Developer's Guide

Ayla Embedded Agent for Marvell WMSDK

Version: 3.0

Date Released: April 29, 2017

Document Number: AY006DAM6-3

Ayla Networks

Copyright Statement

Trademarks Statement

Referenced Documents

Ayla Networks does not supply all documents that are referenced in this document with the equipment. Ayla Networks reserves the right to make the decision on which of the documents it supplies with the equipment.

Contact Information

Ayla Networks TECHNICAL SUPPORT and SALES

Contact Technical Support: https://support.aylanetworks.com
or via email at support@aylanetworks.com

Contact Sales: https://www.aylanetworks.com/company/contact-us

Ayla Networks REGIONAL OFFICES

Chicago
10 N. Martingale Road, Suite 400
Schaumburg, IL 601073

HEADQUARTERS
Silicon Valley
4250 Burton Drive, Suite 100
Santa Clara, CA 95054
Phone: +1 408 830 9844
Fax: +1 408 716 2621

Boston
275 Grove Street, Suite 2-400
Newton, MA 02466

Table of Contents

1 Introduction

The Ayla Embedded Agent enables modules to connect to Ayla Cloud Services using a white box development model. When using the Ayla Embedded Agent software on modules that have the Marvell WMSDK, there are specific installation, setup, and implementation details to ensure a successful implementation.

1.2 About this Document

This document describes the installation and use of the Ayla Embedded Agent software on modules using the Marvell WMSDK.

This version of the document is for use with Ayla Embedded Agent 1.2 and with WMSDK 3.4.6.

1.3 Intended Audience

This document is for developers who are familiar with the Marvell WMSDK and are evaluating the Ayla Embedded Agent or integrating the Ayla Embedded Agent functionality into a product.

1.4 Related Documentation

We recommend that you refer to the following documents:

- Marvell's user documentation on the WMSDK and its associated hardware, particularly:

  Marvell Wireless Microcontroller Development Host Setup Guide

  Developing with WMSDK Using Eclipse and Command-Line
- *Ayla Device Agent for Embedded Systems Developer's Guide* (AY006DAR6) for the portable aspects of the Ayla Embedded Agent.
- *Ayla Device Wi-Fi for Embedded Systems Developer's Guide* (AY006DAW6-2)
- *Ayla Product Development Overview* (AY006GOV1) for high-level steps on bringing an IoT product to market.
- *Ayla Module and MCU Interface Specification* (AY006MSP0) primarily for the descriptions of properties; the type, length, and value items (TLVs) that make up schedules; and the client configuration.
- *Ayla Factory Service (AFS) API Specification* (AY006AFS6)

NOTE The Ayla documents are available on support.aylanetworks.com.

1.5 Abbreviations and Acronyms

The following acronyms are used in this document.

ADA	Ayla Device Agent. This is a legacy term for Ayla Embedded Agent. The ADA acronym is still used in CLI commands and sources, and descriptions of either of these may refer to the Ayla Embedded Agent as the "device agent."
ADW	Ayla Device Wi-Fi.
EVB	Evaluation Board
FTDI	Future Technology Devices International
I/O	Inputs/Outputs
JTAG	Joint Task Action Group
MCU	Microcontroller unit
OTA	Over-the-Air
UUID	Universally Unique Identifier
WMSDK	Wireless Module Software Development Kit

1.6 Glossary

White Box	This is a type of Ayla endpoint that allows for a more complex and versatile device than the Black-Box class of devices. However, the development effort is significantly longer for OEMs and therefore results in longer time to market than the Black-Box modules. Some primary characteristics are: • Available for embedded or LINUX solutions. • The Ayla Embedded Agent is available as a library or source. • Well-equipped for applications with existing RTOS and networking. • The Ayla White-Box-based Cloud Agent's modular design allows code for additional functions to be included as needed.
Black Box	This is a fully-managed, Ayla-enabled module intended to be used as-is by the manufacturer. Some primary characteristics are: • Available for embedded solutions. • Provides the fastest time to market for OEMs • No custom gateway or other forms of communication agent software, including QA required regardless of the type of end-device. • Any microcontroller-based system can be enabled with cloud connectivity.

