Qeedji

Developer manual 001A

AOSP-9.11.10

Legal notices

AOSP-9.11.10 (001A_en)

Rights and Responsibilities

All rights reserved. No part of this manual may be reproduced in any form or by any means whatsoever without the written permission of the publisher. The products and services mentioned in this document may be trademarks and/or trademarks of their respective owners. The publisher and the author do not pretend to these brands.

Although every precaution has been taken in the preparation of this document, the publisher and the author assumes no responsibility for errors or omissions, or for damages resulting from the use of the information contained in this document or the use of programs and source code who can accompany it. Under no circumstances can the publisher and the author be held responsible for any loss of profits or any other commercial prejudice caused or that would have been caused directly or indirectly by this document.

Product information

The conception and specifications of the product may change without prior notice, and this applies to hardware, embedded software and this guide. Consumable items accessories may slightly differ than herein described as Qeedji is depending on the evolutions of its suppliers. This document contains confidential information; it can't be disclosed to any third parties without prior written authorization of Qeedji.

Safety instructions

Please read carefully the following instructions before switching the product on: - WARNING! Correct fitting and installation is of the utmost importance. Incorrect fitting and/or installation may result in personal injury or loss. Qeedji disclaims all liability, of whatever kind, if the product is assembled, fitted and/or installed in an incorrect manner. - Do not use the product near a water supply. - Do not pour anything on the product, like flammable liquids or material. - Do not expose the product to direct sun, near a heating source or a dust nor vibrations. - Do not obstruct holes, to be sure that air flows freely around the product. - Switch off the product during a storm. - Do not open the product in any circumstances.

Guarantee terms

Qeedji products are eligible for a warranty to cover genuine manufacturing defect for 3 years. Product failure occurring as the result of factors that do not constitute genuine manufacturing defect are not covered under the terms of the warranty and any repairs of this nature would be chargeable. For example: Inappropriate maintenance action, a non-authorized modification, a not specified environment utilization (see 'Safety instructions'), or if the product has been damaged after an impact, a fall, a bad manipulation or a storm consequence, an insufficient protection against heat, moisture or frost. This warranty is not transferrable. In addition, any repairs carried out by non-authorized personnel will invalidate the warranty.

WEEE Directive

This symbol means that your end of life equipment must not be disposed of with household waste but must be deposited at a collection point for waste electrical and electronic equipment or to your reseller. This will benefit the environment. In this context, a system for collecting and recycling has been implemented by the European Union

Table of content

Part I

Introduction 1.1
APK Development 1.2
Qeedji System Java API 1.2.1
Qeedji System JavaScript API 1.2.2
Demo APK 1.2.3
Screensaver demo APK 1.2.4
APK debug 1.2.5
System App 1.3
APK Signing 1.3.1
Set App as System App 1.3.2
Qeedji System service 1.4
Installation by USB 1.4.1
Installation by WebDAV 1.4.2
AOSP device mode 1.4.3
Qeedji preferences 1.5
FAQ 1.6

Part II

Contacts 2.1

Part III

Appendix: Web services 3.1
Appendix: Screensaver APK 3.2
Appendix: WebUI Extension APK 3.3
Appendix: Loss of interactivity when calling wakeUp 3.4
Appendix: HTTPS server for AQS device self-administration 3.5

1.1 Introduction

This documentation is intended for ISVs (`Independent Software Vendors`), wishing to develop `AOSP` `APKs` on `Qeedji` AMP300 devices.

> ⚠ Android `APK` development skills are required to go ahead.
>
> ⚠ It is recommended to read first the AMP300 user manual.

Demo Package Content

Items	Description	Quantity
AMP300	`Qeedji` media-player embedding `AQS` 9	1
USB Type-C cable	Cable - Assembly, Type-C Male to Type-A Male	1

1.2 APK Development

Prerequisite

The software developer needs to have *Java*, *JavaScript*, *HTML/CSS* skills to develop `Android APK`. The software developer already know how to develop, generate and debug an `Android APK` with `Android Studio`.

AOSP Standard API

The standard API of `AQS` 9.10.19 is based on the `AOSP` SDK 28.
The `AQS` 9.10.19 embeds `Chromium Web engine 83`.

The getDeviceId method of the TelephonyManager class allows to get the device identification UUID value.
The getSerial method of the Build class allows to get the raw PSN.
The BASE_OS static string of the Build.VERSION class allows to get the software version.

1.2.1 Qeedji System Java API

The `Qeedji` github hosts the tech-qeedji-system-lib-classes.jar Java library.

The tech-qeedji-system-lib-classes.jar library exposes a Java API for specific features.

```
public class DeviceMode {
    public static final int INVALID = -1;
    public static final int NATIVE = 0;
    public static final int KIOSK = 1;

    public DeviceMode(Context c);
    public void setValue(int devicemode);
    public int getValue();
    public void registerListener(DeviceModeListener listener);
    public void unregisterListener(DeviceModeListener listener);
}

public interface DeviceModeListener {
    public void onDeviceModeChanged(int oldDeviceMode, int newDeviceMode);
}

public class DisplayOutput {
    public static final int TYPE_UNKNOWN = 0;
    public static final int TYPE_INTERNAL = 1;
    public static final int TYPE_EXTERNAL = 2;

    public static final long PORT_TYPE_UNKNOWN = 0;
    public static final long PORT_TYPE_HDMI = 8;
    public static final long PORT_TYPE_USBC_DP = 11;

    public DisplayOutput(Context context);
    public int getType();
    public boolean getAutoPortType();
    public void setAutoPortType(boolean enable);
    public long[] getPortTypes();
    public long getPortType();
    public void setPortType(long portType);
    public String getLabelOfPort(long portType);
    public long getPortTypeFromLabel(String label);
    public boolean getAutoDisplayMode();
    public void setAutoDisplayMode(boolean enable);
    public DisplayOutputMode[] getDisplayModes();
    public DisplayOutputMode getDisplayMode();
    public void setDisplayMode(String labelMode);
    public long[] getRotations();
    public long getRotation();
    public void setRotation(long rotation);
    public boolean getDpmsPowerModeEnabled();
    public void setDpmsPowerModeEnabled(boolean enable);
}

public class DisplayOutputMode {
    public DisplayOutputMode(String label, long width, long height, long refresh_rate);
    public String getLabel();
    public long getWidth();
    public long getHeight();
    public long getRefreshRate();
    public boolean equals(DisplayOutputMode aMode);
}

public class PowerManager {
    public static final int STATE_DEVICE_SLEEP = 0;
    public static final int STATE_SCREEN_ON = 1;
    public static final int STATE_SCREEN_DIM = 2;
    public static final int STATE_SCREEN_SAVER = 3;

    public static final int DEVICE_SLEEP_LEVEL_MIN = 0;
    public static final int DEVICE_SLEEP_LEVEL_LOW = 1;
    public static final int DEVICE_SLEEP_LEVEL_NONE = 2;

    public PowerManager(Context c);
    public int getState();
    public int getDeviceSleepLevel();
    public void goToScreenOn();
    public void wakeUp();
    public void goToSleep();
}

public class PowerManagerCalendar {
    public PowerManagerCalendar(Context c);
    public void activate();
    public void deactivate();
    public boolean isActivated();
    public boolean hasCalendar();
    public int eventCount();
    public void registerListener(PowerManagerCalendarListener listener);
    public void unregisterListener(PowerManagerCalendarListener listener);
}

public interface PowerManagerCalendarListener {
    public void onActivate();
    public void onDeactivate();
}

public class RebootCalendar {
    public RebootCalendar(Context c);
    public void activate();
    public void deactivate();
    public boolean isActivated();
    public boolean hasCalendar();
    public int eventCount();
    public void registerListener(RebootCalendarListener listener);
    public void unregisterListener(RebootCalendarListener listener);
}

public interface RebootCalendarListener {
    public void onActivate();
    public void onDeactivate();
}

public abstract class SharedPreferenceAPI {
    protected abstract String getPreferenceAuthority();
    protected abstract Object[][] initPreferences();
}
```

```
    public class SoundOutput {
        public static final long PORT_TYPE_AUTO = 0;
        public static final long PORT_TYPE_VIDEO = 1;
        public static final long PORT_TYPE_USB_ADC = 3;
        public static final long PORT_TYPE_BT = 4;

        public SoundOutput(Context context);
        public long[] getPortTypes();
        public long getPortType();
        public void setPortType(long portType);
    }
```

Several demo APK are available on github. For further information, refer to the chapter § Demo APK.

Shared preferences API

An APK can be designed to share some preferences which can be then read or written by the `Qeedji System` service:

- either through the configuration script,
- or through the device configuration Web interface.

The APK must link the tech-qeedji-system-lib-classes.jar Java library.

