

# UAFIP TSN Gateway

## ***User's Guide***

Exoligent's Team

Version v1.0.0, 2022-06-21

# Table of Contents

1. Disclaimer .....	1
2. Introduction .....	2
3. Prerequisites .....	3
3.1. Hardware requirements .....	3
3.2. Software requirements .....	3
4. Environment setup .....	4
4.1. Install real-time Linux Kernel .....	4
4.2. Install FIP controller .....	4
4.3. Configure Ethernet network interface .....	6
4.4. Configure TSN parameters .....	7
4.5. Build UAFIP TSN gateway .....	9
4.5.1. XDP Pre-requisties .....	9
4.5.2. Build open62541 .....	10
4.5.3. Build the gateway .....	10
4.5.4. Start the gateway .....	10
4.5.5. Install the gateway as a daemon .....	11
5. Gateway configuration .....	13
6. Ping-Pong example .....	19
Appendix A: JSON files examples .....	23
A.1. Ping (FIP Node 0) .....	23
A.2. Pong (FIP Node 1) .....	25
Appendix B: Glossary of acronyms .....	27
Appendix C: Revision History .....	28

# Chapter 1. Disclaimer

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. EXOLIGENT MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION, INCLUDING BUT NOT LIMITED TO ITS CONDITION, QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR PURPOSE. Exoligent disclaims all liability arising from this information and its use. Use of Exoligent devices in life support and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless Exoligent from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Exoligent intellectual property rights unless otherwise stated.

Exoligent SARL

390 rue d'Estienne d'Orves, 92700 Colombes - France

Tel: +33(0) 1 42 42 42 00

Web Site: <https://www.exoligent.com/>

Copyright © 2022 Exoligent SARL. All rights reserved.

## Chapter 2. Introduction

UAFIP is an open-source software gateway written in C for GNU/Linux operating system whose goal is to create a deterministic gateway between the WorldFIP fieldbus protocol (IEC 61158) and the OPC UA over TSN (IEC 62541) standard for data exchange. This user guide serves as starting point for a user to learn, install and configure this gateway for embedding into their products.



This document is partly based on the very good guide from Kalycito:  
[How to run OPC UA PubSub on real-time Linux and TSN](#)

The structure of the UAFIP TSN gateway is divided into three entities:

- Configurator: json-c – an open-source C JSON parser licensed under MIT ([github](#))
- OPC UA stack: open62541 – an open-source C (C99) implementation of OPC UA licensed under the Mozilla Public License v2.0 ([github](#))
- WorldFIP stack: powerfip – a C communication stack to interface with the Exoligent's FIP coprocessor ([doc](#))

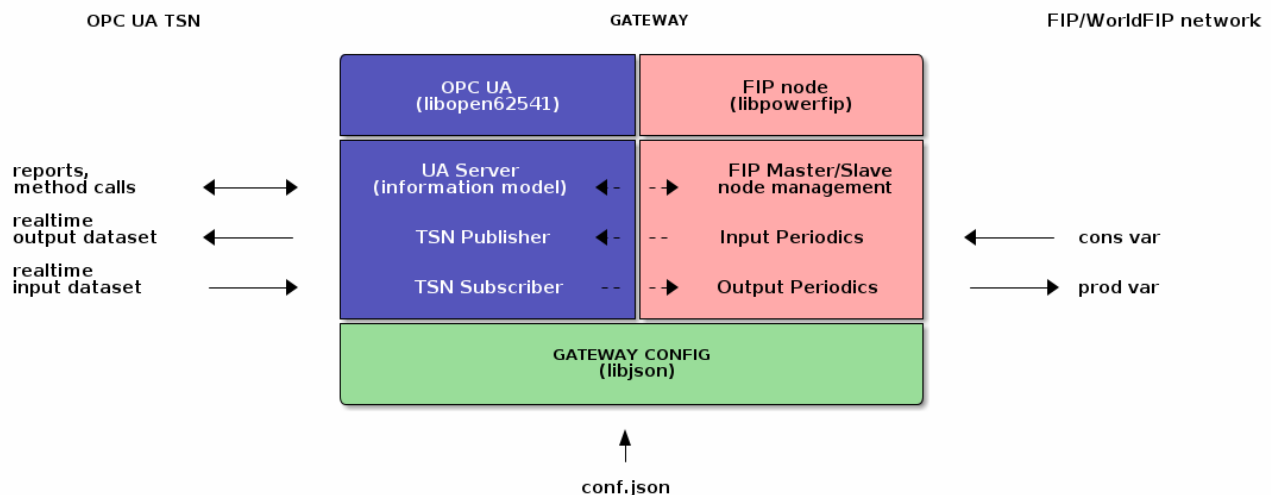


Figure 1. UAFIP software gateway (Principle Diagram)

# Chapter 3. Prerequisites

## 3.1. Hardware requirements

- x86-based system
  - 4-cores
  - Intel i210 Ethernet controller (as this controller supports features necessary for Time-Sensitive Networking)
  - 1 x PCIe slot or 1 x mPCIe slot
- FIP controller
  - PowerFIP Exoligent's board ([mPCIe](#) and its derivatives)

We used Advantech's [UNO-1483G](#) during our tests.

## 3.2. Software requirements

- Debian GNU/Linux 10 and more
  - Pre-compiled real-time kernel
  - A later version of iproute2 package (with support for newer real-time socket options and an IEEE 802.1 Qbv like scheduler)
  - A later version of LinuxPTP package (with support for IEEE 802.1 AS gPTP configuration)
- FIP library/kernel module + UAFIP TSN gateway sources (see last [PowerFIP Package](#))

We used Debian GNU/Linux 11 (bullseye) with PREEMPT-RT 5.10.0-14-rt-amd64 kernel during our tests



We selected Debian 11, as it is one of the more popular distributions. It also has a pre-compiled real-time kernel that can be installed using the Debian package manager.

