to connect to device access points for WiFi setup |
| ACCESS_COARSE_LOCATION | The coarse location permission is required in order to obtain WiFi SSID information to discover devices that need to be set up |
| WRITE_EXTERNAL_STORAGE | External storage access is required to write logs. If this permission is not granted, the SDK will function but will not record any logging information. |

### 4.3.1. Baidu Push permissions

If Baidu push notifications are supported in the application. Additional permissions that are required are listed in Table 3.

**Table 3 Baudu permissions**

| Permission | Reason |
|---|---|
| RECEIVE_BOOT_COMPLETED | Used to start the Baidu service when the phone has booted up |
| DISABLE_KEYGUARD | Allows interaction with notifications while the screen is locked |
| READ_PHONE_STATE | Used to obtain the phone's unique identifier |
| WRITE_SETTINGS | Allows Baidu to save user settings |
| VIBRATE | Allows vibration notifications |
| ACCESS_DOWNLOAD_MANAGER | Used to update Baidu |
| DOWNLOAD_WITHOUT_NOTIFICATION | Used to update Baidu |
| EXPAND_STATUS_BAR | Used to provide expanded notifications |

### 4.3.2. GCM permissions

If Google Cloud Messaging (Google push notifications) are supported in the application, additional permissions are shown in Table 4.

**Table 4 GCM permissions**

| Permission | Reason |
|---|---|
| C2D_MESSAGE | Permission to receive GCM messages |
| WAKE_LOCK | Allows notifications to interrupt the lock screen |

| | |
|---|---|
| c2dm.permission.RECEIVE | Allows the application to receive messages |

## 4.4. Configuring System Settings

Once the application structure has been set up, the first step in using the Ayla Mobile SDK is to initialize it. Initialization is done through a structure called AylaSystemSettings, which is filled out and passed to the initialize method in the top-level AylaNetworks class.Table 5 lists the initialization settings.

**Table 5 AylaSystemSettings initialization**

| Android | iOS |
|---|---|
| void AylaNetworks.initialize() | +[AylaNetworks initializeWithSettings:] |
| com.aylanetworks.aylasdk.AylaSystemSettings | AylaSystemSettings.h |

The default constructor (Android) or method +defaultSystemSettings (iOS) provides an AylaSystemSettings object with default values configured. Some values must be overridden by the application, however, to link the application with the correct service and dataset in the cloud. The settings that should be carefully considered are listed Table 6.

**Table 6 Settings to consider**

| Parameter | Description |
|---|---|
| *appId* | Application ID generated on the OEM Dashboard for this app. See the App ID and Secret section earlier for details |
| *appSecret* | Application secret generated with the appId |
| *serviceType* | Points the app to either the Field or Development services. Development generally |

| | |
|---|---|
| | starts on the Development service and is moved to Field when the application is released to the public |
| *serviceLocation* | USA, China and Europe are supported. Contact your Ayla advisor if you are not sure which service your app should connect to. |
| *deviceDetailProvider* | Application-provided object that gives the SDK information about devices known to your application. Details below. |

### 4.4.1. Device Detail Provider

The DeviceDetailProvider is a helper object created by the application and given to the Ayla SDK during initialization. The DeviceDetailProvider allows the SDK to ask the application for details about a particular device that the SDK is managing.

Currently the DeviceDetailProvider contains a single method, getManagedPropertyNames (Android) / -getMonitoredPropertyNamesForDevice: (iOS). Application developers should create an object that implements this method and pass it to the Ayla SDK during initialization.

The implementation of this method should return an array of Strings containing the name of each AylaProperty that should be monitored by the SDK.

The method should examine the provided AylaDevice object and query its properties (such as “model” or “oemModel”) to determine the type of device. Once the device type has been determined, the method should return the array of property names for that device that the SDK should manage.

Managed properties are ensured to be kept up-to-date by the SDK. Properties that are not managed may still be obtained via calls to AylaProperty.fetchProperties() by passing in their names, or a null array of properties to fetch the entire set of properties. While these additional properties will subsequently be available, they will not be automatically kept up-to-date by the SDK and need to be manually queried by the application.

