INTELLIGENTLY CONNECTED

Eseye-Enabled Gemalto PLS62W-T

Quick Start Guide

- Document Reference: XXX
- 4th November 2019
- Version: 0.3

aws partner network

Advanced
Technology
Partner

IoT Competency

Version	Date	Author	Changes
0.1	18 Oct 2019	J Darley	First Draft
0.2	28 Oct 2019	J Darley	Added Security section
0.3	4 Nov 2019	J Darley	Examples updated

Contents

1 Introduction

The Eseye-Enabled PLS62W-T from Gemalto is a Smart Terminal utilising the Cinterion LTE Cat1 PLS62 worldwide cellular module. The PLS62 brings new simplicity to the deployment and IoT enablement of devices. Leveraging Gemalto's next generation Java embedded technology and Eseye's Anynet Secure zero touch provisioning, the plug-and-play solution powered by the multi-band LTE Cat-1 baseband with seamless fallback to 3G and/or 2G enables secure wireless connectivity to AWS IoT from anywhere in the world for a variety of industrial applications such as metering, remote monitoring, transportation, security and many more.

The terminal provides first time IoT developers and small-scale implementers with a flexible, cost effective solution to quickly launch enterprise optimization solutions that expand the Internet of Things. The zero touch provisioning agent and simplified command interface, the Eseye Telemetry Module (ETM) are written in Java running on the PLS62T-W, this allows you to set up and publish or subscribe to AWS topics in as few as 2 commands.

For more information on the PLS62T-W terminal itself see

https://www.gemalto.com/brochures-site/download-site/Documents/M2M_ELS61T_ELS31T_datasheet.pdf

For more information on the Eseye Anynet Secure solution see

https://www.eseye.com/solutions/anynet-secure/

2 Getting Started

2.1 Sign up for an AWS Account

If you have not already got an AWS account, you will need to set one up before you can connect the PLS62T-W

- Visit **https://aws.amazon.com**
- Sign up for an account

2.2 Subscribe to Anynet Secure Cellular Connectivity

In order to utilize the cellular services and the zero-touch provisioning provided by the Eseye-Enabled Cinterion module you will need to subscribe to an MQTT messaging plan from Eseye.

- Visit **https://aws.amazon.com/marketplace**
- Search for "Anynet Secure"
- Select the AnyNet Secure Cellular Connectivity service and then click subscribe

You will be prompted to login to your AWS Account. The pricing options for the service will be displayed.

- Click the Subscribe button and complete required information.
- Create a link account with Eseye.

This links your AWS to the Eseye systems in order to provision and manage your device connectivity.

Finally

- Install Eseye software into the AWS account.

This adds the zero-touch provisioning to your AWS IoT Core service

For more detail on this process see

https://eseye.zendesk.com/hc/en-us/article_attachments/360005143254/8465_AnyNet_Secure_Cellular_Connectivity_Sign_up.pdf

2.3 Create your Thing in AWS IoT Core

- Sign in to **https://aws.amazon.com/console**
- Navigate to the AWS IoT Core Service
- In the left navigation pane, choose Manage
- Click on the create button
- Click the "Create a Single Thing" button
- Give your Thing a Name
- Select AnynetThingType from the Thing Type drop down list
- Complete SIM ID using the SIM ID printed on the SIM card that you have received from Eseye with your PLS62T-W
- Click NEXT

You now have a choice of certificate options. Because you are using zero touch provisioning using the Eseye Anynet Secure you do not need to install a certificate. The X.509 certificate set required to encrypt your connection to AWS will we created and loaded to the PLS62T-W by the zero-touch process. You will not be required to manually download or install any certificate information to the device.

- Click the "Create thing without Certificate" button

If you haven't done so already please connect the antenna and switch on your PLS62T-W.

The software installed when you subscribed to the Eseye service will now activate your cellular connection. Within your AWS account the Anynet secure zero-touch provisioning process creates the Thing, assigns a default policy and requests a set of encryption keys and certificates from the Amazon Trust Service which are unique to the Thing and deliver them to the PLS62T-W.

These certificates are delivered securely to the PLS62-W module and once received are stored within a Java keystore held securely within the PLS62W module. There is no user access to the keys within the PLS62-W they are used directly by the on board SSL stack to encrypt the MQTT payloads sent in to AWS.

Provisioning can take up to 1 hour to complete although generally it is complete within 5-10 minutes and your PLS62T-W will connect to the cellular network read to publish messages to AWS and subscribe to AWS topics.