1.7 Document Conventions

This document uses these Ayla documentation conventions:

- Text that you type (such as commands) and the contents of downloaded files are shown as:

```
cd wmsdk_bundle-3.1.16.1
tar xfz ada-wmsdk-src-1.0.tgz
```

- File names, scripts, names of commands, properties in a file, code, and the like are also in `courier` new font. For example: Use the `psm-dump` command to … .
- Network paths, file paths, menu paths and the like are shown in `courier` new font and each point that you have to click to navigate to the next is separated by "`/`."
- Ancillary information that is important to emphasize is shown as:

NOTE The commands provided in the example assume your evaluation board is `mw300_rd` and your chip is `mw300`. If otherwise, make the appropriate substitutions.

- Information describing hazards that could damage a product, including data loss, is shown as:

CAUTION

Ensure that the appropriate data buffering is accounted for in deployed devices, where loss of data is critical to the core functionality or the services provided by the systems.

1.8 Revision History

Table 1 provides a summary of updates to the content of this document.

Table 1: Revision History of Content Updates and Changes

Revision #	Date (yyyy-mm-dd)	Change
1.0	2015-08-31	Initial version.
1.1	2015-09-03	Minor updates
2.0	2016-01-08	Minor updates
3.0	2016-27-01	Updates for ADA 1.2, WMSDK 3.4.6, ADW

2 WMSDK Installation

Before installing the Ayla Embedded Agent, you should be able to do the following:

- Build and run one of the demos that Marvell supplies with the WMSDK.
- Connect to the serial console and download programs using JTAG.

Additionally, use the instructions provided by Marvell to install the following:

- The WMSDK and the cross-compilation tools required.
- The drivers to access the serial port and JTAG devices for your evaluation board.

3 Ayla Embedded Agent Installation

Once you have installed everything outlined in Section 2 for the WMSDK, follow the steps in this section to install the Ayla Embedded Agent.

If you copy and paste commands from this document, the results may be UTF-8 characters that look the same as the simple ASCII ones, which may not work. To avoid problems with the commands, make sure that they contain minus signs (dashes), not other symbols, like the em dash.

1. Extract the Ayla Embedded Agent package into the directory created when you extracted the `wmsdk_bundle-3.4.6` as follows:

```
cd wmsdk_bundle-3.4.6
tar xfz ada-wmsdk-src-1.2.tgz
```

2. Copy the files into the WMSDK tree as follows:

```
cp –R ada-wmsdk-src-1.2/* .
```

NOTES

- The first time you build the application, Makefile.ayla applies the patches to WMSDK for the Ayla Embedded Agent. The modifications are minor, but required.
- If you have an earlier version of the Ayla Embedded Agent installed, make sure that only the new or changed patches that were not in the earlier version are applied when you install the later version. The easiest way to do this may be to revert to the earlier patches before applying the new ones.

3. Build the demo application using, for example, the following commands:

NOTE The commands provided below assume your evaluation board is `mw300_rd` and your chip is `wmcore`. If otherwise, make the appropriate substitutions in Makefile.ayla.

```
make -f Makefile.ayla
```

This build results in three files:

- `ayla_demo.axf`
- `ayla_demo.bin`
- `ayla_demo.ftfs`

NOTE The remaining steps in this section use a script `flashprog.sh` to download these and other files to the flash on the module.

4. If you choose to allow `flashprog.sh` to run without being `root` (under `sudo`), configure your FTDI-based USB devices to be writable by you, using the following commands:

```
sudo usermod -a -G dialout $USER
newgrp dialout
newgrp
```

The two `newgrp` commands run subshells to make the group change effective.

5. Format the flash storage using the following command.

```
make -f Makefile.ayla flash_layout
```

6. If the module of the hardware device has not run a 3.4.x WMSDK app before, initialize the flash layout of the module by downloading `boot2` and `wififw` with the following commands. If you have already done this, you may skip this step.

```
make –f Makefile.ayla download_boot2
make –f Makefile.ayla download_wifi
```

NOTE You may follow any of Marvell's instructions in preference to the commands in this step.

