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

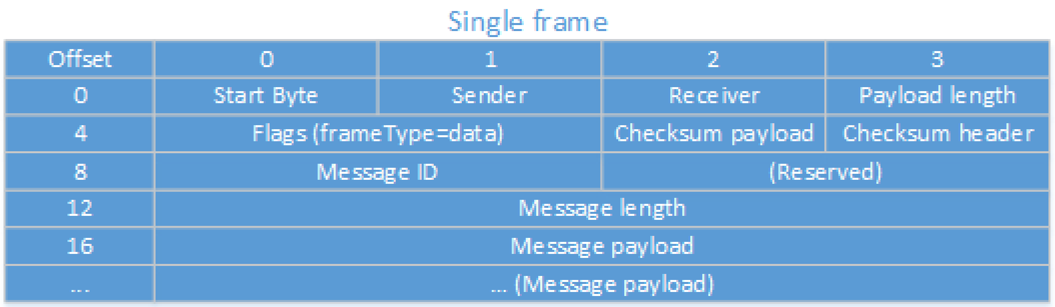
Demonstrate how to implement the data multiplexer over the UART (USB) interface

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

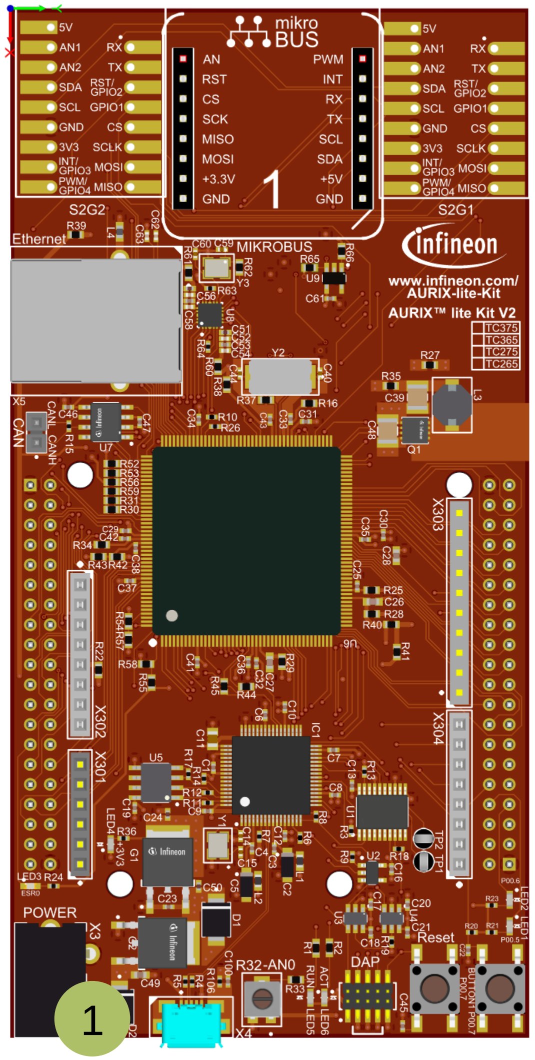
Offset	0	1	2	3
0	Start Byte	Sender	Receiver	Payload length
4	Flags (frameType=data)		Checksum payload	Checksum header
8	Message ID		(Reserved)	
12	Message length			
16	Message payload			
...	... (Message payload)			

› **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

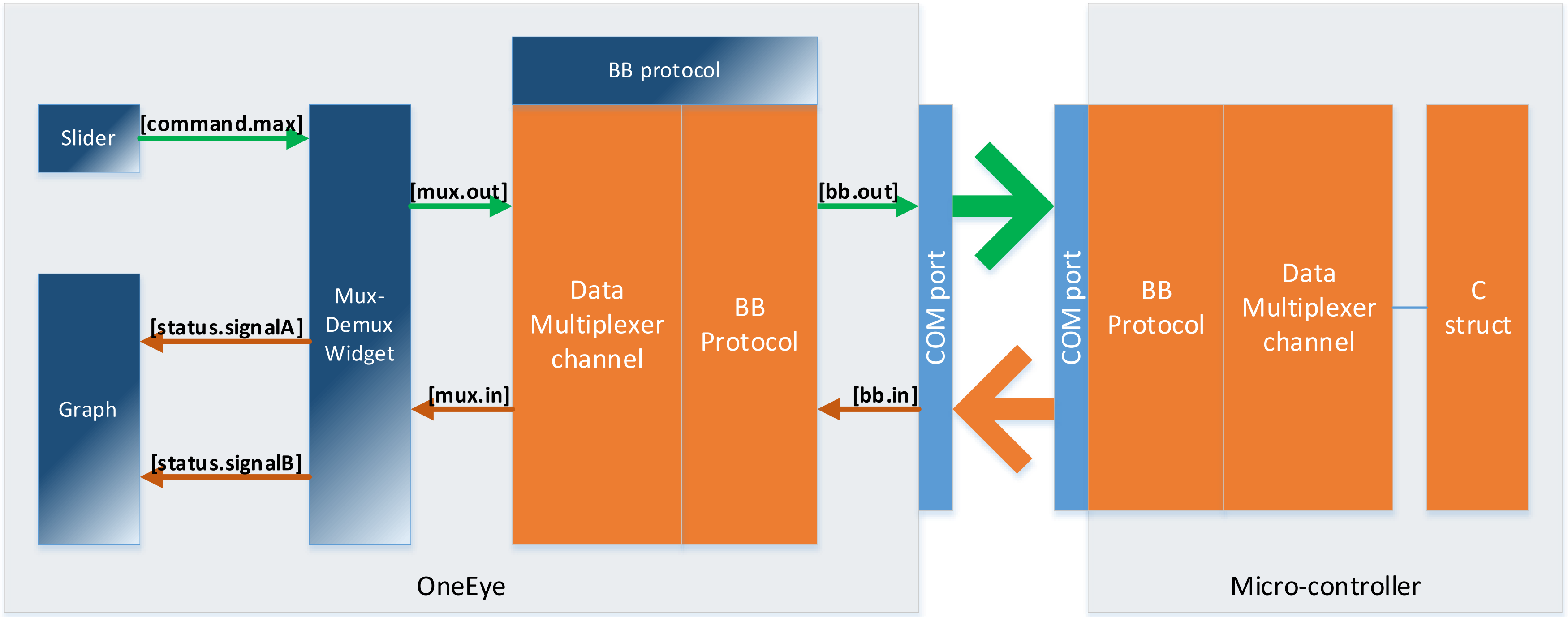
This code example has been developed for the board KIT_A2G_TC375_LITE.

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 massage 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

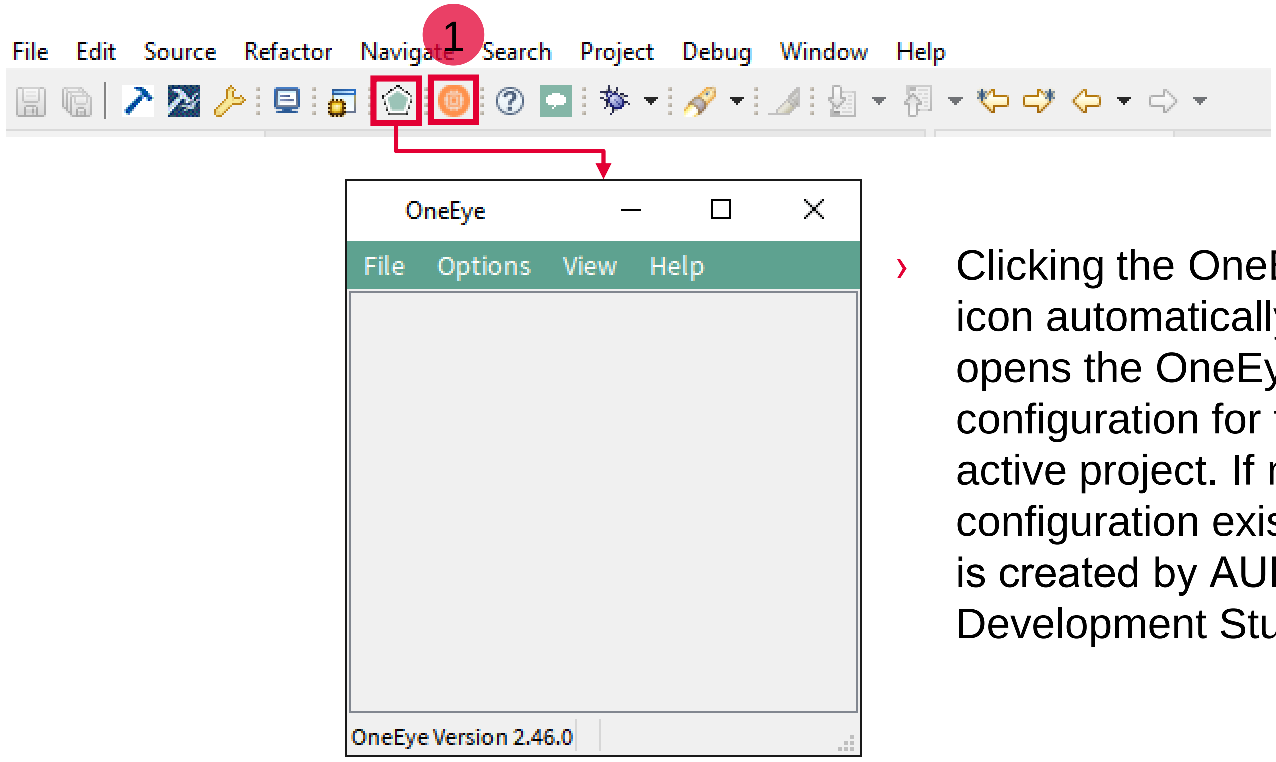
A signal generator is used to provide the user with some value to read / write. The signal generator does nothing more than incrementing two signals, ***signalA*** and ***signalB***, stored in the structure ***g_signalGenerator*** up to a maximum value before resetting them.
The initialization of the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



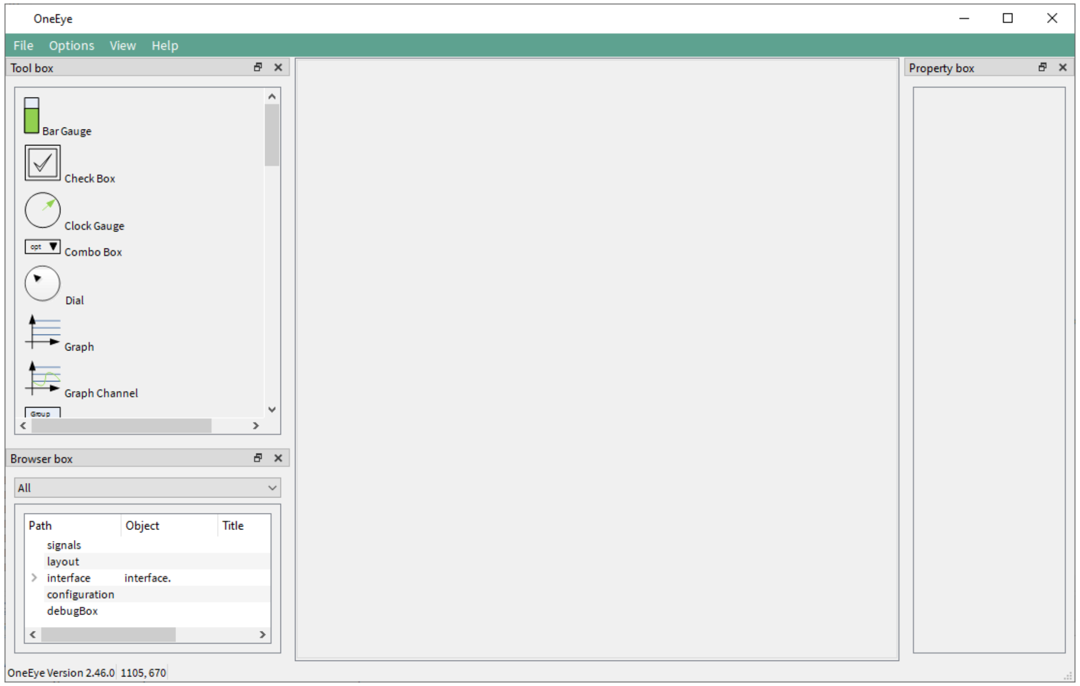
› Clicking the OneEye icon automatically opens the OneEye configuration for the active project. If no configuration exists, it is created by AURIX™ Development Studio