If no properties should be managed for the provided device, the method may return null.

### 4.5. Initialize the SDK

Once the AylaSystemSettings object has been created and initialized, it may be passed to the Ayla SDK via the call to AylaNetworks.initialize() or –[AylaNetworks initializeWithSettings:] along with a unique name for the session.

At this point the SDK has been initialized and is ready to be used.

## 5. SDK API Design Patterns

Most of the APIs provided by the Ayla Mobile SDK are asynchronous in nature, requiring one or more network calls that take time to complete. The SDK was designed to make handling the asynchronous results of API calls as easy as possible. While each platform's implementation is slightly different, the basic structure is the same.

Each API generally takes two "listener" parameters, a "success" listener and an "error" listener. On success, the requested object or objects are delivered to the "success" listener, while errors are delivered to the "error" listener as an AylaError object.

All asynchronous APIs in the Ayla Mobile SDK follow this pattern.

### 5.1. iOS

```
- (AylaHTTPTask *)loginWithAuthProvider:(AylaAuthProvider
*)authProvider

                              sessionName:(NSString *)sessionName

                                  success:(void
(^)(AylaAuthorization *authorization,


AylaSessionManager *sessionManager))successBlock

                                  failure:(void (^)(NSError
*error))failureBlock;
```

On success, the *successBlock* is called with pointers to the new *AylaAuthorization* object created as a result of the successful sign in operation as well as the *AylaSessionManager* that was created for the session.

If an error occurred, the *failureBlock* is called with an NSError containing the details of the error.

### 5.2. Android

```
public void signIn(AylaAuthProvider authProvider,
                   final String sessionName,
                   final Listener<AylaAuthorization>
successListener,
                   final ErrorListener errorListener)
```

Both the Listener and ErrorListener parameters may be anonymous inner classes defined to handle the *onResponse* / *onErrorResponse* methods, much like the blocks in the Objective-C APIs.

On success, the successListener's *onResponse* method is called with an *AylaAuthorization* object created as a result of the successful sign operation.

If an error occurred, the errorListener's *onErrorResponse* method is called with an *AylaError* object containing details about the failure.

## 6. User Accounts

Each developer should set up a user account to work with. User accounts may be created for the Development Service on the Developer Portal or using the account creation APIs in the Ayla Mobile SDK when set up to use the Development service and appID and appSecret generated from the OEM Dashboard.

When moving to the Field service, the application should use the appID and appSecret generated from the Field Dashboard site. It is important to understand that accounts on the Field service are different than accounts on the Development

service. It is strongly recommended to set up separate user accounts for Development and Field services.

## 6.1. Mobile Account APIs

Account creation, retrieval, updates and deletes (CRUD) may be performed by the mobile application. The *AylaLoginManager* class provides methods that may be used to manage user accounts. All of the methods in Table 7 may be used without a signed-in user.

**Table 7 Account creation**

| Call | Description |
|---|---|
| signUp | Creates a new user account. A valid email address, password and user first name are all required. Upon a successful sign-up, an email is sent to the user with a link to confirm the account. |
| confirmSignUp | Requires a token provided by the sign-in email sent when the user signed up. Mobile applications may be configured to provide the token as part of a custom URL that can be used to launch the mobile application. After the token is provided to confirmSignUp and the service accepts it, the user account is verified and the user may sign in. |
| resendConfirmationEmail | Re-sends the email that was sent when the user first signed up. Useful if the email was not received or was lost. |
| requestPasswordReset | Sends an email to the user with a link to reset their password. Like the confirmation email, this link may be configured to launch the mobile application instead of validating through a website. |
| resetPassword | Resets the user's password with a new one. Must include the token provided from the password reset email. |

Additionally, methods are available to delete or modify account details when the user is signed in to the application shown in Table 8.