7. Download `ftfs` and `mcufw` to the evaluation board as follows:

```
make –f Makefile.ayla download_ftfs
make –f Makefile.ayla download
```

8. Press the Reset button to ensure that the device boots into the demo.

4 LEDEVB Demo Overview

The LEDEVB demo is the default app that you build after completing the steps provided in Section 3 to install the Ayla Embedded Agent. The demo is to facilitate your use of the various property types and features of the agent. In the demo, there are a number of properties that you can set from a mobile app or the developer web site.

NOTE If you have used the black-box MCU EVB demo, it is similar to the LEDEVB demo.

This section provides information that pertains to the WMSDK when using the LEDEVB demo.

NOTE In `sample_app/cloud_demo/ayla_demo`, there is another demo called `demo_outlet1.c`. This is an alternative to `demo_ledevb.c`.

4.1 CLI Commands

The LEDEVB demo provides a few CLI commands to configure the device. Some of the commands are standard WMSDK commands , and all other commands are described in the *Ayla Device Agent for Embedded Systems Developer's Guide*.

Make sure that you review the set of CLI commands included in the demo when building a product. Some CLI commands should not be used for an end-user product because these commands can reveal secrets, like the Wi-Fi passphrases. Commands that can show such values are the WMSDK commands `psm-dump` and `psm-get`.

4.2 General Purpose Inputs/Outputs (GPIOs)

The board I/Os are configured using the board file in the source. The demo uses the following:

- board_button_1() and board_button_2()
- board_led_1() and board_led_2()

The board_led_1() is turned on when the module is connected to the Ayla service; otherwise, this LED is turned off.

4.3 Device Properties

The `demo_levdevb` in `sample_app/cloud_demo/ayla_demo` implements a subset of the Ayla development kit demo properties. Following are descriptions of the properties:

- The `Blue_button` property is a Boolean data type from the device. It indicates the input from the WMSDK board_button_1() function and the current state of button 0, (`GPIO_26` on `wm300_rd`). "1" indicates pressed and "0" if not pressed. The value is sent for each change.
- The `Blue_LED` property is a Boolean data type to the device and controls the board_led_2() output. The LED2 (`GPIO_41` on `wm300_rd`) is yellow.
- The `Green_LED` property is a Boolean data type to the device. This property only prints a message with the new value and saves it in memory. If another LED is available, it is easy to control that with this property. However, this property does not control anything, but changes to the property are logged to the console.
- The `decimal_in` property is a floating point data type to the device. This property is reflected back as the `decimal_out` property from the device as an example of how to use decimal properties. The value in the demo is 100 times the value entered. For example, if 1.23 is entered on the developer site, the value set in the `decimal_in` variable is 123.
- The `decimal_out` property is a floating point data type from the device. This property reflects the last value set by the `decimal_in` property.
- The `input` property is an integer data type to the device. This property sets the input variable and reflects the value back in the `output` property. The `input` and `output` properties are linked. Whatever integer value is sent to `input`, the value is sent back to the service as `output`.
- The `output` property is an integer data type from the device. This property provides the value of the `input` property.
- The `cmd` property is a string data type to the device. The `cmd` and `log` properties are strings that are linked. The `cmd` property sets a string in a buffer and reflects that setting back in the `log` property. The maximum length of these strings is 1024 bytes of UTF-8 after JSON encoding. This means that some characters are counted as multiple bytes.
- The `log` property is a string data type from the device. This property gives the value of the `cmd` property.
- The `oem_host_version` property is the template version used to select the appropriate Ayla template. Currently, the template version is "1.0," which is the template that provides all of the properties in this list and has AP-mode registration with LAN mode turned on.
- The version property is a string that indicates the software version number of the demo (host) running on the device.
- There are five schedules, `sched1` through `sched5` associated with the demo. These schedules allow changes to properties on a time schedule. For example, you could program the `Blue_LED` property to turn on every 10 seconds, once a day, and so on.

- Other properties in the `ledevb` template, for example, `stream_*`, are not implemented by the demo.

4.4 Registration Method

By default, the LEDEVB 1.0 template selects AP-Mode Registration Method. If you change to use a template specifying the Button-Push method, you can use the `board_button_2()` input to open the registration process.

NOTE For more information on the Ayla Registration Methods, refer to *Device Onboarding: Ayla Registration Methods* (AY006FOR3-1) available on support.aylanetworks.com.

