Connecting a Delta VFD to Microsoft Azure using ICONICS IoTWorX

Contents

Copyright and Confidentiality

Authors

- Spyros Sakellariadis, Applied Innovation/Enterprise Commercial Business, Microsoft Corporation
- Maksym Mushkin, Applied Innovation/Enterprise Commercial Business, Microsoft Corporation
- Zhi Wei Li, Director of Innovation & Incubation Solutions, ICONICS
- Gamal Mustapha, Product Manager, Delta Controls Inc.
- Sean Cowan, Delta Systems Specialist, Delta Controls Inc.

1 Introduction

Remotely monitoring and managing buildings has many benefits, from reducing operational costs to improving the experience of the tenants in the building. It is not only valuable, in many cases essential, for example when the premises are distant and there are no local resources to address problems or optimize configurations. However, building/controls engineers often lack the knowledge of how to monitor their equipment remotely, and IT/ICT professionals are often equally in the dark about how to connect to building equipment.

In this context, this documentation describes one example of how to connect building systems to a centralized monitoring and control system, often referred to as a Smart Building system. The example here involves monitoring and controlling a motor in a building from the Microsoft Azure cloud, using a Variable Frequency Drive (VFD) and a software gateway installed locally.

VFDs are one of the best energy management tools that can be applied to a building. They can be used to optimize load conditions during non-peak periods and savings can be realized based on the fact that the energy of an induction motor is a function of speed such that a 25 percent reduction in speed would equate to a 60 percent reduction in energy use. Remote monitoring and management of VFDs make this a very appealing application for any Smart Building system.

2 Infrastructure

2.1 Hardware

In the setup described in this paper, the physical device being managed is an Orientalmotor 25W induction motor that is connected to a Delta Electronics CP2000 VFD with a Delta Controls eZVFD BACnet card inserted into an internal expansion slot. The eZVFD card supports external communication over both BACnet/IP and BACnet Ethernet, and in the setup described here the eZVFD talks to a Dell Edge Gateway 5000 over a local Cat6 network. The physical configuration is shown in Figure 1:

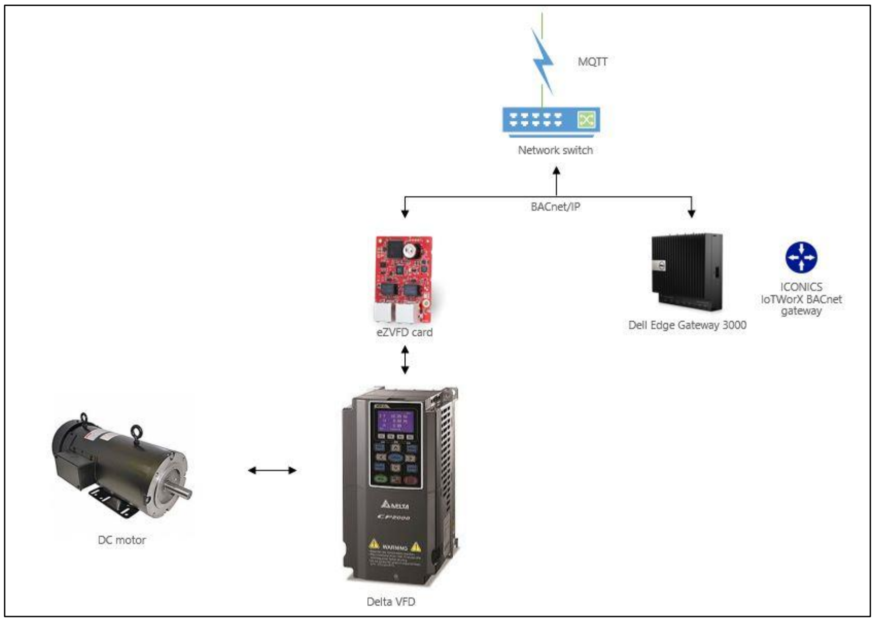


Figure 1 Physical Layout

The eZVFD samples over 700 parameters within the VFD in less than 100ms and can run real-time programs locally to control attached devices in standalone applications and in the event of a communication failure. In this paper, however, we will focus on a networked system, and describe how to use a cloud-based service to monitor and control the attached devices.

2.2 Network

There are many protocols in the building automation and manufacturing industries used to connect and control devices such as motors, fans, and actuators. BACnet is probably the most commonly used protocol in the building space, developed by ASHRAE to be a standard communication protocol between systems from various vendors, providing standardized methods for presenting, requesting, interpreting, and transporting information. In the setup described in this paper we have a gateway application running on a local computer that requests information from a building system over BACnet/IP, and then sends the data over the Internet using the MQTT protocol, to a cloud application for analysis and reporting. Conversely, the cloud application can send commands back to the building system via the on-premises gateway application.

2.3 On-premises Software

In this configuration we are using the ICONICS IoTWorX BACnet gateway to read and write values in the eZVFD. Specifically, we will configure IoTWorX to perform the following functions:

1. Connecting to the eZVFD
2. Requesting values of certain objects on the eZVFD every 5 minutes
3. Reformatting the data into a prescribed format
4. Transmitting that data to Azure IoT Hub

2.4 *Cloud services*

After the data arrives in Azure IoT Hub, an Azure Stream Analytics job reads the incoming data stream and writes it into a SQL database, and a Power BI report displays the current and historical values. Finally, a small application is used to write values back to Azure IoT Hub, for sending back down to IoTWorX and then on to the VFD. The overall data flow is shown in Figure 2:

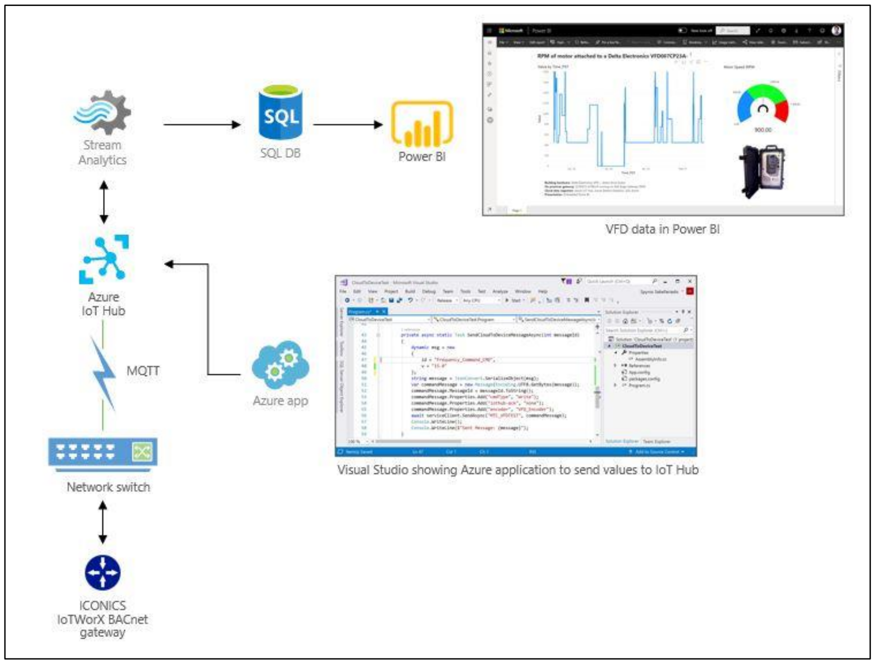

Stream Analytics
SQL DB
Power BI
VFD data in Power BI
Azure IoT Hub
MQTT
Azure app
Visual Studio showing Azure application to send values to IoT Hub
Network switch
ICONICS IoTWorX BACnet gateway

Figure 2 Software components

The following sections contain a description of how to configure the IoTWorX gateway and the Azure components to monitor and manage the motor attached to the VFD.

3 Installing IoTWorX to read from the VFD

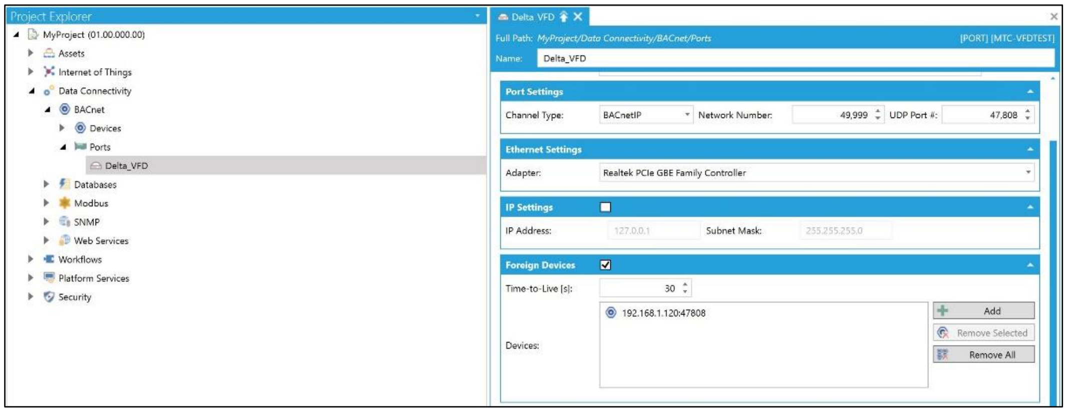
Installing ICONICS IoTWorX to read from BACnet devices is covered in detail in a document published on the ICONICS website here: https://iconics.com/Documents/WhitePapers/Using-IoTWorX-as-a-Gateway (below referred to as *Using IoTWorX*). Follow the instructions in that document for using ICONICS Workbench, inserting the values shown below instead of those shown in *Using IoTWorX*.

3.1 *Specify where devices are to be found*

Use the values below when following instructions in *Using IoTWorX,* Section 3.1:

Section	Variable	Suggested value
3.1.5	BBMD port name	*Delta_VFD*
3.1.8	Channel Type	*BACnetIP*
3.1.8	Network Number	*49,999* To verify this is what the VFD is using, use the keypad of the VFD to display the BACnet network number: • Press Menu • Select 1. Pr Setup, press Enter • Scroll down to 04, press Enter • Scroll down to 51, press Enter In the Delta documentation this is referred to as parameter 04-51, and here we will use the shorthand • 04-51: 49,999 To indicate that 04-51 should be 49,999.
3.1.8	Foreign Devices IP port	*w.x.y.z*:47808 By default you would expect the VFD to use BACnet/IP and get an IP address by DHCP. To verify that the VFD is using BACnet/IP, check or select the following parameters using the keypad as shown above. • 09-91: 1 To verify that the VFD is getting an IP address from a DHCP server, check or select • 09-75: 1 To get the current DHCP-assigned IP address of the VFD, get the numbers from • 09-76: *w* (1st byte) • 09-77: *x* (2nd byte) • 09-78: *y* (3rd byte) • 09-79: *z* (4th byte) Finally, to get the port number used by the VFD, check • 04-50

When this step is complete, the configuration in ICONICS Workbench should look similar to that shown in Figure 3:



Full Path: *MyProject/Data Connectivity/BACnet/Ports* [PORT] [MTC-VFDTEST]
Name: Delta_VFD

Port Settings
Channel Type: BACnetIP Network Number: 49,999 UDP Port #: 47,808

Ethernet Settings
Adapter: Realtek PCIe GBE Family Controller

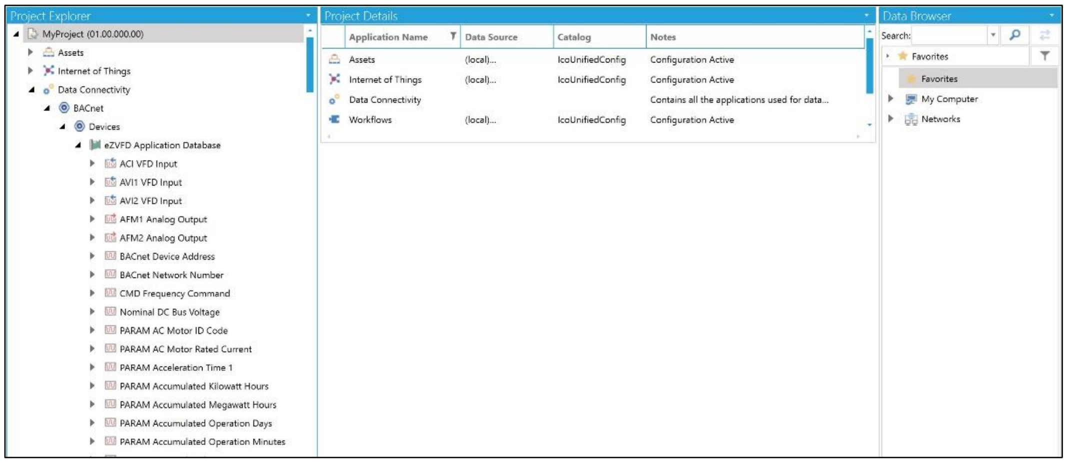
IP Settings ☐
IP Address: 127.0.0.1 Subnet Mask: 255.255.255.0

