ABB ABILITY™ POINT OF VIEW

Making the Case for OT Device Drivers

Standardize at the edge to differentiate on value

- **A standardized device-driver approach allows device manufacturers to compete on features, services and price, instead of interoperability.**

Tim Diekmann
Head of Architecture and Standards, ABB Ability™

Why do we need device drivers for operational technology?

In a recent article in Forbes, ABB Chief Security Officer Satish Gannu made the case for a standards-based approach for connecting factory devices to the edge, and then the cloud. He reminded us of the times when we struggled to get the latest printer working with our computer due to a missing driver. As he suggests, the advent of a standard device driver for most common customer platforms has enabled device manufacturers to compete on features, services, and price instead of simple interoperability. The same could be true for devices in the industrial Internet of Things (IIoT).

Customers shopping around for cost saving opportunities in the IIoT space will always be leveraging equipment from multiple vendors, be it for price or the system capabilities they require. This leaves the burden of integrating the various components and their respective vendor-specific cloud connectivity solutions to the customer or a consulting partner company. When devices cannot be connected and integrated on-premises, their cloud counterparts will need to be integrated in the cloud. This produces additional cost for both the integration and the multiple-vendor hardware required to be installed in the factories and on-premises.

The solution lies in openly defining how devices can connect to a common edge gateway on-premises using their favorite or existing communication means and from there sending normalized or annotated data to the cloud vendor of choice. As such, the edge has to become device, protocol, vendor, and cloud-provider neutral, but not necessarily agnostic. A list of representative operational technology (OT) and information technology (IT) protocols is on the last page.

A simple and open device driver model and a modern container-based architecture on the edge allow for secure, efficient, and reliable connectivity to any cloud vendor to enable value-added services, applications, and analytical insights into the data in the cloud.

Competition is spurred again where it is supposed to be: in the cloud, based on rich and proper data from many vendor devices at once.

This is the beginning of a series of articles where we will describe the approach of democratizing the edge to enable vendors to compete on value based on the data instead of commodity hardware and connectivity.

How to build a device driver

To understand the concept of a device driver for IIoT devices requires some common understanding of the established reference architecture for edge gateways.

Typically, edge gateways are located in the IT networking environment of a company, factory, or otherwise enclosed environment with physical protection boundaries.

Other than the edge gateway, no device should have access to the IT network or direct access to the Internet. (See figure 1).

This is to prevent access to the devices from outside the OT environment, and prevents compromised devices from opening up unwanted backdoors to the premises as was famously exploited in recent botnet attacks.

The devices connect to the edge gateway either directly through a network technology supported by the gateway such as serial bus or Ethernet, or through an intermediate adapter like a product life-cycle management (PLM) or distributed control system (DCS) that bridges the closed loop circle of a device bus with a more common network protocol.

