

# OneEye\_UART\_Mux\_1 for KIT\_AURIX\_TC375\_LK

Data multiplexer over UART using OneEye

AURIX™ TC3xx Microcontroller Training  
V1.0.0



[Please read the Important Notice and Warnings at the end of this document](#)

## Scope of work

---

### **Demonstrate how to implement the data multiplexer over the UART (USB) interface**

After configuring the OneEye UART interface, a data multiplexer is created. OneEye is used to visualize the signal values.

# Introduction

- > **OneEye** is a GUI that enables the creation of interactive Graphical User Interface. Graphical elements can be drag from a toolbox and drop onto the GUI. The behavior of the created GUI can be customized. Different communication interfaces like UART, Ethernet, CAN, DAS can be used to interact with the embedded system
- > **SyncProtocol / ProtocolBB** is a synchronous protocol that enables data streaming between the target microcontroller and OneEye. It enables to open multiple communication channels, provide packet acknowledge and packet checksum. Data are transported within a message with a message ID and a message payload. See the OneEye help for more information.

Single frame

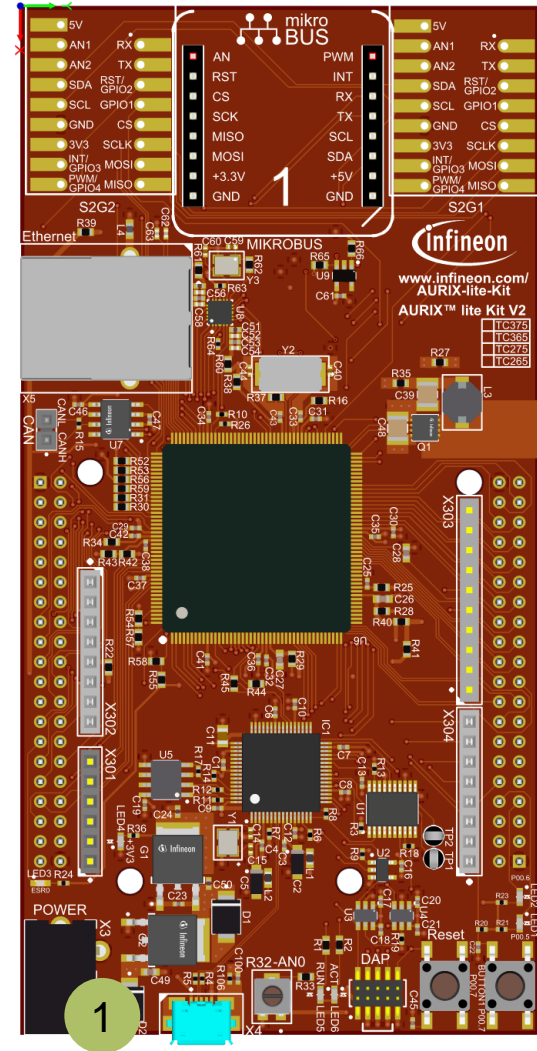
Offset	0	1	2	3
0	Start Byte	Sender	Receiver	Payload length
4	Flags (frameType=data)		Checksum payload	Checksum header
8	Message ID		(Reserved)	
12	Message length			
16	Message payload			
...	... (Message payload)			

- > **Note:** It is recommended to go through some of the **basic tutorials** listed in the help embedded in OneEye (Menu: Help -> OneEye help). This enables a quicker ramp-up in the OneEye concept and ensures a nice journey with OneEye

# Hardware setup

This code example has been developed for the board KIT\_A2G\_TC375\_LITE.

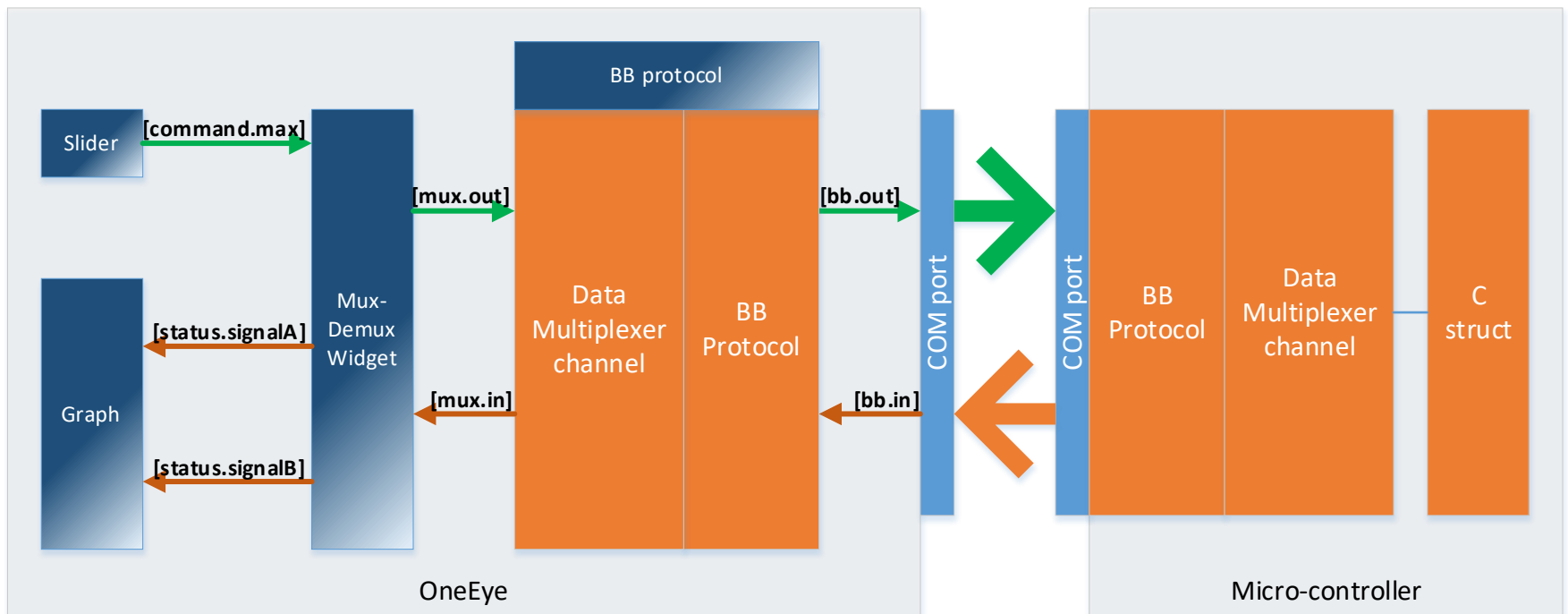
The board should be connected to the PC through the USB port 1



# Configuration overview

In this configuration two C struct are used to exchange data over the COM port between the microcontroller and OneEye.

In OneEye, two signals **bb.in** and **bb.out** are used to connect the COM port data stream to the BB protocol. The BB protocol is configured to open a channel reserved for the data multiplexer. This channel connects to the Mux-Demux widget using the **mux.in** and **mux.out** signals. The Mux-Demux widget connects to a slider with the **command.max** signal and a Graph with the **status.signalA** and **status.signalB** signals.



# Implementation - AURIX

---

## Enabling the OneEye library

The OneEye library must be enabled by adding the following line to *Ifx\_Cfg.h*:  
***#define IFX\_OE\_AL\_USE\_AURIX\_ILLD***

## Configuring the data multiplexer

A OneEye BB protocol client (*Ifx\_Oe\_SyncProtocol\_Client*) is an object that enables raw message data transmission using the BB protocol (*Ifx\_Oe\_SyncProtocol*).

The OneEye BB protocol client is initialized with *initDataMultiplexer() / Ifx\_Oe\_SyncProtocol\_addClient()*. The *ifx\_oe\_syncprotocol.h* file can be found in the Libraries\OneEye directory.

## Configuring the UART communication

The UART communication is initialized with the function *initUart()*, which also initializes the BB protocol.

In the infinite while loop, the function *processUart()* executes the SyncProtocol.

# Implementation - AURIX

---

## Receiving data from OneEye

Receiving data from OneEye is done within ***processDataMultiplexer()***. The client is periodically checked for incoming messages using the ***Ifx\_Oe\_SyncProtocol\_isMessageAvailable()*** passing a pointer to the BB client as parameter.

To decode the received data, a message buffer must be acquired first with the function ***Ifx\_Oe\_SyncProtocol\_getReadMessageBuffer()***. The function takes a pointer to the BB client as input parameter, and pointers to a message ID, a message payload buffer and a message length as output parameters. Then the message payload pointer is cast to the C struct type that defines the data. Finally the data are readout from the C struct and the message is released using ***Ifx\_Oe\_SyncProtocol\_releaseReadMessageBuffer()***.

## Sending data to OneEye

Sending data to OneEye is done within ***processDataMultiplexer()***. Data are send periodically (100ms). The timing is ensured using the ***Ifx\_Oe\_Time\_isDeadline()*** and ***Ifx\_Oe\_Time\_add()*** functions.

To send data, a message buffer must be acquired first with ***Ifx\_Oe\_SyncProtocol\_setSendMessageBuffer()***. The function takes a pointer to the BB client, the message ID and the message size parameters. Then the message payload is cast to the C struct type containing the data and the C struct is filled with data. Finally, the message is send using ***Ifx\_Oe\_SyncProtocol\_sendMessage()*** passing the message pointer as parameter.

**Note:** it is important to ensure the struct member offset and size to enable proper encoding / decoding by OneEye. This memory mapping is specific to the CPU data alignment and compiler. For Hightec compiler, the ***\_\_attribute\_\_((packed))*** is added to the ***DataStreaming\_Data\_0*** and ***DataStreaming\_Data\_1*** struct definition.

# Implementation - AURIX

---

## Configuring the signal generator

A signal generator is used to provide the user with some value to read / write. The signal generator does nothing more than incrementing two signals, **signalA** and **signalB**, stored in the structure **g\_signalGenerator** up to a maximum value before resetting them.

The initialization of the signal generator is done with **initSignalGenerator()**.

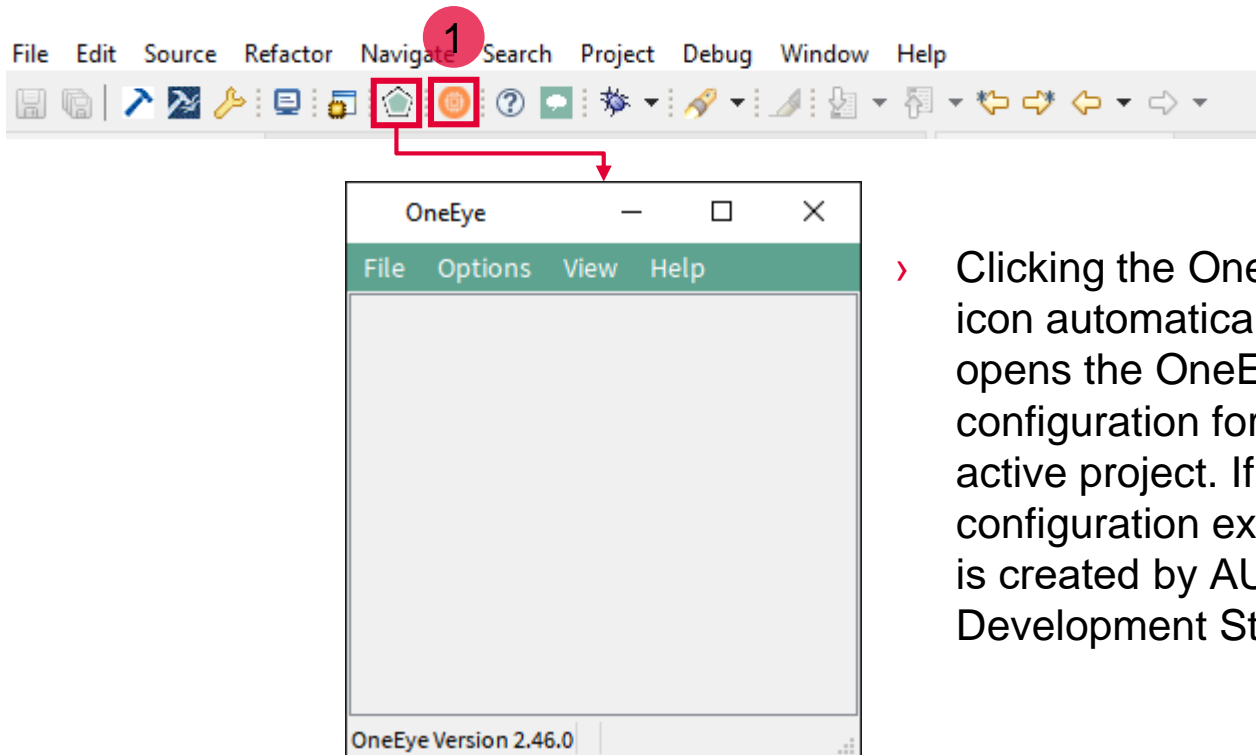
## Running the signal generator

The signal generator is executed in the background loop every 1ms with **processSignalGenerator()**. To ensure the timing, a **deadline** variable is periodically updated with **Ifx\_Oe\_Time\_add()** to obtain the 1ms period.



# Run and Test

- › After code compilation, flash the device using the Flash button **1** to ensure that the program is running on the device
- › For this training, the OneEye application is required for visualizing the values. OneEye can be opened inside the AURIX™ Development Studio using the following icon:



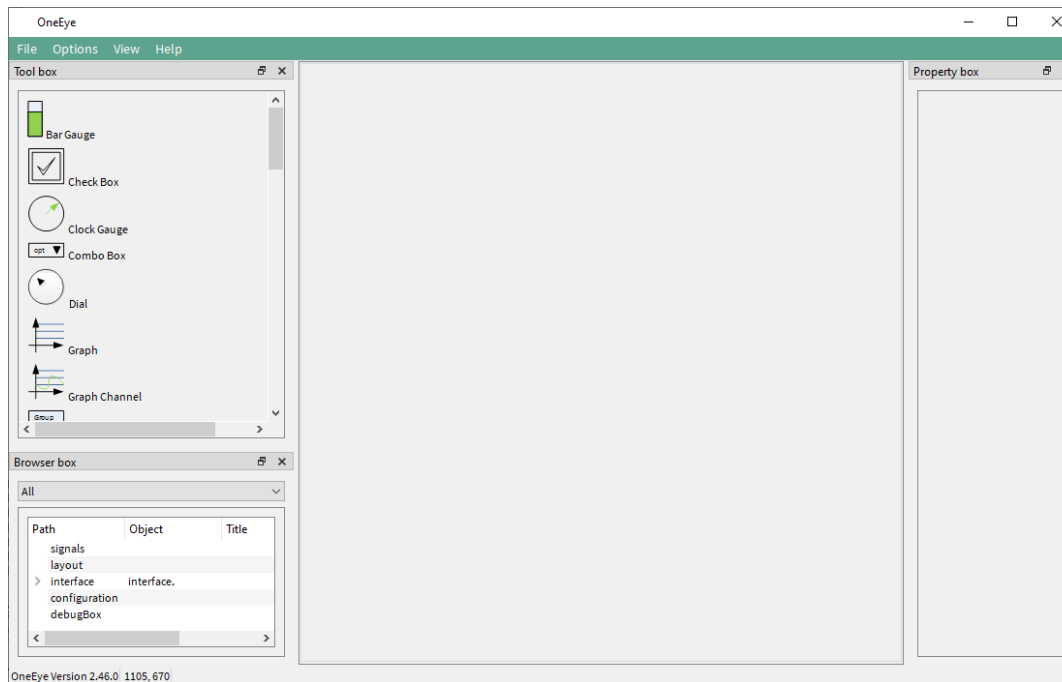
- › Clicking the OneEye icon automatically opens the OneEye configuration for the active project. If no configuration exists, it is created by AURIX™ Development Studio

# Implementation - OneEye

In this training, the OneEye configuration is provided inside the Libraries folder. The following steps are needed to configure the oscilloscope from a brand-new configuration.

