

AJAN

Azure IoT Hub 連携
チュートリアル

目 次

第 1 章	はじめに	3
第 2 章	基本説明	4
2.1	Azure IoT Hub とは.....	4
2.2	Azure IoT Hub を使うための準備作業.....	6
2.3	Azure SDK のサンプルを呼び出してみる.....	8
2.4	AJAN から Python 連携を介して Azure IoT Hub にアクセスする.....	14
第 3 章	サンプルプログラム	19
3.1	サンプルプログラム.....	19
第 4 章	重要な情報	20

第1章 はじめに

本ドキュメントは、AJANのPython連携機能を用いて、Azure IoT Hubサービスとの連携方法について、チュートリアルを紹介します。

本ドキュメントでは、説明で表現している表記として下記のように定義します。

- ・ コマンドの書式の説明において、[]内の引数は省略できます。



本ドキュメント記載の、AJANはIoT用プログラミング言語です。
Interface Linux System上でのみ動作可能です。

第2章 基本説明

2.1 Azure IoT Hubとは

Azureは、マイクロソフトが提供するクラウドコンピューティングサービスです。

この中で提供されるサービスの種類は多岐に及び、データベースや仮想マシン、データ分析、IoTなどの各種分野にまたがって、多くのサービスが提供されています。

Azureの中で、Azure IoT Hubと呼ばれるサービスがあります。

Azure IoT Hubとは、マイクロソフトのWeb siteでは、以下のように説明されています。

IoT Hubは、クラウド内でホストされているマネージド サービスであり、IoTアプリケーションとそれが管理するデバイスの間の双方向通信に対する中央メッセージハブとして機能します。

要約すると、コンピュータなどのIoT端末から、Azureというクラウドサービスに、任意のデータを送ったり受け取ったりする為の出入り口であり、プロトコルやAPIが提供されているサービスの事を指します。

Azure IoT Hubという出入り口を介して、Azureのほかのサービスと連携する事ができます。

例えば コンピュータで回収した計測データを、Azure上のストレージに保存したり、Azure上のアプリケーションから指示データを、コンピュータ宛てに送る事ができます。

Azure IoT Hubは、複数のデータの出入り口を持つ事ができます。Azureでは、この出入り口を「デバイス」と呼称しています。「デバイス」は複数持つ事ができます。

Azure IoT Hubは、インターネットを介して、データの送受信を行います。送受信を行う為に、幾つか通信プロトコルを選択できるようになっており、2020/10時点で、以下のプロトコルが選択可能です。

- ・ HTTPS
- ・ AMQP
- ・ AMQP over WebSocket
- ・ MQTT
- ・ MQTT over WebSocket

例えば、Azure IoT Hubの、とある「デバイス」の通信プロトコルに MQTT を選択したら、IoT端末側も、MQTTプロトコルを使って、データの送受信を行う必要があります。

通信プロトコルに従って一から送受信を行うのは、非常に面倒かつ大変なので、Azure IoT Hub には、Azure IoT device SDKと呼ばれるライブラリが提供されています。

ライブラリは、2020/10時点で、以下のプログラム言語に対応しています。

- C
- C#
- Java
- Python
- Node.js

ライブラリは、WindowsだけでなくLinuxにも提供されており、Interface Linux Systemにもインストールして使用可能です。

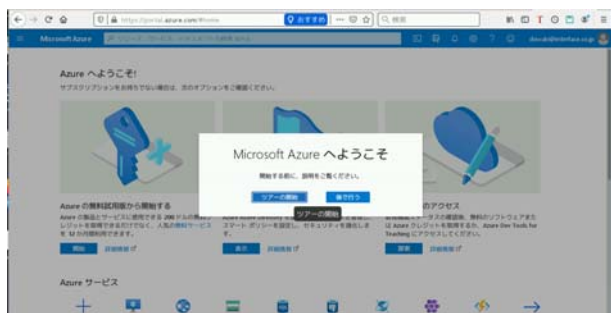
AJANは、Python連携機能を用いて、Python版のAzure SDKを介して、Azure IoT Hubの機能を呼び出す事ができる事を確認しています。

2.2 Azure IoT Hubを使うための準備作業

Azure IoT Hubを利用するには、Azureのアカウントを登録して、Azure IoT Hubサービスの利用契約を結ぶ必要があります。

Azureのアカウントを登録するには、「<https://portal.azure.com>」にアクセスして、新規加入であれば登録作業を行います。その後、IoT Hubの利用権(リソース)を購入します。