As it is not practically possible to provide a variant of the userguide for the many Linux distributions out there, we recommend that you use Debian for your initial tests and then switch to your chosen distribution.

# Chapter 4. Environment setup

In this section, we will set up the environment:

- Install real-time Linux kernel
- Install FIP controller
- Configure Ethernet network interface
- Configure TSN parameters
- Build UAFIP TSN gateway

## 4.1. Install real-time Linux Kernel

After installing the OS, update the node using the below command:

```
$ sudo apt-get update && apt-get upgrade
```

It might take a while for the update and upgrade to complete. You can then download and install the real-time kernel that is distributed by Debian and reboot the node using the below commands:

```
$ sudo apt-get install linux-image-rt-amd64 -y  
$ sudo reboot
```

After reboot, execute the below command and ensure that the newly installed real-time kernel is the one that is currently running.

The appearance of the “rt” keyword in the print message that appears as part of the command output shows that we have booted into the right kernel:

```
$ uname -r  
  
5.10.0-14-rt-amd64
```

## 4.2. Install FIP controller

In this section, we will install FIP controller on your machine.

Once the real time Linux kernel is installed, shutdown your system and plug a PowerFIP PCIe card into a suitable slot

After reboot and log in user, download the archive of the latest version of the library from the Exoligent website ([Download section](#)) or get it from the following linux command (here the example is for the v1.4.0 package):

```
$ cd ~/Documents
$ wget https://www.exoligent.com/wiki/worldfip/pwr/fip/download/1.4.0/powerfip-1.4.0-
linux.tar.gz
$ tar xzvf powerfip-1.4.0-linux.tar.gz
$ cd powerfip-1.4.0-linux
```

Build and install the kernel module for FIP controller:

```
$ cd ~/Documents/powerfip-1.4.0-linux/driver/linux
$ make
$ sudo ./install.sh
```

If you have a PowerFIP PCIe device connected to your machine, you should get the following console trace with the *dmesg* command:

```
$ dmesg

[78983.022874] powerfip: ==> Init
[78983.022934] powerfip: ==> Device probing
[78983.023186] powerfip: BAR 0 (0xf6000000 - 0xf6000fff), len = 4096, flags = 0x040200
[78983.023205] powerfip: BAR 1 (0xf4000000 - 0xf5ffffff), len = 33554432, flags =
0x040200
[78983.023883] powerfip: DMA Capability = YES
[78983.023887] powerfip: MSI Capability = YES
[78983.023888] powerfip: IRQ pin = #1 (0=none, 1=INTA#...4=INTD#)
[78983.023890] powerfip: IRQ line = #69
[78983.023891] powerfip: IRQ vectors = 4
```

Now, install FIP firmware and library:

```
$ cd ../../
$ sudo ./install.sh
```

#### Following files will be copied to your system



1. **powerfip-firmware.bin** file to the path:  
/usr/local/lib/firmware
2. **libpowerfip.a** and **libpowerfip.so** files to the path:  
/usr/local/lib
3. **header (\*.h)** files to the path:  
/usr/local/include/powerfip

The FIP controller is now ready for operation! If you want to test it, refer you to the turnkey

examples in the [tools/](#) directory of the distribution.

## 4.3. Configure Ethernet network interface

In this section, we will set up static IP address and VLAN configuration to the i210 interface. Open the interfaces file using the below commands:

```
$ sudo nano /etc/network/interfaces
```

Copy and paste the following lines at the end of the interfaces file in both the nodes and replace “X” as shown in the screenshot below and save the file. E.g., In node 1 (ex: ua\_pong node example), set the IP address as 192.168.100.1, and in node 2 (ex: uafip tsn gateway node), set the IP address as 192.168.100.2. The PubSub TSN applications are configured to run with VLAN Id 8. For this purpose, VLAN configuration is done in the i210 interface as below:

```
auto enp7s0
iface enp7s0 inet static
    address 192.168.100.X
    netmask 255.255.255.0
    broadcast 192.168.100.255
auto enp7s0.8
iface enp7s0.8 inet static
    address 192.168.8.X
    netmask 255.255.255.0
```

Throughout this QSG we will be using *enp7s0* for the interface name, replace it with your nodes' i210 interface name.

Below is the screenshot of the interfaces files of the two nodes:

```
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

source /etc/network/interfaces.d/*

# The loopback network interface
auto lo
iface lo inet loopback

auto enp2s0
iface enp2s0 inet static
    address 192.168.100.1
    netmask 255.255.255.0
    broadcast 192.168.100.255
auto enp2s0.8
iface enp2s0.8 inet static
    address 192.168.8.1
    netmask 255.255.255.0

# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

source /etc/network/interfaces.d/*

# The loopback network interface
auto lo
iface lo inet loopback

auto enp7s0
iface enp7s0 inet static
    address 192.168.100.2
    netmask 255.255.255.0
    broadcast 192.168.100.255
auto enp7s0.8
iface enp7s0.8 inet static
    address 192.168.8.2
    netmask 255.255.255.0
```

Save the interfaces file and restart the networking service:

```
$ sudo /etc/init.d/networking restart
```

Now verify the static IP address and the connection between two nodes (from node 1 ping node 2



and vice versa) using the below commands:

```
$ ip a

[...]

3: enp7s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 xdpgeneric/id:34 qdisc mqprio
state UP group default qlen 1000
    link/ether 74:fe:48:30:39:eb brd ff:ff:ff:ff:ff:ff
    inet 192.168.100.2/24 brd 192.168.100.255 scope global enp7s0
        valid_lft forever preferred_lft forever
    inet6 fe80::76fe:48ff:fe30:39eb/64 scope link
        valid_lft forever preferred_lft forever

[...]
```

```
$ ping 192.168.100.X
```

## 4.4. Configure TSN parameters

In this section, we will run some scripts from the Kalycito company (`setup.sh`, `application_dependencies.sh`) to configure the TSN parameters.

