4x PDI Builder

Release 6.12.27

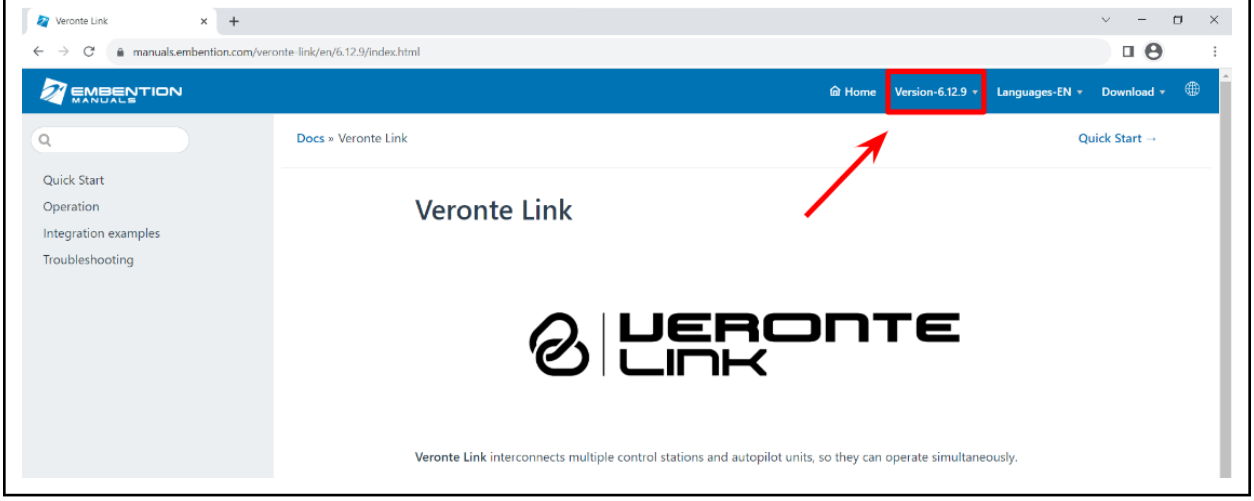
Embention

2023-12-04

CONTENTS

4X PDI Builder is an application for modifying, generating and uploading PDI files for **Arbiters**, the autopilot managers of the redundant **Veronte Autopilot 4x**.

Warning: Select your version before reading any user manual for software. The following image shows where to select a version from any Embention user manual.



CHAPTER

ONE

QUICK START

4x PDI Builder is the main configuration tool to adapt **Arbiters** for a specific system, being its main function to decide which **Autopilot 1x** controls the vehicle at every moment. **4x PDI Builder** includes:

- Communications: through general purpose CAN bus, RS-232, RS-485 and GPIOs.
- Arbitration criterion: **Arbiters** decide which **Autopilot 1x** controls the vehicle according to the criterion decided by the user.

All Veronte devices, including **Arbiters**, are configured with PDI files.

1.1 Download

Once the **Veronte Autopilot 4x** has been purchased, a GitHub release should be created for the customer with the application. To access to the release and download the software, read the Releases section of the **Joint Collaboration Framework** manual.

1.2 Installation

To install **4x PDI Builder** on Windows just execute "4xVerontePDIbuilder.exe" and follow the setup wizard instructions.

Tip: In case of having any problem with the installation, please disable the antivirus and the Windows firewall. Disabling the antivirus depends on the antivirus software. To disable the firewall, go to **Control Panel** → **System and Security** → **Windows Defender Firewall** and then, click on **Turn windows Defender Firewall on or off**.

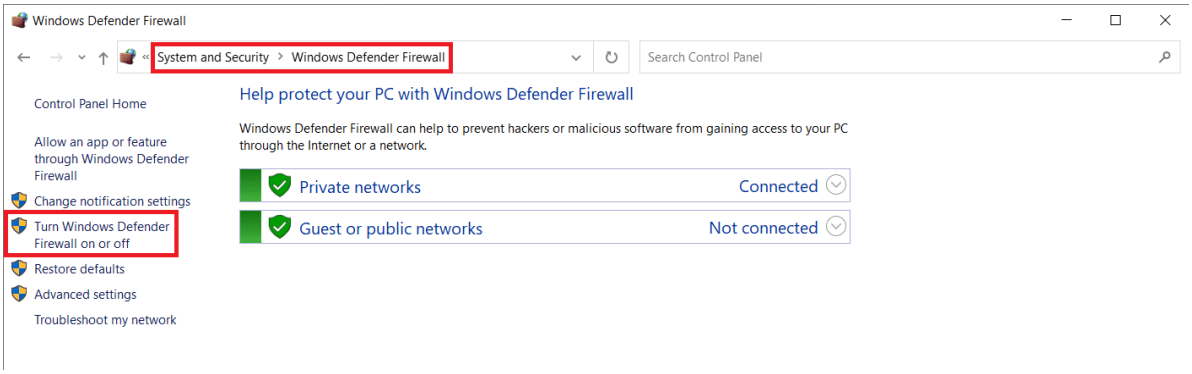


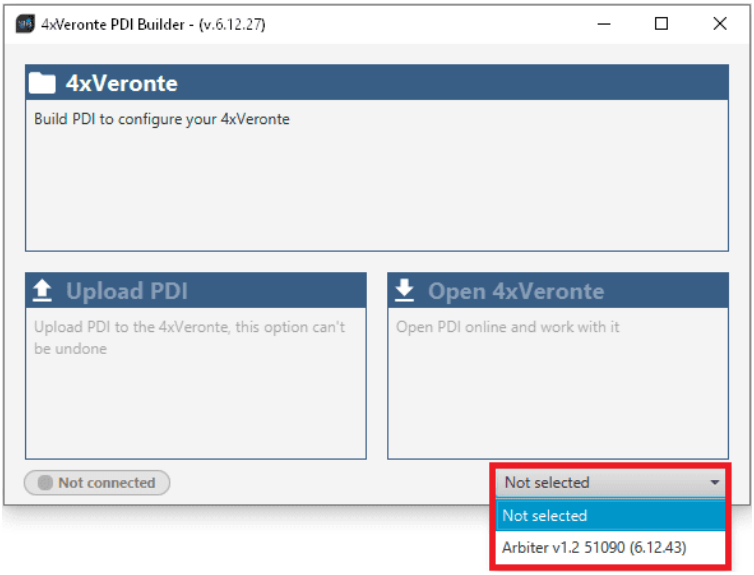
Fig. 1: **Windows Defender Firewall**

CHAPTER

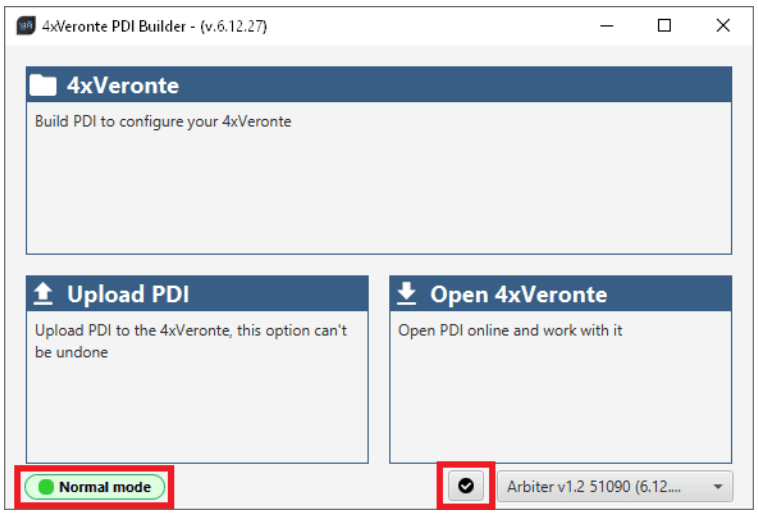
TWO

CONFIGURATION

Once the installation is finished, connect the **Management Board** of the **4x** (using the **Connector 4**) with Veronte Link. Then, open **4x PDI Builder** and select the **Arbiter** to configure.



If the **Veronte Autopilot 4x** is correctly connected, **4x PDI Builder** will display the **mode** in which the connected unit is. In addition, a **PDI error button**  will appear:



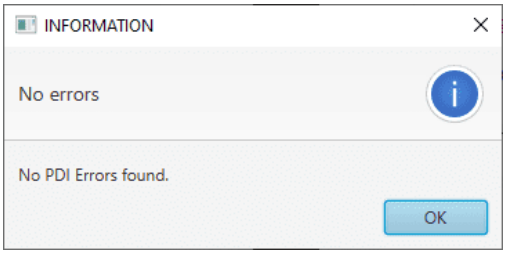
- **4x mode**: The 4x unit should appear in **Normal mode**, as shown in the figure above, or **Maintenance mode**.

  It can also appear as **Maintenance mode (loaded with errors)** or **Normal mode - Disconnected**.

Note: **Maintenance mode (loaded with errors)** appears when something is wrong with the configuration.

For more information, see *Troubleshooting section* of this manual.

- **PDI Errors** button: The user can check if the connected unit has PDI Errors by simply clicking on it. If there are no errors, the following message is displayed:



The user can access now to three configuration options:

- **4xVeronte.** It allows the user to work with **offline** configurations. A previously exported 4x PDI configuration can be opened and modified. It is also possible to build a new one from the default configuration.
- **Upload PDI.** A previously exported 4x PDI configuration can be imported to the linked **Arbiter** (PDI files are transferred from computer to **Arbiter**).
- **Open 4x Veronte.** It opens the configuration (the PDI files) loaded in the **Arbiter** (PDI files are transferred from **Arbiter** to computer). Then, the user can modify them online.

Note: PDI (Parameter Data Item) files are configuration files. They are used by modular control with improved version management. These PDI files are split in 2 folders. Each folder contains several .xml files:

  - **setup:** contains the configuration of the **Arbiter**. All control system and parameters are stored here.
  - **xsd:** holds .xsd files. An XSD file is a definition file specifying the elements and attributes that can be part of an XML document. This ensures that data is properly interpreted, and errors are caught, resulting in

appropriate XML validation.

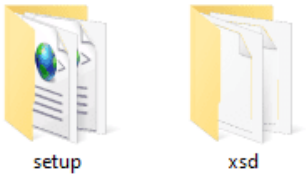


Fig. 1: **PDI files**

Warning: Users should never delete, replace or modify **xsd** files.

Finally, click on **'Open 4xVeronte'** to open the configuration and start editing online. When opened, the unit changes to Maintenance mode.