**Table 8 Additional Methods**

| Method | Results |
|---|---|
| AylaLoginManager.deleteAccount | Deletes the signed-in user's account. This action also signs out the user. |
| AylaSessionManager.updateUserProfile | Updates details of the user's account such as phone number, address, etc. |

## 7. Session Management

When a user signs in, a new Session is created. A Session represents a user's authenticated login session with the Ayla Cloud Service.

To sign in to the Ayla network, the application creates an *AylaAuthProvider* object with the credentials needed to sign in to the service. The Ayla SDK includes several *AylaAuthProvider* objects that may be used to sign in using different sets of credentials:

- *UsernameAuthProvider* is used to sign in with a username and password. This is generally the first method used to sign in to an application.
- *AylaOAuthProvider* may be used to sign in using Google or Facebook credentials. The class must be provided a web view to provide web interaction with the user to sign in.
- *CachedAuthProvider* may be used to store a successful authorization and sign in by refreshing a previous authorization. This is generally used when the user has not signed out and the application is brought to the foreground.
- *SSOAuthProvider* may be used to sign in to the Ayla service using a Single Sign On provider. Additional configuration between Ayla Networks and the SSO provider is required to enable this type of sign-on.

When the selected AylaAuthProvider object has been configured with the appropriate credentials, it can be passed to AylaLoginManager's signIn() method.

On successful sign-in, an *AylaSessionManager* is created and may be obtained by calling AylaNetworks.getSessionManager() with the name of the session that was passed to *signIn()*.

### 7.1. Listening for session changes

The AylaSessionManager supports a listener interface that allows the application to be notified when the session state changes. There are two notifications that are sent out regarding the session:

- Session Closed: The user has signed out or the service rejected an attempt to refresh the session
- Authorization updated: The SessionManager has refreshed the current session's authorization and provides the new *AylaAuthorization* to the application

Once the user has signed in, the application should register a listener with AylaSessionManager to be notified of these changes.

If the session is closed, the application should clear out any existing UI elements and show the sign-in screen.

If the authorization is updated, the application may save the new authorization object to be used to sign in using a CachedAuthProvider later. Applications are responsible for saving / restoring the latest AylaAuthorization objects to sign in with.

Examples of signing in and setting up a SessionManagerListener may be found in Aura:

***iOS**: LoginViewController.swift*

***Android**: LoginActivity.java / MainActivity.java*

## 8. Device Management

This section describes device management options.

### 8.1. AylaDevice

*AylaDevice* is a class that represents a single physical device that has been registered to the user's account. An AylaDevice object contains information about the registered device such as it's type, name, IP address (if it has one), device serial number, etc. Ayla Devices are created by the Ayla Factory Service and then pushed to a region's Development Device Service. Once the IoT Devices

are ready for field trials and/or final release, the devices are pushed to the Ayla Field Device Service where they are accessible to end users.

Exactly one *AylaDevice* object is created by the Mobile SDK for each physical device that is registered to the user's account.

A device generally has one or more Properties that are conduits for exchanging data to or from the device. Properties for devices are configured by the OEM using Templates on the Ayla OEM Dashboard service, and are represented in the mobile SDK with *AylaProperty* objects.

### 8.1.1. Listening for changes

The Ayla Mobile SDK will keep *AylaDevice* objects up-to-date automatically. Changes that occur to *AylaDevice* objects may be monitored by the application by registering a *DeviceChangeListener* with the *AylaDevice*. The listener's methods will be called whenever the device's property values change, when it encounters an error trying to keep itself updated, or if it changes from LAN mode (direct connection between mobile device and target device) to Cloud mode (polls the Cloud service for changes when direct connection is not possible).

### 8.1.2. LAN mode

The Ayla Mobile SDK will automatically find the best means of communication between the mobile application and the IoT device. If the mobile device and IoT device reside on the same local network, the mobile SDK will connect directly to the IoT device rather than poll the Ayla Cloud service for updates.

LAN mode is only supported for devices owned by the account owner. Devices shared with the account owner may not use LAN mode for communication, as account-level security is not available without communicating with the Ayla Cloud.