The `setup.sh` available in the package does the following,

- Updates the installed packages in the node
- Installs `iproute2` package for kernel version 5.10
- Installs Linux PTP version 2.0
- Checks if `8021q` module is loaded; if not, adds the same

The `application_dependencies.sh` available in the package does the following,

- Configures traffic control parameters to transmit and receive
- Sets Egress and Ingress policies
- Tunes real-time behaviors
- Runs Linux PTP and PHC2SYS
- Kernel mechanism to allocate CPU to the processes

The above scripts are included in the `powerfip` package. To access them, go to the `kalycito` directory, and run the `setup.sh` script using the below command:

```
$ cd ~/Documents/powerfip-1.4.0-linux/tools/uafip_tsn_gateway/scripts/kalycito
$ sudo ./setup.sh
```



We have modified the grub settings, so reboot is required.

Reboot the nodes after completing the initial setup

```
$ sudo reboot
```

### Run PTP and PHC2SYS

After the reboot, run the *application\_dependencies.sh*. As shown in the architecture diagram, configure one of the nodes as PTP master and the other node as PTP slave.

To configure the node as PTP master use the below command:

```
$ cd ~/Documents/powerfip-1.4.0-linux/tools/uafip_tsn_gateway/scripts/kalycito
$ sudo ./application_dependencies.sh -i enp2s0 -m
```

To configure the node as PTP slave use the below command:

```
$ sudo ./application_dependencies.sh -i enp2s0 -s
```

After configuring the node as PTP master or slave, the output log can be verified using the below command:

```
$ tail -f /var/log/ptp4l.log
```

If you have configured your node as a PTP master, your output log should be similar to the below image. The text “assuming the grand master role” ensures that you have configured the node as PTP master. Once you have seen the output, press Ctrl + C to return to the console.

```
$ tail -f /var/log/ptp4l.log

ptp4l[3194.498]: port 1: link up
ptp4l[3194.558]: port 1: FAULTY to LISTENING on INIT_COMPLETE
ptp4l[3205.493]: port 1: LISTENING to MASTER on ANNOUNCE_RECEIPT_TIMEOUT_EXPIRES
ptp4l[3205.493]: selected local clock 74fe48.ffff.42c2e4 as best master
ptp4l[3205.493]: port 1: assuming the grand master role
```

If you have configured your node as a PTP slave, your output log should be similar to the below image.



The master offset values are represented in nanosecond (ns). This value should be within the range of -1000 ns to +1000 ns

```
$ tail -f /var/log/ptp4l.log
```

```
ptp4l[4527.664]: master offset      4 s2 freq +39804 path delay      -17
ptp4l[4527.789]: master offset      5 s2 freq +39806 path delay      -17
ptp4l[4527.914]: master offset      0 s2 freq +39799 path delay      -17
ptp4l[4528.039]: master offset      0 s2 freq +39799 path delay      -17
ptp4l[4528.165]: master offset     -6 s2 freq +39791 path delay      -17
ptp4l[4528.290]: master offset     -4 s2 freq +39793 path delay      -17
```

The application\_dependencies.sh would have run PHC2SYS utility to synchronize user clock and hardware clock. You can verify the output log of PHC2SYS in both the nodes using the below command:

```
$ tail -f /var/log/phc2sys.log
```

Your PHC2SYS log should be similar to the below image.



The sys offset values are represented in nanosecond (ns). This value should be within the range of -1000 ns to +1000 ns

Once you have seen the output, press Ctrl + C to return to the console.

## 4.5. Build UAFIP TSN gateway

### 4.5.1. XDP Pre-requisties

XDP (Express Data Path) is used by the gateway subscriber for faster processing the data.



For our test with the Debian 11 OS (5.10.0-14-rt-amd64) we use the bpf library v0.3.

To build and install libbpf library, use the following commands:

```
$ sudo apt install llvm
$ sudo apt install clang

$ cd ~/Documents
$ git clone https://github.com/libbpf/libbpf.git
$ cd libbpf/
$ git checkout 051a4009f94d5633a8f734ca4235f0a78ee90469
$ cd src/
$ sudo OBJDIR=/usr/lib make install
```

#### 4.5.2. Build open62541

```
$ sudo apt install build-essential gcc pkg-config cmake python

$ cd ~/Documents
$ wget https://github.com/open62541/open62541/archive/refs/tags/v1.3.tar.gz -O
open62541-1.3.0.tar.gz
$ tar xzvf open62541-1.3.0.tar.gz
$ cd open62541-1.3.0
$ mkdir build
$ cd build
$ cmake .. -G "Unix Makefiles" -DCMAKE_BUILD_TYPE=Release -DOPEN62541_VERSION=v1.3.0
-DUA_ENABLE_PUBSUB=ON -DUA_ENABLE_PUBSUB_ETH_UADP=ON -DUA_ENABLE_MALLOC_SINGLETON=ON
-DUA_ENABLE_IMMUTABLE_NODES=ON -DUA_ENABLE_PUBSUB_INFORMATIONMODEL=ON
-DUA_MULTITHREADING=150
$ make
$ sudo make install
```

#### 4.5.3. Build the gateway

```
$ sudo apt install libjson-c-dev

$ cd ~/Documents/powerfip-1.4.0-linux/tools/uafip_tsn_gateway
$ make
```