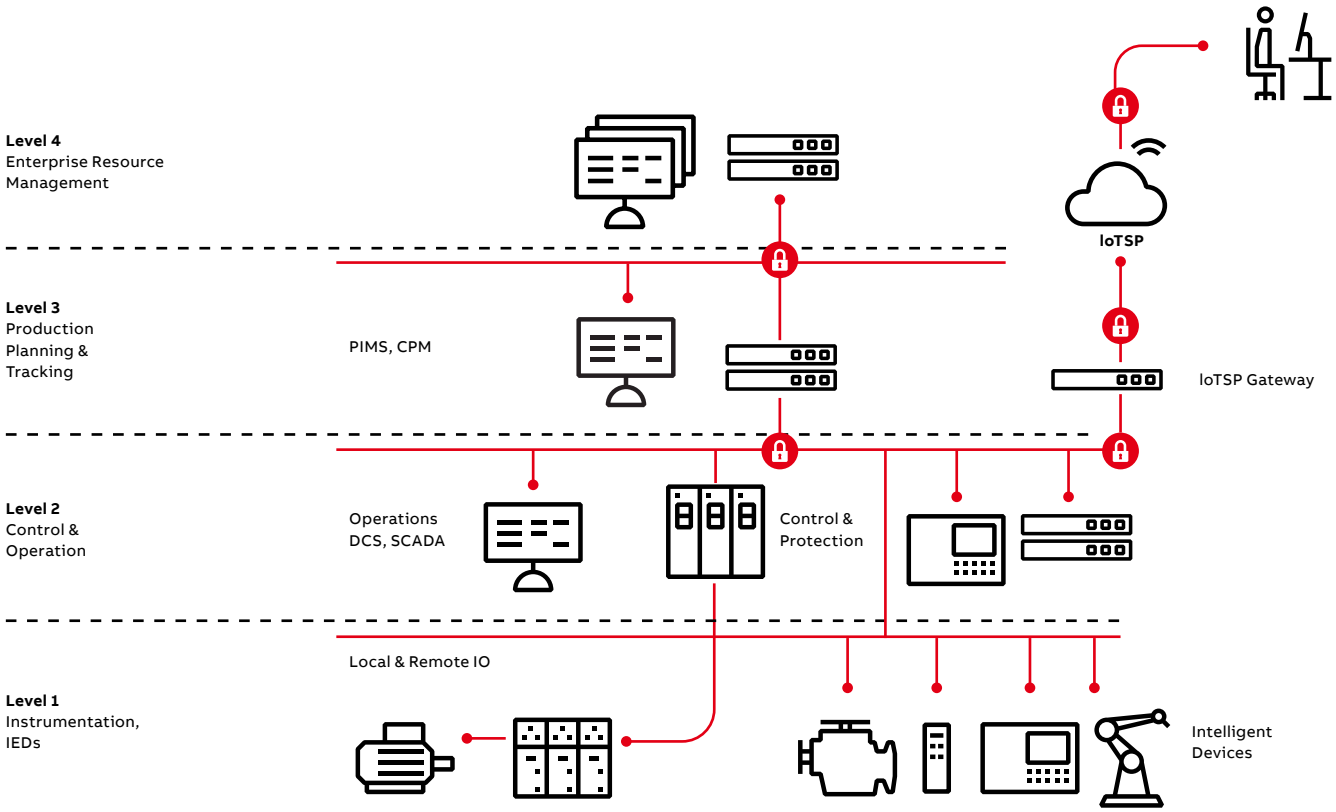


Figure 1
On-premises network topology

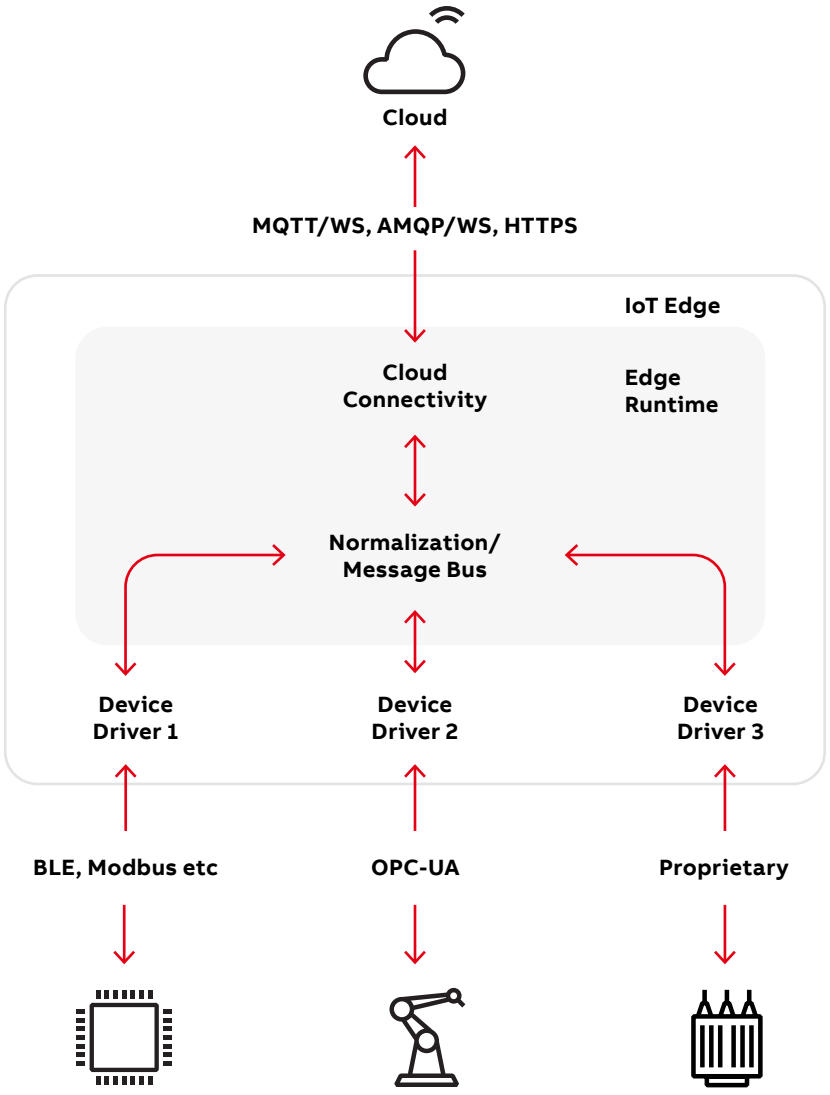
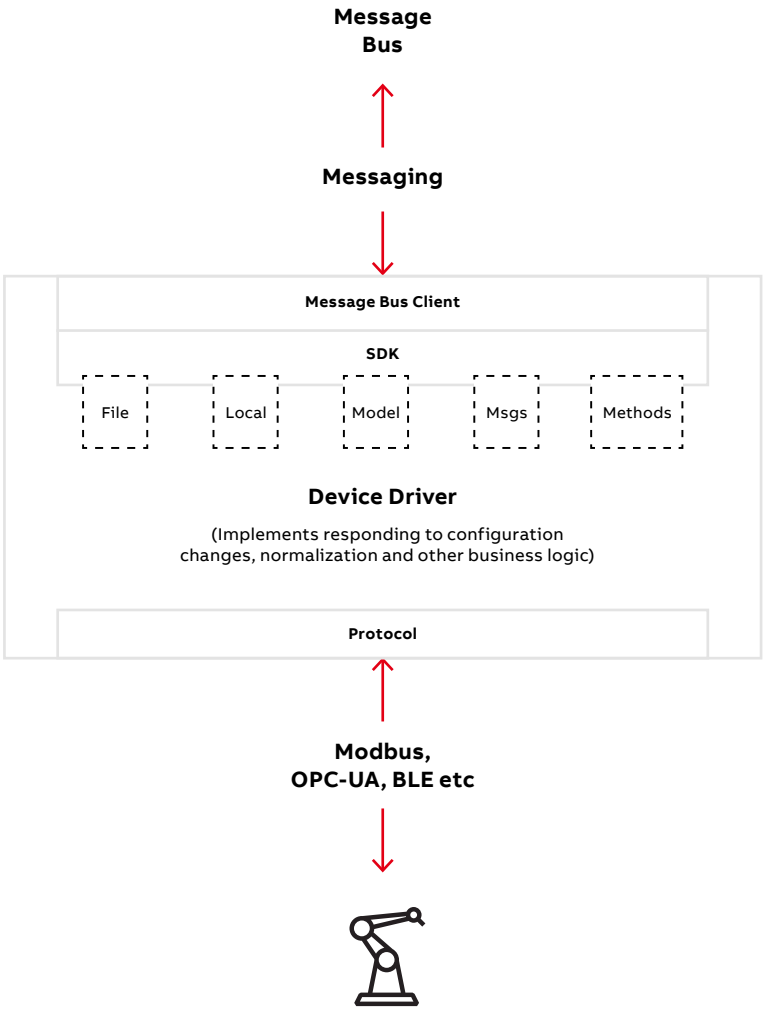