## 8.2. AylaProperty

An *AylaProperty* represents a particular endpoint of a device that may be read from or written to. Each type of device will have a unique set of *AylaProperties* that are created and configured by the device's template in the Ayla OEM Dashboard.

A property generally represents a state of the device, such as “on” or “off”, or “current temperature” or “target temperature”. Some properties are read-only, while others are writable.

A device may have many properties, though only a few may be of importance to the mobile application. Properties that are used by the mobile application should be made known to the Mobile SDK by returning their names from the *DeviceDetailProvider* discussed in the Creating your App section of this document.

Each *AylaDevice* has a method called *getProperties* that returns the set of properties known to the SDK for the device. This set usually will be limited to the properties provided from the *DeviceDetailProvider,* though may contain additional properties if they were manually fetched by the application and therefore now known to the system.

Each *AylaProperty* maintains a historical record of values called *AylaDatapoints*. These values may be queried from the Ayla Cloud Service by calling the *fetchDatapoints* method on the AylaProperty object.

The latest datapoint’s value may always be queried directly from the *AylaProperty* object via the *getValue* method on *AylaProperty*.

Creating a datapoint (e.g. changing the “current value” of a property) may be performed via a call to *createDatapoint* on the *AylaProperty* object

## 8.3. AylaDatapoint

An *AylaDatapoint* represents a value written to a property at a given time. For simplicity, the value of the most recent datapoint is always available via the *AylaProperty’s* getValue method.

Multiple datapoints may be fetched from the cloud service by calling the *AylaProperty’s* fetchDatapoints method. The caller may specify the number of datapoints to be returned as well as a range of dates the returned datapoints should fall within.

Changing the “value” of a property is done by creating a datapoint for that property using the property’s `createDatapoint` method.

#### *8.3.1.1. Aura CreateDatapoint examples*

**iOS:** *PropertyModel.swift*

**Android:** *DeviceDetailFragment.java*

## 8.4. Datapoint Ack

When not connected directly to a device via the LAN, updates to the device must go first to the Ayla Cloud Service and are then propagated to the device. The cloud service will immediately accept a new datapoint (change to the value of a property) and attempt to contact the device to pass it the datapoint.

However, the device may not be reachable or working for whatever reason. The cloud will deliver the datapoint to the device as soon as it is available, but the user creating that datapoint in the first place will need to know that the operation did not complete entirely.

To solve this issue, properties may be configured to “ack” datapoints when they are received. If a property is configured to use datapoint ack, the mobile SDK will poll the Ayla service for the acknowledgement from the device until the API timeout has expired. If the datapoint ack has been observed, the API call will succeed. If the datapoint ACK was not received, a Timeout error will be returned to the caller.

Properties that are not configured to acknowledge datapoints will return success once the datapoint has been created on the Ayla Cloud Service without polling for an acknowledgement.

## 8.5. Batch Datapoints

It is possible to create a set of datapoints for multiple properties across multiple devices using a single API call. This can be useful to configure a group of devices at once, or to configure a single device with multiple properties using only one API call.

*AylaDeviceManager* provides the `createDatapointBatch` API which takes in an array of AylaDatapointBatchRequest objects, each configured with the device, property and value that should be created.

## 8.6. AylaDeviceManager

When an *AylaSessionManager* is created to manage the login session, it creates an *AylaDeviceManager* to manage the devices registered to the account. The *AylaDeviceManager* instance is available as a property on the *AylaSessionManager* and will be available for as long as the session is active.

The *AylaDeviceManager* is responsible for maintaining the list of registered devices and ensuring that the devices are kept up-to-date.

Like *AylaSessionManager*, *AylaDeviceManager* provides a listener interface that may be used to notify the application of various device-related events. Application developers may implement the *DeviceManagerListener* interface and register with *AylaDeviceManager* to receive these notifications.