#### 4.5.4. Start the gateway

```
$ cd ~/Documents/powerfip-1.4.0-linux/tools/uafip_tsn_gateway
$ sudo setcap cap_sys_nice+ep uafipd
$ ./uafipd -f ./config/2sta/sta1.json
```

### 4.5.5. Install the gateway as a daemon

Install UAFIP gateway as a daemon:

```
$ cd ~/Documents/powerfip-1.4.0-linux/tools/uafip_tsn_gateway/svc/  
$ sudo ./install_svc.sh  
  
Created symlink /etc/systemd/system/multi-user.target.wants/uafipd.service →  
/etc/systemd/system/uafipd.service.
```

Once installed, you can manipulate the daemon with the classic commands of systemd:

Daemon status:

```
$ sudo systemctl status uafipd  
  
■ uafipd.service - OPC UA Pub/Sub TSN <-> FIP/WorldFIP gateway  
   Loaded: loaded (/etc/systemd/system/uafipd.service; enabled; vendor preset:  
   enabled)  
   Active: active (running) since Fri 2022-06-10 15:01:52 CEST; 59s ago  
   Main PID: 5910 (uafipd)  
     Tasks: 5 (limit: 9383)  
    Memory: 3.6M  
       CPU: 1.241s  
    CGroup: /system.slice/uafipd.service  
            └─5910 /usr/bin/uafipd -f  
            /etc/exoligent/uafip/config/performance/sta1.json -L /var/log/uafipd.log
```

Stop the daemon:

```
$ sudo systemctl stop uafipd
```

Start the daemon:

```
$ sudo systemctl start uafipd
```

Enable the daemon at system startup:

```
$ sudo systemctl enable uafipd  
  
Created symlink /etc/systemd/system/multi-user.target.wants/uafipd.service →  
/etc/systemd/system/uafipd.service.
```

Disable the daemon at system startup:

```
$ sudo systemctl disable uafipd
```

```
Removed /etc/systemd/system/multi-user.target.wants/uafipd.service.
```

## Chapter 5. Gateway configuration

The UAFIP TSN gateway can be configured using a JSON file. The OPC PUB/SUB parameters are defined there as well as the parameters of the embedded FIP node.

*Example of UAFIP gateway configuration file (config/ping/uno/sta1.json)*

```
{
  "opc": {
    "publisher": {
      "network_interface": "enp7s0",
      "url": "opc.eth://74-fe-48-42-c2-e4:8.3",
      "publisher_id": "5671",
      "dataset_writer_id": "62541",
      "writer_group_id": "101",
      "publish_period_us": "250"
    },
    "subscriber": {
      "network_interface": "enp7s0",
      "url": "opc.eth://74-fe-48-30-39-eb:8.3",
      "publisher_id": "5670",
      "dataset_writer_id": "62541",
      "writer_group_id": "100"
    }
  },
  "fip": {
    "name": "FIP node 1",
    "address": "1",
    "segment": "0",
    "frame_type": "worldfip",
    "bitrate": "1mbps",
    "turn_around_us": "30",
    "silence_us": "150",
    "vars": [
      {
        "name": "[0x3800]",
        "type": "cons",
        "id": "0x3800",
        "payload_bsz": "8",
        "refreshment": "yes",
        "promptness": "yes",
        "prompt_us": "7500",
        "event": "none"
      },
      {
        "name": "[0x3801]",
        "type": "prod",
        "id": "0x3801",

```

```

    "payload_bsz": "8",
    "refreshment": "yes",
    "refresh_us": "7500",
    "event": "none"
  },
  {
    "name": "[0xFF00]",
    "type": "sync",
    "id": "0xFF00",
    "event": "w_op"
  },
  {
    "name": "[0xFF01]",
    "type": "sync",
    "id": "0xFF01",
    "event": "r_op"
  }
]
}

```



See appendix `sta0.json` to get a gateway configuration example with a master FIP node (Bus Arbiter enabled).





The JSON file is a data tree structure composed of nodes (containers) and leaves (data). Below is a description of the layout of the *containers* and *[arrays]*:




- ■ `opc`
  - ■ `publisher`
    - ■ `network_interface, (...)`
  - ■ `subscriber`
    - ■ `network_interface, (...)`
- ■ `fip`
  - ■ `name, (...), [vars]`
    - ■ `name, (...)`
  - ■ `bus_arbiter`
    - ■ `enable, (...), [cycles]`
      - ■ `[windows]`
        - ■ `type, (...)`
        - ■ `[requests] (only inside periodic window)`
          - ■ `type, (...)`



Name	Status	Type	Description
<b>opc</b>	required	object	OPC-UA configuration container
<b>publisher</b>	required	object	OPC-UA PUB container
<b>network_interface</b>	required	string	Publisher network interface
<b>url</b>	required	string	Publisher MAC address ex: opc.eth://74-fe-48-30-39-eb:8.3 (where 8 is the VLAN ID and 3 is the PCP) ex: opc.udp://224.0.0.32:4840/ (multicast address)
<b>publisher_id</b>	required	u16	Unique ID for a publisher with the message-oriented middleware
<b>dataset_writer_id</b>	required	u16	The dataset writer Id identifies the dataset writer in the writer group. It is unique across all dataset writers for a publisher ID
<b>writer_group_id</b>	required	u16	Writer Group Identifier. It is unique across writer groups for a publisher ID
<b>publish_period_us</b>	required	u32	Publisher cycle time in microseconds
<b>subscriber</b>	required	object	OPC-UA SUB container
<b>network_interface</b>	required	string	Subscriber network interface
<b>url</b>	required	string	Subscriber MAC address ex: opc.eth://74-fe-48-42-c2-e4:8.3 (where 8 is the VLAN ID and 3 is the PCP) ex: opc.udp://224.0.0.22:4840/ (multicast address)
<b>publisher_id</b>	required	u16	Unique ID of the publisher to subscribe to
<b>dataset_writer_id</b>	required	u16	Dataset Writer ID to subscribe to
<b>writer_group_id</b>	required	u16	Writer Group ID to subscribe to
<b>fip</b>	required	object	FIP configuration container
<b>name</b>	optional	string	FIP node identification label
<b>address</b>	required	u8	FIP node physical address
<b>segment</b>	required	u8	FIP node segment number

