Developer's Guide

Agile Mobile Applications Platform (AMAP)

Version: 0.3

Date Released: August 3, 2015

Document Number: AY006DMP0-v0.3

Ayla Networks

Table of Contents

1 Introduction 4

2 Getting AMAP 4

2.1 Building AMAP 7

2.1.1 Android *7*

2.1.2 iOS *7*

2.2 Generating Documentation 7

2.2.1 Android *7*

2.2.2 iOS *8*

3 AMAP Code Packages 9

4 Making it yours 11

4.1 Devices and Device Objects 12

4.2 Update Session Parameters 13

4.3 Create Device Creator 13

4.3.1 deviceForAylaDevice *13*

4.3.2 getSupportedDeviceClasses *14*

4.3.3 viewHolderForViewType *14*

4.4 Create Your Devices 14

4.5 Commonly Overridden Device Methods 15

4.5.1 ArrayList<String> getPropertyNames() *15*

4.5.2 String friendlyNameForPropertyName(String propertyName) *15*

4.5.3 String deviceTypeName() *15*

4.5.4 int registrationType() *15*

4.5.5 void postRegistration() *15*

4.5.6 Drawable getDeviceDrawable() *15*

4.5.7 int getItemViewType() *15*

4.5.8 void bindViewHolder(ViewHolder) *16*

4.5.9 String[] getSchedulablePropertyNames() *16*

4.5.10 String[] getNotifiablePropertyNames() *16*

5 Changing Colors and Style 16

5.1 Colors 16

5.2 List Style 20

5.3 Navigation Style 21

6 Tablet Support 21

7 Further Customization 21
7.1.1 Fragment getDetailsFragment() 22
7.1.2 Fragment getScheduleFragment() 22
7.1.3 Fragment getNotificationsFragment() 22
7.2 Fragments 23
8 Contacting Customer Support 26

1 Introduction

This document is intended for mobile application developers and is a guide to using the AMAP framework to create customized applications working with any types of devices.

The Internet of Things is a rapidly growing and exciting field in today's world. More and more smart devices are being created than ever, and in order for these devices to be smart, they need connectivity. Ayla Networks provides this connectivity in a flexible and extensible way, allowing manufacturers to focus on the details of their devices and not worry about creating an online ecosystem from the ground up.

Ayla Networks developed the Agile Mobile Applications Platform (AMAP) to facilitate rapid and easy development of mobile applications using the Ayla Cloud network.

AMAP completes the rapid development cycle by providing an application architecture that is easily customizable and extendable, allowing software developers to create fully functional mobile applications on iOS and Android with very little effort.

2 Getting AMAP

The first step towards building your own AMAP application is to start with a copy of the source code. AMAP resides in Ayla's GitHub repository at these locations:

Version	https	ssh
iOS	git@github.com:AylaNetworks/ AMAP_iOS_Public .git	https://github.com/AylaNetworks/ AMAP_iOS_Public .git
Android	git@github.com:AylaNetworks/ AMAP_Android_Public git	https://github.com/AylaNetworks/ AMAP_Android_Public.git

To ensure that your AMAP application is always up to date, it is highly recommended that you keep the clean AMAP code in a local branch and perform customizations in the master branch of your project. This way the AMAP code base can be easily updated from the AMAP repository and integrated into your application as AMAP is updated.

A recommended flow to begin your project is to clone the AMAP repository into a new directory for your application, create a branch for the AMAP codebase ("`amap_base`"), and

update the remotes so that "origin" (the primary remote) points to your own application repository, and "amap" points to the AMAP source repository.

Whenever a new version of the AMAP codebase is desired, it can be pulled into the amap_base branch from the AMAP remote, which can then be merged into the main application.

An example set of commands to begin a new Android application is listed below:

```
# First clone the AMAP repository into a directory of my choosing

git clone https://github.com/AylaNetworks/AMAP_Android_Public.git
my_application

cd my_application

# Next, rename the "master" branch to "amap_base"

git branch -m master amap_base

# Now rename the AMAP remote

git remote rename origin amap

# Set up the "amap_base" branch to track "master" on "amap"

git branch amap_base --set-upstream-to=amap/master

# Set up my own git remote for my application. This is where my code
changes will

# live, and where I will be pushing to.

git remote add origin
https://github.com/AylaNetworks/my_application.git

# Create my master branch for my application. This will start at the
same

# commit as "amap_base"

git checkout -b master

# Push my master branch to my git remote

git push origin master

# Make sure my master branch is set up to track origin/master
```

```
git branch master --set-upstream-to=origin/master
```

Now you should have a git repository with two remotes, one pointing to the AMAP source repository (`amap_base`) and another pointing to the repository for the application you are working on. To make sure your remotes are set up correctly, run "`git remote -v`". The output should look something like this:

```
 my_application $ git remote -v
amap  https://github.com/AylaNetworks/ AMAP_Android_Public.git (fetch)
amap  https://github.com/AylaNetworks/ AMAP_Android_Public.git (push)
origin     https://github.com/AylaNetworks/my_application.git (fetch)
origin     https://github.com/AylaNetworks/my_application.git (push)
```

When pushing changes to your application, you are pushing your master (or other) branches to the "`origin`" remote. When getting the latest code from AMAP, you are pulling it from the `amap` remote into the `amap_base` branch.

```
# I made changes to my application and want to push them to my remote
git push origin master
# I want to get the latest changes from AMAP and merge them in to my
application
git checkout amap_base
git pull amap master
# Now amap_base has the latest AMAP code. I need to merge it into my
own codebase
git checkout master
git merge amap_base
```

2.1 Building AMAP

Now that the repositories are set up, you can build the base AMAP application.

2.1.1 Android

Before you build the application for the first time, download the library code using existing scripts:

```
cd gradle_scripts/

gradle -q execTasks

cd ..
```

Open Android Studio and select "`Open Existing Project`", and select the "`settings.gradle`" file in the root of the project. At this point, you should be able to run the application on a device or emulator by tapping the play button in Android Studio.

2.1.2 iOS

Install iOS pod and then open AgileLink. xcworkspace to lunch in Xcode.