Note: If an offline configuration is opened, it will be possible to select the hardware version to work with on the marked button:

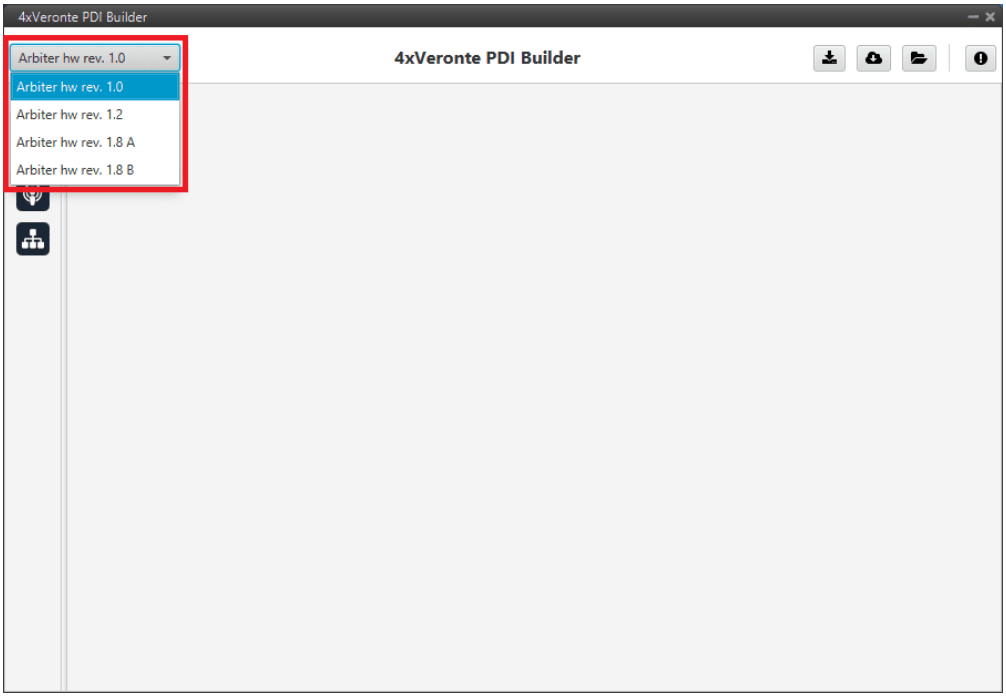
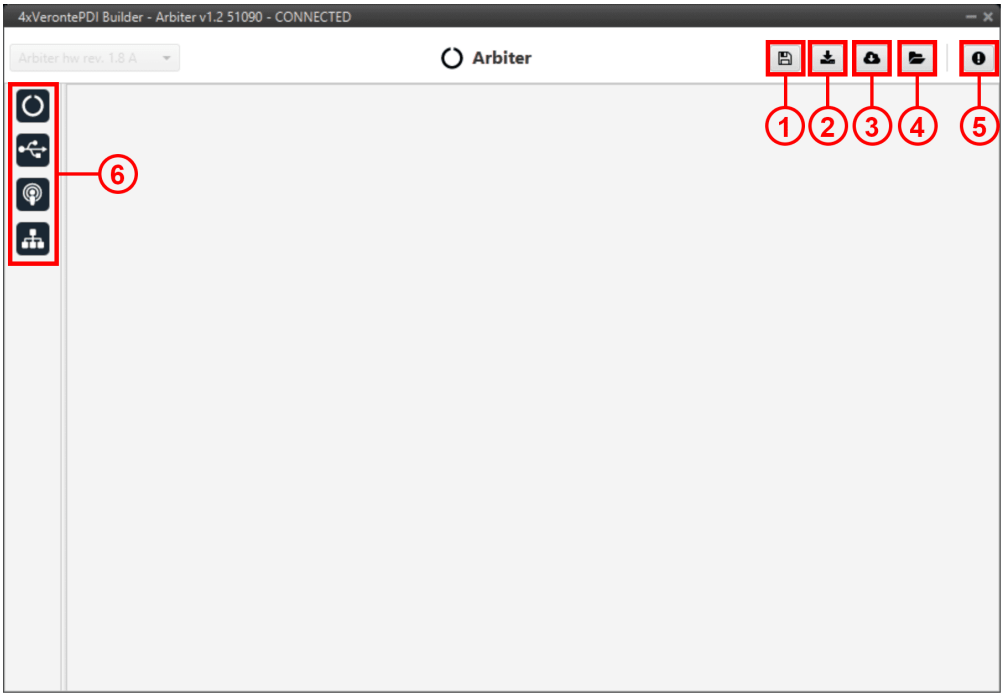


Fig. 2: **Hardware version selector**

- Versions `1.8 A` and `1.8 B` are arbiters A and B of version 1.8.

The different 'buttons' that can be seen in the initial menu are explained below.



1. **Save PDI.** After changes are done, press here to apply changes.

   After saving any changes, the Arbiter will **RESET** and the 4x PDI Builder software will **close**.

   Danger: As the Arbiter is **reseted**, it is **not advisable to save changes during flight tests**.

   Note: This button will only appear if an **Arbiter** is connected. When working offline this button will not be available.

2. **Export PDI.** After modifying a configuration, press here to save the configuration in the local storage (both folders with the PDI files are downloaded). Users can store this configuration in an empty folder or in the folder where the previously imported configuration is stored. With the latter option, the "original" configuration will be overwritten by the one with the new changes.
3. **Import PDI from repository.** The user can import a configuration file from the GitHub repository and modify it. After that, if the **Save PDI** button (1) is pressed, this configuration will be uploaded to the connected **arbiter**.
4. **Import PDI from local storage.** The user can import a configuration file from the local storage and modify it. After that, if the **Save PDI** button (1) is pressed, this configuration will be loaded into the connected **arbiter**.
5. **Feedback.** Users can report a problem they have encountered by creating an issue in their own 'Joint Collaboration Framework'. The 'Download' button downloads a zipped folder with the current arbiter configuration and more information needed for Embention to resolve the issue. It is advisable to attach this folder when creating the issue.

   Note: The user's **'Joint Collaboration Framework'** is simply a *Github repository for each customer*.

   If the user has any questions about this **Joint Collaboration Framework**, please read its user manual or contact sales@embention.com.

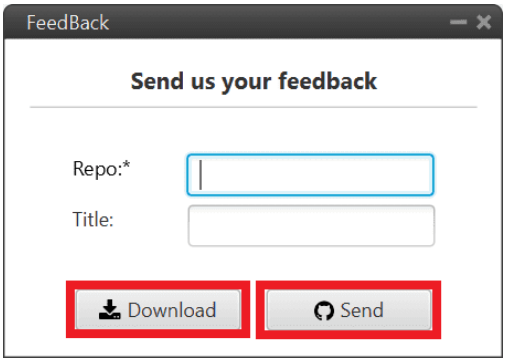


Fig. 3: **Feedback window**

6. These are the different functions of the **arbiter**. They are explained in the following menus.

-  *Arbiter*
-  *Input / Output*
-  *Communications*
-  *Arbitration*

2.1 Arbiter

2.1.1 GPIO

In this window, each individual GPIO (General Purpose Input/Output) behavior can be configured:

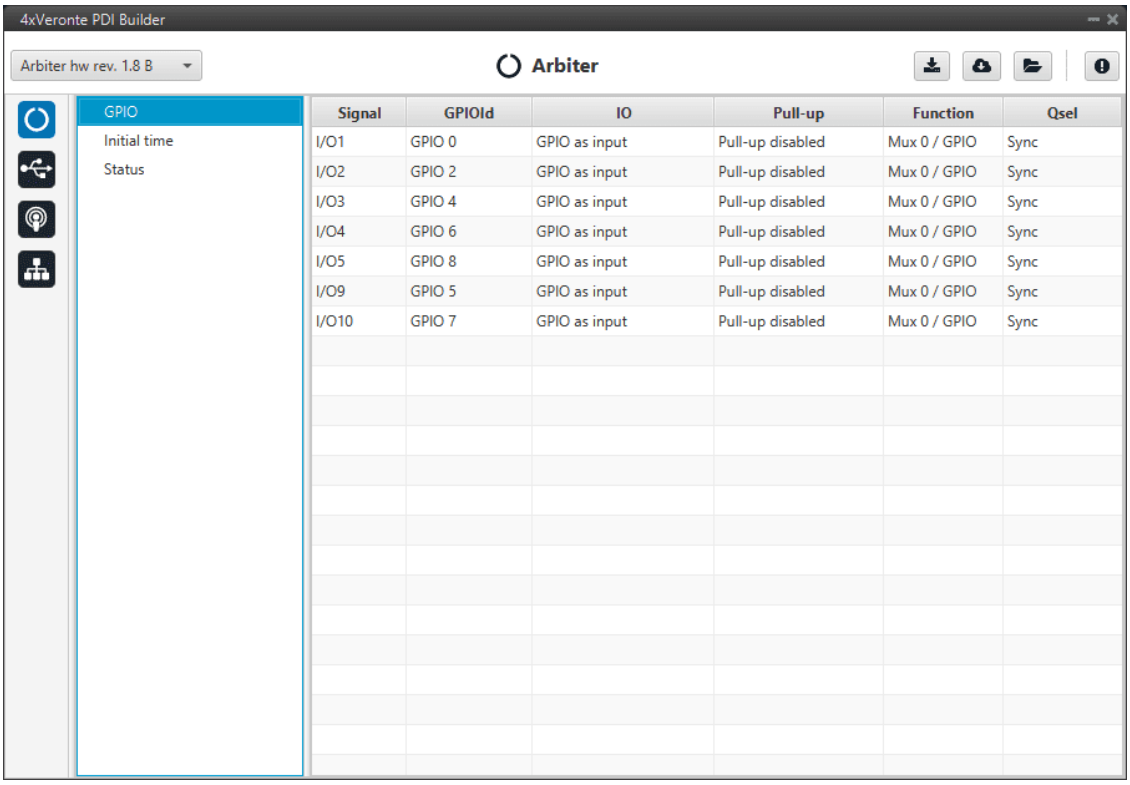


Fig. 4: **GPIO section**

- **Signal**: Pin ID as described in Hardware installation -> Arbiter connector pinout of the **Autopilot 4x** user manual.
- **GPIOId**: GPIO ID of the microcontroller.
- **IO**: Define GPIO as an input or ouput.
- **Pull-up**: Enable or disable the pull-up resistance.
- **Function**: Mux 0/GPIO, Mux 1, Mux 2 or Mux 3. These are the different functionalities that the GPIO can have, depending on the multiplexer.
- **Qsel**: This is the "input qualification", it is used to control how the value of a GPIO is evaluated. The available options are:
  - **Sync**: The value is taken as the time it is checked (synchronously). This is the default mode of all GPIO pins.
  - **3 Samples**: The value is checked 3 times and the value is only changed when the 3 times are the same.
  - **6 Samples**: Same as **3 samples**, but checking 6 times instead of 3.
  - **ASync**: No checks are performed. It is used when it is not used as GPIO.

2.1.2 Initial Time

If any **Autopilot 1x** has not power supply, the **Management Board** will detect it as a fault. The **Initial Time** parameter is the margin of time from the moment when the **Management Board** is powered up until it starts cheking power supplies.

With this parameter it is possible to connect each **1x** one by one after the **Management Board**.