### 8.6.1. Device Manager Initialization

After sign-in, the *AylaDeviceManager* runs through an initialization sequence, fetching all required information about the devices registered to the account. Once this sequence is finished, listeners are notified via a call to *deviceManagerInitComplete* on the listener interface. Once this method has been called, the *AylaDeviceManager* is ready and will continue to keep the device list updated.

Listeners are also notified when the device manager changes state via the `deviceManagerStateChanged` method.

Table 9 shows the states the *AylaDeviceManager* goes through during initialization.

**Table 9 AylaDeviceManager states**

| State | Purpose |
|---|---|
| Uninitialized | Initial state of *AylaDeviceManager* |
| FetchingDeviceList | Fetches the list of registered devices from the Ayla Cloud Service |
| FetchingDeviceProperties | Fetches the list of properties for each device from the Ayla Cloud Service. The list of properties for each device is obtained via calls to the *DeviceDetailProvider* provided to the SDK during initialization. |

| | |
|---|---|
| FetchingLanConfig | Fetches the LAN configuration information from the Ayla Cloud Service for each device. This information is required to connect directly to devices via a local area network. |
| Ready | Normal operation. When this state is reached, listeners are notified via *deviceManagerInitComplete* |
| Error | An unrecoverable error has occurred |
| Paused | Set when the application enters the background |

When devices are added to the account (registered), the device manager goes through these states again to fetch the required information for the new device.

## 8.7. The Device List

*AylaDeviceManager* is responsible for maintaining the list of devices registered to the account. Once it has reached the Ready state, the list of devices may be obtained via a call to *getDevices*.

Notifications of changes to the list of devices may be received by registering a listener to *AylaDeviceManager* via a call to *addListener.* When the list of devices changes (e.g. a device is registered or unregistered), listeners will be notified via a call to *deviceListChanged*.

#### *8.7.1.1. Aura Device List Examples*

Examples of using the device list and listening for changes can be found in Aura:

***iOS**: DeviceListTVController.swift*

***Android**: AllDevicesFragment.java*

## 8.8. WiFi Setup

New devices that connect to a WiFi network need to be configured to connect to the user's home network. Ayla-enabled devices that support WiFi generally are configured out-of-the-box in AP mode, broadcasting an SSID that is identifiable to the mobile application. This AP is the entry point to communication with the mobile device and is used to configure the device to connect to the user's home network.

Android allows applications to modify the mobile device's WiFi configuration, while iOS does not. Therefore, the procedure for setting up a new device on the user's WiFi network is different on each platform.

#### *8.8.1.1. Android: WiFi Setup*

On Android, the steps taken to set up a new device on WiFi are as follows:

1. Scan for WiFi access points that match a supplied pattern (e.g. SSIDs that begin with "Ayla-" and contain 12 hex digits afterward)
2. Connect the mobile device to the discovered AP and establish a secure session
3. Ask the connected device to scan for WiFi networks. This step is necessary even though the mobile device just scanned, as the target device may see a different set of APs than the mobile device
4. Allow the user to select the discovered access point to connect to, and enter the WiFi password if required
5. Provide the SSID and password to the device via the secure session
6. Poll the device to see if it was able to connect successfully to the AP
7. Re-join the original AP
8. Verify with the Ayla Cloud Service that the device has connected

These methods may all be performed by creating an *AylaSetup* object and using its methods.

#### *8.8.1.2. Aura WiFi Setup Examples (Android)*

*WifiSetupFragment.java*

*SetupWizardFragment.java*

#### 8.8.1.3. iOS: MFI and HomeKit

iOS does not provide developers with the ability to control the mobile device's WiFi configuration. However, devices built with Ayla's firmware support can support Apple MFI and be configured to use HomeKit. Ayla does provide HomeKit registration and accessories support in AMAP: a subscription based white-label application framework designed to help OEMs get products to market quickly.