Foreign Devices ☑
Time-to-Live [s]: 30
Devices: 192.168.1.120:47808

Figure 3 Port connecting to VFD

3.2 Discover devices

To identify the data points on the VFD, ignore the instructions in *Using IoTWorX,* Section 3.2.1, Add a device manually. Instead, follow the instructions in *Using IoTWorX,* Section 3.2.2, Add multiple devices through a network scan. That section covers running a network scan to discover objects in or attached to the VFD. To reduce the time taken for a network scan, in *Using IoTWorX,* Section 3.2.2.4, enter a range starting from the value shown on the VFD keypad at 09-52, for example 47874 – 57874. When this step is complete, your configuration in Workbench should look similar to that shown in Figure 4:



Project Explorer

- MyProject (01.00.000.00)
  - Assets
  - Internet of Things
  - Data Connectivity
    - BACnet
      - Devices
        - eZVFD Application Database
          - ACI VFD Input
          - AVI1 VFD Input
          - AVI2 VFD Input
          - AFM1 Analog Output
          - AFM2 Analog Output
          - BACnet Device Address
          - BACnet Network Number
          - CMD Frequency Command
          - Nominal DC Bus Voltage
          - PARAM AC Motor ID Code
          - PARAM AC Motor Rated Current
          - PARAM Acceleration Time 1
          - PARAM Accumulated Kilowatt Hours
          - PARAM Accumulated Megawatt Hours
          - PARAM Accumulated Operation Days
          - PARAM Accumulated Operation Minutes

Project Details

Application Name	Data Source	Catalog	Notes
Assets	(local)...	IcoUnifiedConfig	Configuration Active
Internet of Things	(local)...	IcoUnifiedConfig	Configuration Active
Data Connectivity			Contains all the applications used for data...
Workflows	(local)...	IcoUnifiedConfig	Configuration Active

Data Browser

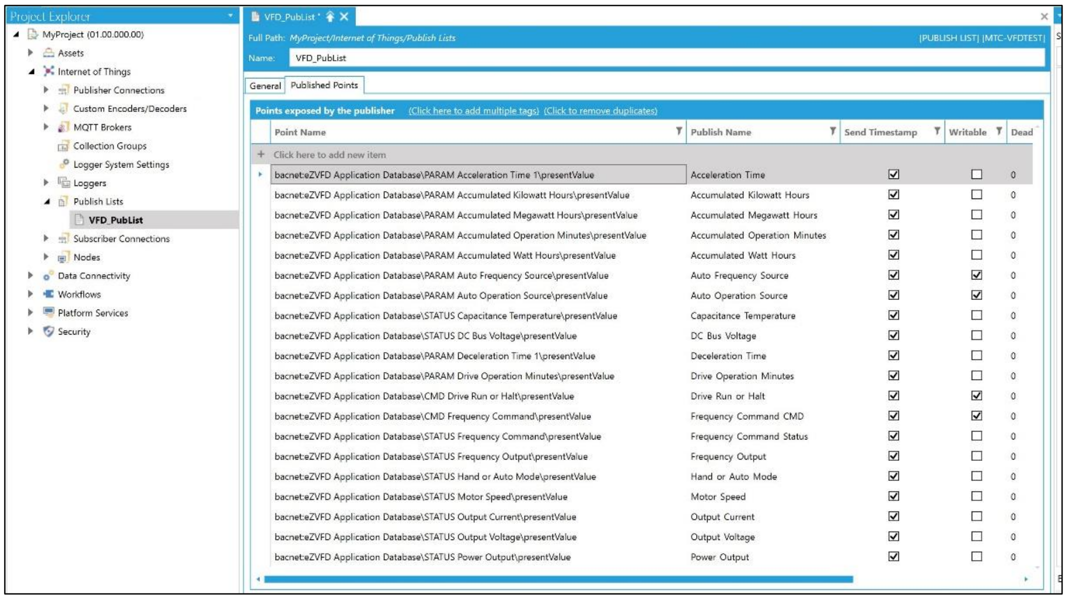
Figure 4 Devices and objects discovered on VFD

3.3 Create a publish list

For this test installation, enter “VFD_PubList” for the name of the Publish List. Select the following points as a starter set:

CMD Drive Run or Halt\presentValue
CMD Frequency Command\presentValue
PARAM Acceleration Time 1\presentValue
PARAM Accumulated Kilowatt Hours\presentValue
PARAM Accumulated Kilowatt Hours\presentValue
PARAM Accumulated Operation Minutes\presentValue
PARAM Accumulated Operation Minutes\presentValue
PARAM Auto Frequency Source\presentValue
PARAM Auto Operation Source\presentValue
PARAM Deceleration Time 1\presentValue
PARAM Drive Operation Minutes\presentValue
STATUS Capacitance Temperature\presentValue
STATUS DC Bus Voltage\presentValue
STATUS Frequency Command\presentValue
STATUS Frequency Output\presentValue
STATUS Hand or Auto Mode\presentValue
STATUS Motor Speed\presentValue
STATUS Output Current\presentValue
STATUS Output Voltage\presentValue
STATUS Power Output\presentValue

When this step is complete, your configuration in ICONICS Workbench should look similar to that shown in Figure 5:



Full Path: *MyProject/Internet of Things/Publish Lists* |PUBLISH LIST| |MTC-VFDTEST|