2.2 Generating Documentation

AMAP classes are documented inline. You can generate documentation as an HTML document (Android) or Apple Help DocSets (iOS).

2.2.1 Android

From the project root, run:

```
gradle generateDebugJavadoc
```

and then open the index found in:

```
app/build/docs/javadoc/index.html
```

2.2.2 iOS

Select the documentation project in Xcode and press the play button. The AMAP DocSets is installed and online help is available in the IDE and Help Viewer.

3 AMAP Code Packages

The AMAP project is divided into several packages.
The **Framework** package contains the heart of the AMAP application and should not be modified in your application. This package includes a set of classes used throughout the application to handle sign in / out, managing lists of devices and their states, user account information, and so on. The framework was designed to work with other classes in the system to provide maximum flexibility and customizability.

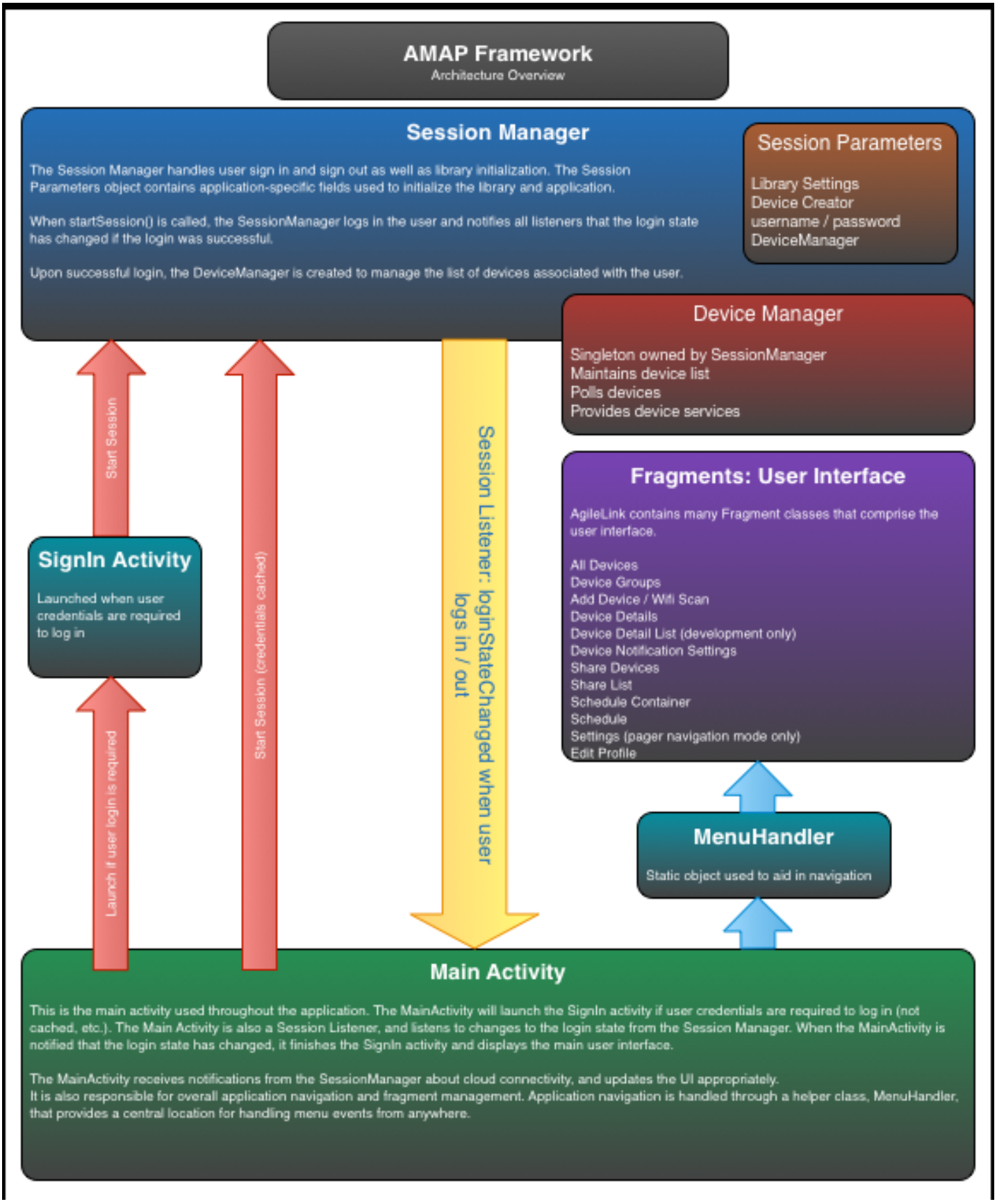
The **Fragments** package provides a complete set of user interface screens that may be used as-is, customized via changes in **UIConfig** or **colors.xml**, or replaced entirely. If you need to replace any fragments, make sure to preserve the navigation flow of the application and all screens are accessible.

The **Device** package contains a set of objects derived from the Framework package. The Device package also supports ViewHolder classes for displaying UI pages related to these devices. If desired, these devices may be used as base classes, or may be used as examples for creating your own Device-derived classes. The majority of work is done in an AMAP-based application in the Device package.

The **Controls** package contains custom controls that you may use in your application. Currently only a `ComboBox` exists on the Android platform, although more controls are expected in the future. If you create a control that might be useful to other applications, consider submitting a pull request with your control to the AMAP repository so that it can be included in other projects.

The AMAP package contains all of the above listed packages, as well as the activities and application object used in the app.

The following graphic shows the AMAP Architecture Overview.


