Embedded System Design Techniques™

Connecting Edge Devices to the IoT using Amazon FreeRTOS

Class 5: Creating your own Application

March 23rd, 2018

Jacob Beningo

Presented by:

CEC CONTINUING EDUCATION CENTER

Digi-Key ELECTRONICS

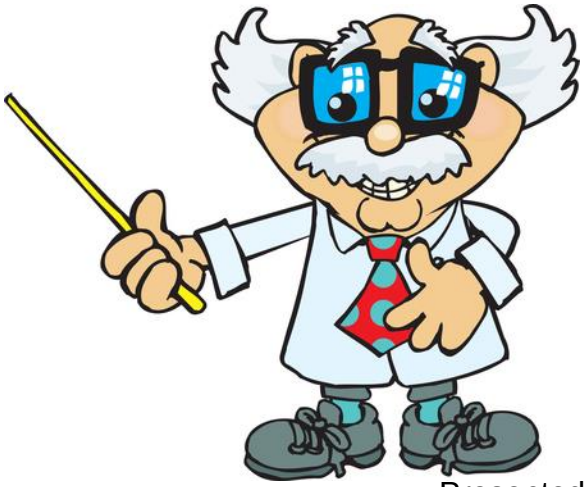
Course Overview

Topics:

- Introduction to Amazon FreeRTOS
- Amazon Web Services Fundamentals
- Setting up and Configuring Amazon FreeRTOS
- Amazon FreeRTOS Behind the Scenes
- **Creating your own a:FreeRTOS Application**

Session Overview

- Publishing a Topic
- Subscribing to Topics
- Visualization
- OTA
- Going Further



Presented by:

Problems you will encounter …

- The learning curve
  - No overarching architecture documents describing how the various components interact
  - Takes time to setup tracing successfully
- The CC3220sf bootloader will on start-up automatically copy the default application from external memory to internal memory
  - Can only run your application in debug
  - Will need to learn how to use Uniflash
- May encounter issues with non XDS110 programmers
- Issues connecting to the cloud
  - Policy or certificate issues

What you will need to know …

RTOS Concepts

- FreeRTOS
- Tasks, Messages, Queues, Semaphores, etc

How to trace the application

Publish / Subscribe Model

How to publish to different topics

Understand advanced features

Publishing to a Topic

Topics are used by the message broker to route messages from the publishing client (edge device) to clients that are subscribing to that client.

freertos/demos/echo

freertos/demos/sensordata

freertos/production/heartbeat

Publishing to a Topic

```
458     if( xReturned == pdPASS )
459     {
460         configPRINTF( ( "MQTT echo test echoing task created.\r\n" ) );
461
462         /* Subscribe to the echo topic. */
463         xReturned = prvSubscribe();
464     }
465
466     if( xReturned == pdPASS )
467     {
468         /* MQTT client is now connected to a broker.  Publish a message
469          * every five seconds until a minute has elapsed. */
470         for( x = 0; x < xIterationsInAMinute; x++ )
471         {
472             prvPublishNextMessage( x );
473
474             /* Five seconds delay between publishes. */
475             vTaskDelay( xFiveSeconds );
476         }
477     }
478
479     /* Disconnect the client. */
480     ( void ) MQTT_AGENT_Disconnect( xMQTTHandle, democonfigMQTT_TIMEOUT );
481
482     /* End the demo by deleting all created resources. */
483     configPRINTF( ( "MQTT echo demo finished.\r\n" ) );
484     vMessageBufferDelete( xEchoMessageBuffer );
485     vTaskDelete( xEchoingTask );
486     vTaskDelete( NULL ); /* Delete this task. */
487 }
```

Publishing to a Topic

```
339 static BaseType_t prvSubscribe( void )
340 {
341     MQTTAgentReturnCode_t xReturned;
342     BaseType_t xReturn;
343     MQTTAgentSubscribeParams_t xSubscribeParams;
344
345     /* Setup subscribe parameters to subscribe to echoTOPIC_NAME topic. */
346     xSubscribeParams.pucTopic = echoTOPIC_NAME;
347     xSubscribeParams.pvPublishCallbackContext = NULL;
348     xSubscribeParams.pxPublishCallback = prvMQTTCallback;
349     xSubscribeParams.usTopicLength = ( uint16_t ) strlen( ( const char * ) echoTOPIC_NAME );
350     xSubscribeParams.xQoS = eMQTTQoS1;
351
352     /* Subscribe to the topic. */
353     xReturned = MQTT_AGENT_Subscribe( xMQTTHandle,
354                                       &xSubscribeParams,
355                                       democonfigMQTT_TIMEOUT );
356
357     if( xReturned == eMQTTAgentSuccess )
358     {
359         configPRINTF( ( "MQTT Echo demo subscribed to %s\r\n", echoTOPIC_NAME ) );
360         xReturn = pdPASS;
361     }
362     else
363     {
364         configPRINTF( ( "ERROR:  MQTT Echo demo could not subscribe to %s\r\n", echoTOPIC_NAME ) );
365         xReturn = pdFAIL;
366     }
367
368     return xReturn;
369 }
```