## Setup OneEye for editing

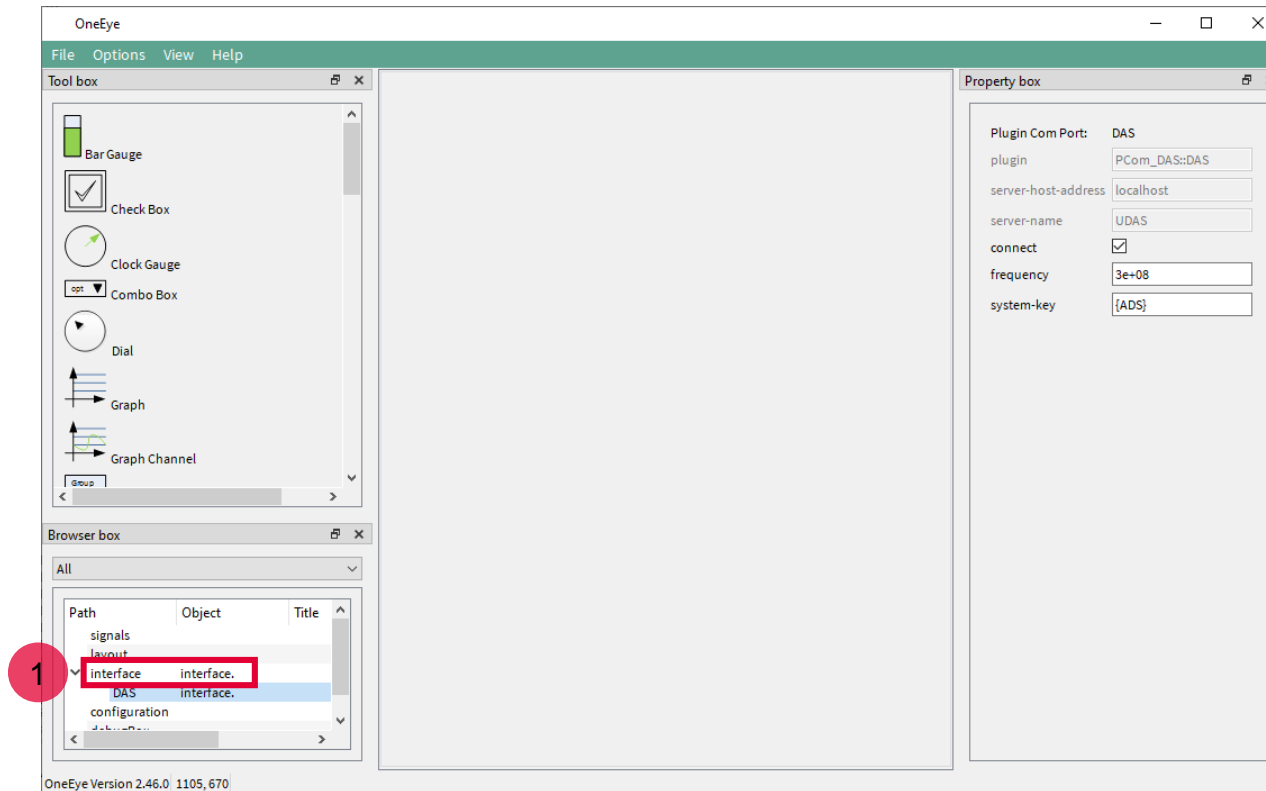
Select the OneEye menu **“Options -> Edit mode”** (if not already checked) to enable the edit mode. Select the OneEye menu **“View -> Browser box”**, **“View -> Property box”**, **“View -> Tool box”** (if not already checked) to display the browser, property box, and tool box. Note that the box can be moved around.



# Implementation - OneEye

## Removing the default DAS interface

When the OneEye configuration is created by ADS, it is already setup with a DAS interface. Select the interface in the Browser box **1** and delete it with “right click and remove” as it is not required in this example.

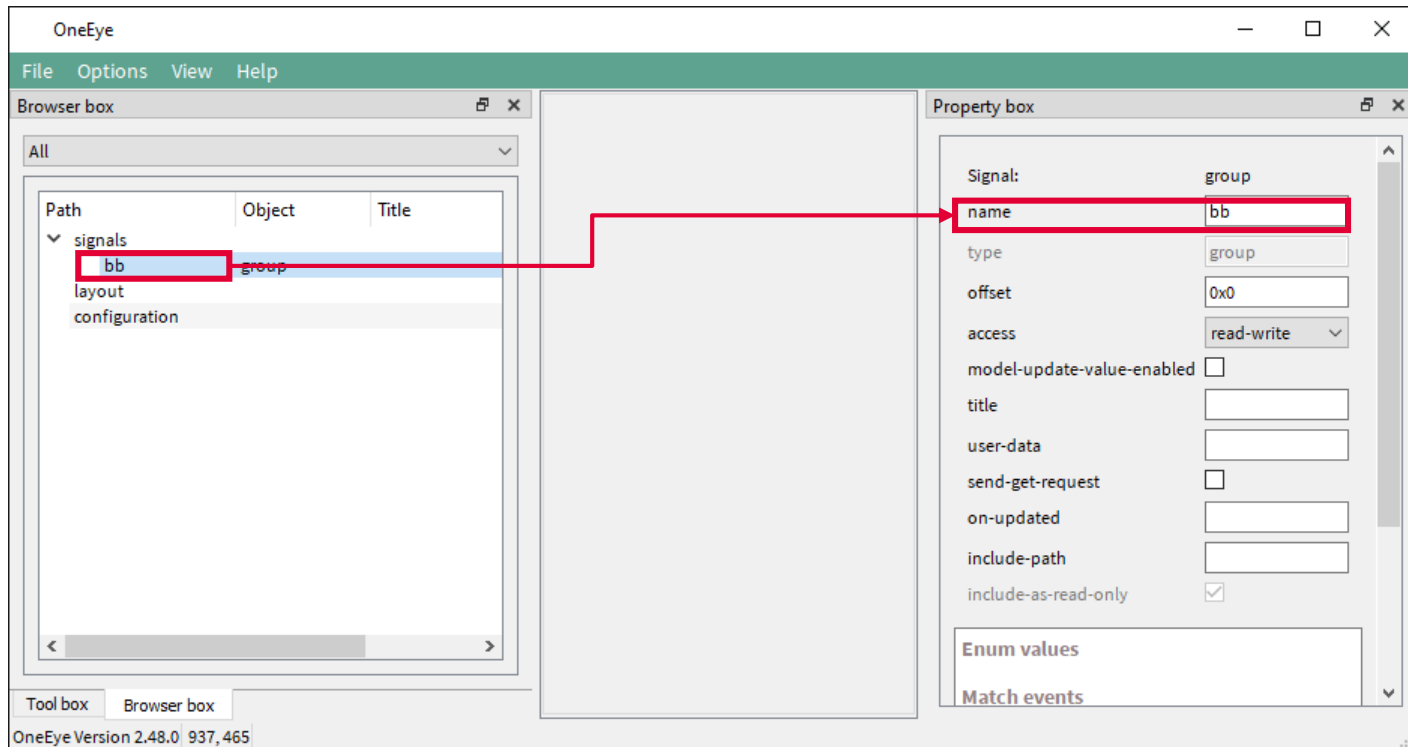


# Implementation - OneEye

## Configuring the UART interface: Signal creation

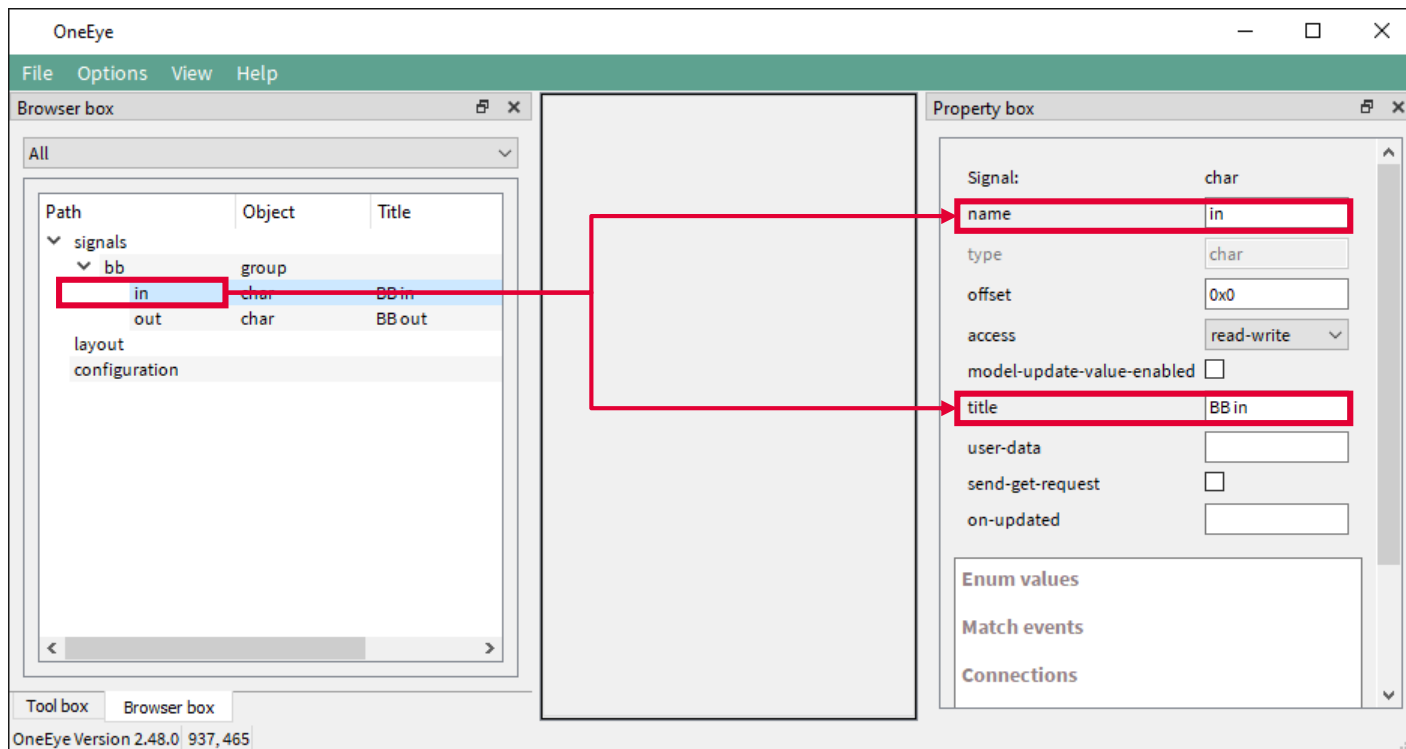
The first step is to create 2 signals to connect the received and transmit data over the UART.

Create a signal group and set its **name** property to **bb**.



# Implementation - OneEye

Add two signals of type **char** into the **bb** group, name them **in** and **out**, and set their **title** property to respectively **BB in** and **BB out**.

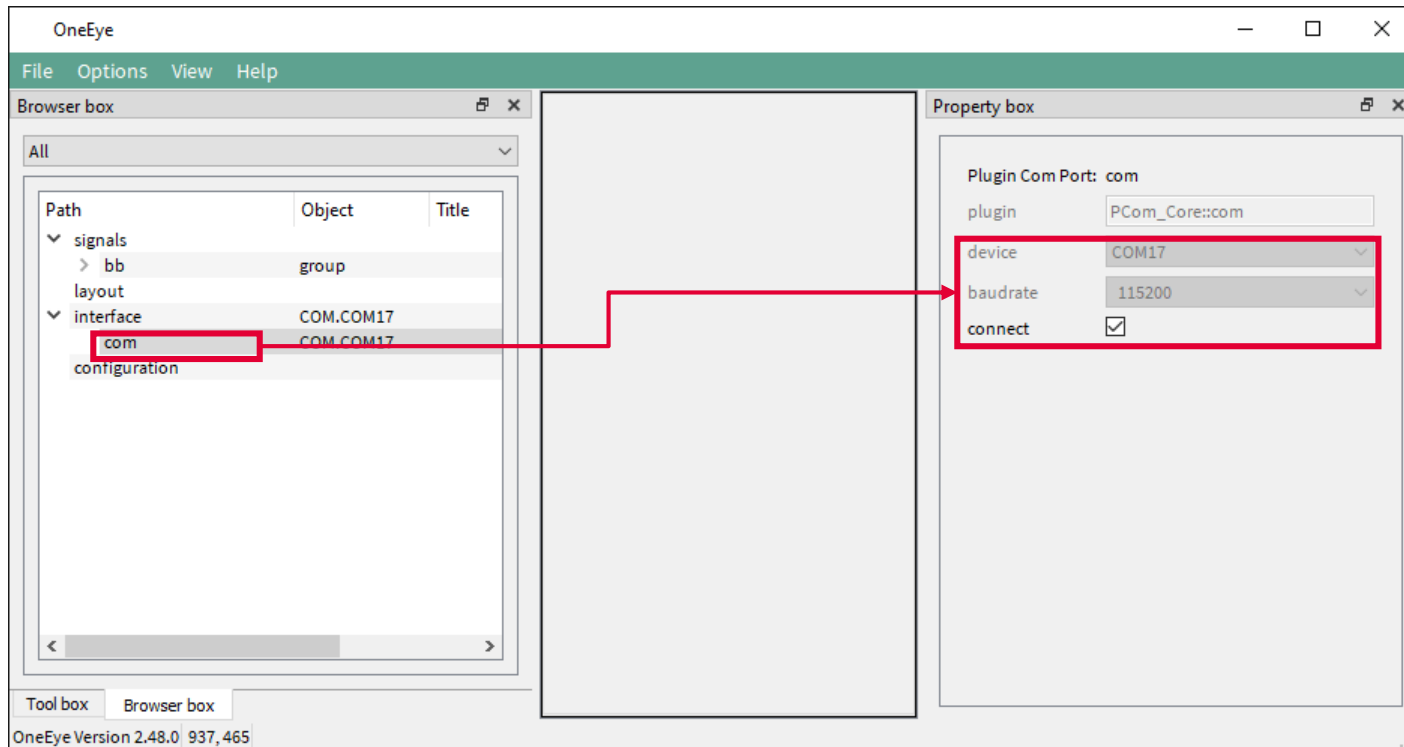


# Implementation - OneEye

## Configuring the UART interface: COM port

Right click in an empty area of the Browser box, and select **Add child -> Interface**. Then right click on the created interface and select **Add child -> com**. Select the **com** item and set its **device** property to the COM port connected to the AURIX board. Set the **baudrate** property to **115200** and click **connect**.

