Using AWS IoT SDK on ARTIK

To use AWS IoT SDK on ARTIK, you need to have basic understanding of AWS IoT. As a prerequisite, please follow AWS IoT Getting Started Guide at https://docs.aws.amazon.com/iot/latest/developerguide/iot-gs.html to familiarize yourself with AWS IoT concepts . Please finish sections:

- Sign in to the AWS IoT Console
- Register a Device in the Registry
- Create and Activate a Device Certificate. Please download the certificate, private key and root CA for AWS, and save to your host machine. Root CA can be downloaded from https://www.symantec.com/content/en/us/enterprise/verisign/roots/VeriSign-Class%203-Public-Primary-Certification-Authority-G5.pem

- Create an AWS IoT Policy
- Attach an AWS IoT Policy to a Device Certificate
- Attach a Certificate to a Thing

Using AWS IoT SDK on ARTIK 5x/7x

Using AWS IoT SDK on ARTIK 530(s) and 710(s) are straightforward. Since ARTIK 530(s)/710(s) runs Ubuntu, you can easily develop applications on ARTIK devices by using AWS IoT SDK.

For C SDK, please refer to https://docs.aws.amazon.com/iot/latest/developerguide/iot-embedded-c-sdk.html.

If you want to use the AWS IoT Device SDK for JavaScript, please follow the instructions at https://docs.aws.amazon.com/iot/latest/developerguide/iot-device-sdk-node.html.

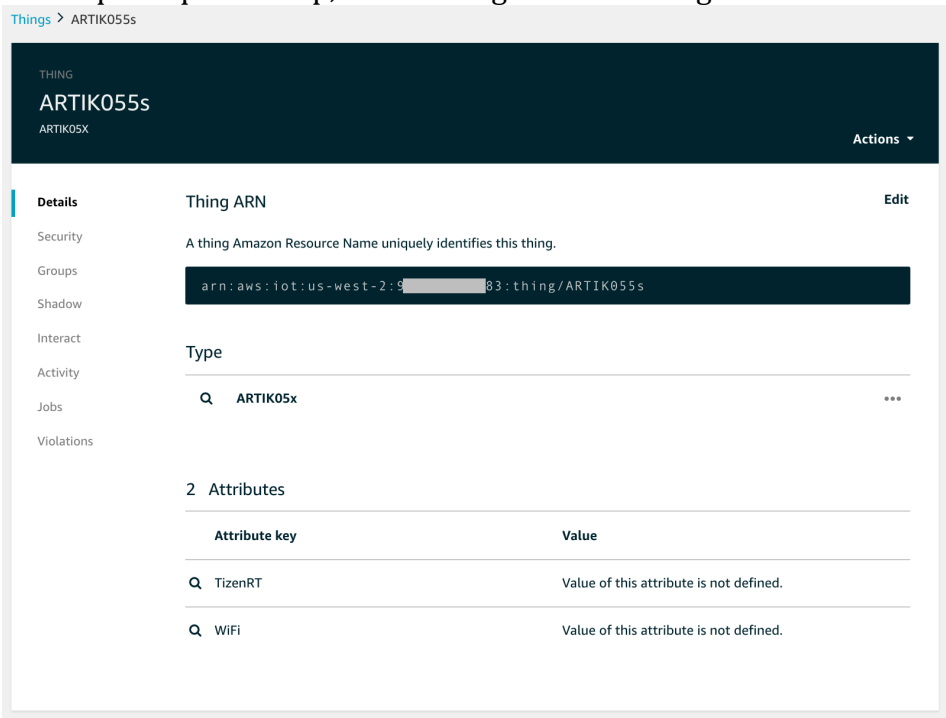
If Python is your preferred programming language, please refer to https://github.com/aws/aws-iot-device-sdk-python.

Using AWS IoT Embedded C SDK on ARTIK 05x

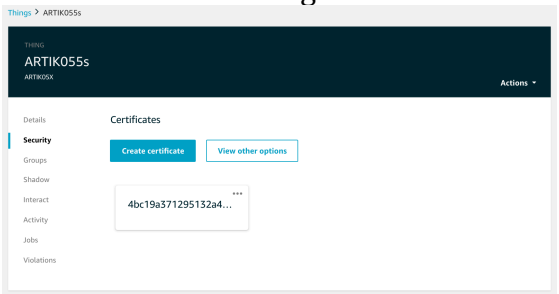
Using AWS IoT Embedded C SDK on ARTIK 05x requires integration with TizenRT. As the primary governing standards in TizenRT are POSIX and ANSI standards, migrating 3rd party libraries to the OS is not complicated.