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

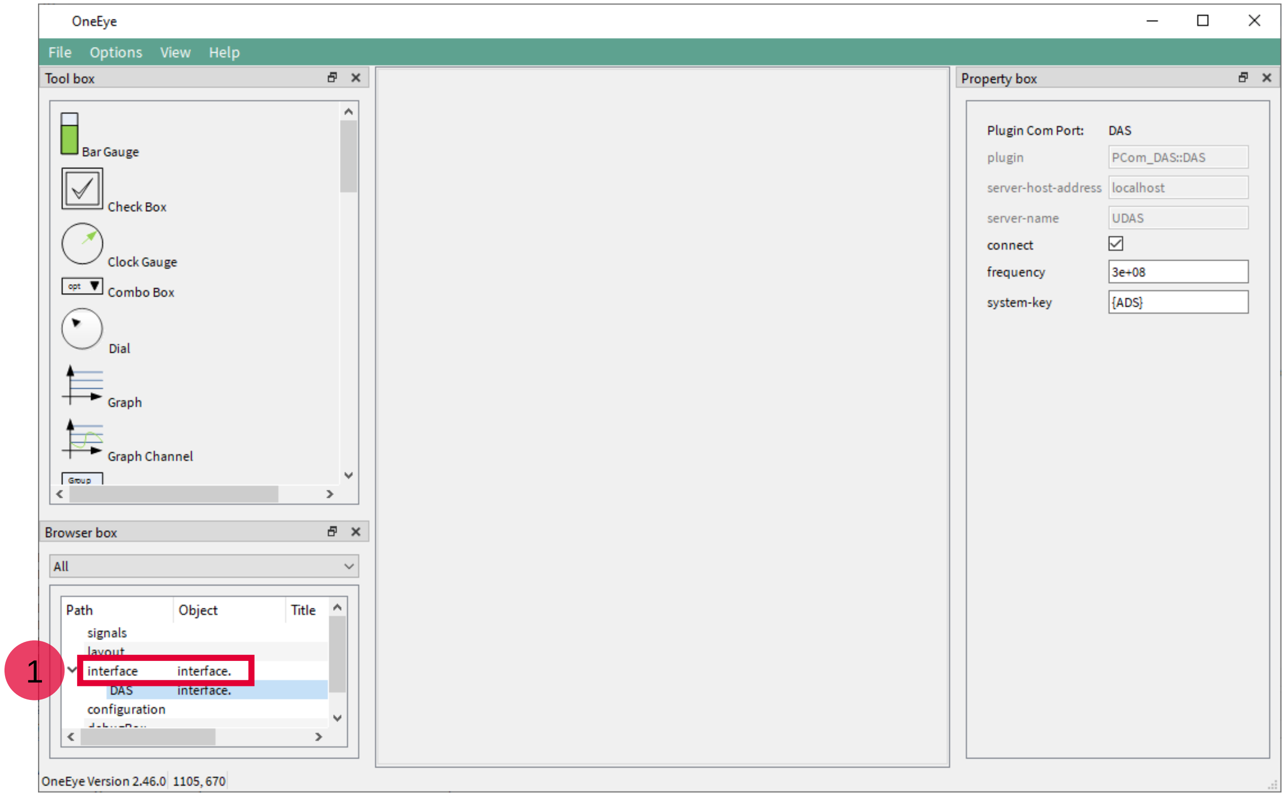
Select the OneEye menu “**Options -> Edit mode**” (if not already checked) to enable the edit mode.
Select the OneEye menu “**View -> Browser box**”, “**View -> Property box**” , “**View -> Tool box**” (if not already checked) to display the browser, property box, and tool box. Note that the box can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



Plugin Com Port:	DAS
plugin	PCom_DAS::DAS
server-host-address	localhost
server-name	UDAS
connect	✓
frequency	3e+08
system-key	{ADS}

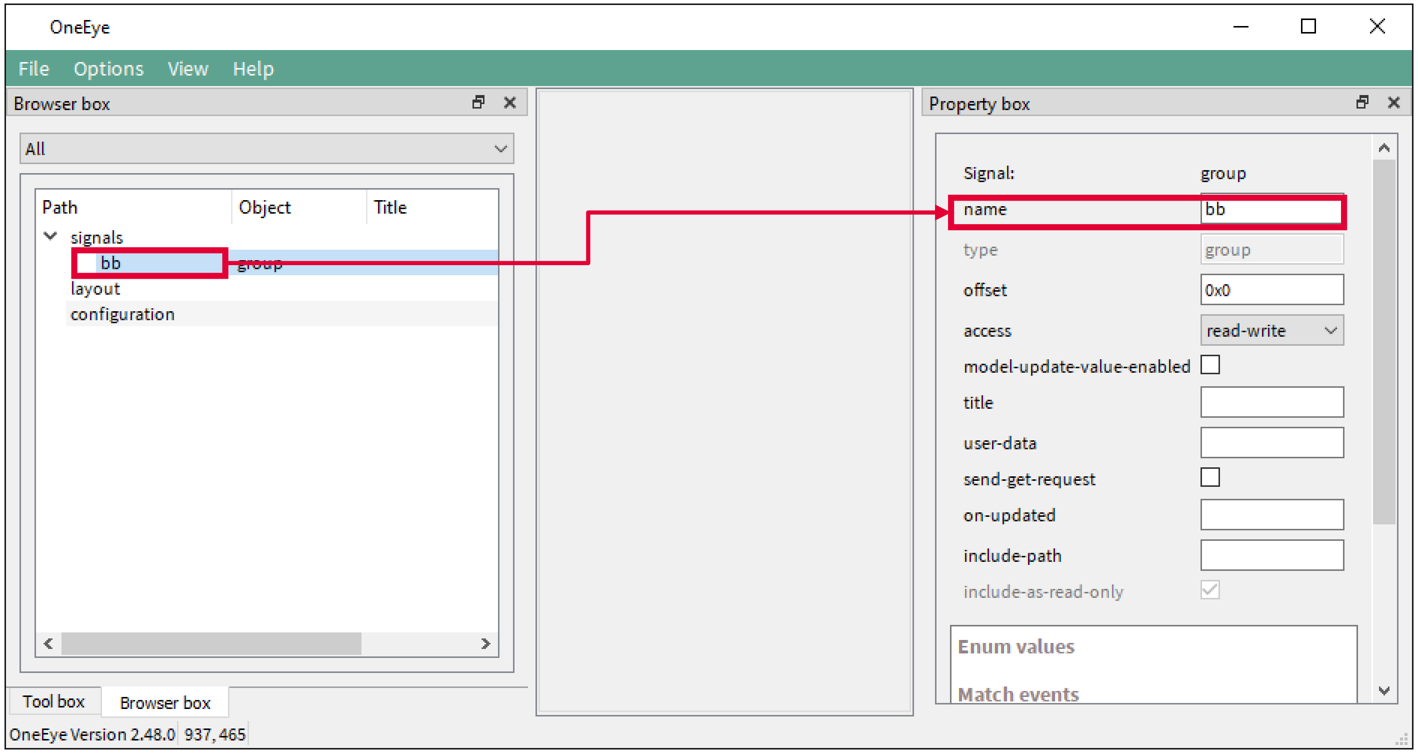
OneEye Version 2.46.0 1105, 670

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.

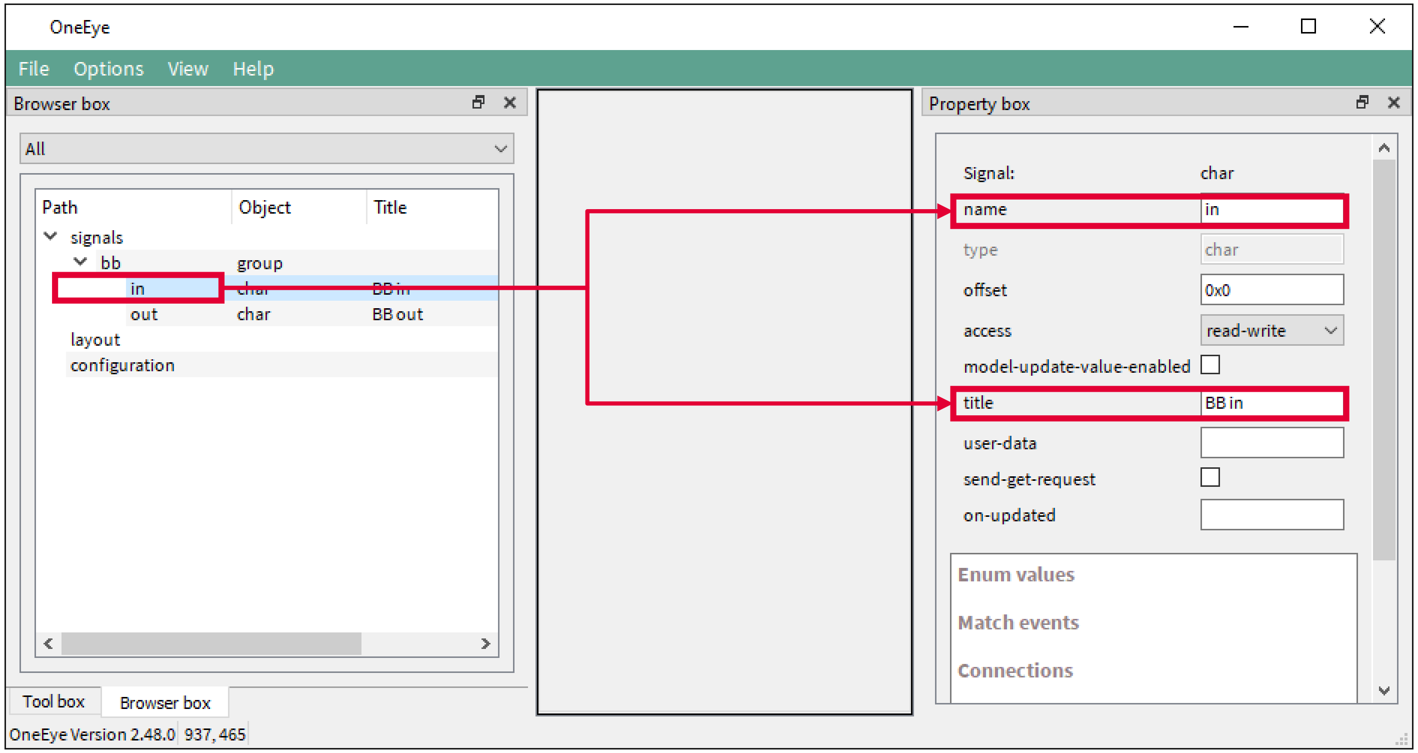
Create a signal group and set its **name** property to **bb**.



Signal:	group
name	bb
type	group
offset	0x0
access	read-write
model-update-value-enabled	☐
title	
user-data	
send-get-request	☐
on-updated	
include-path	
include-as-read-only	✓

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**.



Path	Object	Title
signals		
bb	group	
in	char	BB in
out	char	BB out
layout		
configuration		

Signal:	char
name	in
type	char
offset	0x0
access	read-write
model-update-value-enabled	☐
title	BB in
user-data	
send-get-request	☐
on-updated	

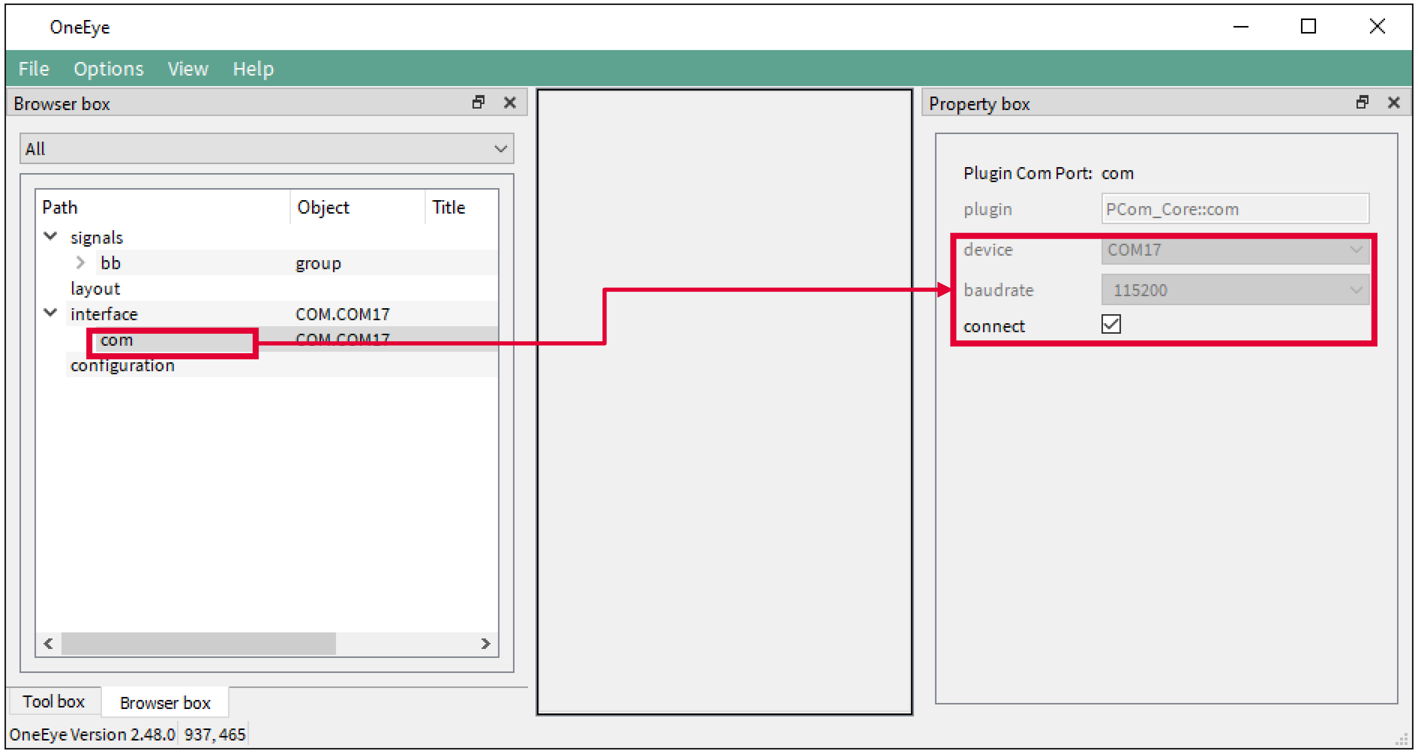
OneEye Version 2.48.0 937, 465

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.



