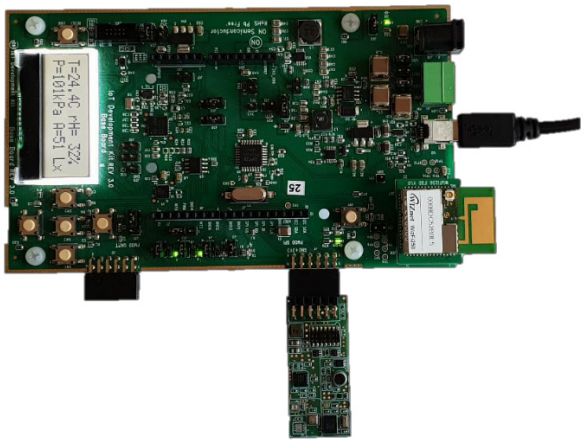
AND9885/D

Connecting the IDK to the Google Cloud IoT Core

ON Semiconductor®

www.onsemi.com

APPLICATION NOTE



INTRODUCTION

This application note provides details on connecting the IoT Development Kit (IDK) to the Google Cloud (GCP). An environmental sensing (temperature, humidity, pressure, ambient light) use case provided as an example within the IDK Software will be used to demonstrate connectivity to GCP.

Sensor data are transmitted over secure connection to AWS IoT Core service. Connection is secured by TLS encryption and the measured sensor data are periodically sent in JSON format over MQTT.

REFERENCES

[Ref 1] IoT IDK User Guide: https://www.onsemi.com/pub/Collateral/AND9666−D.PDF

[Ref 2] Google Cloud IoT Core documentation: https://cloud.google.com/iot/docs/

[Ref 3] Getting started: https://cloud.google.com/iot/docs/how−tos/getting−started

[Ref 4] Creating registries and devices: https://cloud.google.com/iot/docs/how−tos/devices

[Ref 5] PUB/SUB service documentation: https://cloud.google.com/pubsub

PREREQUISITES

The below items are needed in order to successfully configure and run this use case.

Hardware

- IDK Baseboard (BB−GEVK) with WizFi250 Wi−Fi Module
- Multi−Sensor Shield (MULTI−SENSE−GEVB)

The Multi−Sensor shield should be plugged into one of the two available PMOD connectors of the IDK Baseboard. The IDK Baseboard should then be connected to PC to provide power and for downloading the example use case.

Software

- IoT IDK Software (Version 4.3 & above): BB−GEVK Software
  Refer to IoT IDK User Guide (Ref 1) for installation details
- openssl
  Optional utility for generation of public/private key pairs for IoT devices on local machine

Google Cloud account

- Google Account with Access to Google Cloud Console: https://cloud.google.com/
  (Free trial account is sufficient for this example)
- Cloud IoT Core Service Enabled
  This service is used to connect and manage IoT devices. Please refer to the reference documents listed above.

GCP IOT CORE CONFIGURATION

This section covers the creation of Device Registry and IoT devices using web based Cloud Console admin UI.

Create Device Registry

Device Registry is a container of devices with shared properties. At least one registry must be created to connect devices to GCP. Follow the instructions in the Creating registries and devices documentation page to create a Device Registry. The registry created for this example uses the parameters listed below and the filled registry creation page is shown in Figure 1.

- Registry ID: *idk−examples−registry*
- Region: *europe−west1*
- Protocol: *MQTT*
- Default telemetry topic: *projects/{project−id}/topics/events*
- Device state topic: *None*
- CA certificate: *None*
- Stack Driver Logging: *Info*