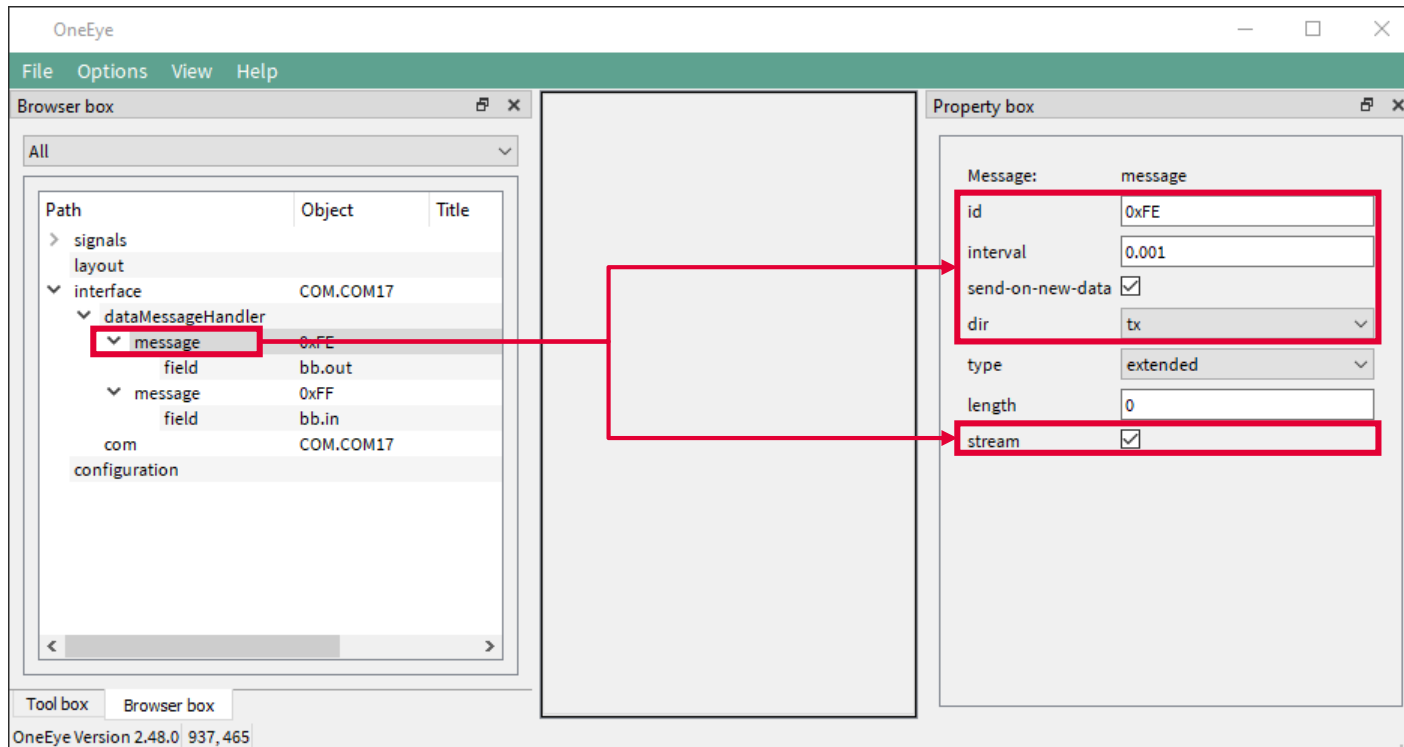
The COM port is now opened and ready for communication.



# Implementation - OneEye

## Configuring the UART interface: Transmit stream

Right click on the **interface** in the Browser box, and select **Add child -> dataMessageHandler**. Then right click on the created **dataMessageHandler** and select **Add child -> message** to create a message item. Configure the **message** with the **id=0xFE**, **interval=0.001**, **send-on-new-data** checked, **dir=tx**, **stream** checked.



The screenshot displays the OneEye software interface with the following configuration details:

Property	Value
Message:	message
id	0xFE
interval	0.001
send-on-new-data	<input checked="" type="checkbox"/>
dir	tx
type	extended
length	0
stream	<input checked="" type="checkbox"/>

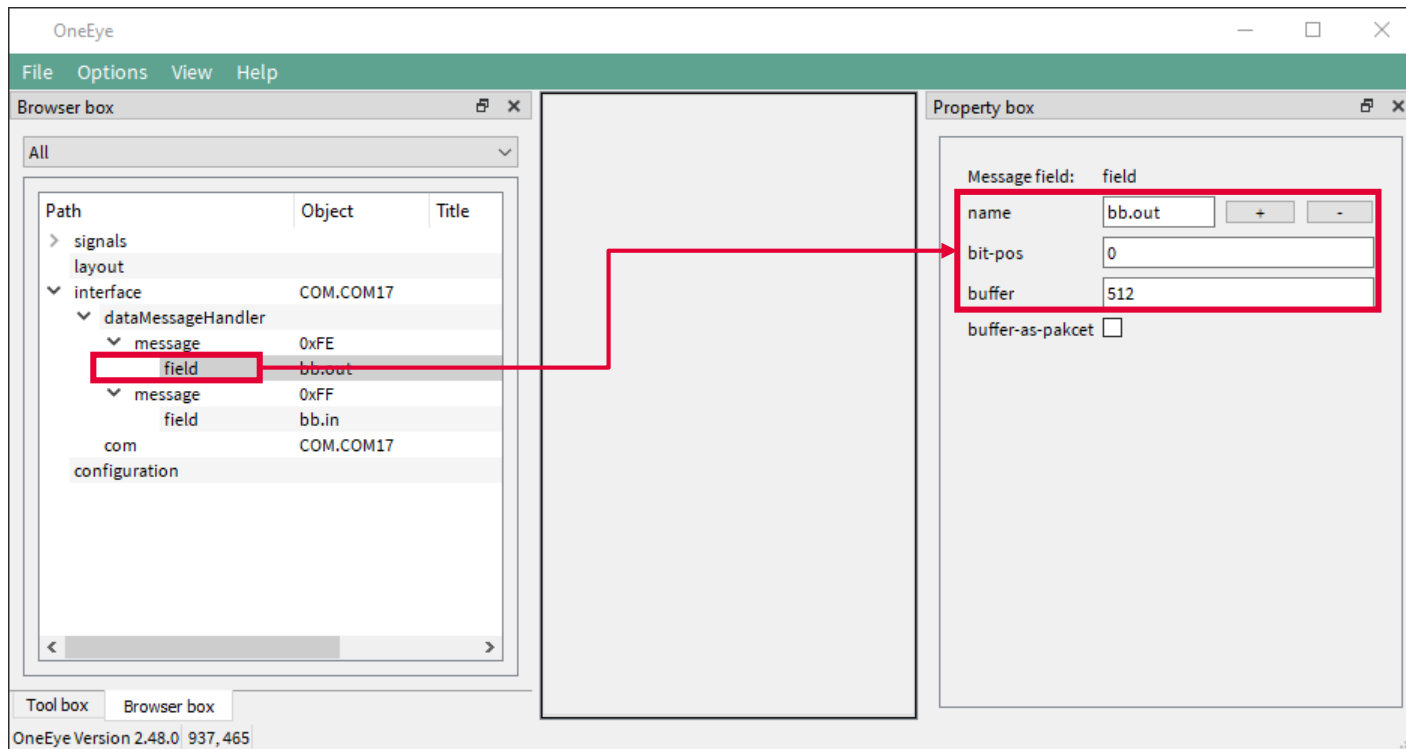
The interface also shows the following tree structure in the Browser box:

- signals
  - layout
  - interface (COM.COM17)
    - dataMessageHandler
      - message (0xFE)
        - field (bb.out)
        - message (0xFF)
          - field (bb.in)
        - com (COM.COM17)
      - configuration

# Implementation - OneEye

Right click on the **message**, and select **Add child -> field**.  
Configure the field with **name=bb.out**, **bit-pos=0**, **buffer=512**.

Now, data will be transmitted over the UART each time the **bb.out** signal is written with some data.

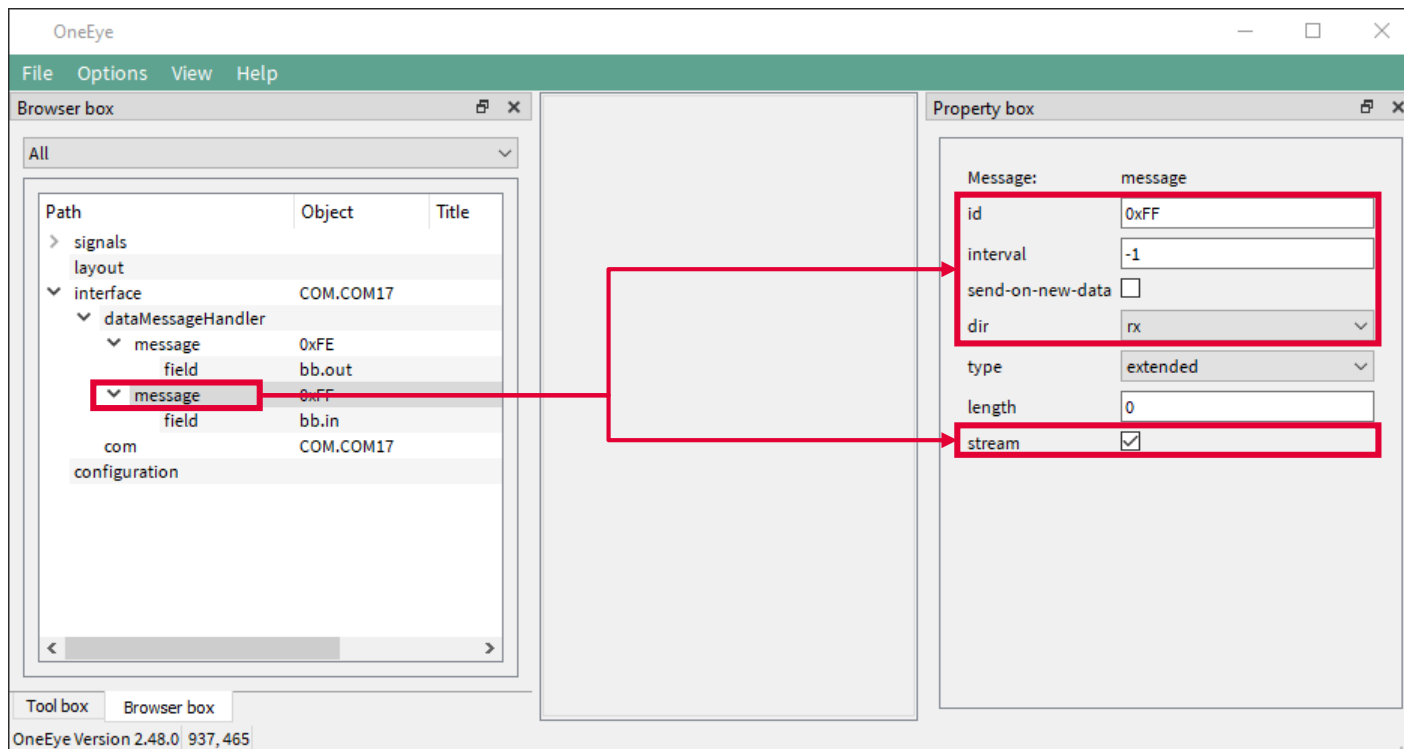




# Implementation - OneEye

## Configuring the UART interface: Receive stream

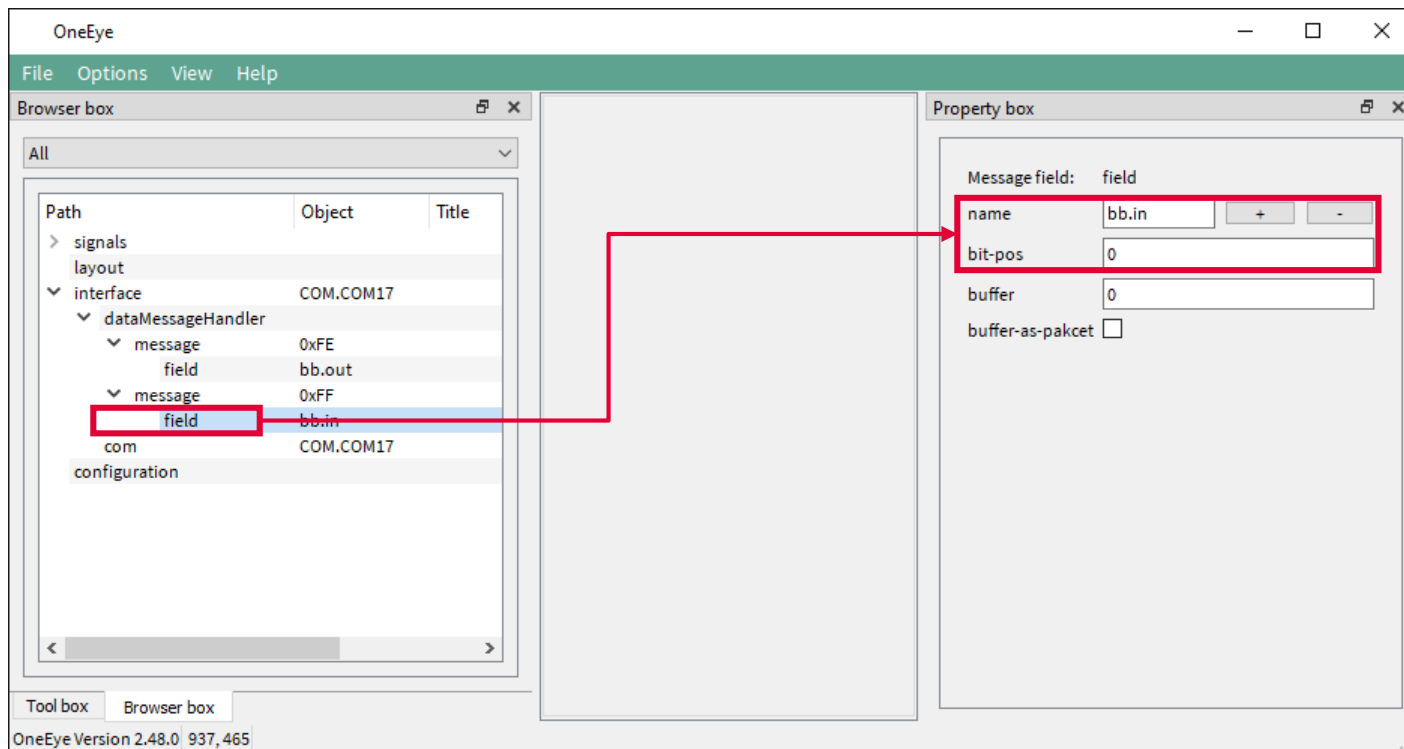
Right click on the **dataMessageHandler** and select **Add child -> message** to create a second message item. Configure the message with the **id=0xFF**, **interval=-1**, **dir=rx**, stream checked.



