ETAS

LABCAR-PINCONTROL V2.2

User's Guide

Copyright

V2.2 R02 DE - 06.2016

Contents

1 Introduction 5
1.1 About This Manual 5
1.1.1 User Profile 5
1.1.2 Using This Manual 6
2 Working with LABCAR-PINCONTROL V2.2 7
2.1 Configuring LABCAR-PINCONTROL V2.2 7
2.1.1 Configuring the Network Interface Card of the Host System 7
2.1.2 Ethernet Configuration 9
2.1.3 ES4440.1 System Configuration 10
2.2 The Wire Harness File 12
2.2.1 Creating a Wire Harness File 12
2.3 Operating LABCAR-PINCONTROL V2.2 17
2.4 The Main Menu 28
3 API Documentation 31
3.1 Introduction 31
3.1.1 Tasks of the COM Controller 31
3.1.2 Using the CAN-API 32
3.1.3 Data Content of the COM- and CAN-API 32
3.2 Configurations and Sequence of the CAN Messages 32
3.2.1 Single Errors in Stand-Alone Applications 33
3.2.2 Multiple Errors in Stand-Alone Applications 34
3.2.3 Single Errors in Master/Slave Applications 35
3.2.4 Multiple Errors in Master/Slave Applications 38
3.3 Initialization 39
3.3.1 Initialization of the COM Controller 39
3.3.2 Initialization for CAN 39
3.4 Detailed Description of the Commands 40

3.4.1 General Command Structure . . . 41
3.4.2 Definitions for all Functions . . . 42
3.4.3 Error Codes . . . 43
3.4.4 The IDN Command . . . 45
3.4.5 Open_Load . . . 46
3.4.6 Open_Load_realtime . . . 47
3.4.7 ShortCut_xUBATTy_20A . . . 48
3.4.8 ShortCut_xUBATTy_20A_realtime . . . 49
3.4.9 Pin2PinFirstChWithoutLoad . . . 50
3.4.10 Pin2PinSecondChannelWithoutLoad . . . 51
3.4.11 Pin2PinFirstChRealtimeWithLoad . . . 52
3.4.12 Pin2PinSecondChRealtimeWithLoad . . . 53
3.4.13 RInline_realtime . . . 54
3.4.14 Pullup_Pulldown_xUBATTy_20A_realtime . . . 56
3.4.15 Open_Load_400V . . . 57
3.4.16 ShortCut_xUBATTy_400V . . . 58
3.4.17 ShortCut_xUBATTy_400V_Ex . . . 59
3.4.18 Pin_2_Pin_400V . . . 60
3.4.19 Pin_2_Pin_400V_Ex . . . 61
3.4.20 Reset_all_errors . . . 62
3.4.21 Activate_relay . . . 63
3.4.22 Activate_realtime_switch . . . 65
3.4.23 Test Fuses . . . 67
3.4.24 CurrentMeasurement . . . 69

4 ETAS Contact Addresses . . . 71

Index . . . 73

1 Introduction

LABCAR-PINCONTROL V2.2 is supplied together with the ES4440.1/.2 Compact Failure Simulation Module. The ES4440.1/.2 Compact Failure Simulation Module is used for real-time error simulation with ECUs.

LABCAR-PINCONTROL V2.2 contains a user interface for the manual control and configuration of an ES4440.1/.2 as well as a COM controller for automatic operation.

The following tasks can be executed with LABCAR-PINCONTROL V2.2:

- Manual testing of the diagnostic functionality of ECUs:
  - Creating and managing failure sets (a failure set is a group of ECU signals, e.g. all lambda signals)
  - Simple selection of a single signal for error simulation
  - Setting the error duration and the parameters for the simulation of loose contacts
  - Activation of the error with a simple click of the mouse
- Configuration of ES4440.1/.2 Compact Failure Simulation Modules:
  - Specifying IP and CAN addresses of the modules used
  - Configuration as a stand-alone, master or slave system
  - Self test and fuse test of the ES4440.1/.2
- COM-API for use in automatic test operation

1.1 About This Manual

This manual consists of the following chapters:

- "Introduction" on page 5

  This chapter
- "Working with LABCAR-PINCONTROL V2.2" on page 7

  This chapter describes preparatory configuration tasks and how to operate LABCAR-PINCONTROL V2.2.
- "API Documentation" on page 31

  This chapter contains information on the automatic operation of the ES4440.1/.2 Compact Failure Simulation Module with the COM controller of LABCAR-PINCONTROL V2.2 or via CAN.