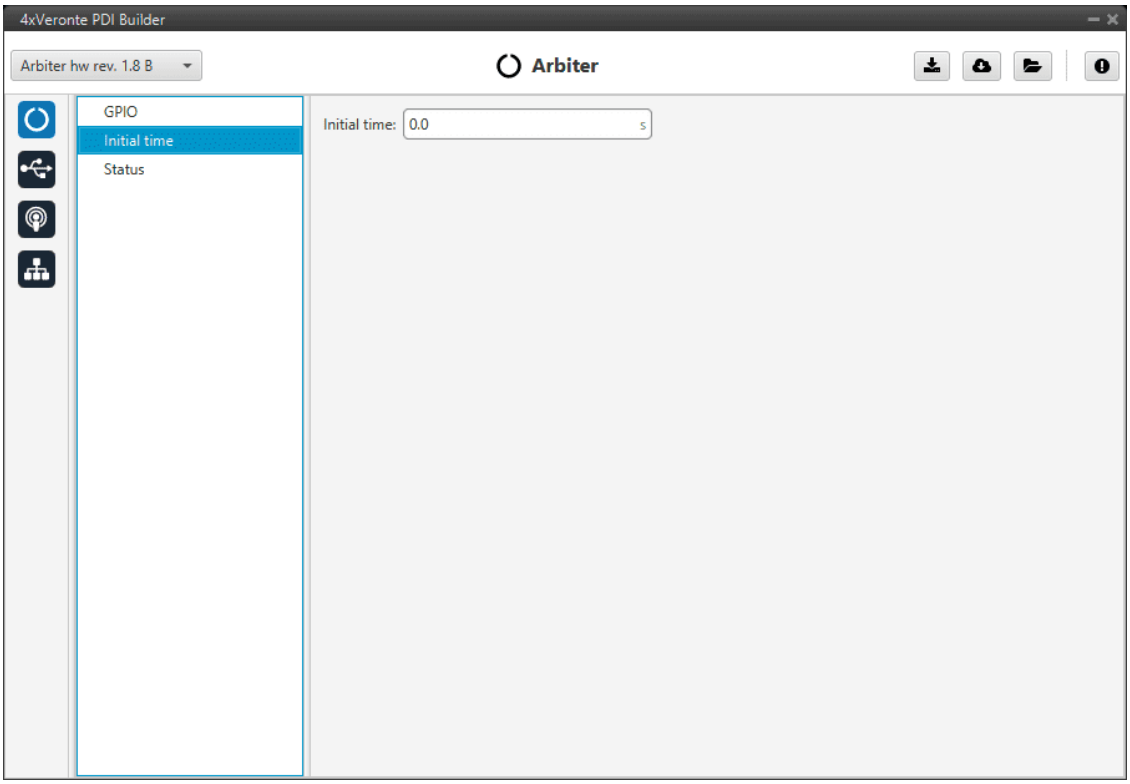


Fig. 5: **Initial Time section**

2.1.3 Status

- **Enable VCP Status Message:** enables the periodic sending of the status message that **Veronte Link** uses to recognise an **arbiter**.
- **Period:** establishes the desired period to send repeatedly the status message.

Note: VCP is the Veronte Communication Protocol. To know more, read the VCP user manual.

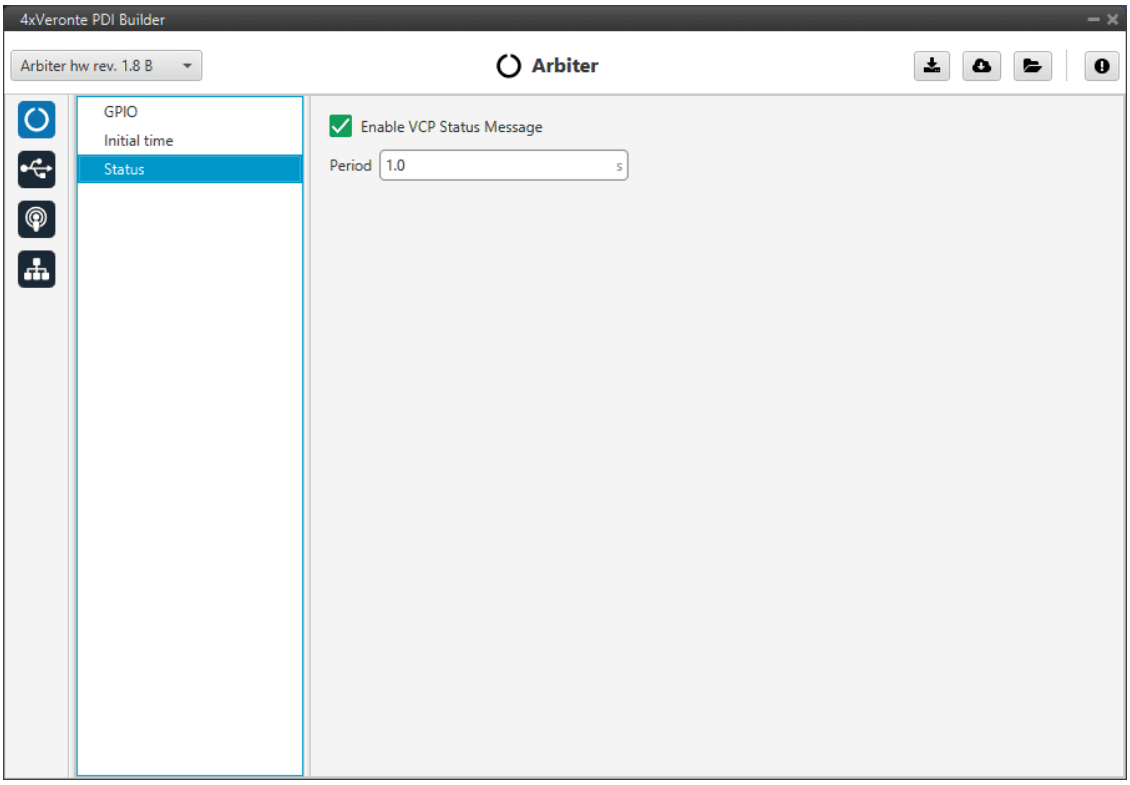


Fig. 6: **Status section**

2.2 Input / Output

2.2.1 SCI

Arbiters can use up to three serial SCIs (Serial Communication Interface). Serial ports A, B and C parameters can be edited in this menu to fit the serial protocol requirements.

Note: SCI A corresponds to RS-232 port A (RS232-1), SCI B to RS-232 port B (RS232-2) and SCI C to RS-485.

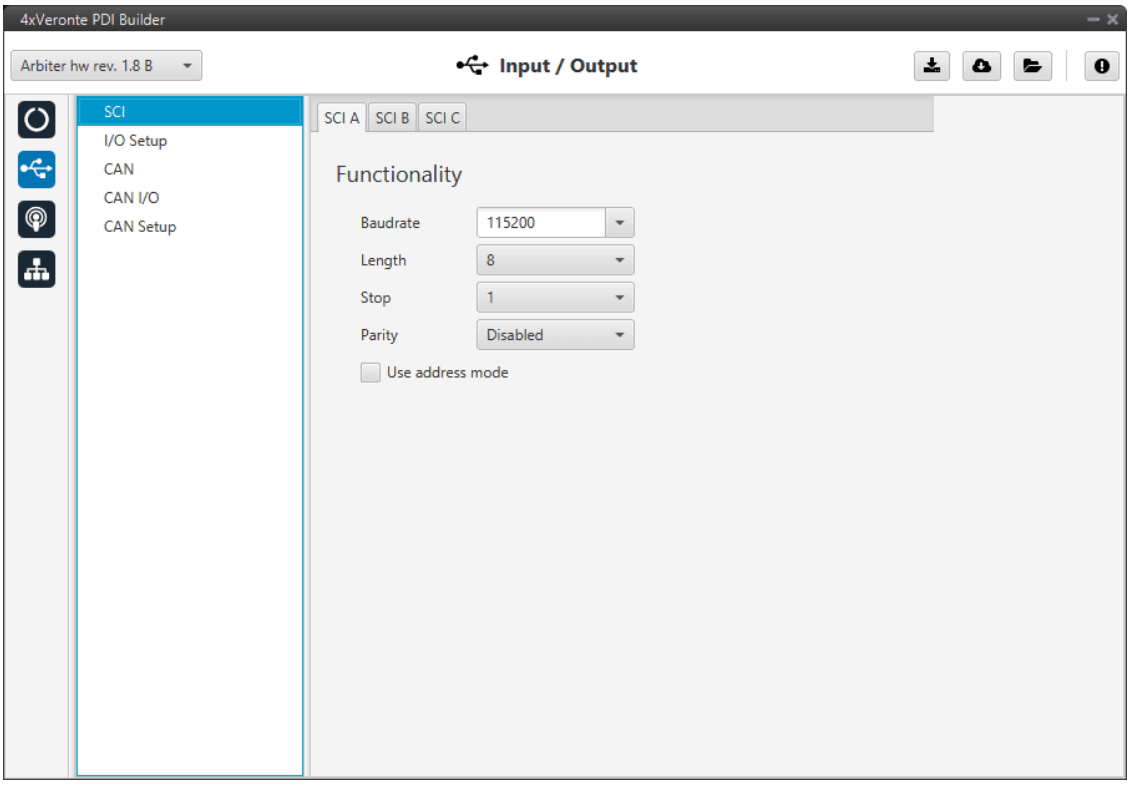


Fig. 7: **SCI section**

- **Baudrate.** How fast data is sent over a serial line.
- **Length.** Number of data bits for each character: 4 to 8 bits.
- **Stop.** Number of stop bits sent at the end of each character: 1, 1.5 or 2.
- **Parity.** Method to detect errors during transmission. When parity is used with a serial port, an extra data bit will be sent with each data character. The bits of each character (including parity bit) will be even or odd according to parity mode (odd, even or disabled).
- **Use address mode.** 9-bit data framing uses the bit typically associated with parity error detection to identify address messages. Sent serial data that does not have the address bit set will be ignored (unless the device had previously identified an address message associated with it). This option can be disabled or enabled.

2.2.2 I/O Setup

In this panel the user can establish the relationship between a determined signal with a I/O port. This allows users to configure external sensors, custom messages, etc.