To do this, we need to set up Linux development environment on the host machine and cross compile TizenRT source code with AWS IoT Embedded C SDK for ARTIK 05x device. If you don't have the development environment ready yet, please follow https://developer.artik.io/documentation/artik-05x/advanced-concepts/prepare-dev-env.html to install Linux environment and cross-compilation toolchain for ARTIK 05x.

In this prerequisite step, we have registered a Thing named ARTIK055s,



created and activated a Device Certificate, generated an AWS IoT Policy and attached the Policy to the Device Certificate. As the last step, we attached the Certificate to the Thing ARTIK055s we registered earlier.



1. Getting TizenRT, AWS IoT Device SDK sources

1.1 Cloning TizenRT source files

Clone the latest TizenRT tree to your Linux host machine.

```
git clone https://github.com/SamsungARTIK/TizenRT
```

In TizenRT source tree, you can find 3rd party libraries under *external* directory, and example applications under *apps/examples* directory.

1.2 Preparing AWS IoT sample apps

Download the attached *aws_examples.tar.gz* to `~/TizenRT/apps/examples/`.

```
cd ~/TizenRT/apps/examples
tar xvfz aws_examples.tar.gz
```

1.3 Preparing AWS IoT Embedded C SDK

If you don't have time to integrate this step by step, or don't want to go through this yourself (strongly suggested)

Untar the attached *aws.tar.gz* to `~/TizenRT/external/`. In the *aws.tar.gz*, we already take care of most part of sections 2 to 4 in this instruction. After this step, please go to step 2.2 directly to configure dependencies, then follow 2.3 to configure Makefile, Kconfig, once you are done, move on to step 5, "5. Configuring Certificates and AWS IoT endpoints"

```
cd ~/TizenRT/external
tar xvfz aws.tar.gz
```

Otherwise,

If you want to understand the nuts and bolts of TizenRT integration

We will walk you through the steps in the following section. Check out AWS IoT Device SDK to the *external* directory. Rename the AWS SDK folder to aws.

```
cd ~/TizenRT/external
git clone https://github.com/aws/aws-iot-device-sdk-embedded-C.git
mv aws-iot-device-sdk-embedded-C aws
```

2. Configuring AWS IoT Embedded C SDK for TizenRT

Now, we will compile AWS IoT Embedded C SDK into *libexternal.a*.

2.1 Copying over required Makefiles

Rename the original SDK Makefile to a different name, and rename `platform/linux` & `samples/linux` directory to `platform/TizenRT` & `samples/TizenRT`. Under `~/TizenRT/external/`,

```
cd aws
mv Makefile Makefile.aws
mv platform/linux    platform/TizenRT
mv samples/linux     samples/TizenRT
```

Download the patch.tar.gz we provide to `~/TizenRT` directory, untar it and recursively copy over all the required Makefiles from the patch file to the SDK directory.

```
cd ~/TizenRT
tar xvfz patch.tar.gz
cp –rp  ./patch/external/aws/* ./TizenRT/external/aws/
```

2.2 Configuring dependencies

AWS IoT Device SDK requires mbedtls, create `external/include` directory. Under `~/TizenRT/external`,

```
mkdir include
cd include
```

Download the attached *mbedtls* header tar file mbedtls_header.tar.gz to `external/include` directory and untar it.

```
tar xvfz mbedtls_header.tar.gz
```

2.3 Configuring external library Makefile, KConfig

The bold lines are required changes.

external/KConfig

In `external/KConfig`, search for `source "$EXTERNALDIR/sysview/Kconfig'`block, and add the bold line before it. This will include AWS IoT Embedded C SDK configuration option into TizenRT menuconfig.

```
source "$EXTERNALDIR/aws/Kconfig"
source "$EXTERNALDIR/sysview/Kconfig"
```

external/Makefile

In `external/Makefile`, add the bold lines below between lines in order to compile AWS IoT Embedded C SDK into TizenRT.
Look for `ifeq ($(CONFIG_DM),y)` and `ifeq ($(CONFIG_SYSVIEW),y)`, and include the `ifeq ($(CONFIG_AWS_SDK), y)` block below in between.

```
ifeq ($(CONFIG_DM),y)
CFLAGS+=-I$(TOPDIR)/../framework/include/dm
endif

ifeq ($(CONFIG_AWS_SDK), y)
include aws/Make.defs
endif

ifeq ($(CONFIG_SYSVIEW),y)
include sysview/Make.defs
endif
```