Azureの登録方法や IoT Hubの利用権購入については、マイクロソフトのWeb siteか、関連する書籍を参照ください。



Azureのアカウントを取得し、Azure IoT Hubサービスの利用権を購入後、Azureの画面から IoT Hubメニューを選択すると、詳細メニューに入れます。



詳細メニューでは、上部に現在使用している名前(リソース名。下図では「InterfaceIoTHub」)と、選択可能な項目が並んでいます。



Azure IoT Hubと端末の間で通信する基本単位は、「デバイス」です。

左ペインのメニュー項目「IoTデバイス」を選択すると、右ペインの構成が デバイス一覧のメニューに切り替わります。



上部の「新規作成」を選択すると、「デバイス」を追加できます。

作成手順の詳細は、マイクロソフトのWeb siteの説明を参照ください。



デバイスを追加すると、「IoTデバイス」メニューの「デバイスID」に、追加した名前が表示されます。

名前を選択すると、デバイスID、主キー、セカンダリキー、プライマリ接続文字列、など各種情報が表示されます。



ここで重要なのは、「プライマリ接続文字列」です。この値を使って、端末は Azure IoT Hubの指定された「デバイス」に対して、通信可能となります。



2.3 Azure SDKのサンプルを呼び出してみる

Azure IoT Hubにアクセスする為に、Azure SDKをインストールして、サンプルを動かしてみよう。

この後の説明で、AJANのPython連携機能を使う予定の為、ここでは Python版のAzure SDKをインストールします。

Interface Linux Systemの左上のメニューから、アプリケーション>システムツール>端末 を選択して、端末を開き、以下の青文字部分のコマンドを入力してください。

```
$ pip3 install azure-iot-device
$ pip3 install azure-iot-hub
```

	上記コマンドを実行する為に、インターネットに接続する必要があります。
	Interface Linux System 9 では、Azure SDKがプリインストール済みなので、上記作業は必要ありません。

次に、サンプルをダウンロードしましょう。以下の青文字部分のコマンドを入力してください。
(このコマンドは、サンプルを含むSDK全体を、ダウンロードして取り込んでいます)
成功すると、「Cloning」の後にメッセージが流れて、「done.」で完了します。

```
$ git clone https://github.com/Azure/azure-iot-sdk-python/
Cloning into 'azure-iot-sdk-python'...
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 46076 (delta 0), reused 1 (delta 0), pack-reused 46072
Receiving objects: 100% (46076/46076), 43.16 MiB | 10.38 MiB/s, done.
Resolving deltas: 100% (28662/28662), done.
```

ファイルの取り込みが成功すると、下位フォルダにサンプルが配置されます。

以下の例では、「azure-iot-sdk-python/azure-iot-device/samples/sync-samples/」に、Python言語のAzure サンプルがあります。

```
$ ls -l
合計 4
drwxr-xr-x 10 user user 4096 10月 13 14:30 azure-iot-sdk-python
$ ls -l azure-iot-sdk-python/
合計 128
-rw-r--r-- 1 user user 1162 10月 13 14:30 LICENSE
-rw-r--r-- 1 user user 16840 10月 13 14:30 README.md
-rw-r--r-- 1 user user 2809 10月 13 14:30 SECURITY.MD
drwxr-xr-x 6 user user 4096 10月 13 14:30 azure-iot-device
drwxr-xr-x 5 user user 4096 10月 13 14:30 azure-iot-hub
```



```

drwxr-xr-x 3 user user 4096 10月 13 14:30 azure-iot-nspkg
drwxr-xr-x 4 user user 4096 10月 13 14:30 azure_provisioning_e2e
-rw-r--r-- 1 user user 907 10月 13 14:30 build_packages.py
<略>
drwxr-xr-x 2 user user 4096 10月 13 14:30 vsts
$ ls -l azure-iot-sdk-python/azure-iot-device/samples/sync-samples/
合計 76
-rw-r--r-- 1 user user 4852 10月 13 14:30 README.md
-rw-r--r-- 1 user user 696 10月 13 14:30 get_twin.py
-rw-r--r-- 1 user user 2202 10月 13 14:30 provision_symmetric_key.py
-rw-r--r-- 1 user user 4343 10月 13 14:30 provision_symmetric_key_group.py
-rw-r--r-- 1 user user 2567 10月 13 14:30 provision_symmetric_key_with_payload.py
-rw-r--r-- 1 user user 2005 10月 13 14:30 provision_x509.py
-rw-r--r-- 1 user user 2137 10月 13 14:30 receive_direct_method.py
-rw-r--r-- 1 user user 1373 10月 13 14:30 receive_message.py
-rw-r--r-- 1 user user 1650 10月 13 14:30 receive_message_on_input.py
-rw-r--r-- 1 user user 1570 10月 13 14:30 receive_message_x509.py
-rw-r--r-- 1 user user 1130 10月 13 14:30 receive_twin_desired_properties_patch.py
-rw-r--r-- 1 user user 2403 10月 13 14:30 send_message.py
-rw-r--r-- 1 user user 1318 10月 13 14:30 send_message_to_output.py
-rw-r--r-- 1 user user 1608 10月 13 14:30 send_message_via_module_x509.py
-rw-r--r-- 1 user user 2699 10月 13 14:30 send_message_via_proxy.py
-rw-r--r-- 1 user user 1802 10月 13 14:30 send_message_x509.py
-rw-r--r-- 1 user user 866 10月 13 14:30 update_twin_reported_properties.py