1.1.1 User Profile

This manual is intended for specialists who develop and test automotive ECUs. Specialist knowledge of measuring and ECU technology is assumed.

1.1.2 Using This Manual

Representation of Information

All activities to be executed by the user are presented in what is referred to as a "Use-Case" format. I.e. the aim is defined in brief as a title and the relevant steps necessary to achieve this aim are then listed. The information is displayed as follows:

Target definition

Any introductory information...

- Step 1

  Possibly an explanation of step 1...
- Step 2

  Possibly an explanation of step 2...
- Step 3

  Possibly an explanation of step 3...

Any concluding remarks...

Specific example:

To create a new file

If you want to create a new file, no other file may be open.

- Select **File** → **New**.

  The "Create File" dialog box appears.
- Enter a name for the file in the "File name" box.

  The file name must not be more than 8 characters long.
- Click **OK**.

The new file is created and saved under the name specified. You can now work with the file.

Typographic Conventions

The following typographic conventions are used:

Select **File** → **Open**.	Menu functions are shown in boldface/blue.
Click **OK**.	Buttons are shown in boldface/blue.
Press <ENTER>.	Keyboard commands are shown in angled brackets in block capitals.
The "Open File" dialog box appears.	Names of program windows, dialog boxes, fields etc. are shown in quotation marks.
Select the file `setup.exe`.	Text in drop-down lists, program code, as well as path and file names are shown in the `Courier` font.
A conversion between the file types logical and arithmetic is *not* possible.	Content markings and newly introduced terms are shown in *italics*

2 Working with LABCAR-PINCONTROL V2.2

This chapter describes preparatory configuration tasks and how to operate LABCAR-PINCONTROL V2.2.

- "Configuring LABCAR-PINCONTROL V2.2" on page 7

  This section contains a description of the preparatory tasks.
- "The Wire Harness File" on page 12

  This section describes how to create a wire harness file.
- "Operating LABCAR-PINCONTROL V2.2" on page 17

  This section contains a description of all actions you can execute in the user interface to simulate errors.
- "The Main Menu" on page 28

  This section contains a brief description of all the essentials of the main menu of LABCAR-PINCONTROL V2.2.

2.1 Configuring LABCAR-PINCONTROL V2.2

Before you can start working with PINCONTROL, there are certain settings which have to be made as regards Ethernet connections and concerning the hardware used.

2.1.1 Configuring the Network Interface Card of the Host System

The following settings have to be made for the network interface card used to control the ESES4440.1/.2 Compact Failure Simulation Module.

