

Qeedji

Developer manual 003B

AOSP-9.10.10



Legal notices

AOSP-9.10.10 (003B_en)

© 2020 Qeedji

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
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
------------------------------	-----

1.1 Introduction

This documentation is intended for ISVs (Independent Software Vendors), wishing to develop AOSP APKs on Qeedji TAB10s devices.

⚠ Android APK development skills are required to go ahead.

⚠ It is recommended to read first the [TAB10s user manual](#).

Demo Package Content

Items	Description	Quantity
TAB10s	Qeedji tablet embedding AQS 9	1
Power supply	USB Type-C	1
USB Type-C cable	Cable - Assembly, Type-C Male to Type-A Male	1
USB hub	USB Type-A (2.0), USB Type-C	1

1.2 APK Development

Prerequisite

The software developer already knows how to develop an `Android APK` and how to generate/debug it with `Android Studio`.

AOSP Standard API

The standard API of `AQS 9.10.10` is based on the `AOSP SDK 28`.

The `AQS 9.10.10` 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.

Qeedji System Java API

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

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

```
public class SurroundLight {
    public static final int OFF = 0;
    public static final int RED = 1;
    public static final int GREEN = 2;
    public static final int ORANGE = 3;

    public SurroundLight();
    public void setColor(int color);
    public int getColor();
}

public class Rfid125KHz {
    public Rfid125KHz(Context c);
    public void setEnabled(boolean value);
}

public class DipSwitch {
    public DipSwitch(Context c);
    public boolean camera();
    public boolean microphone();
}

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

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

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);
}
```

Qeedji System Javascript API

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

The `tech-qeedji-host-webview.aar` Android library exposes an 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();

boolean Host.DipSwitch().getCamera();
boolean Host.DipSwitch().getMicrophone();

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

void Host.NFC().start();
void Host.NFC().stop();
// parameter type is unused
callback : void onHostNFCReading(String type, String id);

void Host.PowerManager().goToSleep();
void Host.PowerManager().wakeUp();
void Host.PowerManager().reboot();

void Host.ProximitySensor().start();
void Host.ProximitySensor().stop();
callback : void onHostProximitySensorReading(float distance);

void Host.Rfid125KHz().start();
void Host.Rfid125KHz().stop();

int Host.SurroundLight().OFF();
int Host.SurroundLight().RED();
int Host.SurroundLight().GREEN();
int Host.SurroundLight().ORANGE();
void Host.SurroundLight().setColor(int color);
int Host.SurroundLight().getColor();

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

```

APK shared preferences handling with configuration script

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"]);
```

Examples

The [Qeedji github for TAB10s](#) is hosting APK examples using the [AOSP SDK](#) for TAB10s:

- [proximity_sensor APK](#): displays a message `Hello Qeedji` when a person is detected, or displays a message `Nobody detected` when no one is detected.



This APK uses the [SensorManager](#) API with a default sensor of type [Sensor.TYPE_PROXIMITY](#).

- [rfid_tag_reader APK](#): detects `NFC` tags or `RFID 125KHz` tags and print their values (type and ID).



This APK uses the [NfcAdapter](#) API for NFC tags and the [KeyEvent.Callback](#) for RFID 125KHz tags.
The [NFC](#) permission and the [android.hardware.nfc](#) feature API are required.
This APK requires user system execution rights.

- [surround_light APK](#): allows to set the surround light color and state with `steady/green`, `steady/orange`, `steady/red`, `off`.



This APK uses the [SurroundLight](#) class described in the specific API.
This APK requires user system execution rights.

- [autorestart APK](#): is launched automatically after a device reboot. It is relaunched also automatically after it crashes.



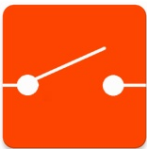
This APK uses the [BroadcastReceiver](#) and [Thread.UncaughtExceptionHandler](#) APIs.
The [RECEIVE_BOOT_COMPLETED](#) permission is required.

- [Device Power Standby](#): allows to go into (or exit from) Android sleep mode (display off, touch screen off).



This APK uses the [PowerManager](#) API.
The `DEVICE_POWER` and `WAKE_LOCK` permissions are required.
This APK requires user system execution rights.

- [System Button](#): print a notification message when a short press on the system button, lower than two seconds, is detected.



This APK uses the [BroadcastReceiver](#) API.
This APK requires user system execution rights.

- [URL Launcher](#): load an 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 has user system execution rights.
A specific `000000000000.js` configuration script allows to configure the URL launcher APK (set URL, set login credentials, ...).
Several websites examples are available on [Qeedji github for TAB10s](#).