```

サンプルの動かし方は、githubの当該URLに書かれています。

重要なのは、環境変数「IOTHUB_DEVICE_CONNECTION_STRING」に、使用したい「デバイス」の「プライマリ接続文字列」を設定した後に、サンプルを動かす事が肝要です。

Azure IoT Hubのページの「IoTデバイス」メニューで、通信したい「デバイス」を選択すると、下図のように表示されるので、「プライマリ接続文字列」の右端アイコンをクリックすると、文字列がクリップボードにコピーされます。



以下の青文字部分のように、「export」コマンドを使って、環境変数「IOTHUB_DEVICE_CONNECTION_STRING」に、コピーした文字列を貼り付けて設定してください。(文字列は、「”(ダブルクォーテーション)」で囲みます)

```
$ export IOTHUB_DEVICE_CONNECTION_STRING="HostName=<略>"
$ env
CLUTTER_IM_MODULE=xim
LD_LIBRARY_PATH=/opt/ros/lunar/lib
<略>
IOTHUB_DEVICE_CONNECTION_STRING=HostName=<略>
<略>
_=/usr/bin/env
```

環境変数が設定できたかどうかは、「env」コマンドを呼び出します。環境変数文字列がずらっと表示されるので、この中で「IOTHUB_DEVICE_CONNECTION_STRING」の値が正しくセットされているか確認してください。

環境変数の設定が出来たら、早速サンプルプログラムを動かしてみましょう。

「send_message.py」サンプルは、単純な文字列を、Azure IoT Hubに向けて送信するサンプルです。これを動かしてみましょう。

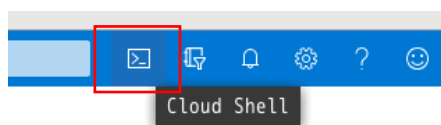
以下の青文字部分のコマンドを入力して、動かしてみます。

正しく動けば、「send message #1」という風にメッセージが、約10回流れて終了します。

```
$ cd azure-iot-sdk-python/azure-iot-device/samples/sync-samples/
$ python3 send_message.py
sending message #1
sending message #2
sending message #3
sending message #4
sending message #5
sending message #6
sending message #7
sending message #8
sending message #9
sending message #10
```

サンプルが送信している文字列は、どのようなものが実際に送られているのか確認するには、Azure IoT Hubの上で、Cloud Shellを起動してコマンドを発行する事で確認できます。