AMAP Framework
Architecture Overview
Session Manager
The Session Manager handles user sign in and sign out as well as library initialization. The Session Parameters object contains application-specific fields used to initialize the library and application.
When startSession() is called, the SessionManager logs in the user and notifies all listeners that the login state has changed if the login was successful.
Upon successful login, the DeviceManager is created to manage the list of devices associated with the user.
Session Parameters
Library Settings
Device Creator
username / password
DeviceManager
Device Manager
Singleton owned by SessionManager
Maintains device list
Polls devices
Provides device services
Start Session
Start Session (credentials cached)
Session Listener: loginStateChanged when user logs in / out
Fragments: User Interface
AgileLink contains many Fragment classes that comprise the user interface.
All Devices
Device Groups
Add Device / Wifi Scan
Device Details
Device Detail List (development only)
Device Notification Settings
Share Devices
Share List
Schedule Container
Schedule
Settings (pager navigation mode only)
Edit Profile
SignIn Activity
Launched when user credentials are required to log in
Launch if user login is required
MenuHandler
Static object used to aid in navigation
Main Activity
This is the main activity used throughout the application. The MainActivity will launch the SignIn activity if user credentials are required to log in (not cached, etc.). The Main Activity is also a Session Listener, and listens to changes to the login state from the Session Manager. When the MainActivity is notified that the login state has changed, it finishes the SignIn activity and displays the main user interface.
The MainActivity receives notifications from the SessionManager about cloud connectivity, and updates the UI appropriately.
It is also responsible for overall application navigation and fragment management. Application navigation is handled through a helper class, MenuHandler, that provides a central location for handling menu events from anywhere.

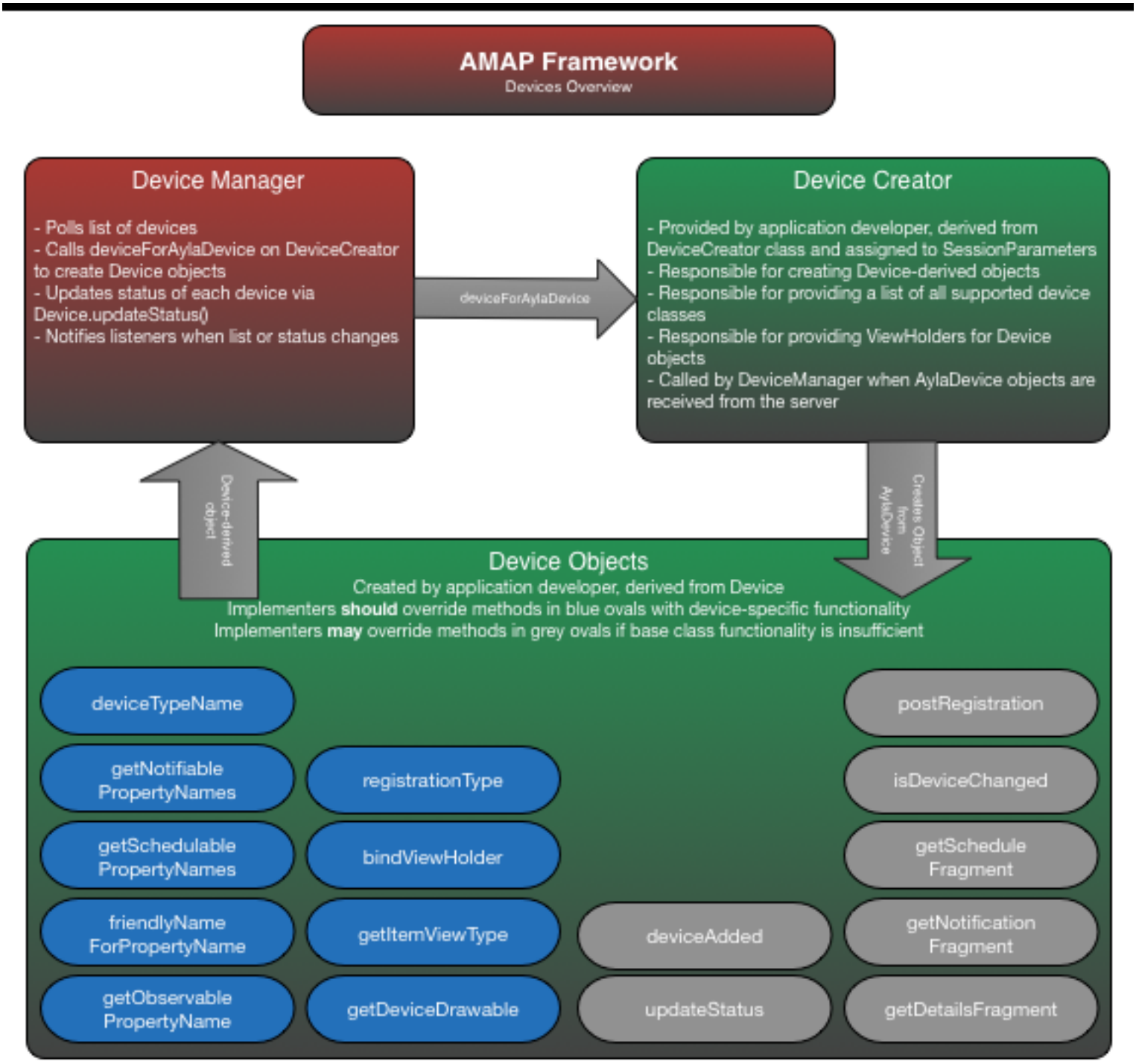
4 Making it yours

Now that the AMAP application builds, it's time to make it your own. The base AMAP application is set up to work with a small set of devices on the Ayla Developer network. If you have an account on the Ayla Developer network, you may run AMAP as-is and see your devices, register devices, and so on.

For each application, use your an app id and secret from the OEM Dashboard. These values can be found on the Ayla Dashboard site: https://dashboard-dev.aylanetworks.com/ for development apps, or https://dashboard.aylanetworks.com/ for production apps. Make sure you have your app id and secret ready before continuing development.

The following graphic provides an overview of devices and device cbjects.

4.1 Devices and Device Objects



4.2 Update Session Parameters

AMAP uses a class called the *SessionManager* to interact with the Ayla service. The *SessionManager* handles the initialization of the Ayla library as well as maintaining the configuration parameters for the application, which are stored in a *SessionParameters* object.

The SessionParameters for AMAP are set up in the `MainActivity` in the `getAppParameters()` method. You can modify these settings and any other settings that require modification using the `getAppParameters()` method. The default values for the `SessionParameters` are generally sufficient, but can be updated using this method.

The app id and secret are set in the SessionParameters and must be updated to reflect your own application's id and secret.

