API Introduction of lib_mqtt

1. IoT_Error_t mqtt_init(MQTT_Client *pClient, IoT_Client_Init_Params *pInitParams);

API	IoT_Error_t mqtt_init(MQTT_Client *pClient, IoT_Client_Init_Params *pInitParams);
Function	mqtt client init
Parameters	pClient point to MQTT object
Parameters	pInitParams point to connect Parameters of mqtt
Return	Type of success or failure

2. IoT_Error_t mqtt_connect(MQTT_Client *pClient, IoT_Client_Connect_Params *pConnectParams);

API	IoT_Error_t mqtt_connect(MQTT_Client *pClient, IoT_Client_Connect_Params *pConnectParams);
Function	Parameters of MQTT connect
Parameters	pClient point to MQTT object
Parameters	pInitParams point to connect Parameters of mqtt
Return	Type of success or failure

3. IoT_Error_t mqtt_publish(MQTT_Client *pClient, const char *pTopicName, uint16_t topicNameLen, IoT_Publish_Message_Params *pParams);

API	IoT_Error_t mqtt_publish(MQTT_Client *pClient, const char *pTopicName, uint16_t topicNameLen,IoT_Publish_Message_Params *pParams);
Function	Publish mqtt messages to a topic
Parameters	pClient point to MQTT object
Parameters	pTopicName: Topic name
Parameters	topicNameLen: Length of topic name
Parameters	pParams: Publish message content
Return	Type of success or failure

4. IoT_Error_t mqtt_subscribe(MQTT_Client *pClient, const char *pTopicName, uint16_t topicNameLen, QoS qos, pApplicationHandler_t pApplicationHandler, void *pApplicationHandlerData);

API	IoT_Error_t mqtt_subscribe(MQTT_Client *pClient, const char *pTopicName, uint16_t topicNameLen,QoS qos, pApplicationHandler_t pApplicationHandler, void *pApplicationHandlerData);
Function	Subscribe to a mqtt topic
Parameters	pClient point to MQTT object
Parameters	pTopicName: Topic name
Parameters	topicNameLen: Length of topic name
Parameters	Argslen: Length of service request array parameters
Parameters	pApplicationHandler_t: handle of this subscribe
Parameters	pApplicationHandlerData: Passing data as parameters to the application handler callback
Return	Type of success or failure

5. IoT_Error_t mqtt_resubscribe(MQTT_Client *pClient);

API	IoT_Error_t mqtt_resubscribe(MQTT_Client *pClient);
Function	Re subscribe to a topic
Parameters	pClient point to MQTT object
Return	Type of success or failure

6. IoT_Error_t mqtt_unsubscribe(MQTT_Client *pClient, const char *pTopicFilter, uint16_t topicFilterLen);

API	IoT_Error_t mqtt_unsubscribe(MQTT_Client *pClient, const char *pTopicFilter, uint16_t topicFilterLen);
Function	Send unsubscribe message to remove mqtt topic
Parameters	pClient point to MQTT object
Parameters	pTopicFilter: Topic name
Parameters	topicFilterLen: The length of the topic name
Return	Type of success or failure

7. IoT_Error_t mqtt_disconnect(MQTT_Client *pClient);

API	IoT_Error_t mqtt_disconnect(MQTT_Client *pClient);
Function	Disconnect MQTT
Parameters	pClient point to MQTT object
Return	void

8. IoT_Error_t mqtt_yield(MQTT_Client *pClient, uint32_t timeout_ms);

API	IoT_Error_t mqtt_yield(MQTT_Client *pClient, uint32_t timeout_ms);
Function	Monitor TCP connection processing receive message
Parameters	pClient point to MQTT object
Parameters	timeout_ms: maximum milliseconds that a thread executes
Return	Type of success or failure

9. IoT_Error_t mqtt_attempt_reconnect(MQTT_Client *pClient);

API	IoT_Error_t mqtt_attempt_reconnect(MQTT_Client *pClient);
Function	MQTT reconnect
Function	Disconnect MQTT
Return	Type of success or failure

