1. Mobile Foundry

Architecture Overview

Draft 0.1
April 12, 2017
Brian King <brian@aylanetworks.com>

Table of Contents

Overview

The Internet of Things comprises three distinct areas: Cloud services, devices and mobile applications. While all three of these areas are critical to a successful IoT platform, the mobile application is the primary part of this infrastructure that is seen and interacted with by users. Regardless of the “thing” being controlled, the user’s experience with the mobile application will be the strongest factor in determining the perceived quality of the product as a whole.

Writing quality IoT mobile applications is not an easy process. In addition to creating the user interface and logic for routine operations such as signing in, account management, password resets, etc., an IoT application needs to handle device discovery and registration, device control, fast and reliable network communications with both the cloud service and directly with devices themselves, schedules, offline usage, push notifications, and many more common IoT tasks.

The Ayla Mobile SDK provides much of the core functionality for these tasks, but does not include application-level code or user interface elements. Designing a mobile application that takes advantage of the Ayla Mobile SDK’s core functionality is still a large undertaking.

The Agile Mobile Application Platform, or AMAP, provides the application structure and all of the user interface components required to support the rich feature set of the Ayla Mobile SDK on both iOS and Android platforms. AMAP allows developers to get up and running very quickly with a fully-featured Ayla IoT application, leaving time for fine-tuning of the look and feel without needing to worry about the underlying application structure.

Architecture Goals

AMAP was designed to make developing mobile Ayla IoT applications easier. Over the years, Ayla has worked with many teams of mobile developers and has found that most IoT applications require the same basic set of functionality and support code. Tasks such as user account management, device management and display, device control, push notifications, etc. are all common to Ayla IoT apps, and require quite a bit of support and user interface code to be written.

While these areas are not necessarily complex to implement, they are often time-consuming and are mostly identical in each application aside from the overall look and feel. AMAP provides solutions for most common IoT functionality, so rather than re-inventing the wheel, developers can simply define the navigation flow, modify the color scheme and other user interface characteristics as desired and with very little coding create applications for both iOS and Android.

Supported Features

AMAP provides built-in support for the following common tasks:

User sign-up	UI for entering account information Callback into application to verify (from email link)
Forgot password link	Send request to cloud Callback into application to verify (from email link)
User sign-in	Username / password Google Sign-In Facebook Sign-In Baidu Sign-In (China) OAuth skeleton (requires OAuth details to be provided)
WiFi Setup	Setup process for devices that join a WiFi network
Device registration	Associate devices with the user account
Device List Display	Show registered devices in various formats (lists, swiped screens, etc.)
Device Details display	Show details about a registered device (DSN, IP address, name, model, etc.)
Device Control	Controls for common IoT device properties: Temperature / Thermostat On / Off switches Color pickers Dimmable light controllers
Device unegistration	Remove a device from the user's account
Device sharing	Share a device with another Ayla account with read / write or read-only access
Contact management	Manage an address book of contacts for sharing or notifications
Device notifications	Set up notifications when certain criteria are met. Can notify via email, SMS or push notifications
Location-based Automations	Control devices based on your location. Supports geofences (GPS location / radius) as well as BLE Beacons (iBeacon, Eddystone)
Fingerprint authentication	Fingerprint-based authentication on supported devices in lieu of username / password

Depending on the nature of the device and application, some or all of these components may be included in the mobile applications. Whether components are included in the application, and where they are presented in the application flow, are determined by the AMAP configuration file discussed in detail below.

AMAP Configuration

Mobile IoT applications vary widely in complexity. A simple application to control a wall outlet might be quite different from a complex application used to control all aspects of home automation from door sensors to colored lighting. As each application has a different set of requirements, AMAP allows developers to define which components to include as well as how they should be presented in a single configuration file that is shared between iOS and Android.

The AMAP configuration file defines the application configuration parameters, such as application ID, application secret, and supported device models. Additionally, the file describes the overall application flow from the sign-in screen to the main screen to menus and other navigation tools. With the exact same Swift or Java code, different configuration files will yield vastly different applications. The same code base with different configuration files can produce a simple application for controlling a single light, or a complex menu-driven application used to control a whole-home IoT solution.