# Implementation - OneEye

Right click on the **message**, and select **Add child -> field**.  
Configure the field with **name=bb.in**, **bit-pos=0**.

Now each time data are received over the UART, the **bb.in** signal will be updated.

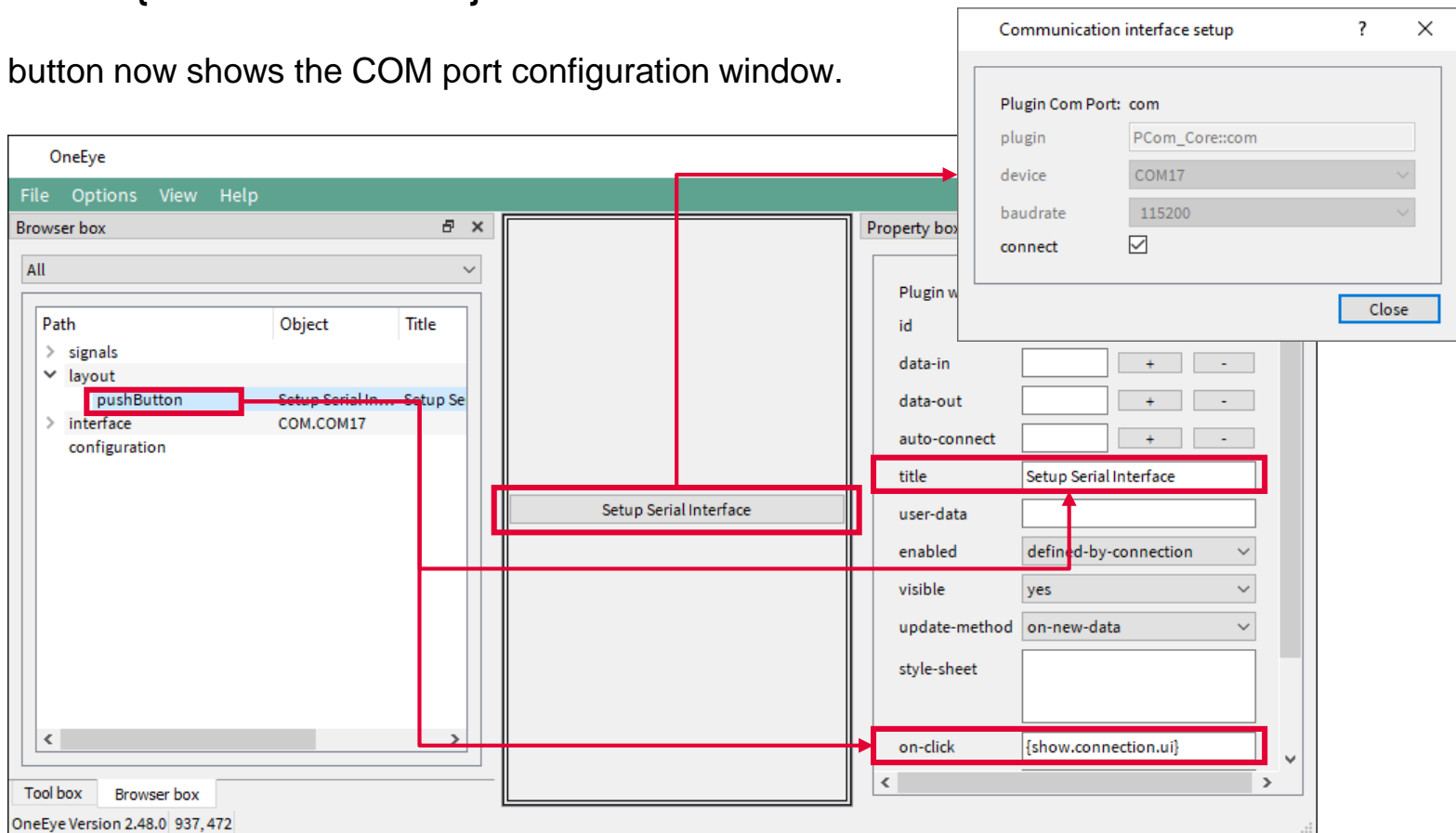


# Implementation - OneEye

## Configuring the UART interface: Push button

Drag and drop a **pushButton** widget from the toolbox onto the layout, configure it with **title=Setup Serial Interface**, **on-click={show.connection.ui}**.

Clicking the button now shows the COM port configuration window.

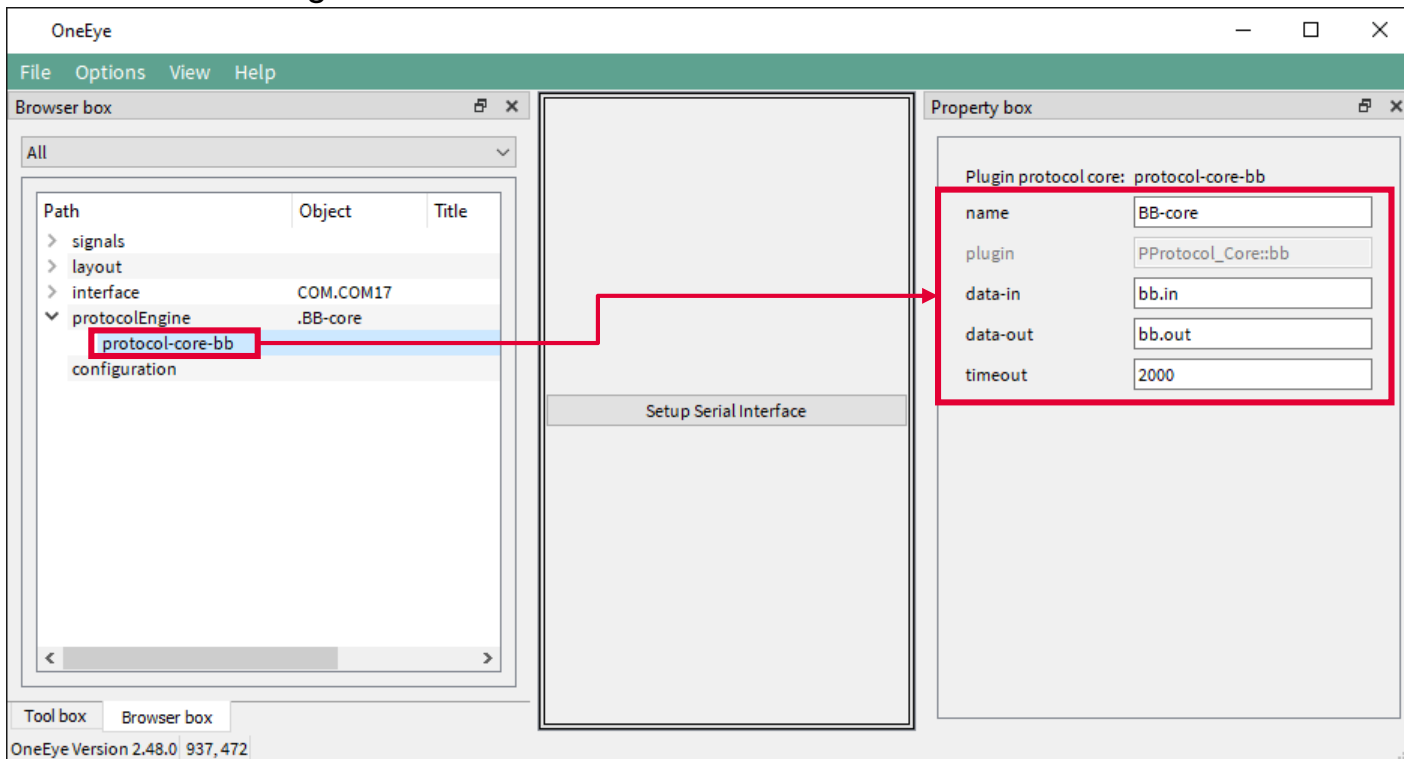


The screenshot illustrates the configuration process in the OneEye software. On the left, the 'Browser box' shows a tree view with 'layout' expanded, containing a 'pushButton' widget. This widget is highlighted with a red box, and a red arrow points from it to the 'Property box' on the right. In the 'Property box', the 'title' property is set to 'Setup Serial Interface' and the 'on-click' property is set to '{show.connection.ui}', both highlighted with red boxes. A red arrow also points from the 'pushButton' in the browser box to the 'on-click' property. In the center, a preview window shows a button labeled 'Setup Serial Interface', also highlighted with a red box. A red arrow points from this button to the 'Communication interface setup' dialog box on the right. This dialog box shows the configuration for the communication interface, including 'Plugin Com Port: com', 'plugin: PCom\_Core::com', 'device: COM17', 'baudrate: 115200', and 'connect' checked. A red arrow points from the 'device' dropdown in the dialog to the 'COM17' entry in the browser box. The 'on-click' property in the 'Property box' is also linked to the 'Communication interface setup' dialog box.

# Implementation - OneEye

## Configuring the BB protocol

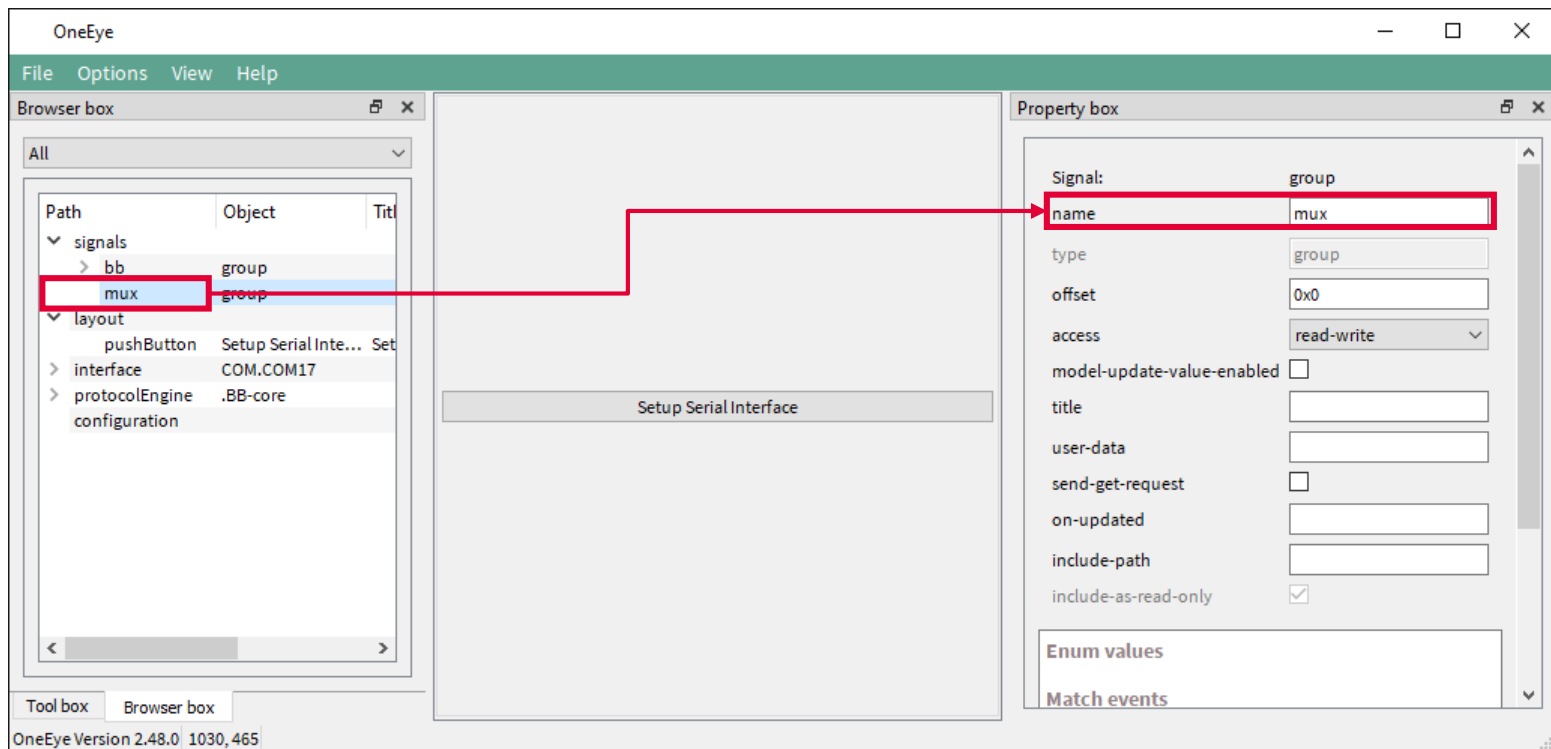
Right click in an empty area of the Browser box, and select **Add child -> protocolEngine**. Then right click on the created **protocolEngine** and select **Add child -> protocol-core-bb**. Connect the BB protocol stream to the **bb.in** and **bb.out** signals by setting respectively the **data-in** and **data-out** properties. Set the **name** property to **BB-core**. And set the **timeout** to **2000** ms so that frames are dropped after 2 seconds in case the microcontroller is not answering.