2.4 Tracking the progress of provisioning

Progress of the provisioning and certificate delivery is reported in the shadow for your Thing.

- Sign in to **https://aws.amazon.com/console**
- Navigate to the AWS IoT Core Service
- In the left navigation pane, choose Manage
- Click the Thing you wish to check progress on
- In the left hand navigation pane click shadow

The shadow document will be updated by the Eseye service as the provisioning and certificate delivery progresses. It will also report message consumption and location information for the device. And example of a completely provisioning device is shown below

```
{
  "reported": {
    "anynet": {
      "status": "Provisioned",
      "metadata": {
        "msisdn": 44762412345678,
        "bundle_info": {
          "package_id": 10001,
          "message": {
            "allowance": 5000,
            "remaining": 5000
          },
          "certificate_update": {
            "allowance": 1,
            "remaining": 0
          }
        },
        "location": {
          "lat": "47.12",
          "lon": "27.55",
          "mcc": "226",
          "mnc": "01"
        },
        "certificate_delivery": {
          "status": "Delivered",
          "when_sent": "2019-09-02 10:52:21",
          "when_delivered": "2019-09-02 11:07:43",
          "progress": "Complete"
        }
      },
      "log": {
        "2019-09-02T10:46:10.599973+00:00": "Info: Perform activation"
      }
    }
  }
}
```

2.5 Publishing data to AWS IoT Core

Two commands are used to publish data into AWS IoT core. Up to 8 concurrent topics can be configured for publish within the PLS62T-W.

AT+EMQPUBOPEN

This command manages publish message topic details.

Create and view publish message topics and associated indices. Upon a write command the parameters will be validated, and OK/ERROR returned.

The topic is not available for use until the URC '+EMQPUBOPEN:<idx>,<status>' is received.

Status is 0 for a successfully installed publish topic. -2 indicates the publish index is already in use and HAS NOT been changed.

Publish topic formats:

A plain text topic used within the +EMQPUBOPEN will be appended with the mqtt thingname in the form <topic>/<thingname> when messages are published to the broker.

Type	command	result
Test	AT+EMQPUBOPEN=?	+EMQPUBOPEN: (0-7),<topic>\r\n
Read	AT+EMQPUBOPEN?	List of publish topics
Write	AT+EMQPUBOPEN=<idx>,<topic>\r\n	OK\r\n or ERROR\r\n

e.g. AT+EMQPUBOPEN=0,demotopic

AT+EMQPUBLISH

Publish data to a topic referenced by index (subsequent to emqpubopen for the index)

pubdata is sent as text.

When a qos 0 message is published or a qos 1 message generates a puback the URC 'SEND OK\r\n' will be reported. If a pubnack is received 'SEND FAIL\r\n' will be reported.

Type	command	result
Test	AT+EMQPUBLISH=?	+EMQPUBLISH: (0-7),(0-1),<pubdata>\r\n
Read	AT+EMQPUBLISH?	OK\r\n
Write	AT+EMQPUBLISH=(0-7),(0-1),<pubdata>\r\n	OK\r\n or ERROR\r\n

e.g. AT+EMQPUBLISH=0,1,{\"temperature\": 24}

(NOTE the Escaped Quotation marks)

Index 0 is configured to topic "demotopic" using the AT+EMQPUBOPEN above

For the example the Thing is configured as "TestThing"

The publish example above will result in the data

{"temperature": 24}

Being delivered to topic

demotopic/TestThing

2.6 Subscribing to data from AWS IoT Core

A single command is used to subscribe to data from AWS IoT core

AT+EMQSUBOPEN

This command manages subscription topics.

Create and view subscription topics and associated indices. Upon a write command the parameters will be validated and OK/ERROR returned.

The topic subscription status is reported later with the URC '+EMQSUBOPEN:<idx>,<status>'.

Status is 0 for a successful subscription, -1 for a subNack from the broker and -2 if the index is already in use.

Subscribe topic formats:

A plain text topic used with SUBOPEN will be appended with the mqtt thingname in the form <topic>/<thingname> when subscribing with the broker.

