SGCWEBSOCKETS · TECHNICAL DOCUMENT

AWS IoT Core

TsgcIoTAmazon_MQTT_Client — MQTT 3.1.1 / 5.0 client tuned for AWS IoT Core with SigV4 and X.509 mutual-TLS.

eSeGeCe

WWW.ESEGECE.COM

Overview

AWS IoT provides secure, bi-directional communication between Internet-connected devices such as sensors, actuators, embedded micro-controllers, or smart appliances and the AWS Cloud. This enables you to collect telemetry data from multiple devices, and store and analyze the data. You can also create applications that enable your users to control these devices from their phones or tablets.

At a glance

COMPONENT CLASS

`TsgcIoTAmazon_MQTT_Client`

STANDARDS / SPEC

AWS IoT — Developer Guide

TRANSPORTS

TCP, TLS

PLATFORMS

Windows, macOS, Linux, iOS, Android

FRAMEWORKS

VCL, FireMonkey, Lazarus / FPC

EDITION

Standard / Professional / Enterprise

Features

- Native Delphi implementation with full ANSI/Unicode support.

Technical specification

Standards & specs	AWS IoT — Developer Guide · AWS IoT MQTT topics · AWS Signature V4
Component class	`TsgcIoTAmazon_MQTT_Client` (unit `sgcIoT_Amazon_MQTT_Client`)
Frameworks	VCL, FireMonkey, Lazarus / FPC
Platforms	Windows, macOS, Linux, iOS, Android

Main properties

The principal published / public properties used to configure and drive the component. Consult the online help for the full list.

`MQTTAuthentication`	Published or public property used to configure or query the component.
`CustomAuthentication`	Published or public property used to configure or query the component.
`Active`	Published or public property used to configure or query the component.
`BoundPort`	Published or public property used to configure or query the component.
`BoundPortMax`	Published or public property used to configure or query the component.
`BoundPortMin`	Published or public property used to configure or query the component.
`MQTTHeartBeat`	Published or public property used to configure or query the component.
`WatchDog`	Published or public property used to configure or query the component.
`OnMQTTSubscribe`	Published or public property used to configure or query the component.
`OnMQTTUnSubscribe`	Published or public property used to configure or query the component.

Main methods

The principal public methods exposed by the component.

`Publish_ShadowGet()`	Public procedure exposed by the component.
`Subscribe_ShadowGet()`	Public procedure exposed by the component.

`Subscribe_ShadowGetAccepted()`	Public procedure exposed by the component.
`Subscribe_ShadowGetRejected()`	Public procedure exposed by the component.
`Publish_ShadowUpdate()`	Public procedure exposed by the component.
`Subscribe_ShadowUpdateAccepted()`	Public procedure exposed by the component.
`Subscribe_ShadowUpdateRejected()`	Public procedure exposed by the component.
`Subscribe_ShadowUpdateDelta()`	Public procedure exposed by the component.
`Subscribe_ShadowUpdateDocuments()`	Public procedure exposed by the component.
`SubscribeCreateCertificateFromCsrResponse()`	Public procedure exposed by the component.

Quick Start

Drop the component on a form, configure the properties below and activate it. The snippet that follows shows the typical **Connect to AWS IoT** configuration sourced from the online help.

> **About this scenario.** First, you must sign in your AWS console, register a new device and create a X.509 certificate for this device. Once is done, you can create a new TsgcIoTAmazon_MQTT_Client and connect to AWS IoT Server. For example:

Delphi (VCL / FireMonkey)

```
oClient := TsgcIoTAmazon_MQTT_Client.Create(nil);
oClient.Amazon.Endpoint := 'a2ohgdjqitsmij-ats.iot.us-west-2.amazonaws.com';
oClient.Amazon.ClientId := 'sgcWebSockets';
oClient.Certificate.CertFile := 'amazon-certificate.pem.crt';
oClient.Certificate.KeyFile := 'amazon-private.pem.key';
oClient.OnMQTTConnect := OnMQTTConnectEvent;
oClient.Active := True;

procedure OnMQTTConnect(Connection: TsgcWSConnection; const Session: Boolean; const ReturnCode: 
begin
ShowMessage('Connected to AWS');
end;
```

C++ Builder

```
oClient = new TsgcIoTAmazon_MQTT_Client();
oClient->Amazon->Endpoint = "a2ohgdjqitsmij-ats.iot.us-west-2.amazonaws.com";
oClient->Amazon->ClientId = "sgcWebSockets";
oClient->Certificate->CertFile = "amazon-certificate.pem.crt";
oClient->Certificate->KeyFile = "amazon-private.pem.key";
oClient->OnMQTTConnect = OnMQTTConnectEvent;
oClient->Active = true;

void OnMQTTConnect(TsgcWSConnection *Connection, const bool Session, const TmqttConnReturnCode
ReturnCode)
{
ShowMessage("Connected to AWS");
}
```

.NET (C#)

Common scenarios

The following scenarios are lifted verbatim from the online help. Each shows the configuration and method calls needed to drive the component through a specific real-world flow.