# Implementation - OneEye

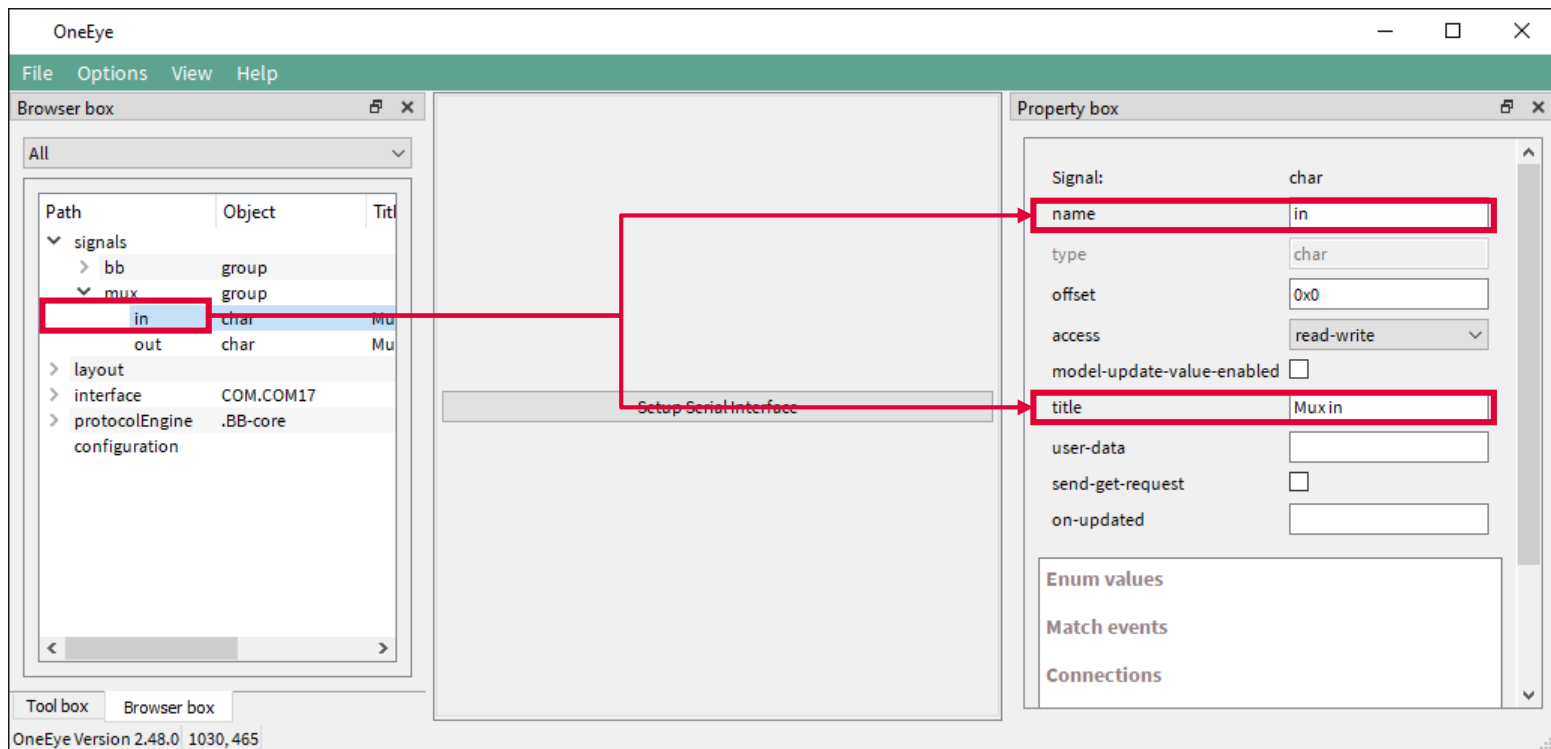
## Configuring the Data multiplexer: signals creation

Create a signal group under the **signals** root and set its **name** property to **mux**.



# Implementation - OneEye

Add two signals of type **char** into the **mux** group, name them **in** and **out**, and set their **title** property to respectively **Mux in** and **Mux out**.

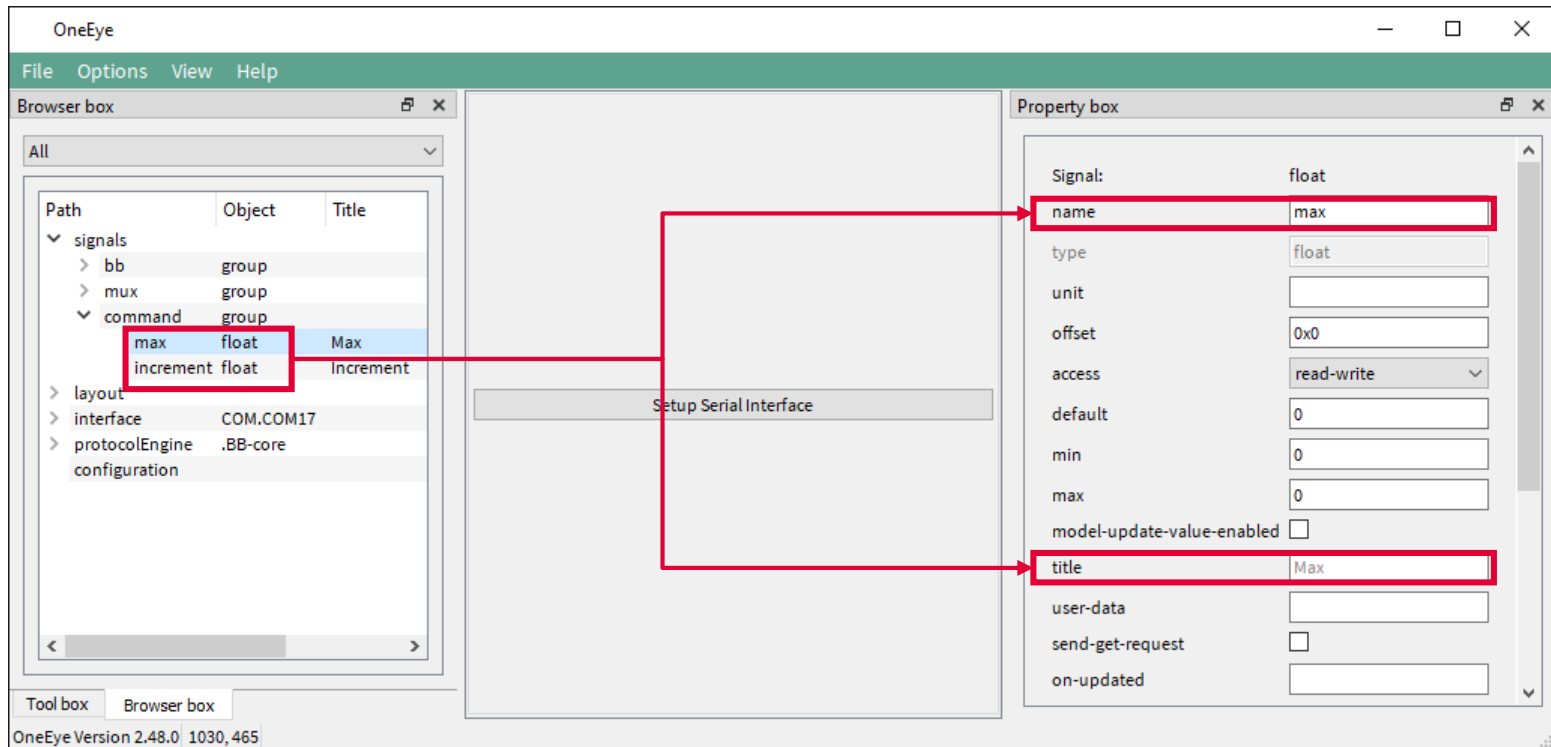


# Implementation - OneEye

## Creating signals to send commands to the AURIX

Create a signal group under the **signals** root and set its **name** property to **command**.

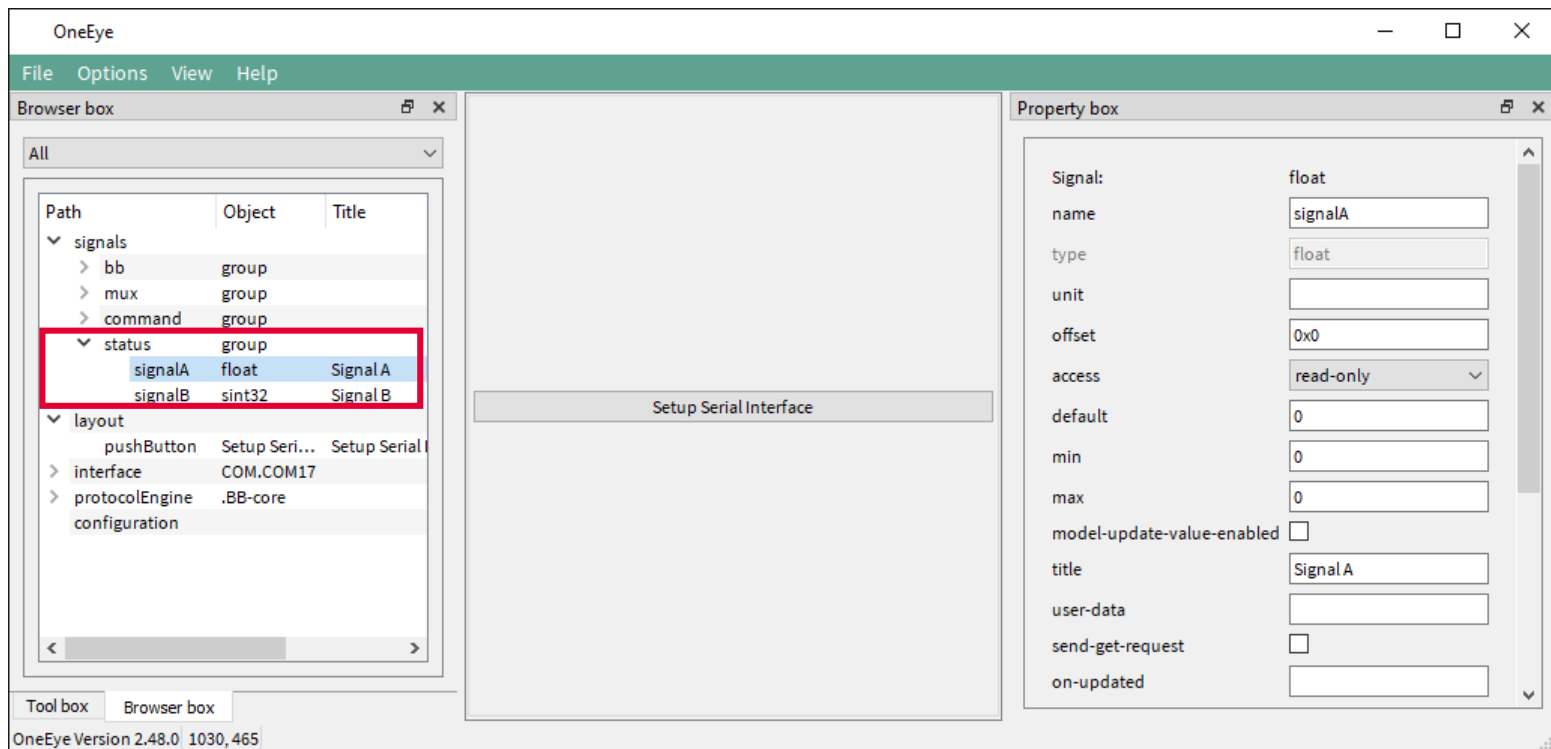
Add two signals of type **float** into the **command** group, name them **max** and **increment**, and set their **title** property to respectively **Max** and **Increment**.



# Implementation - OneEye

## Creating signals for the received data

Create a signal group under the **signals** root and set its **name** property to **status**. Add two signals of type **float** and **sint32** into the **status** group, name them **signalA** and **signalB**, and set their **title** property to respectively **Signal A** and **Signal B**. Note that the data type must match the one defined in the AURIX C struct **DataStreaming\_Data\_0**.

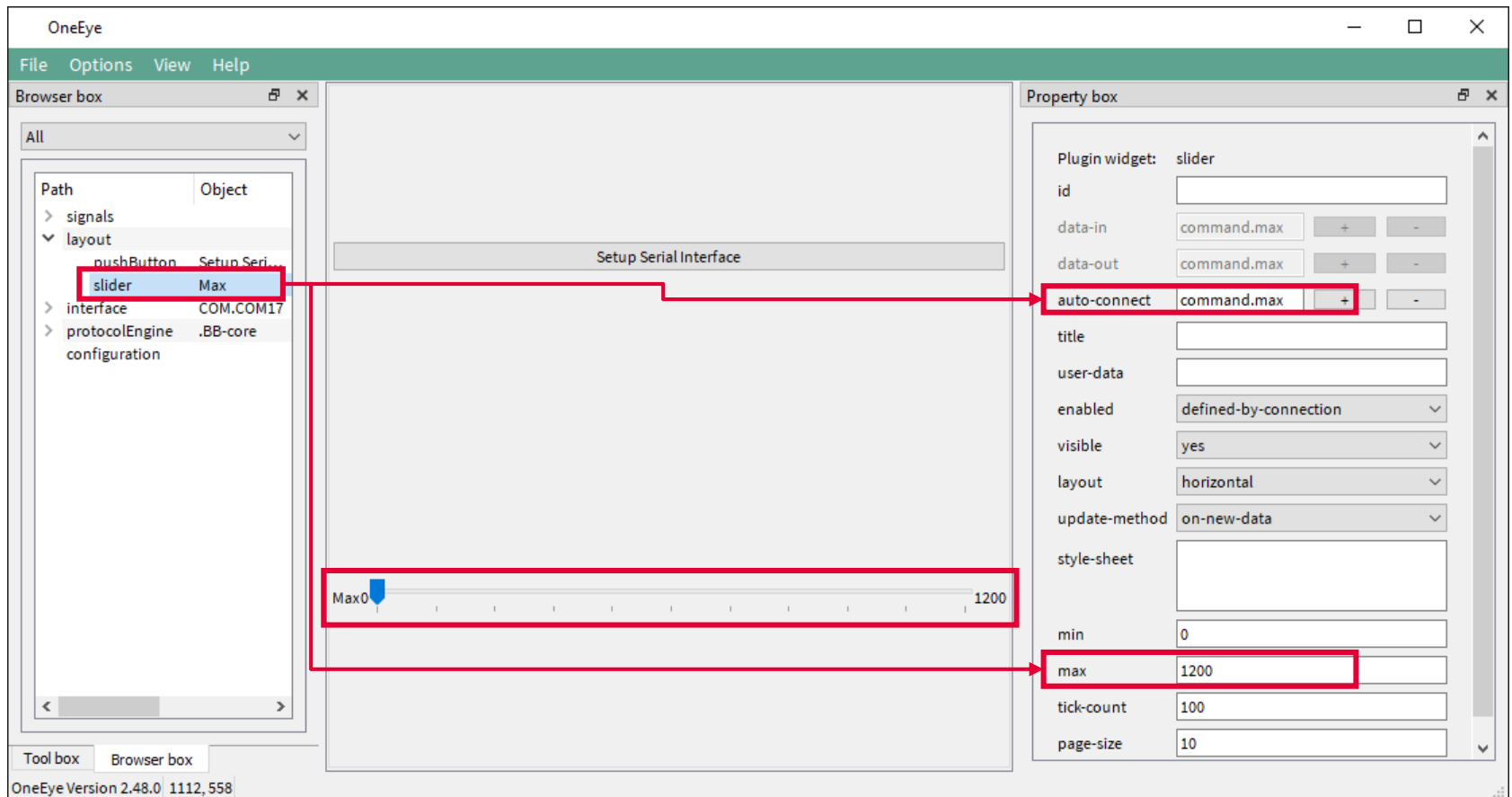




# Implementation - OneEye

## Create the slider widgets to send command to the AURIX

Drag and drop a **slider** widget from the toolbox onto the layout, set the **slider** properties **auto-connect** to **command.max**. and **max** to **1200**,