5 Setting Up the Demo for the First Time

The first time you use the LEDEVB demo, you have to set the serial number and key for the device, as described in this section.

5.1 Obtaining the Device Serial Number (DSN) and Key XML Files

Upon request, we provide the DSN and private/public key pair from the Ayla Factory Service (also referred to as AFS). The main purpose of AFS is to generate this information to authenticate the device with the cloud service. Please contact your Ayla Customer Support representative to obtain the Open Agent XML Provisioning files, which are used in the examples shown throughout this section.

5.2 Setting the Device Serial Number and Key

A different Ayla DSN and key must be configured for each device. Each DSN and key comes in a separate XML file like the following:

```
<?xml version=”1.0 encoding=”UTF-8?>
<f-device>
    <dsn>AC000W00012346</dsn>
    <public-key>-----BEGIN RSA PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA4uBo4pDNe7aS4CJS0ck1
pKaHCS1fv2Oqr+DnPiA5Ue/6Htac7j9DQyc4qD8sxHip/7v9M2HqGKv
FPhkb7SDjbj+v/qYca2cWXhqUYv9H5wEeCUG2MaBYGUZ0HfnBiD/PIULWmYBLZ6D
4yF3dQIi+MCgWJU9GCVR6pCt+02SUfIxOQWPslx5nlkMMpj+E1ORjYPSa7kIeoik
waoPc85UfTA+1TSOq5tNHZu+CuoYBkRPkbXM1bhrQx98TY8FQG/zS+kqmedpV73S
xotItJjtlupKQI6C42Bg867ncuvNufUF+58WZGobaZZIu3I3ouXxGqIgqdOMYi+q
kqidaqab
-----END RSA PUBLIC KEY-----
</public-key>
</f-device>
```

Use the `psm-dump` command to see if these values have already been set up for you, or to verify your entries afterwards.

To set the DSN and key into PSM for the Ayla Embedded Agent, the key must be collapsed into a single line, using the part between the dashed lines and eliminating blanks. The resulting commands are shown below (the prompts and most of the key are omitted).

```
psm-set ada.f id/dev_id AC000W000123456
```

```
psm-set ada.f id/key MIIBIjANBgkqhkiG9w0BAQEFAAOC … DAQAB
```

Reboot the device as follows to show the new parameters in the factory-log.

```
save
pm-reboot
```

5.3 Setting the OEM Secret

The OEM secret is the key that is used to authenticate the OEM when communicating and transmitting information with the Ayla Cloud. To authenticate the device from your OEM, you must enter the OEM secret. This secret (key) is saved in encrypted form on the device and must be entered for each device after entering the public key. Authentication is not required for devices used on the Ayla Networks developer site, including those using the ledevb demo. However, as a best practice, you should authenticate these devices since authentication is required when the devices are switched from the Ayla Developer Service to the Ayla Field Service.

Set the OEM Secret in the OEM Profile page of the OEM Dashboard, as shown in Figure 1.