Publishing to a Topic

```
87 /**
88  * @brief The topic that the MQTT client both subscribes and publishes to.
89  */
90 #define echoTOPIC_NAME          ( ( const uint8_t * ) "freertos/demos/echo" )
```

Create other topics to subscribe and publish to for an example device:

```
87 /**
88  * @brief The topic that the MQTT client both subscribes and publishes to.
89  */
90 #define echoTOPIC_NAME          ( ( const uint8_t * ) "freertos/demos/echo" )
91
92 /**
93  * @brief The topic that the MQTT client both subscribes and publishes to for raw sensor data.
94  */
95 #define echoTOPIC_SENSORRAW     ( ( const uint8_t * ) "freertos/demos/sensorraw")
96
97 /**
98  * @brief The topic that the MQTT client both subscribes and publishes to for current speed.
99  */
100 #define echoTOPIC_SPEED     ( ( const uint8_t * ) "freertos/demos/speed")
```

Identify the topics that make the most sense for your application

Publishing to a Topic

```
349 static BaseType_t prvSubscribe( const uint8_t * SubscribeTopic )
350 {
351     MQTTAgentReturnCode_t xReturned;
352     BaseType_t xReturn;
353     MQTTAgentSubscribeParams_t xSubscribeParams;
354
355     /* Setup subscribe parameters to subscribe to echoTOPIC_NAME topic. */
356     xSubscribeParams.pucTopic = SubscribeTopic;
357     xSubscribeParams.pvPublishCallbackContext = NULL;
358     xSubscribeParams.pxPublishCallback = prvMQTTCallback;
359     xSubscribeParams.usTopicLength = ( uint16_t ) strlen( ( const char * ) SubscribeTopic );
360     xSubscribeParams.xQoS = eMQTTQoS1;
361
362     /* Subscribe to the topic. */
363     xReturned = MQTT_AGENT_Subscribe( xMQTTHandle,
364                                       &xSubscribeParams,
365                                       democonfigMQTT_TIMEOUT );
366
367     if( xReturned == eMQTTAgentSuccess )
368     {
369         configPRINTF( ( "MQTT Echo demo subscribed to %s\r\n", SubscribeTopic ) );
370         xReturn = pdPASS;
371     }
372     else
373     {
374         configPRINTF( ( "ERROR:  MQTT Echo demo could not subscribe to %s\r\n", SubscribeTopic ) );
375         xReturn = pdFAIL;
376     }
377
378     return xReturn;
379 }
```

Publishing to a Topic

```
244 static void prvPublishNextMessage( BaseType_t xMessageNumber, const uint8_t * SubscribeTopic )
245 {
246     MQTTAgentPublishParams_t xPublishParameters;
247     MQTTAgentReturnCode_t xReturned;
248     char cDataBuffer[ echoMAX_DATA_LENGTH ];
249
250     /* Check this function is not being called before the MQTT client object has
251      * been created. */
252     configASSERT( xMQTTHandle != NULL );
253
254     /* Create the message that will be published, which is of the form "Hello World n"
255      * where n is a monotonically increasing number. Note that snprintf appends
256      * terminating null character to the cDataBuffer. */
257     ( void ) snprintf( cDataBuffer, echoMAX_DATA_LENGTH, "Hello World %d", ( int ) xMessageNumber );
258
259     /* Setup the publish parameters. */
260     memset( &( xPublishParameters ), 0x00, sizeof( xPublishParameters ) );
261     xPublishParameters.pucTopic = SubscribeTopic;
262     xPublishParameters.pvData = cDataBuffer;
263     xPublishParameters.usTopicLength = ( uint16_t ) strlen( ( const char * ) SubscribeTopic );
264     xPublishParameters.ulDataLength = ( uint32_t ) strlen( cDataBuffer );
265     xPublishParameters.xQoS = eMQTTQoS1;
266
267     /* Publish the message. */
268     xReturned = MQTT_AGENT_Publish( xMQTTHandle,
269                                     &( xPublishParameters ),
270                                     democonfigMQTT_TIMEOUT );
271
272     if( xReturned == eMQTTAgentSuccess )
273     {
274         configPRINTF( ( "Echo successfully published '%s'\r\n", cDataBuffer ) );
275     }
276     else
277     {
278         configPRINTF( ( "ERROR:  Echo failed to publish '%s'\r\n", cDataBuffer ) );
279     }
280
281     /* Remove compiler warnings in case configPRINTF() is not defined. */
282     ( void ) xReturned;
283 }
```