1 · Device Provisioning Service

Azure IoT allows you to register devices from code using DPS. Currently, the library supports registering a device passing the Scope Id and Registration Id as parameters.

Delphi (VCL / FireMonkey)

```
oClient := TsgcIoTAzure_MQTT_Client.Create(nil);
Try
  oClient.Certificate.CertFile := 'cert.pem';
  oClient.Certificate.KeyFile := 'key.pem';
  oClient.Certificate.Enabled := True;
  oResponse := TsgcIoT_Azure_OperationRegistrationState.Create;
  Try
    if oClient.ProvisioningDeviceClient_Register('scope_id', 'registration_id', oResponse) then
      ShowMessage('#Provisioning Register OK: ' + oResponse.Status)
    else
      ShowMessage('#Provisioning Register Error: ' + oResponse.Status);
  Finally
    FreeAndNil(oResponse);
  End;
Finally
  FreeAndNil(oClient);
End;
```

C++ Builder

```
TsgcIoTAzure_MQTT_Client* oClient = new TsgcIoTAzure_MQTT_Client(NULL);
try
{
    oClient->Certificate->CertFile = L"cert.pem";
    oClient->Certificate->KeyFile = L"key.pem";
    oClient->Certificate->Enabled = true;
    TsgcIoT_Azure_OperationRegistrationState* oResponse = new TsgcIoT_Azure_OperationRegistratio
    try
    {
        if (oClient->ProvisioningDeviceClient_Register(L"scope_id", L"registration_id", oRespons
            ShowMessage(L"#Provisioning Register OK: " + oResponse->Status);
        else
            ShowMessage(L"#Provisioning Register Error: " + oResponse->Status);
    }
    __finally
    {
        delete oResponse;
    }
}
__finally
{
    delete oClient;
}
```

.NET (C#)

```
TsgcIoTAzure_MQTT_Client oClient = new TsgcIoTAzure_MQTT_Client();
oClient.Certificate.CertFile = "cert.pem";
oClient.Certificate.KeyFile = "key.pem";
oClient.Certificate.Enabled = true;
TsgcIoT_Azure_OperationRegistrationState oResponse = new TsgcIoT_Azure_OperationRegistrationStat
if (oClient.ProvisioningDeviceClient_Register("scope_id", "registration_id", oResponse))
  MessageBox.Show("#Provisioning Register OK: " + oResponse.Status);
else
  MessageBox.Show("#Provisioning Register Error: " + oResponse.Status);
```

2 · Topics

The message broker uses topics to route messages from publishing clients to subscribing clients. The forward slash (/) is used to separate topic hierarchy. The following table lists the wildcards that can be used in the topic filter when you subscribe. # Must be the last character in the topic to which you are subscribing. Works as a wildcard by matching the current tree and all subtrees.

Delphi (VCL / FireMonkey)

```
oClient := TsgcIoTAmazon_MQTT_Client.Create(nil);
...
oClient.OnSubscribe := OnSubscribeEvent;

vPacketIdentifier := oClient.Subscribe('Sensor/moisture/room1');

procedure OnMQTTSubscribe(Connection: TsgcWSConnection; aPacketIdentifier: Word; aCodes: TsgcWSS
begin
if vPacketIdentifier = aPacketIdentifier then
ShowMessage('Subscribed to topic Sensor/moisture/room1');
end;

// Client, can send a message using Publish method.
oClient.Publish('Sensor/moisture/room1', '{"temp"=10}');

// Messages received from server, are dispatched OnMQTTPublishEvent.
// For extended payload access (string, bytes or stream), use OnMQTTPublishEx.
procedure OnMQTTPublish(Connection: TsgcWSConnection; aTopic, aText: string);
begin
DoLog('Received Message: ' + aTopic + ' ' + aText);
end;
```

C++ Builder

```
oClient = new TsgcIoTAmazon_MQTT_Client();
...
oClient->OnSubscribe = OnSubscribeEvent;

vPacketIdentifier = oClient->Subscribe("Sensor/moisture/room1");

void OnMQTTSubscribe(TsgcWSConnection *Connection, Word aPacketIdentifier, TsgcWSSUBACKS *aCodes
{
if (vPacketIdentifier == aPacketIdentifier)
{
ShowMessage("Subscribed to topic Sensor/moisture/room1");
}
}

// Client, can send a message using Publish method.
oClient->Publish("Sensor/moisture/room1", "{"\"temp"\"=10}");

// Messages received from server, are dispatched OnMQTTPublishEvent.
// For extended payload access (string, bytes or stream), use OnMQTTPublishEx.
void OnMQTTPublish(TsgcWSConnection *Connection, string aTopic, string aText)
{
DoLog("Received Message: " + aTopic + " " + aText);
}
```

.NET (C#)

3 · Upload Files

IoT hub facilitates file uploads from connected devices by providing them with shared access signature (SAS) URIs or X509 certificates.