The APK must also implement a child class of the *SharedPreferenceAPI* class supported in the tech-qeedji-system-lib-classes.jar Java library.

```
package tech.qeedji.test1;
import tech.qeedji.system.lib.SharedPreferenceAPI;
public class MySharedPreferenceAPI extends SharedPreferenceAPI {

    @Override
    public String getPreferenceAuthority() {
        return BuildConfig.APPLICATION_ID;
    }

    @Override
    public Object[][] initPreferences() {
        Object[][] preferences = {
            // filename, key, access, type, default_value
            {"test1", "test1Integer", "rw", int.class, 1},
            {"test1", "test1Long", "rw", long.class, 10000000L},
            {"test1", "test1Float", "rw", float.class, 0.897546F},
            {"test1", "test1Boolean", "rw", boolean.class, true},
            {"test1", "test1String", "rw", String.class, "test1StringValue"},
            {"test1", "test1StringSet", "rw", String[].class, new String[]{"http://Val0", "http://Val1", "http://Val2"}},
        };
        return preferences;
    }
}
```

The APK must declare a provider in its manifest.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="tech.qeedji.test1">

    <application
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <provider android:name="tech.qeedji.test1.MySharedPreferenceAPI"
            android:authorities="tech.qeedji.test1"
            android:multiprocess="false"
            android:exported="true"
            android:singleUser="false"
            android:initOrder="100"
            android:visibleToInstantApps="true"/>
    </application>
</manifest>
```

Extract example from a configuration script:

```
Android.Preferences("SharedPreferences", "tech.qeedji.test1", "test1").setInt("test1Integer", 8);
Android.Preferences("SharedPreferences", "tech.qeedji.test1", "test1").setLong("test1Long", 99999999);
Android.Preferences("SharedPreferences", "tech.qeedji.test1", "test1").setFloat("test1Float", 0.123456);
Android.Preferences("SharedPreferences", "tech.qeedji.test1", "test1").setBoolean("test1Boolean", false);
Android.Preferences("SharedPreferences", "tech.qeedji.test1", "test1").setString("test1String", "Newtest1StringValue");
Android.Preferences("SharedPreferences", "tech.qeedji.test1", "test1").setStringArray("test1StringSet", ["http://NewVal0", "http://NewVal1", "http://NewVal2"]);
```

1.2.2 Qeedji System JavaScript API

The `Qeedji` github hosts the tech-qeedji-host-webview.aar Android library.

The tech-qeedji-host-webview.aar Android library exposes a JavaScript API for specific functionalities in a WebView.

The tech-qeedji-host-webview.aar Android library embeds the tech-qeedji-system-lib-classes.jar library.

```
    String Host.Bluetooth().getHardwareAddress();

    String Host.Device().getModel();
    String Host.Device().getManufacturer();
    String Host.Device().getSerial();
    String Host.Device().getPsn();
    String Host.Device().getSoftwareVersion();
    String Host.Device().getDeviceName();
    String Host.Device().getUUID();
    String Host.Device().getHostname();
    String Host.Device().getMacId();
    String Host.Device().getField1();
    String Host.Device().getField2();
    String Host.Device().getField3();
    String Host.Device().getField4();
    String Host.Device().getField5();

    int Host.NetworkInterfaces().length();
    NetworkInterface Host.NetworkInterfaces().get(int index);
    String NetworkInterface.getName();
    String NetworkInterface.getHardwareAddress();
    boolean NetworkInterface.isUp();

    void Host.PowerManager().goToSleep();
    void Host.PowerManager().wakeUp();
    void Host.PowerManager().reboot();
    void Host.PowerManager().keepScreenOn(boolean value);
    void Host.PowerManager().goToScreenOn();

    void Host.SystemButton().start();
    void Host.SystemButton().stop();
    callback : void onHostSystemButtonReading(int count);
```

Several demo APK are available on github. For further information, refer to the chapter § Demo APK.

1.2.3 Demo APK

The `Qeedji` github for AMP300 hosts demo APK using the `AOSP` `SDK` for AMP300:

> Autorestart

The `Autorestart` APK is launched automatically after a device reboot. It is relaunched also automatically after it crashes.
This APK uses the BroadcastReceiver and Thread.UncaughtExceptionHandler APIs.
The RECEIVE_BOOT_COMPLETED permission is required.

> Device Power Standby

The `Device Power Standby` APK allows to go into, or exit from, the Android *sleep* state. In the *sleep* state, the display is black and the touch screen is inactivated.
This APK uses the PowerManager API.
The DEVICE_POWER and WAKE_LOCK permissions are required.
This APK requires system user execution rights.

> System Button

The `System Button` APK prints a notification message when a short press on the system button, during less than two seconds, is detected.
This APK uses the BroadcastReceiver API.
This APK requires system user execution rights.

> URL Webview

The `URL Webview` APK plays the content of a Web site URL.

This APK uses the WebView API and the tech-qeedji-host-webview.aar Android library for Qeedji.
The RECEIVE_BOOT_COMPLETED and INTERNET permissions are required.
This APK requires system user execution rights.
A specific 000000000000.js configuration script allows to configure the URL Webview APK.

> Power Manager Calendar

The `Power Manager Calendar` APK allows to sleep the device at *10:00 AM* (hard coded) and wake up the device at *10:15 AM* (hard coded), by taking care to deactivate the OS power manager task. It allows also to reboot the device at *6:00 AM* (hard coded) by taking care to *deactivate* the OS reboot task. In case the APK is notified from any OS power manager tasks and OS reboot tasks activation, the APK forces to *deactivate* the OS power manager tasks and OS reboot tasks.
When the APK is running, the text *Hello Qeedji!* is printed on the screen.
This APK uses the `PowerManagerCalendar` API to *deactivate* the OS power manager tasks. It uses also the `PowerManagerCalendarListener` API to be notified from any OS power manager tasks activation.
This APK uses the `RebootCalendar` API to *deactivate* the OS reboot tasks. It uses also the `RebootCalendarListener` API to be notified from any OS reboot tasks activation.
This APK uses the `PowerManager` API to *goToSleep* and *wakeUp* the device.
This APK requires system user execution rights.

WebUI Extension

The `WebUI Extension` APK supports an APK configuration form using admin preferences and an APK application form using applicative preferences.
This APK uses the SharedPreferenceAPI API.
For further information refer to appendix linked to the chapter § Appendix: WebUI Extension APK.

For further information, refer to the chapter § Appendix: WebUI Extension APK .

> ⚠ Designing an APK, requiring `system user` execution rights, requires for ISV to either sign its APK with a `Java Keystore` having a `certificate` signed by `Qeedji` or set its APK as `system App` . For further information, refer respectively to the chapter § APK Signing and to the chapter § Set App as System App.

1.2.4 Screensaver demo APK

> Screensaver for AV stream reader

The `Screensaver for AV stream reader` APK plays an AV stream.
This APK implements a DreamService.
The BIND_DREAM_SERVICE and INTERNET permissions are required.
A specific 000000000000.js configuration script allows to activate and configure the `Screensaver for AV stream reader` APK.
This APK requires system user execution rights.

For further information, refer to the chapter § Appendix: Screensaver APK.

> ⚠ Designing an APK, requiring `system user` execution rights, requires for ISV to either sign its APK with a `Java Keystore` having a `certificate` signed by `Qeedji` or set its APK as `system App`. For further information, refer respectively to the chapter § APK Signing and to the chapter § Set App as System App.

1.2.5 APK debug

The `AQS` Operating system for the AMP300 device is compatible with Android Studio and Android Debug Bridge (ADB)² software development suite.

²*ADB is included in the Android SDK Platform-Tools package*.

You can debug with ADB using:

- `USB`,
- `WLAN`,
- `LAN`.

USB debug

> Connect a cable between the USB-C connector of the AMP300 device and the USB 2.0 connector of your computer. Then wait for the AMP300 device is booting up.

Before supplying the AMP300 device with the USB connector of your computer, check with your computer's manufacturer that its USB connectors are protected against over-intensity to warranty that its USB output will be never damaged. Check also that the USB output is able to deliver a sufficient power.

WLAN debug

> Connect the power cable of your USB-C wall plug to one of the USB-C connectors of the AMP300 device. Then wait for the AMP300 device is booting up. Then, either connect a RJ45 cable supporting network connectivity then use the device configuration Web user interface to configure the `WLAN` interface then reboot and unplug the RJ45 cable. If you have a touch screen display device connected to the AMP300 device, go in the `Settings` application then configure the `WLAN` interface, or inject a configuration script having a suitable `WLAN` configuration with an USB-C storage device.

LAN debug

> Either connect an RJ45 cable supporting network connectivity then connect the power cable of your USB-C wall plug to the USB-C connector of the AMP300 device then wait for the AMP300 device is booting up or connect a RJ45 cable supporting network connectivity and PoE to the RJ45 connector of the AMP300 device then wait for the AMP300 device is booting up.

Debug mode setting
The debug mode setting required to have a touch screen display device connected to the AMP300 device.

Launch the `Settings` application:

- press on the `About tablet` menu,
- press 5 times on the button `Build number (9.yy.zz release keys)`. The message *You are now a developer* should appear showing that the debug mode is activated,
- go in the `Advanced` item of the `System` menu. The `Developer options` menu is now available,
- activate `Network debugging` or `USB debugging` according to your needs.

☛*To activate the debug mode on the AMP300 device, download the configuration script available on the Qeedji github, and uncomment the line `enableAllowDeveloperOptions();`. To inactivate the debug mode, comment the previous line, and uncomment the line `disableAllowDeveloperOptions();`. Then to inject this `.js` configuration script, refer to the chapter § Installation by USB and to the chapter § Installation by WebDAV.*

Network access permissions

To access the network, an APK needs to declare INTERNET and ACCESS_NETWORK_STATE permissions in its manifest:

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Android Studio version

Some very recent version of `Android Studio` may prevent to debug `APK` requiring `system user` execution rights. Indeed, in this case, when trying to launch a debug session on the device, this message is prompted:

Example of message:

```
10/17 10:25:47: Launching 'app' on Qeedji AMP300.
Couldn't terminate the existing process for <package name>.
```

The problem can be random or systematic. To work around this trouble, the developer can:

> 1. either uninstall the App from the device, reboot the device then launch a debug session again each time the problem happens (solution not advised for the developer),
> 2. or use an older version of `Android Studio` (ex : *3.4.2* or below) (solution advised for the developer).

Example of `Android Studio` working properly: *android-studio-ide-183.5692245-windows.exe*

