OPC UA: the communication protocol for industrial automation applications

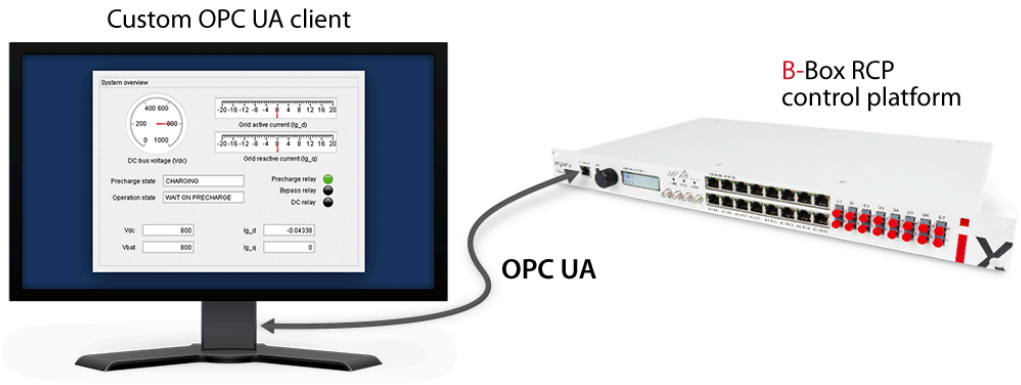
PN177 | Posted on March 24, 2022 | Updated on May 7, 2025

Benoît STEINMANN
Software Team Leader
imperix

Table of Contents

Imperix relies on the increasingly popular **OPC UA industrial protocol** to remotely control and monitor its power electronic controllers over Ethernet. To do so, the B-Board PRO and B-Box RCP controllers embed an OPC UA server to communicate with the in-house monitoring software imperix Cockpit. One of the advantages of using an open standard such as OPC UA is that it enables users to use **third-party clients to visualize data and send commands**.

This page is aimed at users who wish to implement a custom OPC UA client to interact with an imperix controller and provides guidance on how to connect to and exchange data with the OPC UA server embedded in imperix controllers.



What is OPC UA?

OPC Unified Architecture (UA) is an open communication standard developed by the OPC Foundation targeting industrial automation. This Ethernet-based protocol allows clients to remotely control OPC UA compatible devices from various vendors. OPC UA is designed to be a secure industrial communication and supports signed and encrypted communication as well as user authentication.

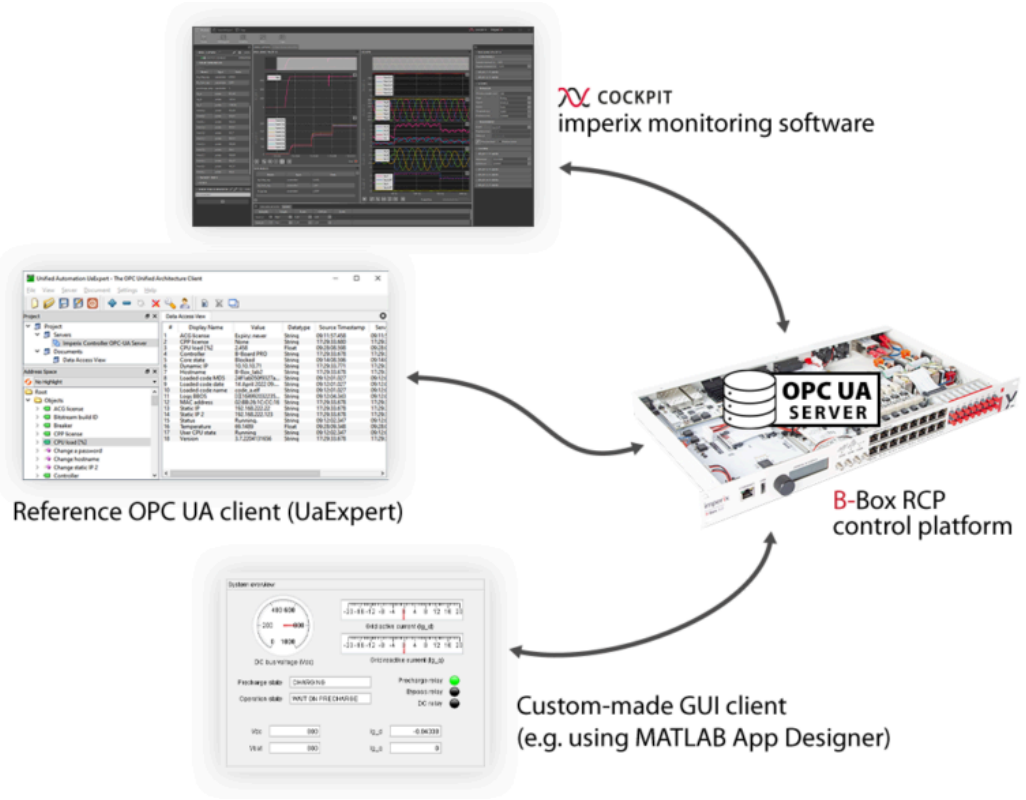
Moreover, thanks to the openness of OPC UA, a multitude of resources and libraries are available online, allowing the user to build his own client. Here are a few of them:

- Official tools and SDK from Unified Automation: High Perf C99, ANSI C, C++, .NET C#, Java, Delphi
- MATLAB Industrial Communication Toolbox, also covered on our product note
- open62541: open-source C implementation
- NodeOPCUA: Javascript or Typescript for NodeJS
- Freeopcua Python: with a simple GUI client example made using PyQT 5

Monitoring an imperix controller using OPC UA

Imperix Cockpit, the official monitoring software to remotely control imperix products, relies on OPC UA for most of the communication with the B-Box RCP and B-Board PRO. Consequentially, third-party clients may perform the same actions, including:

- read and write the user variables (probe and tunable parameters) at a frequency up to 100 Hz
- enable or disable the PWM outputs
- get the device status (is the code running, did a fault occur, what is the CPU load, etc)
- read the generated log messages



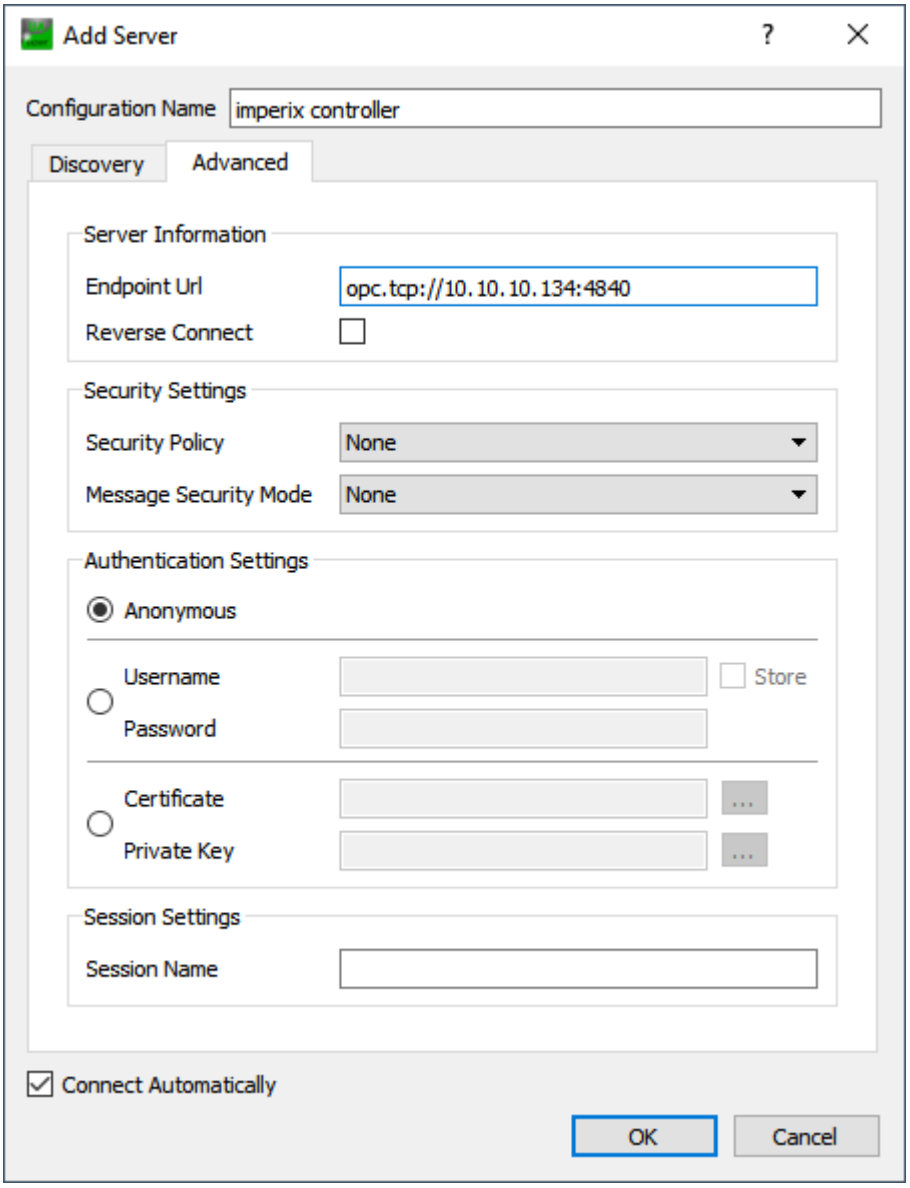
OPC UA enables different clients to interact with a B-Box controller