The other parameter that must be specific to your application is the *DeviceCreator*, which is described in the next section.

4.3 Create Device Creator

The *DeviceCreator* is an object implemented by the developer that is responsible for supplying custom Device-derived classes to the framework. AMAP contains its own device creator, `AMAPDeviceCreator`, which may be used as a reference, or modified to create your own devices.

Once your DeviceCreator-derived class is complete, create an instance of it and store it in the `SessionParameters.deviceCreator` member during initialization of the SessionParameters as outlined in the previous section.

The methods of DeviceCreator that need to be overridden by the developer are listed below.

4.3.1 deviceForAylaDevice

The `deviceForAylaDevice` method of the DeviceCreator is where the device creation actually happens. When the framework fetches device lists from the Ayla cloud, they are returned as Ayla Device objects, part of the Ayla library. The DeviceCreator is given an AylaDevice received from the cloud and constructs a Device-derived object, which is returned to the framework.

The `deviceForAylaDevice` method needs to examine each AylaDevice object it receives and determine the correct Device object to return to the framework. This is usually done by examining fields of the AylaDevice object, often the oemModel or model, and checking for known values of these objects.

If unknown devices should be shown in the user interface (useful during development while new devices are not yet implemented), the `deviceForAylaDevice` may return a generic Device object. If unknown devices should be ignored and not displayed, the method may return null for unknown devices.

4.3.2 getSupportedDeviceClasses

The *getSupportedDeviceClasses* method of the DeviceCreator returns an array of Class objects with each supported Device-derived class in the application. This method is called from the *AddDeviceFragment* to populate the drop-down list of devices that may be added to the user's account.

4.3.3 viewHolderForViewType

This method of the DeviceCreator is responsible for creating the appropriate ViewHolder for Devices displayed in RecyclerViews. Each Device returns an integer representing the type of view that should be created for the device when shown in a RecyclerView. This method receives that integer and returns the view holder appropriate for that device.

If multiple Device classes use the same ViewHolder, they may return the same integer from *getItemViewType()* on the Device-derived object.

4.4 Create Your Devices

Each physical device type supported by the application needs to have a Device-derived object added to the application. AMAP contains a number of example Device objects that can be used for reference, extended, modified, or new Device-derived classes may be created, depending on the functionality of the device in question.

The majority of new code written for an AMAP application is creating the Device objects and their supporting UI classes. The Device class was designed to hide the details of the Ayla cloud service and it allows focusing on the specifics of the device itself. There are many methods of the Device class that may be extended to provide device-specific functionality to your application.

After creating each Device object, make sure to update the DeviceCreator to create an instance of your object based on the underlying AylaDevice.

4.5 Commonly Overridden Device Methods

4.5.1 ArrayList<String> getPropertyNames()

This method returns an array of strings each one is the name of a property that should be fetched during device polling. This set of properties is requested from the server during each device status update.

The method defaults to returning an empty array, which causes all properties to be fetched during each update. This is inefficient, however, and should be avoided, if possible.

4.5.2 String friendlyNameForPropertyName(String propertyName)

This method converts the name of a property, such as "`outlet1`", to a string to be presented to the user, such as "`Switch`". Whenever a property needs to be displayed to the user, this method is called to get the appropriate string to display.

Strings returned from this method should be localized.

4.5.3 String deviceTypeName()

This method returns the name for the device type, such as "`Smart Plug`" or "`Thermostat`". This name is displayed in the drop-down list of devices in the `AddDeviceFragment`.

4.5.4 int registrationType()

This method returns the registration type for this device. The default is `AML_REGISTRATION_TYPE_SAME_LAN`. If your device requires a different registration type, override this method and return it to the same location.

4.5.5 void postRegistration()

This method is called after your device has been registered (added to the account). Do any additional work here, if necessary. The default implementation does nothing.

4.5.6 Drawable getDeviceDrawable()

This method returns a Drawable that represents the device. Examples might be a picture of a wall plug, or a thermostat. The image is shown in the drop-down list of devices in the `AddDeviceFragment`, as well as, in the `DeviceDetailsFragment`.

4.5.7 int getItemViewType()

This method returns an integer representing which type of view should be used for this device. If each Device object in the system requires a different view when presented in the `RecyclerView`

(AllDevicesFragment, DeviceGroupsFragment), then each Device object should return a different value from this method. If multiple Device objects share the same view (for example, two different types of Smart Plug might use the same user interface but have different implementations), they may return the same value from this method.

4.5.8 void bindViewHolder(ViewHolder)

This method is called when the device is being displayed within a view holder, within a RecyclerView. The implementation should update the various views contained in the view holder with values contained in the Device object.

The specific ViewHolder passed to this method depends on the value returned from `getItemViewType()` and the ViewHolder returned from the DeviceCreator's `viewHolderForViewType()` method.

4.5.9 String[] getSchedulablePropertyNames()

Return an array containing the name of each property that can be set in a schedule. Generally these are the properties that the user can change, such as, on / off, temperature, and so on.

4.5.10 String[] getNotifiablePropertyNames()

Return an array containing the name of each property that supports property notifications (push, email, SMS) when the property changes. Generally these are properties that change on their own, such as, temperature, door sensors, and so on.

5 Changing Colors and Style

Now that your app is up and running, your new Device objects are being created and displayed in the `AllDevicesFragment`, it's time to change how things look to make your app stand apart from the rest.

