MainCpu Unit Test Documentation

Generated by Doxygen 1.9.3

Chapter 1

Module Index

1.1 Modules

Here is a list of all modules:

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

Module Documentation

4.1 canDriverInterface class Unit Test module

This module describes the tests on the canDriverInterface class.

The test makes use of the followng test classes:

- driverClass: is the implementation of the class under test;
- usingDriver: this is a class using the driverClass for test purposes;

4.1.1 TEST DESCRIPTION

- test_canDriverInterface::initTestCase();
- test_canDriverInterface::test_canDriverInterface_1();
- test_canDriverInterface::test_canDriverInterface_2();
- test_canDriverInterface::test_canDriverInterface_3();
- test_canDriverInterface::test_canDriverInterface_4();
- test_canDriverInterface::test_canDriverInterface_5();
- test_canDriverInterface::test_canDriverInterface_6();
- test_canDriverInterface::test_canDriverInterface_7();
- test_canDriverInterface::test_canDriverInterface_8();
- test_canDriverInterface::cleanupTestCase();

4.2 canOpenInterface class Unit Test module

This module describes the tests on the canOpenInterface class.

4.2.1 TEST DESCRIPTION

- test_canOpenInterface::initTestCase();
- test_canOpenInterface::cleanupTestCase();
- test_canOpenInterface::test_odRegister_1();
- test_canOpenInterface::test_odRegister_2();

4.3 commandProtocol class Unit Test module

This module describes the tests on the commandProtocol class.

4.3.1 TEST DESCRIPTION

4.3.1.1 INVALID-FRAME type identification test

Function: test_commandProtocol::test_commandProtocol_1()
Description: the test verifies the ability to detect a non valid frame.

Execution:

1. Creates an invalid frame with an invalid frame lenght field;
    - Verify that the detected frame is invalid;
2. Creates an invalid frame missing the frame initiator character;
    - Verify that the detected frame is invalid;
3. Creates an invalid frame with a wrong '%' character usage;
    - Verify that the detected frame is invalid;

4.3.1.2 COMMAND-FRAME type identification test

Function: test_commandProtocol::test_commandProtocol_2().
Description: The test verifies the ability to detect a valid COMMAND-FRAME.

Execution:

1. Creates a valid Command Frame with a list of three parameters;
    - Verify that the detected frame is a COMMAND frame;
    - Verify the "Command" field is correct;
    - Verify the "Parameters" fields are correct

4.3.1.3 COMMAND-FRAME type and parameter identification test

Function: test_commandProtocol::test_commandProtocol_3().
Description: The test verifies the ability to detect a valid COMMAND-FRAME and its parameters format.

Execution:

1. Creates a valid Command Frame with two parametrs: a "string" parameter and a normal prameter;
    - Verify that the detected frame is a COMMAND frame;
    - Verify the "Command" field is correct;
    - Verify the ID field is correct;
    - Verify the number of parameter is correct;
    - Verify the "Parameters" fields are correct;

4.3.1.4 OK-FRAME type and parameter identification test

Function: test_commandProtocol::test_commandProtocol_4().
Description: The test veifies the ability to detect a valid OK-FRAME and its parameters.

Execution:

1. Creates a valid OK Command Frame type ;
    - Verify that the detected frame is a OK frame;
    - Verify the ID field is correct;
    - Verify the CODE field is correct;
    - Verify the number of parameter is correct;
    - Verify the "Command" fields are correct;
    - Verify the "Parameters" fields are correct;

4.3.1.5 DELAYED-FRAME type and parameter identification test

Function: test_commandProtocol::test_commandProtocol_5().
Description: The test veifies the ability to detect a valid DELAYED-FRAME and its parameters.

Execution:

1. Creates a valid DELAYED Frame type ;
    - Verify that the detected frame is a DELAYED frame;
    - Verify the ID field is correct;
    - Verify the CODE field is correct;
    - Verify the number of parameters is correct;
    - Verify the "Command" fields are correct;
    - Verify the "Parameters" fields are correct;

4.3.1.6 NOK-FRAME type and parameter identification test

Function: test_commandProtocol::test_commandProtocol_6().
Description: The test verifies the ability to detect a valid NOK-FRAME and its parameters.