Delphi (VCL / FireMonkey)

```
procedure UploadFileToAzure;
begin
  oDialog := TOpenDialog.Create(nil);
  Try
    if oDialog.Execute then
      AzureIoT.UploadFile(oDialog.FileName);
  Finally
    oDialog.Free;
  End;
end;
```

C++ Builder

```
void UploadFileToAzure()
{
    TOpenDialog* oDialog = new TOpenDialog(NULL);
    try
    {
        if (oDialog->Execute())
        {
            AnsiString fileName = oDialog->FileName;
            AzureIoT::UploadFile(fileName.c_str());
        }
    }
    __finally
    {
        oDialog->Free();
    }
}
```

.NET (C#)

```
void UploadFileToAzure()
{
  TOpenDialog oDialog = new TOpenDialog();
  if (oDialog.Execute())
  {
    AzureIoT.UploadFile(oDialog.FileName);
  }
}
```

4 · Connect to Azure IoT Hub

First, you must sign in to your Azure account, register a new device, and create an authentication method for this device. Once that is done, you can create a new TsgcIoTAzure_MQTT_Client and connect to Azure IoT Hub.

Delphi (VCL / FireMonkey)

```
oClient := TsgcIoTAzure_MQTT_Client.Create(nil);
oClient.Azure.IoTHub := 'youriothub.azure-devices.net';
oClient.Azure.DeviceId := 'YourDeviceId';
oClient.SAS.Enabled := True;
oClient.SAS.SecretKey := 'YourSecretKey';
oClient.OnMQTTConnect := OnMQTTConnectEvent;
oClient.Active := True;

procedure OnMQTTConnect(Connection: TsgcWSConnection; const Session: Boolean; const
ReturnCode: TmqttConnReturnCode);
begin
  ShowMessage('Connected to Azure IoT Hub');
end;
```

C++ Builder

```
TsgcIoTAzure_MQTT_Client *oClient = new TsgcIoTAzure_MQTT_Client();
oClient->Azure->IoTHub = "youriothub.azure-devices.net";
oClient->Azure->DeviceId = "YourDeviceId";
oClient->SAS->Enabled = true;
oClient->SAS->SecretKey = "YourSecretKey";
oClient->OnMQTTConnect = OnMQTTConnectEvent;
oClient->Active = true;

private void OnMQTTConnect(TsgcWSConnection *Connection, const bool Session, const TmqttConnRetu
{
  ShowMessage("Connected to Azure IoT Hub");
}
```

5 · Cloud To Device

IoT Hub provides three options for device apps to expose functionality to a back-end app:

```
Delphi (VCL / FireMonkey)
```

```
oClient.Subscribe_DirectMethod;
```

```
C++ Builder
```

```
oClient→Subscribe_DirectMethod();
```

6 · Device To Cloud

When sending information from the device app to the solution back end, IoT Hub exposes the following options:

```
Delphi (VCL / FireMonkey)
```

```
oClient.Send_DeviceToCloud('{"temp": 10}', azuIoTQoS1);
```

```
C++ Builder
```

```
oClient→Send_DeviceToCloud("{\"temp\": 10}", azuIoTQoS1);
```

Sources used to build this document

Every external claim links back to a primary source. The online-help references decode the canonical deep-link the company maintains for this component.

Primary standard / spec — AWS IoT — Developer Guide	docs.aws.amazon.com/iot/latest/developerguide/what-is-aws-iot.html
Primary standard / spec — AWS IoT MQTT topics	docs.aws.amazon.com/iot/latest/developerguide/topics.html
Primary standard / spec — AWS Signature V4	docs.aws.amazon.com/IAM/latest/UserGuide/reference_aws-signing.html
Online help — component page	www.esegece.com/help/sgcWebSockets/Components/IoT/Amazon/IoT_Amazon_MQTT_Client.htm
Delphi demo project (in the sgcWebSockets package)	`Demos\10.IoT_Clients`
Component page	www.esegece.com/products/websockets/iot/aws-iot/
Product page	www.esegece.com/products/websockets/

> **Document scope.** This document covers the publicly-documented surface of the AWS IoT Core component shipped with sgcWebSockets. For full property, method and event reference consult the online help linked above.