Add the following lines right after the code above.

```
# External Directories

# BUILDIRS is the list of top-level directories containing
Make.defs files

BUILDIRS   := $(dir $(wildcard */Make.defs))

# CONFIGURED_EXT is the external directories that should be
built in
#   the current configuration.

CONFIGURED_EXT =

define Add_EXTLIB
  include $(1)Make.defs
endef

$(foreach BDIR, $(BUILDIRS), $(eval $(call
Add_EXTLIB,$(BDIR))))

define SDIR_template
$(1)_$(2):
     $(Q) $(MAKE) -C $(1) $(2) TOPDIR="$(TOPDIR)"
EXTDIR="$(EXTDIR)"

endef
```

```
$(foreach SDIR, $(CONFIGURED_EXT), $(eval $(call
SDIR_template,$(SDIR),all)))
$(foreach SDIR, $(CONFIGURED_EXT), $(eval $(call
SDIR_template,$(SDIR),depend)))
$(foreach SDIR, $(BUILDIRS), $(eval $(call
SDIR_template,$(SDIR),clean)))
$(foreach SDIR, $(BUILDIRS), $(eval $(call
SDIR_template,$(SDIR),distclean)))
```

Look for the part that generates the final $(BIN) , and add the bold part below. This will include AWS SDK into the libexternal.a.

```
$(BIN): $(OBJ) $(foreach SDIR, $(CONFIGURED_EXT),
$(SDIR)_all) iotivity_build
     $(call ARCHIVE, $@, $(OBJS))
```

Look for the part that cleans the binaries, and add the bold lines below.

```
artiksdk_clean:
ifeq ($(CONFIG_ARTIK_SDK),y)
     $(Q) $(MAKE) -C artik-sdk clean
endif

awssdk_clean:
ifeq ($(CONFIG_AWS_SDK), y)
     $(Q) echo "Cleaning AWS IoT Device SDK"
     $(Q) rm -f $(TOPDIR)/../external/aws/*.o
endif

custom_clean:
     $(foreach OBJFILE, $(OBJS), rm -f $(OBJFILE))

clean: custom_clean iotivity_clean artiksdk_clean
awssdk_clean
     $(call DELFILE, $(BIN))
     $(call CLEAN, $(OBJS))
     $(call CLEAN)
```

If you configured your AWS SDK by using the aws.tar.gz we provided, you don't need to go through the steps below. Please jump to step 5 directly.

2.4 external/aws/Makefile

Add the newly added mbedtls header file path `external/include/` to the Makefile.

```
CFLAGS += -Iinclude
CFLAGS += -I../include
CFLAGS += -Iexternal_libs/jsmn
```

```
CFLAGS += -I$(AWS_PLATFORM_DIR)/common
```

3. Connecting AWS IoT sample application to AWS SDK library

The AWS sample apps under `~/TizenRT/apps/examples/aws` directory actually link to the sample apps under `~/TizenRT/external/aws/samples`.

We need to make few changes to the source code.

3.1 shadow_sample.c

In `external/aws/samples/TizenRT/shadow_sample/shadow_sample.c`, , change

```
int main(int argc, **char argv)
```

to

```
int aws_shadow(int argc, **argv)
```

3.2 shadow_console_echo.c

In `external/aws/samples/TizenRT/shadow_sample_console_echo/shadow_console_echo.c`, change

```
int main(int argc, **char argv)
```

to

```
int aws_shadow_console_echo(int argc, char** argv)
```

3.3 subscribe_publish_library_sample.c

In `external/aws/samples/TizenRT/subscribe_publish_library_sample/subscribe_publish_library_sample.c`, make the following variables, change

```
int main(int argc, **char argv)
```

to

```
int aws_subscribe_publish_library(int argc, char **argv)
```

3.4 subscribe_publish_sample.c

In `external/aws/samples/TizenRT/subscribe_publish_sample/subscribe_publish_sample.c`, , change

```
int main(int argc, **char argv)
```

to

```
int aws_subscribe_publish(int argc, char **argv)
```

4. Preparing for compilation

4.1 shadow_sample.c

In `external/aws/samples/TizenRT/shadow_sample/shadow_sample.c`, make `parseInputArgsForConnectParams` a static function.

```
static void parseInputArgsForConnectParams(int argc, char
**argv) {
…
}
```