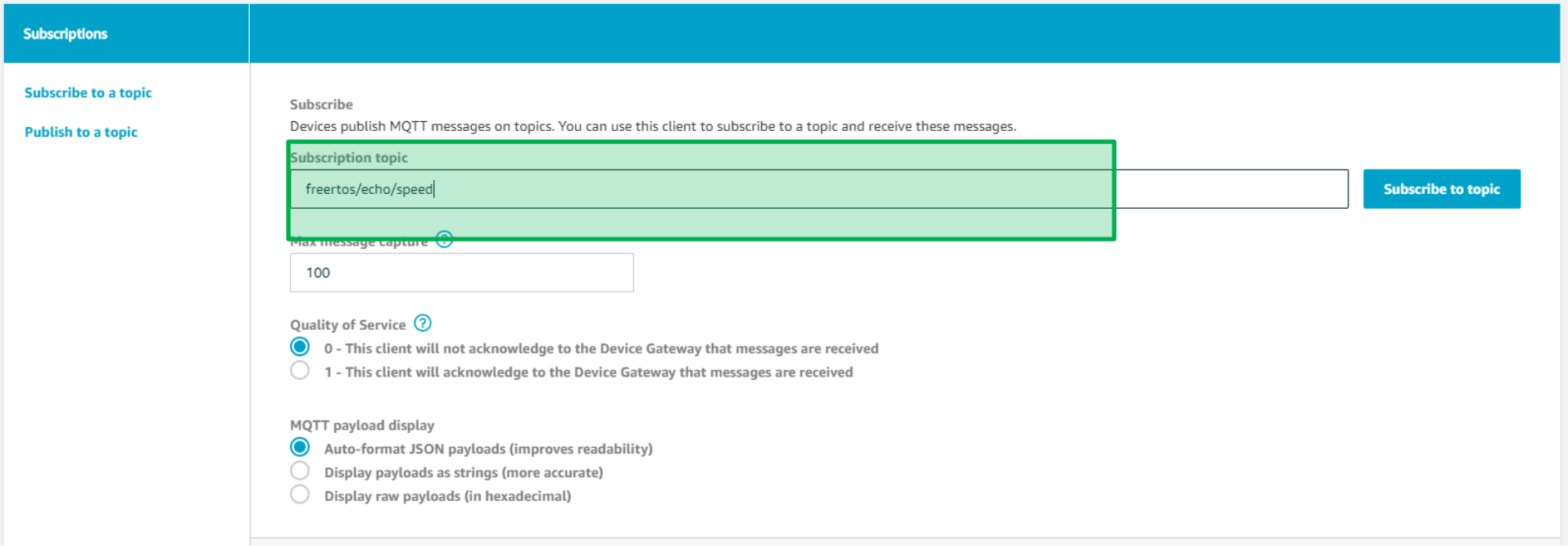
Publishing to a Topic

```
286 static void prvPublishSpeedMessage( BaseType_t SpeedData )
287 {
288     MQTTAgentPublishParams_t xPublishParameters;
289     MQTTAgentReturnCode_t xReturned;
290     char cDataBuffer[ echoMAX_DATA_LENGTH ];
291
292     /* Check this function is not being called before the MQTT client object has
293      * been created. */
294     configASSERT( xMQTTHandle != NULL );
295
296     /* Create the message that will be published, which is of the form "Hello World n"
297      * where n is a monotonically increasing number. Note that snprintf appends
298      * terminating null character to the cDataBuffer. */
299     ( void ) snprintf( cDataBuffer, echoMAX_DATA_LENGTH, "Speed: %d", ( int ) SpeedData );
300
301     /* Setup the publish parameters. */
302     memset( &( xPublishParameters ), 0x00, sizeof( xPublishParameters ) );
303     xPublishParameters.pucTopic = echoTOPIC_SPEED;
304     xPublishParameters.pvData = cDataBuffer;
305     xPublishParameters.usTopicLength = ( uint16_t ) strlen( ( const char * ) echoTOPIC_SPEED );
306     xPublishParameters.ulDataLength = ( uint32_t ) strlen( cDataBuffer );
307     xPublishParameters.xQoS = eMQTTQoS1;
308
309     /* Publish the message. */
310     xReturned = MQTT_AGENT_Publish( xMQTTHandle,
311                                     &( xPublishParameters ),
312                                     democonfigMQTT_TIMEOUT );
313
314     if( xReturned == eMQTTAgentSuccess )
315     {
316         configPRINTF( ( "Echo successfully published '%s'\r\n", cDataBuffer ) );
317     }
318     else
319     {
320         configPRINTF( ( "ERROR:  Echo failed to publish '%s'\r\n", cDataBuffer ) );
321     }
322
323     /* Remove compiler warnings in case configPRINTF() is not defined. */
324     ( void ) xReturned;
325 }
```