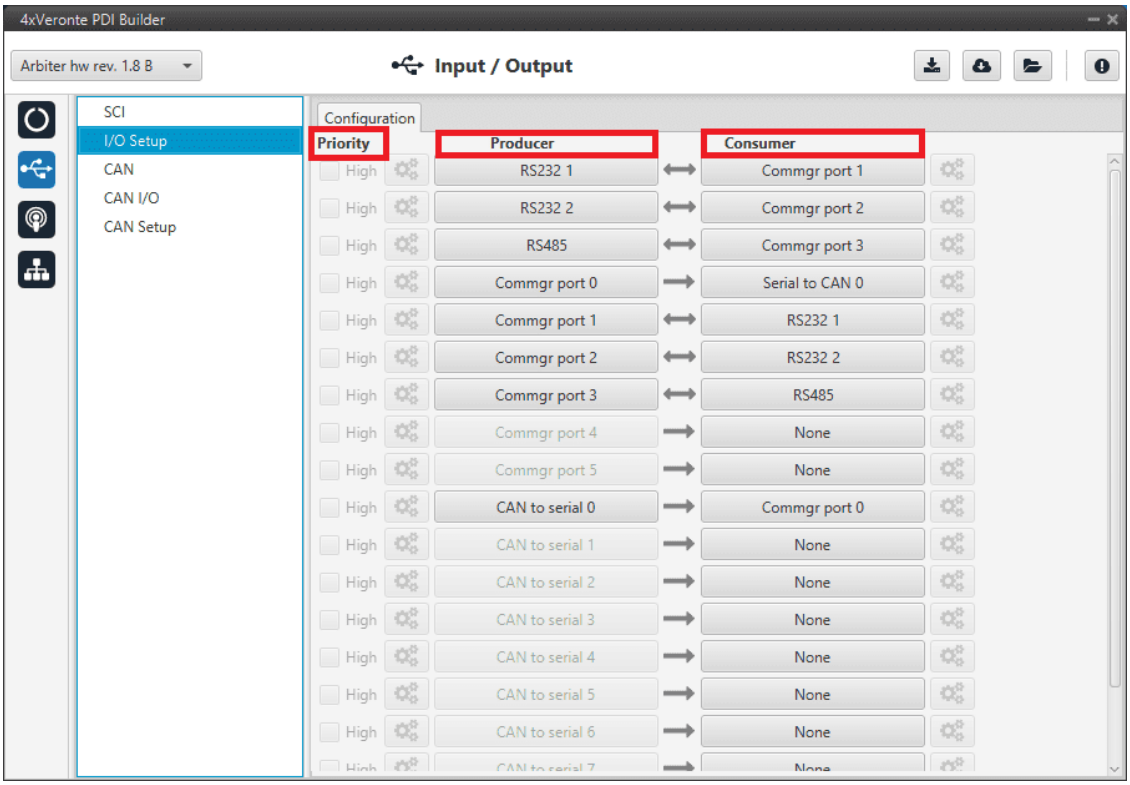


Fig. 8: **I/O setup section**

- **Priority.** Certain connections between I/O ports can be marked with high priority with this checkbox. If it is enabled, they will run at high frequency: 1000 Hz.
- **Producer.** Functions to create and send messages.
- **Consumer.** Functions to receive and parse messages.

Firstly, user has to configure the **Producer** selecting the I/O port or information to use. Later, user has to configure the **Consumer** by clicking on an element, a new window will be displayed to select an item. The relationship between them can be unidirectional (Bind $\rightarrow$) or bidirectional (Bind Bidirectional $\leftrightarrow$), the last enables a port to receive or send information.

The following I/O ports are available:

Field	Description
RS232-1	Serial Port 232 A
RS232-2	Serial Port 232 B
RS485	Serial Port 485
Commgr port	COM Manager ports send and receive VCP messages. This is the protocol used by Veronte products to communicate For more information on VCP, read its user manual.
CAN to serial / Serial to CAN	Serial to CAN sends serial streams over a CAN Bus / CAN to serial undoes the transformation 'Serial to CAN'
CAN wrapper for serial transmission / Serial CAN unwrapper	CAN wrapper sends CAN streams over a serial Bus / Serial CAN unwrapper undoes this transformation

Warning: By default, RS232 1-2 are set up for **Arbiter** configuration. If these connections are removed, it will not be possible to communicate with the **Arbiter** in **Normal mode**. In this situation, force the **arbiter** into **Maintenance mode** in order to recover the communication.

2.2.3 CAN

A CAN (Controller Area Network) Bus is a robust standard communication protocol for vehicles widely used in the aviation sector. The **Management Board** has two CAN buses that can be configured independently.

The structure of a CAN message can be seen in the following image:

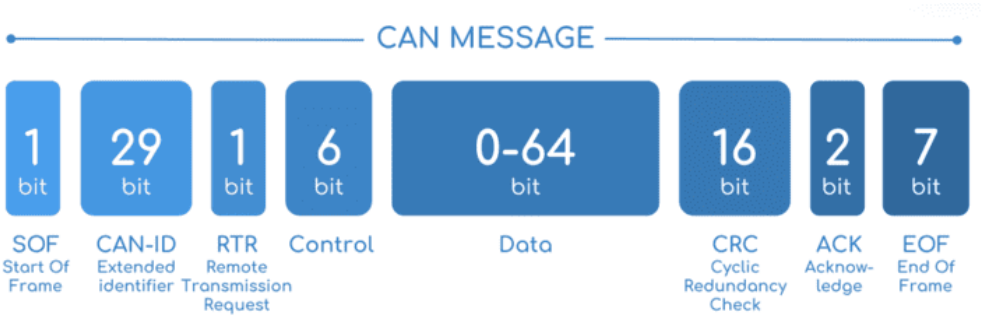
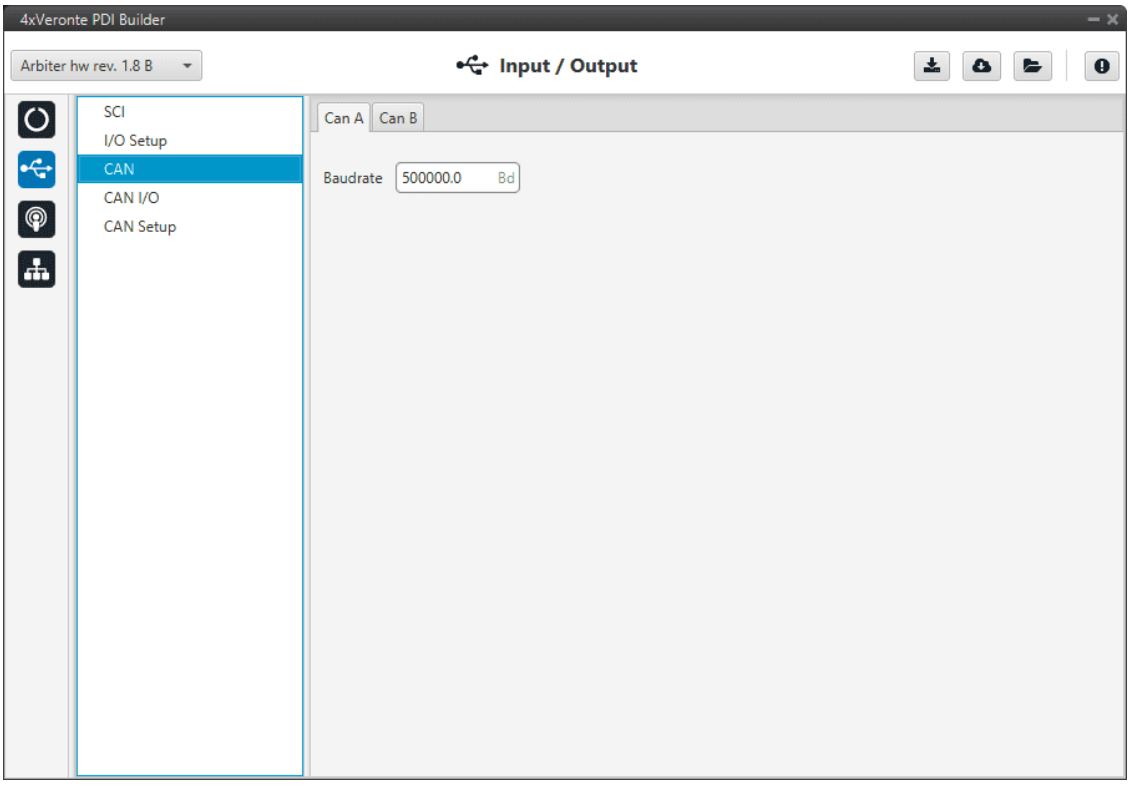


Fig. 9: **CAN message structure**

Only the ID is introduced in the system, the rest of the message layout is already coded. The data field is built by the user to send, and parsed when received.

For more information on the CAN Bus protocol, see CAN Bus protocol section of the 4x Software manual.

The baudrate of both CAN buses can be configured in the **Baudrate** box.



2.2.4 CAN I/O

2.2.4.1 Configuration

This menu allows the configuration of CAN inputs and outputs.

> **Warning:** In CAN, in Low state the specified period is not guaranteed, but it is guaranteed with High state.
>
> However, only those **messages that are critical for external devices** should be set as **high priority**, as this may disrupt the proper functioning of the **arbiter**.

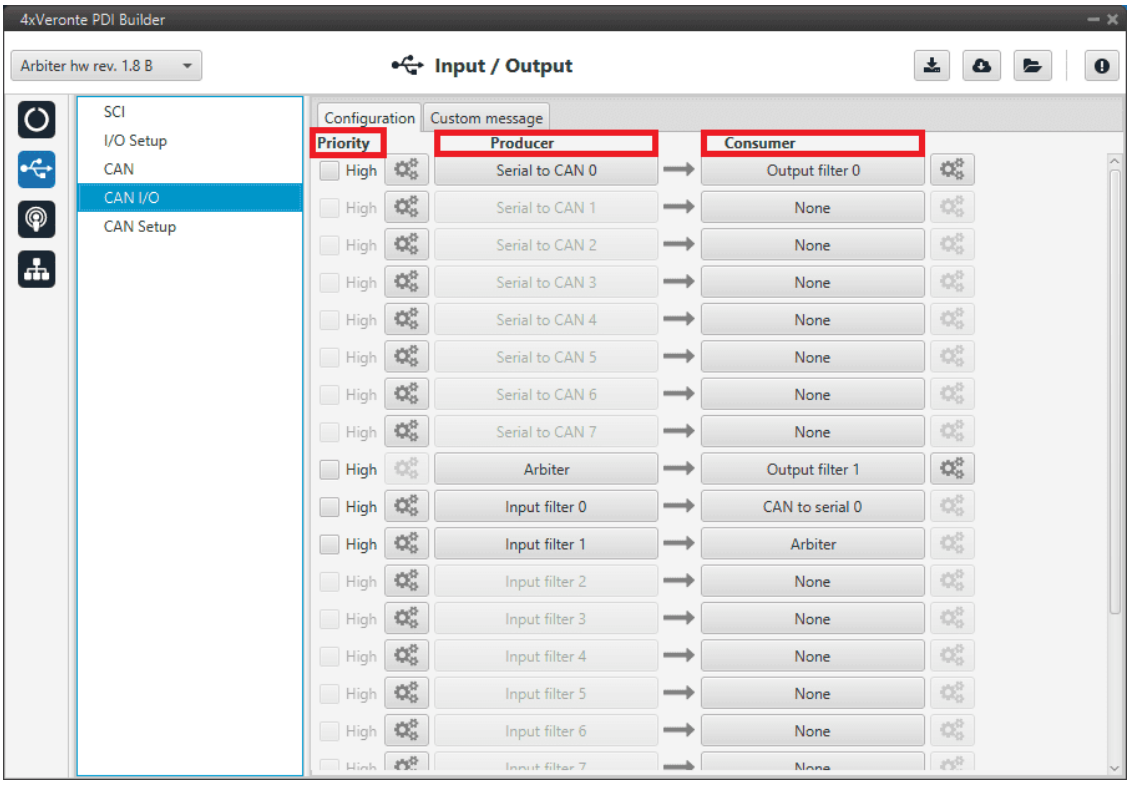


Fig. 10: **CAN configuration section**

In this menu, the user can find the same columns as in the *I/O Setup menu*.

