AMAP Architecture Specification

v0.6 2015.03.16

# Overview

The AMAP SDK is a software development kit created by Ayla Networks to aid in the development of mobile applications utilizing the Ayla service network.

The goal of the AMAP SDK is to make it as easy as possible to create iOS and Android applications that use the Ayla network of connected devices. The app can be easily customized for look and feel as well as supporting custom devices with minimal effort.

# Architecture goals

The primary goal of producing this SDK is to provide a starting point for developers to produce iOS and Android applications with minimal effort. While the types and capabilities of devices connected to an Ayla network can vary widely, there are many tasks that need to be performed on all of them:

## AMAP Functionality

* User authentication / login
* New user account creation
* Account updates (edit profile)
* Device discovery / setup
* Log out / cache clearing
* Polling the list of connected devices
* Polling the state of each connected device
* Adding / removing devices from the collection
* Enabling / disabling LAN mode as appropriate
* Device Groups
* Push Notification handling
* Timers and Schedules
* Device management UI
  + Display a list of all connected devices
  + Display a filtered list of all connected devices (favorites, etc.)
  + Display details of a user-selected device
  + UI to remove a device
  + UI to add a device
  + UI to remove a device
  + UI to change the state of a device
* Handling connectivity changes (app-modal message when network is not accessible)

## Environment

Setting up the development environment should be a simple task. To achieve this, dependencies should be managed by the build system (Gradle for Android) or by an external tool such as CocoaPods (iOS).

The Android version of the app should support both Eclipse and Android Studio so as not to force development in a certain environment. It should support Android version 4.2+ (JellyBean).

The iOS version of the app should support Xcode 6.1 and iOS 7 at a minimum.

# Software Architecture

To facilitate ease of development, the AMAP SDK will be distributed as a functional application built to use The sample application will use / derive from the classes provided in the AMAP SDK as an example of how a developer might use the SDK to implement a client’s own devices.

The system can be divided into several components:

* Session Manager
  + Store all configuration parameters
  + Handle user login
    - Local authentication (LAN mode only, property updates)
  + Handle log out / cache cleanup
  + Host the DeviceManager
* Device Manager
  + Fetch / poll list of devices
  + Enter LAN mode if appropriate
  + Handle adding devices
  + Handle removing devices
  + Handle device groupings (favorites, etc.)
    - Groups, bindings, scenes for Zigbee
  + Poll device statuses for changes
  + Triggers / Trigger Apps
  + Schedule support
* Device Object
  + Base object class, meant to be derived from
  + Contains AylaDevice object
  + Returns list of properties to be polled by Device Manager
  + Provides UI elements for list views, grid views and detail views
  + Derived classes can support additional functionality / properties / etc
* UI
  + Login screen
  + Sign-up screen
  + Edit Profile screen
  + Device list
  + Groups list
  + Add device
  + Remove device
  + Device details page
  + Schedules page

## Object Details

The following sections describe in detail the functionality and interfaces of the system objects. For ease of reading, pseudocode is used to define APIs or notifications in a platform-independent manner.

### Notifications

On Android platforms, objects can be notified by implementing a listener interface and registering themselves with the appropriate system object.

On iOS devices, objects can be notified by registering for notifications via the NSNotificationCenter.

## Session Manager

The Session Manager is a static / singleton object used to initiate a login session.

#### Interfaces

void startSession(SessionParameters params)

void startOAuthSession(Message)

void stopSession()

DeviceManager deviceManager()

void setParameters(SessionParameters)

SessionParameters sessionParameters()

void clearSavedUser()

getInstance()

#### Notifications

void loginStateChanged(bool loggedIn, AylaUser aylaUser)

void reachabilityChanged(int reachabilityState)

void lanModeChanged(bool lanModeEnabled)

## Session Parameters

This class contains configuration information required to start a session

#### Members

context (Android only, needed for resources, etc.)

deviceSsidRegex

appVersion

pushNotificationSenderId

appId

appSecret

username

password

enableLANMode

serviceType (= AML\_STAGING\_SERVICE)

loggingLevel (=AML\_LOGGING\_LEVEL\_ERROR)

DeviceCreator deviceCreator

registrationEmailTempateId

registrationEmailSubject

registrationEmailBodyHTML

#### Interfaces

## DeviceCreator

Device deviceForAylaDevice(AylaDevice aylaDevice)

The DeviceCreator object implements deviceForAylaDevice. This method is called by the DeviceManager to create user-defined Device objects for each AylaDevice returned by the service. This allows the user to define custom device classes that will be managed by the application framework.

List<Class> getSupportedDeviceClasses()

This method returns a list of Device-derived classes supported by the DeviceCreator. This list is used during the registration flow to provide the user with a list of possible devices to find, and sets the appropriate registration type (push-button, same-LAN, etc.) when selected.

ViewHolder viewHolderForViewType(int viewType)

This method returns a ViewHolder for the appropriate type. This method is called when displaying a list of devices in a RecyclerView (Android only).

## Device Manager

The Device Manager is created by and can be obtained from the Session Manager once login has successfully completed.

#### Methods

Gateway getGatewayDevice()

Array<Device> deviceList()

Array<Device> getFilteredDeviceList(filter function)

Boolean isLANModeEnabled()

void setDeviceListPollInterval(int timeInMs)

void setDeviceStatusPollInterval(int timeInMs)

// Start and stop polling the devices

void startPolling()

void stopPolling()

// Add / remove listeners for changes in the device list

// or the status of a device

void addDeviceStatusListener(listener)

void removeDeviceStatusListeners(listener)

void addDeviceListListener(listener)

void removeDeviceListListener(listener)

#### Notifications

void deviceListChanged()

void deviceStatusChanged(Device changedDevice)

## Device

The Device object is a base class representing the common properties of a device connected to the network. Implementers should create new class objects derived from the Device class that contain device-specific information and functionality.