```
Build #AI-183.6156.11.34.5692245, built on June 27, 2019
JRE: 1.8.0_152-release-1343-b01 amd64
JVM: OpenJDK 64-Bit Server VM by JetBrains s.r.o
Windows 10 10.0
```

1.3 System App

A `System App` is an Android notion telling that the `APK` requires `system user` execution rights to be executed.

An `APK` developped by an `ISV` becomes `System App` as soon as this `APK` uses some specific `AOSP` features or some specific `AQS` features requiring `system user` execution rights.

> ☛ *If the `ISV` designs its `APKs` to not use these specific `AQS` features requiring `system user` execution rights, no specific signing procedure with Qeedji CSR is required.*

In this `AQS` version, this is the exhaustive list of `AQS` features requiring automatically `system user` execution rights:

- *Device Mode*: feature allowing to set dynamically the device mode of the AQS into `kiosk` mode or into `native` mode,
- *System Button*: feature allowing to be notified when the system button is pushed (short push, long push),
- *Display Output Setting*: feature allowing to command the video output:
  - USB-C or HDMI,
  - automatic resolution or predefined resolution,
- *Pptx*: feature allowing an App using `Android System WebView` to play MS-PowerPoint medias (`.ppsx` and `.pptx`),
- *Pdfjs*: feature allowing an App using `Android System WebView` to play PDF medias (`.pdf`),
- *setAppAsSystemApp*: feature granting `system user` execution rights for a list of app,
- *goToScreenOn*: feature allowing to prevent the screensaver to be launched.

In this `AQS` version, some native `AOSP` features require automatically `system user` execution rights. The list of features is not exhaustive.

- *Device Power Standby*: feature allowing to put the connected display device into standby or to wake up the connected display device,
- *Reboot*: feature allowing to launch a device reboot,

This is the exhaustive `AOSP` features that do not require `system user` execution rights.

- *SharedPreferenceAPI*: feature allowing to make shared some preferences,
- *Autorestart*: feature allowing to `autostart` the `APK` after the device has reboot.

To be able to execute `APK` requiring `system user` execution rights:

- the `ISV` must first create a `public certificate key` (.pk12) with a `CSR` ,
- then the `ISV` has two ways to finalize the procedure:
  - either signing its `APK` by using its `System Java Keystore` (.jks) with its `public certificate key` (.pk12),
  - or declaring a list of `APK` to be granted to `System App` , stored in a .xml file signed with its `public certificate key` (.pk12) with the *AQS-setAppAsSystemApp* PowerShell tool.

It is possible to check whether an `APK` has been granted as `System App` by taking a look at the `logcat` traces. Indeed, the `<myApplicationId>/1000` trace is printed as soon as the App uses the `system user id` , meaning that the `APK` has been granted as `System App` .

Procedure to create a public certificate key (.pk12) with a CSR

> ☛ *In the example, it is considered that the company name is Contoso. ISD means IT Service Department. In the procedure, it is required to use the generic email of the Chief Information Security Officer (CISO) of the company, for example `ciso@contoso.com` .*
>
> ⚠ In the following procedure, the example values have been used.
>
> | Label type | Label value examples |
> |---|---|
> | C | US |
> | ST | California |
> | L | San-Francisco |
> | O | Contoso |
> | OU | Contoso_ISD |
> | CN | contoso.com |
> | E | ciso@contoso.com |
> | Passphrase | 1234 |
> | Java_keystore basename file | contoso_qeedji_java_keystore |
> | Java_keystore password | 567890 |
> | Friendly_name / name / key_alias | qeedji_aosp_key |

1 . GENERATE YOUR PRIVATE KEY

> ⚠ *You are responsible for your private key storing which has to be never communicated to a third party.*
> Generate your private key with a length of 2048 bits with the *RSA 2048 Bits* key type.
> For example:
> **openssl genrsa -f4 2048 > contoso_private_key_for_android.key**

2 . GENERATE YOUR OWN CSR (CERTIFICATE SIGNING REQUEST)

> Generate your own `.csr` certificate signing request thanks to your private key and some applicant identification used to digitally sign the request. Thanks to match the filename pattern by replacing contoso by your own organization name.
> For example:
> **openssl req -new -key contoso_private_key_for_android.key -subj '/C=US/ST=California/L=San-Francisco/O=Contoso/OU=Contoso_ISD/CN=contoso.com/emailAddress=ciso@contoso.com' > contoso-for_qeedji_aosp.csr**

3 . SEND YOUR CSR TO QEEDJI

> Once generated, send a email to the *csr@qeedji.tech* with your `CSR` (`contoso-for_qeedji_aosp.csr` file for example) in attachment.

4 . WAIT FOR THE QEEDJI ANSWER

> `Qeedji` should then return an answer within 7 days.
> ⚠ *Qeedji will send its answer to the email defined into the CSR file (ciso@contoso.com for example), which may be not the same email used to send the CSR to Qeedji.*
> `Qeedji` sends two files: the signed certificate (extension .crt) and the CA file (extension .pem).
> For example:
>
> - `contoso-qeedji_aosp-certificate-001A.crt` ,
> - `contoso-qeedji_aosp-certificate_authority-001A.pem`

5 . GENERATE YOUR PUBLIC CERTIFICATE KEY

> You have first to generate your public certificate key. For example:
> **openssl pkcs12 -export -in contoso-qeedji_aosp-certificate-001A.crt -inkey contoso_private_key_for_android.key -out contoso_certificate_and_key_for_qeedji_aosp.pk12 -password pass:1234 -name qeedji_aosp_key -chain -CAfile contoso-qeedji_aosp-certificate_authority-001A.pem**
>
> ⚠ *In case the security conditions or the saling conditions are not fully filled, Qeedji keeps the rights to revocate a ISV certificate.*

Now your public certificate key is generated. You can go to the next signing step.

1.3.1 APK Signing

With this signing procedure, the `system Java Keystore` (.jks) must be generated by the `ISV` with its public certificate key (.pk12).

Prerequisite: the steps to generate a `public certificate key` (.pk12) have been done once by the `ISV` .

6 . GENERATE THE JAVA KEYSTORE

> Generate then a `Java Keystore` from your public certificate key with the `keytool` [1] toolbox.
> The `Java Keystore` system is now usable in `Android Studio` .
> For example:
> **keytool -importkeystore -deststorepass 567890 -storetype JKS -destkeystore contoso_qeedji_java_keystore.jks -srckeystore contoso_certificate_and_key_for_qeedji_aosp.pk12 -srcstoretype PKCS12 -srcstorepass 1234**

[1] *Keytool is a toolbox to handle certificates for Java products. It is provided by default in the JDK since version 1.1.*

> ☛ *The `ISV` must use its own `Java Keystore` for all their `APK` requiring `system user` execution rights and the same certificate for all their `AQS` devices. When signing is required for your `APK` , the `ISV` must follow, at least once per `APK` , this procedure to create its system Java Keystore.*

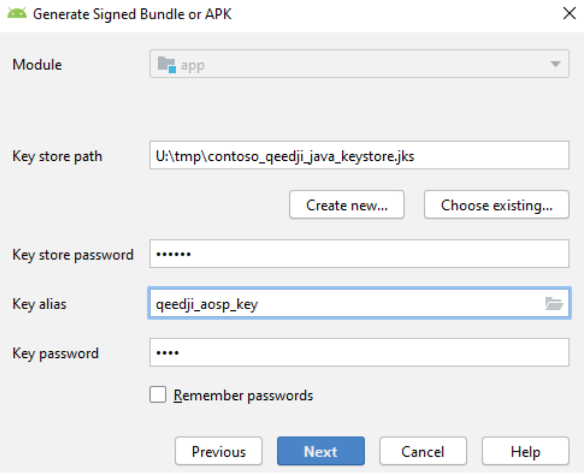
7 . MODIFY THE MANIFEST

> Modify the APK manifest by adding this string: `android:sharedUserId="android.uid.system"`
> Sample manifest (*AndroidManifest.xml* file):

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="tech.qeedji.reboot"
    android:sharedUserId="android.uid.system">
    <uses-permission android:name="android.permission.REBOOT" />
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

8 . SIGN THE APPLICATION WITH YOUR SYSTEM JAVA KEYSTORE

> When creating the APK, sign the APK with your own System Java Keystore (.jks).


Generate Signed Bundle or APK

Module: app

Key store path: U:\tmp\contoso_qeedji_java_keystore.jks

Key store password: ••••••

Key alias: qeedji_aosp_key

Key password: ••••

□ Remember passwords

With the previous example, you would have to use the following parameters values:

- Key store password = 567890
- Key password = 1234

The signing procedure is now over. Your `APK` requiring `system user` execution rights can now be installed then executed on your `AQS` devices.

1.3.2 Set App as System App

This procedure allows to declare a list of `APK` to be granted as `System App`, stored in a .xml file signed with the `ISV` `public certificate key` (.pk12) with the *AQS-setAppAsSystemApp* PowerShell tool.

> ☛ *With this procedure, there is no need to use your `System Java Keystore`.*

Prerequisite: the steps to generate a `public certificate key` (.pk12) have been done once by the `ISV`.

6 . GET YOUR APK APPLICATION ID

> This is an example to get the `applicationId` of `APK` generated with the Gradle plugin for Android Studio: https://github.com/Qeedji/aosp-AMP300-sdk/blob/main/examples/device_power_standby/app/build.gradle.