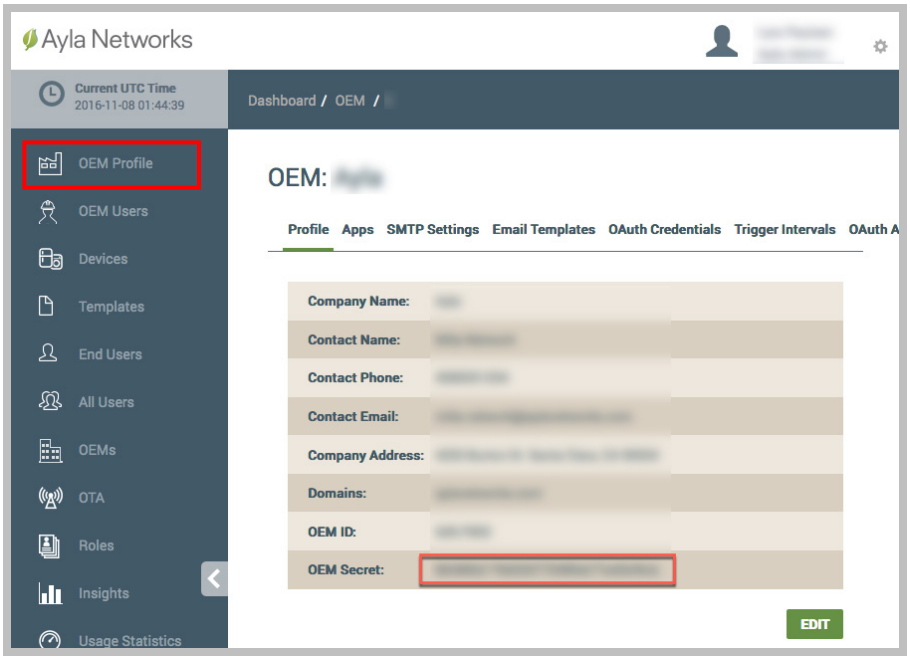

Figure 1: Setting the OEM Secret

NOTE Refer to the *OEM Dashboard User's Guide* (on support.aylanetworks.com) for more information on making changes to the OEM Profile.

The resulting commands to do this are as follows (note that the prompts and most of the key are omitted):

```
oem key ab…cd
```

5.4 Sending Log Entries

If the modules indicate that their connection is part of the manufacturing test, the modules may immediately connect to Ayla Device Service (ADS). To indicate this, internally set `ada_conf.test_connect` to 1, using a CLI or other method. This is the most convenient way to provision devices.

NOTE This indication is not persisted across reboots.

Do not set `test_connect` on every boot, and make sure that the Ayla module part number (which is AY008MVL1 in the example in this section) is specified when the DSN is reserved.

If `test_connect` is not used, the alternative is to send a log entry to Ayla to enable the device on the ADS. You send the log entry to your Ayla representative or the AFS using an API. The log entry provides information about the manufacturer and the device, including its MAC address and hardware ID from the flash. To generate the log line, set the time and use the `factory-log` CLI command as shown below.

```
# time-get
Thu 2015 Jan 01 00:00:05

# time-set 2015 8 28 20 47 20
New time set:
Fri 2015 Aug 28 20:47:20

# factory-log
factory-log line:
3,1440794854,2015/08/28 20:47:34
UTC,label,0,AY008MVL1,AC000W000432408,00504321e5a0,mw300_rd,p1,d663604
5ce20691500504321e5a0,,0dfc7900,smartplug1,0,ADA demo
customer,ayla_outlet1_demo 1.0 Aug 28 2015 13:38:08
```

Upload the last line (starting with "3,...") to support.aylanetworks.com. You can upload several device logs in the same file. Please be careful not to add spaces or newlines to that single line.

5.5 Setting Up Wi-Fi

NOTE This section assumes that you use the Ayla Device Wi-Fi (ADW) facilities to manage your Wi-Fi since the LEDEVB uses ADW. ADW maintains the Wi-Fi configuration, keeping the device connected if possible, and supports Wi-Fi setup either with a web page or a secure Wi-Fi setup mobile application.

The Wi-Fi configuration is typically done using a smart phone application, such as Ayla's Aura or AMAP. However, Wi-Fi may also be configured directly from the console interface of the device. The following example shows the commands involved when configuring the Wi-Fi Profile from the console.

```
wifi profile 0
wifi ssid <ssid> security <security_type> key <password>
    where security_type is WPA2_Personal, WPA, WEP, or none
wifi join
```

In the example above, the first command selects the current profile from the available ones, which are 0 through 9. The default profile is 0, so unless another profile is selected, you can skip this command.

To join the network called MyNet with a passphrase of "Pass-phrase-122," use the following command.

```
wifi ssid MyNet security WPA2_Personal key Pass-phrase-122
wifi join
```

NOTE For a complete description of the wifi command, refer to *Ayla Device Wi-Fi for Embedded Systems* available on support.aylanetworks.com.

6 Running the Demo

When you first run the demo, you must register the device as described in this section and set up the device properties described in Section 4.3.