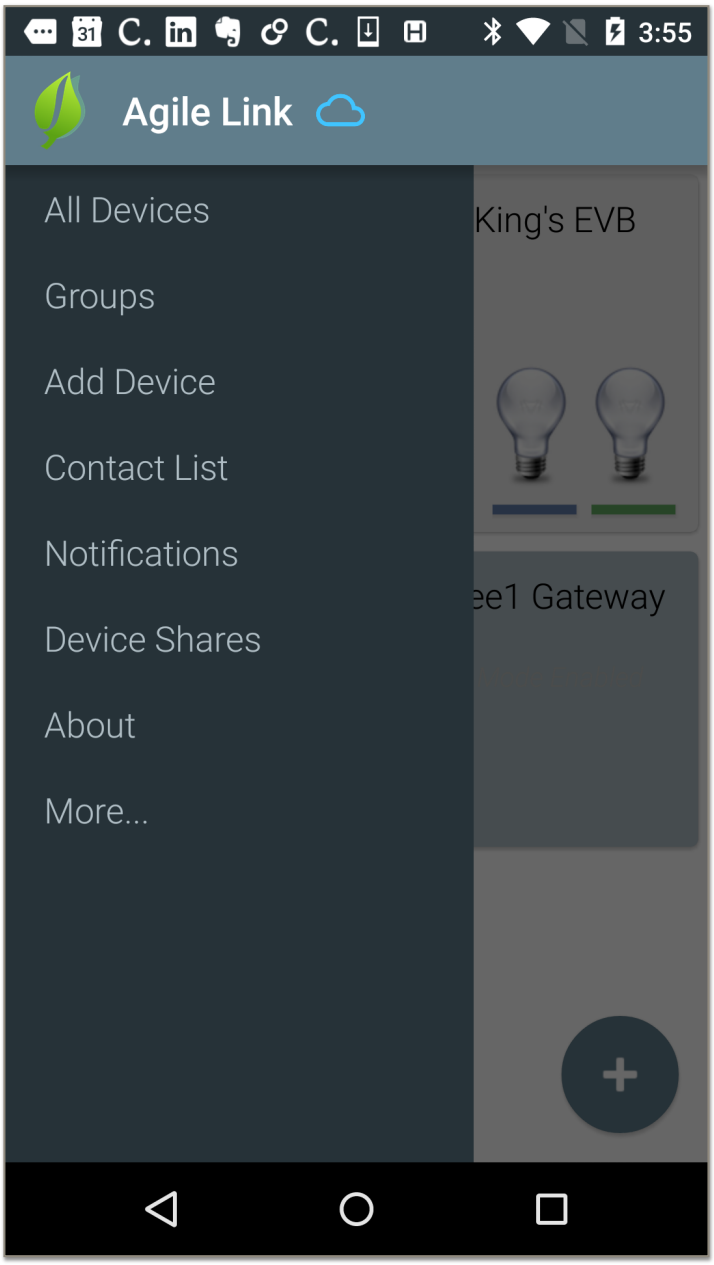
AMAP makes it easy to customize the look and feel of your application with minimal effort, and also allows you to take customization even further.

5.1 Colors

The `colors.xml` file included with AMAP contains a set of six "`theme`" colors that are used throughout the application. The beginning of the file defines the set of Android Material Design colors, and the end of the file defines the AMAP application theme colors.

Editing the six theme colors changes the color scheme for the entire application.

The graphic below shows the color scheme set to blue_grey with a light_blue accent.




```
<color name="app_theme_accent">@color/light_blue_A200</color>
    <color name="app_theme_primary_light">@color/blue_grey_200</color>
```

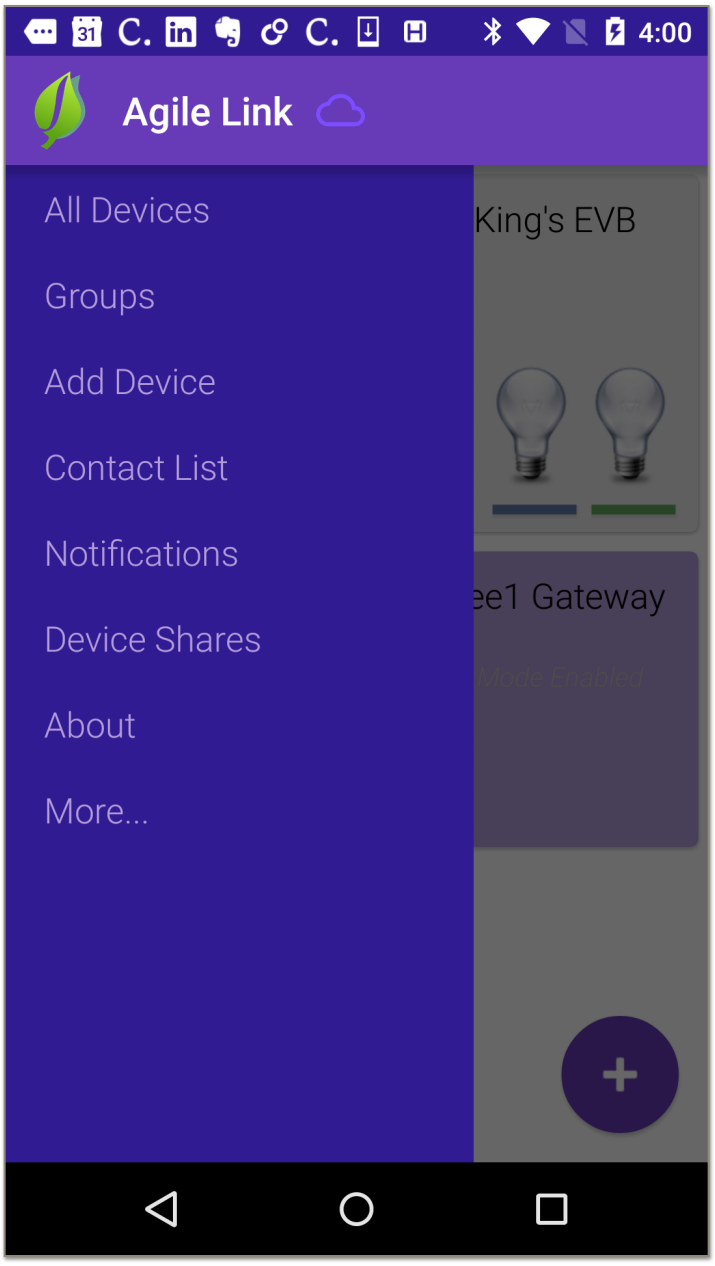
```
    <color
name="app_theme_primary_medium_light">@color/blue_grey_400</color>

    <color name="app_theme_primary">@color/blue_grey_500</color>

    <color
name="app_theme_primary_medium_dark">@color/blue_grey_800</color>

    <color name="app_theme_primary_dark">@color/blue_grey_900</color>
```