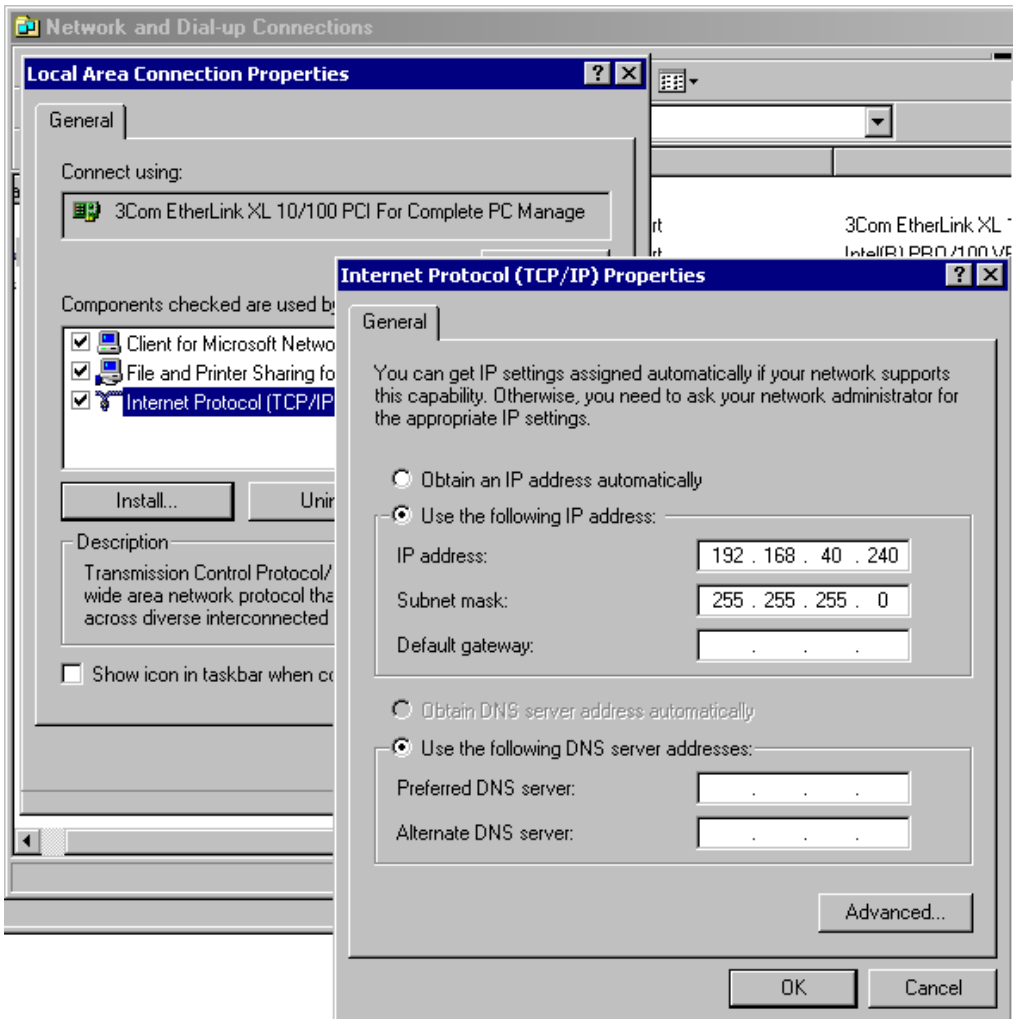
To configure TCP/IP

- Select **Settings** → **Control Panel** from the Windows Start menu.
- In the Control Panel window, double-click **Network and Dial-up Connections**.
- Select the connection/device you require.
- Right-click the entry and select **Properties**.

  The "Local Area Connection Properties" window opens.
- Select the component "Internet Protocol (TCP/IP)".

- Click **Properties**.

  The "Internet Protocol (TCP/IP) Properties" window opens.



- Select the settings shown in the figure:
  IP address: 192.168.40.240
  Subnet mask: 255.255.255.0

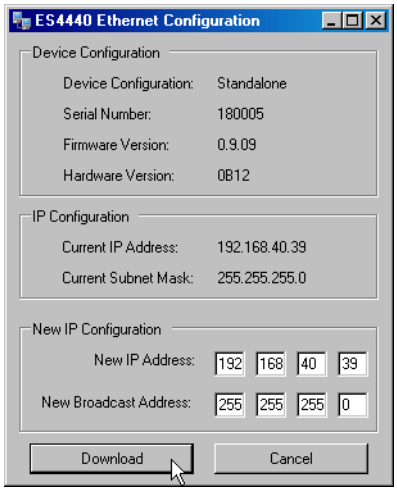
  You can also, however, use any other valid IP address.
- Close all windows with **OK**.

2.1.2 Ethernet Configuration

This section describes how to assign Ethernet addresses to the ES4440.1/.2 Compact Failure Simulation Modules used.

To define Ethernet settings

- In the main menu of LABCAR-PINCONTROL V2.2 select **Tools** → **ES4440 Ethernet Configuration**.
- A warning is issued that only one device (i.e. the ES4440.1/.2 to be configured) can be switched on during configuration.
- Click **OK**.
- The "ES4440 Ethernet Configuration" dialog box opens.



This window contains configuration information, particularly on the IP address and subnet mask used.

- New data can be entered in the "New IP Configuration" field.
- Click **Download** to download the modified data for the hardware.

or

- Click **Cancel** to quit the dialog box without making any changes.