Path	Object	Title
signals		
bb	group	
layout		
interface	COM.COM17	
com	COM.COM17	
configuration		

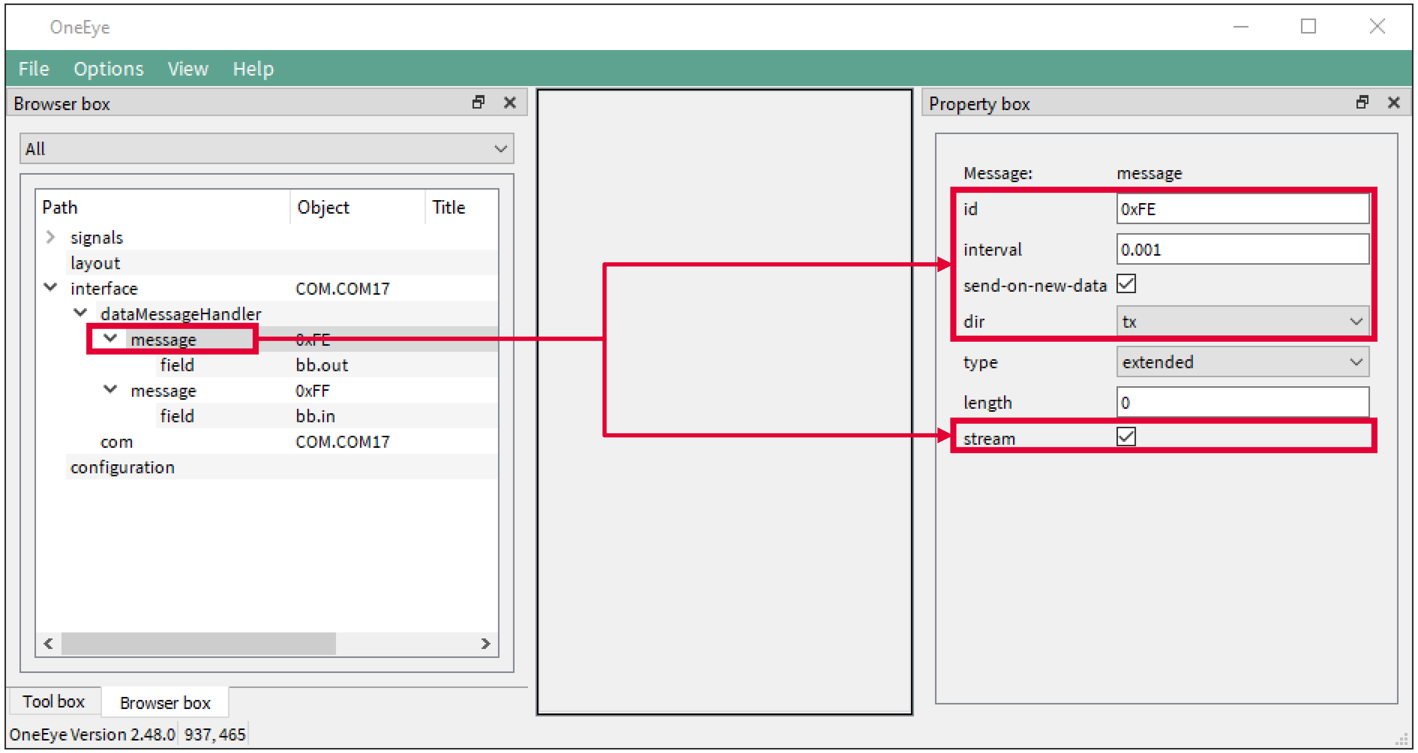
Plugin Com Port: com

plugin	PCom_Core::com
device	COM17
baudrate	115200
connect	✓

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.



Path	Object	Title
signals		
layout		
interface	COM.COM17	
dataMessageHandler		
message	0xFE	
field	bb.out	
message	0xFF	
field	bb.in	
com	COM.COM17	
configuration		

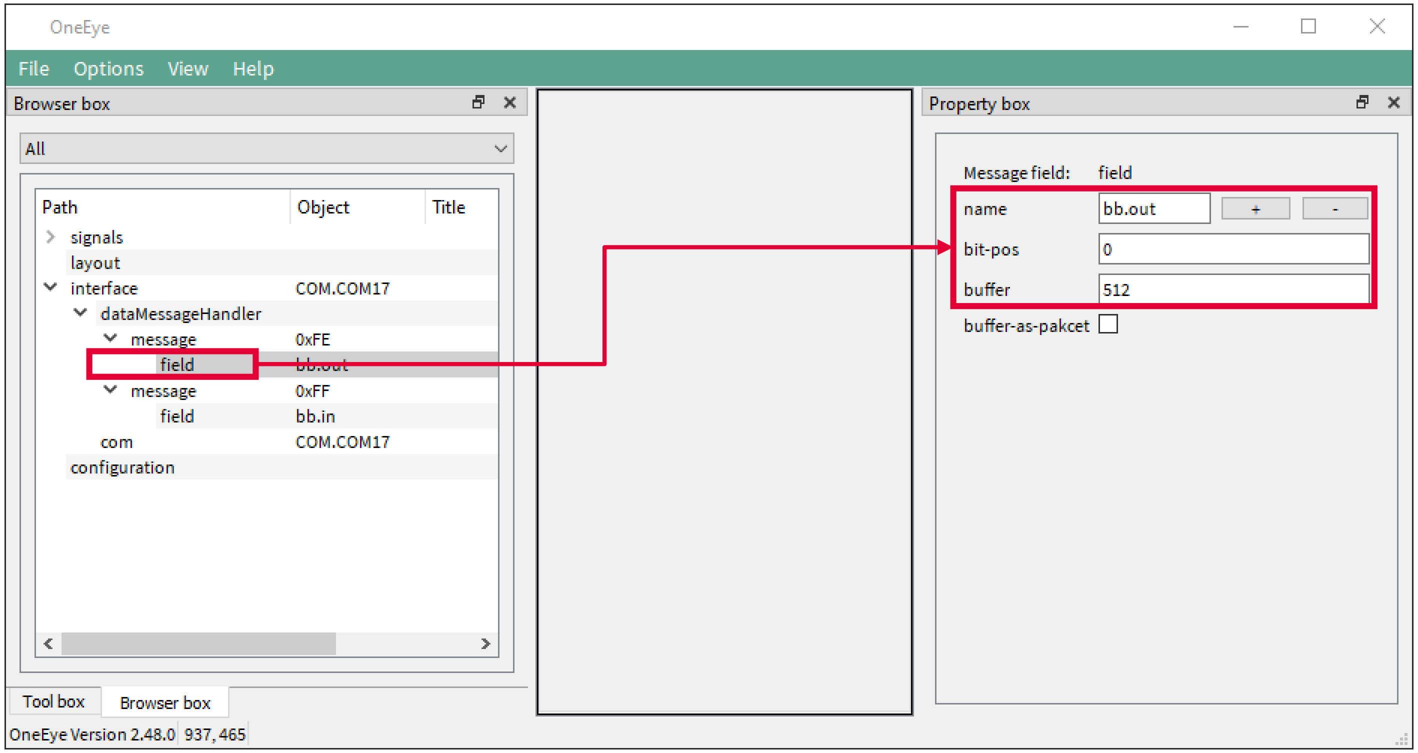
Message:	message
id	0xFE
interval	0.001
send-on-new-data	✓
dir	tx
type	extended
length	0
stream	✓

OneEye Version 2.48.0 937, 465

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.



Path	Object	Title
signals		
layout		
interface	COM.COM17	
dataMessageHandler		
message	0xFE	
field	bb.out	
message	0xFF	
field	bb.in	
com	COM.COM17	
configuration		

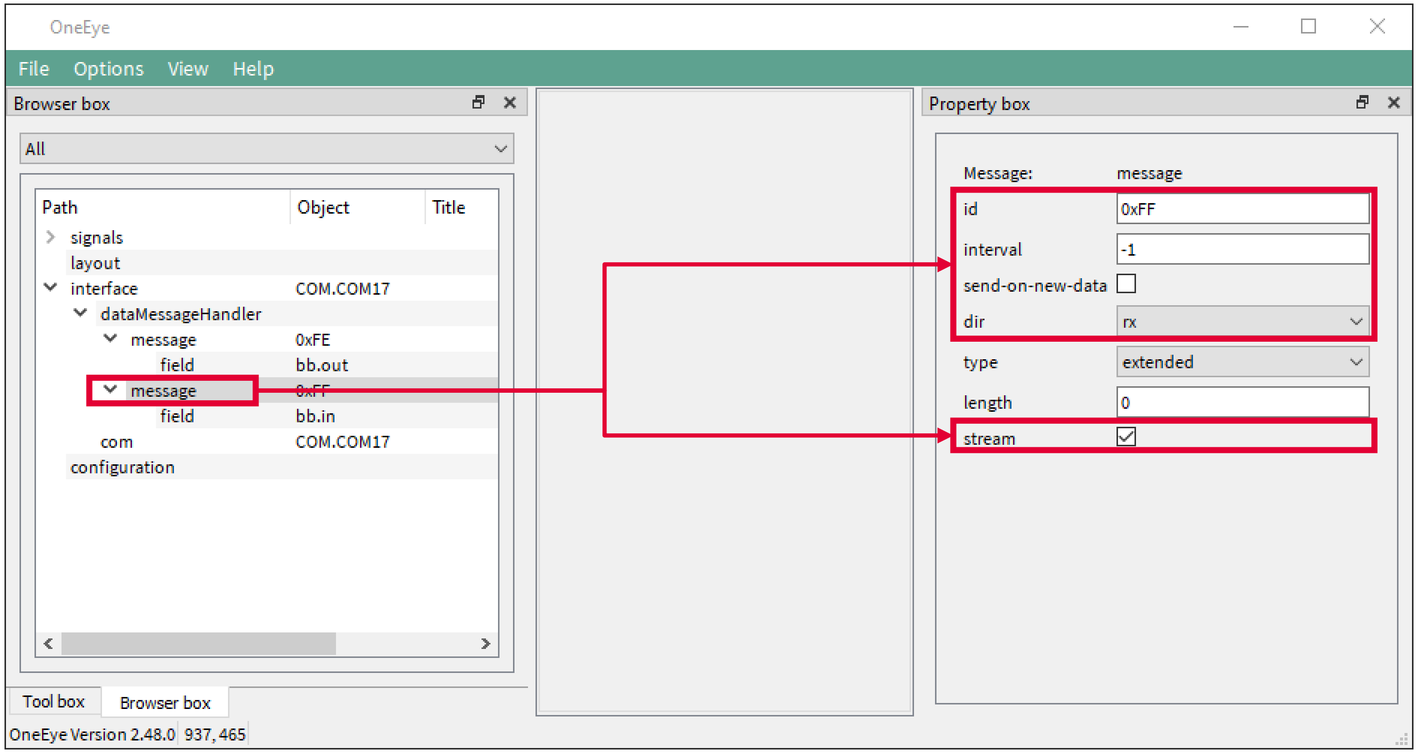
Message field: field

name	bb.out
bit-pos	0
buffer	512
buffer-as-pakcet	☐

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.



Path	Object	Title
signals		
layout		
interface	COM.COM17	
dataMessageHandler		
message	0xFE	
field	bb.out	
message	0xFF	
field	bb.in	
com	COM.COM17	
configuration		

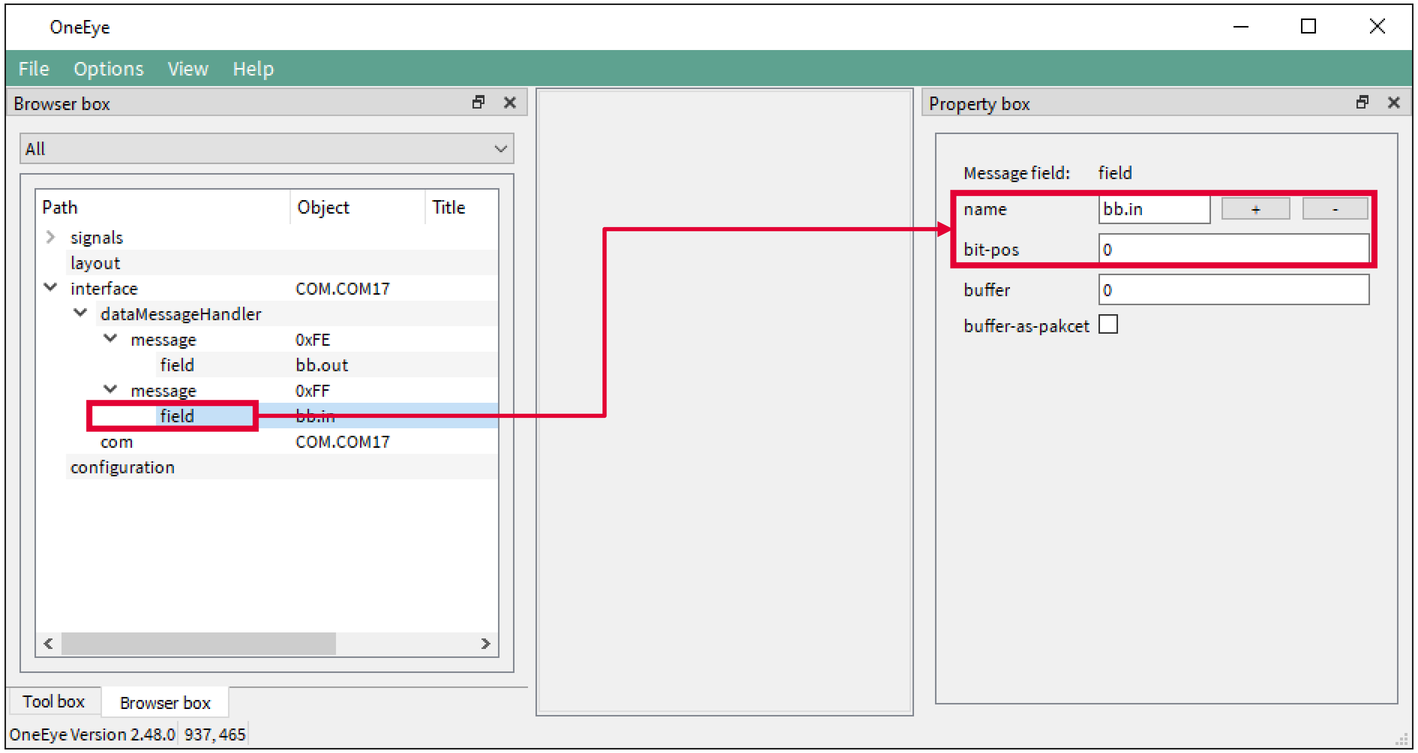
Message:	message
id	0xFF
interval	-1
send-on-new-data	☐
dir	rx
type	extended
length	0
stream	✓

OneEye Version 2.48.0 937, 465

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.