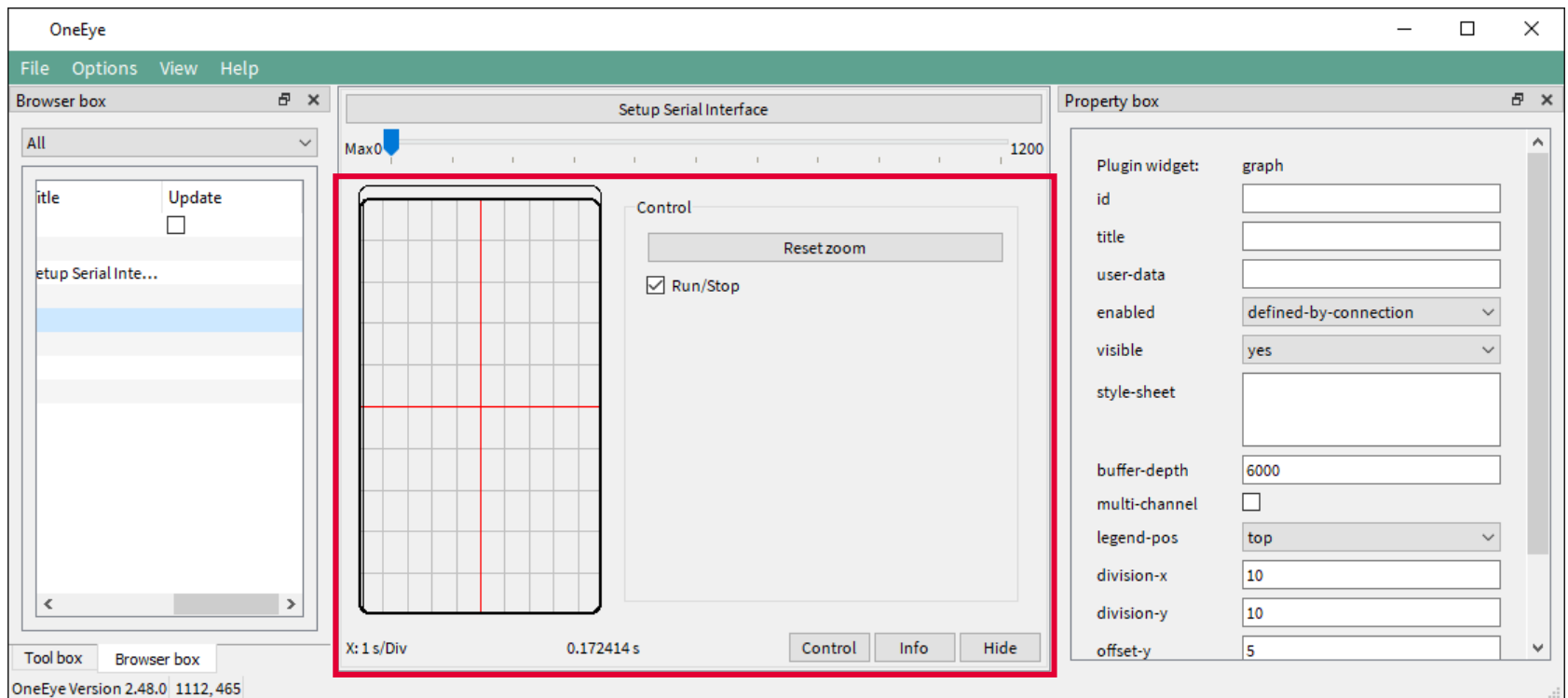


The screenshot shows the OneEye IDE interface. On the left, the 'Browser box' displays a tree view with 'signals', 'layout', 'interface', and 'protocolEngine'. The 'slider' widget is highlighted under 'layout'. In the center, the 'Setup Serial Interface' layout contains a slider widget. On the right, the 'Property box' shows the configuration for the 'slider' widget. The 'auto-connect' property is set to 'command.max', and the 'max' property is set to '1200'. Red boxes and arrows highlight these specific configurations and their connection to the slider widget in the layout.

# Implementation - OneEye

Create the graph widgets to display the signals value

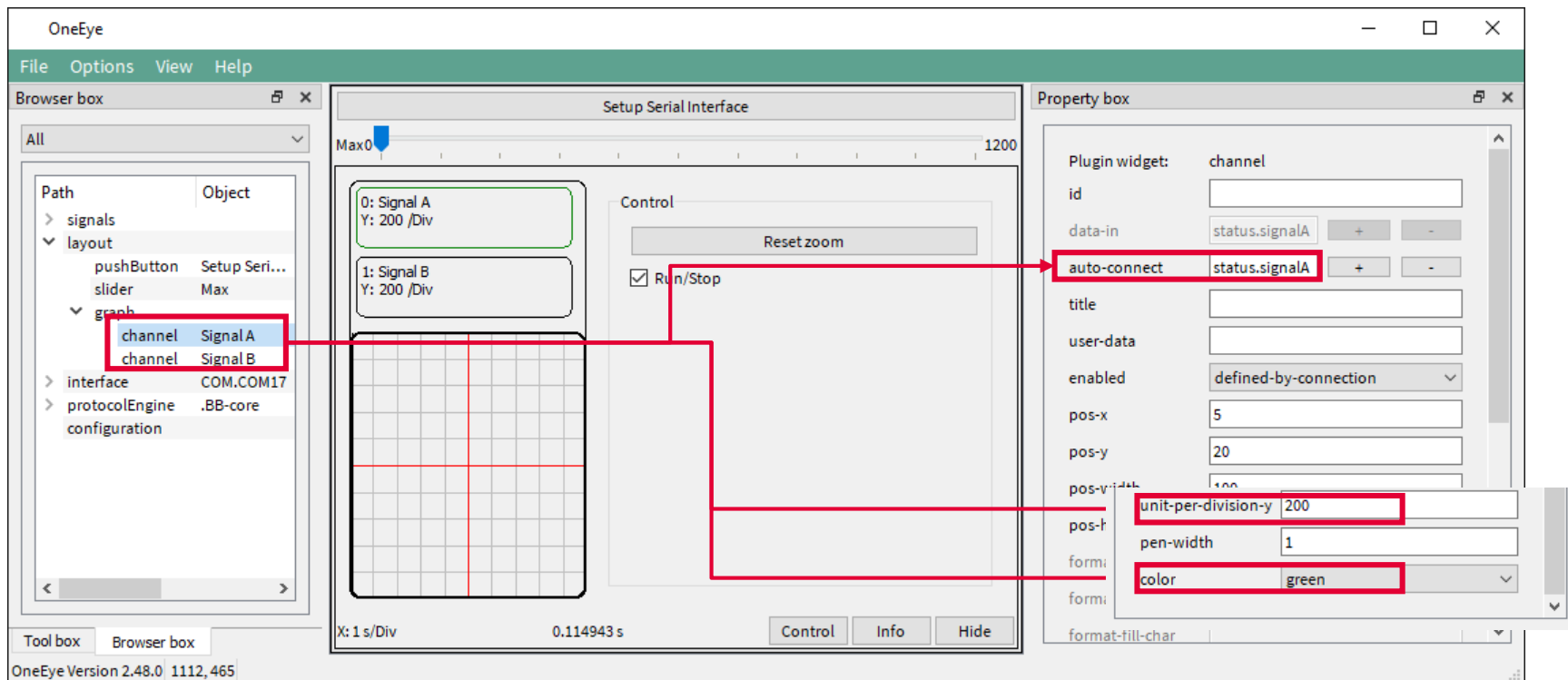
Drag and drop a **graph** widget from the toolbox onto the layout.



# Implementation - OneEye

## Create the graph widgets to display the signals value

Drag and drop a **Graph Channel (channel)** widget from the toolbox onto the layout, set the **channel** properties **auto-connect** to **status.signalA**, **unit-per-division-y** to **200**, and **color** to **green**. Repeat the operation for a second channel and set the **channel** properties **auto-connect** to **status.signalB**, and **color** to **black**.



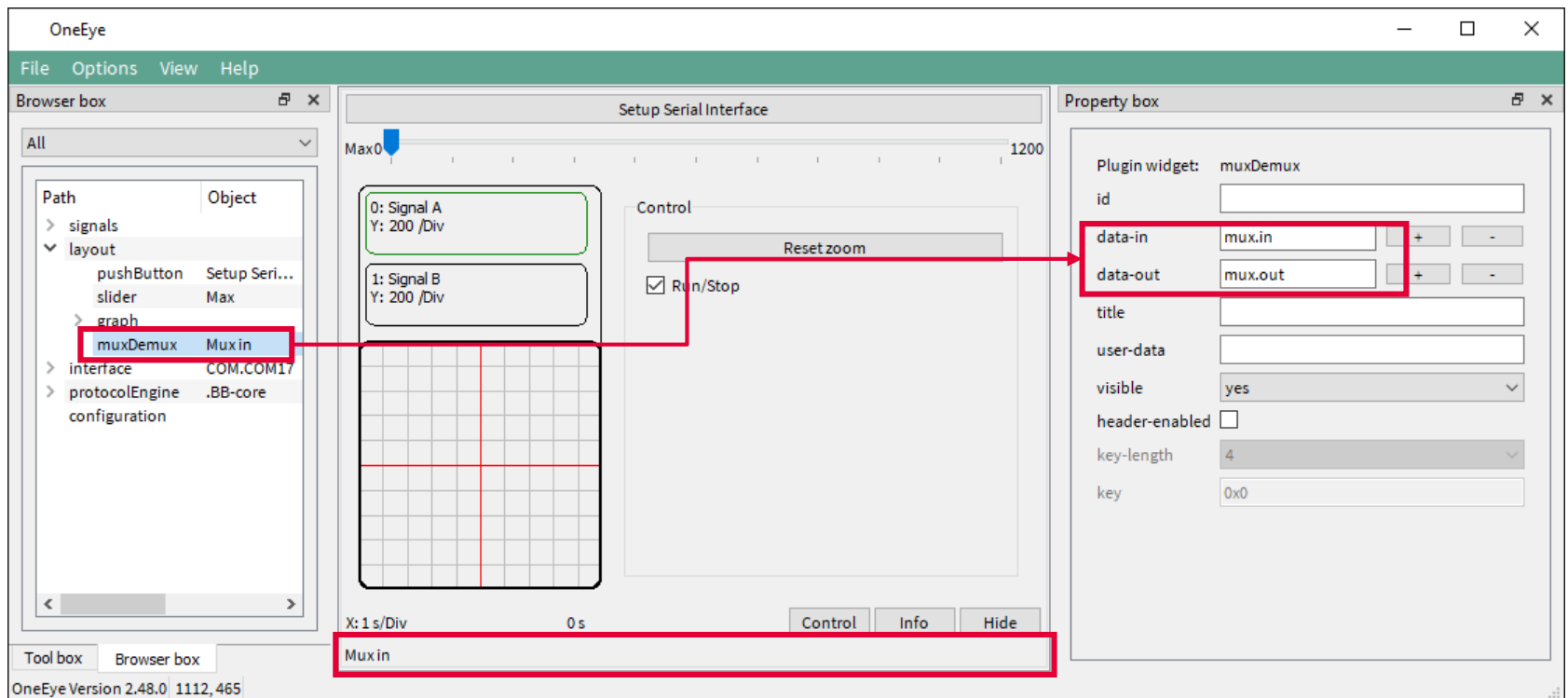
The screenshot shows the OneEye software interface with the following components:

- Browser box:** A tree view on the left showing the project structure. Under the 'graph' folder, two 'channel' widgets are listed, one for 'Signal A' and one for 'Signal B'.
- Setup Serial Interface:** The main workspace containing a graph with two channels. Channel 0 is labeled 'Signal A' with 'Y: 200 /Div'. Channel 1 is labeled 'Signal B' with 'Y: 200 /Div'. The graph has a grid and a 'Reset zoom' button.
- Property box:** A panel on the right showing the configuration for the selected 'channel' widget. The 'auto-connect' property is set to 'status.signalA', 'unit-per-division-y' is set to '200', and 'color' is set to 'green'. Red boxes and arrows highlight these specific settings.

# Implementation - OneEye

Create the muxDemux widgets to connect the BB protocol to the graph and slider widgets

Drag and drop a **Mux-Demux (muxDemux)** widget from the toolbox onto the layout, set the **muxDemux** properties **data-in** and **data-out** to respectively **mux.in** and **mux.out**.