#### 8.8.1.4. iOS: Manual configuration

If MFI is not supported on the IoT device, the user of the mobile app will need to enter the Settings app in iOS, connect to the IoT device's open AP and can configure the SSID and password for the home network via a web page provided by the device. The mobile app will launch based on the matching customer URL setting in both the application (Xcode:TARGETS/<myApp>/Info/URL types) and the IoT device (wifi setup_ios_app <name_of_the_app>).

#### 8.8.1.5. Aura WiFi Setup Examples (iOS)

*SetupViewController.swift*

## 8.9. Device Registration

Once a device is on the user's local network and has access to the internet, it still needs to be associated with the user's account. The Ayla Cloud Service needs to ensure device ownership before it will register a device with to user.

Several methods of registration are available. Each type of device supports only one registration method, which is decided upon at the time of manufacture. The requirements for each registration method vary, including proof of ownership, and the chosen method will depend on the physical characteristics of the device. A device's registration window is limited, for security reasons, so it is recommended to register the device within a few minutes after connecting with the Ayla Cloud Service.

The *AylaRegistration* class is provided by the SDK to assist with device registration.

### 8.9.1. Registration Candidates

A device must first be known to the mobile application before it can be registered to the user's account. *AylaRegistration.fetchCandidate* method may be used to find a registration candidate on the same local network as the mobile device. Once discovered, the *AylaRegistrationCandidate* returned from this method may be registered with the Ayla Cloud Service via a call to *AylaRegistration.registerCandidate*.

#### *8.9.1.1. Same-LAN*

Same-LAN registration requires that the mobile device and the device to be registered are on the same local network. The mobile device connects to the device to be registered and requests it's registration token, which is passed to the registration method in the mobile SDK. The Ayla Cloud Service confirms that the token from the mobile app and the token sent to the device are the same, and adds the device to the user's account.

#### *8.9.1.2. Button-Push*

Similar to Same-LAN registration, with the exception that the device requires physical intervention to begin the registration, such as pressing a button or specific key combination. Otherwise the behavior for Button-Push registration is the same as Same-LAN.

#### *8.9.1.3. AP-Mode*

AP-Mode registration is used in conjunction with WiFi Setup. The registration token is created by the mobile application and delivered to the device during the Setup process. Next, the device passes the registration token to the device, which forwards it on the Ayla Cloud Service. The application completes registration of the device by including the self generated registration token as proof ownership in the registration API call.

#### 8.9.1.4. Display

Display registration is used with devices that have a physical display. The registration token is presented to the user via the display and entered manually into the application to be passed to the Ayla Cloud Service.

#### 8.9.1.5. DSN

The Device Serial Number (DSN) is provided to the mobile application by the end user. The number is generally provided on a sticker or other physical medium connected to the device.

#### 8.9.1.6. Aura Registration Examples

Examples of device registration may be found in Aura:

***iOS**: RegistrationViewController.swift*

***Android**: RegistrationFragment.java / SetupWizardFragment.java*

## 9. Contacts

The Ayla Cloud Platform uses Contacts to notify users of events. An *AylaContact* object may be created by the mobile application using the *AylaSessionManager* API `createContact`.

Contacts may contain a first and last name, email address, phone number and country code (both required for SMS notifications) as well as additional information that may be desirable to save on the service.

The list of contacts known to the user's account may be retrieved from the service via the *AylaSessionManager's* `fetchContacts` API.

Contacts are mainly used to indicate the method for notification for a schedule, device notification or property notification, addressed in the next section.

## 10. Notifications

The Ayla Cloud Service provides several methods to notify your users of events. These include email, SMS (text message), and push notifications (APNS for iOS, GCM for Android, or Baidu for either platform).

There are two basic types of notifications: Device notifications, which notify about the device losing or restoring connection to the Ayla service, and Property notifications, which can be configured to notify when a property meets application-defined criteria.

Notifications are divided into two components: the Trigger, which is a condition that must be met, and a set of Applications, which are executed when the Trigger's conditions are met. Multiple applications may be attached to a Trigger, and multiple Triggers may be attached to a Device or Property.

