SGCWEBSOCKETS · TECHNICAL DOCUMENT

Azure IoT Hub

TsgcIoTAzure_MQTT_Client — MQTT 3.1.1 client tuned for Azure IoT Hub with SAS-token and X.509 authentication.

eSeGeCe

WWW.ESEGECE.COM

Overview

IoT Hub is a managed service, hosted in the cloud, that acts as a central message hub for bi-directional communication between your IoT application and the devices it manages. You can use Azure IoT Hub to build IoT solutions with reliable and secure communications between millions of IoT devices and a cloud-hosted solution backend. You can connect virtually any device to IoT Hub.

At a glance

COMPONENT CLASS

`TsgcIoTAzure_MQTT_Client`

STANDARDS / SPEC

Azure IoT — MQTT support

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	Azure IoT — MQTT support · Azure IoT — device-to-cloud
Component class	`TsgcIoTAzure_MQTT_Client` (unit `sgcIoT_Azure_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.

`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.
`Certificate`	Published or public property used to configure or query the component.
`SAS`	Published or public property used to configure or query the component.

Main methods

The principal public methods exposed by the component.

`SendAndWait_DeviceToCloud()`	Public function exposed by the component.
`PublishAndWait()`	Public function exposed by the component.

`ReadConnectionString()`	Public procedure exposed by the component.
`Subscribe_CloudToDevice()`	Public function exposed by the component.
`UnSubscribe_CloudToDevice()`	Public function exposed by the component.
`Send_DeviceToCloud()`	Public function exposed by the component.
`Subscribe_DeviceTwins()`	Public function exposed by the component.
`UnSubscribe_DeviceTwins()`	Public function exposed by the component.
`Subscribe_DirectMethod()`	Public function exposed by the component.
`UnSubscribe_DirectMethod()`	Public function 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 Azure IoT Hub** configuration sourced from the online help.

> **About this scenario.** 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");
}
```

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 · 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);
  }
}
```

3 · 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();
```

4 · 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 — Azure IoT — MQTT support	learn.microsoft.com/en-us/azure/iot-hub/iot-hub-mqtt-support
Primary standard / spec — Azure IoT — device-to-cloud	learn.microsoft.com/en-us/azure/iot-hub/iot-hub-devguide-messages-d2c
Online help — component page	www.esegece.com/help/sgcWebSockets/Components/IoT/Azure/IoT_Azure_MQTT_Client.htm
Delphi demo project (in the sgcWebSockets package)	`Demos\10.IoT_Clients\AzureWindowsService`
Component page	www.esegece.com/products/websockets/iot/azure-iot/
Product page	www.esegece.com/products/websockets/

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