Execution:

1. Creates a valid NOK-FRAME Frame type ;
    - Verify that the detected frame is a NOK-FRAME frame;
    - Verify the ID field is correct;
    - Verify the CODE field is correct;
    - Verify the number of parameters is correct;
    - Verify the "Command" fields are correct;
    - Verify the "Parameters" fields are correct;

4.3.1.7 ACK-FRAME type and parameter identification test

Function: test_commandProtocol::test_commandProtocol_7().
Description: The test verifies the ability to detect a valid ACK-FRAME and its parameters.

Execution:

1. Creates a valid ACK-FRAME Frame type ;
    - Verify that the detected frame is a ACK-FRAME frame;
    - Verify the ID field is correct;
    - Verify the CODE field is correct;
    - Verify the number of parameters is correct;
    - Verify the "Command" fields are correct;
    - Verify the "Parameters" fields are correct;

4.3.1.8 STATUS-FRAME type and parameter identification test

Function: test_commandProtocol::test_commandProtocol_8().
Description: The test verifies the ability to detect a valid STATUS-FRAME and its parameters.

Execution:

1. Creates a valid STATUS-FRAME Frame type ;
    - Verify that the detected frame is a STATUS-FRAME frame;
    - Verify the ID field is correct;
    - Verify the CODE field is correct;
    - Verify the number of parameters is correct;
    - Verify the "Command" fields are correct;
    - Verify the "Parameters" fields are correct;

4.3.1.9 COMMAND-TYPE frame format creation test

Function: test_commandProtocol::test_commandProtocol_9().
Description: The test verifies the ability to format a frame of COMMAND-TYPE.

Execution:

1. Creates a Command frame with a list of parameters;
2. Decode the frame with the commandProtocol class constructor;
    - Verify that the detected frame is a COMMAND-FRAME ;
    - Verify the ID field is correct;
    - Verify the number of parameters is correct;
    - Verify the "Command" fields are correct;
    - Verify the "Parameters" fields are correct;

4.3.1.10 DELAY-COMPLETED frame format creation test

Function: test_commandProtocol::test_commandProtocol_10().
Description: The test verifies the ability to format a frame of DELAY-COMPLETED.

Execution:

1. Creates a DELAY-COMPLETED frame with a list of parameters;
2. Decode the frame with the commandProtocol class constructor;
    - Verify that the detected frame is a DELAY-COMPLETED ;
    - Verify the ID field is correct;
    - Verify the CODE field is correct;
    - Verify the number of parameters is correct;
    - Verify the "Command" fields are correct;
    - Verify the "Parameters" fields are correct;

4.3.1.11 NA frame format creation test

Function: test_commandProtocol::test_commandProtocol_11().
Description: The test verifies the ability to format a frame of NA.

Execution:

1. Creates a NA frame ;
2. Decode the frame with the commandProtocol class constructor;
    - Verify that the detected frame is a NA ;

4.3.1.12 NOK frame format creation test

Function: test_commandProtocol::test_commandProtocol_12().
Description: The test verifies the ability to format a frame of NOK.

Execution:

1. Creates a Command frame with a list of parameters;
2. Decode the frame with the commandProtocol class constructor;
3. Creates a NOK frame with a code from the decoded protocol;
4. Decode the NOK frame with the commandProtocol class constructor;
    - Verify that the detected frame is a NOK ;
    - Verify the CODE field is correct;

4.3.1.13 OK frame format creation test

Function: test_commandProtocol::test_commandProtocol_13().
Description: The test verifies the ability to format a frame of OK.

Execution:

1. Creates a Command frame with a list of parameters;
2. Decode the frame with the commandProtocol class constructor;
3. Creates a OK answer frame with a code from the decoded protocol;
4. Decode the OK frame with the commandProtocol class constructor;
    - Verify that the detected frame is a OK ;
    - Verify the ID field is correct;
    - Verify the CODE field is correct;
    - Verify the number of parameters is correct;
    - Verify the "Command" fields are correct;
    - Verify the "Parameters" fields are correct;

4.3.1.14 DELAYED frame format creation test

Function: test_commandProtocol::test_commandProtocol_14().
Description: The test verifies the ability to format a frame of DELAYED.