Name	Status	Type	Description
<b>frame_type</b>	required	enum	Values: <ul style="list-style-type: none"> <li>• <b>worldfip</b> WorldFIP frame encoding type (IEC [International Electrotechnical Commission])</li> <li>• <b>fip</b> FIP frame encoding type (UTE [Union Technique de l'Electricité])</li> </ul>
<b>bitrate</b>	required	enum	Values: <ul style="list-style-type: none"> <li>• <b>31.25kbps</b> FIP/WorldFIP bitrate at 31.25Kbps</li> <li>• <b>1mbps</b> FIP/WorldFIP bitrate at 1Mbps</li> <li>• <b>2.5mbps</b> FIP/WorldFIP bitrate at 2.5Mbps</li> <li>• <b>5mbps</b> FIP/WorldFIP bitrate at 5Mbps</li> <li>• <b>12.5mbps</b> FIP/WorldFIP bitrate at 12.5Mbps <i>[experimental]</i></li> <li>• <b>25mbps</b> FIP/WorldFIP bitrate at 25Mbps <i>[experimental]</i></li> </ul>
<b>turn_around_us</b>	required	u32	Turn-Around time in microseconds
<b>silence_us</b>	required	u32	Silence time in microseconds
<b>vars</b>	optional	array	Array of FIP variables configuration
<b>name</b>	optional	string	FIP variable identification label
<b>type</b>	required	enum	Values: <ul style="list-style-type: none"> <li>• <b>cons</b> Input FIP variable (consumed)</li> <li>• <b>prod</b> Output FIP variable (produced)</li> </ul>
<b>id</b>	required	u16	Unique identifier of a FIP variable

Name	Status	Type	Description
<b>payload_bsz</b>	required	u16	FIP variable payload size in bytes   Only significant for FIP <b>prod</b> and <b>cons</b> variable type.
<b>refreshment</b>	optional	boolean	Enable/Disable the production status byte (refreshement) for the FIP variable
<b>refresh_us</b>	optional	u32	Refreshment period in microseconds   Only significant for FIP <b>prod</b> variable type.
<b>promptness</b>	optional	boolean	Enable/Disable the promptness status for a FIP consumed variable   Only significant for FIP <b>cons</b> variable type.
<b>prompt_us</b>	optional	u32	Promptness period in microseconds   Only significant for FIP <b>cons</b> variable type.
<b>event</b>	optional	enum	Values: <ul style="list-style-type: none"> <li>• <b>none</b> No FIP event attached to the variable</li> <li>• <b>r_op</b> Read operation of the FIP variables by the gateway</li> <li>• <b>w_op</b> Write operation of the FIP variables by the gateway</li> <li>• <b>rw_op</b> Read and write operation of the FIP variables by the gateway</li> </ul>
<b>bus_arbiter</b>	optional	object	FIP master configuration container
<b>enable</b>	required	boolean	Enable/Disable bus arbiter (master) capability
<b>priority</b>	required	u8	Should be in [0..15] range (0: highest priority)
<b>cycles</b>	required	array	Array of Fip master macrocycles

Name	Status	Type	Description
<b>windows</b>	required	array	Array of BA windows inside a macrocycle
<b>type</b>	required	enum	Values: <ul style="list-style-type: none"> <li>• <b>periodic</b> Periodic BA window</li> <li>• <b>aperiodic_var</b> Aperiodic Variable BA window</li> <li>• <b>aperiodic_msg</b> Aperiodic Message BA window</li> <li>• <b>wait</b> Internal Resync waiting BA window</li> </ul>
<b>end_ustime</b>	optional	u32	End time of the aperiodic BA window in microseconds <div>  Only significant for <b>aperiodic_var</b>, <b>aperiodic_msg</b> and <b>wait</b> BA window types           <div>  The end time is relative to the first macrocycle operation           </div> </div>
<b>requests</b>	optional	array	 Only significant for <b>periodic</b> BA window type
<b>type</b>	required	enum	Values: <ul style="list-style-type: none"> <li>• <b>id_dat</b> FIP variable BA request</li> <li>• <b>id_msg</b> FIP message BA request</li> </ul>
<b>id</b>	required	u16	FIP identifier to request during a periodic BA window

## Chapter 6. Ping-Pong example

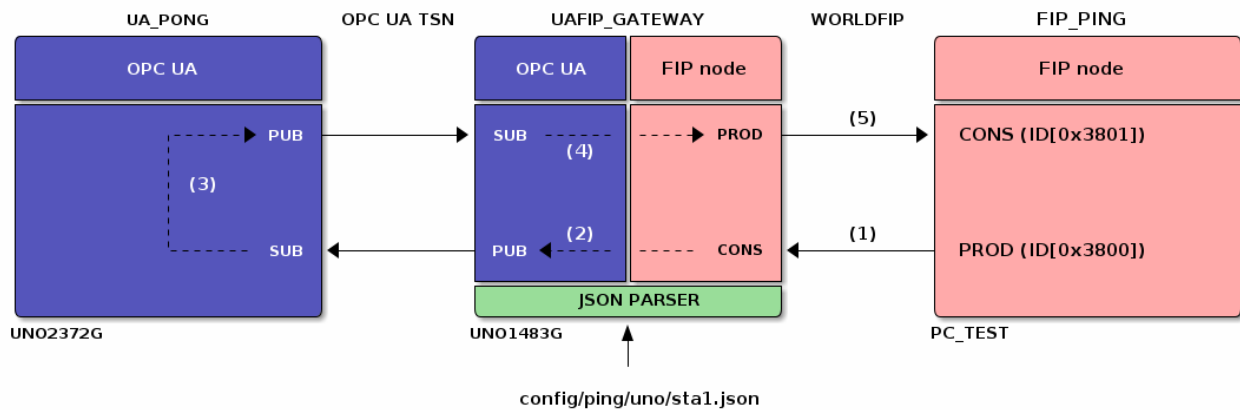


Figure 2. Ping/Pong Test (Principle Diagram)

The "ping-pong" example is a test between a FIP master node (BA) and an OPC-UA node (client) where an UAFIP TSN gateway acts as an intermediary between the two heterogeneous nodes.