Path	Object	Title
signals		
layout		
interface	COM.COM17	
dataMessageHandler		
message	0xFE	
field	bb.out	
message	0xFF	
field	bb.in	
com	COM.COM17	
configuration		

Message field: field

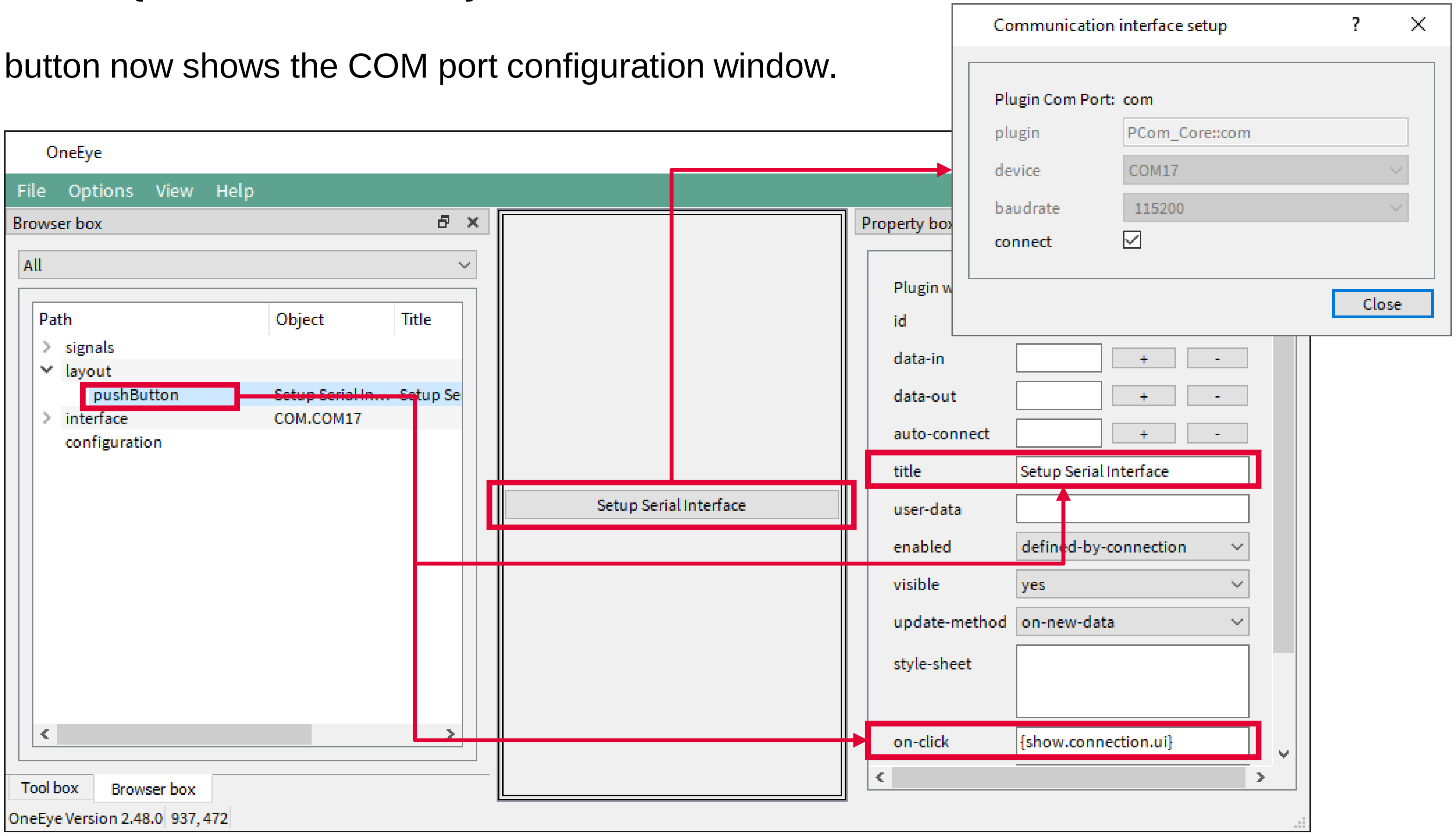
name	bb.in
bit-pos	0
buffer	0
buffer-as-pakcet	☐

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.



Path	Object	Title
signals		
layout		
pushButton	Setup Serial In...	Setup Se
interface	COM.COM17	
configuration		

Plugin Com Port: com

plugin	PCom_Core::com
device	COM17
baudrate	115200
connect	✓

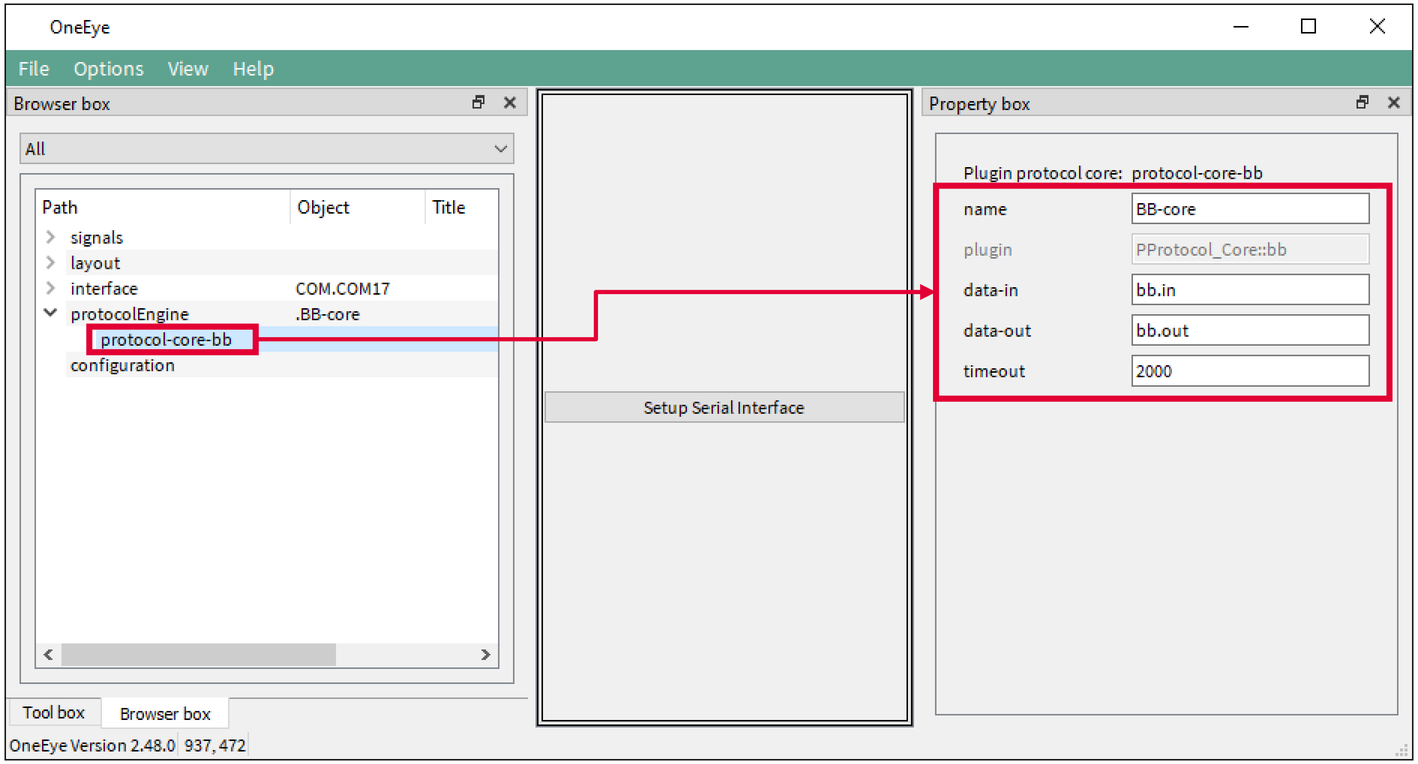
Plugin w	
id	
data-in	
data-out	
auto-connect	
title	Setup Serial Interface
user-data	
enabled	defined-by-connection
visible	yes
update-method	on-new-data
style-sheet	
on-click	{show.connection.ui}

OneEye Version 2.48.0 937, 472

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.



Path	Object	Title
signals		
layout		
interface	COM.COM17	
protocolEngine	.BB-core	
protocol-core-bb		
configuration		

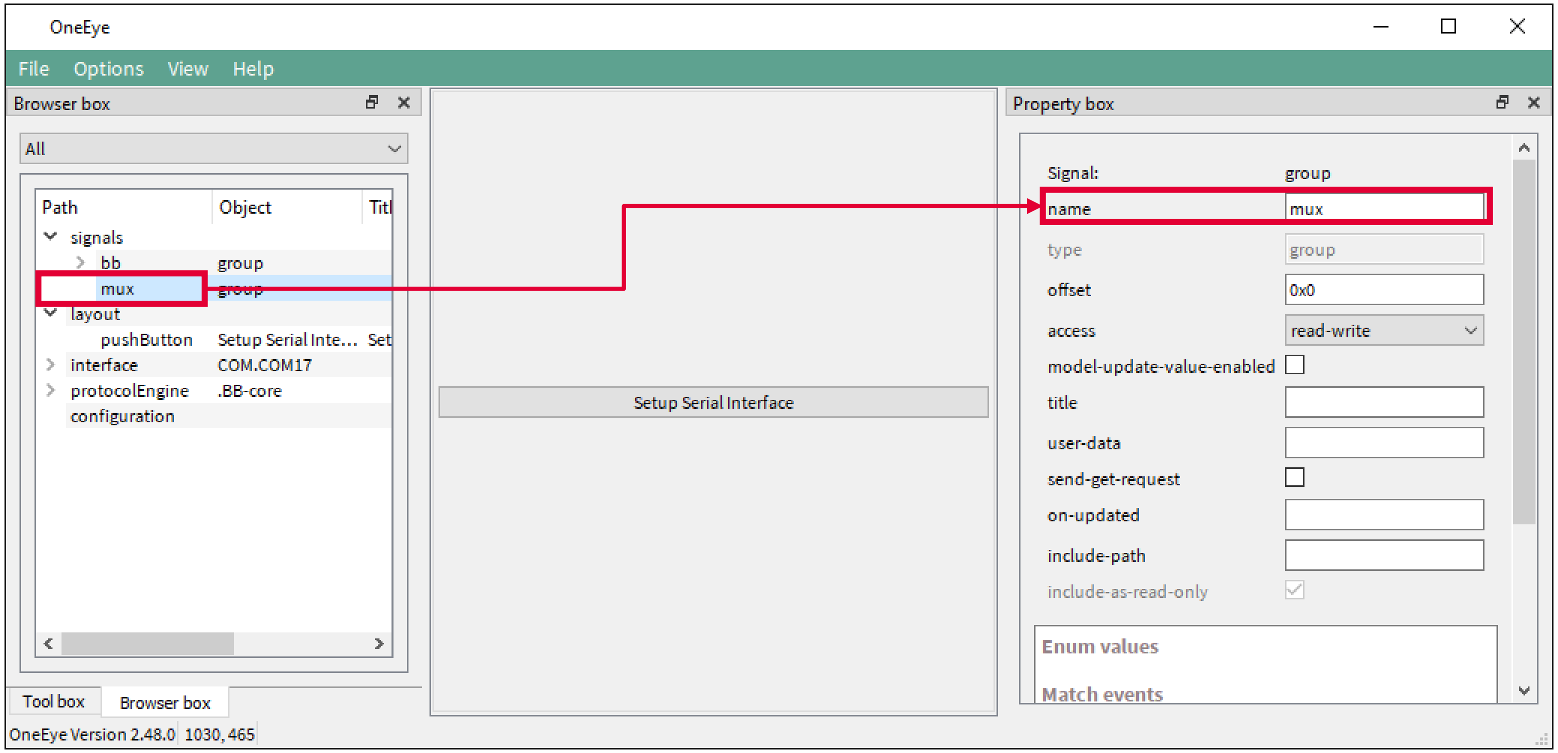
Plugin protocol core: protocol-core-bb

name	BB-core
plugin	PProtocol_Core::bb
data-in	bb.in
data-out	bb.out
timeout	2000