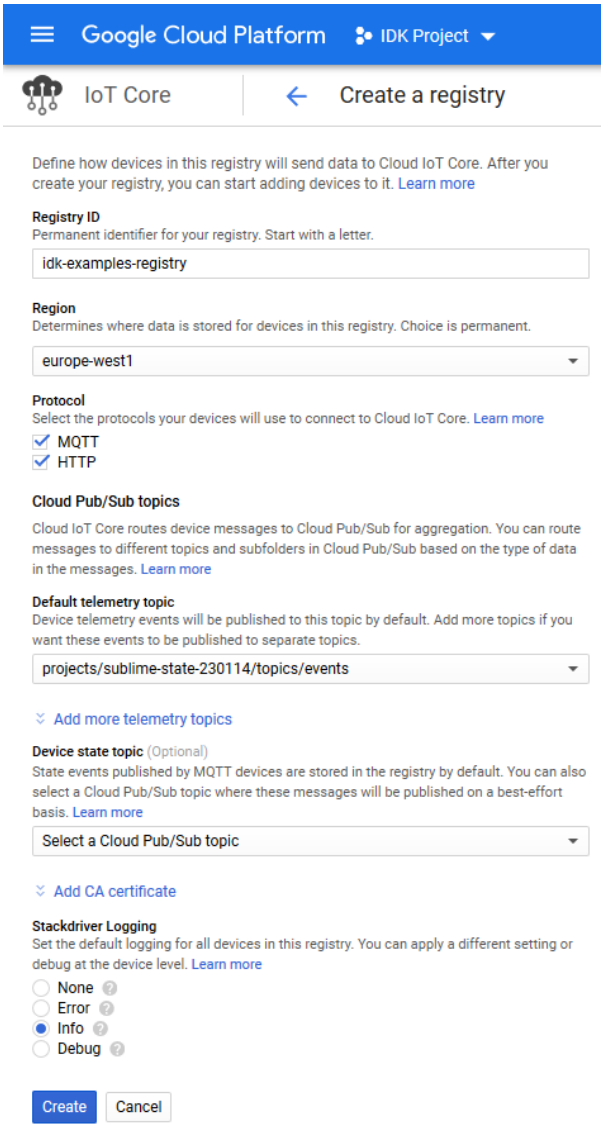


Figure 1.

Generate Public/Private Key Pair for Device

Before creating an IoT device under a device registry it is required to generate a public/private key pair for the device that will be used to authenticate the device when it connects to the cloud. The public key will be uploaded to Cloud Console during device creation. Private Key will be stored in the device itself as part of firmware.

The IDK supports the use of RSA−256 private key with 2048−bit key size.

The key pair can be generated using the *openssl* utility program as described in Creating public/private key pairs documentation of IoT Core service.

The commands listed in Figure 2 can be used to create private key (rsa_private.pem) and then extract the public key (rsa_public.pem).

```
openssl genrsa-out rsa_private.pem 2048
openssl rsa-in rsa_private.pem-pubout -out rsa_public.pem
```

Figure 2. Openssl Commands used to Generate Public/Private Key Pair

If openssl is not installed on your machine as a standalone program it can still be distributed as part of other software packages like Git Tools or Cygwin. If neither of that is applicable to generate keys locally it is possible to use Cloud Shell from the Cloud Console UI to generate the keys directly in Web browser as described in next section.

Using Google Cloud Shell to Generate Public/Private Key Pair

The Google Cloud UI provides Cloud Shell console interface that can be used to access all GCP resources directly from Web Browser. The Cloud Shell environment has openssl utility preinstalled and can be used to generate and download public/private key pairs:

1. Activate Cloud Shell from the toolbar of Cloud UI web page as shown in Figure 3 step 1
2. Enter the openssl commands from Figure 2 into the console to generate public/private key pair
3. Use Download file button to download generated keys from shell to your local machine. When asked for file name enter *rsa_private.pem* and then repeat for *rsa_public.pem* file
4. After downloading the files clean up the shell environment by deleting the generated keys using rm command followed by the names of files you want to remove