⚠ 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](#) or to the chapter § [Set App as System App](#).

APK debug

The `AQS` Operating system for the TAB10s device is compatible with [Android Studio](#) and [Android Debug Bridge \(ADB\)](#)² software development suite.

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

You can debug with ADB using:

- `USB` ,
- `WLAN` ,
- `LAN` ³.

³ *Debugging with the LAN interface of the your computer requires to have an `Ethernet to USB` bridge.*

USB debug

Connect a cable between the USB-C connector of the TAB10s device and the USB 2.0 connector of your computer. Then wait for the TAB10s device is booting up.
Unlike an Android Mobile tablet, the TAB10s device has no battery and is completely powered by the USB-C connector. Before supply the TAB10s 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 else the TAB10s device may not stop rebooting.

WLAN debug

Connect the power cable of your USB-C wall plug to the USB-C connector of the TAB10s device. Then wait for the TAB10s device is booting up. Then go in the `Settings` application and configure the WLAN.

LAN debug

Prerequisite: have a suitable `Ethernet to USB` (USB-C or POGO type connector) bridge which is connected to the LAN network.
Connect the `Ethernet to USB` bridge to on the TAB10s device (USB-C or POGO type connector).

Debug mode setting

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 TAB10s 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](#) or 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 TAB10s.
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** developed 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:

- **Surround Light**: feature allowing to command the **surround light**,
- **Rfid 125KHz**: feature allowing to get a badge ID when a badging is done with a **RFID 125KHz** badge,
- **Dip Switch**: feature allowing to get the current configuration for the **micro** and **camera** **Dip switch**,
- **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),
- **Pptx**: feature allowing an app using a **Android System WebView** to play **MS-PowerPoint** medias (**.ppsx** and **.pptx**),
- **Pdfts**: feature allowing an app using a **Android System WebView** to play **PDF** medias (**.pdf**),
- **setAppAsSystemApp**: feature granting **system user** execution rights for a list of app.

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.

- **NFC**: feature allowing to be notify and to get the badge ID when a badging is done with a **NFC** badge,
- **Proximity Sensor**: feature allowing to be notified of the **presence detection** distance,
- **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 declare 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.

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 following example values have been used.

Label type	Label value examples
C	US
ST	California
L	San-Francisco
O	Contoso
OU	Contoso_ISD
CN	CISO
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=CISO/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 2 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 or commercial conditions are not fully filled, Qeedji keeps the rights to revoke 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 -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 its APK requiring system user execution rights and the same certificate for all its 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

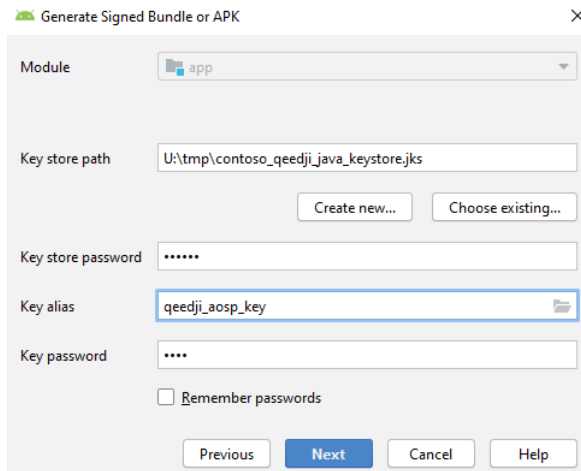
Modify the APK manifest by adding this string: `android:sharedUserId="android.uid.system"`

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:supportRtl="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).



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

- Key store password = 567890
- Key password = 1234

The signing procedure is 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/Queedji/aosp-TAB10s-sdk/blob/master/examples/surround_light/app/build.gradle.

```
android {
    compileSdkVersion 29
    buildToolsVersion "29.0.3"
    defaultConfig {
        applicationId "tech.queedji.tablet.surround_light"
        minSdkVersion 28
        targetSdkVersion 29
        versionCode 2
        versionName "1.10.12"
        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 the `NFC tools` application, the URL is <https://play.google.com/store/apps/details?id=com.wakdev.wdnfc>. 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 required to request 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 [Queedji 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 inconsistent lines.