Implementation - OneEye

Configuring the Data multiplexer: signals creation

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



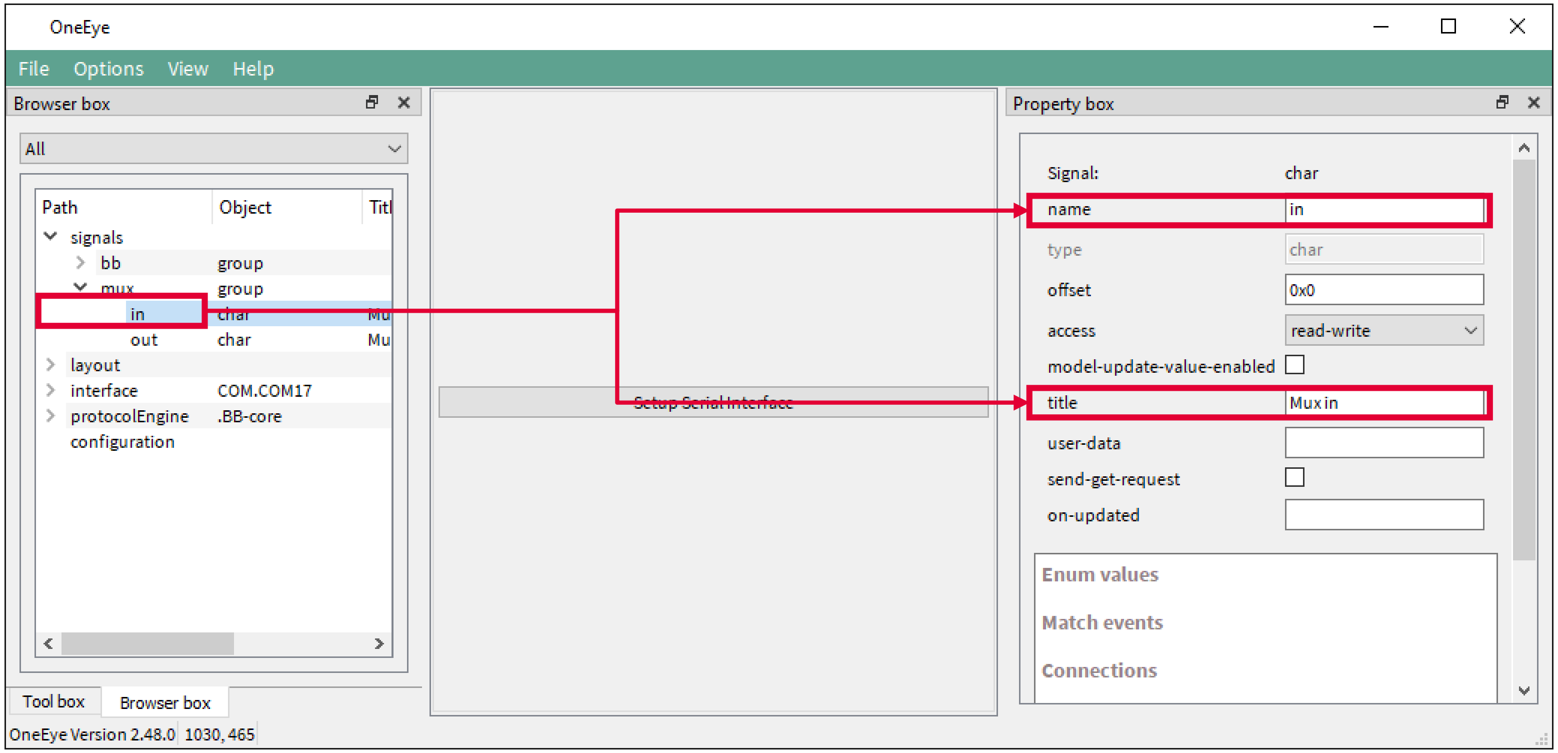
Path	Object	Titl
signals		
bb	group	
mux	group	
layout		
pushButton	Setup Serial Inte…	Set
interface	COM.COM17	
protocolEngine	.BB-core	
configuration		

Signal:	group
name	mux
type	group
offset	0x0
access	read-write
model-update-value-enabled	☐
title	
user-data	
send-get-request	☐
on-updated	
include-path	
include-as-read-only	✓

OneEye Version 2.48.0 1030, 465

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**.



Path	Object	Titl
signals		
bb	group	
mux	group	
in	char	Mu
out	char	Mu
layout		
interface	COM.COM17	
protocolEngine	.BB-core	
configuration		

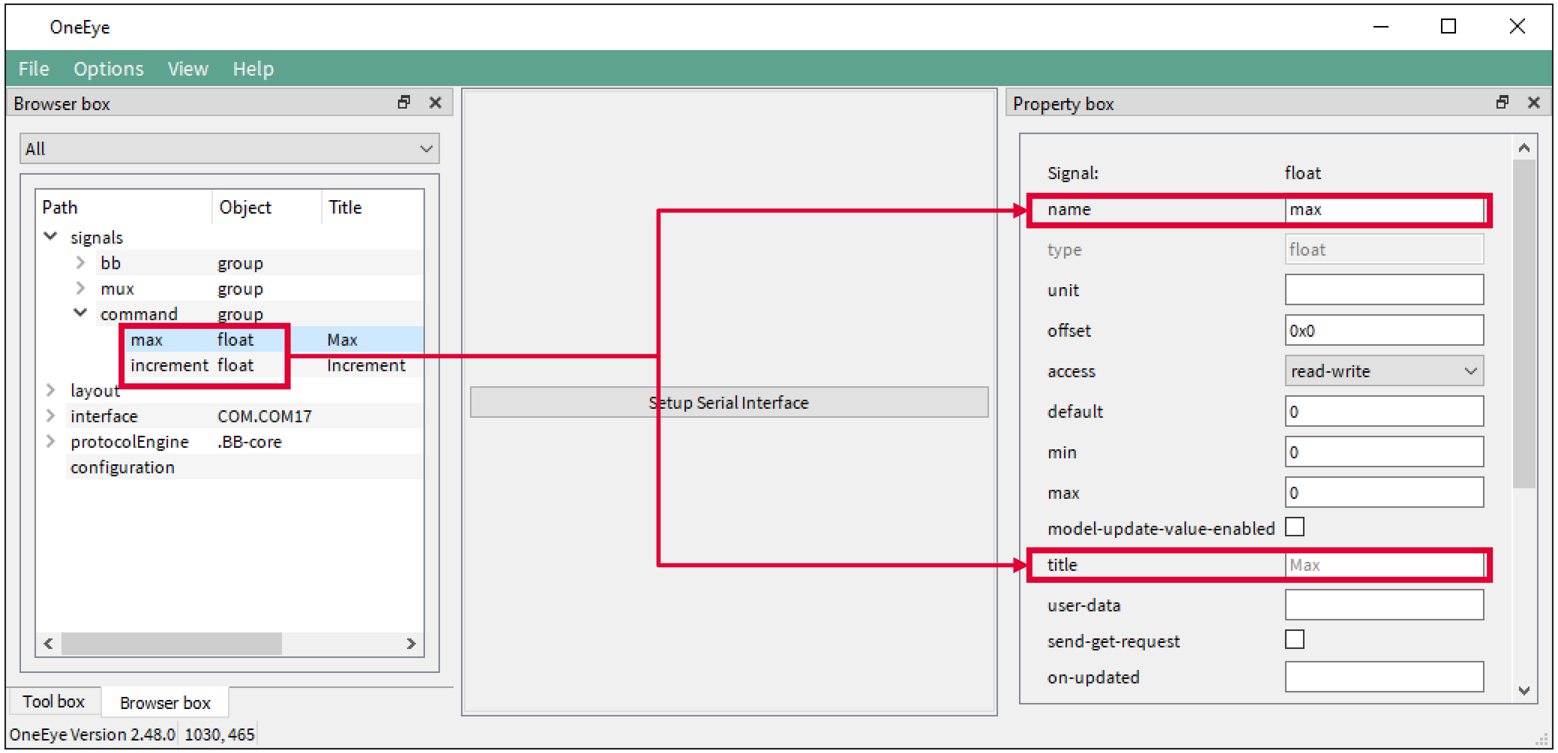
Signal:	char
name	in
type	char
offset	0x0
access	read-write
model-update-value-enabled	☐
title	Mux in
user-data	
send-get-request	☐
on-updated	

OneEye Version 2.48.0 1030, 465

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**.



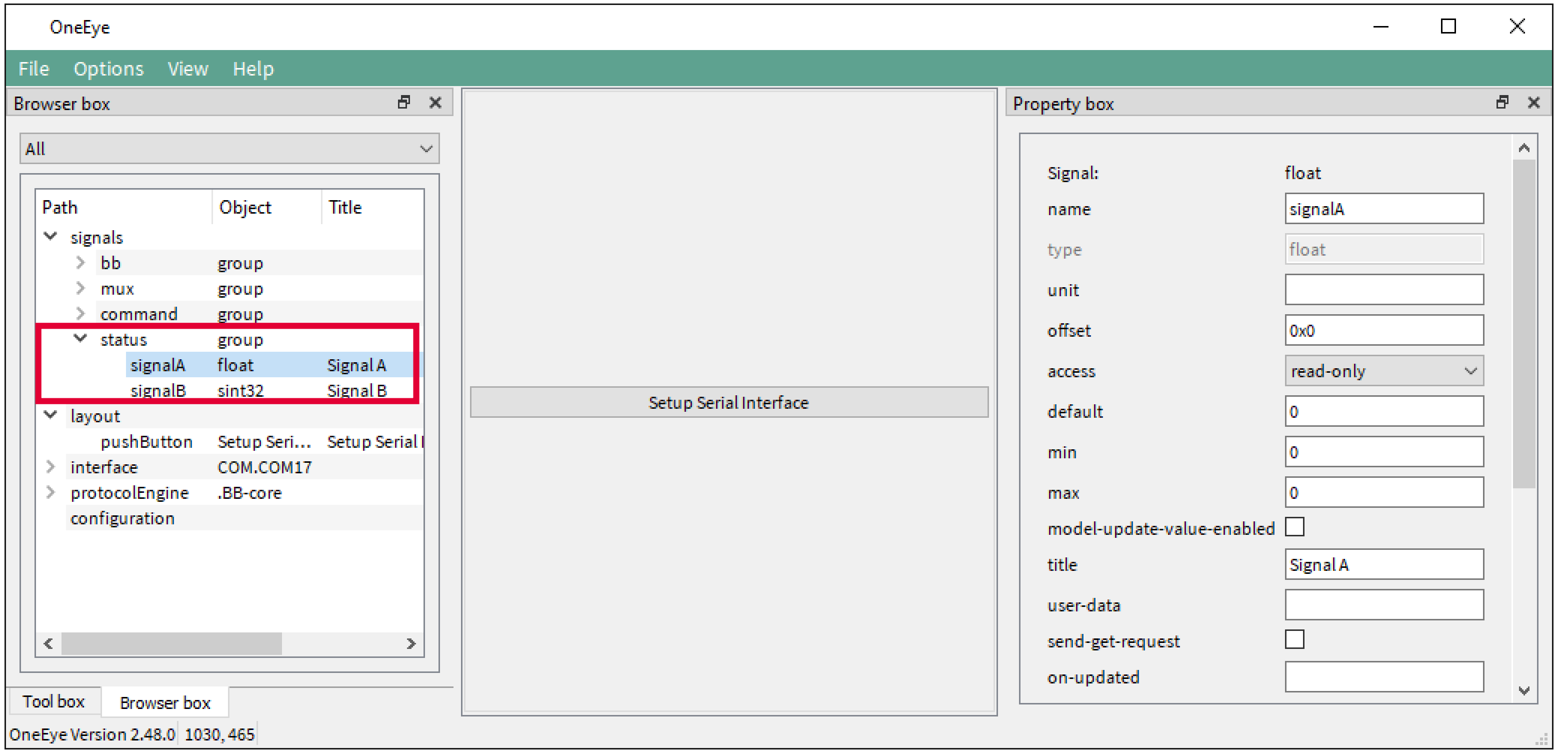
Path	Object	Title
signals		
bb	group	
mux	group	
command	group	
max	float	Max
increment	float	Increment
layout		
interface	COM.COM17	
protocolEngine	.BB-core	
configuration		

Signal:	float
name	max
type	float
unit	
offset	0x0
access	read-write
default	0
min	0
max	0
model-update-value-enabled	☐
title	Max
user-data	
send-get-request	☐
on-updated	

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**.