The AMAP configuration file is broken up into several sections: The Root section, containing general application parameters; the SDK section, containing SDK-specific initialization parameters; the Devices section, specifying details about devices supported by the application; and the User Interface section describing the various screens, controls and application flow.

Root Section

The root of the configuration file contains the following objects:

version	Version of the configuration. This may be used by the application to ensure the version of the configuration matches an expected version, but is unused internally by the AMAP platform
name	Name for the configuration. Also not used internally by the AMAP platform, this field provides a way for the application to identify the configuration it was built with.
applicationIdentifier	Identifier used for the application. On Android, this is used as the application ID; on iOS, this is the bundle identifier.
packages	(Android-specific) array of package names used by classes within the configuration. These packages are pre-pended to class names to find them within the application.
devices	Array of Device objects with details about the types of devices the application supports, detailed later in this document
sdkConfig	Configuration parameters used to initialize the Ayla Mobile SDK. Details can be found later in this document.
userExperience	Configuration parameters used to construct the user interface for the application. This includes screen definitions, controls, colors,

	menus, etc. More details can be found later in this document.

SDK Section

The SDK object in the configuration file (sdkConfig) contains the following objects:

appId	Application ID, provided by Ayla Networks
appSecret	Application secret, provided by Ayla Networks
allowMobileDSS	True to allow mobile DSS, false otherwise
allowOfflineUse	True to allow the user to sign in without an Internet connection, false otherwise
serviceType	Dynamic | Field | Development | Staging | Demo
serviceLocation	USA | China | Europe
consoleLogLevel	Verbose | Debug | Info | Warning | Error | None
fileLogLevel	Verbose | Debug | Info | Warning | Error | None
xPlatformId	Unique string shared between applications to share data. Used as a prefix for datum keys by the platform.
defaultNetworkTimeoutMs	Default timeout for network operations. Defaults to 5000 ms.

Devices Section

The devices section in the configuration contains an array of objects, one for each type of device supported by the application:

class	Name of the class that should be instantiated within Ayla Device Manager. This class must exist in the project, and must derive from AylaDevice (Android) or ALDevice (iOS). This allows user-defined device objects to be created and managed by Ayla Device Manager
detailScreen	Name of the screen to be presented if the user wishes to view details about this device
managedProperties	Array of property details for this device, described in more detail below
model	The model (string) of the device, used to identify devices of this

	type
oemModel	The OEM model (string) of the device, used to identify devices of this type
scheduleScreen	Screen shown for this device when the user wishes to set a schedule on the device
icon	Reference to an image to be shown to represent this device
deviceControl	Reference to a Control in the configuration that can be used to represent a device. For example, a thermostat device might use a deviceControl that shows the current temperature, setpoint, and provides up / down controls to change the setpoint. This control would be used to represent the thermostat device in a list of devices.
ssidRegex	Regular expression used when searching for WiFi access points from a device. This allows the system to filter only for devices known to the system when scanning access points.

Managed Properties

Each device should provide a set of Property details that describe the various properties of the device that should be managed by the Ayla SDK, as well as additional hints to the AMAP platform that allow it to display an intelligent user interface to manipulate or display the property.

control	Optional field indicating the control that should be used for this property to display or modify its value. If not provided, the framework will use a generic control based on the property's base type.
name	Name of the property
notify	True if the property can be used for notifications, false otherwise
schedule	True if the property can be scheduled, false otherwise
actions[]	Array of actions available to this property. Actions are described in more detail in the next section.
roles[]	Array of strings used to identify properties with specific purposes in the application. For example, a control might look for a property with the "temperature_setpoing" role to discover a property used for setting temperature.

Actions

An action is a mapping of “something to do” to a property value. For example, a simple wall switch might have a property that controls the on / off state of the switch. This property might use 0 to indicate “off” and 1 to indicate “on”. The Actions array for this property might contain:

```
"actions": [
  {
    "name": "turn_off",
    "value": 0
  },
  {
    "name": "turn_on",
    "value": 1
  },
```

where the values “turn_off” and “turn_on” are entries in the localized String table of the application that can be presented to the user as options on this device.