2.1.3 ES4440.1 System Configuration

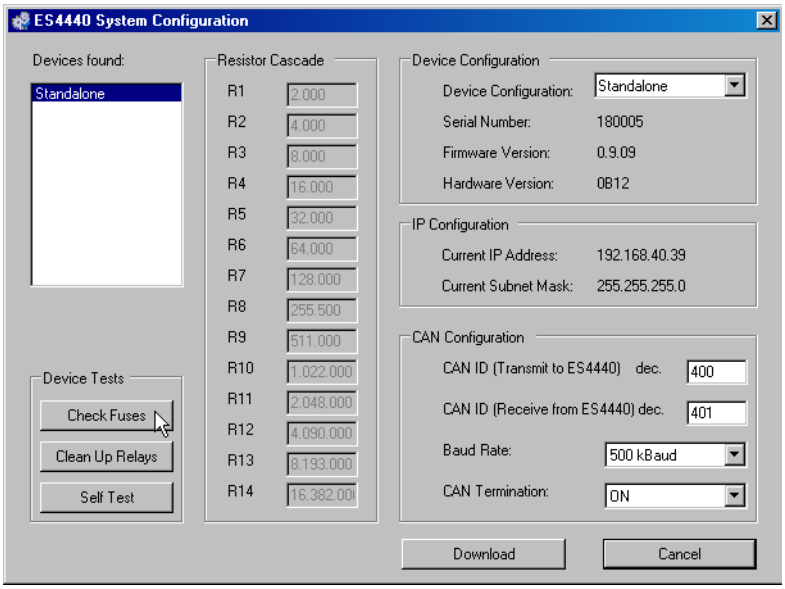
The system configuration consists of assigning the status of the individual hardware (stand-alone, master, slave) and of configuring the CAN interface (only when CAN is to be used for control purposes).

Note

The system configuration described in this section is only possible if there is an Ethernet connection to the relevant hardware!

- In the main menu of LABCAR-PINCONTROL V2.2 select **Tools** → **ES4440 System Configuration**.

  The "ES4440 System Configuration" dialog box opens.



  Here, you can make a number of settings which are described below.

To configure the device

- In the "Devices found" field, select those ES4440.1/.2s for which the settings are to apply.
- In the "Device Configuration" box, you can assign the relevant board its status (Standalone, Master, Slave1, ...,Slave14).
- If the configuration is complete once you have executed this step, click **Download**.

  The configuration is executed and the dialog box closed.
- Otherwise, continue making settings and then terminate the configuration as described above.

To view the IP configuration

The "IP Configuration" field contains the IP address and subnet mask of the hardware currently selected.

To configure the CAN interface

- If you want to control your hardware via a CAN interface, you must make the necessary settings in the "CAN Configuration" field.
  - CAN ID (Transmit to ES4440) dec.

    This is where you specify the ID of the relevant device which must be contained in a send message for this device.
  - CAN ID (Receive from ES4440) dec.

    This is where you specify the ID of the relevant device which must be contained in a receive message from this device.
  - Baud Rate:

    This is where you specify the transfer rate – you can choose between "500 kBaud" and "1 Mbaud".
  - CAN Termination:

    This is where you can specify whether the relevant device has CAN termination or not.

To view information on the resistor cascade

The actual resistor values of the internal resistor cascade are shown in the "Resistor Cascade" field.

To clean up relays and execute tests

- For details of the actions you find in the "Device Tests" field, please refer to "To clean up the relay contacts" on page 25, "To test fuses" on page 26 and "To run a self test" on page 27.

2.2 The Wire Harness File

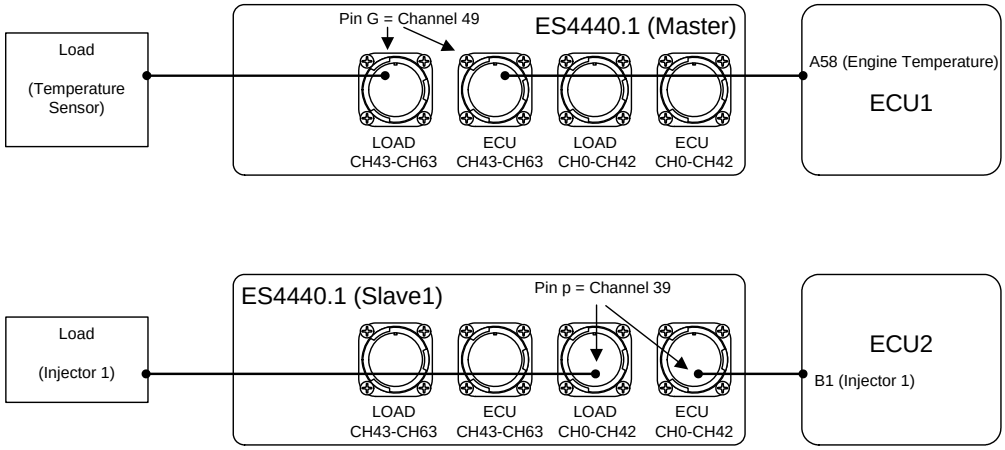
The wire harness file is an important part of a project: it contains a description of which ECU signals are connected to which channels of an ES4440.1/.2 Compact Failure Simulation Module.