## 10.1. Device Notifications

Device notifications are generally installed by the application when a device is registered to the user's account. There are several possible notification types that may be used to create a Trigger for a Device:

**Table 10 Device Notifications**

| Notification Type | Description |
|---|---|
| on_connect | Fired when the device first connects to the service. |
| ip_change | Fired when the IP address for the device changes |
| on_connection_lost | Fired when the device's connection to the cloud has been lost |
| on_connection_restore | Fired when the device has restored connection to the cloud |

The on_connection_lost and on_connection_restore notification types are the most useful, and are often set up to email the account holder when devices connect to or disconnect from the cloud service.

The process for creating a device notification is in two steps. First, the Notification (Trigger) must be created. This is done by creating a new AylaDeviceNotification object, setting the notification type, and passing it to the device's createNotification() method, which will return a new AylaDeviceNotification object.

Once the new AylaDeviceNotification object has been returned, the application creates one or more AylaDeviceNotificationApp objects, configures the object for email, SMS or push, provides the contact to be notified, and calls createNotificationApp on the returned AylaDeviceNotification object.

## 10.2. Property Notifications

Property notifications are similar to device notifications except that the triggers are based on property values rather than changes to the device's network state.

Like device notifications, property notifications require two steps to create: one to create the notification itself (the trigger conditions), and another to add applications that notify when the trigger condition is met.

Multiple triggers (conditions) may be set on a single property, and multiple apps (notification events) may be created on a single trigger.

To create a notification for a property, first create a new AylaPropertyTrigger object. The trigger object may be configured with one of three comparison types shown in Table 11.

**Table 11 Property Trigger objects**

| Property Trigger object | Results |
|---|---|
| on_change | Fire whenever the property value changes |
| compare_absolute | Fire when the comparison indicated by compareType has been met |
| always | Fire whenever a new datapoint is created, whether or not the value is different |

For the compare_absolute type, setCompareType must also be called with one of "==", "<=", ">=", "<" or ">". The value to be compared against must also be set via a call to *setValue*. When the value of the property compared to the value of the trigger meets the condition of the operator, then the trigger is fired and the apps are run to notify users of the event.

Once the trigger has been created via a call to the AylaProperty's createTrigger method, the returned trigger may be used to create AylaPropertyTriggerApp objects via calls to the trigger's createApp method.

#### 10.2.1.1. Aura Property Notification Examples

Examples of creating property triggers and apps can be found in Aura:

***iOS**: PropertyNotificationDetailsViewController.swift*

***Android**: PropertyNotificationFragment.java*

# 11. Schedules

Properties may be updated via a schedule that can be configured by the mobile application. A schedule is associated with an *AylaDevice* and can be configured to run one or more "schedule actions".

Schedules reside on the devices themselves. The number of schedules and their names are determined at the time of manufacture. Because of this, it is strongly recommended to create schedule templates for your devices that reflect the schedules residing on the physical devices. If this is done, applications will not need to create or destroy schedule objects- they will already exist and can be updated instead.

If schedules are not added to the default device template, applications will need to know the number and names of the schedules on the device and create AylaSchedule objects for each device manually.

The set of schedules for a device may be fetched by calling the *AylaDevice's fetchSchedules* method.

Schedules may be updated by obtaining an *AylaSchedule* object (either by creating one or fetching from the device), modifying it and passing it to the *AylaDevice's updateSchedule* method.

## 11.1. Configuring a schedule

AylaSchedule objects are highly configurable.Table 12 lists AyalSchedule objects.