A fan controller might have multiple settings for the same property:

```
"actions": [
  {
    "name": "fan_auto",
    "value": 0
  },
  {
    "name": "fan_silent",
    "value": 1
  },
  {
    "name": "fan_low",
    "value": 2
  },
  {
    "name": "fan_medium",
    "value": 3
  },
  {
    "name": "fan_high",
    "value": 4
  }
]
```

When AMAP presents a view to control a device, the Actions of the Managed Properties of that device are queried to provide the user with a complete set of options to control the device.

User Experience Section

The User Experience section of the configuration file defines the application flow, look and feel. AMAP applications use the User Experience section to determine which screens to load, if a menu should be created and populating its contents, determining navigation as well as defining the color scheme and user interface tweaks that make each application unique.

The User Experience section is broken up into several sub-sections:

Default Configuration

This section specifies the default configuration for the user interface. All UI elements will use the values specified in this section by default. Most visible components in the application can be customized further if desired by overriding values found in the default configuration with values defined in the component's own configuration section, described in more detail below.

The default configuration consists of the following fields:

windowBackgroundColor	Color reference to the default background color of screens
wantsTitleBar	True if a title bar should be shown, false otherwise
disableDrawerMenu	True if the application does not use a drawer menu, false otherwise
fontColor	Color reference to the default font color

Screens

The Screens section contains an array of Screen objects that are used by the application. Each screen listed in this section must have a name, which is application-defined, and a class, which must refer to an existing Java or Swift class in the project. Additionally screens may include a title string reference which is displayed in a menu or title bar, and a configuration section used to specify details about how the screen should be presented.

Screens are referenced in the configuration file and throughout the application by the screen name. The AMAP framework provides methods to find and instantiate screens within code, allowing custom application code to seamlessly work within the framework.

AMAP provides a large set of screens to choose from that handle many of the most common IoT tasks, including device setup and registration, sharing, contacts, sign-in, etc. While these screens are customizable to a degree via configuration settings, some applications will require custom UI or functionality that AMAP does not provide. Custom screens may be written by the application developer and can be used throughout AMAP like any of the built-in screens.

Screens can be passed information from the configuration file via name/value pairs stored in the optional "config" section of each screen.

Controls

UI controls such as switches, sliders and other widgets within an AMAP application are defined in this section. Controls, similar to screens, are referenced by name and can be provided additional information via a "config" section.

Screens that allow control of a device will check each property for a “control” reference, and present that control to the user for that property.

signInScreen / homeScreen

The User Experience section contains two variables, signInScreen and homeScreen, that let the framework know where to start the application flow when the application is started. If the user is signed in, the homeScreen will be the first screen to display. If the user has not yet signed in, the signInScreen will be displayed.

These variables must be set with a screen name for every AMAP application.

drawerMenu

If the application chooses to navigate via a drawer menu, this variable should be set with an array of screen names in the order they should be presented in the menu. Each screen listed in this section will be displayed in the drawer menu using the screen’s icon and title. Users may tap on any of the items in the drawer menu to launch the indicated screen.

defaultConfig

The defaultConfig section contains name / value pairs that are common to every screen. Individual screens may override any of these settings within their own “config” section. Those that do not will use the default values found in this section.

Some configuration options that may be set are:

windowBackground	Color reference of the window background
fontColor	Color of the font
wantsTitleBar	True if the screen should show the title bar
disableDrawerMenu	True if the drawer menu should be disabled while this screen is visible

Specific screens may also support additional configuration options. See the documentation for the screen for more details.

Building an App

Creating a new application using AMAP 6 is a relatively simple process. Clone the repositories for AMAP 6 for both iOS and Android and follow the instructions provided in the README file at the root of each project. Although the initial setup of each project is different, each project will use the same AMAP configuration file and maintain consistency between them through the configuration.

The Android repository can be found here:

https://github.com/AylaNetworks/Android_Sepia_Public

The iOS respository can be found here:

https://github.com/AylaNetworks/iOS_Sepia_Public

Customizing the Configuration

This section discusses various means of application customization using the AMAP configuration file. Applications using only the supplied screens and controls can entirely customize the look, feel and flow of the mobile application solely by changing the configuration file. Additional screens and controls may also be created and referenced within the configuration file, allowing full customization of the application while retaining the simplicity of configuration-driven development.