It must be noted that actions such as loading the user code file or a custom FPGA bitstream file into the target or updating the target firmware can not be performed using OPC UA. Additionally, due to complexity and performance considerations, the scope module of Cockpit also uses an in-house protocol.

Connecting to the imperix OPC UA server

Imperix uses the default OPC UA port 4840, meaning that the OPC UA endpoint URL to connect to a B-Box or B-Board controller is **`opc.tcp://<IP-ADDRESS>:4840`**

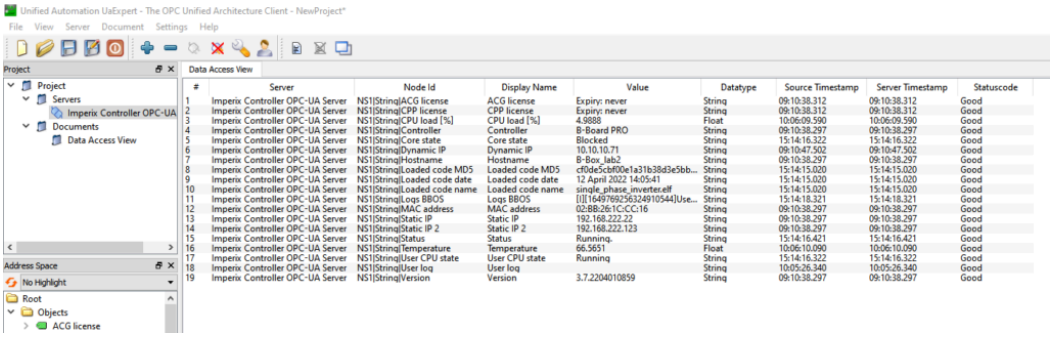
Screenshots from the OPC UA client UaExpert will be used as a reference, however, the explanations apply to any other client. Using UaExpert is also a good way to test the connection to the OPC UA server as well as to obtain a list of the available variables and functions.



Connecting to an imperix controller OPA UA server from UaExpert

Imperix system variables

Below are documented the read-only “system” OPC UA node variables that can be of use to the user. They provide configuration and status information about the target. These system node variables are stored in the namespace 1 and are available at all times (contrary to user code variables described later).



imperix controller OPA UA server variables displayed in UaExpert

Some variables are not documented because they are designed to be used exclusively by Cockpit.

Name	Datatype	Description
`ACG license`	String	The loaded ACG license status
`CPP license`	String	The loaded CPP license status
`CPU load [%]`	Float	User CPU load. Returns 0 when the user code is stopped
`Controller`	String	Returns “B-Board” or “B-Box”
`Core state`	String	“Offline”: The user code is stopped “Blocked”, “Operating” or “Fault” when the user code is running
`Dynamic IP`	String	The automatically assigned IP address
`Hostname`	String	The device hostname
`Loaded code MD5`	String	The MD5 hash of the loaded user code file
`Loaded code date`	String	The last modification date of the loaded user code file
`Loaded code name`	String	The name of the loaded user code file