By changing only these six values, you can change the color scheme to dark_purple.




```
<color name="app_theme_accent">@color/dark_purple_A200</color>
    <color name="app_theme_primary_light">@color/dark_purple_200</color>
```

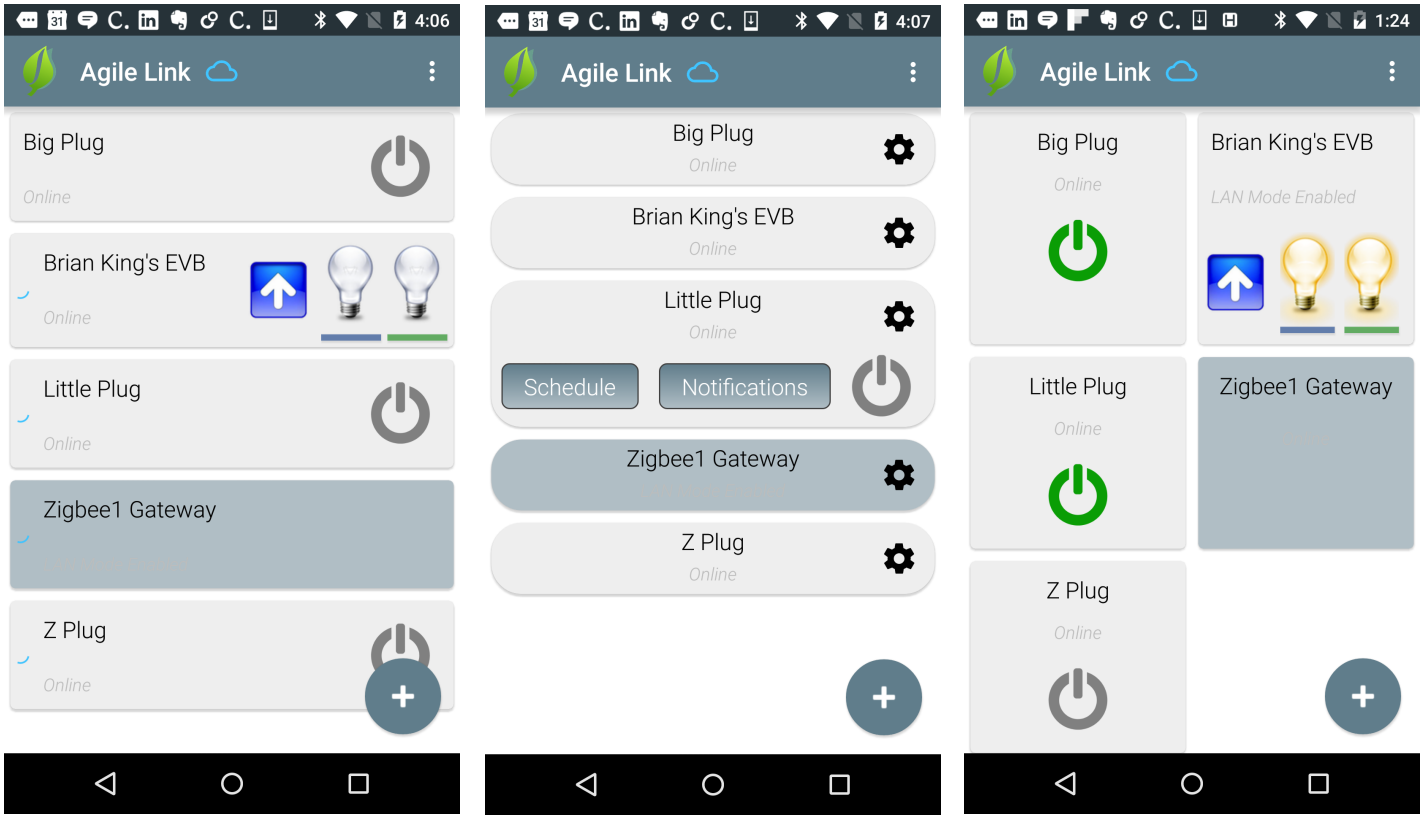
```
    <color
name="app_theme_primary_medium_light">@color/dark_purple_400</color>

    <color name="app_theme_primary">@color/dark_purple_500</color>

    <color
name="app_theme_primary_medium_dark">@color/dark_purple_800</color>

    <color name="app_theme_primary_dark">@color/dark_purple_900</color>
```

If further color customization is desired, you can update individual elements. Most elements throughout AMAP refer by default to one of these theme colors.



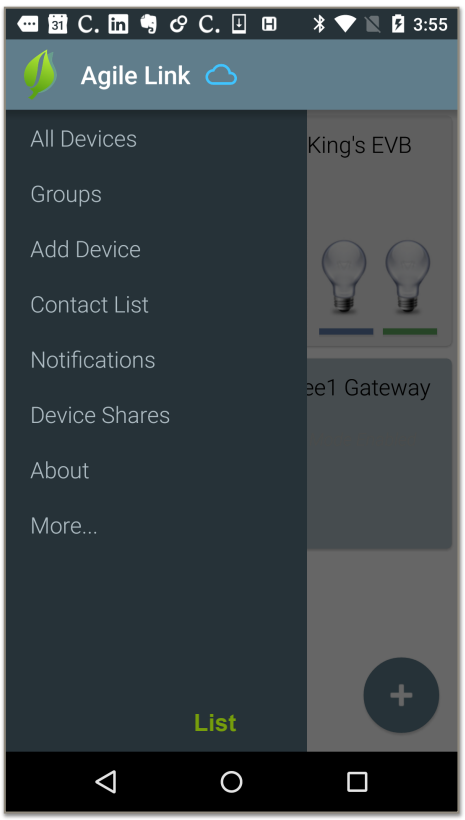
5.2 List Style

AMAP currently supports three styles of lists: `List`, `ExpandingList`, and `Grid`. Choose which style you want to use, and return the appropriate value in the `UIConfig` structure returned from MainActivity's `getUIConfig()` method.