Type	command	result
Test	AT+EMQSUBOPEN=?	+EMQSUBOPEN: (0-7),<topic>[,qos]\r\n
Read	AT+EMQSUBOPEN?	List of subscribed topics
Write	AT+EMQSUBOPEN=(0-7),<topic>[,qos]\r\n	OK\r\n or ERROR\r\n

e.g. AT+EMQSUBOPEN=0,TestSub

Now publish data to the topic TestSub/TestThing where TestThing is the configured Thing name

- Sign in to **https://aws.amazon.com/console**
- Navigate to the AWS IoT Core Service
- In the left navigation pane, choose Test
- Click "Publish to a topic"
- Specify the topic, for this example we would use TestSub/TestThing (note the topic is case sensitive)
- Enter some data to be published to the topic e.g {Hello World}
- Click PUBLISH

Data will then be published to the topic TestSub/TestThing

This will then be sent to all clients subscribed to this topic, therefore the data will be sent to the PLS62T-W

A URC (unsolicited Response Code) will be displayed reporting the arrival of the data

+EMQ:<idx>,<len>\r\n<data>\r\n

Indicates data received on one of the subscribed topics. The topic is indicated by idx and len forewarns of the length of data to follow.

e.g.

+EMQ: 0,13

{Hello World}

2.7 Using the PLS62T-W with AWS IoT core shadow

The code within the PLS62T-W can also interact with the shadow. The example shadow interaction uses the ORANGE LED in the PLS62T-W.

When the PLS26T-W connects to AWS IoT core it subscribes to the shadow updates and performs a GET to read the correct state

The “desired” state is read and the LED set accordingly. It then publishes a “reported” state into the device shadow.

To control the LED.

Using the publish instruction in 2.6 above

Publish the data

```
{
  "state":
  {
    "desired":
    {
      "led":"off"
    }
  }
}
```

On topic

    $aws/things/<THINGNAME>/shadow/update

e.g. for our example TestThing this would be

    $aws/things/TestThing/shadow/update

As the PLS62T-W is subscribed to shadow updates it will receive the update delta notification and will switch OFF the Orange LED. It will then publish back to the shadow the following data

```
{
  "state":
  {
    "reported":
    {
      "led":"off"
    }
  }
}
```

2.8 Security using the PLS62T-W and Anynet Secure

The AWS developers guide states

“Each connected device must have a credential to access the message broker or the Device Shadow service. All traffic to and from AWS IoT must be encrypted over Transport Layer Security (TLS). Device credentials must be kept safe in order to send data securely to the message broker.”

The PL62T-W and the Eseye Anynet Secure service addresses this requirement while keeping deployment and securing of devices simple.

When the Thing is created the Eseye provisioning service for the associated SIM ID is invoked from within your AWS account. This allows the PLS62-W to register on the network and be read for the X.509 certificate delivery.

A set of X.509 certificates and a private are requested from the Amazon Trust Service (ATS) and are delivered securely over the cellular network to the Anynet Secure SIM card along with the unique Thing name and the Amazon Resource Name (ARN) to define the AWS endpoint supporting the Thing. Once the certificates are received a Java midlet within the PLS62T-W moves the certificates and key into a Java keystore held securely within the PLS62W module. Java keystores (SE keystore) provide a mechanism for storing and deploying X.509 certificates and private keys. The SE keystore contains the key pairs for signing data.

The material stored within the keystore is used directly by the SSL stack within the PLS62W module to encrypt the MQTT payloads over TLS. The end user does not see or handle the security materials throughout its use.

Updates to the keys and certificates can also be delivered to the device throughout its in-service life using the same mechanisms. This allows managed a certificate rotation policy to be applied and also protects against changes in rootCA providers without the need to visit the Thing.

The microcontroller within the PLS62T-W provides no access to the Java keystore and hence no access to the security keys deployed to the Thing. This ensures that the whatever privileges are applied to access of the external ports of the PLS62-W the security of the keys is maintained. Deletion of the keys from the PLS62-W is also supported when the Thing is deleted.

2.9 Policy Management with the Eseye Anynet Secure

By default, the Anynet provisioning service will create Things with an open policy. This occurs because the provisioning has no knowledge of your application and the publish and subscribe topics and processing you are using with your AWS account.

It is best practice to limit the policy to allow access to only the required resource and to limit that access to only authenticated devices. Therefore editing or replacing the default policy installed is advised.

It is advised to only "Allow" required Actions. i.e. if the Thing only publishes and never subscribes remove the subscribe Action from the Allow policy statement, alternatively specifically "Deny" the Action

For each Action within the statement the resource access should also be restricted

e.g. using a Resource control such as

```
"Resource": ["arn:aws:iot:us-east-1:123456789012:client/${iot:Connection.Thing.ThingName}"]
```

Would restrict the connection to a Thing using a Thing name registered in the AWS IoT registry and authenticated against the ARN.

For more detailed examples of how to adjust policies to manage resource access please review:

https://docs.aws.amazon.com/iot/latest/developerguide/iot-policies.html