Azureの画面上部で、下図のようなアイコンをクリックすると「Cloud Shell」が起動します。



「Cloud Shell」は、Azureの中で動く、一種の仮想シェルです。このシェルは、Bash形式とPowerShell形式の2種類を選択可能です。

ここでは、Bash形式を選択したと仮定します。

「Cloud Shell」を起動した後、以下の青文字部分のコマンドを入力します。

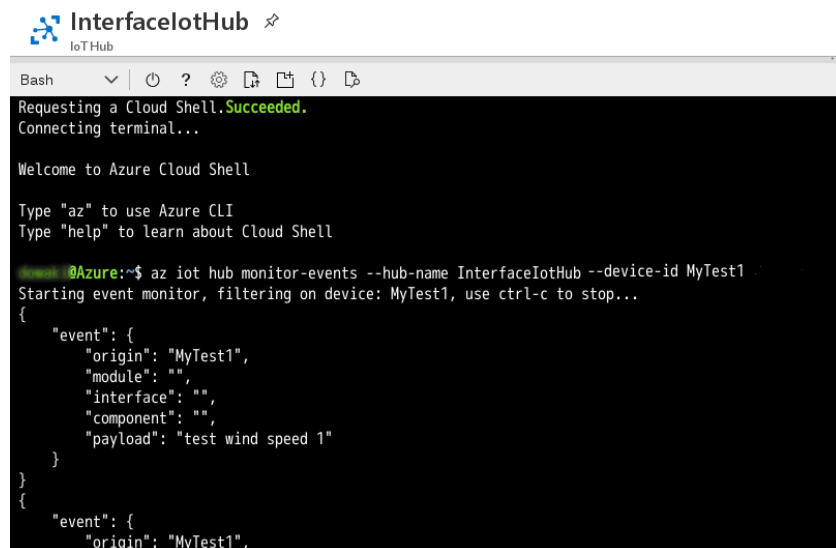
青文字部分の内、「<リソース名>」と「<デバイス名>」は、「2.2 Azure IoT Hubを使うための準備作業(P. 6)」で説明した、「リソース名」と「デバイス名」を指定します。

```
$ az iot hub monitor-events --hub-name <リソース名> --device-id <デバイス名>
Starting event monitor, use ctrl-c to stop...
```

その後、端末に戻り、サンプルプログラムを動かします。

すると、「send_message.py」サンプルが、Azure IoT Hubに対して送信したメッセージが、「Cloud Shell」に出力&表示されます。

実際に動いている様子を、下図に示します。



メッセージの送信を試してみたので、今度は Azure IoT Hubからメッセージを送信して、端末で受信してみましょう。

「recv_message.py」サンプルは、Azure IoT Hubから送信したメッセージを受信するサンプルです。これを動かしてみましょう。

以下の青文字部分のコマンドを入力して、動かしてみます。

「Press Q to quit」と表示されます。

```
$ cd azure-iot-sdk-python/azure-iot-device/samples/sync-samples/
$ python3 recv_message.py
Press Q to quit
```

次に、Azure IoT Hubページの「IoTデバイス」メニューから、通信したい「デバイス」を選択し、上部に「デバイスへのメッセージ」があるので、クリックします。



すると、「デバイスへのメッセージ」メニューに切り替わるので、メッセージの本文欄に、任意の文字列を入力した後、「メッセージの送信」をクリックします。



すると、「recv_message.py」サンプルの表示が変化し、送信したメッセージが表示されます。

```
$ cd azure-iot-sdk-python/azure-iot-device/samples/sync-samples/  
$ python3 recv_message.py  
Press Q to quit  
the data in the message received was  
b'hello Azure'  
custom properties are  
{}  
the data in the message received was  
b'hello Azure'  
custom properties are  
{}
```