Execution:

1. Creates a Command frame with a list of parameters;
2. Decode the frame with the commandProtocol class constructor;
3. Creates a DELAYED (<OK code>) answer frame with a code from the decoded protocol;
4. Decode the DELAYED frame with the commandProtocol class constructor;
    - Verify that the detected frame is a DELAYED ;
    - Verify the ID field is correct;
    - Verify the CODE field is correct;
    - Verify the number of parameters is correct;
    - Verify the "Command" fields are correct;
    - Verify the "Parameters" fields are correct;

4.3.1.15 STATUS frame format creation test

Function: test_commandProtocol::test_commandProtocol_15().
Description: The test verifies the ability to format a frame of STATUS.

Execution:

1. Creates a STATUS frame with a list of parameters;
2. Decode the STATUS frame with the commandProtocol class constructor;
    - Verify that the detected frame is a STATUS ;
    - Verify the ID field is correct;
    - Verify the CODE field is correct;
    - Verify the number of parameters is correct;
    - Verify the "Command" fields are correct;
    - Verify the "Parameters" fields are correct;

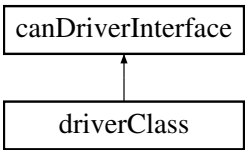
Chapter 5

Class Documentation

5.1 driverClass Class Reference

This class implements the DUT class canDriverInterface.

Inheritance diagram for driverClass:



Public Member Functions

Can Driver Api implementation section

- bool **hardwareInit** ()
  Implements the Hardware initialization operations.
- bool **hardwareShutdown** ()
  Implements the Hardware Shutdown Operations.
- bool **openPort** (uint addr, uint msk, bool md)
  Implements the reception filter aperture.
- bool **closePorts** (void)
  Closes all the reception filters.
- void **run** (void)
  Run Bus arbitration.
- void **pause** (void)
  Pause Bus arbitration.
- canDataFrame driverTxRxData (canDataFrame frame, int tmo)
  driverTxRxData This function implements a device simulation. The simulated device gets the frame then answer with a test purpose protocol in order to test the driver functionnalities.

Public Attributes

- bool **HARDWARE_INI**

  The Hardware has been initialized.
- uint **FILTER_ID**

  Curent filter active.
- uint **FILTER_MASK**

  Current filter mask.
- uint **TIMEOUT**

  Timeout waiting data from the bus.
- bool **EXTENDED_MODE**

  Extended/Standard mode.
- bool **RUNNING**

  Arbitration running.

5.1.1 Detailed Description

This class implements the DUT class canDriverInterface.

This is the driver class implementation test. The class inherit the canDriverInterface that is the class under test.

The class implements the hardware interface methods that a can driver should implement:

- hardwareInit(): shall be implemented in order to initi the specific hardware inetrface;
- hardwareShutdown(): shall be implemented in order to terminates the hardware activities;
- openPort(): this shall open a given reception filter;
- closePorts(): closes all the reception filters;
- run(): activates the bus arbitration;
- pause(): stops the bus arbitration;
- driverTxRxData(): implements the simulated tx/rx data exchange

The TxRx simulated received frame (rxframe) implementation are:

- txframe.canId == 1 -> rxframe.data[0] = hardware initialization status (1 or 0);
- txframe.canId == 2 -> rxframe.data[0][1] = FILTER_ID (LSB first);
- txframe.canId == 3 -> rxframe.data[0][1] = FILTER_MASK (LSB first);
- txframe.canId == 4 -> rxframe.data[0] = Extended mode status;
- txframe.canId == 5 -> rxframe.data[0] = Running mode status;
- txframe.canId == 6 -> rxframe.data[0][1] = TIMEOUT (LSB first);
- txframe.canId == 7 -> Simulate no answer: rxframe.canId = 0;

5.1.2 Member Function Documentation

5.1.2.1 driverTxRxData()

```
canDataFrame driverClass::driverTxRxData (
            canDataFrame frame,
            int tmo )  [inline]
```

driverTxRxData This function implements a device simulation. The simulated device gets the frame then answer with a test purpose protocol in order to test the driver functionnalities.