```
android {
    compileSdkVersion 29
    buildToolsVersion "29.0.3"
    defaultConfig {
        applicationId "tech.qeedji.device_power_standby"
        minSdkVersion 28
        targetSdkVersion 29
        versionCode 2
        versionName "1.10.11"
        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
        }
    }
}
```

> This is an example to get the `applicationId` of `APK` delivered on the `Google Play` store. For example, for this Google Play application URL example, the applicationId is the suffix of the URL behind `id=` (in the example `com.wakdev.wdnfc`).
>
> If you are not the developer of the `APK`, if the `APK` is not available on the `Google Play` store, it may be requested to the `APK` developer to provide the `applicationId` of the `APK`.

7 . CONFIGURE THE POWERSHELL SCRIPT BY ADDING YOUR APK APPLICATION ID

> Download the `AQS-setAppAsSystemApp~001B.zip` archive from the Qeedji Website.
> Extract the archive in your favorite folder (for example `C:\Powershell-Script\AQS-setAppAsSystemApp\`) and open the folder. The folder contains two files:
>
> - `app-list.xml`,
> - `AQS-setAppAsSystemApp.ps1`.

> Edit the `app-list.xml` file and enter the respective `applicationId` of your different APKs between the `<AppId>` and `</AppId>` tags. The example is given for three fake `applicationId`. Remove the unconsistent lines.

```
<?xml version="1.0"?>
<AppList>
  <AppId>tech.qeedji.app1</AppId>
  <AppId>tech.qeedji.app2</AppId>
  <AppId>tech.qeedji.app3</AppId>
</AppList>
```

8 . COPY THE PUBLIC CERTIFICATION KEY (PK12 FILE) INTO THE POWERSHELL FOLDER

> Copy and paste your public certificate key (.pk12) (in the example: contoso_certificate_and_key_for_qeedji_aosp.pk12) in the `PowerShell` folder (in the example: `C:\Powershell-Script\AQS-setAppAsSystemApp\`)

9 . EXECUTE THE POWERSHELL SCRIPT

> Open a `PowerShell` command window and go into your `PowerShell` folder (in the example: `C:\Powershell-Script\AQS-setAppAsSystemApp\`). Execute the `PowerShell` script with the name of your own public certificate key (.pk12`) file, and the name of the xml files, as arguments. For example, with contoso_certificate_and_key_for_qeedji_aosp.pk12:

.\AQS-setAppAsSystemApp.ps1 -pk12File .\contoso_certificate_and_key_for_qeedji_aosp.pk12 -xmlFile .\app-list.xml -outputFile .\app-list-signed.xml

> When asked, enter the password for your public certificate key (.pk12).

10 . EDIT THE FILE GENERATED WITH THE POWERSHELL SCRIPT AND COPY THE CONTENT

> Edit the generated `app-list-signed.xml` file and copy the entire file content.

11 . CONFIGURE A CONFIGURATION SCRIPT FOR THE DEVICE

> Open the configuration script example on the Qeedji Website. Uncomment the two lines below and replace the value `<?xml version="1.0"?><AppList>...` of the `xmlSignedFileData` variable by the entire file content of the of the `app-list-signed.xml` file.

```
let xmlSignedFileData = `<?xml version="1.0"?><AppList><AppId>...`;
setAppAsSystemApp(xmlSignedFileData);
```

> ☛ *The configuration script version must be V1.10.17 (or above).*
>
> ☛ *It may be required to inject the script containing the setAppAsSystemApp(...) before being able to install your APK.*

12 . INJECT THE CONFIGURATION SCRIPT IN THE DEVICE

> Copy the configuration script in the `.configuration` `WebDAV` directory of the device or copy the configuration script on an USB storage device and inject it on an USB connector of the device. After the automatic device reboot, your `APK` requiring `system user` execution rights, whose `applicationId` is declared in the `app-list-signed.xml` file, should be executed properly on the device.
>
> ⚠ After configuration script installation, if an `APK` concerned by the list was already installed on the device, it will lose all its user parameters. This is also the case when an `APK` is removed from the list. Qeedji advises you to install the configuration script first, and afterwards install the `APK` that will have consequently the `system user` execution rights granted.
>
> ☛ *To return to a configuration that does not grant `system user` execution rights for `APK` anymore, the `app-list-signed.xml` must be generated with an empty `app-list-signed.xml` content like shown below.* `xml <?xml version="1.0"?><AppList></AppList>`

1.4 Qeedji System service

The AMP300 device embeds the `Qeedji System` service. The `Qeedji System` service is defined as a privileged application of the `AQS` device.

This service allows to:

- install one or more `APK` on the AMP300 device:
    - by uploading an APK with the device Web user interface,
    - by pushing an APK on the `.apps/` directory of the `WebDAV` server with a `WebDAV` client,
    - by inserting an USB storage device containing `.apk` files,
- update the `AQS` operation system of the AMP300 device:
    - by uploading a `.fqs` firmware with the device Web user interface,
    - by pushing a `.fqs` firmware on the `.software/` directory of the `WebDAV` server with a `WebDAV` client,
    - by inserting an USB storage device containing an `.fqs` file,
- configure the AMP300 device thanks to a suitable `.js` configuration script:
    - by pushing a suitable `.js` configuration script on the `.configuration/` directory of the `WebDAV` server with a `WebDAV` client,
    - by inserting an USB storage device containing an `.fqs` file,
    - by getting `.js` configuration script hosted on a `TFTP` server (`DHCP` , *code 66*),
- push your data on the `.data/` directory of the `WebDAV` server with a `WebDAV` client.

This service allows also to configure the AQS device mode as soon as the device has started.

1.4.1 Installation by USB

Refer to the AMP300 user manual to install, with an USB storage device:

- a new `APK` (.apk),
- a new `AQS` firmware (.fqs),
- a new configuration script (.js).

1.4.2 Installation by WebDAV

The available `WebDAV` directories are:

- `.apps` ,
- `.software` ,
- `.configuration` ,
- `.data` .

The connection profile (ex: *Administration user*, *Web Service* or *Publishing software* connection profile) used to push a file on a WebDAV directory must have access rights to push file on the WebDAV directories of the device.

Refer to the AMP300 user manual to install, with a `WebDAV` client:

- a new `APK` (.apk),
- a new `AQS` firmware (.fqs),
- a new configuration script (.js).

Examples using CURL

In the examples:

- the credential identifier/password of the user connection profile (ex: *Administration user*, *Web Service* or *Publishing software* connection profiles) having access rights to push on the WebDAV directories is `admin` / `admin` .
- the IPV6 address and the IPV4 address of the device are respectively [fe80::21c:e6ff:fe02:5694] and 192.168.8.201.

Example to push an APK on the `.apps/` WebDAV directory of the device:

```
curl -u admin:admin --digest -T "myAPKfile.apk" "http://192.168.8.201/.apps/"
```

```
curl -u admin:admin --digest -T "myAPKfile.apk" "http://[fe80::21c:e6ff:fe02:5694]/.apps/"
```

Example to push a configuration script on the `.configuration/` WebDAV directory of the device:

```
curl -u admin:admin --digest -T "000000000000.js" "http://192.168.8.201/.configuration/"
```

```
curl -u admin:admin --digest -T "000000000000.js" "http://[fe80::21c:e6ff:fe02:5694]/.configuration/"
```

Example to push a firmware on the `.software/` WebDAV directory of the device:

```
curl -u admin:admin --digest -T "aosp-amp300-setup-9.10.18.fqs" "http://192.168.8.201/.software/"
```

```
curl -u admin:admin --digest -T "aosp-amp300-setup-9.10.18.fqs" "http://[fe80::21c:e6ff:fe02:5694]/.software/"
```

Data

To push user data with a `WebDAV` client, drop them in the `.data/` directory of the `WebDAV` server.

> ☛ On the file system of the device, the `.data/` directory is `/storage/emulated/0/Android/data/tech.qeedji.system/files/.data` . This directory is available by apps with READ_EXTERNAL_STORAGE and WRITE_EXTERNAL_STORAGE permissions.

1.4.3 AOSP device mode

The `Qeedji System` service allows to configure the `AOSP device mode` dynamically. It is handled thanks to the `persist.sys.device_mode` system property, used by the *SystemUI* and *Launcher3* `AOSP` services.

The two supported values for the `persist.sys.device_mode` system property are:

- `native` (default value): thanks to `AOSP` menu, the user can, whenever he wants, stop the APK, returns to the `AOSP` home screen, launch another APK, access to `AOSP` functions like, for example, the *Back* button or the *Settings* application.
- `kiosk`: all the `AOSP` user interfaces are unavailable. However the `AOSP` virtual keyboard remains available.

> ☛ Note for developers: if the `persist.sys.device_mode` system property value is invalid, the default `AOSP` device mode is `native`. If the `persist.sys.device_mode` system property value is `kiosk`, the *SystemUI* service inhibits the *system bars* and the *Launcher3* service hides the *AllApps* view and the *OptionsPopupView* dialog box.

The `persist.sys.device_mode` system property can be changed by using the configuration script:

- `native`:

```
setDeviceModeNative(); /* default mode */
//setDeviceModeKiosk();
```

- `kiosk`:

```
//setDeviceModeNative(); /* default mode */
setDeviceModeKiosk();
```

For further information, refer to the AMP300 user manual.

[1] *To be launched automatically in kiosk mode, the application requires a subscription to the event ACTION_BOOT_COMPLETED. In this case, it is recommended to have only one* `APK` *with this subscription. For further information, refer to https://developer.android.com/reference/android/content/Intent#ACTION_BOOT_COMPLETED. In case no* `APK` *has subscribed to the event ACTION_BOOT_COMPLETED, the Qeedji wallpaper ² is displayed.*

[2] *A next* `AQS` *version will allow to load a custom wallpaper.*

1.5 Qeedji preferences

AOSP system properties added by Qeedji

Find hereafter the table of system properties added in the AOSP by Qeedji.

Name	Type	R/W	Default value	Values	Description
persist.sys.delivery-software-version	String	RO	9.10.18	<x>.<v>.<z><software-extraversion>	Allows to define the version of the last AQS firmware upgrade.
persist.sys.device_mode	String	RW	native	native , kiosk	Allows to define the AOSP device mode.
persist.sys.hostname	String	RW	AMP300	For example AMP300-00001	Allows to define the hostname whether the preference persist.sys.hostname.enabled is true.
persist.sys.hostname.enabled	Boolean	RW	false	true , false	Allows to activate/deactivate the hostname defined by the preference persist.sys.hostname.
persist.sys.webserver.http.port	Integer	RW	80	1 to 65535	Allows to define the port of the http server.
persist.sys.webserver.webdav.credential	String	RW	default		Allows to define the credential ID for the Publishing software connection profile.
persist.sys.webserver.webuiappli.credential	String	RW	default		Allows to define the credential ID for the Application user connection profile.
persist.sys.webserver.webuiadmin.credential	String	RW	default		Allows to define the credential ID for the Administration user profile.
persist.sys.webserver.webservice.credential	String	RW	default		Allows to define the credential ID for the Web Service profile.
persist.sys.webserver.webserviceappli.credential	String	RW	default		Allows to define the credential ID for the Web Service Appli profile.
persist.sys.device_info.field1	String	RW			Allows to define the custom device field1 variable value.
persist.sys.device_info.field2	String	RW			Allows to define the custom device field2 variable value.
persist.sys.device_info.field3	String	RW			Allows to define the custom device field3 variable value.
persist.sys.device_info.field4	String	RW			Allows to define the custom device field4 variable value.
persist.sys.device_info.field5	String	RW			Allows to define the custom device field5 variable value.
persist.sys.testcard.start_after_boot_completed	Boolean	R/W	true	true, false	Allows to activate/deactivate the tech.qeedji.testcard App launching after the device booting-up is completed.
persist.sys.connector.hdmi_1.all.tmds.dcu_1.enabled	Boolean	RW	true	true , false	Allows to activate/deactivate the video rendering on the hdmi_1 connector.
persist.sys.connector.usbc_1.all.dpmain.dcu_1.enabled	Boolean	RW	true	true , false	Allows to activate/deactivate the video rendering on the usbc_1 connector.
persist.sys.connector.hdmi_1.all.tmds.scu_1.enabled	Boolean	RW	true	false	Allows to activate/deactivate the sound rendering on the hdmi_1 connector.
persist.sys.connector.usbc_1.all.dpmain.scu_1.enabled	Boolean	RW	true	false	Allows to activate/deactivate the sound rendering on the usbc_1 connector.
persist.sys.display-output_1.auto_port_type.enabled	Boolean	RW	true	true , false	Allows to activate/deactivate the automatic display port type. The first screen connected is selected.

Name	Type	R/W	Default value	Values	Description
`persist.sys.display-output_1.auto_mode.enabled`	Boolean	RW	`true`	`false`	Allows to activate/deactivate the automatic display mode. The display mode is chosen thanks to the EDID get from the screen.
`persist.sys.display-output_1.mode`	String	RW	`1920x1080 60Hz SMPTE (1080p)`	`<modes list>`	Allows to define the display output mode.
`persist.sys.display-output_1.rotation`	Integer	RW	`0`	`90` , `180` , `270`	Allows to define the display output rotation.
`persist.sys.display-output_1.power-mode.dpms.enabled`	Boolean	RW	`true`	`true` , `false`	Allows to activate/deactivate the VESA DPMS on the HDMI video output.
`persist.sys.power-manager.level.min.display-output.power-mode` [1]	Integer	RW	`0`	`0` , `1`	Allows to define the power mode of the display output during a `power manager` task having a *min* profile.
`persist.sys.power-manager.level.min.sound-output.volume`	Integer	RW	`0`	`0` to `100`	Allows to define the volume of the sound output during a `power manager` task having a *min* profile.
`persist.sys.power-manager.level.min.sound-output.mute`	Boolean	RW	`true`	`true` , `false`	Allows to define the mute of the sound output during a `power manager` task having a *min* profile.
`persist.sys.power-manager.level.min.hid.pointer-event.enable`	Boolean	RW	`false`	`true` , `false`	Allows to activate/deactivate the HID pointer event during a *power manager* task having a *min* profile.
`persist.sys.power-manager.level.min.externalstorage.copy.enable`	Boolean	RW	`true`	`true` , `false`	Allows to activate/deactivate the file installation from an external storage device during a power manager task having a *min* profile.
`persist.sys.power-manager.level.min.screen-off-timeout`	Integer	RW	`60`	`10` to `86400`	Allows to define the duration during which the device can exit temporarily from a `power manager` task having the *min* profile after a tap or a short press on the *system button* done by the end-user.
`persist.sys.power-manager.level.low.display-output.power-mode` [1]	Integer	RW	`1`	`0` , `1`	Allows to define the power mode of the display output during a `power manager` task having a *low* profile.
`persist.sys.power-manager.level.low.sound-output.volume`	Integer	RW	`50`	`0` to `100`	Allows to define the volume of the sound output during a `power manager` task having a *low* profile.
`persist.sys.power-manager.level.low.sound-output.mute`	Boolean	RW	`false`	`true` , `false`	Define the mute of the sound output during a `power manager` task having a *low* profile.
`persist.sys.power-manager.level.low.hid.pointer-event.enable`	Boolean	RW	`true`	`true` , `false`	Allows to activate/deactivate the HID pointer event during a `power manager` task having a *low* profile.
`persist.sys.power-manager.level.low.externalstorage.copy.enable`	Boolean	RW	`true`	`true` , `false`	Allows to activate/deactivate the file installation from an external storage device during a power manager task having a *low* profile.
`persist.sys.power-manager.level.low.screen-off-timeout`	Integer	RW	`60`	`10` to `86400`	Allows to define the duration during which the device can exit temporarily from a `power manager` task having the *low* profile after a tap or a short press on the *system button* done by the end-user.
`persist.sys.power-manager.device-sleep.level.default`	String	RW	`min`	`min` , `low`	Allows to define the default power manager level for the `Device sleep` state.
`persist.sys.testcard.key-event.all.authorized`	Boolean	RW	`true`	`true` , `false`	Allows to activate/deactivate the test card launching by the key sequence [L,R,L,R] pressed in less than ten seconds thanks to an USB keyboard or a display device's remote control.
`persist.sys.av-cmd@tv.cec_1.hdmi_1.all.authorized`	Boolean	RW	`true`	`true` , `false`	Allows to activate/deactivate CEC over HDMI feature (CEC passthrough, AV command).
`persist.sys.av-cmd@tv.cec_1.hdmi_1.all.features.power-mode`	Boolean	RW	`true`	`true` , `false`	Allows to activate/deactivate the PowerMode AV command on CEC over HDMI.
`persist.sys.av-cmd.i2c_2.usbc_1.all.authorized`	Boolean	RW	`true`	`false` , `false`	Allows to activate/deactivate the AV command on DDC over USB-C (*USB1 DP*).
`persist.sys.av-cmd.i2c_2.usbc_1.all.features.power-mode`	Boolean	RW	`false`	`true` , `false`	Allows to activate/deactivate the PowerMode AV commands on DDC over USB-C (*USB1 DP*).
`persist.sys.av-cmd.i2c_2.usbc_1.all.features.brightness`	Boolean	RW	`false`	`true` , `false`	Allows to activate/deactivate the Brightness AV commands on DDC over USB-C (*USB1 DP*).

Name	Type	R/W	Default value	Values	Description
`persist.svs.power-manager.level.min.av-cmd.power-mode`	Integer	RW	`0`	`0` to `1`	Allows to define the level [0: OFF: 1: ON] for the Power Mode AV command for the *Very Highly Optimized* profile power manager task.
`persist.svs.power-manager.level.min.av-cmd.brightness`	Integer	RW	`0`	`0` to `100`	Allows to define the level for the Brightness AV command for the *Very Highly Optimized* profile power manager task.
`persist.svs.power-manager.level.low.av-cmd.power-mode`	Integer	RW	`1`	`0` to `1`	Allows to define the level [0: OFF: 1: ON] for the Power mode AV command for the *Highly Optimized* profile power manager task.
`persist.svs.power-manager.level.low.av-cmd.brightness`	Integer	RW	`80`	`0` to `100`	Allows to define the level for the Brightness from 0 to 100% AV command for the *Highly Optimized* profile power manager task.
`persist.svs.power-manager.level.none.av-cmd.brightness`	Integer	RW	`100`	`0` to `100`	Allows to define the brightness level from 0 to 100% for Brightness AV command in nominal mode.

`power-mode` [1] *0 : POWERMODE_OFF (i.e. display output is off) 1 : POWERMODE_ON (i.e. display output is on)*

Settings preferences added by Qeedji

Name	Namespace	Type	R/W	Default value	Values	Description
`developer_options_allowed`	secure	Integer	RW	`0`	`0` , `1`	Allows to activate/deactivate debug mode (`1` = activated).
`adb_tcp_enabled`	global	Integer	RW	`0`	`0` , `1`	Allows to activate/deactivate *adb* over network (`1` = activated).
`adb_tcp_port`	global	Integer	RW	`5555`	`0` to `64738`	Allows to define the TCP port for *adb*.
`ptp_allowed`	global	Integer	RW	`0`	`0` , `1`	Allows to activate/deactivate the PTP (Picture Transfer Protocol).
`screen_stay_on`	system	Integer	RW	`1`	`0` , `1`	Allows to force the screen to stay in state `Screen On` except when an App or the `PowerManagerCalendarService` asks to go to sleep (`1` = never go in Sleep mode).
`screen_brightness_dim`	system	Integer	RW	`1`	`0` to `100`	Allows to define the screen brightness in `Screen DIM` state.

Preferences modified by Qeedji System service

Name	Namespace	Type	R/W	Default value	Values	Description
`settings.secure.android_id`	secure	String	R	depends on device PSN value	Value example (hexa): `087600AF`	This hexadecimal and 64 bits value is computed from the unic value of the device's PSN (product Serial Number). The `AQS` operating system reinitializes this preference to the right value, at device start-up each time the value has been erased, for example, after the user has cleared the user data partition.

Shared preferences for Qeedji System service

Namespace	Name	Type	R/W	Default value	Values	Description
`tech.qeedji.system.app`	`externalstorage.copy.apk.enabled`	Boolean	RW	`true`	`true` , `false`	Allows to activate/deactivate the `APK` installation from an USB storage device with a `.fqs` file stored at the root of the file system of the USB stick (`true` = activated).

Shared preferences for Qeedji Webserver

Namespace	Filename	Name	Type	R/W	Default value	Values	Description
`tech.qeedji.webserver.app`	`credentials`	`*.username`	String	RW			Allows to define a credential username.
`tech.qeedji.webserver.app`	`credentials`	`*.password`	String	RW			Allows to define a credential password.
`tech.qeedji.webserver.app`	`credentials`	`*.type`	String	RW	`user-password`	`user-password`	Allows to define the credential type.

Shared preferences for the URL Webview APK

The shared preferences for `URL Webview APK` is stored in the `tech.qeedji.url_webview.prefs.xml` file. In case login credentials are required to connect to the URL, an additional shared preferences `tech.qeedji.url_webview.credential.<credential_label>.prefs.xml` file is required.

> ⚠ The shared preferences files for URL Webview `APK` must be created and updated with the specific 000000000000.js configuration script.

`tech.qeedji.url_webview.prefs.xml`	Type	R/W	Default value	Values	Description

`url`	String	RW		for example:
				`https://www.demo.contoso.com/`
					Allows to define the URL of the Web page or Web site to view.

`start_after_boot_completed`	Boolean	RW	`true`	`true` , `false`	Allows the APK to start automatically after the `AQS` has started (`true` = start automatically after the `AQS` has started).
`autorefresh_url_enabled`	Boolean	RW	`false`	`true` , `false`	Allows to activate/deactivate the periodic URL page reloading.
`autorefresh_url_interval`	Long	RW	`60`	`1` to `86400`	Allows to define the URL page reloading period in seconds.
`credential`	String	RW		for example: If `<credential_label>` is `native` , the value is `native`	Allows to define the subpart of the expected filename for the additional file required when login credentials are needed to connect to the URL.

> ☛ *The `<credential_label>` subpart of the filename is defined in the `tech.qeedji.url_webview.prefs.xml` file above.*

`tech.qeedji.url_webview.credential.<credential_label>.prefs`	Type	R/W	Default value	Values	Description
`type`	String	RW	`native`	`native` , `user-password-webpageform`	Allows to define the credential type: `native` for identifier/password for a basic authentication, `user-password-webpageform` for identifier/password for a Webpage form.
`username`	String	RW			Allows to define the credential username to access to the Web page content.
`password`	String	WO			Allows to define the credential password to access to the Web page content.

1.6 FAQ

How to launch an App in `kiosk` device mode?

Qeedji implements an demo `APK` named URL Webview. Have a look at the two files below:

- AndroidManifest.xml,
- StartActivityAtBootReceiver.java.

For further information, refer to the chapter § AOSP device mode.

Is it possible to download a configuration script and .apk files from a remote server?

No, but you can develop your own `APK` to do this.

2.1 Contacts

For further information, please contact us by e-mail:

- **Technical support**: support@qeedji.tech,
- **Sales department**: sales@qeedji.tech.

Refer to the `Qeedji` Website for FAQ, application notes, and software downloads:

https://www.qeedji.tech/