このように、Azure SDKを用いると、比較的簡単に Azure IoT Hubとの通信が可能です。

2.4 AJANからPython連携を介してAzure IoT Hubにアクセスする

先の説明では、Python言語向けのAzure SDKのサンプルを用いて、Azure IoT Hubにアクセスしていました。

AJANは、Python連携機能を用いて、Python言語で書かれたプログラムコードを呼び出す事ができます。

ここでは、AJANからPython連携機能を用いて、Azure IoT Hubにアクセスしてみましょう。

「2.3 Azure SDKのサンプルを呼び出してみる(P.8)」で、「send_message.py」サンプルを紹介しました。

このサンプルと同じように、メッセージを送信するAJANのプログラムを、以下に示します。

(下記のサンプルが「AZURE_IOTHUB_SEND.AJN」に用意しています。詳しくは『第3章サンプルプログラム』を参照してください。)

```
' Azure IoT Hub へのメッセージ送信プログラム

s$ = '''
# Azure IoT Hub に対して、メッセージ送信する為の Python モジュールコード
import sys
import os
import time
import uuid
from azure.iot.device import IoTHubDeviceClient, Message

device_client = None

def open(id):
    # IoT Hub に対する通信路を開きます。id は、プライマリ接続文字列を指定します
    global device_client
    #conn_str = os.getenv("IOTHUB_DEVICE_CONNECTION_STRING")
    conn_str = id
    assert len(conn_str) > 0, "接続文字列が空です"
    # The client object is used to interact with your Azure IoT hub.
    device_client = IoTHubDeviceClient.create_from_connection_string(conn_str)

    # Connect the client.
    device_client.connect()
    return 0

def send(i_msg):
    # IoT Hub に対して、メッセージを送信します
    global device_client
    msg = Message(i_msg)
    msg.message_id = uuid.uuid4()
    #msg.correlation_id = "correlation-1234"
    msg.custom_properties["tornado-warning"] = "yes"
    device_client.send_message(msg)
    return 0
```

```

def close():
    # IoT Hub に対する、通信路を閉じます。
    global device_client
    device_client.disconnect()
    return 0
,,,

' Azure IoT Hub にアクセスする、Python モジュールコードを読み込みます
id = PYOBJ CREATE CODE(s$)
PRINT "pyobj create:"; id

PRINT "Azure IoT Hub への通信路を開きます"
conn_id$ = ENVIRON$("IOTHUB_DEVICE_CONNECTION_STRING")
? PYOBJ CALL FUNCTION(id, "open", conn_id$)

PRINT "メッセージを送信します"
? PYOBJ CALL FUNCTION(id, "send", "hello AJAN")

PRINT "Azure IoT Hub の通信路を閉じます"
? PYOBJ CALL FUNCTION(id, "close")

' Python 連携を閉じます
PYOBJ CLOSE id

END

```

このプログラムコードを、コンパイルして動かすと「hello AJAN」という文字列が、Azure IoT Hub に対して送信されます。

上記プログラムコードは、以下の3つのルーチンで構成されています。

- openルーチン。Azure IoT Hubにプライマリ接続文字列を使って接続します
- sendルーチン。文字列引数を、Azure IoT Hubにメッセージ送信します。
- closeルーチン。Azure IoT Hubとの接続を閉じます。

次は、「recv_message.py」サンプルのような、Azure IoT Hubから送信されたメッセージを、受信するサンプルを以下に示します。

(下記のサンプルが「AZURE_IOTHUB_RECV.AJN」に用意しています。詳しくは『第3章サンプルプログラム』を参照してください。)

```
' Azure IoT Hub からの、メッセージ受信プログラム

s$ = '''
# Azure IoT Hub に対して、メッセージ受信する為の Python モジュールコード
import sys
import os
from six.moves import input
import threading
from azure.iot.device import IoTHubDeviceClient
import queue

device_client = None

g_queue = queue.Queue() # 受信キュー

# define behavior for receiving a message
def message_handler(message):
    global g_queue
    print("""the data in the message received was """)
    print(message.data)
    print("""custom properties are""")
    print(message.custom_properties)
    # 受信メッセージのデータ部を、受信キューに追加します
    g_queue.put(message.data)

def open(id):
    # IoT Hub に対する通信路を開きます。id は、プライマリ接続文字列を指定します
    global device_client
    #conn_str = os.getenv("""IOTHUB_DEVICE_CONNECTION_STRING""")
    conn_str = id
    assert len(conn_str) > 0, """接続文字列が空です"""
    # The client object is used to interact with your Azure IoT hub.
    device_client = IoTHubDeviceClient.create_from_connection_string(conn_str)

    # Connect the client.
    device_client.connect()

    # set the message handler on the client
    device_client.on_message_received = message_handler
    return 0

def recvsize():
    # 受信キューのサイズを得ます
    global g_queue
    return g_queue.qsize()

def recvdata():
```



```

    # 受信キューから1個取り出します
    global g_queue
    if g_queue.empty():
        return ""
    else:
        it = g_queue.get()
        return it

def close():
    # IoT Hub に対する、通信路を閉じます。
    global device_client
    device_client.disconnect()
    return 0
'''

' Azure IoT Hub にアクセスする、Python モジュールコードを読み込みます
id = PYOBJ CREATE CODE(s$)
PRINT "pyobj create:"; id

PRINT "Azure IoT Hub への通信路を開きます"
conn_id$ = ENVIRON$("IOTHUB_DEVICE_CONNECTION_STRING")
? PYOBJ CALL FUNCTION(id, "open", conn_id$)

PRINT "メッセージ受信を、10 秒間 待機します"
ST = CLOCK
DO WHILE (CLOCK - ST) < 10.0
    ' 受信キューのサイズを得ます
    SZ = PYOBJ CALL FUNCTION(id, "recvsize")
    ? SZ
    ' 受信キューが空じゃないなら、受信データを取り出して表示します
    IF SZ > 0 THEN
        PRINT PYOBJ CALL FUNCTION(id, "recvdata")
    END IF
    SLEEP 0.5
LOOP

PRINT "Azure IoT Hub の通信路を閉じます"
? PYOBJ CALL FUNCTION(id, "close")

' Python 連携を閉じます
PYOBJ CLOSE id

END