Key code annotation

```
static void mqtt_sub_pub_main( mico_thread_arg_t arg )
{
    IoT_Error_t rc = FAILURE;

    char clientid[40];
    char cPayload[100];
    int i = 0;
    MQTT_Client client;
    IoT_Client_Init_Params mqttInitParams = iotClientInitParamsDefault;
    IoT_Client_Connect_Params connectParams = iotClientConnectParamsDefault;
    IoT_Publish_Message_Params paramsQOS0;
    IoT_Publish_Message_Params paramsQOS1;

    /*
     * Enable Auto Reconnect functionality. Minimum and Maximum time of Exponenti
al backoff are set in aws_iot_config.h
     *  #AWS_IOT_MQTT_MIN_RECONNECT_WAIT_INTERVAL
     *  #AWS_IOT_MQTT_MAX_RECONNECT_WAIT_INTERVAL
     */
    mqttInitParams.enableAutoReconnect = true;
    mqttInitParams.pHostURL = MQTT_HOST;
    mqttInitParams.port = MQTT_PORT;
    mqttInitParams.mqttPacketTimeout_ms = 20000;
    mqttInitParams.mqttCommandTimeout_ms = 20000;
    mqttInitParams.tlsHandshakeTimeout_ms = 5000;
    mqttInitParams.disconnectHandler = disconnectCallbackHandler;
    mqttInitParams.disconnectHandlerData = NULL;
    mqttInitParams.isBlockOnThreadLockEnabled = true;
#ifdef MQTT_USE_SSL
    mqttInitParams.pRootCALocation = MQTT_ROOT_CA_FILENAME;
    mqttInitParams.pDeviceCertLocation = MQTT_CERTIFICATE_FILENAME;
    mqttInitParams.pDevicePrivateKeyLocation = MQTT_PRIVATE_KEY_FILENAME;
    mqttInitParams.isSSLHostnameVerify = false;
    mqttInitParams.isClientnameVerify = false;
    mqttInitParams.isUseSSL = true;
#else
    mqttInitParams.pRootCALocation = NULL;
    mqttInitParams.pDeviceCertLocation = NULL;
    mqttInitParams.pDevicePrivateKeyLocation = NULL;
    mqttInitParams.isSSLHostnameVerify = false;
    mqttInitParams.isClientnameVerify = false;
    mqttInitParams.isUseSSL = false;
#endif
```

```
//MQTTinit
    rc = mqtt_init( &client, &mqttInitParams );
    if ( SUCCESS != rc )
    {
        mqtt_log("aws_iot_mqtt_init returned error : %d ", rc);
        goto exit;
    }
//Set MQTT connect parameters
    connectParams.keepAliveIntervalInSec = 30;
    connectParams.isCleanSession = true;
    connectParams.MQTTVersion = MQTT_3_1_1;
    connectParams.pClientID = MQTT_CLIENT_ID;
    connectParams.clientIDLen = (uint16_t) strlen( MQTT_CLIENT_ID );
    connectParams.isWillMsgPresent = false;
    connectParams.pUsername = MQTT_USERNAME;
    connectParams.usernameLen = strlen(MQTT_USERNAME);
    connectParams.pPassword = MQTT_PASSWORD;
    connectParams.passwordLen = strlen(MQTT_PASSWORD);
//MQTT connecting
    mqtt_log("Connecting...");
    rc = mqtt_connect( &client, &connectParams );
    if ( SUCCESS != rc )
    {
        mqtt_log("Error(%d) connecting to %s:%d", rc, mqttInitParams.pHostURL, mqttIni
tParams.port);
        goto exit;
    }

//MQTT subscribing
    mqtt_log("Subscribing...");
    rc = mqtt_subscribe( &client, MQTT_SUB_NAME, strlen( MQTT_SUB_NAME ), QOS
0,
                    iot_subscribe_callback_handler, NULL );
    if ( SUCCESS != rc )
    {
        mqtt_log("Error subscribing : %d ", rc);
        goto exit;
    }

    mqtt_log("publish...");
    sprintf( cPayload, "%s : %d ", "hello from SDK", i );

    paramsQOS0.qos = QOS0;
    paramsQOS0.payload = (void *) cPayload;
    paramsQOS0.isRetained = 0;

    paramsQOS1.qos = QOS1;
    paramsQOS1.payload = (void *) cPayload;
    paramsQOS1.isRetained = 0;

    while ( 1 )
    {
        //Max time the yield function will wait for read messages
        rc = mqtt_yield( &client, 100 );
        if ( NETWORK_ATTEMPTING_RECONNECT == rc )
        {
            // If the client is attempting to reconnect we will skip the rest of the loop.
            mico_rtos_thread_sleep( 1 );
            continue;
        } else if ( NETWORK_RECONNECTED == rc )
        {
            mqtt_log("Reconnect Successful");
        }

        mqtt_log("-->sleep");
        mico_rtos_thread_msleep( 500 );
        sprintf( cPayload, "%s : %d ", "hello from SDK QOS0", i++ );
        paramsQOS0.payloadLen = strlen( cPayload );
//MQTT publish
        mqtt_publish( &client, MQTT_SUB_NAME, strlen( MQTT_SUB_NAME ), &params
QOS0 );

        sprintf( cPayload, "%s : %d ", "hello from SDK QOS1", i++ );
        paramsQOS1.payloadLen = strlen( cPayload );
        rc = mqtt_publish( &client, MQTT_SUB_NAME, strlen( MQTT_SUB_NAME ), &par
amsQOS1 );
        if ( rc == MQTT_REQUEST_TIMEOUT_ERROR )
        {
            mqtt_log("QOS1 publish ack not received");
            rc = SUCCESS;
        }
    }

    if ( SUCCESS != rc )
    {
        mqtt_log("An error occurred in the loop.\n");
    } else
    {
        mqtt_log("Publish done\n");
    }

    exit:
    mico_rtos_delete_thread( NULL );
}
```