SDK Configuration

Each manufacturer is assigned an app ID and app secret by Ayla Networks to use in their mobile applications. These values must be supplied to the AMAP framework in the configuration file in the **sdkConfig** section. Update the configuration file (my_config.json, if you are following the previous example) with the values assigned to your company by Ayla:

```
"sdkConfig": {
   "appId": "sepiaapp-0dfc7900-id",
   "appSecret": "sepiaapp-0dfc7900-6s3Wn_kLZpbrV2ZomcCqK0EuIeQ",
   "allowMobileDSS": false,
   "allowOfflineUse": true,
   "serviceType": "Development",
   "serviceLocation": "USA",
   "consoleLogLevel": "Debug",
   "fileLogLevel": "Debug",
   "xPlatformID": "12345",
   "defaultNetworkTimeoutMs": 5000
 }
```

Devices

Each manufacturer has different types of IoT devices that will be supported by the mobile application. While the functionality of each device will vary, AMAP can provide intelligent handling of your devices by providing information to the engine through the device configuration settings.

Each type of device on the Ayla network is identified using a text field called **oemModel**. This field is used by the AMAP configuration to associate device types with screens and icons, as well as providing details about the device properties.

The "devices" section in the configuration file is an array of objects, each representing a single device type. Each device registered to the user is represented as an AylaDevice class in the application code. Custom classes may be created to provide additional functionality to AylaDevices, though this is not always necessary.

To add support for your device, create an entry in the devices section of the configuration file for each device your application wishes to support. The following fields are required:

class	The AylaDevice subclass that should be created for this device. If no custom class is used, AylaDevice itself may be the class
oemModel	Used to identify the device

Additionally, the device entry may define which screen is launched to show details about this particular device via the **detailScreen** field. Switches, for example, may wish to show the [TBD] screen to control the switch, while a thermostat may wish to show the [TBD] screen to control the temperature.

To show a custom screen for this device when selected, simply set the **detailScreen** field to the name of the screen to be shown for this device.

Some screens show an icon for the device. To customize the image shown to represent this device, update the **icon** field with the name of the icon resource to display (e.g. "ic_generic_device").

Managed Properties

Properties are used to control devices as well as gather information from them. The Ayla Mobile SDK is designed to manage sets of properties for devices, keeping them up to date and notifying the application when changes occur. Each device in the configuration file can specify the set of properties that the SDK should manage. These properties should contain only properties that are important to the application and are expected to change regularly. While all properties may be included in this set, the fewer properties there are to manage, the more efficient the management will be.

In addition to specifying the properties, each property may include additional hints as to how it is intended to be used. For example, a thermostat might have a "set_temperature" value, a "current_temperature" value, and an "on / off" value. The "current_temperature" property would be a good candidate for notifications (email / sms / push notification when a certain temperature is reached), but does not make any sense to be something scheduled. The "set_temperature", on the other hand, would be a good candidate for a schedule, but does not make much sense for notifications. The "On / off" value might not want either, but it should be controllable as a toggle button in the UI.

This particular device might be configured as follows:

```
    {
      "class": "AylaDevice",              // or your own custom class
      "detailScreen": "amap_thermostat", // or your own custom class
      "icon": "ic_ayla_evb",
      "managedProperties": [
        {
          "control": "decimal_input",     // Control for input / display
          "name": "temp_setpoint",        // Property name
          "notify": false,                // Do not enable notifications
          "schedule": true                // Enable schedules
        },
        {
          "name": "current_temp",         // Property name
          "control": "readonly_text",     // Control for display
          "notify": true,                 // Enable notifications
          "schedule": false               // Do not enable schedules
        },
        {
          "name": "on_off",               // Property name
          "notify": false,                // Do not enable notifications
          "schedule": true                // Enable schedules
          "actions": [
            {
              "name": "turn_off",
              "value": 0
            },
            {
              "name": "turn_on",
              "value": 1
            }

          ],
      },
      "name": "My Thermostat",            // Device name
      "oemModel": "tstat-1",              // OEM model
      "scheduleScreen": "schedule"        // Screen to show for scheduling
    },
```

User Experience

The **userExperience** section of the configuration file drives the look and feel of the application. On application startup, the AMAP engine first initializes the Ayla Mobile SDK and then presents either the **signInScreen** (if the user has not yet logged in) or the **homeScreen** (if the user is signed in).