```
<?xml version="1.0"?>
<AppList>
  <AppId>tech.queedji.app1</AppId>
  <AppId>tech.queedji.app2</AppId>
  <AppId>tech.queedji.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_queedji_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_queedji_aosp.pk12`:

```
.\AQS-setAppAsSystemApp.ps1 -pk12File .\contoso_certificate_and_key_for_queedji_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 [Queedji Website](#). Uncomment the 2 lines below and replace the value `<?xml version="1.0"?><AppList>...` of the `xmlSignedFileData` variable by the entire file content of the generated `app-list-signed.xml` file.

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

☞ The configuration script must be V1.10.17 (or above).

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 a USB storage device and inject it on a USB hub connected to 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. Queedji advises you to install the configuration script first, and after, install the `APK` which, will have consequently the `system user` execution rights granted.

☞ To return to a configuration which do 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 version="1.0"?>
<AppList>
</AppList>
```

1.4 Qeedji System service

The TAB10s 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 TAB10s device:
 - by uploading an `APK` with the device Web user interface,
 - by pushing one `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 TAB10s 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 TAB10s 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 [TAB10s 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 [TAB10s 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 of the device is `[fe80::21c:e6ff:fe02:5694]` and the IPV4 address of the device is `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-tab10-setup-9.10.10.fqs" "http://192.168.8.201/.software/"
```

```
curl -u admin:admin --digest -T "aosp-tab10-setup-9.10.10.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.ceedji.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.

Two values are possible for the `persist.sys.device_mode` system property:

- `native` (default value): thanks to AOSP menu, the user can, whenever he wants, stops the APK, returns to the AOSP home screen, launches 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 or if it does not exist, 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 [TAB10s 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 release will allow to load a custom wallpaper.

1.5 Qeedji preferences

AOSP system properties added by Qeedji

Name	Type	R/W	Default value	Values	Description
<code>persist.sys.delivery-software-version</code>	String	RO	9.10.10	<X>.<V>.<Z> <software-extraversion>	Save the version of the last AQS firmware upgrade.
<code>persist.sys.device_mode</code>	String	RW	native	native , kiosk	Handle the AOSP device mode.
<code>persist.sys.rfid125khz.enable</code>	Boolean	RW	true	true , false	Handle the RFID 125KHz.
<code>persist.sys.rfid125khz.keyboard_wedge.key_interval</code>	String	RW	0	0 to 1000	Define the key interval of RFID 125KHz keyboard edge.
<code>persist.sys.rfid125khz.keyboard_wedge.key_press_duration</code>	String	RW	0	0 to 1000	Define the key press duration of RFID 125KHz keyboard edge.
<code>persist.sys.proximity_sensor.type</code>	String	RW	ir	ir	Define the proximity sensor.
<code>persist.sys.proximity_sensor.max_distance</code>	String	RW	200	50 100 150 200 250 300 350 400 450 500 550 600	Define the max distance in cm of the proximity sensor.
<code>persist.sys.hostname</code>	String	RW	TAB10s	For example TAB10s-00001	Define the hostname. If the hostname is empty, the network hostname corresponds to the canonized device name. Else, the network hostname corresponds to the canonized hostname.
<code>persist.sys.webserver.http.port</code>	Integer	RW	80	1 to 65535	Define the port of the http server.
<code>persist.sys.webserver.webdav.credential</code>	String	RW	default		Define the credential ID for the Publishing software connection profile.
<code>persist.sys.webserver.webuiappli.credential</code>	String	RW	default		Define the credential ID for the Application user connection profile.
<code>persist.sys.webserver.webuiadmin.credential</code>	String	RW	default		Define the credential ID for the Administration user profile.
<code>persist.sys.webserver.webservice.credential</code>	String	RW	default		Define the credential ID for the Web Service profile.
<code>persist.sys.webserver.credential.default.type</code>	String	RW	native	native	Define the default credential type.
<code>persist.sys.webserver.credential.default.username</code>	String	RW	admin	admin	Define the default credential username.
<code>persist.sys.webserver.credential.default.password</code>	String	RW	admin	admin	Define the default credential password.
<code>persist.sys.webserver.credential.<credential_ID>.type</code>	String	RW		native	Define the credential type.
<code>persist.sys.webserver.credential.<credential_ID>.username</code>	String	RW			Define the credential username for the native type.
<code>persist.sys.webserver.credential.<credential_ID>.password</code>	String	RW			Define the credential password for the native type.
<code>persist.sys.device_info.field1</code>	String	RW			Custom device field1 variable.
<code>persist.sys.device_info.field2</code>	String	RW			Custom device field2 variable.
<code>persist.sys.device_info.field3</code>	String	RW			Custom device field3 variable.
<code>persist.sys.device_info.field4</code>	String	RW			Custom device field4 variable.
<code>persist.sys.device_info.field5</code>	String	RW			Custom device field5 variable.