After the initial re-boot (20 or more seconds at least), the demo looks like the example below.

```
ayla_ledevb_demo 1.0 Nov 14 2016 09:58:52
-------------------------------------------------
[ada] 2016-11-21T22:09:13.500 i m client: ADA 1.2-eng wmsdk-3.4
2016-11-14 11:42:09 c5a63fc
[ada] 22:09:15.530 i w wifi: connecting to SSID MyNet sec
WPA2_Personal signal -45
[ada] 22:09:16.273 i m mod: demo: event 8 normal init
[ada] 22:09:16.274 i m mod: demo: event 9 normal connecting
[ada] 22:09:20.217 i m mod: demo: event 10 normal connected
[ada] 22:09:20.218 i m client: ada_client_up: IP 192.168.1.249
[ada] 22:09:20.224 i w wifi: join succeeded
[ada] 22:09:20.229 i c client: get DSN AC000W000123456
[ada] 22:09:20.367 i m mod: demo: event 17 dhcp renew
[ada] 22:09:25.471 i c client: module name "ledevb" key 55438.
[ada] 22:09:25.475 i c client: listen enabled
[ada] 22:09:25.488 i notify: ANS server ans.aylanetworks.com at
54.89.90.220
demo_int_set: input set to 321
demo_led_set: Blue_LED set to 0
[demo_led_set: Green_LED set to 0
[ada] 22:09:28.613 i c client: ANS check event
```

Once connectivity with the Ayla Device Service is established, LED1 is illuminated. For the `wm300_rd` board, this is `GPIO_40`, which is an amber LED.

6.1 Registering the Device

Create a user account on https://developer.aylanetworks.com. There you can register the device, or you can use the AMAP or Aura application for iOS or Android. Before completing the registration or using the application:

- Make sure that the computer you are using for the browser or your mobile device is on the same local network as the WMSDK device.
- Verify that you are using the same wireless access point (AP).

After creating a user account, log in to the developer site and click **Register New Device**. This is a link that takes you to the instructions to register a new device.

7 Sample Application Code

The sample applications on are provided as working examples of how the Ayla Embedded Agent APIs are used. These applications should be evaluated only as examples and not considered the only way or even the best way to use the APIs.

The sample application source is in `sample_apps/cloud_demo/ayla_demo/src` and is composed of the following C files:

- `demo_ledevb.c`
- `demo_common.c`
- `conf.c`
- `ota.c`
- `sched_conf.c`
- `board.c` (which is copied there by the wmsdk build)

This section provides descriptions of the files and how they should work.

7.1 demo-ledevb.c

This file contains the code specific to the demo and its properties.

The demo starts in `demo-ledevb.c` in `main()`, which calls `demo_start()` and is part of `demo_common.c`. This initializes the demo, the device agent, and the WMSDK application framework and then launches an application thread that calls `demo_init()` and runs `demo_idle()` in this file.

`demo_init()` uses the `ada_sprop_mgr_init()`API to register the properties handled by the device through the simple property manager. The table used is called `demo_props`.

`demo_led_set()` is used to handle the setting of the `Blue_LED` and `Green_LED` properties.

`demo_int_set()` is used to handle the setting of the `input` and `decimal_in` properties.

`demo_cmd_set()` is used to handle the setting of the `cmd` property.

`demo_idle()`does the following:

1. Reads the two buttons.

   The two buttons are as follows:

   - Button 1 is set as the `Blue_button` property.
   - Button 2 is used to start a registration window (for the Button-Push registration method). This method is not used for the `ledevb` template, but may be useful for other demos, including `demo_outlet1`.

2. Sends the `version` and `oem_host_version` properties

3. Enters an idle loop that sleeps for 100 ms
4. Updates the Link LED
5. Sends any property updates that have been requested, and samples the buttons.

7.2 demo_common.c

As stated in Section 6.1, the demo starts in `demo_ledevb.c` in `main()`, which calls `demo_start()` and is part of `demo_common.c`.

`demo_common.c` contains the code that is common among most demos running under the WMSDK.

The first call to `demo_common.c` is from `main()`, calling `demo_start()`, which does the following:

1. Calls `module_init()`.
2. Prints the software version.
3. Calls `app_framework_start()`, which triggers callbacks to `common_event_handler()`.

`modules_init()` initializes the serial port, the CLI, and the real-time clock.