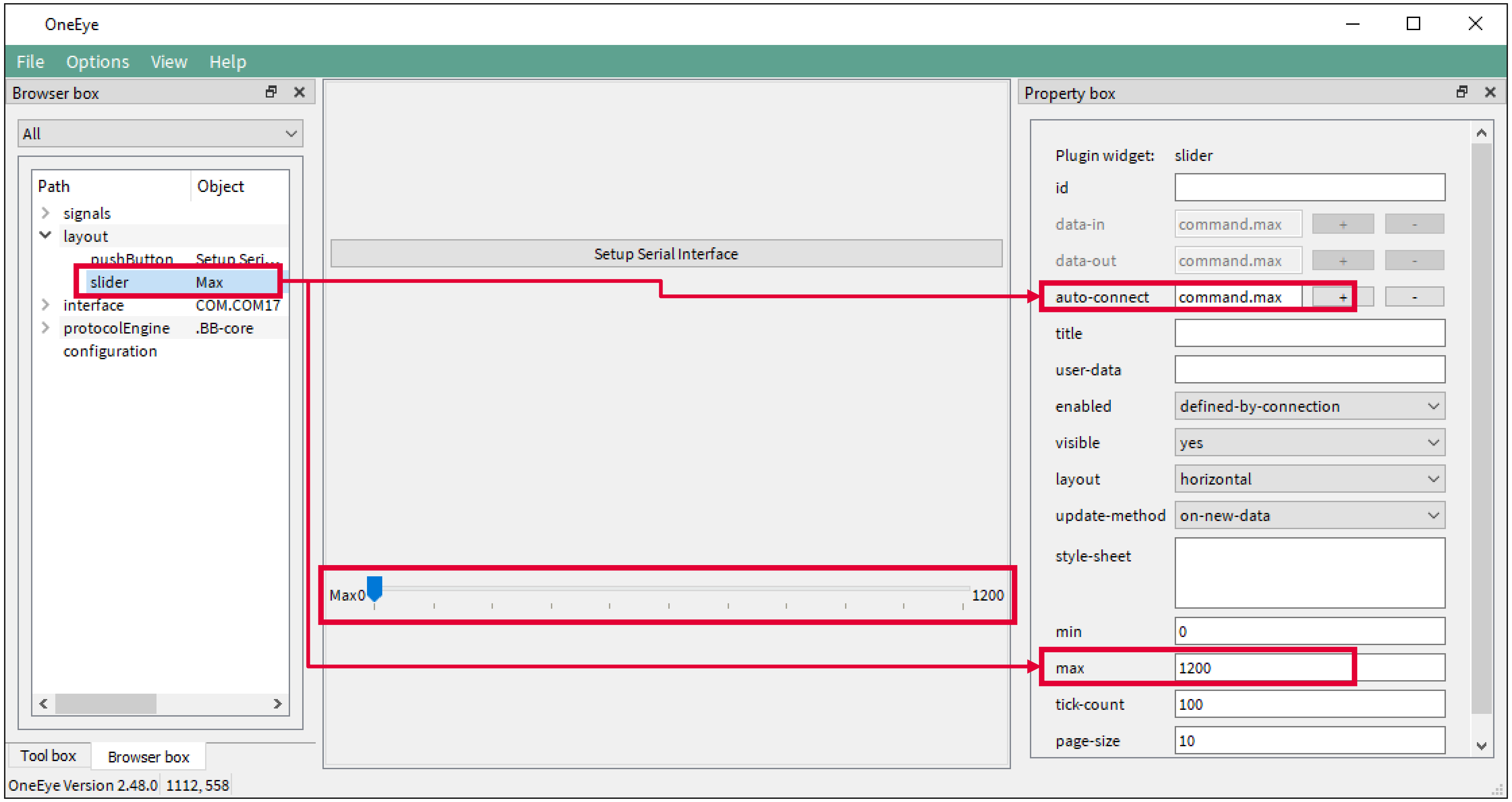
Path	Object	Title
signals		
bb	group	
mux	group	
command	group	
status	group	
signalA	float	Signal A
signalB	sint32	Signal B
layout		
pushButton	Setup Seri...	Setup Serial I
interface	COM.COM17	
protocolEngine	.BB-core	
configuration		

Signal:	float
name	signalA
type	float
unit	
offset	0x0
access	read-only
default	0
min	0
max	0
model-update-value-enabled	☐
title	Signal A
user-data	
send-get-request	☐
on-updated	

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**,

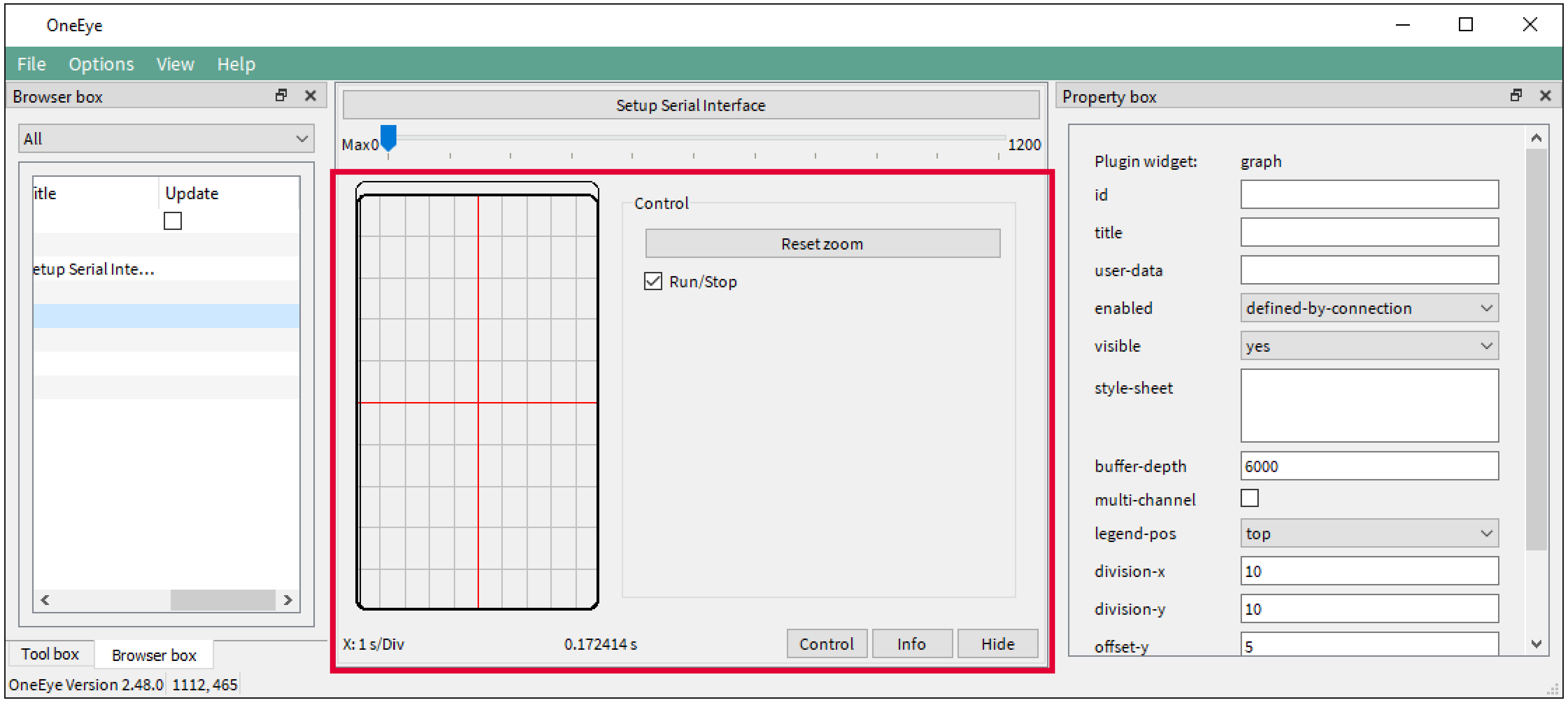


Plugin widget:	slider
id	
data-in	command.max
data-out	command.max
auto-connect	command.max
title	
user-data	
enabled	defined-by-connection
visible	yes
layout	horizontal
update-method	on-new-data
style-sheet	
min	0
max	1200
tick-count	100
page-size	10

Implementation - OneEye

Create the graph widgets to display the signals value

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

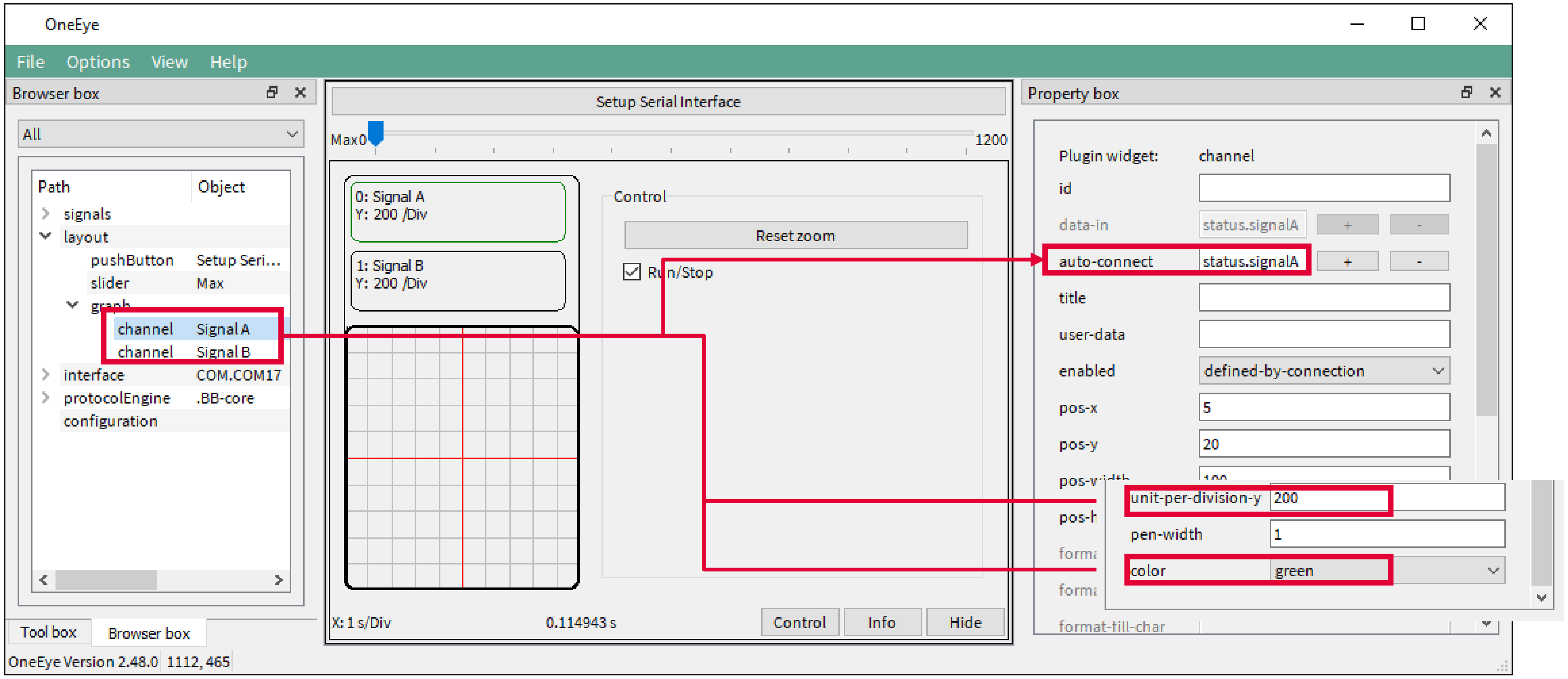


Plugin widget:	graph
id	
title	
user-data	
enabled	defined-by-connection
visible	yes
style-sheet	
buffer-depth	6000
multi-channel	☐
legend-pos	top
division-x	10
division-y	10
offset-y	5

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**.