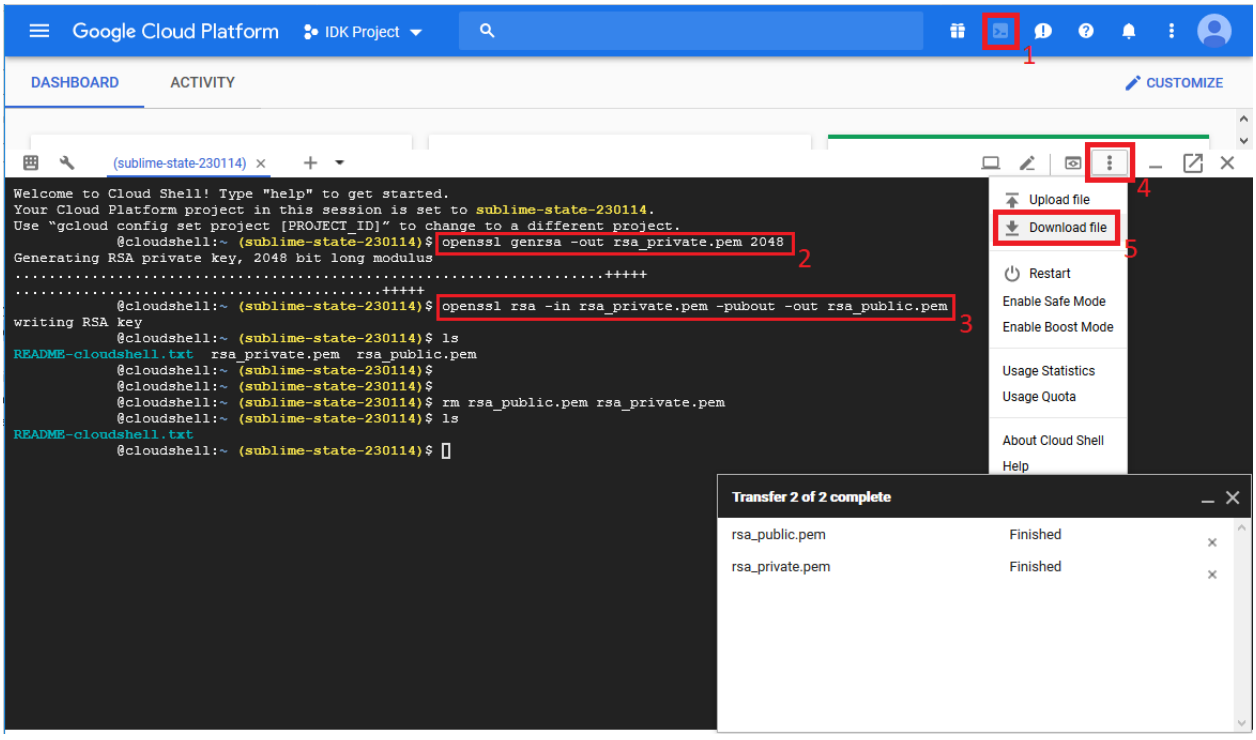


Figure 3. Steps for Generation of Public/Private Key Pair using Google Cloud Shell

Create Device

To add new device to your device registry follow the instructions in Creating or editing a device section of IoT Core service documentation. This example will use the following device parameters:

- Device ID: *idk−device*
- Device communication: *Allow*
- Authentication Input Method: *Upload*
- Public Key Format: *RS256* – Depends on type of generated public/private key pair. IDK supports only RS256
- Public key value: *rsa_public.pem* – Use Browse button to select pem file containing generated public key
- Public key expiration date: *None*
- Device metadata: *None*
- Stackdriver Logging: *Use registry default settings*

Figure 4. Example of Device Creation Page

CONFIGURING IDK EXAMPLE PROJECT

With the device created in cloud it is now time to prepare the IDK to connect to cloud using these credentials.

Importing Project

The example code for this application is bundled with the IoT IDK software package and can be imported using the Examples toolbar menu. The project is listed in the menu as *Complex → Secure Cloud Connectivity (TLS) by Ubiquios → Multi Sensor Node → Google Cloud* as shown in Figure 5.