`common_event_handler()` logs each event as it occurs and handles the `WLAN_INIT_DONE` and `NORMAL_CONNECTED` events. When the WLAN initialization is completed, it does the following:

1. Calls `psm_cli_init()` to enable the persistent storage manager command line interface.
2. Calls `demo_client_start()` to initialize the device agent.

   On the `NORMAL_CONNECTED` event, `common_event_handler()` starts the device agent client and creates the demo thread that runs in `demo_idle_enter()`.

`demo_client_start()` calls initialization routines that are part of the device agent interfaces and in `conf.c` and `sched_conf.c`.

`clock_init()` makes sure that the clock is in range to make the SSL certificates valid.

`client_conf_init()` is described in Section 7.3, conf.c, of this document.

`client_task_init()` starts the device agent client thread.

`ada_sched_enable()` turns on the schedule handler inside the device agent.

NOTE If Ayla Device Wi-Fi support (ADW) is enabled, `demo_client_start()` calls `demo_wifi_init()`.

`demo_client_start()` calls `demo_init()` in `demo_ledevb.c` to register the properties.

7.3 conf.c

This file loads the Ayla Embedded Agent client configuration from the WMSDK PSM and persists values for the device agent as needed. The file also provides several platform-specific interfaces required by the device agent. The platform-provided interfaces are prefixed with "`adap_`" and the requirements for these interfaces are detailed in the *Ayla Embedded Agent Developer's Guide.*

The device agent configuration variables have hierarchical names separated by slash characters, like `id/dev_id`. For the WMSDK PSM, the factory configuration names are prefixed with "`ada.f.`" and start-up configuration names are prefixed with "`ada.`" The full name for `id/dev_id`, for example, would be `ada.f.id/dev_id`.

`client_conf_init()` does the following:

- Reads the MAC address and the system UUID, which the device agent sends to the Ayla Device Service to confirm that the device is not using the same serial number as another device.
- Calls `sched_conf_load()` to load the saved schedules.
- Sets the `enable` and `get_all` flags in the `client_conf` structure, as well as the `poll_interval`, which is used in cases where connectivity to the Ayla Notification Service (ANS) is lost.
- Calls `log_init()` to initialize the device agent logging interfaces.

The device agent requires `adap_conf_reset_factory()`to clear a variable from the PSM startup configuration so that the factory-configured or default value becomes effective.

7.4 ota.c

This file contains code to demonstrate the OTA firmware download interfaces. A host OTA can be deployed from the developer's site, and the code in this file receives the host OTA and performs a CRC calculation on the code without storing it anywhere. In a real application, the new code would be stored to flash and set to boot after being verified.

Refer to the *Ayla Embedded Agent Developer's Guide* for information on the OTA interfaces.

7.5 sched_conf.c

This file contains code to support schedules, which are optional. Schedules allow actions defined in the properties to occur at specific times and in a specific order — either on a regular or irregular schedule. For example, you could set a schedule to turn on an output once every 10 seconds or on the third Tuesday of each month.

NOTE As stated at the beginning of Section 6, the sample applications and code are provided as working examples of how the Ayla Embedded Agent APIs are used. They should be evaluated as examples only and not considered the only way or even the best way to do things.

Table 1 provides descriptions of the functions in the `sched_conf.c` file.

Table 2: Functions in the sched_conf.c File

Function (s)	Description
`sched_conf_load()`	Sets the number of schedules and the name for each one. The names should match the names in the template on the service, and the names known by the mobile application. This function also loads the values saved for any schedules that have been set.
`adap_sched_conf_persist()` (platform-support function)	Saves any schedules that have been changed by ADA. The in-memory value of the schedule is read using `ada_sched_get_index()` and compared to the saved value obtained by `adap_conf_get()`. If there is a difference, the in-memory value is written using `adap_conf_set()`.
`sched_conf_name()`	Fills in the configuration name for the value of a schedule, given its index.
`adap_sched_run_time_write()` and `adap_sched_run_time_read()`	Save and restore the last time that schedules were processed. These functions should put that time in non-volatile storage, if possible, to avoid having to reprocess events that have already taken place after rebooting the system. This saves power if non-volatile memory is retained and may be important if schedules are heavily used. Be sure to validate that the non-volatile storage is valid when reading this value, as it can be scrambled if power is not maintained. A CRC and magic number could be used. If the value is not valid, "0" should be returned.