1. The FIP node increments a *PING* counter (64-bit value) at each macrocycle and produces a `RP_DAT[0x3800]` FIP frame with the updated counter to the FIP network.
2. This FIP data is consumed by the UAFIP gateway and is transferred via an OPC-UA publisher to the Ethernet network.
3. A remote OPC-UA node (client) subscribes to the gateway's publisher and returns this value by republishing it (*PONG* counter).
4. The gateway in turn consumes this OPC value via a subscriber and produces a `RP_DAT[0x3801]` FIP frame with the updated *PONG* counter (64-bit value).
5. At each new macrocycle, the FIP node consumes the `RP_DAT[0x3801]` data [*PONG*] and compares it to the new value of the `RP_DAT[0x3800]` variable to be produced [*PING*].  
If the real time capacity of the whole system is maintained, the difference between the consumed and produced counters **should never exceed 1 unit** for the FIP node.



In order to avoid cycle losses due to initial conditions, the starting order of the different nodes is important!

You should start by launching UAFIP\_GATEWAY, then UA\_PONG and finally FIP\_PING.

### UAFIP\_GATEWAY

```
uno1483g@uno1483g:~/Documents/powerfip/tools/uafip_tsn_gateway$ sudo
./start_rt_gateway.sh "./uafipd -f ./config/ping/uno/sta1.json"
```



1. Here we use the `start_rt_gateway.sh` script to start the gateway, locks the process to a specific CPU, and sets the scheduling policy.
2. We configure the gateway using the `config/ping/uno/sta1.json` file (see its description [here](#))

### Console trace of the UAFIP TSN gateway started

```
[06-16 12:58:31.162685] fip => [info] [fip] device info
[06-16 12:58:31.162711] fip => [info]     index      : 1
[06-16 12:58:31.162726] fip => [info]     fsn       : 0x2873c623afaa
[06-16 12:58:31.162744] fip => [info]     vid       : 0x11aa
[06-16 12:58:31.162756] fip => [info]     did       : 0x1556
[06-16 12:58:31.162782] fip => [info]     ssvd      : 0x11aa
[06-16 12:58:31.162799] fip => [info]     ssdid     : 0x5811
[06-16 12:58:31.162810] fip => [info]     bar_cnt   : 2
[06-16 12:58:31.162820] fip => [info]     bar_bsz[0] : 4096
[06-16 12:58:31.162830] fip => [info]     bar_base[0] : 0xf6000000
[06-16 12:58:31.162840] fip => [info]     bar_bsz[1] : 33554432
[06-16 12:58:31.162852] fip => [info]     bar_base[1] : 0xf4000000
[06-16 12:58:31.162863] fip => [info]     irq_number : 69
[06-16 12:58:31.162880] fip => [info]     drv_version : 1.4.0
[06-16 12:58:31.162896] app => [info] ua-fip gateway: v1.0.0 - opc lib: v1.3.0 - fip lib: v1.1.0
[06-16 12:58:31.162912] app => [info] [ok] gateway configuration read
[06-16 12:58:31.217185] fip => [event] reset component (powerfip)
[06-16 12:58:31.286442] fip => [info] [ok] fip node load
[06-16 12:58:31.286778] fip => [info] [ok] fip node start
[2022-06-16 12:58:31.288 (UTC+0200)] info/session      SecureChannel 0 | Session "Administrator" | AddNode (i=15303)
: No TypeDefinition. Use the default TypeDefinition for the Variable/Object
[06-16 12:58:31.288880] app => [info] [ok] ua-fip gateway is active. press Ctrl + C to exit.
[06-16 12:58:31.299885] opc => [warn] AccessControl: Unconfigured AccessControl. Users have all permissions.
[06-16 12:58:31.299917] opc => [info] AccessControl: Anonymous login is enabled
[06-16 12:58:31.299935] opc => [info] AccessControl: x509 certificate user authentication is enabled
[06-16 12:58:31.299952] opc => [warn] Username/Password Authentication configured, but no encrypting SecurityPolicy.
This can leak credentials on the network.
[06-16 12:58:31.299971] opc => [warn] AcceptAll Certificate Verification. Any remote certificate will be accepted.
[2022-06-16 12:58:31.303 (UTC+0200)] info/userland   PubSub channel requested
[2022-06-16 12:58:31.303 (UTC+0200)] info/server      Open PubSub ethernet connection.
[2022-06-16 12:58:31.304 (UTC+0200)] info/userland   PubSub channel requested
[2022-06-16 12:58:31.304 (UTC+0200)] info/server      Open PubSub ethernet connection.
[06-16 12:58:31.310939] opc => [info] publisher task: thread priority is 78
[06-16 12:58:31.311081] opc => [info] publisher task: cpu_core=2
[06-16 12:58:31.311110] opc => [info] publisher task: thread id is 140584042329856
[06-16 12:58:31.312353] opc => [info] subscriber task: thread priority is 81
[06-16 12:58:31.312425] opc => [info] subscriber task: cpu_core=2
[06-16 12:58:31.312441] opc => [info] subscriber task: thread id is 140584033937152
[06-16 12:58:31.312519] opc => [info] [ok] publisher start (opc.eth://74-fe-48-42-c2-e4:8.3)
[06-16 12:58:31.312532] opc => [info] [ok] subscriber start (opc.eth://74-fe-48-30-39-eb:8.3)
[06-16 12:58:31.312805] opc => [info] TCP network layer listening on opc.tcp://UNO-1483G:4840/
```

