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 is built with Android Studio and uses a script to set up the appropriate frameworks when starting a new project. The scripts can be found in the gradle_scripts directory for Mac OS and Windows hosts.

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:

- AMAP Core
    - Initializes the Ayla SDK
    - Stores application configuration parameters
    - Provides access to SDK components
    - Manages user account settings and details
    - Handle user login
        - Local authentication (LAN mode only, property updates)
    - Handle log out / cache cleanup
    - Hosts the various managers used by AMAP and the Ayla SDK

- AylaDeviceManager (Ayla SDK)
    - Fetch / poll list of devices
    - Enter LAN mode if appropriate
    - Handle adding devices
    - Handle removing devices
    - Handle device groupings (favorites, etc.)
    - Poll device statuses for changes
    - DSS device updates (BETA)
    - Triggers / Trigger Apps
    - Schedule support

- ViewModel
  - Represents a device as presented to the user
  - Contains an 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
  - ViewModels are the means for AMAP to present a device within the user interface

- 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.

AMAPCore

AMAPCore is a static / singleton object used to initialize the AMAP engine, and provides methods to sign in a user (start a session), stop a session (sign out a user) as well as helper objects to manage account settings,

Interfaces

```
void startSession()
void startOAuthSession()
void stopSession()
```

```
AylaDeviceManager getDeviceManager()
AylaSessionManager getSessionManager()
getInstance()
```

Session Notifications

Listeners may be added to the AylaSessionManager to receive notifications of changes to the session state:

```
void sessionClosed()
void authorizationRefreshed()
```

Session Parameters

This class contains configuration information required to start a session. Details regarding these members may be found in the JavaDoc or AppleDoc SDK documentation.

Members

```
context (Android only, needed for resources, etc.)
sessionName
deviceSsidRegex
appVersion
pushNotificationType
pushNotificationSenderId
appId
appSecret
username
password
enableLANMode
allowLANLogin
allowDSS
ssoManager
ssoLogin
serviceType (= AML_STAGING_SERVICE)
loggingLevel (=LogLevel.Error)
fileLoggingLevel (=LogLevel.Error)
viewModelProvider
defaultNetworkTimeoutMs
registrationEmailTempateId
registrationEmailSubject
registrationEmailBodyHTML
```

Interfaces

AMAPViewModelProvider

```
ViewModel viewModelForDevice(AylaDevice aylaDevice)
String[] getManagedPropertyNames(AylaDevice aylaDevice)
```

The ViewModelProvider interface contains two methods, *viewModelForDevice* and *getManagedPropertyNames*. These methods must be implemented by a user-created object that is provided to the AMAPCore via the SystemSettings field, viewModelProvider, when AMAP is first initialized.

The ViewModelProvider implementation should return a ViewModel-derived class for the given AylaDevice. The ViewModel class will contain implementations that allow the SDK to find out more information about the device, such as which fragment to display for schedules or device details, what view the device should use when displayed in a grid or list, etc.

The getManagedPropertyNames method should return an array of strings containing the set of property names that the application wishes to be managed by the Ayla SDK. Properties that are managed by the SDK will be guaranteed to always be kept up-to-date. Limiting the set of managed properties to those properties that change frequently or are often updated by the user will improve the overall performance of the system, so it is important to limit this list of properties to those that are used most frequently or are critical in nature.

AMAP provides a default ViewModelProvider called AMAPViewModelProvider. This provider implements the ViewModelProvider interface as well as providing an additional method, getSupportedDeviceClasses, that is used to help users choose from a set of supported devices when performing device setup. AMAP applications should modify or override this class to return appropriate device classes that are supported by the application.

AylaDeviceManager

The Device Manager can be obtained from AMAPCore once login has successfully completed.

The AylaDeviceManager is an Ayla SDK component that contains the list of devices registered to the user as well as a notification system that allows application developers to know when the list of devices changes in any way.

See the Ayla Mobile SDK developer's guide and AylaDeviceManager Javadoc / Appledoc for more information.

ViewModel

The ViewModel 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 ViewModel class that contain device-specific information and functionality.

Creation of device objects are handled by the viewModelProvider 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:

`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.

GenericGateway : ViewModel
The Gateway object is derived from the ViewModel 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

Consult the README.md file for instructions on how to build AMAP.