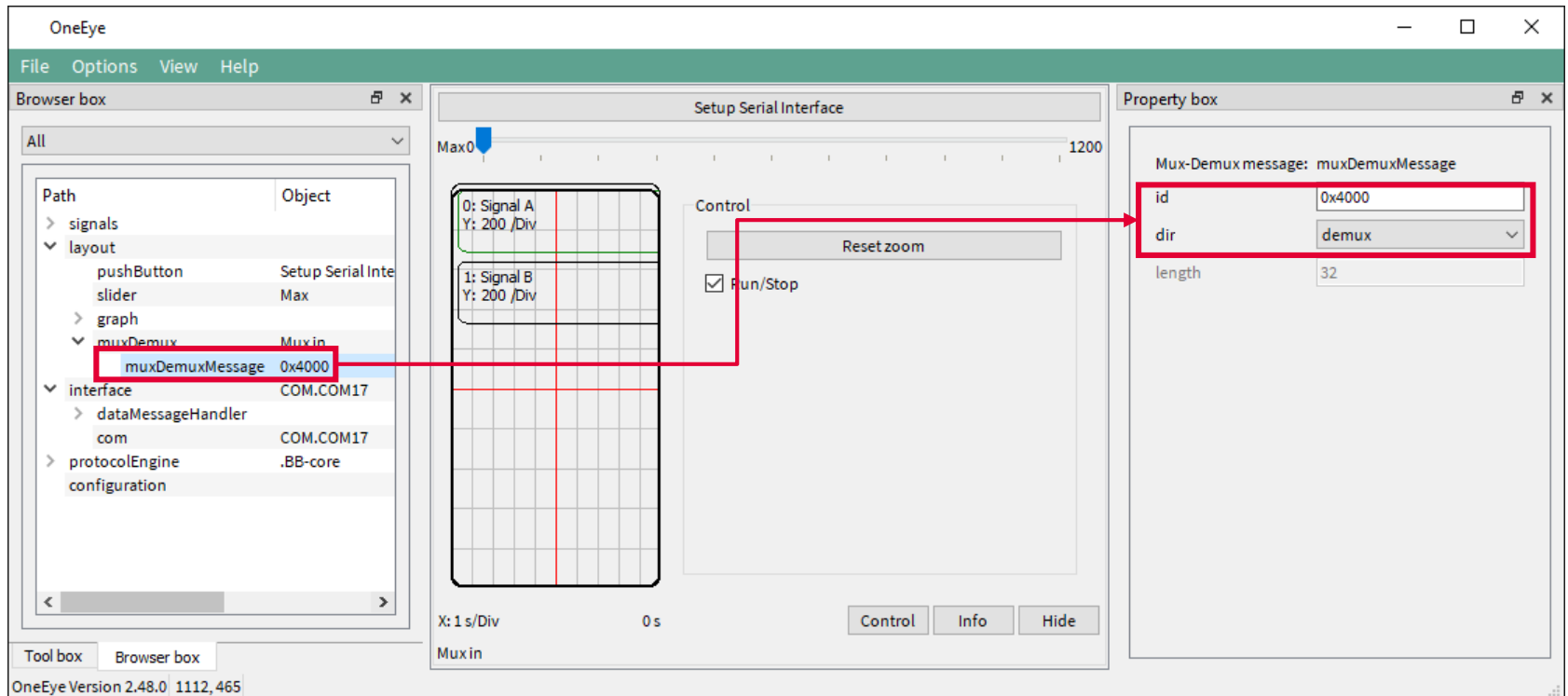


The screenshot displays the OneEye software interface with the following components and configurations:

- Browser box:** A tree view on the left showing the project structure. The **graph** folder is expanded, and the **muxDemux** widget is highlighted with a red box.
- Layout:** The main workspace contains a graph with two signal inputs:
  - 0: Signal A (Y: 200 /Div)
  - 1: Signal B (Y: 200 /Div)
 A red box highlights the **muxDemux** widget on the graph, with a red arrow pointing to its configuration in the Property box.
- Property box:** A panel on the right showing the configuration for the **Plugin widget: muxDemux**. The **data-in** and **data-out** properties are highlighted with a red box and set to **mux.in** and **mux.out**, respectively. Other visible properties include **id**, **title**, **user-data**, **visible** (set to **yes**), **header-enabled** (unchecked), **key-length** (set to **4**), and **key** (set to **0x0**).
- Control Panel:** Located below the graph, it includes a **Reset zoom** button, a **Run/Stop** checkbox (checked), and **Control**, **Info**, and **Hide** buttons.
- Status Bar:** At the bottom, it shows **X: 1s/Div**, **0s**, and a red box highlighting the **Mux in** label.

# Implementation - OneEye

Right click on the **muxDemux** widget in the browser box, and select **Add child -> muxDemuxMessage**, set the **muxDemuxMessage** properties **id** to 0x4000 and **dir** to **demux** to decode received messages.



The screenshot displays the OneEye software interface with three main panels: Browser box, Setup Serial Interface, and Property box.

- Browser box:** Shows a tree view of objects. Under the **layout** folder, the **muxDemux** folder is expanded, and the **muxDemuxMessage** object is selected and highlighted with a red box. Its object ID is listed as **0x4000**.
- Setup Serial Interface:** The central workspace shows a graph with two signals: **0: Signal A** and **1: Signal B**. A red box highlights the **Control** panel, which includes a **Reset zoom** button and a checked **Run/Stop** checkbox. A red arrow points from the **muxDemuxMessage** object in the browser box to this control panel.
- Property box:** Shows the configuration for the selected **Mux-Demux message: muxDemuxMessage**. The **id** property is set to **0x4000** and the **dir** property is set to **demux**. Both the **id** and **dir** fields are highlighted with a red box. The **length** property is set to **32**.

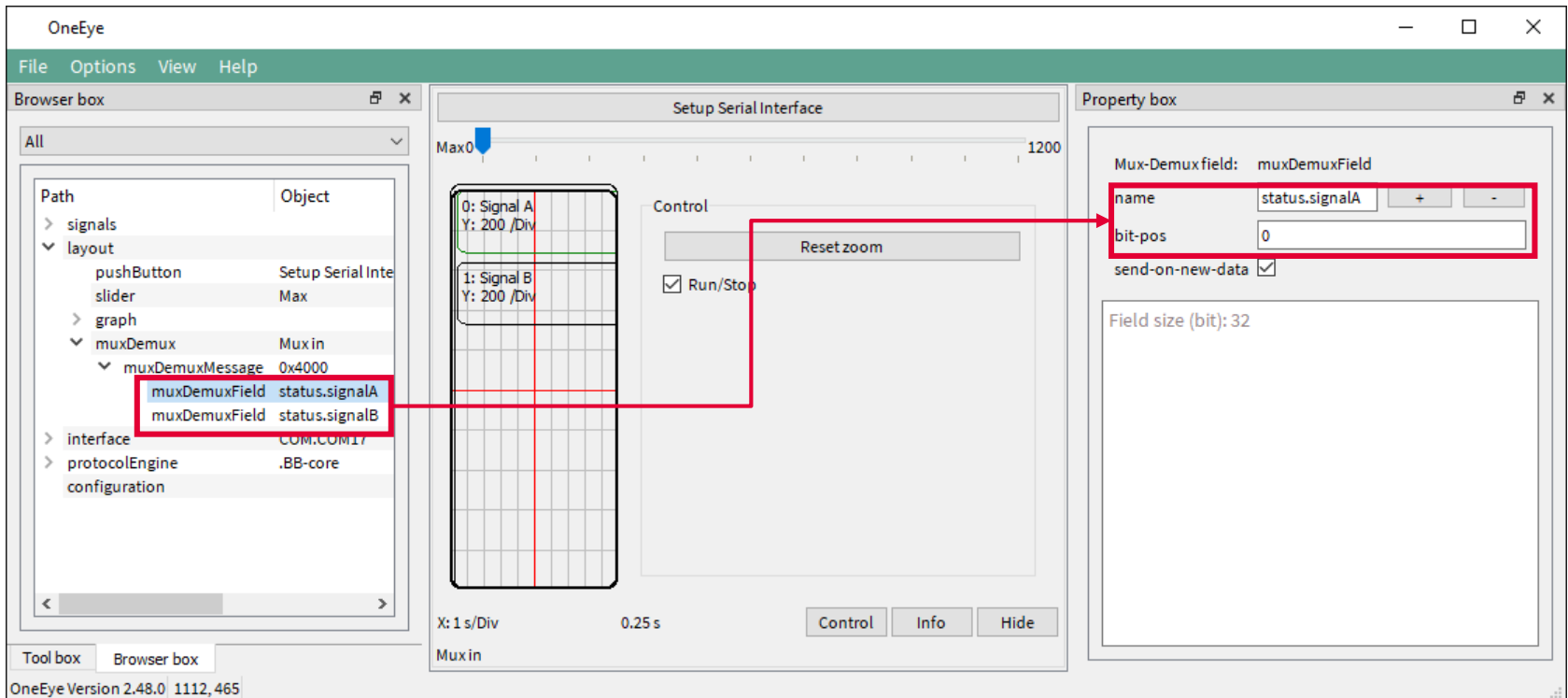
At the bottom left, the version information is displayed: **OneEye Version 2.48.0 1112,465**.

# Implementation - OneEye

Right click on the **muxDemuxMessage** in the browser box, and select **Add child -> muxDemuxField**, set the **muxDemuxField** properties **name** to **status.signalA**, **bit-pos** to **0**.

Repeat the operation for a second signal and set its properties **name** to **status.signalB**, **bit-pos** to **32**.

**Note:** the **status.signalA** and **status.signalB** signal size (32 bits), type (float / sint32) and offset (0 / 32) must match the data member **signalA** and **signalB** of the C struct **DataStreaming\_Data\_0**.



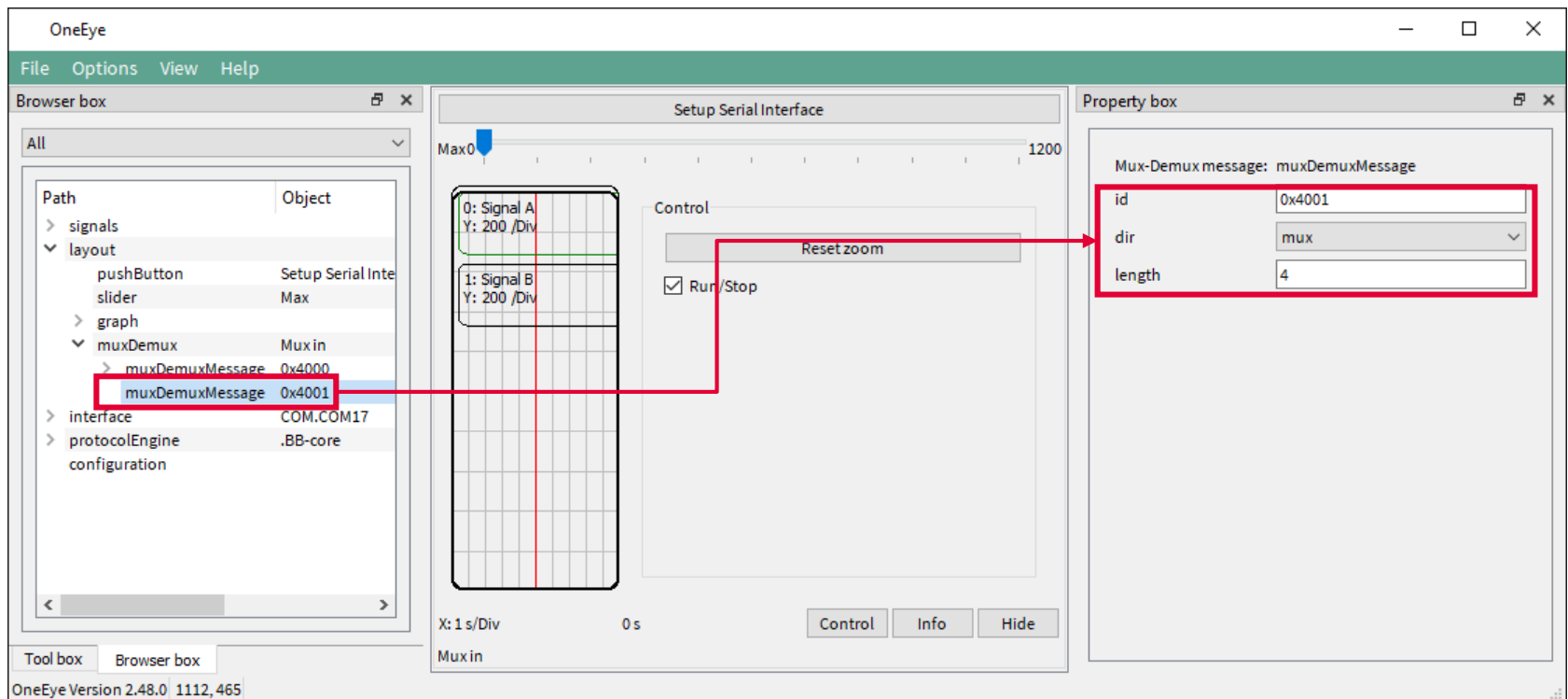
The screenshot displays the OneEye software interface with three main panels:

- Browser box:** Shows a tree view of objects. Under the **layout** folder, **muxDemuxMessage** (0x4000) is expanded to show two child objects: **muxDemuxField status.signalA** and **muxDemuxField status.signalB**. Both are highlighted with a red box.
- Setup Serial Interface:** A central workspace showing a signal graph with two traces: "0: Signal A" and "1: Signal B". A red box highlights the **muxDemuxField status.signalA** object in the browser box, with a red arrow pointing to the **name** field in the property box.
- Property box:** Shows the configuration for the selected **muxDemuxField** object. The **name** is set to **status.signalA** and the **bit-pos** is set to **0**. Both fields are highlighted with a red box. The **Field size (bit)** is set to **32**.

At the bottom of the window, the version information is displayed: **OneEye Version 2.48.0 1112,465**.

# Implementation - OneEye

Right click on the **muxDemux** widget in the browser box, and select **Add child -> muxDemuxMessage**, set the **muxDemuxMessage** properties **id** to 0x4001 and **dir** to **mux** to encode and send messages. Set the **length** property to **4** bytes, which corresponds of the size of the C struct **DataStreaming\_Data\_1**.