Producers:

- **Arbiter.** It sends a **status** and a **score** mesage.

  To know more, read 4x Software Manual -> Arbitration messages.

- **Serial to CAN.** Serial messages through CAN output, it has to be connected to I/O Setup consumer. It can be configured by clicking on  , a pop-up window will appear:

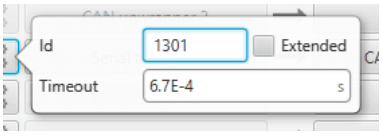


Fig. 11: **Serial to CAN configuration**

  - **Id.** CAN Id must be set and it is used to identify messages. The value set has to be decimal format.
  - **Extended.** If it is enabled, the frame format will be 'Extended', i.e. with a 29-bit identifier. Otherwise, the frame format 'Standard' (11-bit identifier) is set by default.
  - **Time out.** This is the threshold time between receptions to consider that it is not being received correctly.

- **Input Filter.** Those CAN messages received in one filter can no longer be received in subsequent filters. The following parameters need to be configured by clicking on  :

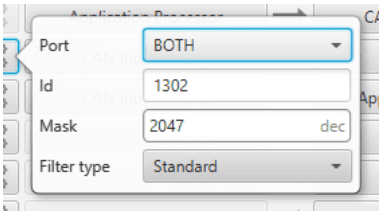


Fig. 12: **CAN Input Filter configuration**

- **Port.** It is required to configure the CAN bus from which it listens, the user can choose between *CAN A, CAN B* or *BOTH*.
- **Id.** CAN Id must be set and it is used to identify messages. The value set has to be decimal format.
- **Mask.** A CAN Id mask can be set to filter messages. The mask defines the bits that should match.

  For example, to admit standard Ids (11 bits) from 8 to 11 (100 to 111 in binary) the user should set the mask to binary 11111111100, that is 2044 in decimal.

  > **Warning:** Make sure that mask is set properly to be able to receive the desired CAN messages.

- **Filter type.** The available options are *Standard* (frame format with a 11-bit identifier), *Extended* (frame format with a 29-bit identifier) and *Both*.

- **Custom messages.** CAN custom messages transmission. They are configured in the next section, see *Custom Messages*.
- **CAN unwrapper.** This undoes the 'CAN wrapper' action, it has to be connected to *I/O Setup* **consumer**.

Consumers:

- **Arbiter.** It sends **ready** and **arbitration** messages.

  To know more, read 4x Software Manual -> Arbitration messages.
- **CAN to serial.** This undoes the 'Serial to CAN' action, it has to be connected to *I/O Setup* **producer**.
- **Output filter.** CAN output filters. The user can choose between *CAN A*, *CAN B* or *BOTH* in the **configuration button**  .

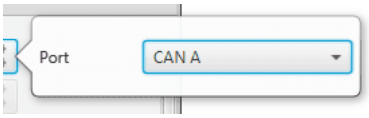


Fig. 13: **Output filter configuration**

- **Custom messages.** CAN custom messages reception. They are configured in the next section, see *Custom Messages*.
- **CAN GPIO consumer:** CAN message from Veronte Autopilot 1x or 4x for GPIO inputs. An example of how to implement it can be found in *Integration Examples -> GPIO command*.
- **CAN wrapper.** CAN message through serial output, it has to be connected to *I/O Setup* **producer** (CAN wrapper for serial transmission).

Warning: The **Input Filter** connected to the **Arbiter consumer** must have a mask that allows all arbitration CAN IDs.

2.2.4.2 Custom messages

The user chooses the variables to be sent/received through CAN buses. The following elements can be configured:

- **TX Ini.** It is used to configure transmitted messages that are only sent once at the beginning of the operation (sent when the **arbiter** boots up). They can be used to initialize some devices.
- **TX.** Configuration for transmitted messages.
- **RX.** Configuration for the reception messages (where they are stored).

Warning:

- The **maximum capacity** of a **CAN message** is 64 bits (8 bytes), so to send more information it must be divided into several messages.
- Each **arbiter** has a **CAN limitation** of **40 TX** messages, **40 TX Ini** messages and **80 RX** messages. In addition, the following limits apply:
  - Maximum number of **vectors** (fieldset): **104**
  - Maximum number of **fields: 2000**

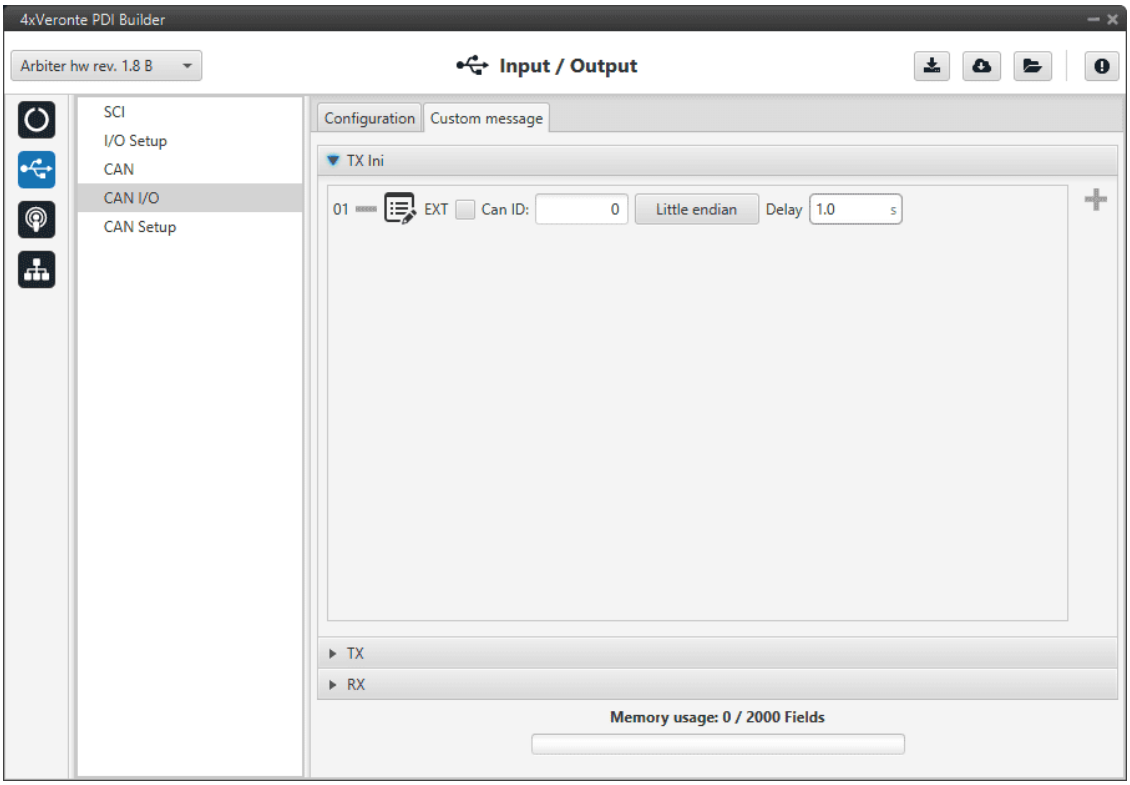


Fig. 14: **CAN Custom Message section**

Since this section works in a similar way to the CAN Custom Message configuration in the 1x PDI Builder software, the explanation to configure the telemetry messages via CAN can be found in the CAN Setup -> Input/Output section of the **1x PDI Builder user manual**.

2.2.5 CAN setup

Main screen to configure baudrate and reception mailboxes of each CAN Bus.

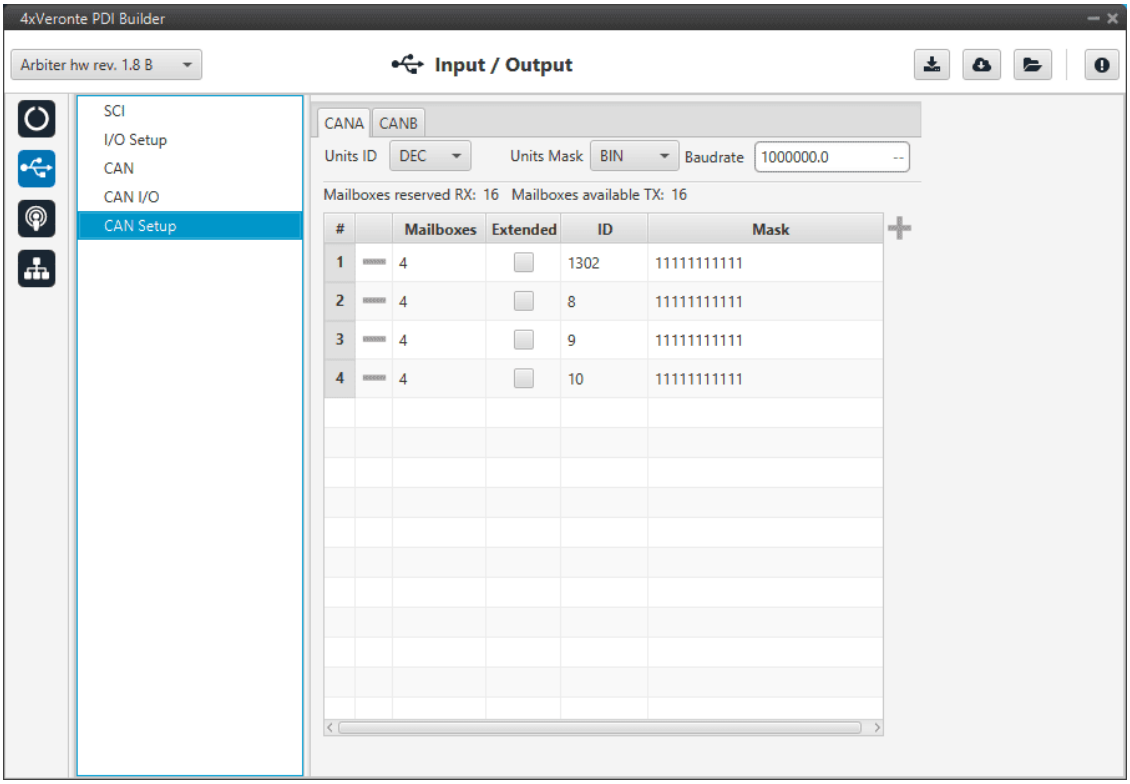


Fig. 15: **CAN Setup section**

More information about Mailboxes can be found in the Mailboxes section of the **1x PDI Builder** user manual.

2.3 Communications

Ports configuration allows the user to configure which communication ports (Commgr Ports in *I/O setup*) will be used for communication. When using the Route feature, **arbiters** can be configured to route VCP messages for an external Veronte device with a known address (ID) through a given port.

Note: To know more about VCP messages, read the VCP manual.