4.2 subscribe_publish_library_sample.c

In `external/aws/samples/TizenRT/subscribe_publish_library_sample/subscribe_publish_library_sample.c`, make the following variables and function

```
/**
* @brief Default cert location
*/

static char certDirectory[PATH_MAX + 1] = “../../../certs”;
/**
* @brief Default MQTT HOST URL is pulled from the
aws_iot_config.h
*/
static char HostAddress[255] = AWS_IOT_MQTT_HOST;

/**
* @brief Default MQTT port is pulled from the
aws_iot_config.h
*/
static uint32_t port = AWS_IOT_MQTT_PORT;
static void parseInputArgsForConnectParams(int argc, char
**argv) {
…
}
```

4.3 subscribe_publish_sample.c

In `external/aws/samples/TizenRT/subscribe_publish_sample/subscribe_publish_sample.c`, make the following variables and function

```
/**
* @brief Default cert location
```

```
*/

static char certDirectory[PATH_MAX + 1] = "../../../certs";
/**
* @brief Default MQTT HOST URL is pulled from the
aws_iot_config.h
*/
static char HostAddress[255] = AWS_IOT_MQTT_HOST;

/**
* @brief Default MQTT port is pulled from the
aws_iot_config.h
*/
static uint32_t port = AWS_IOT_MQTT_PORT;


/**
* @brief This parameter will avoid infinite loop of publish
and exit the program after certain number of publishes
*/
static uint32_t publishCount = 0;

static void iot_subscribe_callback_handler(…) {
…
}

static void disconnectCallbackHandler(AWS_IoT_Client
*pClient, void *data){
…
}

static void parseInputArgsForConnectParams(int argc, char
**argv) {
…
}
```

4.4 network_mbedtls_wrapper.c

In external/aws/platform/TizenRT/mbedtls/network_mbedtls_wrapper.c,

Add #include "certData.c",

```
#include "aws_iot_error.h"
#include "aws_iot_log.h"
#include "network_interface.h"
#include "network_platform.h"
```

#include “certData.c”

In iot_tls_connect() function, make the following changes to match the function signature.

Search for `IOT_DEBUG("  . Loading the CA root certificate ...");`

In the line below, replace
`ret = mbedtls_x509_crt_parse_file(&(tlsDataParams->cacert), pNetwork->tlsConnectParams.pRootCALocation);`

with
`ret = mbedtls_x509_crt_parse(&(tlsDataParams->cacert), root_ca_pem, rootCaLen);`

Moving down a few lines, and look for
`IOT_DEBUG("  . Loading the client cert. and key...");`

In the line below, replace
`ret = mbedtls_x509_crt_parse_file(&(tlsDataParams->clicert), pNetwork->tlsConnectParams.pDeviceCertLocation);`

with
`ret = mbedtls_x509_crt_parse(&(tlsDataParams->clicert), client_cert_pem, clientCertLen);`

Then look for beneath it, replace
`ret = mbedtls_pk_parse_keyfile(&(tlsDataParams->pkey), pNetwork->tlsConnectParams.pDevicePrivateKeyLocation, "");`

with
`ret =  mbedtls_pk_parse_key(&(tlsDataParams->pkey), client_private_key_pem, clientPrivateKeyLen, NULL, 0);`