The screenshot displays the OneEye software interface with three main panels:

- Browser box:** A tree view showing the project structure. Under the 'muxDemux' widget, a 'muxDemuxMessage' widget with ID '0x4001' is highlighted with a red box. A red arrow points from this widget to the Property box.
- Setup Serial Interface:** The main workspace showing a signal graph with two channels: '0: Signal A' and '1: Signal B'. A 'Control' panel is visible with a 'Reset zoom' button and a 'Run/Stop' checkbox. A red arrow points from the 'Reset zoom' button to the Property box.
- Property box:** A panel showing the configuration for the selected 'muxDemuxMessage' widget. The properties are:
 

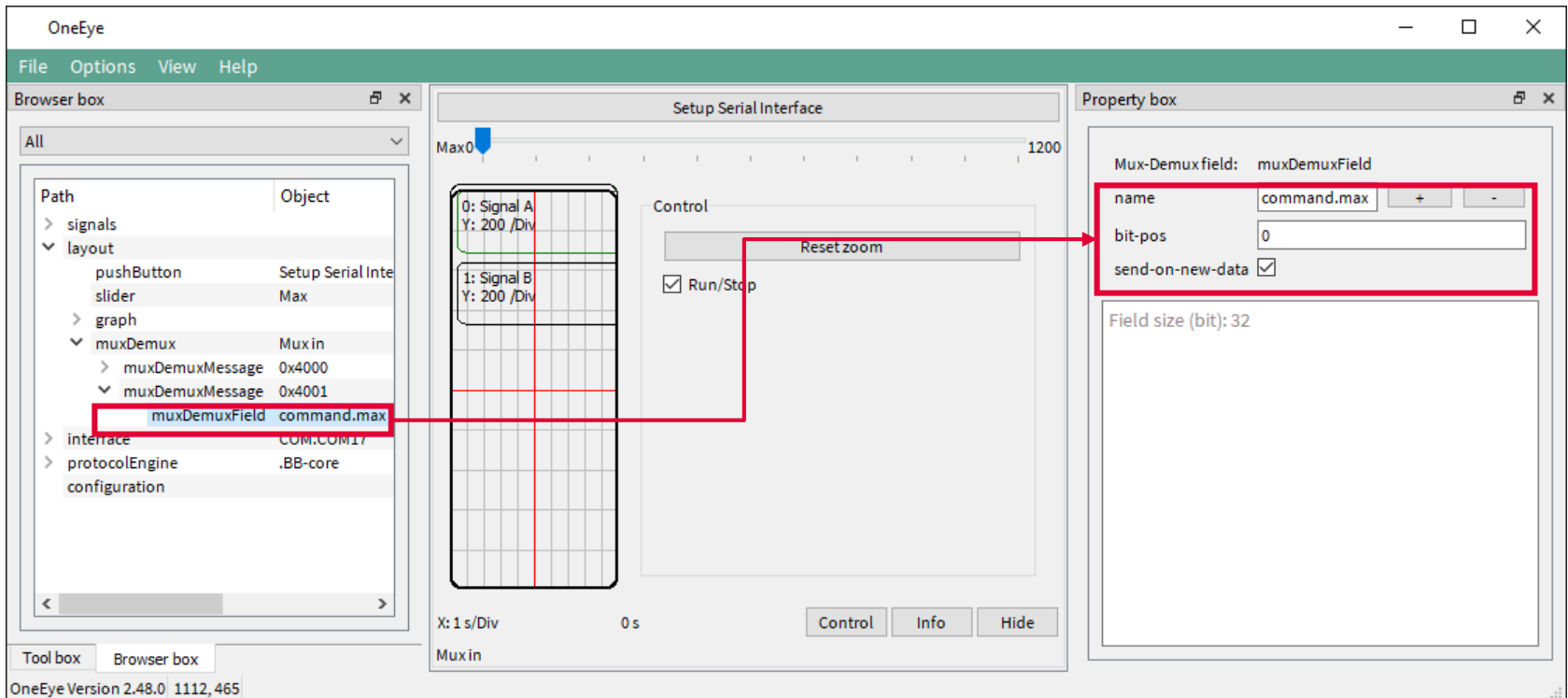
Mux-Demux message: muxDemuxMessage	
id	0x4001
dir	mux
length	4

The status bar at the bottom indicates 'OneEye Version 2.48.0 1112,465'.

# Implementation - OneEye

Right click on the **muxDemuxMessage** in the browser box, and select **Add child -> muxDemuxField**, set the **muxDemuxField** properties **name** to **command.max**, **bit-pos** to **0**, and **check send-on-new-data**.

**Note:** the **command.max** signal size (32 bits), type (float) and offset (0) must match the data member **max** or the C struct **DataStreaming\_Data\_1**.



The screenshot displays the OneEye software interface with three main panels:

- Browser box:** Shows a tree view of objects. The **muxDemuxField** object under the **muxDemuxMessage 0x4001** path is highlighted with a red box.
- Setup Serial Interface:** The main workspace showing a signal graph with two signals (Signal A and Signal B) and a control panel with a **Reset zoom** button and a **Run/Stop** checkbox.
- Property box:** Shows the configuration for the selected **muxDemuxField** object. The properties are:
  - name:** `command.max`
  - bit-pos:** `0`
  - send-on-new-data:**
 The **Field size (bit): 32** is also displayed below the properties.

Red arrows indicate the selection of the **muxDemuxField** object in the browser box and the **Reset zoom** button in the main workspace.

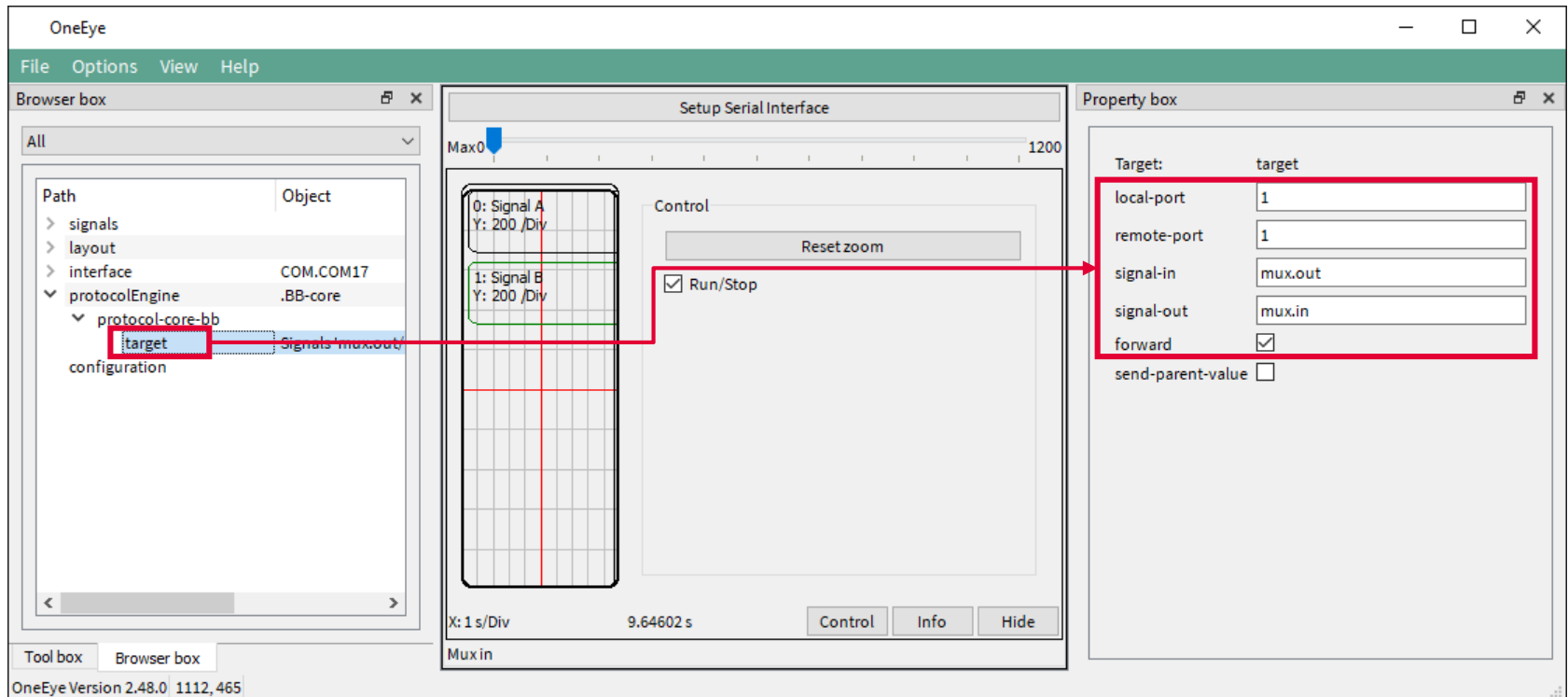
OneEye Version 2.48.0 1112,465



# Implementation - OneEye

## Connect the Mux-Demux widget to the BB protocol

Right click on the **protocol-core-bb** and select **Add child -> target**. Select the **target** item and set **local-port** and **remote-port** to **1** to match the AURIX settings, set **signal-in=mux.out**, **signal-out=mux.in**, and **forward** checked.



The screenshot shows the OneEye software interface with the following components:

- Browser box:** A tree view showing the project structure. The path is: `signals` > `layout` > `interface` (COM.COM17) > `protocolEngine` (.BB-core) > `protocol-core-bb` > `target` (Signals 'mux.out' configuration). The `target` item is highlighted with a red box.
- Plot area:** A graph titled "Setup Serial Interface" showing two signals: "0: Signal A" and "1: Signal B". The x-axis is labeled "X: 1 s/Div" and the y-axis is "Y: 200 /Div". A red arrow points from the `target` item in the browser box to the "Run/Stop" checkbox in the plot area.
- Property box:** A panel on the right showing the configuration for the selected `target` item. The configuration is as follows:
 

Property	Value
Target:	target
local-port	1
remote-port	1
signal-in	mux.out
signal-out	mux.in
forward	<input checked="" type="checkbox"/>
send-parent-value	<input type="checkbox"/>

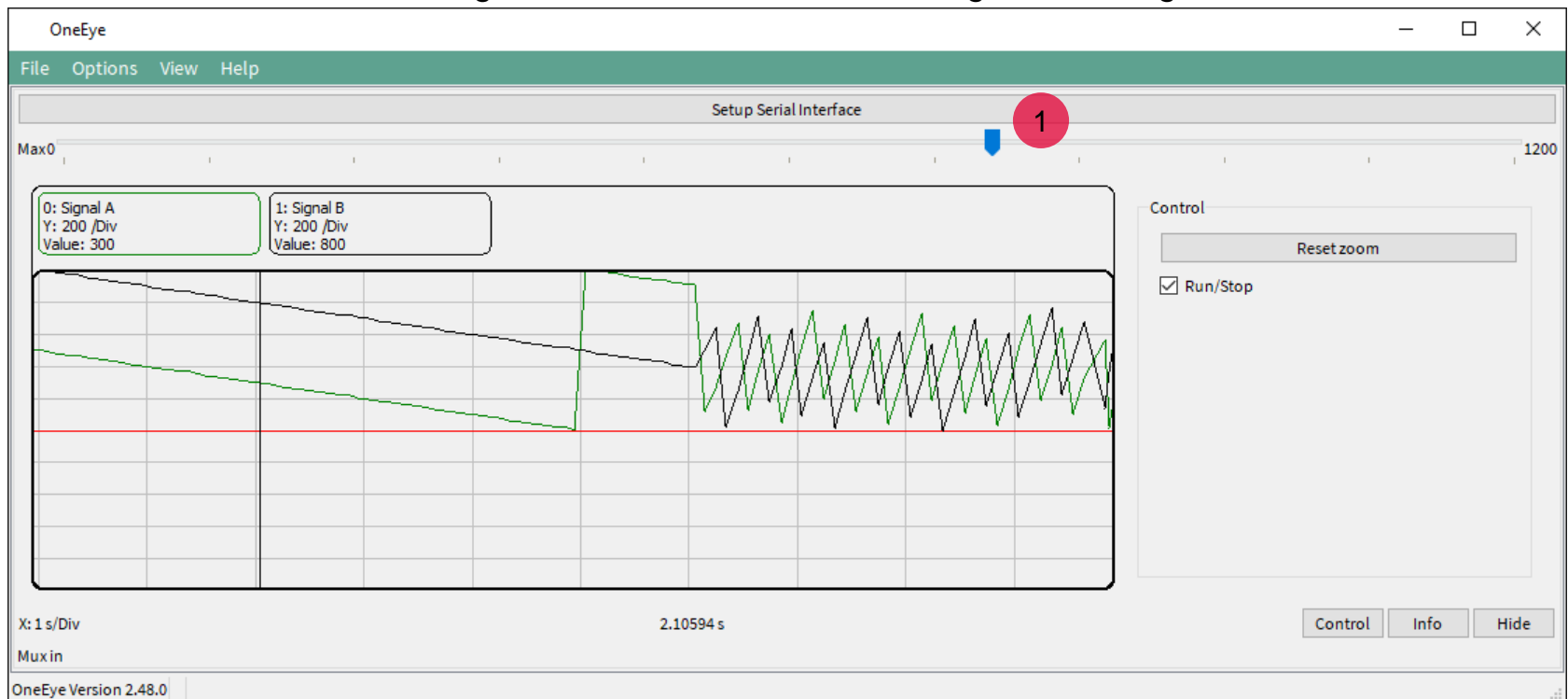
# Implementation - OneEye

## Test the data multiplexer interface

Save your configuration with CTRL+S and, exit the edit mode with the OneEye menu “**Options -> Edit mode**” to only see the GUI.

Restart the AURIX software.

Move the slider cursor **1** to change the max value and affect the generated signals value.



# References



- › AURIX™ Development Studio is available online:
- › <https://www.infineon.com/aurixdevelopmentstudio>
- › Use the „*Import...*“ function to get access to more code examples.



- › More code examples can be found on the GIT repository:
- › [https://github.com/Infineon/AURIX\\_code\\_examples](https://github.com/Infineon/AURIX_code_examples)



- › For additional trainings, visit our webpage:
- › <https://www.infineon.com/aurix-expert-training>



- › For questions and support, use the AURIX™ Forum:
- › <https://www.infineonforums.com/forums/13-Aurix-Forum>

## Trademarks

All referenced product or service names and trademarks are the property of their respective owners.

**Edition 2022-06**

**Published by**

**Infineon Technologies AG  
81726 Munich, Germany**

**© 2021 Infineon Technologies AG.  
All Rights Reserved.**

**Do you have a question about this document?**

**Email: [erratum@infineon.com](mailto:erratum@infineon.com)**

**Document reference**

**OneEye\_UART\_Mux\_1  
\_KIT\_TC375\_LK**

## IMPORTANT NOTICE

The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics (“Beschaffenheitsgarantie”).

With respect to any examples, hints or any typical values stated herein and/or any information regarding the application of the product, Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind, including without limitation warranties of non-infringement of intellectual property rights of any third party.

In addition, any information given in this document is subject to customer’s compliance with its obligations stated in this document and any applicable legal requirements, norms and standards concerning customer’s products and any use of the product of Infineon Technologies in customer’s applications.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer’s technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

For further information on the product, technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies office ([www.infineon.com](http://www.infineon.com)).

## WARNINGS

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies’ products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.