Parameters

frame	This is the frame sent by the userClass.
tmo	This is the Timeout that the driver sets for the data reception.

Returns

The txframe is filled with a simulated answer.

- txframe.canId == 1 -> rxframe.data[0] = hardware initialization status (1 or 0);
- txframe.canId == 2 -> rxframe.data[0][1] = FILTER_ID (LSB first);
- txframe.canId == 3 -> rxframe.data[0][1] = FILTER_MASK (LSB first);
- txframe.canId == 4 -> rxframe.data[0] = Extended mode status;
- txframe.canId == 5 -> rxframe.data[0] = Running mode status;
- txframe.canId == 6 -> rxframe.data[0][1] = TIMEOUT (LSB first);
- txframe.canId == 7 -> Simulate no answer: rxframe.canId = 0;

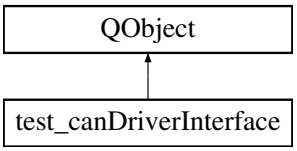
The documentation for this class was generated from the following file:

- C:/Users/marco/Documents/GitHub/GantryZ190/GANTRY/UNIT-TEST/MAIN-CPU/SOURCE/CanDriver↩ Interface/tst_testcandriverclass.cpp

5.2 test_canDriverInterface Class Reference

Executing class.

Inheritance diagram for test_canDriverInterface:



Public Attributes

- usingDriver * **deviceDriver**
- canDriverInterface::canDataFrame **txframe**

Private Slots

- void initTestCase ()
  Driver creation and reception filter aperture.
- void cleanupTestCase ()
  Driver deleting.
- void test_canDriverInterface_1 ()
  The driver is set to Run Mode.
- void test_canDriverInterface_2 ()
  Read the Filter setting.
- void test_canDriverInterface_3 ()
  Read the Filter Mask.
- void test_canDriverInterface_4 ()
  Read the Extended/Standard mode setting.
- void test_canDriverInterface_5 ()
  Read the Run/Pause mode status.
- void test_canDriverInterface_6 ()
  Read the Timeout value.
- void test_canDriverInterface_7 ()
  Run the CAN BUS test the no-answer scenario.
- void test_canDriverInterface_8 ()
  Driver Stop mode.

5.2.1 Detailed Description

Executing class.

See also

canDriverInterface class Unit Test module

5.2.2 Member Function Documentation

5.2.2.1 cleanupTestCase

```
void test_canDriverInterface::cleanupTestCase ( )  [private], [slot]
```

Driver deleting.

The user class is deleted and the driver should Shutdown. The worker and the thread of the driver shall be deleted successfully.

5.2.2.2 initTestCase

```
void test_canDriverInterface::initTestCase ( )  [private], [slot]
```

Driver creation and reception filter aperture.

This test creates the driver and open a reception filter. The test shall verify that the Test should not communicate beacuse it should not be in run mode now.

5.2.2.3 test_canDriverInterface_1

```
void test_canDriverInterface::test_canDriverInterface_1 ( )  [private], [slot]
```

The driver is set to Run Mode.

The test verify the runDriver() method: After the runDriver() command is used the test reads the Hardware Init flag from the implemented Driver.

5.2.2.4 test_canDriverInterface_2

```
void test_canDriverInterface::test_canDriverInterface_2 ( )  [private], [slot]
```

Read the Filter setting.

The Test read the current filter setting from the implemented Driver: the current expected filter setting should be the one provided in the TestInit with the openPort command.

5.2.2.5 test_canDriverInterface_3

```
void test_canDriverInterface::test_canDriverInterface_3 ( )  [private], [slot]
```

Read the Filter Mask.

The Test reads the current filter Mask from the implemented Driver: the current expected filter setting should be the one provided in the TestInit with the openPort command.

5.2.2.6 test_canDriverInterface_4

```
void test_canDriverInterface::test_canDriverInterface_4 ( )  [private], [slot]
```

Read the Extended/Standard mode setting.

The Test reads the current Extended/Standard mode from the implemented Driver: the current expected Extended/Standard mode setting should be the one provided in the TestInit with the openPort command.