4.5 timer_platform.h

In `external/aws/platform/TizenRT/common/timer_platform.h`, add the bold lines below.

```
/**
 * definition of the Timer struct. Platform specific
 */
struct Timer {
     struct timeval end_time;
};
```

```
/* Convenience macros for operations on timevals.
 *    NOTE: `timercmp' does not work for >= or <=.  */
#define timerisset(tvp)   ((tvp)->tv_sec || (tvp)->tv_usec)
#define timerclear(tvp)   ((tvp)->tv_sec = (tvp)->tv_usec = 0)
#define timercmp(a, b, CMP)
    (((a)->tv_sec == (b)->tv_sec) ?
     ((a)->tv_usec CMP (b)->tv_usec) :
     ((a)->tv_sec CMP (b)->tv_sec))
#define timeradd(a, b, result)
    do {
        (result)->tv_sec = (a)->tv_sec + (b)->tv_sec;
        (result)->tv_usec = (a)->tv_usec + (b)->tv_usec;
        if ((result)->tv_usec >= 1000000)
        {
            ++(result)->tv_sec;
            (result)->tv_usec -= 1000000;
        }
    } while (0)

#define timersub(a, b, result)
    do {
        (result)->tv_sec = (a)->tv_sec - (b)->tv_sec;
        (result)->tv_usec = (a)->tv_usec - (b)->tv_usec;
        if ((result)->tv_usec < 0) {
            --(result)->tv_sec;
            (result)->tv_usec += 1000000;
        }
    } while (0)
```

4.6 aws_iot_config.h

In `external/aws/samples/TizenRT/subscribe_publish_sample/aws_iot_config.h`, look for `#define DISABLE_METRICS false`, change it to

```
#define DISABLE_METRICS 0
```

5. Configuring certificates and AWS IoT endpoints

5.1 certData.tar.gz

Download the attached certData.tar.gz to `~/TizenRT/external/aws/certs/` and untar it.

```
cd ~/TizenRT/external/aws/certs/
tar xvfz certData.tar.gz
```

5.2 Configuring certificates

Open `certData.c`, follow the comments to copy the whole lines in the root CA certificate file, cert.pem, and privateKey.pem, and paste the lines in string format to arrays root_ca_pem, client_cert_pem and client_private_key_pem respectively.

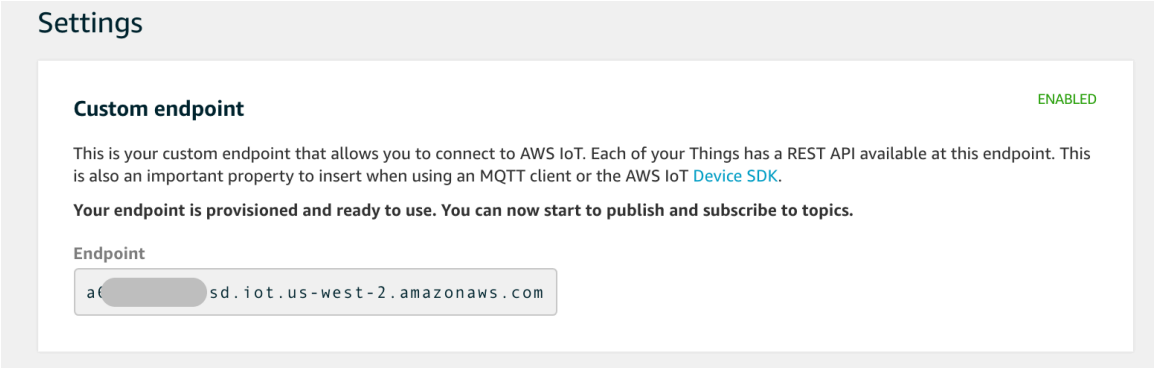
Again, root CA certificate can be downloaded from https://www.symantec.com/content/en/us/enterprise/verisign/roots/VeriSign-Class%203-Public-Primary-Certification-Authority-G5.pem.