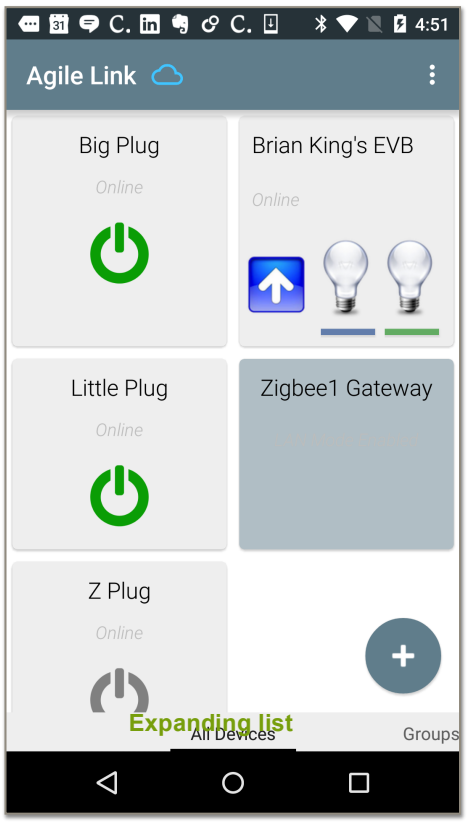
Remember that if you change list styles, you need to update your view holders to match your selection.

5.3 Navigation Style

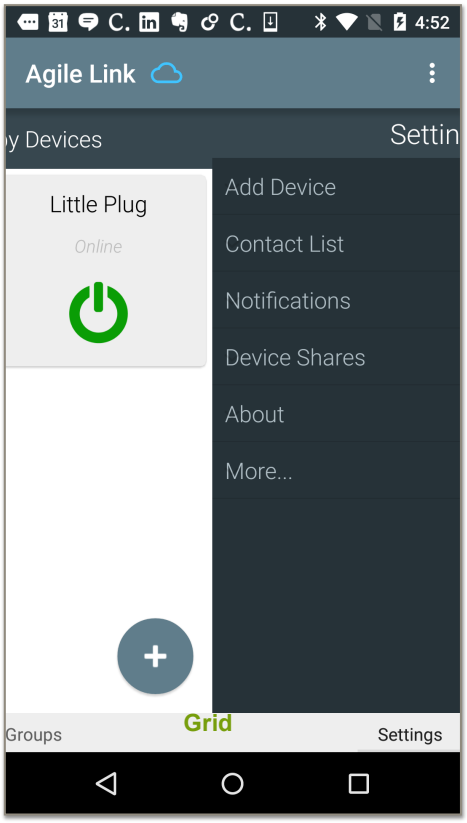
AMAP supports two navigation styles, "`Drawer`" and "`Pager`". Like the grid style, this preference is set in the `UIConfig` class in `MainActivity`.



Drawer



pager



pager (between pages)

6 Tablet Support

AMAP applications support both phone and tablet-sized devices. The UI is the same throughout with the exception that in landscape mode on a tablet in the Drawer navigation mode, the Drawer menu is presented as a fragment in the left pane rather than sliding out.

7 Further Customization

While AMAP provides a robust set of screens and not every implementation can be designed for ahead of time. The Device class contains methods that can be overridden to return Fragments for several of the device-specific screens. This means you can have different DeviceDetail, Schedule or Notification fragments for each device.

These methods are:

7.1.1 Fragment getDetailsFragment()

Returns the details fragment for this device. Defaults to `DeviceDetailsFragment`.

7.1.2 Fragment getScheduleFragment()

Returns the schedule fragment for this device. Defaults to `ScheduleFragment`.

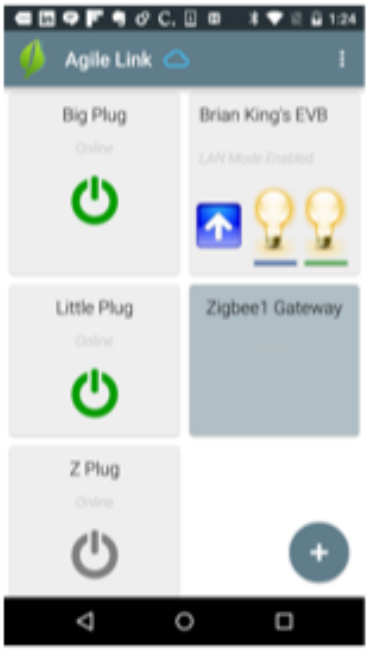
7.1.3 Fragment getNotificationsFragment()

Returns the notifications fragment for this device. Defaults to `PropertyNotificationFragment`.

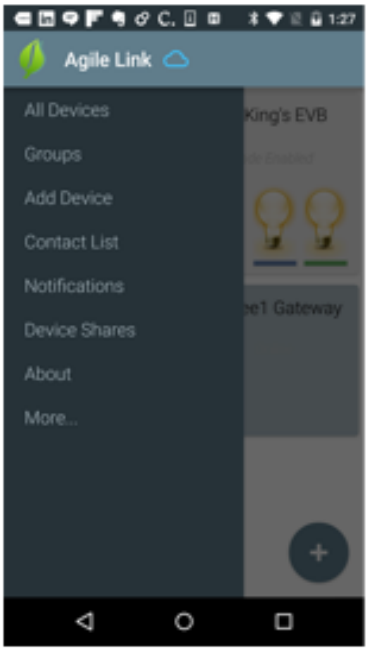
7.2 Fragments

SignInFragment

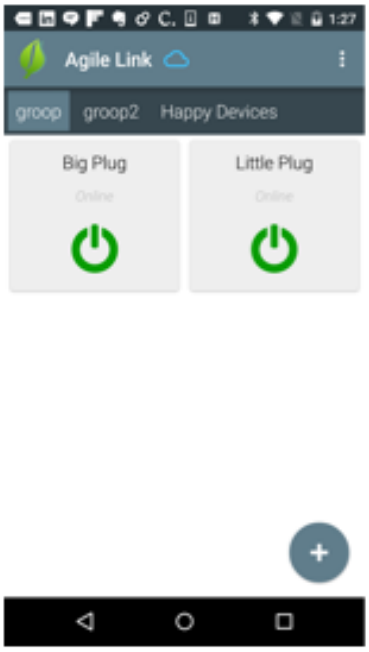
AllDevicesFragment (expanding list)