5.2.2.7 test_canDriverInterface_5

```
void test_canDriverInterface::test_canDriverInterface_5 ( )  [private], [slot]
```

Read the Run/Pause mode status.

The Test reads the current Run/Pause mode from the implemented Driver: the expected value shall be Run Mode.

5.2.2.8 test_canDriverInterface_6

```
void test_canDriverInterface::test_canDriverInterface_6 ( )  [private], [slot]
```

Read the Timeout value.

The Test reads the current Timeout value from the implemented Driver: the expected value is the one passed with the txData() function in the test.

5.2.2.9 test_canDriverInterface_7

```
void test_canDriverInterface::test_canDriverInterface_7 ( )  [private], [slot]
```

Run the CAN BUS test the no-answer scenario.

No Answer test .

The test verifies the case on witch there is no answer from the remote dvice. In this case the driver shall returns a frame with canId == 0.

5.2.2.10 test_canDriverInterface_8

```
void test_canDriverInterface::test_canDriverInterface_8 ( )  [private], [slot]
```

Driver Stop mode.

The test verifies the stopDriver() command. This function shall keep the driver alive but in a non communicating status.

The test verify that the driver doesn't communicate after the command is used and the hardware inti status flag is still true;

The test then calls the runDriver() method to restore the communication.

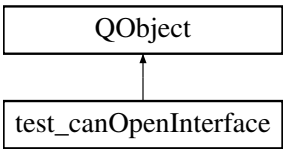
The documentation for this class was generated from the following file:

- C:/Users/marco/Documents/GitHub/GantryZ190/GANTRY/UNIT-TEST/MAIN-CPU/SOURCE/CanDriver↩ Interface/tst_testcandriverclass.cpp

5.3 test_canOpenInterface Class Reference

Executing class.

Inheritance diagram for test_canOpenInterface:



Private Slots

- void **initTestCase** ()
- void **cleanupTestCase** ()
- void test_odRegister_1 ()

  odRegister class, Object Register data access
- void test_odRegister_2 ()

  odRegister class, canDataFrame conversion features

5.3.1 Detailed Description

Executing class.

See also

canOpenInterface class Unit Test module

5.3.2 Member Function Documentation

5.3.2.1 test_odRegister_1

```
void test_canOpenInterface::test_odRegister_1 ( )  [private], [slot]
```

odRegister class, Object Register data access

This is a multi test sequence, testing the data access feature:

- Data type formatting (8 to 32 bit dtaa types);
- class Data content access;

5.3.2.2 test_odRegister_2

```
void test_canOpenInterface::test_odRegister_2 ( )  [private], [slot]
```

odRegister class, canDataFrame conversion features

This is a multi test sequence, testing the class to properly convert its content in a correct canDataFrame format.

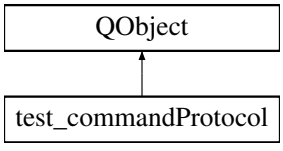
The documentation for this class was generated from the following file:

- C:/Users/marco/Documents/GitHub/GantryZ190/GANTRY/UNIT-TEST/MAIN-CPU/SOURCE/CanOpen↩ Interface/tst_testcanopeninterface.cpp

5.4 test_commandProtocol Class Reference

Executing class.

Inheritance diagram for test_commandProtocol:



Private Slots

- void **initTestCase** ()
- void **cleanupTestCase** ()
- void test_commandProtocol_1 ()

  INVALID-FRAME type identification test.
- void test_commandProtocol_2 ()

  COMMAND-FRAME type identification test.
- void test_commandProtocol_3 ()

  COMMAND-FRAME type and parameter identification test.
- void test_commandProtocol_4 ()

  OK-FRAME type and parameter identification test.
- void test_commandProtocol_5 ()

  DELAYED-FRAME type and parameter identification test.
- void test_commandProtocol_6 ()

  NOK-FRAME type and parameter identification test.
- void test_commandProtocol_7 ()

  ACK-FRAME type and parameter identification test.
- void test_commandProtocol_8 ()

  STATUS-FRAME type and parameter identification test.
- void test_commandProtocol_9 ()

  COMMAND-TYPE frame format creation.
- void test_commandProtocol_10 ()

  DELAY-COMPLETED frame format creation.
- void test_commandProtocol_11 ()

  "NA" frame format creation.
- void test_commandProtocol_12 ()

  "NOK" frame format creation.
- void test_commandProtocol_13 ()

  "OK" frame format creation.
- void test_commandProtocol_14 ()

  "DELAYED" frame format creation.
- void test_commandProtocol_15 ()

  "STATUS" frame format creation.