Figure 2
Democratized IoT edge gateway

The purpose of the device driver is to translate the device-specific protocol data (whether raw, binary, aggregated, or pre-processed) into a messaging model that is understood by an open and standardized gateway that can help the drivers communicate with each other and/or with the cloud.

In addition, other application logic should be able to further process the data and decide to store it locally, process it further by applying local analytical models, or send the data off to the cloud. This additional processing should be independent of the device driver.

Device drivers cover the entire lifecycle of a device, including device discovery or registration, configuration, enablement, file upload, and communication management. In addition, they provide standard means for important cybersecurity measures such as certificate and patch management, firmware update, and maybe even remote management access.



Edge Module (Device Driver)

- Distributed and bundled as an image and runs as a container
- Communicates with the devices using the corresponding protocol such as OPC-UA, Modbus, Serial, etc.
- Communicates with the runtime using the message bus client. Optionally can use SDK that provides higher level abstraction for the interfaces such as outbound messages, model CRUD, method invocations, file upload/ download, inter module local communication

Figure 3
Device driver architecture

The edge gateway hosts a flexible collection of modules. Modern architectures recommend the use of containerization techniques where each module is run as a separate container and process.

Containerization implementations like Docker and runc have become available on any range of hardware and processors and established themselves as a natural choice.

The minimum edge environment consists of:

- A broker module that provides a common messaging bus, such as an MQTT broker
- A proxy that is able to connect the edge securely with the target IoT infrastructure in the cloud
- An agent module that manages the configuration of the edge and controls the lifecycle of module deployments

All other modules are optional to either connect devices to the edge, autonomously process the data locally, or assist the user by providing user interfaces (UIs) and other applications.

To write a module, the vendor may choose one of the available device driver SDKs to implement the necessary functions that translate from the device-specific protocol to a common messaging format as well as control the configuration and the device lifecycle.