Here is an example of how root_ca_pem looks like. Put each line in quotes and append "\r\n" to each line of the certificate file.

```
const unsigned char root_ca_pem[] =
"-----BEGIN CERTIFICATE-----\r\n"
"MIIE0zCCA7ugAwIBAgIQGNrRniZ96LtKIVjNzGs7SjANBgkqhkiG9w0BAQUFADCB\r\n"
"yjELMAkGA1UEBhMCVVMxFzAVBgNVBAoTDlZlcmlTaWduLCBJbmMuMR8wHQYDVQQL\r\n"
"ExZWZXJpU2lnbiBUcnVzdCBOZXR3b3JrMTowOAYDVQQLEzEoYykgMjAwNiBWZXJp\r\n"
"U2lnbiwgSW5jLiAtIEZvciBhdXRob3JpemVkIHVzZSBvbmx5MUUwQwYDVQQDEzxW\r\n"
"ZXJpU2lnbiBDbGFzcyAzIFB1YmxpYyBQcmltYXJ5IENlcnRpZmljYXRpb24gQXV0\r\n"
"aG9yaXR5IC0gRzUwHhcNMDYxMTA4MDAwMDAwWhcNMzYwNzE2MjM1OTU5WjCByjEL\r\n"
"MAkGA1UEBhMCVVMxFzAVBgNVBAoTDlZlcmlTaWduLCBJbmMuMR8wHQYDVQQLExZW\r\n"
"ZXJpU2lnbiBUcnVzdCBOZXR3b3JrMTowOAYDVQQLEzEoYykgMjAwNiBWZXJpU2ln\r\n"
"biwgSW5jLiAtIEZvciBhdXRob3JpemVkIHVzZSBvbmx5MUUwQwYDVQQDEzxWZXJp\r\n"
"U2lnbiBDbGFzcyAzIFB1YmxpYyBQcmltYXJ5IENlcnRpZmljYXRpb24gQXV0aG9y\r\n"
"aXR5IC0gRzUwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCvJAgIKXo1\r\n"
"nmAMqudLO07cfLw8RRy7K+D+KQL5VwijZIUVJ/XxrcgxiV0i6CqqpkKzj/i5Vbex\r\n"
"t0uz/o9+B1fs70PbZmIVYc9gDaTY3vjgw2IIPVQT60nKWVSFJuUrjxuf6/WhkcIz\r\n"
"SdhDY2pSS9KP6HBRTdGJaXvHcPaz3BJ023tdS1bTlr8Vd6Gw9KIl8q8ckmcY5fQG\r\n"
"BO+QueQA5N06tRn/Arr0PO7gi+s3i+z016zy9vA9r911kTMZHRxAy3QkGSGT2RT+\r\n"
"rCpSx4/VBEnkjWNHiDxpg8v+R70rfk/Fla4OndTRQ8Bnc+MUCH7lP59zuDMKz10/\r\n"
"NIeWiu5T6CUVAgMBAAGjgbIwga8wDwYDVR0TAQH/BAUwAwEB/zAOBgNVHQ8BAf8E\r\n"
"BAMCAQYwbQYIKwYBBQUHAQwEYTBfoV2gWzBZMFcwVRYJaW1hZ2UvZ2lmMCEwHzAH\r\n"
"BgUrDgMCGgQUj+XTGoasjY5rw8+AatRIGCx7GS4wJRYjaHR0cDovL2xvZ28udmVy\r\n"
"aXNpZ24uY29tL3ZzbG9nby5naWYwHQYDVR0OBBYEFH/TZafC3ey78DAJ80M5+gKv\r\n"
"MzEzMA0GCSqGSIb3DQEBBQUAA4IBAQCTJEowX2LP2BqYLz3q3JktvXf2pXkiOOzE\r\n"
"p6B4Eq1iDkVwZMXnl2YtmAl+X6/WzChl8gGqCBpH3vn5fJJaCGkgDdk+bW48DW7Y\r\n"
"5gaRQBi5+MHt39tBquCWIMnNZBU4gcmU7qKEKQsTb47bDN0lAtukixlE0kF6BWlK\r\n"
"WE9gyn6CagsCqiUXObXbf+eEZSqVir2G3l6BFoMtEMze/aiCKm0oHw0LxOXnGiYZ\r\n"
"4fQRbxC1lfznQgUy286dUV4otp6F01vvpX1FQHKOtw5rDgb7MzVIcbidJ4vEZV8N\r\n"
"hnacRHr2lVz2XTIIM6RUthg/aFzyQkqFOFSDX9HoLPKsEdao7WNq\r\n"
"-----END CERTIFICATE-----\r\n";
```

5.3 Configuring endpoint and device name

Go to AWS IoT Console, select "Settings" from left pane, and you can find custom endpoint for your Thing.



In this tutorial, we will verify the `subscribe_publish_sample` application(same configuration is required if you want to explore other sample apps), Navigate to ~/`TizenRT`

`/external/aws/samples/TizenRT/subscribe_publish_sample` directory and change configurations in `aws_iot_config.h`.