8 Source Code Organization

The library source code for the device agent, not including `sample_app`, is delivered in the subdirectory `sdk/external/ayla`. This directory contains a `Makefile` file and the libraries `libada`, `libayla`, and `libnet`. These libraries are described in detail in the *Ayla Device Agent for Embedded Systems Developer's Guide*. Details specific to WMSDK are documented in this section of this document.

8.1 Library ext_xyssl

This is an open-source SSL library from which the device agent uses some encryption routines. This library is not used for SSL/TLS itself. XYSSL is copyrighted by Christophe Devine and provided under the BSD license, which is included in the source.

8.2 Library libada

This library contains the Ayla Embedded Agent. Most of the library is portable and described in the *Ayla Device Agent for Embedded Systems Developer's Guide*. The platform-specific portions are under the `al` subdirectory. The abbreviation "al" is for adaptation layer.

The descriptions of the files in this library are as follows:

al/wmsdk/ada_lock.c

This file implements an interface to the Threadx mutexes.

al/wmsdk/client_task.c

This file contains code for the client thread, its initialization, timers, locking, and work queue.

The function `ada_init()` does the following:

- Initializes the logging subsystem, including setting severity levels to be logged for each subsystem.
- Initializes the timers, and the mutex, which protects most of the client data structures.
- Calls clock_init(), client_init(), net_init(), and ada_conf_load(), and then creates the client_task_queue and the client thread.
- Calls client_mdns_init().

The client thread runs in the function `client_idle()`. This initializes the TLS library and the `client_redir_client_html()`handler (described later in this section), takes the client lock, and then enters the idle loop.

The idle loop handles any pending timers that need to be handled in the client thread while still holding the client lock. Then, the idle loop waits for callbacks on the `client_task_queue`. This waits until the next scheduled timer or until an event occurs. If a callback is found, its handler is called and the loop repeats.

`al/wmsdk/client_task.c` contains code in the function `client_redir_client_html()` that redirects HTTP requests for `/client` to `/client.html`. This is necessary for same-LAN Web-based device registration.

The remainder of the code in this file implements client timers and synchronization needed with the server thread.

The functions `client_lock_int()` and `client_unlock_int()` implement the locking around the client structures. These functions keep some debugging information that can be useful if there is trouble in this area. The global string pointer `client_mutex_func` holds the name of the function that last successfully locked the `client_lock`. The global integer `client_mutex_line` gives the line number in that function. The global pointer `client_mutex_owner` is a pointer to the OS task that holds the mutex, and is used to detect recursive attempts on the lock.

al/wmsdk/http_client.c

This file implements the interface between the HTTP client of the device agent and the httpc functionality of the WMSDK.

al/wmsdk/notify_task.c

This file implements timers for the notify subsystem, which interacts with the Ayla Notification Service.

al/wmsdk/server.c

This file interfaces between the device agent server API and WMSDK.

al/wmsdk/stubs.c

This file contains miscellaneous required routines, some for features that are not functional in this architecture; some that are simple interfaces between the device agent and WMSDK.

8.3 Library libadw

The sources in this library are described in the *Ayla Device Wi-Fi for Embedded Systems Developer's Guide*, except for `wifi_mrvl.c`, which provides interfaces between Ayla Device Wi-Fi (ADW) and WMSDK for Wi-Fi functionality.

8.4 Library libayla

The sources in this library are portable. These are described in the *Ayla Device Agent for Embedded Systems Developer's Guide*.

8.5 Library libnet

This library contains a thin layer of code and macros that interface to LwIP and WMSDK.

8.6 Using JSMN Parser

JSMN (pronounced Jasmine) is a small JSON parser. The device agent uses a different version of JSMN than the one in the WMSDK. The Ayla Embedded Agent version is in `wmsdk|wmsdk|external|jsmn`. This includes a set of preprocessor symbols defined in `jsmn.h` that redefine all the external functions and types to avoid name conflicts with WMSDK.

4250 Burton Drive, Santa Clara, CA 95054
Phone: +1 408 830 9844
Fax: +1 408 716 2621