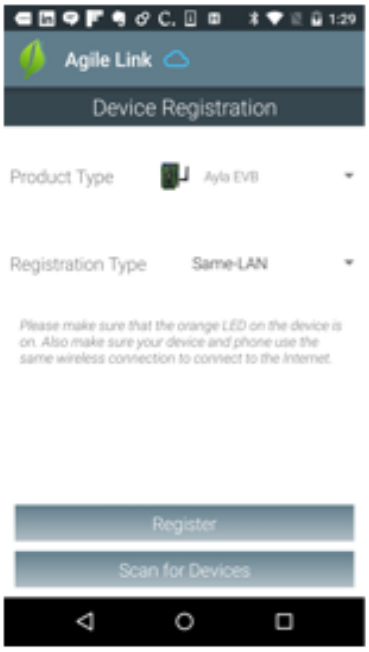
AllDevicesFragment (grid)



Navigation Drawer

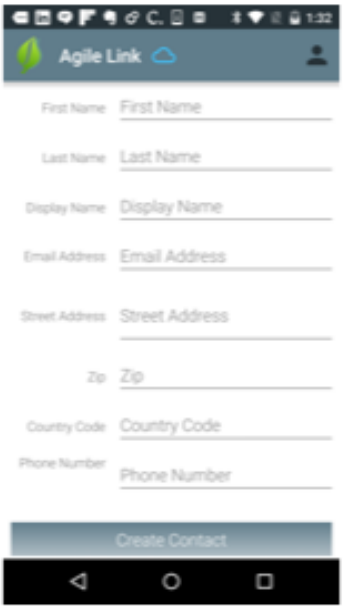


DeviceGroups



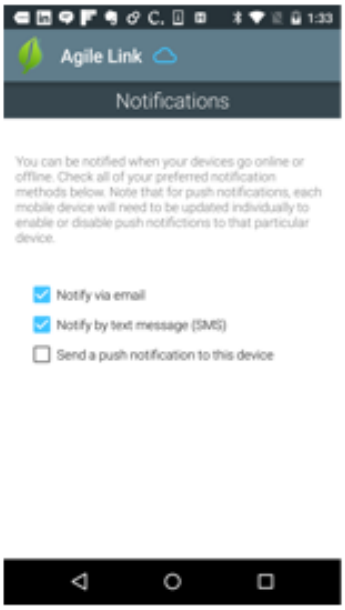
AddDevice

ContactList

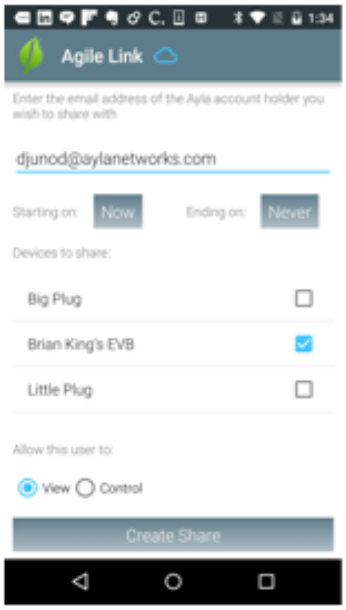


EditContact

AboutFragment



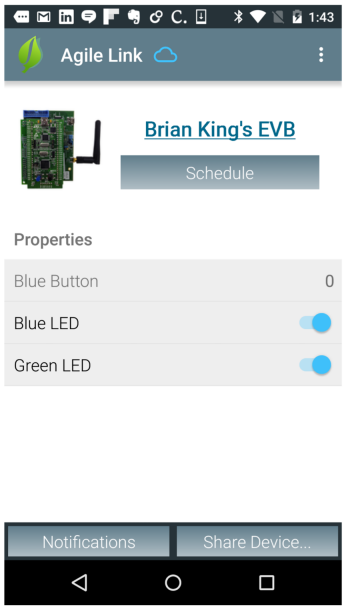
DeviceNotifications



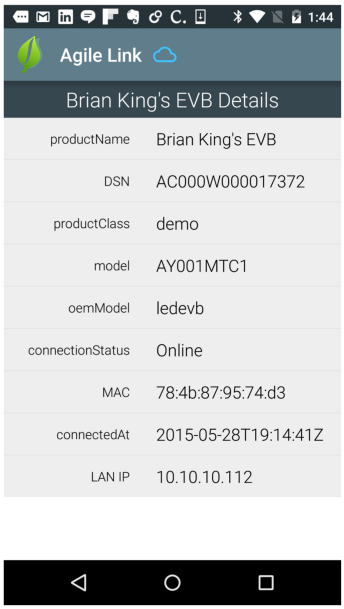
ShareDevices

Shares

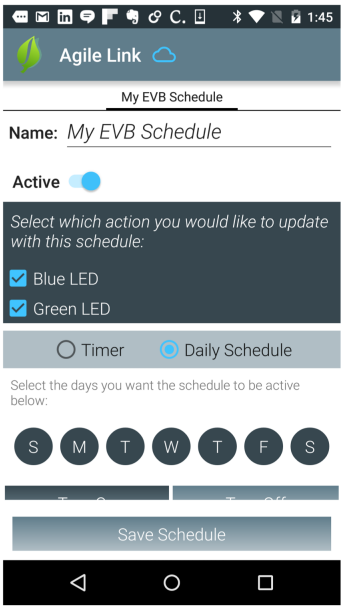
Agile Link Framework
Fragments p3



DeviceDetail



DeviceDetailList



ScheduleFragment

8 Contacting Customer Support

Ayla Networks provides online support at: support.aylanetworks.com.
The Ayla support website provides the following information:
Technical Support – Technical support requests, Frequently Asked Questions (FAQ), support documents, and development resources.
Product Support – Product data sheets, application notes and sample programs, design resources, getting started and user's guides and hardware support documents, as well as, the latest software releases.
Users of Ayla Networks products can receive assistance through several channels:

- Technical Support
- Field Application Engineer (FAE)
- Application Engineer (AE)

Technical support is available through the Ayla Support website at:
https://support.aylanetworks.com, or via email at support@aylanetworks.com