```
Qeedji FRANCE
INNES SA
5A rue Pierre Joseph Colin
35700 RENNES

Tel: +33 (0)2 23 20 01 62
Fax: +33 (0)2 23 20 22 59
```

3.1 Appendix: Web services

To access to the Web services, you must use the credential of one of these user connection profiles:

- *Administration web Service*,
- *Administration user*.

OData API

The Web services can be used with an *OData* API supported by the `AQS` operating system.

☛ *To support OData API, the Qeedji device must have an* `AQS` *version 9.10.11 (or above).*

OData data model

The documentation related to the *OData* data model can be shown by entering this url syntax in a Web browser.
`http://<device_IP_address>/odata.qs/v1/$metadata`

> ☛ `<device_IP_address>` *is the IPV4 or IPV6 address of the device.*

OData singletons

The available *OData* API singletons can be get by entering this url syntax in a Web browser.
`http://<device_IP_address>/odata.qs/v1/`

> ☛ `<device_IP_address>` *is the IPV4 or IPV6 address of the device.*

Result example:

```
{"@odata.context":"/odata.qs/v1/$metadata","value":
    [
        {"name":"WifiConfiguration","url":"WifiConfiguration","kind":"EntitySet"},
        {"name":"WifiScan","url":"WifiScan","kind":"EntitySet"},
        {"name":"CACertificates","url":"CACertificates","kind":"EntitySet"},
        {"name":"WebserverCredentials","url":"WebserverCredentials","kind":"EntitySet"},
        {"name":"SecurePreferences","url":"SecurePreferences","kind":"EntitySet"},
        {"name":"WebserverProfiles","url":"WebserverProfiles","kind":"Singleton"},
        {"name":"Servers","url":"Servers","kind":"Singleton"},
        {"name":"SupportedLanguages","url":"SupportedLanguages","kind":"EntitySet"},
        {"name":"SupportedDisplayRotations","url":"SupportedDisplayRotations","kind":"EntitySet"},
        {"name":"GlobalPreferences","url":"GlobalPreferences","kind":"EntitySet"},
        {"name":"InfosDisplayOutput","url":"InfosDisplayOutput","kind":"EntitySet"},
        {"name":"SupportedTimezones","url":"SupportedTimezones","kind":"EntitySet"},
        {"name":"InfosGeneral","url":"InfosGeneral","kind":"Singleton"},
        {"name":"GeneralSettings","url":"GeneralSettings","kind":"Singleton"},
        {"name":"SystemProperties","url":"SystemProperties","kind":"EntitySet"},
        {"name":"SystemPreferences","url":"SystemPreferences","kind":"EntitySet"},
        {"name":"SupportedDisplayPortTypes","url":"SupportedDisplayPortTypes","kind":"EntitySet"},
        {"name":"WifiCertificateUserCredentials","url":"WifiCertificateUserCredentials","kind":"EntitySet"},
        {"name":"SystemAdapters","url":"SystemAdapters","kind":"EntitySet"},
        {"name":"InfosUsbAdapters","url":"InfosUsbAdapters","kind":"EntitySet"},
        {"name":"SystemMaintenance","url":"SystemMaintenance","kind":"Singleton"},
        {"name":"InfosNetworkIf","url":"InfosNetworkIf","kind":"EntitySet"},
        {"name":"SupportedDisplayModes","url":"SupportedDisplayModes","kind":"EntitySet"},
        {"name":"VpnCertificateUserCredentials","url":"VpnCertificateUserCredentials","kind":"EntitySet"}
    ]
}
```

Examples using CURL to get device's configuration values

In the example,

- the IPV6 address and the IPV4 address of the device are respectively [fe80::21c:e6ff:fe02:5694] and 192.168.8.201,
- the credentials of the connection profile to access to the Web Services are `admin` / `admin`.
  Example to get the values of all the properties of a given singleton (ex: *InfosGeneral* singleton):

```
curl -u admin:admin --digest --request GET "http://192.168.8.201/odata.qs/v1/InfosGeneral"
```

```
curl -u admin:admin --digest --request GET "http://[fe80::21c:e6ff:fe02:5694]/odata.qs/v1/InfosGeneral"
```

Result example:

```
{
    "@odata.context":"/odata.qs/v1/$metadata#InfosGeneral/$entity",
    "@odata.type":"#Infos.General",
        "Uuid":"08760008-0000-0000-0000-001ce6025a7b",
        "Psn":"01620-00008",
        "Platform":"AMP300",
        "Version":"9.10.18",
        "Mac":"00:1C:E6:02:5A:7B",
        "Hostname":"Display-floor1"
}
```

Example to get the *psn* property of the *InfosGeneral* singleton:

```
curl -u admin:admin --digest --request GET "http://192.168.8.201/odata.qs/v1/InfosGeneral/psn"
```

```
curl -u admin:admin --digest --request GET "http://[fe80::21c:e6ff:fe02:5694]/odata.qs/v1/InfosGeneral/psn"
```

Result example:

```
{"@odata.context":"/odata.qs/v1/$metadata#InfosGeneral/Psn",
    "value":"01356-00008"
}
```

Example to get the value of the *psn* property of the *InfosGeneral* singleton:

```
curl -u admin:admin --digest --request GET "http://192.168.8.201/odata.qs/v1/InfosGeneral/psn/$value"
```