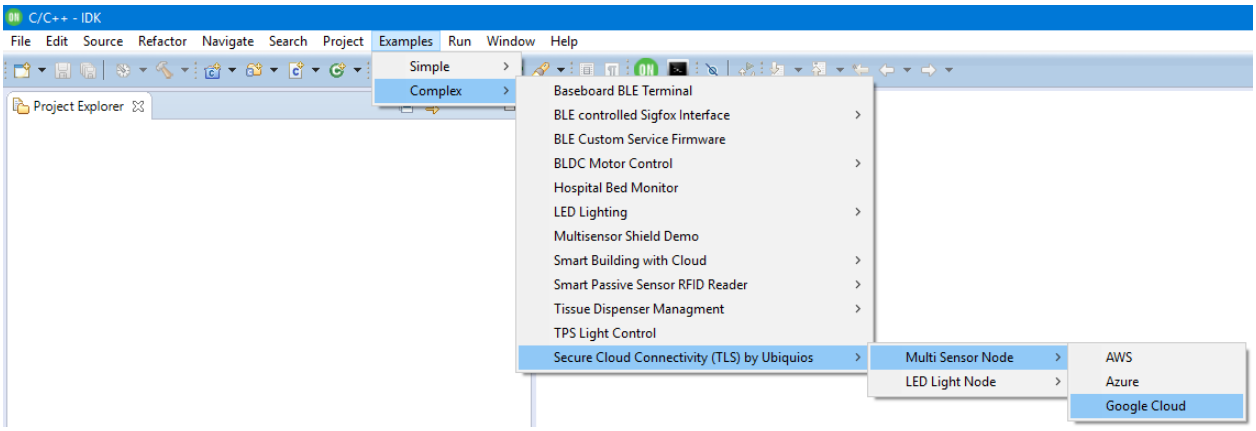


Figure 5. Examples Import Menu in IoT IDK

Project structure

After importing the project will appear in the project explorer view.

The internal structure of the project is as follows:

- src – Folder containing all source code files
  - configuration.h – Contains macro definitions for Wi−Fi credentials and cloud related variables that need to be modified for each user / environment
  - gcp_device_keystore.def – Contains preprocessed device certificates that are used to authenticate the device when it connects to cloud. This file needs to be generated and replaced for each device (steps in next section)
  - gcp_iot.c – Main application file that handler Wi−Fi connectivity, MQTT session to GCP and timers for periodic publishing of sensor data to cloud
  - jwt.* – Function for generation of signed JWT tokens used for authentication
  - sensor_node.* – Abstraction layer for the IDK sensor libraries used in this example
  - platform.* – Part of Ubiquios TLS stack. Should not be modified
  - pmalloc_pools.* – Part of Ubiquios TLS stack. Should not be modified
  - hal – Folder containing abstraction layers for the Ubiquios TLS stack. Should not be modified

Entering Credentials in Configuration.h File

The example program contains file **configuration.h** that is located inside of the **src** folder. This file is divided into three sections used to configure Wi−Fi, GCP connectivity and application related variables which need to be adjusted for proper operation of the application

1. Wi−Fi configuration consists of entering valid Wi−Fi access point credentials consisting of SSID and passphrase. To do this modify the **AP_SSID** and **AP_PASSPHRASE** macros

```
#define AP_SSID ”my_ap_ssid”

#define AP_PASSPHRASE ”my_ap_passphrase”
```

2. To configure GCP connectivity settings modify *GCP_PROJECT_ID, GCP_REGISTRY_REGION, GCP_REGISTRY_ID, GCP_DEVICE_ID* and *GCP_TELEMETRY_TOPIC* with values used to create Device Registry and device in the GCP IoT Core configuration section

```
#define GCP_PROJECT_ID                ” sublime-state-230114”

#define GCP_REGISTRY_REGION           ” europe-west1”

#define GCP_REG ISTRY_ID              ” idk-examples-registry”

#define GCP_DEVICE_ID                 ” idk-device”

#define GCP_TELEMETRY_TOPIC           ”/devices/” GCP_DEVICE_ID ”/events”
```