5.4.1 Detailed Description

Executing class.

See also

commandProtocol class Unit Test module

5.4.2 Member Function Documentation

5.4.2.1 test_commandProtocol_1

```
void test_commandProtocol::test_commandProtocol_1 ( )  [private], [slot]
```

INVALID-FRAME type identification test.

The test verifies the ability to detect a non valid frame.

5.4.2.2 test_commandProtocol_10

```
void test_commandProtocol::test_commandProtocol_10 ( )  [private], [slot]
```

DELAY-COMPLETED frame format creation.

The test veifies the ability to format a frame of DELAY-COMPLETED type.

5.4.2.3 test_commandProtocol_11

```
void test_commandProtocol::test_commandProtocol_11 ( )  [private], [slot]
```

"NA" frame format creation.

The test veifies the ability to format a frame of "NA" type.

5.4.2.4 test_commandProtocol_12

```
void test_commandProtocol::test_commandProtocol_12 ( )  [private], [slot]
```

"NOK" frame format creation.

The test veifies the ability to format a frame of "NOK" type.

5.4.2.5 test_commandProtocol_13

```
void test_commandProtocol::test_commandProtocol_13 ( )  [private], [slot]
```

"OK" frame format creation.

The test veifies the ability to format a frame of "OK" type.

5.4.2.6 test_commandProtocol_14

```
void test_commandProtocol::test_commandProtocol_14 ( )  [private], [slot]
```

"DELAYED" frame format creation.

The test veifies the ability to format a frame of "DELAYED" type.

5.4.2.7 test_commandProtocol_15

```
void test_commandProtocol::test_commandProtocol_15 ( )  [private], [slot]
```

"STATUS" frame format creation.

The test veifies the ability to format a frame of "STATUS" type.

5.4.2.8 test_commandProtocol_2

```
void test_commandProtocol::test_commandProtocol_2 ( )  [private], [slot]
```

COMMAND-FRAME type identification test.

The test verifies the ability to detect a valid COMMAND-FRAME

5.4.2.9 test_commandProtocol_3

```
void test_commandProtocol::test_commandProtocol_3 ( )  [private], [slot]
```

COMMAND-FRAME type and parameter identification test.

The test veifies the ability to detect a valid COMMAND-FRAME and its parameters.

5.4.2.10 test_commandProtocol_4

```
void test_commandProtocol::test_commandProtocol_4 ( )  [private], [slot]
```

OK-FRAME type and parameter identification test.

The test veifies the ability to detect a valid OK-FRAME and its parameters.

5.4.2.11 test_commandProtocol_5

```
void test_commandProtocol::test_commandProtocol_5 ( )  [private], [slot]
```

DELAYED-FRAME type and parameter identification test.

The test veifies the ability to detect a valid DELAYED-FRAME and its parameters.

5.4.2.12 test_commandProtocol_6

```
void test_commandProtocol::test_commandProtocol_6 ( )  [private], [slot]
```

NOK-FRAME type and parameter identification test.

The test veifies the ability to detect a valid NOK-FRAME and its parameters.

5.4.2.13 test_commandProtocol_7

```
void test_commandProtocol::test_commandProtocol_7 ( )  [private], [slot]
```

ACK-FRAME type and parameter identification test.

The test veifies the ability to detect a valid ACK-FRAME and its parameters.

5.4.2.14 test_commandProtocol_8

```
void test_commandProtocol::test_commandProtocol_8 ( )  [private], [slot]
```

STATUS-FRAME type and parameter identification test.

The test veifies the ability to detect a valid STATUS-FRAME and its parameters.

5.4.2.15 test_commandProtocol_9

```
void test_commandProtocol::test_commandProtocol_9 ( )  [private], [slot]
```

COMMAND-TYPE frame format creation.