Name: VFD_PubList

Points exposed by the publisher

Point Name	Publish Name	Send Timestamp	Writable	Dead
bacnet:eZVFD Application Database\PARAM Acceleration Time 1\presentValue	Acceleration Time	✓	☐	0
bacnet:eZVFD Application Database\PARAM Accumulated Kilowatt Hours\presentValue	Accumulated Kilowatt Hours	✓	☐	0
bacnet:eZVFD Application Database\PARAM Accumulated Megawatt Hours\presentValue	Accumulated Megawatt Hours	✓	☐	0
bacnet:eZVFD Application Database\PARAM Accumulated Operation Minutes\presentValue	Accumulated Operation Minutes	✓	☐	0
bacnet:eZVFD Application Database\PARAM Accumulated Watt Hours\presentValue	Accumulated Watt Hours	✓	☐	0
bacnet:eZVFD Application Database\PARAM Auto Frequency Source\presentValue	Auto Frequency Source	✓	✓	0
bacnet:eZVFD Application Database\PARAM Auto Operation Source\presentValue	Auto Operation Source	✓	✓	0
bacnet:eZVFD Application Database\STATUS Capacitance Temperature\presentValue	Capacitance Temperature	✓	☐	0
bacnet:eZVFD Application Database\STATUS DC Bus Voltage\presentValue	DC Bus Voltage	✓	☐	0
bacnet:eZVFD Application Database\PARAM Deceleration Time 1\presentValue	Deceleration Time	✓	☐	0
bacnet:eZVFD Application Database\PARAM Drive Operation Minutes\presentValue	Drive Operation Minutes	✓	☐	0
bacnet:eZVFD Application Database\CMD Drive Run or Halt\presentValue	Drive Run or Halt	✓	✓	0
bacnet:eZVFD Application Database\CMD Frequency Command\presentValue	Frequency Command CMD	✓	✓	0
bacnet:eZVFD Application Database\STATUS Frequency Command\presentValue	Frequency Command Status	✓	☐	0
bacnet:eZVFD Application Database\STATUS Frequency Output\presentValue	Frequency Output	✓	☐	0
bacnet:eZVFD Application Database\STATUS Hand or Auto Mode\presentValue	Hand or Auto Mode	✓	☐	0
bacnet:eZVFD Application Database\STATUS Motor Speed\presentValue	Motor Speed	✓	☐	0
bacnet:eZVFD Application Database\STATUS Output Current\presentValue	Output Current	✓	☐	0
bacnet:eZVFD Application Database\STATUS Output Voltage\presentValue	Output Voltage	✓	☐	0
bacnet:eZVFD Application Database\STATUS Power Output\presentValue	Power Output	✓	☐	0

Figure 5 Publish List with selected objects of interest

It is useful to enter a Publish Name manually in the Publish Name column, as that will make it easier to parse the data in Azure. Also note that some of the fields should be marked as writable so you can control the VFD remotely. In the figure above, the following parameters are checked as writable, so the speed of the motor can be changed remotely:

```
PARAM Auto Frequency Source\presentValue
PARAM Auto Operation Source\presentValue
CMD Drive Run or Halt\presentValue
CMD Frequency Command\presentValue
```

3.4 Create a custom encoder

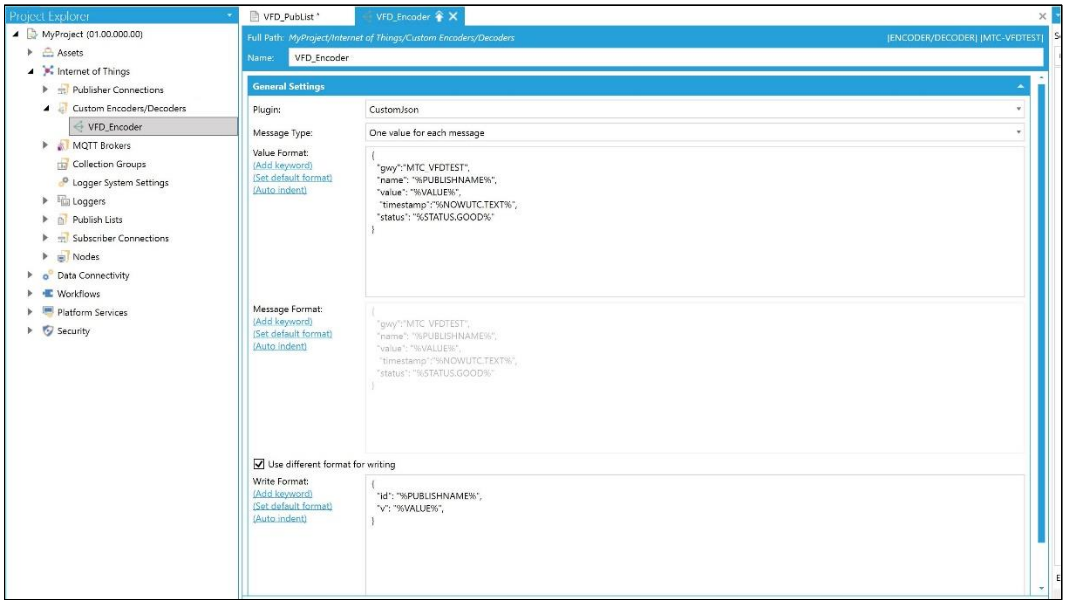
For this test installation, enter “VFD_Encoder” for the name of the Custom Encoder. In the General Settings section, select “One value for each message” for the Message Type. In the Value Format enter:

```
{
  "gwy": "%DEVICENAME%",
  "name": "%PUBLISHNAME%",
  "value": "%VALUE%",
   "timestamp":"%NOWUTC.TEXT%",
  "status": "%STATUS.GOOD%"
}
```

Check the box labelled Use different format for writing, and enter the following:

```
{
  "id": "%PUBLISHNAME%",
  "v": "%VALUE%",
}
```

When this step is complete, your configuration in Workbench should look similar to that shown in Figure 6:



Full Path: MyProject/Internet of Things/Custom Encoders/Decoders [ENCODER/DECODER] [MTC-VFDTEST]

Name: VFD_Encoder

Plugin: CustomJson

Message Type: One value for each message

Value Format:

```
{
  "gwy":"MTC_VFDTEST",
  "name": "%PUBLISHNAME%",
  "value": "%VALUE%",
   "timestamp":"%NOWUTC.TEXT%",
  "status": "%STATUS.GOOD%"
}
```