Device drivers are deployed as containers and follow the best practices for containerizations, including minimal image, single process, unprivileged user, immutability and others.

Here are some of the functions that are covered by the device driver SDK:

- **Type Definition**
  Each model requires a type definition and an identity. We will explore the details of this in a later article of the series.
- **Connecting to the MQTT broker**
  Each module container is injected with the URI of the broker and the module specific credentials. The credentials are stored in a local key vault and protected from external access.
- **Persistence**
  Each module container has a mounted directory with read/write access which allows it to store any files that need to be persisted between restarts. The local file system is encrypted and protected from external access.
- **Receiving configuration updates**
  Modules can subscribe to a specific topic to receive updates to their device or configuration definitions.
- **Direct method invocations**
  Modules can subscribe to a specific topic that contains messages with details of an operation that it is instructed to perform. The permitted operations are defined in the module definition.
- **Module-to-module communication**
  Each module defines an input and output queue. With the proper permissions configured, any module can put a message into the input topic of another module.
- **Device-to-cloud communication**
  Modules can send data to the cloud by publishing messages into an outbound topic. These messages can contain:
  - Telemetry data
  - Events
  - Alarms
  - Configuration data
  - Events intended for an external management application, such as deviceCreated, deviceUpdated, deviceDeleted
- **File upload**
  Files can be uploaded to a file storage in the cloud by posting the local file location on a topic with the broker. A flag determines whether to remove the local file after transmission or not.
- **File download**
  Used for firmware update and remote patch management.

In addition, the democratized edge adheres to the industry best practices regarding monitoring, logging, auditing, and connection management. For example, device driver modules are prevented or at least discouraged from connecting to anything other than the devices they manage; all access is audit logged and role based.

How the devices are managed remotely from the cloud is subject of a later article.

Who is out there to connect?
The easiest and most straightforward protocols to write device drivers for are those we are familiar with from IT environments, listed below in Table 1.

AFP	BGP	DHCP
DNS	FTP	HTTP
IMAP	Kerberos	LDAP
LDP	MS-SQL	NTP
NetBIOS	OpenRDA	POP3
PVSS	Radius	RDP
RFB/VNC	RPC/DCOM	RTSP
SMB	SMTP	SNMP
SSDP	SSH	SSL
SunRPC	Telnet	TFTP

Table 1
IT protocols

However, this set of protocols will only cover the IT network that in a typical factory environment will be isolated from the actual devices operating on the floor or sensing humidity in the room.

The majority of devices will be connected using standard or proprietary OT protocols over different networks not limited to TCP/IP over Ethernet. These are listed below in Table 2.

Standard OT protocols*	Proprietary OT protocols
BACnet	CSLib (ABB 800xA)
DNP3	DMS (ABB AC 800F)
EtherNet/IP + CIP	MMS (ABB AC 800 M)
Foundation Fieldbus HSE	PN800 (ABB)
IEC 60870-5-104	ADS/AMS (Beckhoff)
ICCP TASE.2	CygNet SCADA (CygNet)
IEC 61850 (MMS, GOOSE, SV)	DeltaV (Emerson)
IEEE C37.118	Ovation (Emerson)
Modbus/TCP	SRTP (GE)
OPC-DA	Experion (Honeywell)
OPC-AE	ADE (Phoenix Contact)
PROFINET(RPC,RTC,RTA,DCP,PTCP)	CIP Extension (Rockwell/AB)
	OASys (Schneider Electric)
	Modbus Extensions (Schneider Electric)
	Telnet Extensions (SEL)
	Step7 (Siemens)
	S7COMM+/OMS (Siemens)
	Vnet/IP (Yokogawa)

Table 2
OT protocols

While some of these protocols can be addressed by generic device drivers (Modbus/TCP and IEC 61850 are fairly well described and transferable), the more proprietary counterparts or extensions will require those vendors to supply the binding to their protocol.

By supplying the device driver as a container on the marketplace, this can be done entirely without exposing their intellectual property.

Next Steps
In the next installment of this series of articles, we will dive deeper into understanding the devices and their data models. We will explain and motivate the need for a common device information meta model that allows the modules to communicate with each other while the models for the device data are supplied by the vendors and not commonly shared.

In a later article we will examine the specifics of the connectivity from the edge to the cloud provider and what this means to the device provisioning flows and lifecycle. After that, we will look into a reference architecture in the cloud that is able to provide value to consumer applications independent of the cloud provider and devices, based on the data from the edge.

ABB Inc.
3055 Orchard Drive
San Jose, CA 95134
USA
ability.abb.com