3. Optionally modify application specific configuration macros to change sensor data publish interval or to use simulated data if MULTI−SENSE−GEVK shield is not used

```
/** Time to wait after a message is sent before sending the next message. */
#define SEND_INTERVAL                 (UB_SECONDS(12))

/** Time to wait before retrying when  send fails. */
#define SEND_RETRY_INTERVAL           (UB_SECONDS(5))

/** Change to true if simulated data are to be used.
 *
 * When set to true it is not required to connect the MULTI -SENSE-GEVK board to IDK
Baseboard.
 */
#define USE_SIMULATED_SENSOR_DATA   (false)
```

Generating Device Keystore from Private Key

The last configuration step is to generate a keystore that contains device's private key used to authenticate the device when connecting to GCP. To overcome the low RAM footprint of the processor located on IDK Baseboard the downloaded certificates cannot be used directly but have to be preprocessed and included as a source code file into the project.

The following steps need to be done to get preprocessed keystore header file that can be used in the project:

1. Open IDK installation directory and navigate to *C:\\IDK_INSTALL_DIR\external\ubiquios\bin* directory.
2. Copy the downloaded private key file *rsa_private.pem* into this directory
3. Open Command line terminal (*cmd.exe*) and change working directory to this folder.
   This can be done from File Explorer by typing *cmd.exe* into the navigation bar and hitting enter as shown in Figure 6

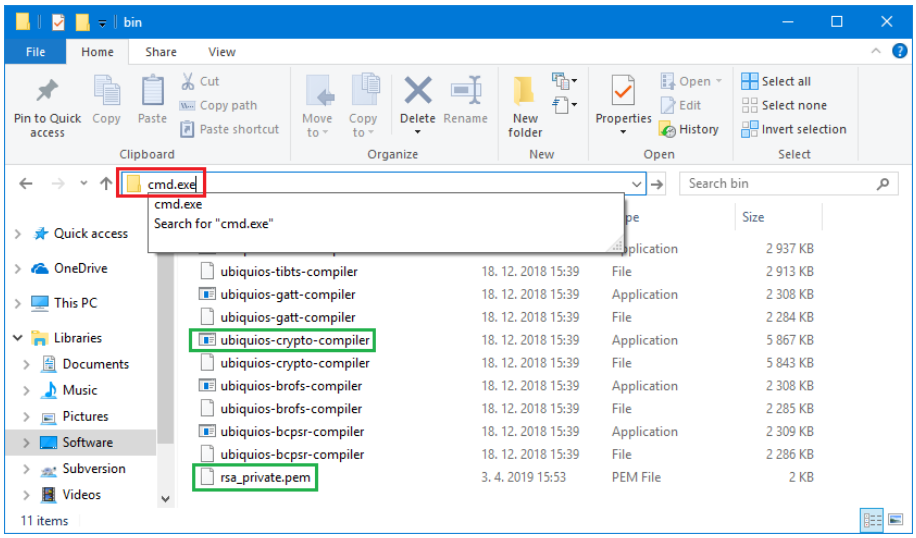


Figure 6. Entering cmd.exe into the Navigation Bar in File Explorer to Open Command Line with Working Directory Set to this Folder

4. In command line enter the following command to generate keystore for this application:

```
ubiquios-crypto-compiler.exe keystore -n gcp_device_keystore -o gcp_device_keystore.def
rsa_private.pem
```

   This command will extract the private key encoded in *rsa_private.pem* and turn it into C header file named *gcp_device_keystore.def* that contains keystore structure named *gcp_device_keystore*. This structure is then used by application when operations involving private key are executed.
5. Copy the generated *gcp_device_keystore.def* file into *src* subdirectory of the project, replacing the original placeholder file
6. Compile project to verify that no errors are generated from the new keystore header

VERIFYING DEMO FUNCTIONALITY

Once all configuration macros in configuration.h header file were modified to valid values and keystore with private key generated the application can be compiled and flashed into the IDK Baseboard.