Message Format:

```
{
  "gwy":"MTC_VFDTEST",
  "name": "%PUBLISHNAME%",
  "value": "%VALUE%",
   "timestamp":"%NOWUTC.TEXT%",
  "status": "%STATUS.GOOD%"
}
```

[✓] Use different format for writing

Write Format:

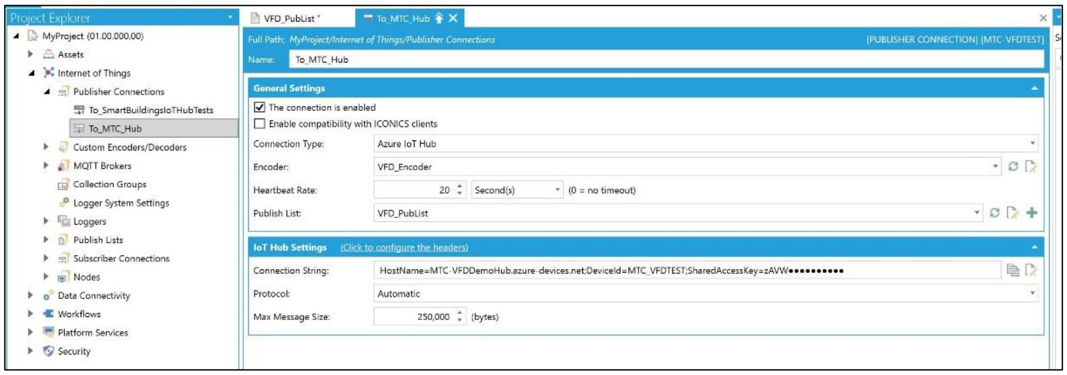
```
{
  "id": "%PUBLISHNAME%",
  "v": "%VALUE%",
}
```

Figure 6 Custom encoder

3.5 Create a publisher connection

For this test installation, enter “To_IoTHub” for the name of the Publisher Connection. Enter VFD_Encoder for the Encoder, VFD_PubList for the Publish List, and the Primary Connection String for the IoT Device in the IoT Hub that was created to receive the messages from the VFD.

When this step is complete, your configuration in Workbench should look similar to that shown in Figure 7:



Full Path: MyProject/Internet of Things/Publisher Connections [PUBLISHER CONNECTION] [MTC-VFDTEST]

Name: To_MTC_Hub

[✓] The connection is enabled

[ ] Enable compatibility with ICONICS clients

Connection Type: Azure IoT Hub

Encoder: VFD_Encoder

Heartbeat Rate: 20 Second(s) (0 = no timeout)

Publish List: VFD_PubList

Connection String: HostName=MTC-VFDDemoHub.azure-devices.net;DeviceId=MTC_VFDTEST;SharedAccessKey=zAVW••••••••••

Protocol: Automatic

Max Message Size: 250,000 (bytes)

Figure 7 Publisher Connection

After you create and save the Publisher Connection, click the button in the top menu bar to start or restart the Publisher Service. At this point, data should start flowing to Azure IoT Hub, which you can confirm by viewing the data being received at the hub. Follow the instructions in *Using IoTWorX,* Section 4, Viewing the data in Azure IoT Hub, to do this.

3.6 Modify an IoTWorX configuration file

We need to set the following INI setting to instruct the IoTWorX BACnet runtime to allow ‘out of service points’ to be treated as readable and writable.

1. Go to C:\ProgramData\ICONICS
2. Open IcoSetup64.ini
3. Paste this at the end of the file:
   ```
   [BACNET\Compatibility]
   TreatBadStatusFlagsAsGoodQuality=1
   ```
4. Save the file
5. Restart the BACnet Runtime

Depending upon how the hardware was configured, the Delta VFD may flag the parameter `Frequency_Command_CMD as` “out of service”. In the case of the code below, we do not check this flag, but some programs (such as ICONICS AnalytiX) interpret the out of service state to indicate a ‘locked’ point, such that the value is not being updated by the sensor but ‘locked’ to a certain user defined value. To accommodate those programs, we need to change the above INI setting so those programs can still interact with the `Frequency_Command_CMD` point.

4 Creating a Power BI application to display the incoming data

In addition to viewing the data as described in *Using IoTWorX,* Section 4, you can build an Azure dashboard to display the data in real time. It is beyond the scope of this paper to describe this in detail, but the basic steps on one way to do this are as described below. You may need assistance from an IT/ICT professional familiar with SQL and Power BI to complete these steps.

4.1 Create a SQL database to store the data

First, using SQL Server Management Studio create or connect to a SQL Server in the same Azure subscription as the Azure IoT Hub to which IoTWorX is sending data. In that database, create a table to store the data from the VFD. In the setup described in this paper, the table EventHistorian was created with the following script:

```
CREATE TABLE [dbo].[EventHistorian](
       [Name] [varchar](max) NULL,
       [TimeStamp] [datetime] NULL,
       [Value] [float] NULL
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
```

This table will hold data indefinitely, allowing us to get both the current status of the parameters the gateway is polling, and to chart historical trends.

4.2 *Create an Azure Stream Analytics job to send data to SQL*

Second, create a function to read the data coming in to Azure IoT Hub and send it to the SQL table created above, for example with an Azure Stream Analytics job. Using the Azure portal create a job where the input is the Azure IoT Hub, the output the SQL table above, and a query that transforms the structure of the incoming data into that specified for the SQL table. In the setup here, the structure of query was essentially as follows:

```
SELECT
      Name AS [Name],
      Value AS [Value],
      Timestamp AS [TimeStamp]
INTO
    [VFDOutput]
FROM
    [VFDInput]
```

However, this query won't work exactly as written above, due to the format of the incoming data, and you will need to modify each of the SELECT lines a bit. For example, the incoming Timestamp is formatted by the gateway as

```
"timestamp":"%NOWUTC.TEXT%"
```

It arrives at IoT Hub in a string format, for example

```
"timestamp":"2020-04-27T12:00:29.930Z",
```

and to write it to the SQL table it needs to be sent in the datetime format. To do this, you need to parse the timestamp string into parts and then concatenate those parts into the DateTime format. In the SELECT statement above, you can do this by replacing the line

```
Timestamp AS [TimeStamp]
```