`Logs BBOS`	String	Returns the log messages generated by the user code system (BBOS). It uses the following format:
		`[severity][timestamp]Message`
		For instance:
		`[I][1648114327359324364]User code started (BBOS v3.7 build 2203211046)`

`MAC address`	String	The device’s MAC address
`Static IP`	String	The default static IP. It is always 192.168.222.22
`Static IP 2`	String	The user-configured static IP
`Status`	String	Display status similar to the B-Box screen. “Initializing…”, “Discovering…” or “Synchronizing…”: the device is in its

		starting phase “Ready.”: the user code can be started “Starting…”: the user code is being started “Running.”: a user code is running All other messages are error messages
`Temperature`	Float	The chip temperature in degrees Celsius
`User CPU state`	String	“Running” or “Offline”
`User log`	String	Returns the user-created log messages generated using the LOG block. It uses the following format: [severity][timestamp]Message
`Version`	String	The firmware version of the device

Imperix methods

The following methods allow the user to interact with the target (start the code, enable the PWM outputs) or to change system parameters (hostname, IP).

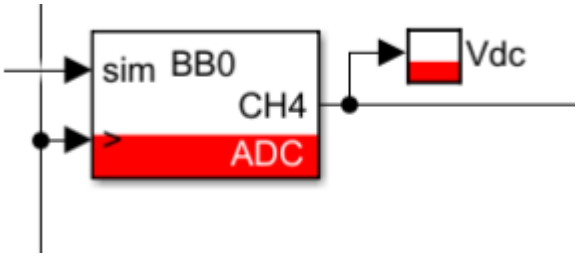
Some methods are not documented because they are designed to be used exclusively by Cockpit.

Method name		**Argument**	**Description**
`Change hostname`		new hostname	Change the target hostname
`Change static IP 2`		new static IP	Change the target static IP 2. The static IP 1 is always 192.168.222.22
`Enable outputs`		N/A	Enable the PWM outputs
`Disable outputs`		N/A	Disable the PWM outputs
`Set DAC`		<variable name>, <DAC channel>, <device ID>, <gain>, <offset>	Connect a user variable to an analog output of the target, similar to the DAC module of Cockpit

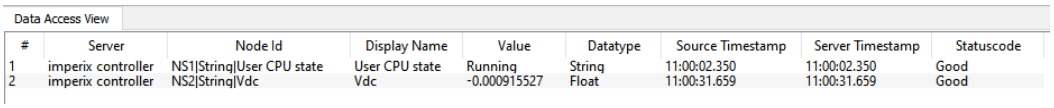
`Start code`		N/A	Start the loaded user code.
`Stop code`		N/A	Stop the loaded user code.

User-created OPC UA variables

When a user code is started, an OPC UA variable is created for each probe variable and tunable parameter present in the user model (Simulink, PLECS or C++). They are stored in the namespace index 2.



The probe “Vdc” is connected to the output of an ADC block



Data Access View

#	Server	Node Id	Display Name	Value	Datatype	Source Timestamp	Server Timestamp	Statuscode
1	imperix controller	NS1|String|User CPU state	User CPU state	Running	String	11:00:02.350	11:00:02.350	Good
2	imperix controller	NS2|String|Vdc	Vdc	-0.000915527	Float	11:00:31.659	11:00:31.659	Good

The probe “Vdc” of the user model is available as an OPC UA variable (UaExpert)

Going further

Making an OPC UA GUI client using MATLAB

To use MATLAB as an OPC UA client, the user may read the Industrial Communication Toolbox tutorial. This Add-On may be combined with MATLAB App Designer to build an HMI GUI for imperix power converters such as the one shown in the illustration on this page.

Making an OPC UA GUI client using C++

To go on the completely free-of-cost route, we would recommend checking out the C library open62541 and using a C++ framework such as Qt or wxWidgets to build a GUI.

Making an OPC UA GUI client using Python

To stay on the free and open-source route, developing a client using Python is also possible thanks to Freeopcua. A GUI can be built quite easily using the popular Python package tkinter.

Ethernet is not the only way to communicate with the imperix controllers. Indeed, the CAN protocol is also supported thanks to the CAN input and CAN output mailboxes.