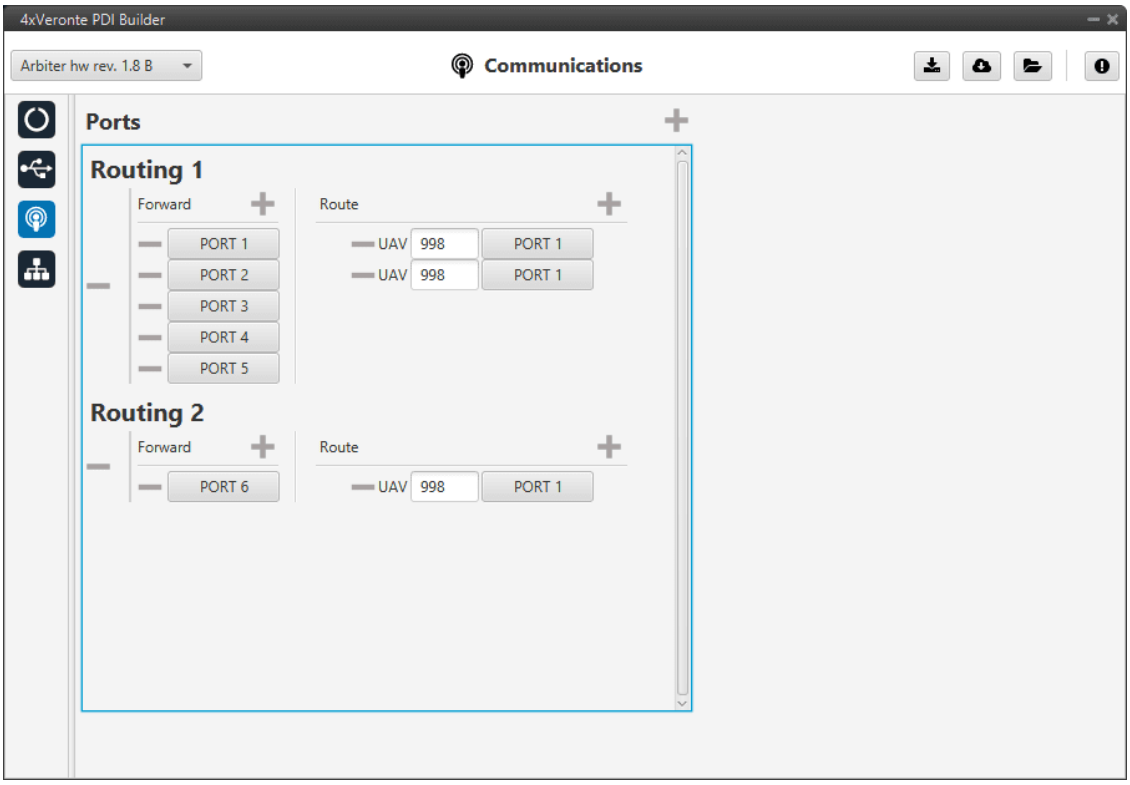


Fig. 16: **Communications menu**

Each port can be configured with the following options:

- **Forward.** Any message generated by this unit (i.e. Telemetry or response messages to certain commands) will be sent through these ports.
- **Route.** Any message received at any Commgr Port with the defined address will be re-sent through the defined port. It is possible to route several addresses through the same port, but is not possible to route the same address through several ports. Only the first configured port will be used. Routing also applies to messages generated by the unit for the defined address.

Note: The same port cannot be used as Forward and Route at the same time.

It is possible to define up to 4 routing setups, which can be switched using the Ports action of **1x PDI Builder**. Routing 1 will always be selected by default when booting an **arbiter**.

Warning: An incorrect port configuration can disable USB communications. If this happens, the **arbiter** will not be detected through Veronte Link. If this is the case, please visit *Troubleshooting -> Maintenance mode* of this manual.

2.4 Arbitration

The arbitration algorithm in **Autopilot 4x** is based on a **scoring system**. Each autopilot must send continuously a set of arbitration variables that will be used by the arbiter in order to calculate the score for each unit. Then, based on the scores and the current arbitration mode, the arbiter will choose to keep the current selected autopilot, or switching to one of the other units.

A **Score** is a 32 bit, single precission, floating-point value. This parameter is first computed resulting in a range between **0** and **0xFFFFFFFF**, where **0** is a perfect score (*Score'*). To achieve a better understanding, *Score'* is converted to a value comprehended between 0 an 100 (*Score*), being **100** the best possible score in the end.

Scores are calculated using the **arbitration variables** received from each autopilot at their dedicated addresses. After receiving the value, the following formula is used to compute the score for their respective unit:

$$score'_i = \frac{\sum_j^N min\left[\left(\frac{x_{i,j}-\mu_j}{d_{j,max}}\right)^2, 1\right] \cdot w_j}{\sum_j^N w_j}$$

$$score_i = 100 \cdot (1 - score'_i)$$

Where:

j	Variable index
$x_{i,j}$	*j* variable from autopilot *i*
μ_j	*j* variable reference
$d_{j,max}$	Maximum allowed error
w_j	Weight of *j* variable
N	Number of variables
$score'_i$	Score' of unit *i*
$score_i$	Score of unit *i*

Tip: The Arbitration weights should be used to increase or decrease the relevance that a certain arbitration variable has over the calculation of the score.

Any variable in Veronte can be used as an arbitration variable. Depending on the platform, operation, application etc. the more relevant variables can be selected for its use as arbitration references.

2.4.1 Config

2.4.1.1 Absolute Arbitration Variables

Absolute arbitration variables are indicators that are **inherently good** or **bad**, and so they are added directly to the score.

Examples of absolute arbitration variables are **Link Quality**, **GNSS accuracy** or warnings such as **Sensors error** or **Position not fixed**.

2.4.1.2 Relative Arbitration Variables

Relative arbitration variables are **not inherently good** or **bad**, and hence need to be compared against the other autopilots in order to calculate its score contribution.

The contribution to the score from a relative arbitration variable will be its **Deviation** from the **Average** of the same variable from each autopilot.

Examples of relative arbitration variables are **Attitude**, **Position**, measurements from sensors, etc.

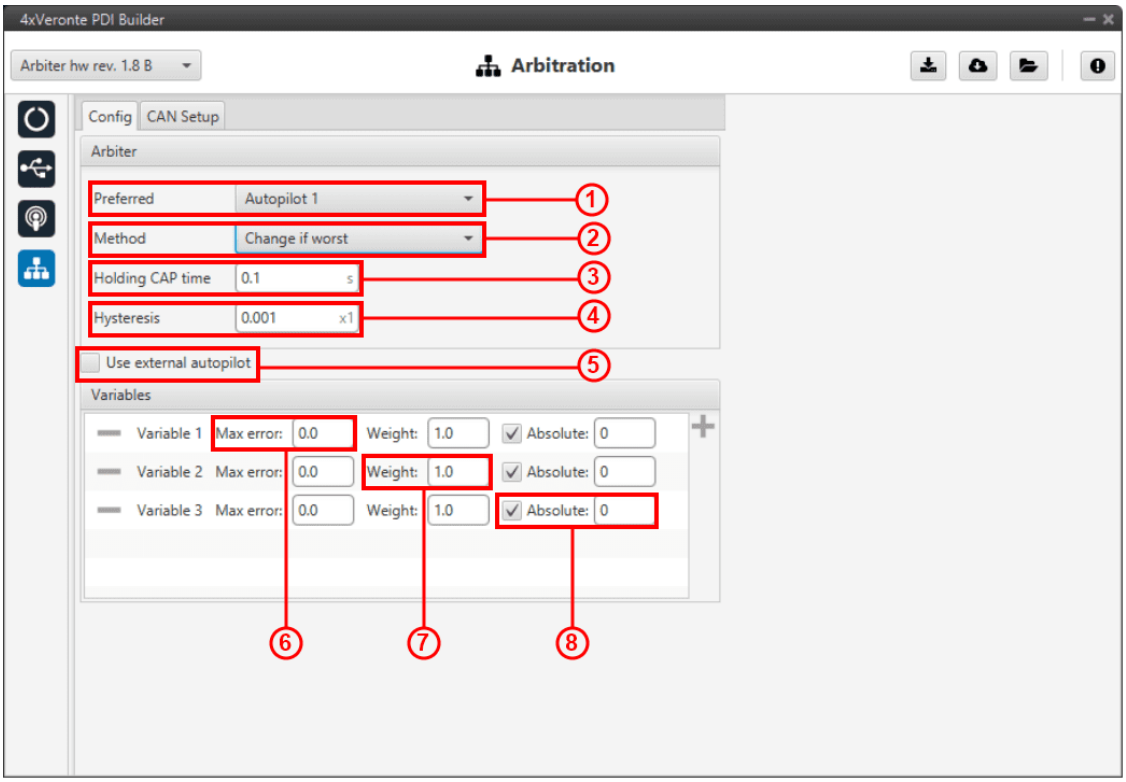
2.4.1.3 Arbitration Example

Autopilot	Var. Nº	Veronte Variable	Type	$x_{i,j}$	μ_j	w_j	Relative score	$Score'_i$	$Score_i$
1	1	Roll	Relative	0.12	0.096	1	0.054	0.043	95.67
	2	Pitch	Relative	0.30	0.283	1	0.027		
	3	GNSS 1 Accuracy	Absolute	1.7		0.01	0.004		
2	1	Roll	Relative	0.10	0.096	1	0.0011	0.0039	99.60
	2	Pitch	Relative	0.28	0.283	1	0.0011		
	3	GNSS 1 Accuracy	Absolute	1.9		0.01	0.005		
3	1	Roll	Relative	0.07	0.096	1	0.071	0.046	95.39
	2	Pitch	Relative	0.27	0.283	1	0.017		
	3	GNSS 1 Accuracy	Absolute	1.5		0.01	0.0036		

In the above example, **AP2** is considered to be **the best**, since it has the **highest total score**. Even though its **GNSS accuracy** is the worst of all 3, its values for pitch and roll are the ones with the lower deviation from the mean.

2.4.1.4 Config menu

In Config menu the parameters of the arbiter algorithm are adjusted.



1. **Preferred.** The preferred autopilot will be chosen in case of a score draw. **Fixed while ok** mode will always select this autopilot first.
2. **Method.** The method of arbitration can be chosen by the user. The available options are:
   - **Always best.** It chooses always the best autopilot.
   - **Change if worst.** The arbiter will only switch if the currently selected autopilot has the worst score. In that case, it will switch to the one with the best score.
   - **Round robin control.** Using the **Holding CAP** time parameter, the **arbiter** will periodically switch between autopilots. This mode is meant for testing purposes only.
   - **Fixed.** Arbitration is disabled and one autopilot is selected. In this mode **Autopilot 4x** will behave as an **Autopilot 1x**.
     - **Fixed 0.** Autopilot 1 is selected.
     - **Fixed 1.** Autopilot 2 is selected.
     - **Fixed 2.** Autopilot 3 is selected.
   - **Fixed while ok.** This mode does not take into account scores. In this mode, the **Preferred** autopilot will be selected by default. A switch will only happen if the current autopilot is considered **Dead**
3. **Holding CAP time.** Amount of time needed from last switch in order to allow a new switch.
4. **Hysteresis.** When comparing scores, the difference between them needs to be bigger than this proportional value, in order to assess scores. The difference is proportional to the score of the selected autopilot.

   i.e. If current selected autopilot is the number 1, arbitration mode is **Always best**, hysteresis is **0.5** and score for AP 1 is **0.3**, AP 2 will need a score lower than **0.15** in order to be selected.
5. Enable arbitration of **external autopilot**.