Publishing to a Topic

```
286 static void prvPublishSpeedMessage( BaseType_t Data, const uint8_t * SubscribeTopic, const uint8_t * Message )
287 {
288     MQTTAgentPublishParams_t xPublishParameters;
289     MQTTAgentReturnCode_t xReturned;
290     char cDataBuffer[ echoMAX_DATA_LENGTH ];
291
292     /* Check this function is not being called before the MQTT client object has
293      * been created. */
294     configASSERT( xMQTTHandle != NULL );
295
296     /* Create the message that will be published, which is of the form "Hello World n"
297      * where n is a monotonically increasing number. Note that snprintf appends
298      * terminating null character to the cDataBuffer. */
299     ( void ) snprintf( cDataBuffer, echoMAX_DATA_LENGTH, Message, ( int ) Data );
300
301     /* Setup the publish parameters. */
302     memset( &( xPublishParameters ), 0x00, sizeof( xPublishParameters ) );
303     xPublishParameters.pucTopic = SubscribeTopic;
304     xPublishParameters.pvData = cDataBuffer;
305     xPublishParameters.usTopicLength = ( uint16_t ) strlen( ( const char * ) SubscribeTopic );
306     xPublishParameters.ulDataLength = ( uint32_t ) strlen( cDataBuffer );
307     xPublishParameters.xQoS = eMQTTQoS1;
308
309     /* Publish the message. */
310     xReturned = MQTT_AGENT_Publish( xMQTTHandle,
311                                     &( xPublishParameters ),
312                                     democonfigMQTT_TIMEOUT );
313
314     if( xReturned == eMQTTAgentSuccess )
315     {
316         configPRINTF( ( "Echo successfully published '%s'\r\n", cDataBuffer ) );
317     }
318     else
319     {
320         configPRINTF( ( "ERROR:  Echo failed to publish '%s'\r\n", cDataBuffer ) );
321     }
322
323     /* Remove compiler warnings in case configPRINTF() is not defined. */
324     ( void ) xReturned;
325 }
```

Subscribing to Topics



Visualizing AWS Data

We are just getting the data to AWS!

Developers still need to

- Create their own dashboards
- Design visualizations
- Analyze the data
- Think through big data issues
- Pay for their data usage

AWS IoT OTA Demonstration

1) Create an Amazon S3 bucket
2) Grant OTA Update Permissions to AWS
3) Create an OTA Update Service Role
4) Create OTA User Policy
5) Attach the policy
6) Grant access to Code Signing Service
7) Test the OTA Feature

Going Further

1) Try to modify your own a:FreeRTOS application
   - Create different topics
   - Attempt to receive a published message from the server
2) Investigate AWS IoT jobs
3) Experiment with the OTA beta feature
4) Dig deeper into MQTT and the a:FreeRTOS libraries

Try Different Hardware

Espressif ESP32-DevKitC

Curiosity PIC32MZ EF

NXP LPC54018 Module

STM32L4 Discovery Kit

Presented by:

CEC CONTINUING EDUCATION CENTER

Digi-Key ELECTRONICS

Additional Resources

- Download Course Material for
  - C/C++ Doxygen Templates
  - Example source code
  - Blog
  - YouTube Videos
- Embedded Bytes Newsletter
  - http://bit.ly/1BAHYXm

From www.beningo.com under

- Blog > CEC – Connecting Edge Devices to the IoT using Amazon FreeRTOS