With the following lines:

```
CAST(DATETIMEFROMPARTS(
      DATEPART(year,[Timestamp]),
      DATEPART(month,[Timestamp]),
      DATEPART(day,[Timestamp]),
      DATEPART(hour,[Timestamp]),
      DATEPART(minute,[Timestamp]),
      DATEPART(second,[Timestamp]),
      00) AS DateTime)
AS [TimeStamp]
```

In addition, you will need logic to parse the incoming name data. Depending upon how the eZVFD and VFD was set up, the name data might arrive in the following format:

"name":"bacnet:eZVFD Application Database\\PARAM Drive Operation Days\\presentValue", or
"name":"bacnet:eZVFD Application Database\\STATUS Drive Direction\\presentValue"

In this case, you will need to use various string functions to extract the name of the tag "Drive Operation Days" or "Drive Direction" before writing to SQL. An example of extracting the parameter name would be to use the following logic:

```
SUBSTRING(
	name,
	CHARINDEX('PARAM',name) + 6,
	(CHARINDEX('presentValue',name)-1)-(CHARINDEX('PARAM',name) + 6)
	)
AS [Name]
```

This statement will work for the first of the two strings above, but not the second, or any other structure that is coming in to IoTHub. Consequently, you will need to build a CASE statement with multiple variations of the above to Again, depending upon how the drive is set up, you will need a CASE statement for each of the types of strings you would see at IoT Hub, each of which will be similar to the SUBSTRING statement above.

Finally, the value parameter arriving at IoT Hub is a string in the format:

"value":0.5

To write this to the SQL table defined above, you need to convert this to a float. For which you will use the CAST statement. The basic version of this is

```
TRY_CAST(value as float)
AS [Value]
```

However, you may find that the incoming string has one or more spaces in it, or is blank, or is a null, so you will again need a CASE statement for each of the variations.

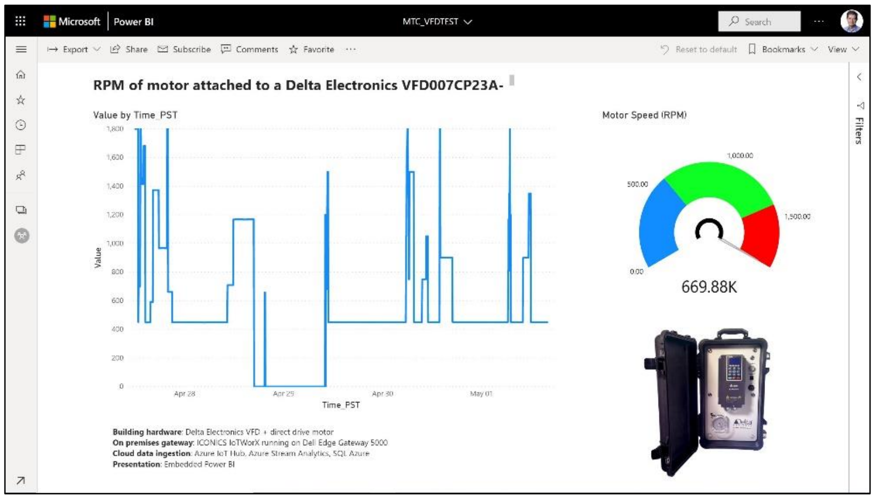
You can access the completed ASA job on GitHub, here.

4.3 Create a Power BI report

Once the data is being logged into a SQL table you can easily create a SQL query to list out all the data:

```
SELECT * FROM EventHistorian
```

However, it is useful to have a real-time dashboard with the data displayed in a chart. Figure 8 shows a Power BI report pulling the data from the SQL table:



RPM of motor attached to a Delta Electronics VFD007CP23A-

Building hardware: Delta Electronics VFD + direct drive motor
On premises gateway: ICONICS IoTWorX running on Dell Edge Gateway 5000
Cloud data ingestion: Azure IoT Hub, Azure Stream Analytics, SQL Azure
Presentation: Embedded Power BI

Figure 8 PowerBI.com report

To create such a dashboard you would use the free Power BI desktop application, and do the following:

1. Specify the connection string for the SQL database and table
2. Specify the query against the database
3. Specify the chart type (on the left in Figure 8 is a Line Chart, on the top right a third party Power BI widget downloaded from the store)
4. Specify the axes
5. Add any text or JPG
6. Publish the chart to http://powerbi.com
7. Share the workspace to authorized users

There are multiple tutorials on the Internet on setting up Power BI dashboards and reports, which you can find easily with an simple search.

5 Writing to the VFD from Azure using IoTWorX

Installing ICONICS IoTWorX to write to BACnet devices is covered in detail in a document published on the ICONICS website here: https://iconics.com/Documents/WhitePapers/MSFT-Writing-to-Devices-Using-IoTWorX (below referred to as *Writing to Devices*). All the instructions in that document are covered in the topics above, except for the design of the Azure application in *Writing to Devices,* Section 5, Creating an application to write to a device. Refer to the instructions in that document for building an Azure application.

On the VFD you need to make sure several parameters are set, so that the VFD can be controlled remotely. You can do this via the keypad on the VFD or using the Azure application. If you are using the keypad, set

00-20:9
00-21:6

If you are using an Azure application as described in *Writing to Devices,* Section 5, edit the dynamic msg lines to be as follows and run the application.

```
dynamic msg = new
{
        id = "Auto_Frequency_Source",
        v = "9"
};
```

`Then change those lines to be`

```
dynamic msg = new
{
        id = "Auto_Operation_Source",
        v = "6"
};
```

You only need to do this once. Then, to change the motor speed, edit those lines to the following and run the program:

```
dynamic msg = new
{
        id = " Frequency_Command_CMD",
        v = "30"
};
```

This will change the VFD output frequency to 30Hz, which will change the motor speed (RPM) proportionately. You may also need to give the VFD a new 'run' command (this is necessary when the control source of the VFD is changed; the VFD needs to see a transition from 'stop' to 'run'). Change the lines to the following to set the command to 'stop':

```
dynamic msg = new
{
        id = " Drive_Run_or_Halt",
        v = "1"                               /* "1" = stop, "2" = run */
};
```

Followed by these to set the command to 'run', which will start the drive output again:

```
dynamic msg = new
{
        id = " Drive_Run_or_Halt",
        v = "2"                               /* "1" = stop, "2" = run */
};
```

You can find access a listing the completed program on GitHub, here.