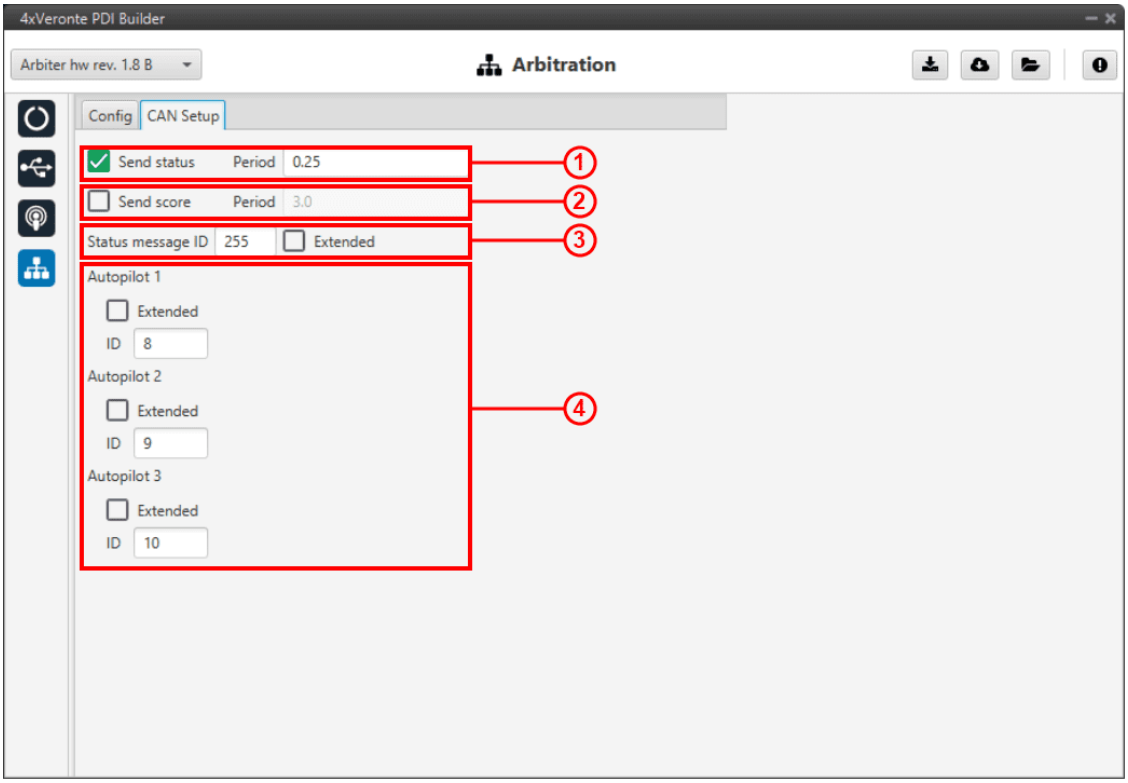
6. **Variable - Max error.** Arbitration maximum error for each variable.
7. **Variable - Weight.** Arbitration weight for each variable.
8. **Variable - Absolute.** If it is enabled, it will indicate that it is an absolute variable. The value set here will be used to compare the variables in order to choose the best autopilot.

Click on ➕ to add variables and ➖ to delete them. Currently, the **maximum** amount of **arbitration variables** supported is **32**.

The number for each variable is assigned with the **CAN I/O** configuration in the *arbiter consumer*. With this consumer, arbitration variables are selected with their corresponding numbers.

2.4.2 CAN Setup

Here users can configure the received CAN Ids for each one of the 3 possible **Autopilots 1x** that are sending data to **arbiters**. Therefore, to send data from any of them to **arbiters**, these Ids must be specified.



1. **Send status.** The user can enable status sending and set a status message period.
2. **Send score.** Score sending can be enabled and a period of scoring messages can be set.
3. **Status message ID.** CAN ID to which Status and Score messages.
    - **Extended.** If enabled, the frame format will be this, 'Extended', i.e. with a 29-bit identifier. Otherwise, the frame format 'Standard' (11-bit identifier) is set by default.
4. CAN IDs used for the reception of autopilot 4x arbitration messages for each AP.
    - **Extended.** If enabled, the frame format will be this, 'Extended', i.e. with a 29-bit identifier. Otherwise, the frame format 'Standard' (11-bit identifier) is set by default.

CHAPTER

THREE

INTEGRATION EXAMPLES

3.1 GPIO command

The following are the steps to send a GPIO command from the **Veronte Autopilot 1x**, receive it at **Arbiter** and process it, so that **Arbiter** carries out the command. GPIO Command is very similar to PWM command with a few exceptions.

> **Warning:** For the reception of CAN messages, Mailboxes need to be configured accordingly.

1x PDI Builder side

1. Go to **Input/Output** menu → **CAN Setup** section → **Configuration tab**.

   Connect a **CAN GPIO remote** producer to an **Output filter** consumer:

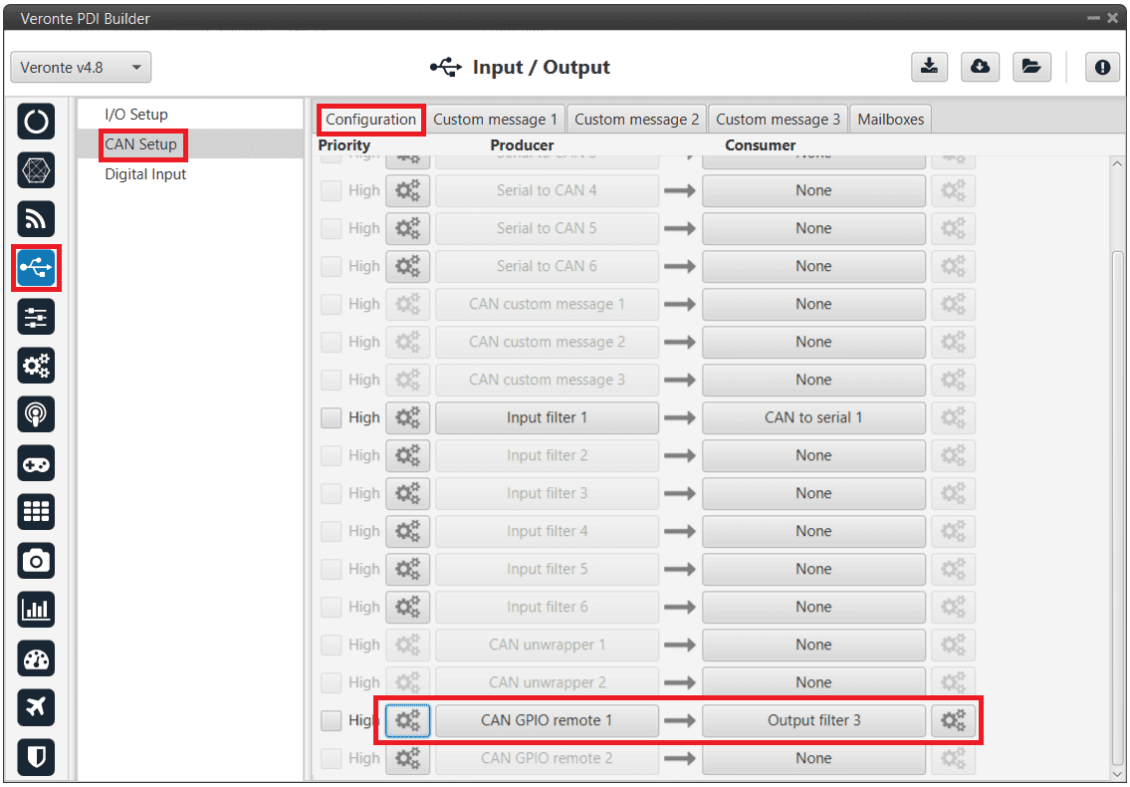


Fig. 1: **GPIO Command - 1x CAN Setup**

CAN GPIO remote must be configured. For more information on its configuration, see 1x PDI Builder manual -> CAN Setup.

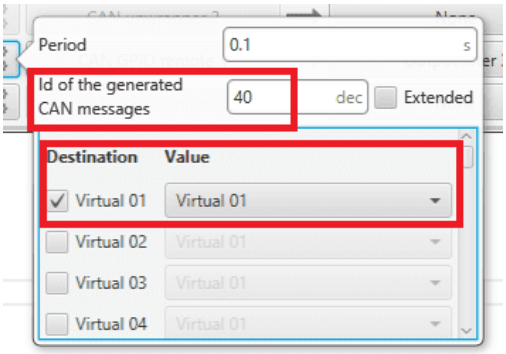


Fig. 2: **GPIO Command - 1x CAN Setup configuration**

2. Go to **Automations** menu. GPIO must be activated using an **Output action**:

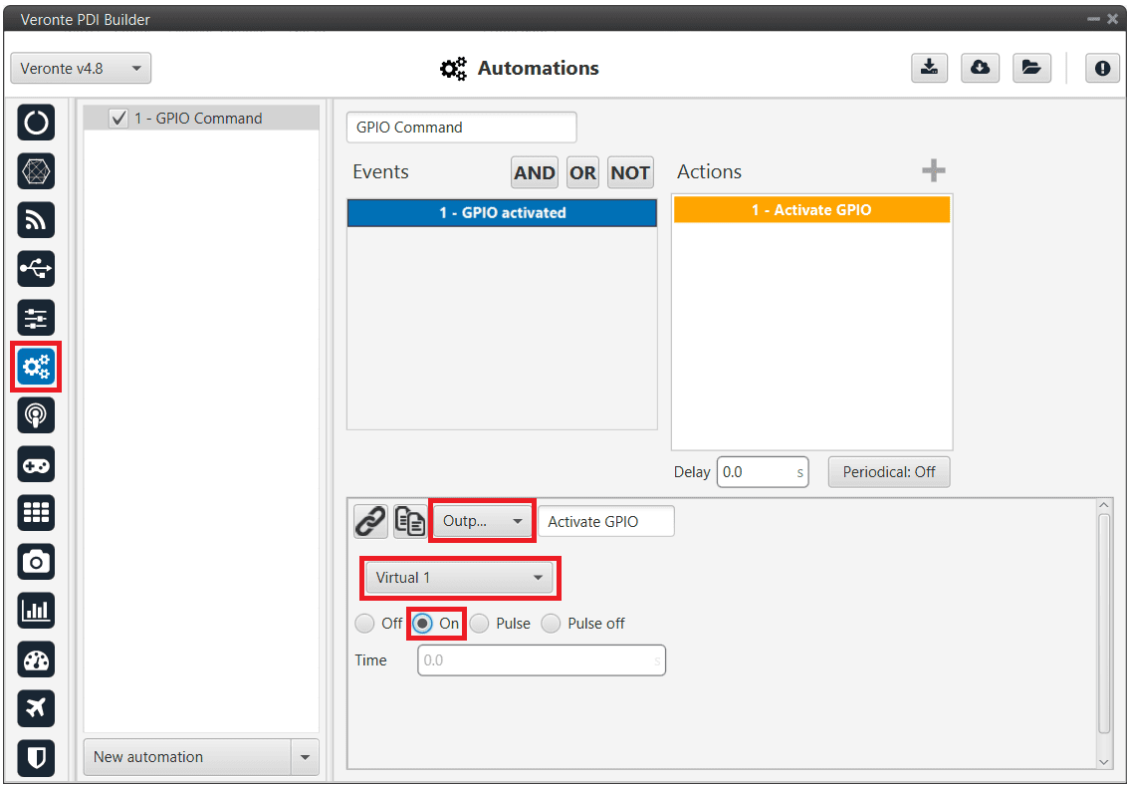


Fig. 3: **GPIO Command - 1x Automation configuration**

4x PDI Builder side

3. Go to **Input/Output** menu → **CAN I/O** section → **Configuration tab**.

   Finally, connect an **Input filter** to a **CAN GPIO consumer**.