## UA\_PONG

Then, launch the OPC node which will play the PONG role with the command below:

```
uno2372g@uno2372g:~/Documents/powerfip/tools/uafip_tsn_gateway/tools/ua_pong$ sudo
./ua_pong
```



Make sure that the settings defined in the `powerfip/tools/uafip_tsn_gateway/tools/ua_pong/test.h` header file match those of the UAFIP gateway.

### Console trace of the UA\_PONG example started

```
[06-16 14:18:44.443074] app => [info] [ok] app is active. press enter to close...
[2022-06-16 14:18:44.443 (UTC+0200)] info/session SecureChannel 0 | Session "Administrator" | AddNode (i=15303):
No TypeDefinition. Use the default TypeDefinition for the Variable/Object
[06-16 14:18:44.467316] opc => [warn] AccessControl: Unconfigured AccessControl. Users have all permissions.
[06-16 14:18:44.467397] opc => [info] AccessControl: Anonymous login is enabled
[06-16 14:18:44.467424] opc => [info] AccessControl: x509 certificate user authentication is enabled
[06-16 14:18:44.467451] opc => [warn] Username/Password Authentication configured, but no encrypting SecurityPolicy.
This can leak credentials on the network.
[06-16 14:18:44.467481] opc => [warn] AcceptAll Certificate Verification. Any remote certificate will be accepted.
[06-16 14:18:44.467565] opc => [info] SecureChannel 0 | Session "Administrator" | AddNode (i=50400): No TypeDefinitio
n. Use the default TypeDefinition for the Variable/Object
[2022-06-16 14:18:44.467 (UTC+0200)] info/userland PubSub channel requested
[2022-06-16 14:18:44.467 (UTC+0200)] info/server Open PubSub ethernet connection.
[2022-06-16 14:18:44.469 (UTC+0200)] info/userland PubSub channel requested
[2022-06-16 14:18:44.469 (UTC+0200)] info/server Open PubSub ethernet connection.
[06-16 14:18:44.482603] opc => [info] publisher task: thread priority is 78
[06-16 14:18:44.482825] opc => [info] publisher task: cpu_core=2
[06-16 14:18:44.482864] opc => [info] publisher task: thread id is 140238353356544
[06-16 14:18:44.485050] opc => [info] subscriber task: thread priority is 81
[06-16 14:18:44.485231] opc => [info] subscriber task: cpu_core=2
[06-16 14:18:44.485267] opc => [info] subscriber task: thread id is 140238344963840
[06-16 14:18:44.485382] opc => [info] pubsub_app task: thread priority is 75
[06-16 14:18:44.485676] opc => [info] pubsub_app task: cpu_core=3
[06-16 14:18:44.485714] opc => [info] [ok] publisher start (opc.eth://74-fe-48-30-39-eb:8.3)
[06-16 14:18:44.485739] opc => [info] [ok] subscriber start (opc.eth://74-fe-48-42-c2-e4:8.3)
[06-16 14:18:44.486290] opc => [info] TCP network layer listening on opc.tcp://uno2372g:4840/
```

## FIP\_PING

Finally, launch the FIP node which will play the PING role with the command below:

```
pctest@pctest:~/Documents/powerfip/tools/pwrifip_ping$ sudo ./pwrifip_ping -i 1 -n 0 -c
10000
```

Here we use the parameters:



- -i 1: FIP device index number 1
- -n 0: FIP node address 0 (.i.e: PING station)
- -c 10000: FIP macrocycle period of 10ms

*Console trace of the FIP\_PING example started*

```

06-16 16:01:39.804314] app => [info] [fip] device info
06-16 16:01:39.804393] app => [info]     index      : 1
06-16 16:01:39.804411] app => [info]     fsn        : 0xea3f4e704ea7
06-16 16:01:39.804427] app => [info]     vid        : 0x11aa
06-16 16:01:39.804464] app => [info]     did        : 0x1556
06-16 16:01:39.804488] app => [info]     sssvid     : 0x11aa
06-16 16:01:39.804511] app => [info]     ssdid      : 0x5811
06-16 16:01:39.804535] app => [info]     bar_cnt     : 2
06-16 16:01:39.804559] app => [info]     bar_bsz[0] : 4096
06-16 16:01:39.804571] app => [info]     bar_base[0] : 0xa6000000
06-16 16:01:39.804583] app => [info]     bar_bsz[1] : 33554432
06-16 16:01:39.804596] app => [info]     bar_base[1] : 0xa4000000
06-16 16:01:39.804607] app => [info]     irq_number  : 163
06-16 16:01:39.804623] app => [info]     drv_version : 1.4.0
06-16 16:01:39.804640] app => [info] test: v1.0.0 - pwrflib lib: v1.1.0
06-16 16:01:39.804659] ping => [info] [fip] node configuration
06-16 16:01:39.804675] ping => [info] [fip] bus arbiter infos
06-16 16:01:39.804690] ping => [info]     start      : 1306800us
06-16 16:01:39.804706] ping => [info]     election    : 77700us
06-16 16:01:39.804722] ping => [info] [fip] node init
06-16 16:01:39.877959] ping => [event] reset component (powerfip)
06-16 16:01:39.947186] ping => [info] [fip] node infos
06-16 16:01:39.947201] ping => [info]     cpu_id     : 0xea3f4e704ea72301
06-16 16:01:39.947204] ping => [info]     bss_bsz    : 6188B [max_bsz: 15500B]
06-16 16:01:39.947207] ping => [info]     gateway    : v1.9.0 [build date: 02/21/22]
06-16 16:01:39.947209] ping => [info]     firmware   : v1.2.0 [build date: 03/09/22]
06-16 16:01:39.947211] ping => [info]     driver     : v1.4.0 [build date: 06/15/22]
06-16 16:01:39.947214] ping => [info]     library    : v1.1.0 [build date: 03/09/22]
06-16 16:01:39.947221] ping => [info] [fip] node start
06-16 16:01:39.947538] ping => [info] [fip] ba start
06-16 16:01:39.947563] app => [info] app started. press enter to close...
06-16 16:01:41.255072] ping => [event] BA_STATE [=> idle] nok=0
06-16 16:01:41.332877] ping => [event] BA_STATE [=> active]
06-16 16:02:03.461167] ping => [info] => ping/pong: ok=4421 nok=0