`AWS_IOT_MQTT_HOST` should be defined as your Rest API Endpoint.
`AWS_IOT_MQTT_CLIENT_ID` should be unique for every device. Here, we are using "`ARTIK055s_1`" as an example.
`AWS_IOT_MY_THING_NAME` is the Thing we defined. ARTIK055s.

```
#define AWS_IOT_MQTT_HOST             "a6e…..9sd.iot.us-west-
2.amazonaws.com" ///< Customer specific MQTT HOST. The same will be
used for Thing Shadow
#define AWS_IOT_MQTT_PORT              443 ///< default port for
MQTT/S
#define AWS_IOT_MQTT_CLIENT_ID         "ARTIK055s_1" ///< MQTT
client ID should be unique for every device
#define AWS_IOT_MY_THING_NAME           "ARTIK055s" ///< Thing Name
of the Shadow this device is associated with
```

6. Building TizenRT Image

To compile and load images to ARTIK 05x, please refer to tutorial at https://developer.artik.io/documentation/artik-05x/advanced-concepts/communicate-053.html.

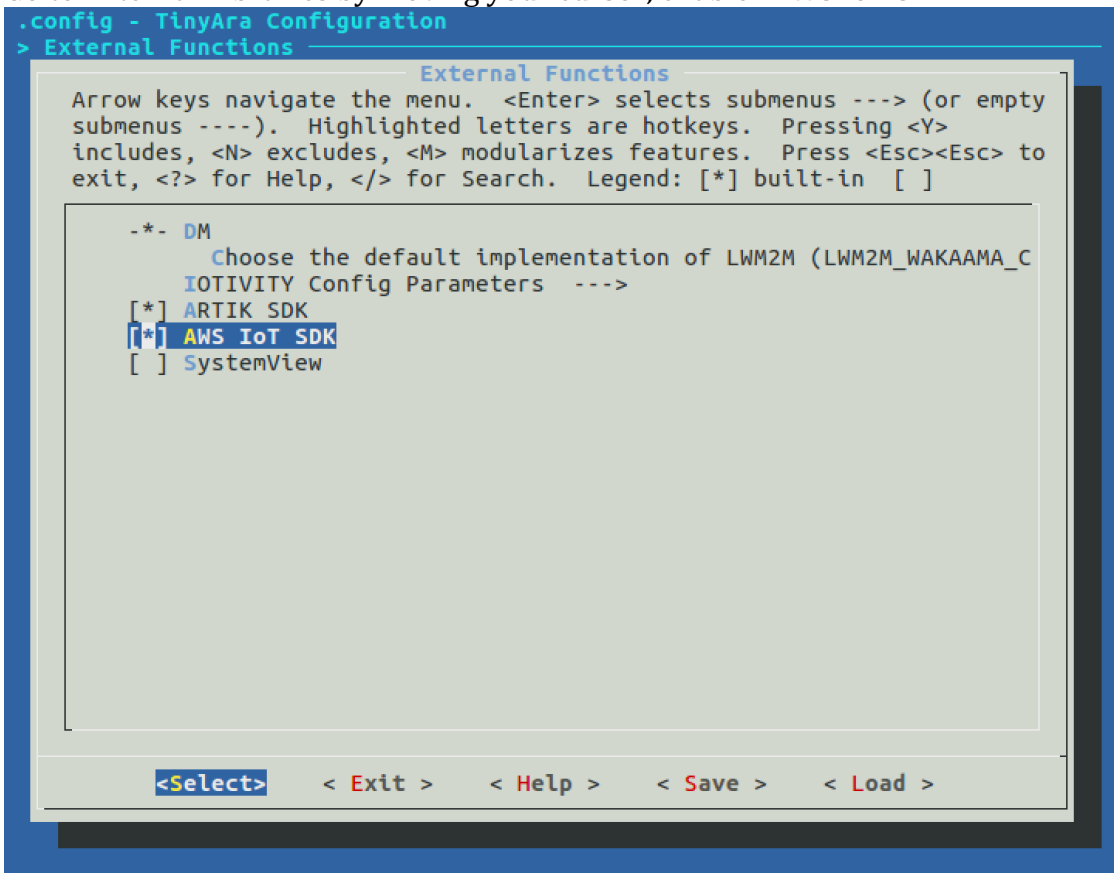
6.1 Configuring ARTIK055s build

```
cd  ~/TizenRT/os/,
cd tools/
./configure.sh artik055s/extra
cd ..
```