The application will print status messages to serial terminal showing sensor data and connectivity status. Serial terminal output can be accessed either from the Flash Utility if it is left open after flash process is finished or from any other serial terminal application. The serial terminal uses 115200 baud rate, 8 data bits, no parity, 1 stop bit and no flow control.

Working application will show the following status messages:

- Connection to Wi−Fi network was successful and assigned IP address of the kit is shown
- NTP request for current network time succeeds and current time in UTC is shown
- Signed JWT authentication token is generated and MQTT connection is established
- Sensor data are measured and successfully published to cloud periodically according to configured *SEND_INTERVAL*
- Payload of all packets received on one of the */devices/{device_name}/commands/* sub topic is printed to terminal

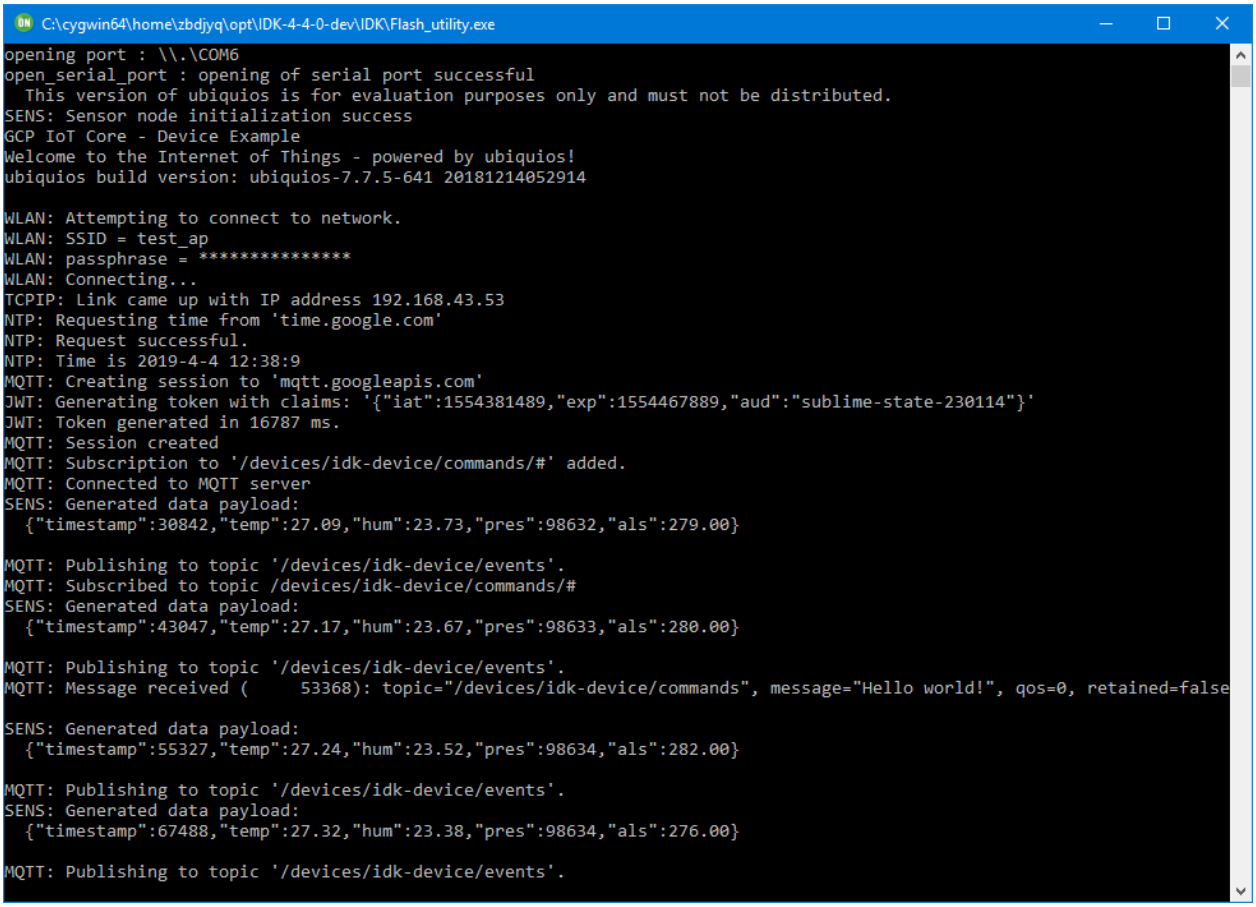


Figure 7. Terminal Output from Baseboard showing Periodically Published Data and Received Messages