**Table 12 Schedule objects**

| AylaSchedule object | Explanation |
|---|---|
| active | True if the schedule should be run, false if it should be disabled |
| name | Name of the schedule. Must match the name of the schedule on the device. |
| displayName | User-configurable friendly name for the schedule |
| utc | True if times in the schedule are in UTC. False if times in the schedule are in the device's local time zone. |
| startDate | Date the schedule begins. No actions will be taken prior to this date. May be null for a schedule that takes effect immediately. |
| endDate | Date the schedule ends. No actions will be taken after this date. May be null for a schedule that does not end. |
| startTimeEachDay | Time of day the schedule "starts". Actions configured to fire "at_start" will fire at this time each day. |
| endTimeEachDay | Time of day the schedule "ends". Actions configured to fire "at_end" will fire at this time each day. |
| daysOfWeek | Array of integers indicating which days of the week the schedule is active. Days start with Sunday with a value of 1. Optional field: all days are active if this is not specified. |
| daysOfMonth | Array of integers (1-31) indicating which days of the month the schedule should be active on. May be null: all days are considered active if this field is not present. |
| monthsOfYear | Array of integers (1-12) indicating which months of the year the schedule should be active on. May be null: all months are considered active if this field is not present. |

| | |
|---|---|
| dayOccurOfMonth | Array of integers (1-6) indicating a week in the month for which a specified dayOfWeek the schedule will be active. For example: when daysOfWeek is 2 and dayOccurOfMonth is 2, the schedule is valid on the 2nd Monday of the Month. When set to 6, the schedule is always valid on the last day of the month. |
| duration | If endTimeEachDay is omitted, this field may specify the number of seconds after the startTimeEachDay the schedule should be ended. |
| interval | If specified, the time in seconds the schedule should cycle. |
| fixedActions | If set to true (decided at manufacturing time and set in the device template), disallows schedule actions from being created or destroyed. Applications may enable or disable actions, but cannot add or remove them. |

### 11.1.1. Schedule Actions

Schedules have a start and an end, and can be configured to execute actions at either time. Schedule actions may be fixed, when the *fixedActions* field is set to true. In this case, the actions for each schedule are defined in the device template and may not be added to or removed by the application, though they may be enabled or disabled.

Schedules without fixed actions require the application to create, update or delete actions as needed.

Schedule actions are used to create a datapoint (modify a value) for a particular property on a device when the schedule event fires. The schedule action is configured with the fields shown in Table 13.

**Table 13 Schedule Action fields**

| Field | Explanation |
|---|---|
| name | Name of the property to be updated. This field is required |
| value | Value the property should be set to when the |

| | |
|---|---|
| | schedule fires. This field is required |
| baseType | Base type of the field, must match the base type of the AylaProperty being updated by this action. Required. |
| atStart<br><br>atEnd<br><br>inRange | Exactly one of these three fields must be set to true and the others false. This indicates whether the action should be fire at the start of the schedule, the end of the schedule, or within the start and end of the schedule (for example if a device comes online in the middle of a schedule, *inRange* will cause the device to execute the actions while *atStart* will not) |
| type | Type of action. Currently only SchedulePropertyAction is supported, and will be used by default if this field is omitted. |

#### 11.1.1.1. Aura Schedule Examples

Examples of creating schedules and schedule actions may be found in Aura:

***iOS**: ScheduleEditorTableViewController.swift*

***Android**: ScheduleFragment.java*

## 12. How do I…

This section reviews common scenarios that will come up during mobile app development using the Ayla Mobile SDK and discusses how to deal with them. Many examples will also provide references into *Aura*, the sample application included with the Ayla SDK.

#### 12.1.1.1. Sign up a new user?

Method: *AylaLoginManager – signUp*

Required information: *AylaUser* object with email, password and first name at a minimum filled out; *AylaEmailTemplate* with the template ID or HTML for the email form that is mailed out.

Returned on success: a new *AylaUser* object

Note: A valid login session is not required to use this method.

**Aura Examples:**

Android: *LoginActivity.java*

iOS: *LoginViewController.swift*

#### *12.1.1.2. Change the user's password?*

Method: AylaLoginManager – requestPasswordReset

Required information: User's email address

Returned on success: nothing, but an email will be sent to the specified address

**Note:** An email will be sent to the user with a link to reset the password. The actual link provided is determined by the email template, and may be configured to launch the mobile application if desired.

**Aura Examples:**

Android: *LoginActivity.java*

iOS: *LoginViewController.swift*