The test verifies the ability to format a frame of COMMAND-TYPE.

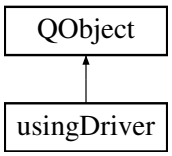
The documentation for this class was generated from the following file:

- C:/Users/marco/Documents/GitHub/GantryZ190/GANTRY/UNIT-TEST/MAIN-CPU/SOURCE/Command↩ Protocol/tst_commandprotocol.cpp

5.5 usingDriver Class Reference

The usingDriver class.

Inheritance diagram for usingDriver:



Public Slots

- void deviceRxSlot (canDriverInterface::canDataFrame frame, int odIndex)

  deviceRxSlot

Signals

- void txToDriver (canDriverInterface::canDataFrame frame, int odIndex, uint tmo, QObject ∗device)

  txToDriver This signal functon is required in order to interact with the driver. The signal is connected with the Out of thread driver slot, handling the TxRx data activity.
- void rxDone (void)

  rxDone

Public Member Functions

- **usingDriver** (void)

  usingDriver Constructor of the class: the constructor connects the class signals and slots necessary to interact with the handled driver.
- void txData (canDriverInterface::canDataFrame frame, int odIndex, uint tmo)

  txData

Public Attributes

- driverClass **driver**

  instance of the driverClass to be tested
- canDriverInterface::canDataFrame **rxframe**

  handles the received frame;
- int **rxIndex**

  handles the received register index

5.5.1 Detailed Description

The usingDriver class.

This class implements a test class using the driver under test.

The class implements the necessary interface methods:

- signal txToDriver(): this slot is connected with the Driver reception slot;
- public txData(): method to be used to send data: internally it triggers the txToDriver() signal;
- public slot deviceRxSlot(): required to receive the reception data from the driver

In order to let the test function to wait for Queued data exchange the class implements an extra signal:

- signal rxDone(): this signal is emitted as soon as the deviceRxSlot() is called.

In the class datata the received frame and odIndex is stored into the rxframe and rxIndex variables.

5.5.2 Member Function Documentation

5.5.2.1 deviceRxSlot

```
void usingDriver::deviceRxSlot (
            canDriverInterface::canDataFrame frame,
            int odIndex ) [inline], [slot]
```

deviceRxSlot

This is the **mandatory** function that every driver user class shall implement in order to receive the driver can data frame.

Parameters

frame	This is the can received data frame.
odIndex	This is the Object Register index the driver userClass linked to the data exchange.

The test implementation of this function slot stores the received data into public variables that can easily be inspected by the test function.

Finally the function triggers the rxDone() signal to trigger the test that the data has been received from the driver.

5.5.2.2 rxDone

```
void usingDriver::rxDone (
            void  )  [signal]
```

rxDone

This is a test purpose signal used to trigger the test function when data are received from the Out of Thread driver. The test function shall use the QSignalSpy() looking ath this signal generation

5.5.2.3 txData()

```
void usingDriver::txData (
            canDriverInterface::canDataFrame frame,
            int odIndex,
            uint tmo )  [inline]
```

txData

This public methid is used to let the test function sending data frames to the can driver.

Parameters

frame	This is the frame to be sent.
odIndex	This is the register index that will be received back from the driver.

The function emits the txToDriver() with the userClass target set to this, because the user Class in this case is **this** class.

5.5.2.4 txToDriver

```
void usingDriver::txToDriver (
            canDriverInterface::canDataFrame frame,
            int odIndex,
            uint tmo,
            QObject * device )  [signal]
```

txToDriver This signal funcion is required in order to interact with the driver. The signal is connected with the Out of thread driver slot, handling the TxRx data activity.

Parameters

frame	This is the can frame sent to the driver
odIndex	This is the register index target of the transaction.
device	This is the pointer to the userClass that finally will receive the can received data frame.

Attention

the userClass is the class that shall implement the slot deviceRxSlot()!

The documentation for this class was generated from the following file:

- C:/Users/marco/Documents/GitHub/GantryZ190/GANTRY/UNIT-TEST/MAIN-CPU/SOURCE/CanDriver↩ Interface/tst_testcandriverclass.cpp

Index