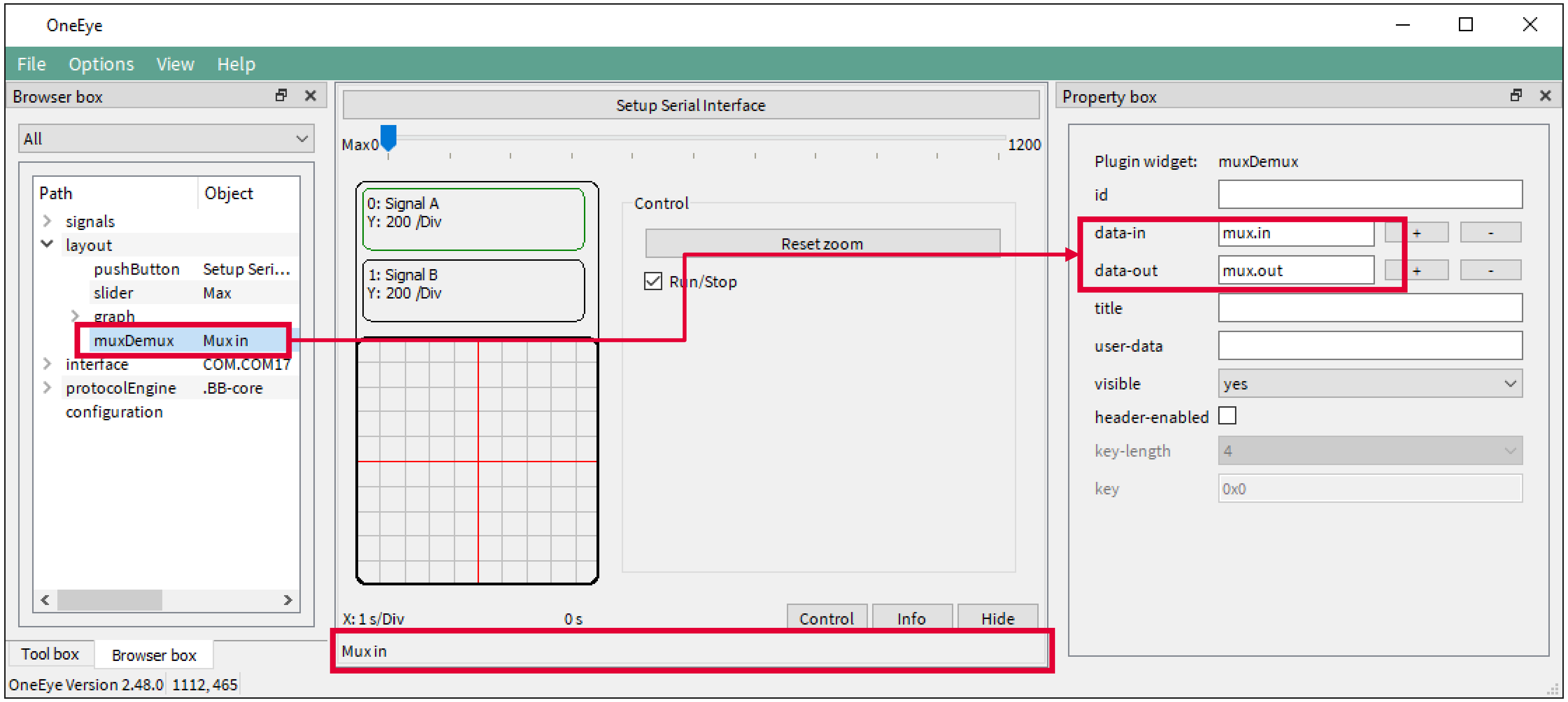
Path	Object
signals	
layout	
pushButton	Setup Seri...
slider	Max
graph	
channel	Signal A
channel	Signal B
interface	COM.COM17
protocolEngine	.BB-core
configuration	

Plugin widget:	channel
id	
data-in	status.signalA
auto-connect	status.signalA
title	
user-data	
enabled	defined-by-connection
pos-x	5
pos-y	20
unit-per-division-y	200
pen-width	1
color	green
format-fill-char	

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**.

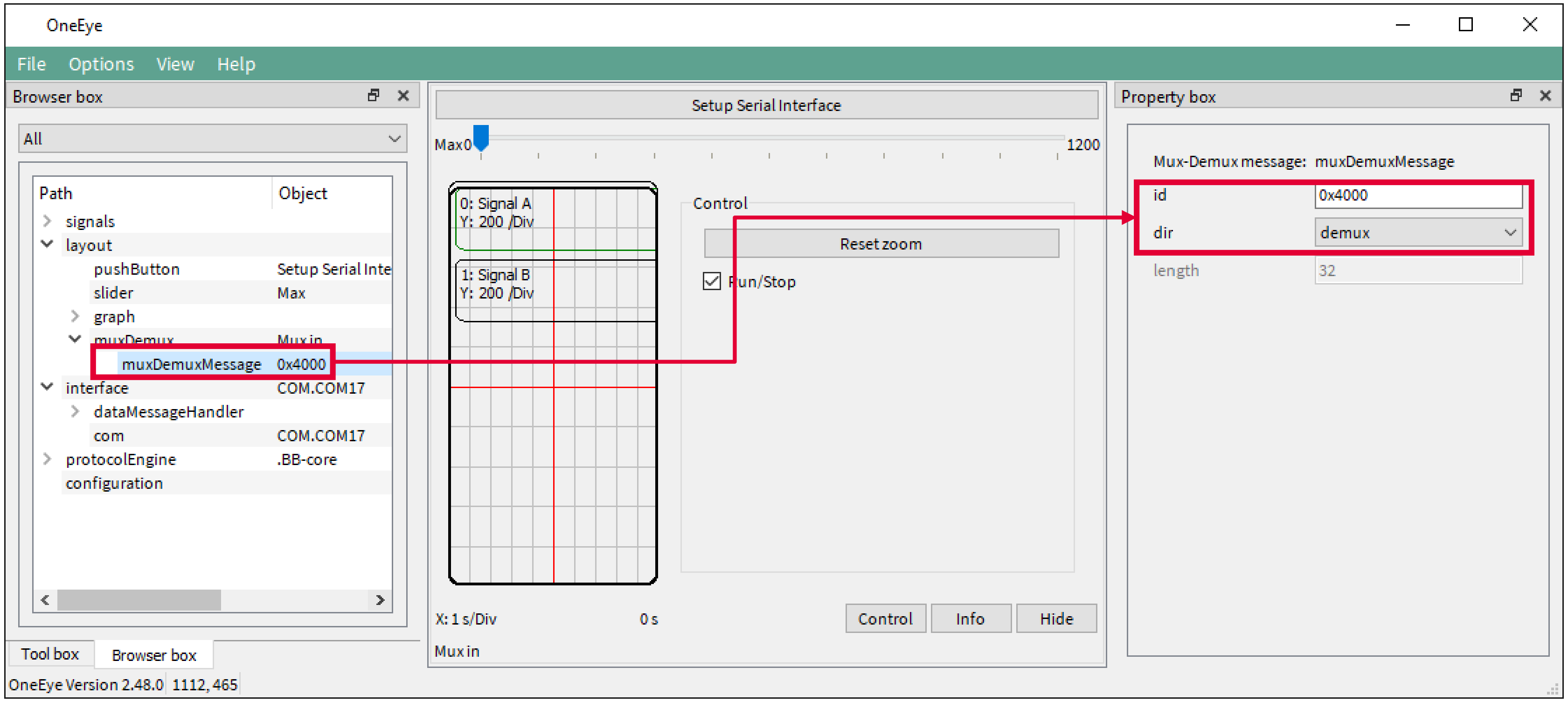


Path	Object
signals	
layout	
pushButton	Setup Seri…
slider	Max
graph	
muxDemux	Mux in
interface	COM.COM17
protocolEngine	.BB-core
configuration	

Plugin widget:	muxDemux
id	
data-in	mux.in
data-out	mux.out
title	
user-data	
visible	yes
header-enabled	☐
key-length	4
key	0x0

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.



Path	Object
signals	
layout	
pushButton	Setup Serial Inte
slider	Max
graph	
muxDemux	Mux in
muxDemuxMessage	0x4000
interface	COM.COM17
dataMessageHandler	
com	COM.COM17
protocolEngine	.BB-core
configuration	

Mux-Demux message: muxDemuxMessage

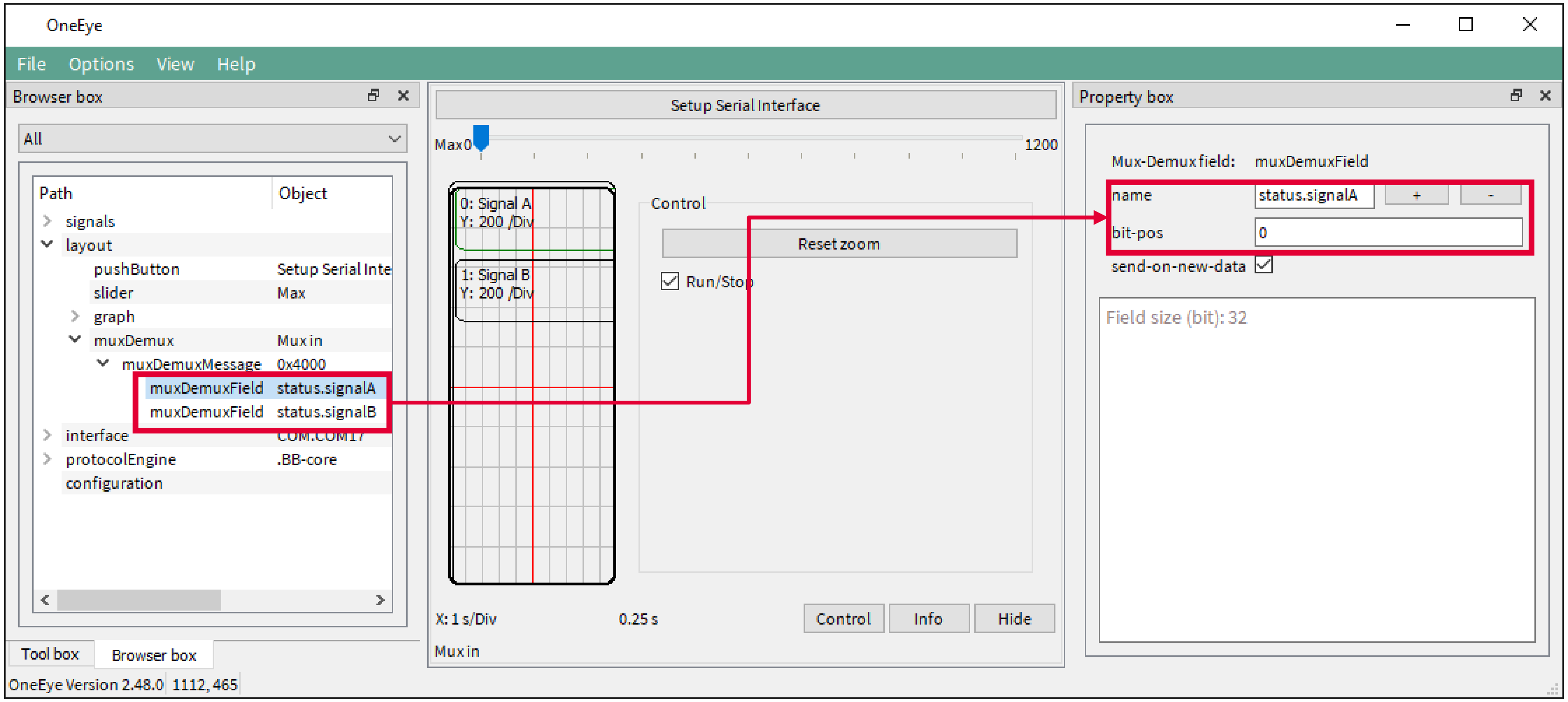
id	0x4000
dir	demux
length	32

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***.



Path	Object
signals	
layout	
pushButton	Setup Serial Inte
slider	Max
graph	
muxDemux	Mux in
muxDemuxMessage	0x4000
muxDemuxField	status.signalA
muxDemuxField	status.signalB
interface	COM.COM17
protocolEngine	.BB-core
configuration	

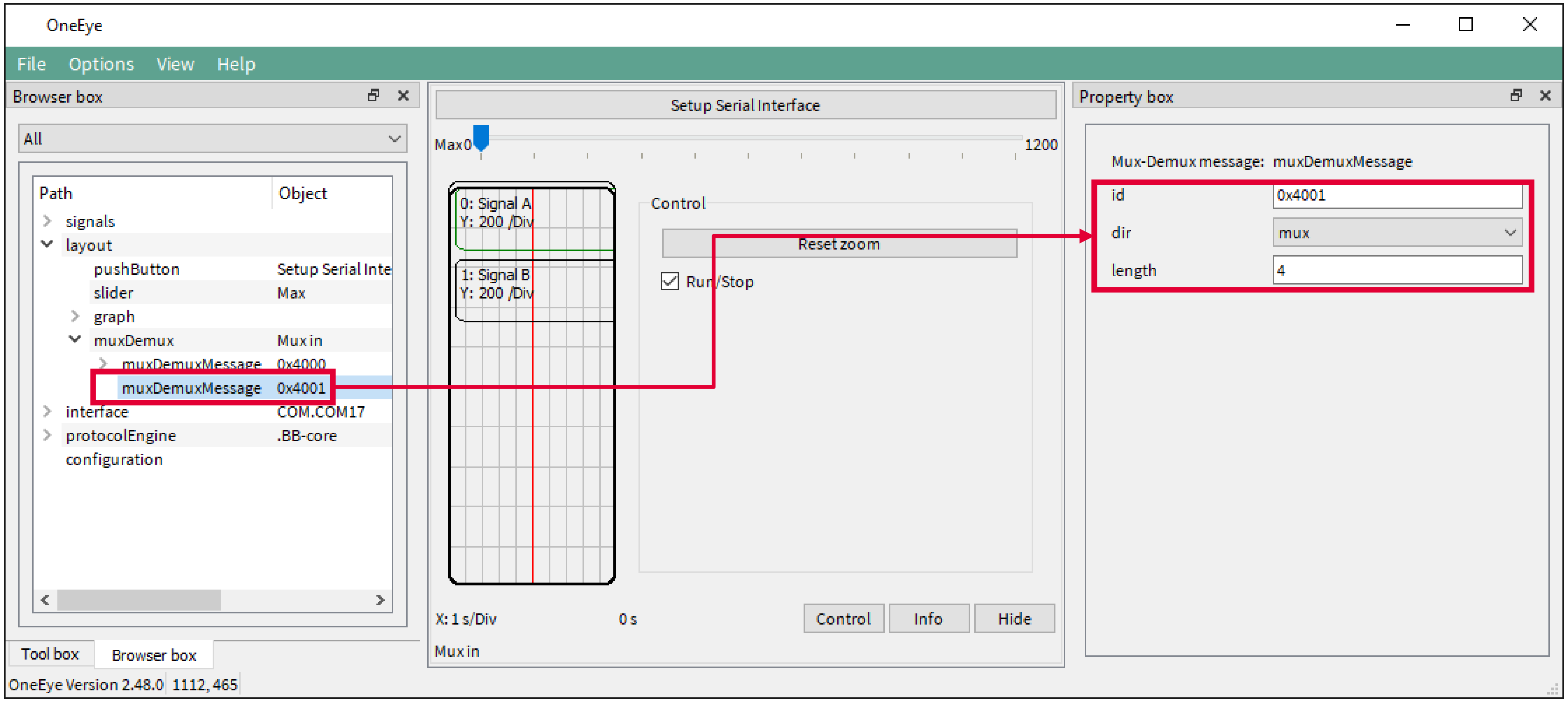
Mux-Demux field: muxDemuxField

name	status.signalA	+	-
bit-pos	0		
send-on-new-data	✓		

Field size (bit): 32

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***.