6 Using GENESIS64 as a no code client

The previous sections address ways to use custom code to display data from the Delta VFD and to write back values. It is also possible to do this without code, using third party solutions. For example, ICONICS GENESIS64 can be used as a no code client to the published VFD data. To do so, the following configuration needs to be setup on a VM with GENESIS64 installed. **Note**: despite a similarity to the instructions in the previous sections, the following instructions are not about using ICONICS Workbench on the gateway with IoTWorX installed, they are about using ICONICS Workbench on a virtual machine in Azure to read the data coming in from the gateway.

6.1 Create a custom encoder

To decode the published VFD data, we have to first setup the custom encoder that instructs GENESIS64 how to understand the published data.

To set this up, in the Workbench configuration tool, create a custom encoder like in section 3.4, name it as "VFD_Encoder" and define the Value Format as follows:

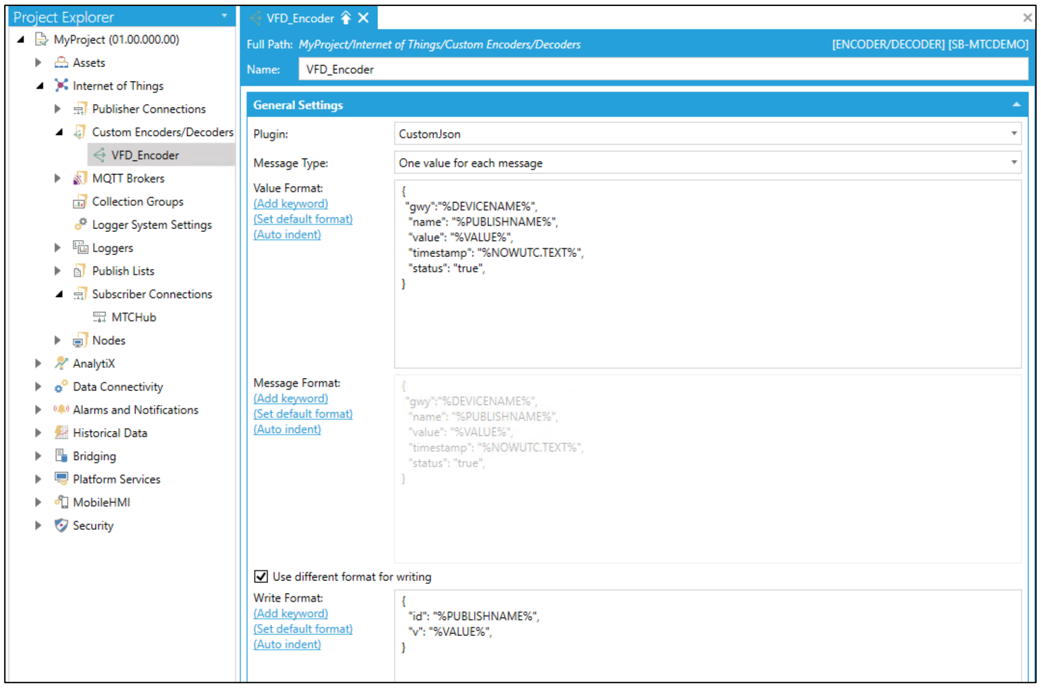
```
{
    "gwy": "%DEVICENAME%",
    "name": "%PUBLISHNAME%",
    "value": "%VALUE%",
    "timestamp":"%NOWUTC.TEXT%",
    "status": "%STATUS.GOOD%"
}
```

Check the box labelled Use different format for writing and enter the following:

```
{
    "id": "%PUBLISHNAME%",
    "v": "%VALUE%",
}
```

Click Apply to save the configuration.

When this step is complete, your configuration in Workbench should look similar to that shown in Figure 9:



Full Path: MyProject/Internet of Things/Custom Encoders/Decoders [ENCODER/DECODER] [SB-MTCDEMO]

Name: VFD_Encoder

General Settings

Plugin: CustomJson

Message Type: One value for each message

Value Format:

```
{
 "gwy":"%DEVICENAME%",
 "name": "%PUBLISHNAME%",
 "value": "%VALUE%",
 "timestamp": "%NOWUTC.TEXT%",
 "status": "true",
}
```

Message Format:

```
{
 "gwy":"%DEVICENAME%",
 "name": "%PUBLISHNAME%",
 "value": "%VALUE%",
 "timestamp": "%NOWUTC.TEXT%",
 "status": "true",
}
```

☑ Use different format for writing

Write Format:

```
{
 "id": "%PUBLISHNAME%",
 "v": "%VALUE%",
}
```

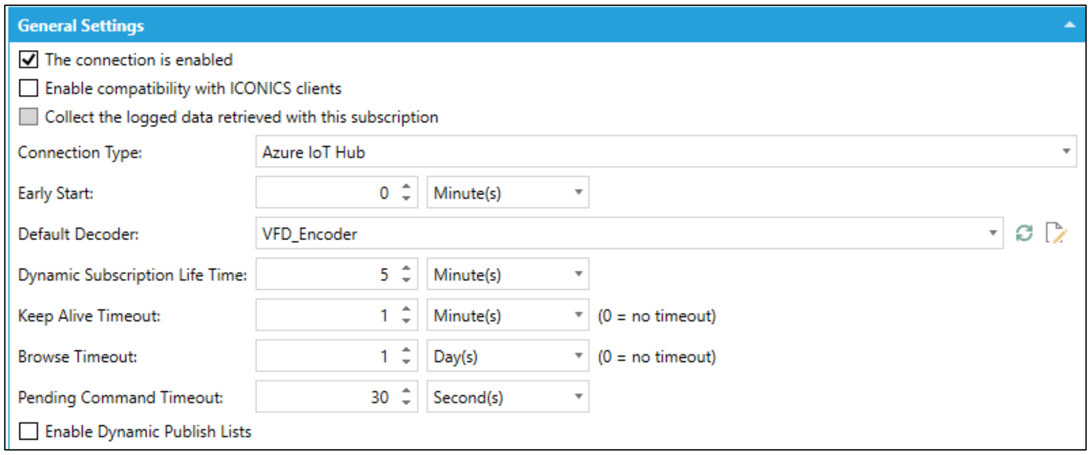
Figure 9 Custom encoder

6.2 Create a subscriber connection

To start receiving published data from the VFD, we have to first subscribe to the IoT Hub with a subscriber connection.