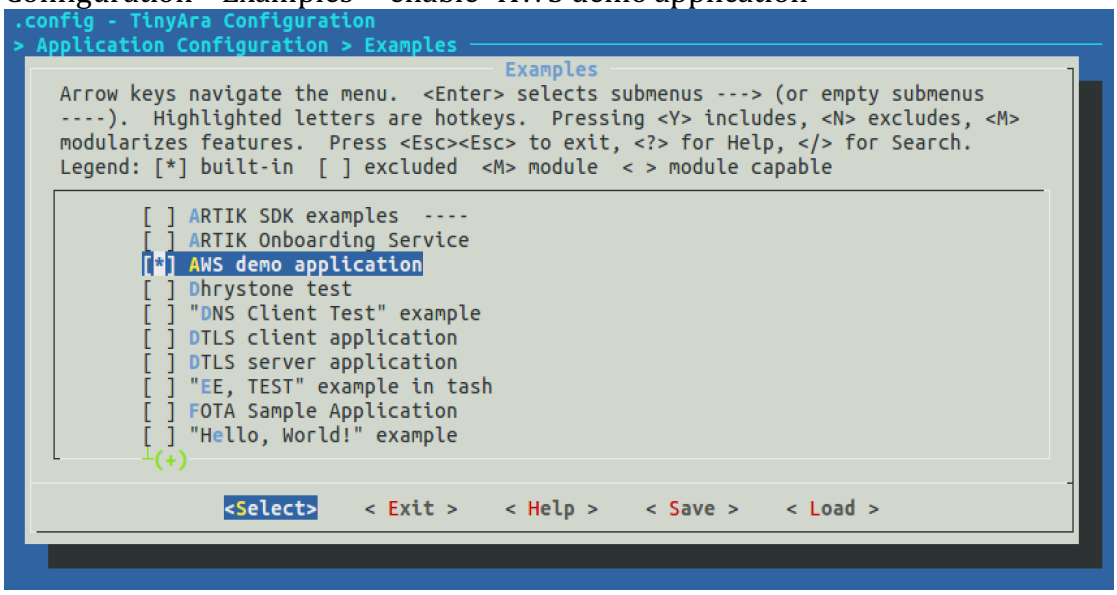
6.2 Configuring menuconfig,

`make menuconfig`

Go to External Libraries by moving your cursor, enable “AWS IoT SDK”.



Exit this configuration page and from the main menu, go to Application Configuration->Examples-> enable “AWS demo application”



Exit menuconfig.

6.3 Building TizenRT image and re-flashing the device

```
make
make download ALL
```

7. Connecting ARTIK 05x to AWS IoT

7.1 Configuring WiFi on ARTIK055s

Restart 055s, you can manually configure WiFi by following https://developer.artik.io/documentation/artik-05x/getting-started/communicate.html "Manually Set Up Wi-Fi" section.

7.2 Configuring device time

From TASH, configure your date to make sure it matches UTC time. This step is required otherwise AWS server certificate can not be verified.

```
TASH >> date -s Aug 12 12:00:05 2018
```

7.3 Testing AWS IoT integration

From TASH, you should be able to see a TizenRT application `aws_sample`,

```
TASH>>aws_sample
usage : aws_sample <example_no>
    ex) aws_sample 4 ==> runs aws_susbcribe_publish

    1. aws_shadow
    2. aws_shadow_console_echo
    3. aws_subscribe_publish_library
    4. aws_subscribe_publish
```

Launch the application with option 4, which invokes the subscribe_publish_sample app we configured earlier.

```
AWS IoT SDK Version 3.0.1-

Connecting...
Subscribing...
-->sleep
Subscribe callback
sdkTest/subhello from SDK QOS0 : 0 hello from SDK QOS0 : 0
```

```
Subscribe callback
sdkTest/subhello from SDK QOS1 : 1 hello from SDK QOS1 : 1
-->sleep
Subscribe callback
sdkTest/subhello from SDK QOS0 : 2 hello from SDK QOS0 : 2
Subscribe callback
sdkTest/subhello from SDK QOS1 : 3 hello from SDK QOS1 : 3
-->sleep
Subscribe callback
…
```