```
curl -u admin:admin --digest --request GET "http://[fe80::21c:e6ff:fe02:5694]/odata.qs/v1/InfosGeneral/psn/$value"
```

Result example:

```
01356-00008
```

Example using CURL to configure the device then reboot

In the example,

- the IPV6 address and the IPV4 address of the device are respectively [fe80::21c:e6ff:fe02:5694] and 192.168.8.201,
- the credentials of the connection profile to access to the Web Services are `admin` / `admin` .

The example below shows how to:

- get the values of all the properties for the *GeneralSettings* singleton,
- update one of the *property* value (ex: *DeviceName*) of the *GeneralSettings* singleton with a patch,
- check the new values of all the properties for the *GeneralSettings* singleton,
- launch a device reboot with a patch.

Example to get the current values of all the properties for the *GeneralSettings* singleton:

```
curl -u admin:admin --digest --request GET http://192.168.8.201/odata.qs/v1/GeneralSettings
```

Result example:

```
{
    "@odata.context":"/odata.qs/v1/$metadata#GeneralSettings/$entity",
    "@odata.type":"#GeneralSettings.GeneralSettings",
    "ForcedHostname":"AMP300",
    "ForcedHostnameEnabled":false,
    "DeviceName":"Display-floor1",
    "Regionality":{"Languages":["fr-FR","en-US","de-DE","es-ES","it-IT","ru-RU"]},
    "DeviceMode":"Native",
    "Field1":"",
    "Field2":"",
    "Field3":"",
    "Field4":"",
    "Field5":""
}
```

Example to update one of the *property* value (ex: *DeviceName*) of the *GeneralSettings* singleton with a patch:

```
curl -u admin:admin --digest --request PATCH http://192.168.8.201/odata.qs/v1/GeneralSettings ../..
--header "Content-Type: application/json" --data "{'DeviceName':'Display-floor2'}"
```

Example to check the new values of all the properties for the *GeneralSettings* singleton:

```
curl -u admin:admin --digest --request GET http://192.168.8.201/odata.qs/v1/GeneralSettings/DeviceName/$value
```

Result example:

```
Display-floor2
```

Example to get the values of all the properties for the *SystemMaintenance* singleton:

```
curl -u admin:admin --digest --request GET http://192.168.8.201/odata.qs/v1/SystemMaintenance
```

Result example:

```
{
    "@odata.context":"/odata.qs/v1/$metadata#SystemMaintenance/$entity",
    "@odata.type":"#SystemMaintenance.SystemMaintenance",
    "Reboot":{"Status":"Stopped","Error":null},
    "ResetPreferences":{"Status":"Stopped","Error":null}
}
```

Example to update one of the *property* value (ex: *Reboot*) of the *SystemMaintenance* singleton with a patch:

```
curl -u admin:admin --digest --request PATCH http://192.168.8.201/odata.qs/v1/SystemMaintenance ../..
--header "Content-Type: application/json" --data "{'Reboot':{'Status':'Running'}}"
```

3.2 Appendix: Screensaver APK

It is possible to develop your own screensaver APK.

A package can implement a dream service.
When a dream service runs, the activity lifecycle is impacted. The activity of the current application is paused and stopped. It is restarted and resumed when the dream service ended.

> ⚠ If an application plays a media containing a video in java or in a webview, the screensaver is not reached because the device keeps screen on.

The default screensaver is `com.android.dreams.basic/com.android.dreams.basic.Colors` .

When implementing your custom `screensaver` App, it is recommended that the value for the `screensaver_components` user preference matches the package name of your custom `screensaver` APK (`<package name>/<package name>.Screensaver`), for example for *Screensaver for AV Stream reader* demo `screensaver` APK: `tech.qeedji.av_stream_reader_screensaver/tech.qeedji.av_stream_reader_screensaver.Screensaver` .

The settings preferences below handle the screensaver:

Name	Namespace	Type	R/W	Default value	Values
`screen_stay_on`	system	Integer	RW	`1`	`0` , `1`
`screen_off_timeout`	system	Integer	RW	`60000`	`10000` to `86400000`
`screensaver_enabled`	secure	Integer	RW	`1`	`0` , `1`

`screensaver_components`	secure	String	RW	`com.android.dreams.basic/com` ../.. `.android.dreams.basic.Colors`	`com.android.dreams.basic/com` ../.. `.android.dreams.basic.Colors`
					`tech.qeedji.av_stream_reader_screensaver/` ../.. `tech.qeedji.av_stream_reader_screensaver.Screensaver`

3.3 Appendix: WebUI Extension APK

The `WebUI Extension` APK is a demo APK supporting:

- a demo Web page to configure the administrator preferences of the APK. This interface is usually used by the DSI.
- a demo Web page to configure the applicative preferences of the APK. This interface is usually used by the user.

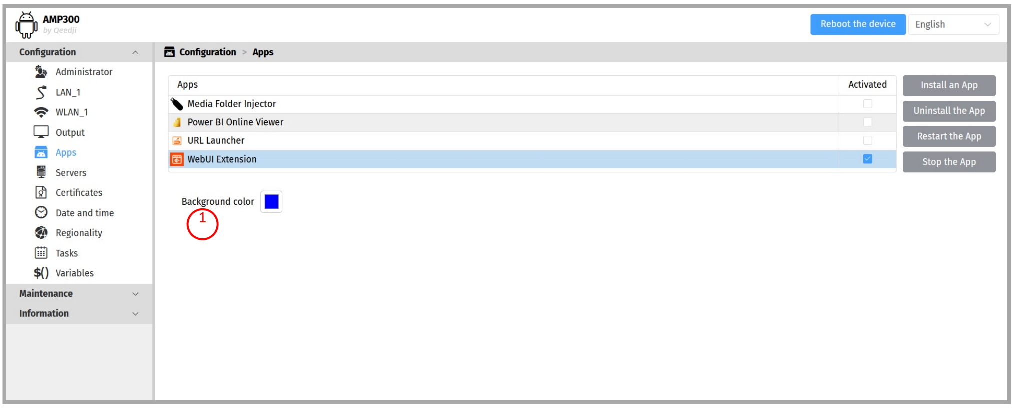
App manifest

The App manifest is here: ../src/main/AndroidManifest.xml

APK configuration WebUI

In the *WebUI Extension* demo APK, the *Background color* ① custom user preference has been added.

The entry point file *index.html* for your APK configuration Web page is here: ../src/main/assets/webuiex_admin/index.html



Apps	Activated
Media Folder Injector	☐
Power BI Online Viewer	☐
URL Launcher	☐
WebUI Extension	☑

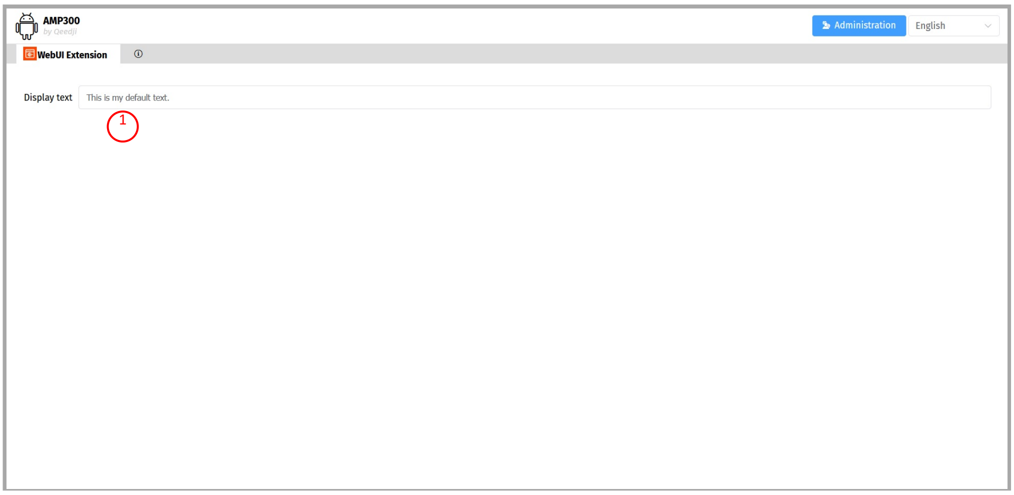
Background color 🟦

The *SharedPreferenceAPI* overloading for this configuration user preferences is here: ../src/main/java/tech/qeedji/webui_extension/MySharedAdminPreferenceAPI.java.

APK applicative WebUI

In the *WebUI Extension* demo APK, the *Display text* custom user preference ① has been added in a new *WebUI Extension* applicative tab.

The entry point file *index.html* for your APK applicative Web page is in this folder: ../src/main/assets/webuiex_appli/index.html.



Display text: This is my default text.

The *SharedPreferenceAPI* overloading for this applicative user preference is done here: ../src/main/java/tech/qeedji/webui_extension/MySharedAppliPreferenceAPI.java.

Runtime

The runtime of the *WebUI Extension* demo APK is implemented in this file: ../src/main/java/tech/qeedji/webui_extension/MainActivity.java

When the APK is launched, the text value available in the *Display text* input is displayed on the screen with the *Background color* input value as background color.

In this version, the APK must be restarted so that the new *Display text* input value and the new *Background color* input value are taken into account.

3.4 Appendix: Loss of interactivity when calling wakeUp

A loss of interactivity could be faced when using your App when the `PowerManager().wakeUp()` API is called to exit *Device Sleep* state, typically when exiting from a `Power Manager` task. This is due to a wrong management in Webview of the App visibility during a *Pause* to *Resume* activity state transition. To work around, override the `onResume()` and `onPause()` methods in your App like explained below:

```
@Override
protected void onResume() {
    super.onResume();
    webView.setVisibility(View.VISIBLE);
}

@Override
protected void onPause() {
    super.onPause();
    webView.setVisibility(View.GONE);
}
```

3.5 Appendix: HTTPS server for AQS device self-administration

The AQS device supports the *HTTPS tasks manager* feature allowing it to self-administrate by connecting perioically to a remote HTTPS server able to provide administration tasks to:

- update its firmware version,
- update its configuration,
- install an App.

AQS version

The *HTTPS tasks manager* feature is supported in the AQS 9.11.10 version (and above). To keep the *HTTPS tasks manager* feature working, you must not install a firmware version below 9.11.10.

After each task execution, the device is rebooting once

The device can execute only one task at a time. This is a table showing the installation duration after each task type execution:

Task	Task type	time gap before next task execution
App installation	*setup.app*	2 minutes
configuration update	*setup.configuration*	2 minutes
firmware version update	*setup.firmware*	6 minutes

HTTPS server

To support the AQS device slef-administration with a HTTPS server, you must have a HTTPS server available with or without *basic* authentication, which is able to provide a specific JSON content.

JSON content

Application installation task type

To install an *.apk* App, the HTTPS server must return a content with this JSON content whose:

- the task *id* is unic (in the example *uuid1*, you can use also UUID),
- the task *type* is *setup.app*,
- the *uri* is defining the relative location of the child directory and the file name of the *.apk* App to install.

For example

```
{
    "uuid1" : {
        "type": "setup.app",
        "uri": "./applications/system_button-qeedjisystem_aosp-setup-1.11.10.apk"
    }
}
```

Configuration update task type

To install an *.js* configuration script, the HTTPS server must return a content with this JSON format whose:

- the task *id* is unic (in the example *uuid2*, you can use also UUID),
- the task *type* is *setup.configuration*,
- the *uri* is defining the relative location of the child directory and the file name of the *.js* configuration script to install.

```
{
    "uuid2" : {
        "type": "setup.configuration",
        "uri": "./configurations/000000000000.js"
    }
}
```

Firmware installation task type

To install an *.frm* firmware, the HTTPS server must return a content with this JSON format whose:

- the task *id* is unic (in the example *uuid3*, you can use also UUID),
- the task *type* is *setup.configuration*,
- the *uri* is defining the relative location of the child directory and the file name of the *.js* configuration script to install.

```
{
    "uuid3" : {
        "type": "setup.firmware",
        "uri": "./firmwares/aosp-amp300-setup-9.11.10.fqs",
    },
}
```

Task scheduling

It is possible to program several tasks of different types. They can appear in the same JSON content accompagnied with the *execDateTime* parameter defining the date&time (UTC) for each task execution.

The device is not able to warranty the tasks execution when several tasks must be executed at the same time. It is required to execute the tasks, the one after the other. Consequently, in this case, it is required to schedule the tasks with a time gap between them, for example:

Task name	Task scheduling
uuid10	2024-09-01T19:00:00
uuid11	2024-09-01T19:02:00
uuid20	2024-09-01T19:04:00
uuid30	2024-09-09T19:06:00

> ☛ *A task is executed only once. To be executed again, the JSON content must be modified.*
>
> ☛ *A task whose scheduling is programmed in the past allows the task to execute execution only when the JSON content is modified (for example: uuid value modified).*

Example of JSON content:

```
{
    "uuid10" : {
        "type": "setup.app",
        "uri": "./applications/device_power_standby-qeedjisystem_aosp-setup-1.10.11.apk",
        "execDateTime": "2024-09-01T19:00:00.000Z"
    },
    "uuid11" : {
        "type": "setup.app",
        "uri": "./applications/system_button-qeedjisystem_aosp-setup-1.11.10.apk",
        "execDateTime": "2024-09-01T19:02:00.000Z"
    },
    "uuid20" : {
        "type": "setup.configuration",
        "uri": "./configurations/001ce6024e93.js",
        "execDateTime": "2024-09-01T19:04:00.000Z"
    },
    "uuid21" : {
        "type": "setup.configuration",
        "uri": "./configurations/001ce6024e94.js",
        "execDateTime": "2024-09-01T19:04:00.000Z"
    },
    "uuid30" : {
        "type": "setup.firmware",
        "uri": "./firmwares_tab10/aosp-tab10-setup-9.11.10.fqs",
        "execDateTime": "2024-09-01T19:06:00.000Z"
    }
}
```

HTTPS server implementation

JSON content in JSON file

the JSON content can be simply stored in a JSON file. for example:

- `https://<server_domain_URL>/<path>/<filename_with_json_format>`

In this case, to administrate your park of TAB10 devices, create this directory tree with the JSON file at the root of your HTTPS server Web directory:

- fill the empty JSON file (ex: `<filename_with_json_format>` = *tasks_TAB10.json*) with the suitable JSON content (the file naming is not important),
- add the the suitable *.js* configuration scripts with the naming pattern `<device_MAC_ID>.js` for your AQS devices in the *configurations/* directory.
- add the suitable *.apk* App in the *applications/* directory,
- add the suitable *.frm* firmware in the *firmwares/* directory.

Tree result:

- *tasks_TAB10.json*,
- *tasks_AMP300.json*,
- *applications/*
    - *devicepowerstandby-qeedjisystem_aosp-setup-1.10.11.apk*
    - *systembutton-qeedjisystemaosp-setup-1.11.10.apk*
- *firmware_tab10/*
    - *aosp-tab10-setup-9.11.10.fqs*
- *firmware_amp300/*
    - *aosp-amp300-setup-9.11.10.fqs*
- *configurations/*
    - *001ce6024e94.js*
    - *001ce6024e93.js*

You are free to organize the filetree like you wish.

JSON content with a service

Your HTTPS server can implement a service able to provide the JSON content.

HTTPS request

To allow your HTTPS server to be able to provide suitable JSON content depending on the device PSN and on the device MAC address, the suffix `&PSN=<device_short_PSN>&MAC=<device_MAC_ID>` is added systematically at the end of the HTTPS server URL when a request is done to the HTTPS server. For example, if the URL is `https://<server_domain_URL>/<path>?data=val` or `https://<server_domain_URL>/<path>/task.json` in the device configuration Web user interface, the URL used in the HTTPS request becomes `https://<server_domain_URL>/<path>?data=val&PSN=<device_MAC_ID>&MAC=<device_short_PSN>` or `https://<server_domain_URL>/<path>/task.json?&PSN=<device_MAC_ID>&MAC=<device_short_PSN>` .

Consequently, when your HTTPS server receive this request from a AQS device, your HTTPS server must return the appropriate JSON content. Given that the suffix `&PSN=<device_MAC_ID>&MAC=<device_short_PSN>` is systematically included in the request, you can choose or not to implemend an HTTPS server which is returning different JSON content depending on the `<device_MAC_ID>` and `<device_short_PSN>` parameter values.

Request

HTTPS request parameter	Value
URL	`https://<myHttpsServerDomain>/<path>/<json_file>?&PSN=<device_short_PSN>&MAC=<device_MAC_ID>`
Auth	*Basic Authentication* or no authentication
Content-Type	*application/json*
Method	*GET*

Examples of HTTPS request URL:

- `https://demo.server/administration/task.json?&PSN=01540-01111&MAC=00-1c-e6-02-5e-b0` ,
- `https://demo.server/administration?type=tasks&PSN=01620-01111&MAC=00-1c-e6-02-5e-b0` ,

with:

- `https://demo.server/administration/task.json` : HTTPS server URL example #1 in the device configuration Web user interface,
- `https://demo.server/administration?type=tasks` : HTTPS server URL example #2 in the device configuration Web user interface,
- *00-1c-e6-02-5e-b0*: example of MAC ID of your AQS device,
- *01540-01111*: example of PSN for AMP300 device,
- *01620-01111*: example of PSN for TAB10b device.

Response to the HTTPS request

```
{
    "uuid10" : {
        "type": "setup.app",
        "uri": "./applications/device_power_standby-qeedjisystem_aosp-setup-1.10.11.apk",
        "execDateTime": "2024-09-01T19:00:00.000Z"
    },
    "uuid11" : {
        "type": "setup.app",
        "uri": "./applications/system_button-qeedjisystem_aosp-setup-1.11.10.apk",
        "execDateTime": "2024-09-01T19:02:00.000Z"
    },
    "uuid20" : {
        "type": "setup.configuration",
        "uri": "./configurations/001ce6024e93.js",
        "execDateTime": "2024-09-01T19:04:00.000Z"
    },
    "uuid21" : {
        "type": "setup.configuration",
        "uri": "./configurations/001ce6024e94.js",
        "execDateTime": "2024-09-01T19:04:00.000Z"
    },
    "uuid30" : {
        "type": "setup.firmware",
        "uri": "./firmwares_tab10/aosp-tab10-setup-9.11.10.fqs",
        "execDateTime": "2024-09-09T19:06:00.000Z"
    }
}
```

Device configuration Web user interface

To enter the URL and the credential of your HTTPS server for AQS device self-administration, connect to the device configuration Web user interface. In the *Servers* pane of the *Configuration* menu, activate the *Retrieve tasks to be executed from a http(s) server* then:

- enter the *URL* of your HTTPS server,
  - for example:
    - `https://demo.server/administration/task.json`
    - `https://demo.server/administration?type=tasks`
- enter the *credential* type:
  - *None*,
  - *Identifier/password*:
    - enter the *Identifier* value,
    - enter the *Password* value,
- enter the heartbeat value (default: *10* min), periodic connection duration to your HTTPS server.