To set this up, in Workbench → Internet of Things, right click on Subscriber Connections, choose Add Subscriber Connection and give the subscriber connection a name, i.e. MTCHub.

Set up the general settings of the subscriber connection as shown in Figure 10:



General Settings

☑ The connection is enabled

☐ Enable compatibility with ICONICS clients

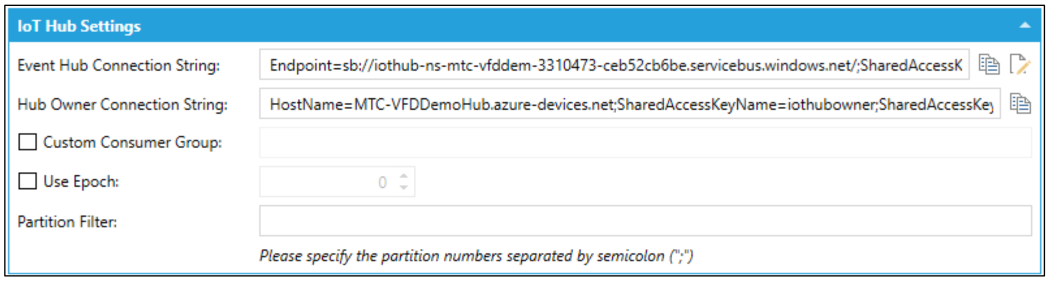
☐ Collect the logged data retrieved with this subscription

Connection Type:	Azure IoT Hub		
Early Start:	0	Minute(s)	
Default Decoder:	VFD_Encoder		
Dynamic Subscription Life Time:	5	Minute(s)	
Keep Alive Timeout:	1	Minute(s)	(0 = no timeout)
Browse Timeout:	1	Day(s)	(0 = no timeout)
Pending Command Timeout:	30	Second(s)	

☐ Enable Dynamic Publish Lists

Figure 10 Subscriber Connection General Settings

In the IoT Hub Settings section, enter the appropriate IoT Hub connection strings similar to that shown in Figure 11:



IoT Hub Settings

Event Hub Connection String:	Endpoint=sb://iothub-ns-mtc-vfddem-3310473-ceb52cb6be.servicebus.windows.net/;SharedAccessK
Hub Owner Connection String:	HostName=MTC-VFDDemoHub.azure-devices.net;SharedAccessKeyName=iothubowner;SharedAccessKey
☐ Custom Consumer Group:	
☐ Use Epoch:	0
Partition Filter:	

Please specify the partition numbers separated by semicolon (";")

Figure 11 Subscriber Connection IoT Hub Settings

Click Apply to save the configuration and start the Subscriber service

6.3 *Visualize and interact with published data*

To visualize data from the subscribed IoT Hub, go to the gateway system's Start Menu, launch the Data Explorer application in the ICONICS Tools folder, and browse under the My Computer → Internet of Things branch. You should see a structure similar to that shown in Figure 12:

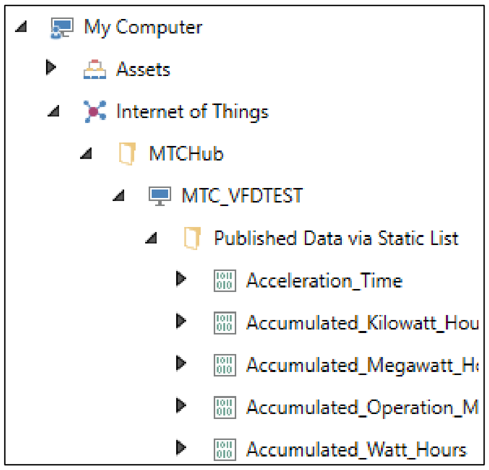
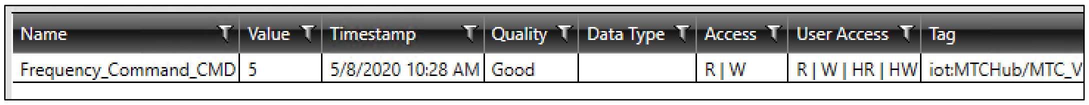


Figure 12 Browsing for published data

Select the desired published data point and you should see the data values in a table to the right similar to that shown in Figure 13:

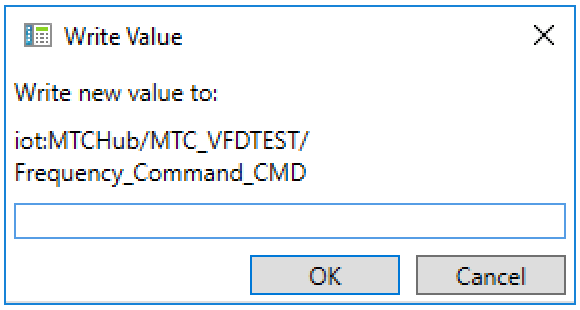


Name	Value	Timestamp	Quality	Data Type	Access	User Access	Tag
Frequency_Command_CMD	5	5/8/2020 10:28 AM	Good		R | W	R | W | HR | HW	iot:MTCHub/MTC_V

Figure 13 Published data values

The frequency of data updates and availability will be dependent upon the publish rate set on the IoTWorX gateway.

Double click on the Value cell to initiate a data write similar to that shown in Figure 14.



Write new value to:

iot:MTCHub/MTC_VFDTEST/
Frequency_Command_CMD

Figure 14 Writing to a published writable data point

7 Next steps

If you have successfully completed the above steps you have an end-to-end example of remotely monitoring and controlling a simple piece of equipment attached to a VFD. From here it is relatively straightforward to extend this to more complex scenarios – either more complex pieces of equipment or to build more sophisticated analytical tools in the web. The best way of learning is by doing, and we recommend trying out various configurations to build your expertise and comfort with these extremely valuable systems.