```

このプログラムコードを、コンパイルして動かすと 10秒程度 Azure IoT Hubからのメッセージを受信待ちします。受信すると、そのメッセージ内容を表示します。

上記プログラムコードは、以下の4つのルーチンで構成されています。

- openルーチン。Azure IoT Hubにプライマリ接続文字列を使って接続します
- openルーチン登録時、Azure IoT Hubから送信されたメッセージを受信した際に呼ばれるコールバックルーチン(message_handler)を登録しています
コールバックルーチン内では、受け取ったメッセージを、受信キューに追加します。
- recvsizeルーチン。Azure IoT Hubから受け取ったメッセージを溜めている受信キューのサイズを得ます。
- recvdataルーチン。受信キューから1つデータを取得して返します。無ければ空文字列を返します。
- closeルーチン。Azure IoT Hubとの接続を閉じます。

このように、Python連携機能を用いて、AJANでAzure IoT Hubとの送受信が可能となります。

第3章 サンプルプログラム

AJANのサンプルプログラムについて記載します。

サンプルプログラムは「/usr/share/interface/AJANPro/samples/TUT/」に格納されています。

AJAN統合開発環境を起動すると、左ペインのエクスプローラウィンドウ内の「Samples/TUT/」に、ファイルが取り込まれて配置されます。

3.1 サンプルプログラム

#	ファイル名	内容
TUT/		
1	AZURE_IOTHUB_SEND.AJN	Azure IoT Hub に対して、メッセージを送信するサンプルプログラムです。 内部でPython 言語連携を用いて、Azure SDK を呼び出して、メッセージを送信します。
2	AZURE_IOTHUB_RECV.AJN	Azure IoT Hub から、メッセージを受信するサンプルプログラムです。 内部でPython 言語連携を用いて、Azure SDK を呼び出して、メッセージを受信します。

第4章 重要な情報

保証の内容と制限

弊社は本ドキュメントに含まれるソースプログラムの実行が中断しないこと、またはその実行に誤りが無いことを保証していません。

本製品の品質や使用に起因する、性能に起因するいかなるリスクも使用者が負うものとします。

弊社はドキュメント内の情報の正確さに万全を期しています。万一、誤記または誤植などがあつた場合、弊社は予告無く改訂する場合があります。ドキュメントまたはドキュメント内の情報に起因するいかなる損害に対しても弊社は責任を負いません。

ドキュメント内の図や表は説明のためであり、ユーザ個別の応用事例により変化する場合があります。

著作権、知的所有権

弊社は本製品に含まれるおよび本製品に対する権利や知的所有権を保持しています。

本製品はコンピュータ ソフトウェア、映像/音声(例えば図、文章、写真など)を含んでいます。

医療機器/器具への適用における注意

弊社の製品は人命に関わるような状況下で使用される機器に用いられる事を目的として設計、製造された物では有りません。

弊社の製品は人体の検査などに使用するに適する信頼性を確保する事を意図された部品や検査機器と共に設計された物では有りません。

医療機器、治療器具などの本製品の適用により、製品の故障、ユーザ、設計者の過失などにより、損傷/損害を引き起こす場合が有ります。

複製の禁止

弊社の許可なく、本ドキュメントの全て、または一部に関わらず、複製、改変などを行うことはできません。

責任の制限

弊社は、弊社または再販売者の予見の有無にかかわらず発生したいかなる特別損害、偶発的損害、間接的な損害、重大な損害について、責任を負いません。

本製品(ハードウェア、ソフトウェア)のシステム組み込み、使用、ならびに本製品から得られる結果に関する一切のリスクについては、本製品の使用者に帰属するものとします。

本製品に含まれる不都合、あるいは本製品の供給(納期遅延)、性能もしくは使用に起因する付帯的損害もしくは間接的損害に対して、弊社に全面的に責がある場合でも、弊社はその製品に対する改良(有償サービスの利用)、代品交換までとし、製品の予防交換並びに、代金減額等、金銭面での賠償の責任は負わないものとします。

本製品は、日本国内仕様です。

商標/登録商標

本書に掲載されている会社名、製品名は、それぞれ各社の商標または登録商標です。

改訂履歴

Ver.	年 月	改 訂 内 容
0.91	2021年6月	新規作成

このマニュアルは、製品の改良その他により将来予告なく改訂しますので、予めご了承ください。