```

If all goes well, the bus arbiter should start and the ping/pong **ok** counter start to increment. No loss of FIP macrocycle occurs while the **nok** counter remains at 0.



# Appendix A: JSON files examples

## A.1. Ping (FIP Node 0)

*config/ping/uno/sta0.json*

```
{
  "opc": {
    "publisher" : {
      "network_interface": "enp2s0",
      "url": "opc.eth://74-fe-48-30-39-eb:8.3",
      "publisher_id": "5670",
      "dataset_writer_id": "62541",
      "writer_group_id": "100",
      "publish_period_us": "250"
    },
    "subscriber" : {
      "network_interface": "enp2s0",
      "url": "opc.eth://74-fe-48-42-c2-e4:8.3",
      "publisher_id": "5671",
      "dataset_writer_id": "62541",
      "writer_group_id": "101"
    }
  },
  "fip": {
    "name": "FIP node 0",
    "address": "0",
    "segment": "0",
    "frame_type": "worldfip",
    "bitrate": "1mbps",
    "turn_around_us": "30",
    "silence_us": "150",
    "vars": [
      {
        "name": "[0x3800]",
        "type": "prod",
        "id": "0x3800",
        "payload_bsz": "8",
        "refreshment": "yes",
        "refresh_us": "7500",
        "event": "none"
      },
      {
        "name": "[0x3801]",
        "type": "cons",
        "id": "0x3801",
        "payload_bsz": "8",
        "refreshment": "yes",

```

```

    "promptness": "yes",
    "prompt_us": "7500",
    "event": "none"
  },
  {
    "name": "[0xFF00]",
    "type": "sync",
    "id": "0xFF00",
    "event": "r_op"
  },
  {
    "name": "[0xFF01]",
    "type": "sync",
    "id": "0xFF01",
    "event": "w_op"
  }
],
"bus_arbiter": {
  "enable": "yes",
  "priority": "1",
  "cycles": [
    {
      "windows": [
        {
          "type": "periodic",
          "requests": [
            {
              "type": "id_dat",
              "id": "0x3800"
            },
            {
              "type": "id_dat",
              "id": "0xFF00"
            }
          ]
        }
      ]
    },
    {
      "type": "wait",
      "end_ustime": "2500"
    },
    {
      "type": "periodic",
      "requests": [
        {
          "type": "id_dat",
          "id": "0x3801"
        },
        {
          "type": "id_dat",

```

```

        "id": "0xFF01"
      }
    ]
  },
  {
    "type": "wait",
    "end_ustime": "5000"
  }
]
}
]
}
}
}
}

```

## A.2. Pong (FIP Node 1)

*config/ping/uno/sta1.json*

```

{
  "opc": {
    "publisher" : {
      "network_interface": "enp7s0",
      "url": "opc.eth://74-fe-48-42-c2-e4:8.3",
      "publisher_id": "5671",
      "dataset_writer_id": "62541",
      "writer_group_id": "101",
      "publish_period_us": "250"
    },
    "subscriber" : {
      "network_interface": "enp7s0",
      "url": "opc.eth://74-fe-48-30-39-eb:8.3",
      "publisher_id": "5670",
      "dataset_writer_id": "62541",
      "writer_group_id": "100"
    }
  },
  "fip": {
    "name": "FIP node 1",
    "address": "1",
    "segment": "0",
    "frame_type": "worldfip",
    "bitrate": "1mbps",
    "turn_around_us": "30",
    "silence_us": "150",
    "vars": [
      {
        "name": "[0x3800]",

```

```
    "type": "cons",
    "id": "0x3800",
    "payload_bsz": "8",
    "refreshment": "yes",
    "promptness": "yes",
    "prompt_us": "7500",
    "event": "none"
  },
  {
    "name": "[0x3801]",
    "type": "prod",
    "id": "0x3801",
    "payload_bsz": "8",
    "refreshment": "yes",
    "refresh_us": "7500",
    "event": "none"
  },
  {
    "name": "[0xFF00]",
    "type": "sync",
    "id": "0xFF00",
    "event": "w_op"
  },
  {
    "name": "[0xFF01]",
    "type": "sync",
    "id": "0xFF01",
    "event": "r_op"
  }
]
}
```

## Appendix B: Glossary of acronyms

<b>BA</b>	Bus Arbiter
<b>FIP</b>	Factory Instrumentation Protocol
<b>OPC-UA</b>	Open Platform Communications Unified Architecture
<b>TSN</b>	Time-Sensitive Networking

# Appendix C: Revision History

Revision	Changes	Authors	Date
1.0.0	First version	MC	2022-06-21