SepiaActivity

On Android platforms, the entry point into the application is the main activity. AMAP apps should derive their main activity class from SepiaActivity, which drives the application flow.

SepiaActivity contains several helpful methods that can be leveraged by custom application code to allow new components to work within the AMAP framework:

`sharedInstance()` can be called to obtain an instance of the main activity from anywhere in the application `getConfig()` returns the loaded configuration object

`getSessionManager()` returns the active session manager

`getDeviceManager()` returns the active device manager

`loadConfiguration()` initializes the framework with the specified configuration file

`startApplication()` is called after loadConfiguration to start the application flow

`signInComplete()` is called when the user has successfully signed in

`resIdFromName()` (Android only) returns a resource ID from a resource name string

`navigateHome()` can be called to return the navigation to the home screen

`createSepiaScreen()` can be called to instantiate a Screen

`pushScreen()` can be called to push a Screen onto the navigation stack

`sessionClosed()` is called when the user signs out or their authorization expires

SepiaActivity is also a listener of the current session and device manager. Developers may override the listener methods to receive notifications of session or device events.

Screens

Screens represent a page in the UI flow. There are two required screens that must be defined: the **homeScreen** and the **signInScreen**. Additional screens are provided with AMAP, and may be added to the drawerMenu if desired, or navigated to from code via

Depending on the configuration, a drawer menu may be instantiated to allow many screens to be accessed from a single location. Additionally, screens may be configured with UI elements to launch other screens. For example, the DeviceList screen, included with AMAP, will launch the DeviceDetails screen for a device when tapped in the list.

In addition to the provided set of screens, AMAP supports custom Screen classes. Simply derive your class from SepiaScreen and reference it from the configuration file. Custom screens are no different than included screens- they all can be updated and modified via the configuration file.

Controls

Controls are most often used to control or present information from a device. Depending on the nature of the device, sliders, thermostats, switches or color pickers might be the best way to change or represent the state of the device.

Controls, like screens, can be customized via the configuration file. Controls have a field called “extras” that allows custom parameters to be passed to the control when it is created. Multiple versions of controls may be created with different configuration parameters if desired.

The configuration below defines two controls using the ACDeviceControl class, included with AMAP 6. One control uses one color scheme defined in the "extras" section, and the other uses a different color scheme:

```
"controls": [
  {
    "name": "ac_device_control",
    "class": "ACDeviceControl",
    "extras": {
      "bgHeat": "heatColor",
      "bgCool": "coolColor",
      "bgDehumidify": "dehumidifyColor",
      "bgOff": "offColor"
    }
  },
  {
    "name": "ac_alt_device_control",
    "class": "ACDeviceControl",
    "extras": {
      "bgHeat": "altHeatColor",
      "bgCool": "altCoolColor",
      "bgDehumidify": "altDehumidifyColor",
      "bgOff": "altOffColor"
    }
  },
```

Android:

Controls are first-class ViewGroup objects and may be used in XML layout files. To have a control within an XML layout reference a particular configuration, the XML attribute "configName" in the "app" namespace may be used to provide this information to the system. Below is an example of a Control within an XML layout file referencing the "temp_setpoint_control" in the configuration file:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              xmlns:app="http://schemas.android.com/apk/res-auto"
              android:orientation="vertical"
              android:layout_margin="12dp"
              android:layout_width="match_parent"
              android:layout_height="wrap_content">

        <com.aylanetworks.sepia.controls.TempSetpointControl
            android:id="@+id/temp_setpoint"
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:layout_margin="8dp"
            app:configName="temp_setpoint_control"
            android:layout_weight="1">
        </com.aylanetworks.sepia.controls.TempSetpointControl>
</LinearLayout>
```

Colors

The Colors section allows named colors to be defined. Having a central location for color definitions makes it easy to customize the entire application by changing the color values in this section. Colors are defined with a name and a value, which is a hexadecimal string akin to HTML color definitions:

```
"colors": [
  {"name": "primary_color", "value": "#006998"},
  {"name": "heatColor", "value": "#F76707"},
  {"name": "coolColor", "value": "#0093C7"},
  {"name": "offColor", "value": "#4a4a4a"},
  {"name": "dehumidifyColor", "value": "#00ccdd"}
]
```