Creation of device objects are handled by the deviceCreator method passed in to the Session Manager via the Session Parameters. This allows the framework to create and manage devices of the object type desired by the implementer.

#### Methods

AylaDevice getDevice()

void updateStatus()

AylaProperty getProperty(String propertyName)

// UI methods

View getListItemView(Context context, View convertView, ViewGroup parent)

View getGridItemView(Context context, View convertView, ViewGroup parent)

Fragment getDetailsFragment(Context context)

String toString()

String getDeviceState()

String deviceTypeName()

String registrationType()

ArrayList<String> getPropertyNames()

// UI methods

Drawable getDeviceDrawable()

Fragment getDetailsFragment()

Void bindViewHolder(holder)

Implementers of the Device class should pay particular attention to these methods:

updateStatus()

This method is responsible for fetching information about the device’s status. It will be called whenever the device manager status timer is called. The default implementation fetches properties (returned from getPropertyNames()). Custom devices may require additional functionality, which should be implemented in an override of this method.

toString()

This method is called to determine the text displayed in the default list view or grid view. The default implementation returns the friendly name of the device.

getDeviceState()

This method is called along with toString to provide additional information about the state of the device, such as “ON”, “OFF”, “Open”, “Closed”, etc. It is optional, and defaults to an empty string.

getPropertyNames()

This method should be overridden to add properties to be fetched during device status updates.

getDeviceDrawable()

This method should be overridden to return a Drawable (Android only) that represents the device, such as the image of a plug, or switch, or door sensor

getDetailsFragment()

This method should be overridden to return a Fragment that should be displayed when the user taps on a Device item in a list. The default implementation shows the image of the device (from getDeviceDrawable()) as well as a list of the properties the device has and their values.

bindViewHolder()

This method should be overridden in devices that use a custom ViewHolder returned by the DeviceCreator’s viewHolderForViewType() method. The method should bind the views held by the ViewHolder with data from the device object, such as the device name, image, any controls or buttons in the view, etc.

## Gateway : Device

The Gateway object is derived from the Device object, and contains additional interfaces used to query gateway-owned devices or to configure the gateway.

#### Interfaces

Array<Device> getNodes()

# Steps to Building a Custom App

## Themes

#### Android

The application color scheme can easily be set by editing the colors.xml file and changing the following colors:

* app\_theme\_accent
* app\_theme\_primary\_light
* app\_theme\_primary\_medium\_light
* app\_theme\_primary
* app\_theme\_primary\_medium\_dark
* app\_theme\_primary\_dark

These colors are used throughout the application, and should be compatible with each other. If a more detailed level of customization is desired, additional color elements (which generally default to using one of the above colors) can be individually set.

The application icons found in the various drawable\_xxx\_dpi folders (ic\_launcher.png) should be updated to use the new application icon.

## Devices

AMAP initially has support for two device types: The Ayla EVB, and the smart plug. There are two Device-derived classes in the project, DevkitDevice and SwitchedDevice, that override the framework’s Device class to provide functionality for those specific devices.

Using these classes as a guide, implement your own classes derived from the Device class to provide the desired functionality for your devices:

* Create classes derived from Framework.Device for each device type you wish to implement.
* Implement the following methods in each Device class:
  + getPropertyNames()
    - Call super.getPropertyNames() and add your own properties to the list before returning it
  + deviceTypeName()
    - Return the friendly name of your device, such as “Smart Plug”
  + getDeviceDrawable()
    - Return a Drawable to represent your device in various contexts
  + registrationType()
    - Return your device’s preferred registration type
  + getItemViewType()
    - Return an integer unique to your device or set of devices using the same user interface for display in a list or grid
  + bindViewHolder()
    - Override this if you have a custom ViewHolder for your device
  + If your device supports schedules on one or more of its properties, implement getSchedulablePropertyNames() and friendlyNameForrPropertyName(). This will allow the scheduler to know what properties to present to the user as options to enable / disable with schedules.
  + To display additional elements in the CardView UI, create your own CardView-derived classes
* Create a class derived from DeviceCreator and implement the following methods:
  + deviceForAylaDevice(AylaDevice)
    - Return a newly-created Device object for the supplied AylaDevice. This is where your custom classes are created in response to receiving a list of devices from the server
  + viewHolderForViewType(viewType)
    - Return the appropriate ViewHolder for the supplied ViewType. This allows for different views / holders to be created within a single RecyclerView. The viewType parameter is set based on the value returned from the device’s getItemViewType() method. This is where the appropriate ViewHolder can be created for each device type.
  + getSupportedDeviceClasses()
    - Return a list of Class objects, one for each custom Device type your application supports.

## SessionParameters

Create a SessionManager.SessionParameters object and fill out the fields as appropriate to your application.

Update MainActivity to use your SessionParameters object instead of the example SessionParameters.

Once these tasks are done, your app should be ready to run.

# Building AMAP

## Android Studio

git clone https://github.com/AylaNetworks/Agile\_Link\_Android.git

cd Agile\_Link\_Android

mkdir libraries

cd libraries

git clone https://github.com/AylaNetworks/Android\_AylaLibrary.git

cd Android\_AylaLibrary

git checkout -b zigbee\_lan\_merge3 origin/zigbee\_lan\_merge3

Then open Android Studio and select "Open existing project"

Open build.gradle in the project root directory.

## Eclipse

[TBD]

## iOS

[TBD]