Settings preferences added by Qeedji

Name	Namespace	Type	R/W	Default value	Values	Description
developer_options_allowed	secure	Integer	RW	0	0 , 1	When the preference value is 1 , the debug mode is allowed.
adb_tcp_enabled	global	Integer	RW	0	0 , 1	When the preference value is 1 , adb over network is 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 the PTP (Picture Transfer Protocol).

Preferences modified by Qeedji System service

Name	Namespace	Type	R/W	Default value	Values	Description
settings.secure.android_id	secure	String	R	not communicated	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

Name	Type	R/W	Default value	Values	Description
externalstorage.copy.apk.enabled	Boolean	RW	true	true , false	When the preference value is true , the APK installation from the root of the USB storage is authorized.

Shared preferences for the URL Launcher APK

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

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

tech.qeedji.url_launcher.prefs.xml	Type	R/W	Default value	Values	Description
start_after_boot_completed	Boolean	RW	true	true , false	When the preference value is true , The URL launcher APK is automatically started after the AOSP has started.
autorefresh_url_enabled	Boolean	RW	false	true , false	When the preference value is true , The URL launcher APK relaunches periodically the URL.
autorefresh_url_interval	Long	RW	60	1 to 86400	Defines the reload period in seconds for the URL launcher APK .
url	String	RW		for example: https://www.demo.contoso.com/	Defines the URL to launch.
credential	String	RW		for example: If <credential_label> worths native , the value is native	Defines 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_launcher.prefs.xml file above.

tech.qeedji.url_launcher.credential.<credential_label>.prefs	Type	R/W	Default value	Values	Description
type	String	RW	native	native ¹	Define the credential type.
username	String	RW			Define the URL credential username.
password	String	WO			Define the URL credential password.

¹ In this version, only the native value is possible.

1.6 FAQ

Do you have special adapters ?

Several PoE adapters can be ordered to Qeedji.

Commercial reference	Device model	Information
EXC.ETH.POGO	NAPOE109kt	Ethernet PoE Krone to TAB10s built-in adapter
EXC.ETH.USBC	NAPOE109ku	Ethernet PoE Krone to USB-C built-in adapter

Is it possible to change the brightness of the surround light?

No, the surround light can only be:

- Green,
- Red,
- Orange,
- Off

How to deploy the APK in production mode without an USB hub?

To deploy your `APK` in production for the first time, there is several ways:

- either drop your `APK` in the `/.apps/` directory of the `WebDAV` server,
- or, if the TAB10s device is installed on a `EXC.ETH.POGO` adapter:
 - put your `.apk` files at the root directory of the USB-C storage device,
 - plug the USB-C storage device on the free USB-C connector of the TAB10s device.
- or, if the debug mode is enabled:
 - use the Android tool named `adb` (`adb install -g <apk_file>`).

Once your `APK` is installed:

- you can use a method described above,
- your `APK` can install `.apk` files with the API [PackageInstaller](#).

How to deploy the .apk files in production mode with an USB hub?

- plug on the TAB10s device an USB hub supporting an USB-C connector for power delivery,
- put your `.apk` files at the root directory of the USB storage device,
- plug the USB storage device on the USB hub.

How to launch an app in kiosk device mode?

Explain in the paragraph [AOSP device mode](#). Qeedji implements an `APK` example named `url_launcher`. Look at the two files below:

- [AndroidManifest.xml](#),
- [StartActivityAtBootReceiver.java](#).


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

No, but you can develop your own `APK`.

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  Website for FAQ, application notes, and software downloads:

<https://www.qeedji.tech/>

Qeedji FRANCE
INNÉS 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:

- *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 9.10.10 beta11 (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 of the device is [fe80::21c:e6ff:fe02:5694] and the *IPV4* address of the device is 192.168.8.201,
- the credentials of the connection profile to access to the Web Services is `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": "01356-00008",
  "Platform": "TAB10s",
  "Version": "9.10.10",
  "Mac": "00:1C:E6:02:5A:7B",
  "Hostname": "TAB10s-FRD"
}
```

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 of the device is [fe80::21c:e6ff:fe02:5694] and the IPV4 address of the device is 192.168.8.201,
- the credentials of the connection profile to access to the Web Services is `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": "TAB10s",
  "ForcedHostnameEnabled": false,
  "DeviceName": "myTAB10-1",
  "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':'myTAB10-2'}"
```

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:

```
myTAB10-2
```

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'}}"
```