Sending Commands to Device from Cloud UI

The device is subscribed to the */devices/{device−id}/commands/#* wildcard topic which allows it to receive messages from all its sub topics, e.g. */devices/idk−device/commands/my−command*

The example firmware set up to print payload of all such messages to serial terminal as can be seen in Figure 7 where message with the payload “Hello world!” was received. Additional functionality can be added in the receive handler in the application.

Commands can be send from device page of Cloud UI as described in the Sending a command section of Cloud IoT Core documentation and illustrated in Figure 8.

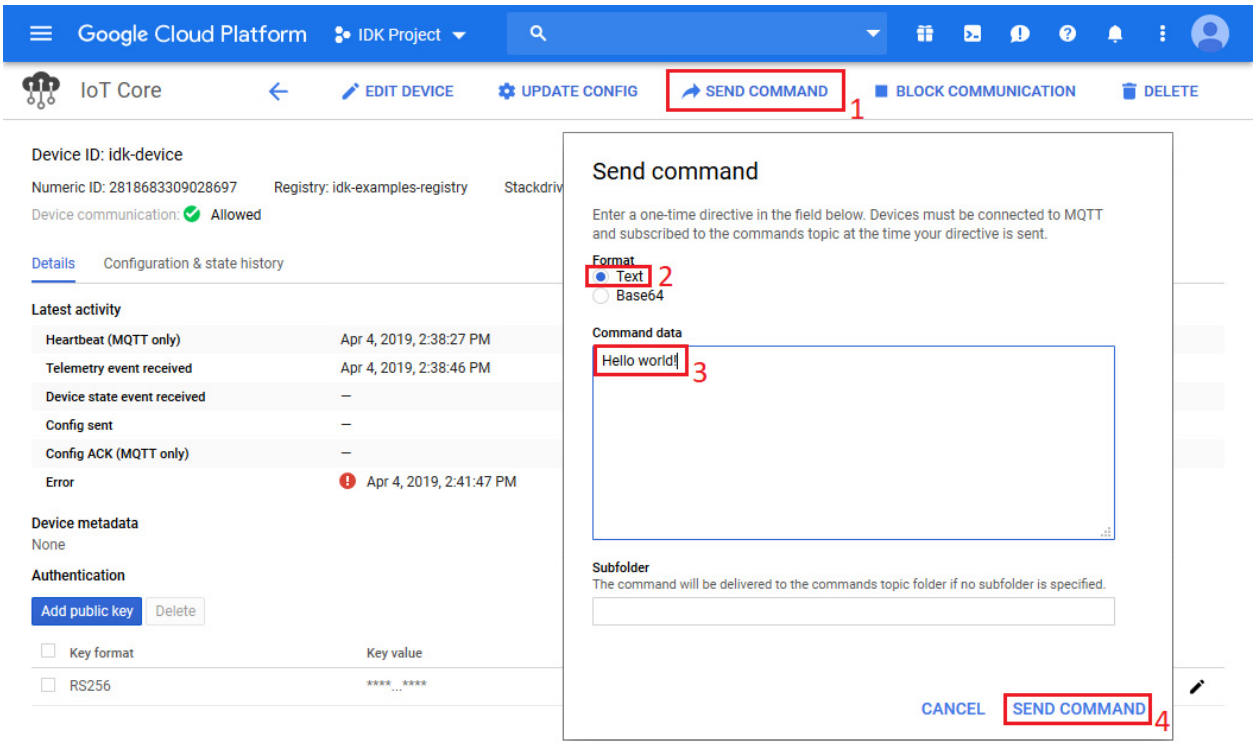


Figure 8. Sending a Command to Device from Cloud UI

Reading Telemetry Data using PUB/SUB Service

Google Cloud UI does not offer direct method for viewing of payloads of published device messages. To view telemetry data it is necessary to create a subscription to devices default telemetry topic using the PUB/SUB service and then pull the messages using the gcloud command−line tool.