Path	Object
signals	
layout	
pushButton	Setup Serial Inte
slider	Max
graph	
muxDemux	Mux in
muxDemuxMessage	0x4000
muxDemuxMessage	0x4001
interface	COM.COM17
protocolEngine	.BB-core
configuration	

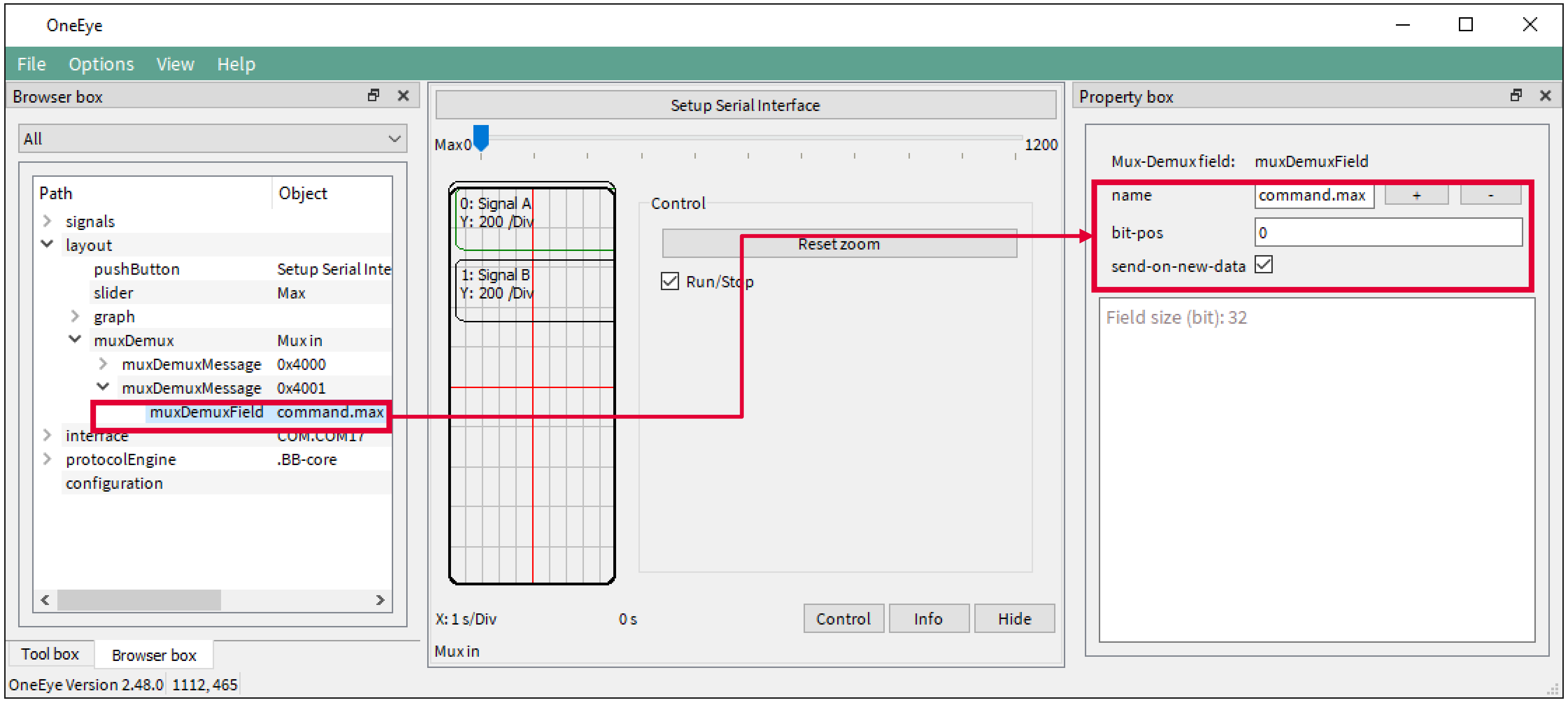
Mux-Demux message: muxDemuxMessage

id	0x4001
dir	mux
length	4

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***.



Path	Object
signals	
layout	
pushButton	Setup Serial Inte
slider	Max
graph	
muxDemux	Mux in
muxDemuxMessage	0x4000
muxDemuxMessage	0x4001
muxDemuxField	command.max
interface	COM.COM17
protocolEngine	.BB-core
configuration	

Property box

Mux-Demux field: muxDemuxField

name: command.max

bit-pos: 0

send-on-new-data: ✓

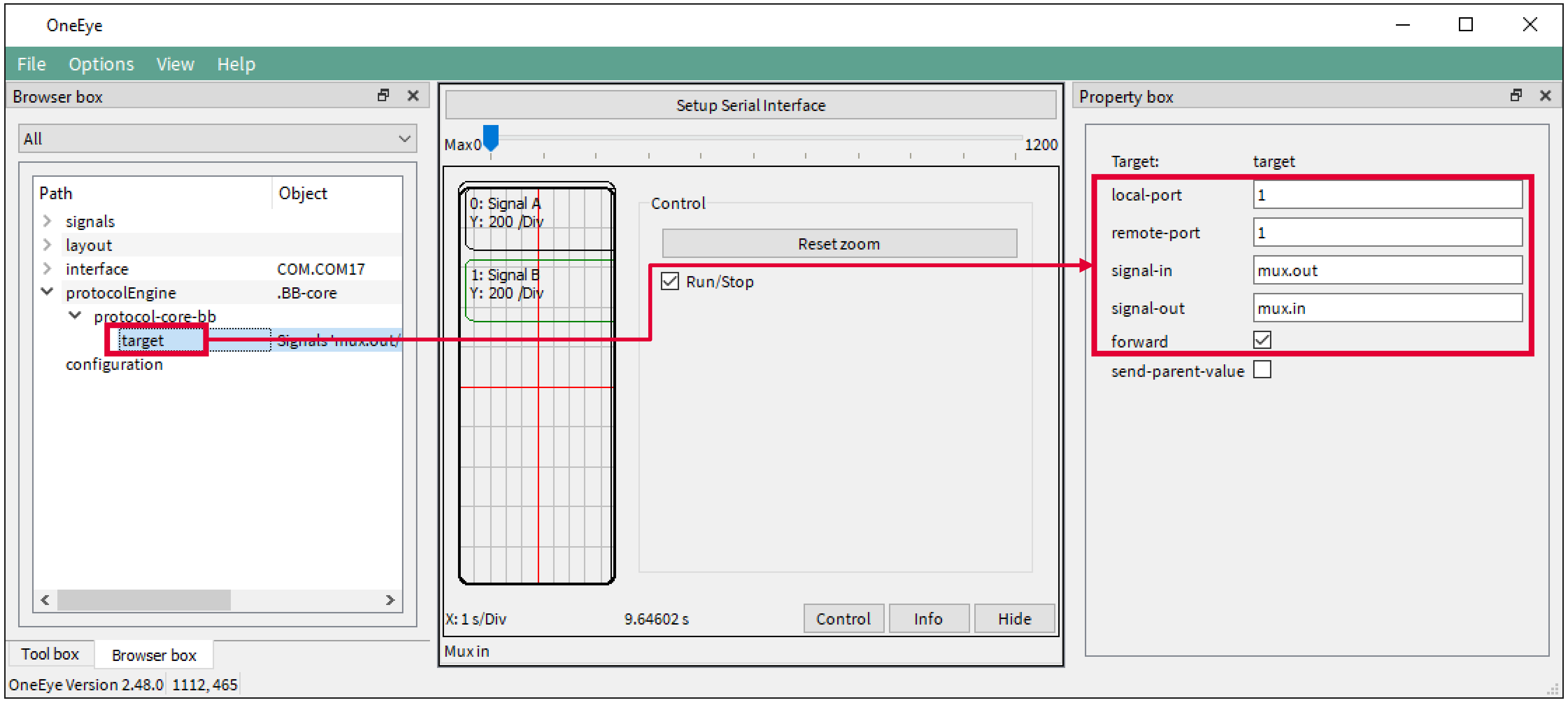
Field size (bit): 32

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**.



Path	Object
signals	
layout	
interface	COM.COM17
protocolEngine	.BB-core
protocol-core-bb	
target	Signals 'mux.out/
configuration	

Target:	target
local-port	1
remote-port	1
signal-in	mux.out
signal-out	mux.in
forward	✓
send-parent-value	☐

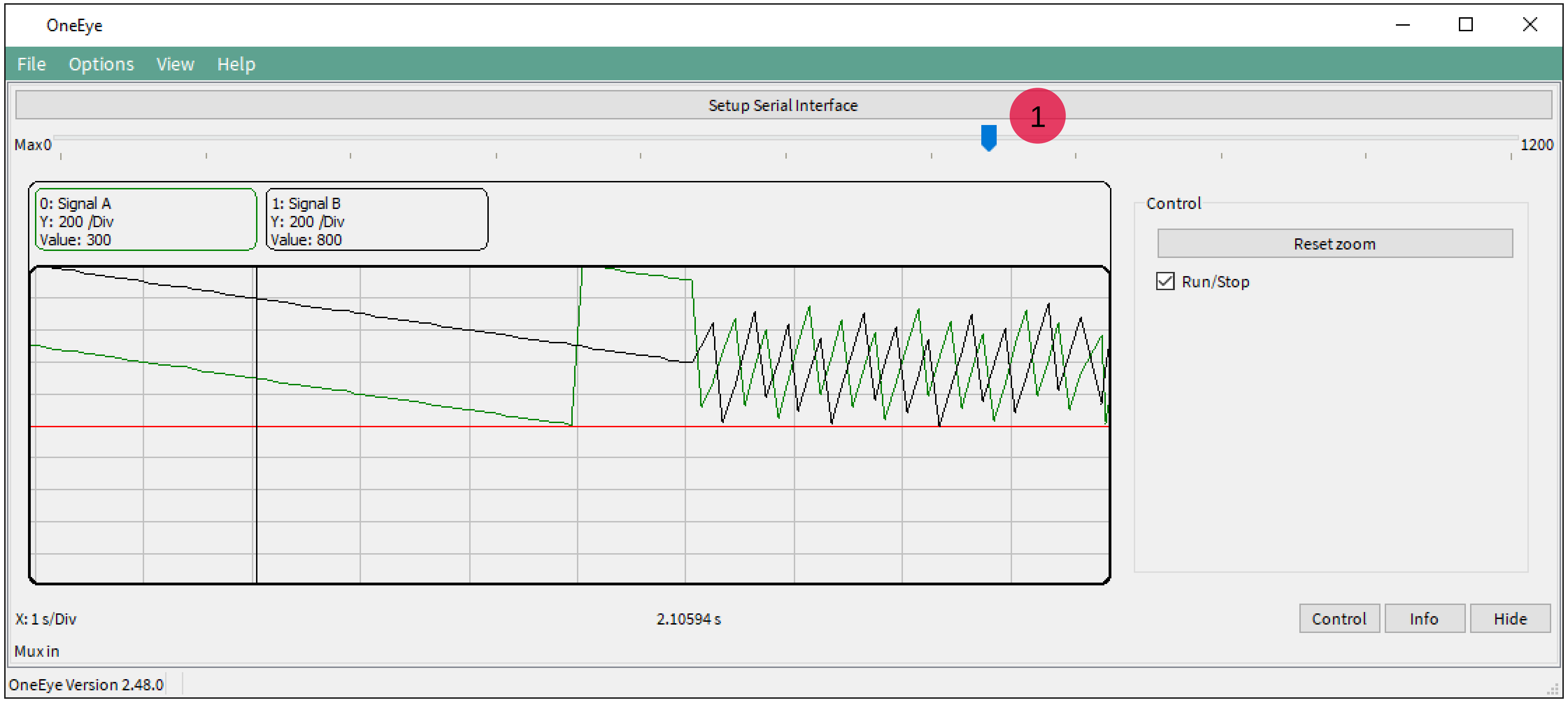
Implementation - OneEye

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.



0: Signal A
Y: 200 /Div
Value: 300

1: Signal B
Y: 200 /Div
Value: 800

X: 1 s/Div

2.10594 s

Mux in

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

- 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

Edition 2022-06
Published by
Infineon Technologies AG
81726 Munich, Germany

Do you have a question about this document?
Email: erratum@infineon.com

Document reference
OneEye_UART_Mux_1
_KIT_TC375_LK