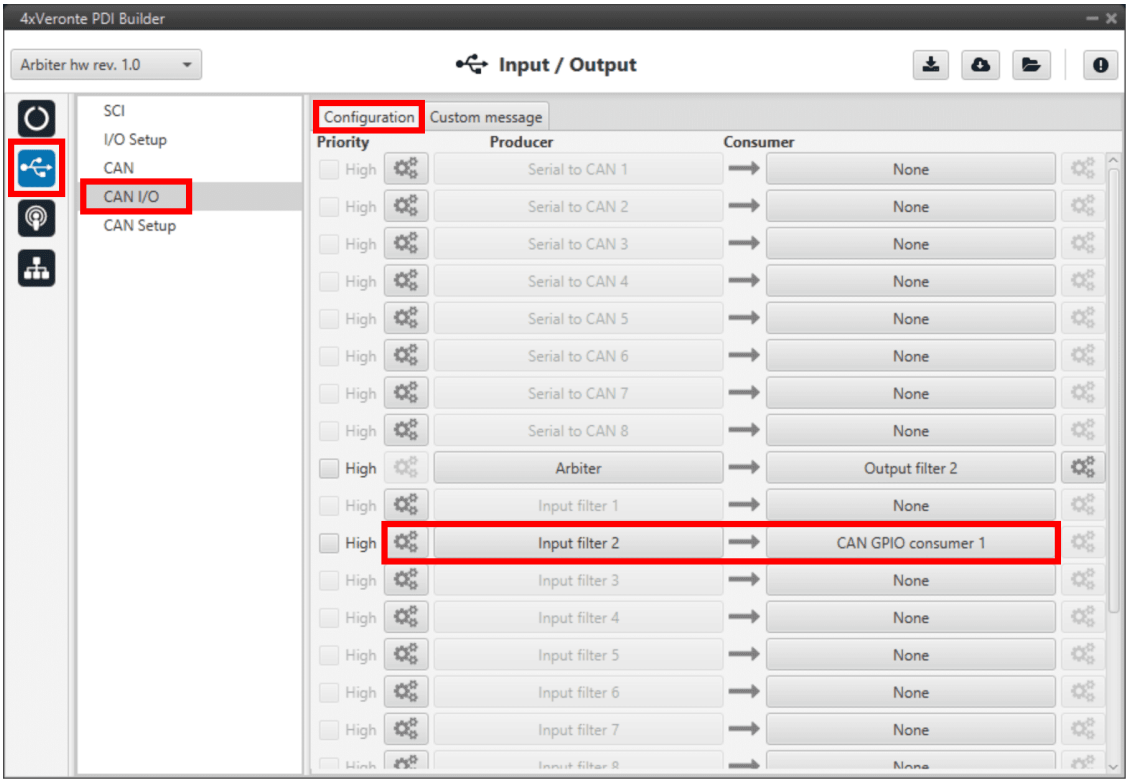


Fig. 4: **GPIO Command - CAN I/O**

The Id must match the one configured in the **1x PDI Builder** Output filter:

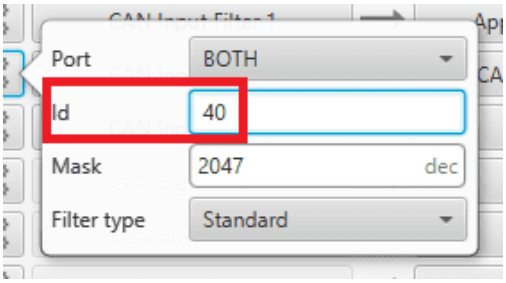


Fig. 5: **GPIO Command - Input filter configuration**

CHAPTER

FOUR

TROUBLESHOOTING

4.1 Maintenance mode

The user can simply enter maintenance mode with **4x PDI Builder** by clicking on the “Normal mode” button in the initial menu. Press the same button to return to “normal mode”.

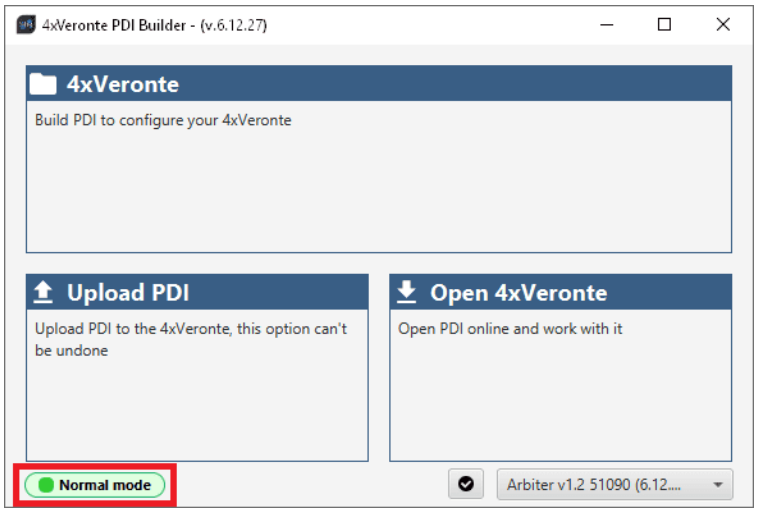


Fig. 1: **Press here to enter in maintenance mode**

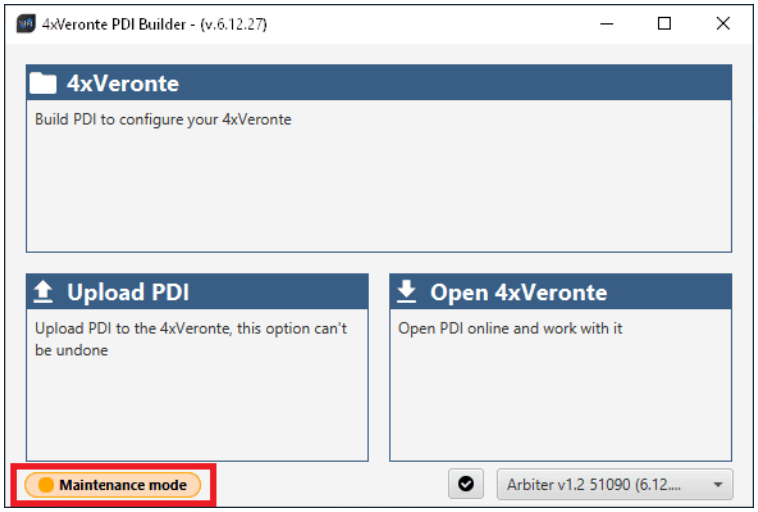


Fig. 2: **Press here to exit from maintenance mode**

4.2 Maintenance mode (loaded with errors)

The following error message may appear when trying to save a change or import a configuration.

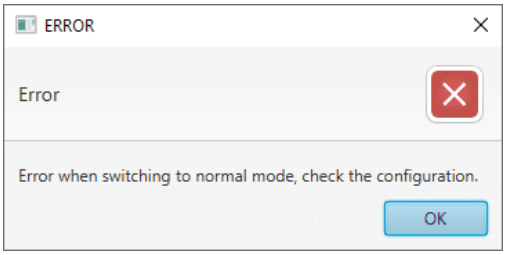
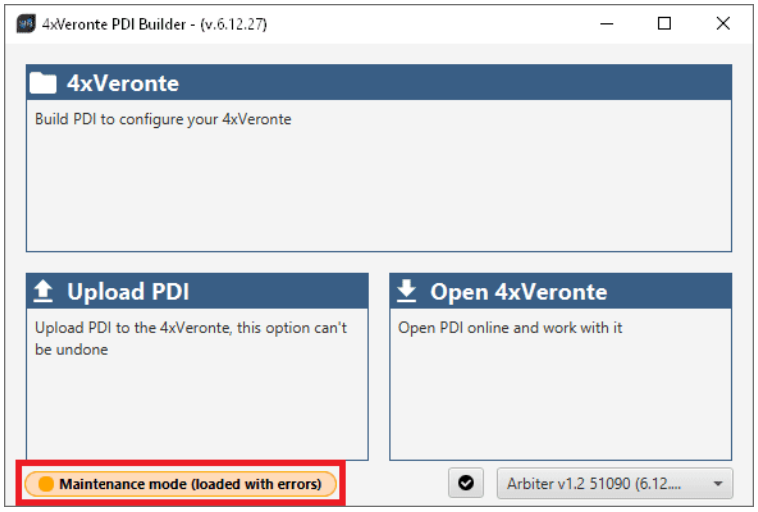


Fig. 3: **Error message**

Therefore, the **arbiter** will be in 'Maintenance mode (loaded with errors)':



To figure out the cause of the problem, the user can simply click on the **PDI Error button**  , which will show what the PDI Error is. This is useful while the configuration is in progress, however, if the user encounters this situation during the operation, it is also possible to look up the cause of the PDI Error directly on the **Veronte Ops Platform panel**. For more information about this panel, see Platform panel section of the **Veronte Ops** user manual.

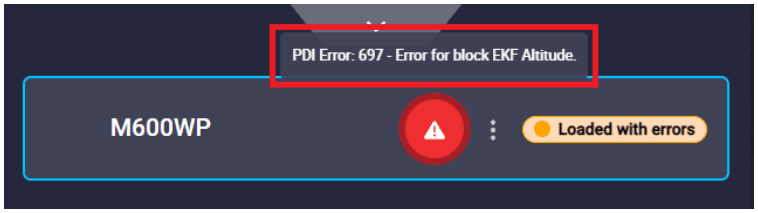


Fig. 4: **Error message**

Then, it will be possible to modify the **Arbiter** configuration to fix the error.

In addition, the list of PDI Errors can be read in the List of PDI errors section of the 4x Software Manual.