Following the instructions from PUB/SUB Quick start guide create a new subscription to the telemetry topic which was created when first device was added to device registry. Once the subscription is created all new messages that are published to the telemetry topic can be pulled using the gcloud command line tool that is available in the Cloud Shell.

In this case the topic *projects/sublime−state−230144/topics/events* was selected as default telemetry topic for IoT devices. A subscription named *projects/sublime−state−230144/subscriptions/sensors* was created to receive messages published to events topic. Using the Cloud Shell environment it is possible to pull messages with the gcloud command line tool as illustrated in Figure 9.

```
gloud pubsub subscriptions pull  −auto−ack sensors
```

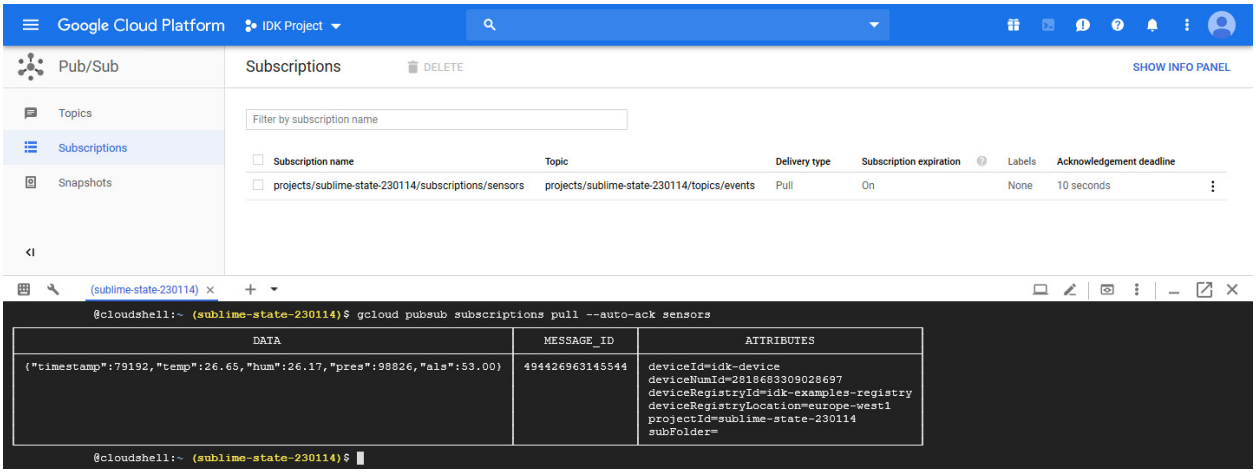


Figure 9. Using Gloud Command Line Tool in Cloud Shell to Display Payloads of Messages Published by IoT Devices

PUBLICATION ORDERING INFORMATION

LITERATURE FULFILLMENT:
Literature Distribution Center for ON Semiconductor
19521 E. 32nd Pkwy, Aurora, Colorado 80011 USA
Phone: 303−675−2175 or 800−344−3860 Toll Free USA/Canada
Fax: 303−675−2176 or 800−344−3867 Toll Free USA/Canada
Email: orderlit@onsemi.com

N. American Technical Support: 800−282−9855 Toll Free USA/Canada
Europe, Middle East and Africa Technical Support:
Phone: 421 33 790 2910

ON Semiconductor Website: **www.onsemi.com**

Order Literature: http://www.onsemi.com/orderlit

For additional information, please contact your local Sales Representative