This means that a set containing the following data must be available for an ECU channel:

- ECU Name
- Pin Number
- ES4440 Name
- Channel Number

The name of the pin (Pin Name) is an explanatory extension to the pin number and does not have to be unique.

The following figure contains an example:



In the figure, two ECUs are connected to two ES4440.1/.2s:

- ECU Name: ECU1/ECU2
- Pin Number: A58/B1
- Pin Name: Engine Temperature/Injector 1
- ES4440 Name: Master/Slave1
- ES4440 Connector: ECU/LOAD CH43-CH63 / ECU/LOAD CH0-CH42
- ES4440 Pin: G = Channel 49 / p = Channel 39

2.2.1 Creating a Wire Harness File

This section contains an example of how you can create a wire harness file for your LABCAR-PINCONTROL V2.2 project from wire harness data stored in an Excel sheet.

- In Windows XP:

  `C:\Documents and Settings\All Users\ Application Data\ETAS\LABCAR-PINCONTROL\2.1\ Excel\LABCAR-PINCONTROLV2.0_Example.xls`.

- In Windows Vista/Windows 7:

  `C:\ProgramData\ETAS\LABCAR-PINCONTROL\2.1\Excel\LABCAR-PINCONTROLV2.0_Example.xls`

The Excel file consists of 3 tables:

- **WireHarnessData**

  The mapping data between your ECU ports and the ES4440 ports is defined in this table.

- **ES4440WireHarnessSignals**

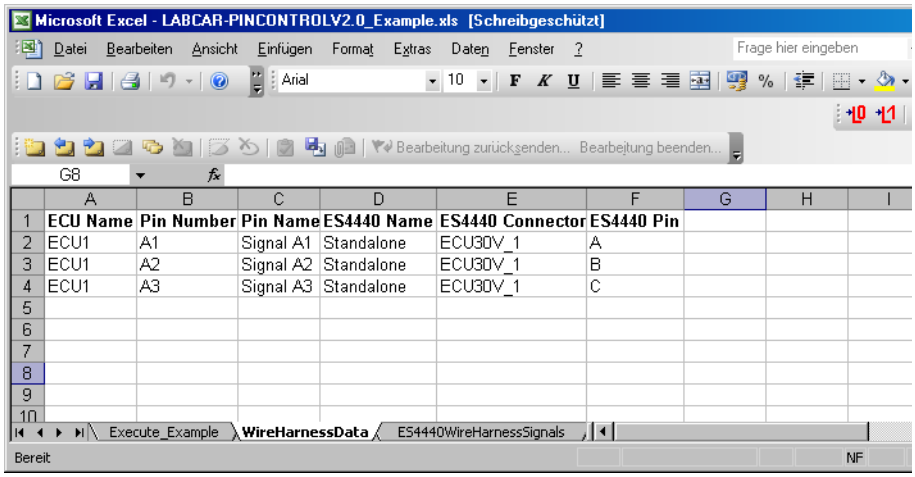
  Specific settings are made in this table so the data in the "WireHarness-Data" table can be evaluated.

- **Execute_Example**

  This table makes it possible to execute the macro and to apply a simple "Open Load" error to the ES4440 (only works with this example)

The "WireHarnessData" Table

The assignment between the ECU ports and the ES4440 ports must be made in this table.



	A	B	C	D	E	F	G	H	I
1	**ECU Name**	**Pin Number**	**Pin Name**	**ES4440 Name**	**ES4440 Connector**	**ES4440 Pin**			
2	ECU1	A1	Signal A1	Standalone	ECU30V_1	A			
3	ECU1	A2	Signal A2	Standalone	ECU30V_1	B			
4	ECU1	A3	Signal A3	Standalone	ECU30V_1	C			
5									
6									
7									
8									
9									
10									

The individual columns contain the following information:

- **ECU Name**

  The name of your ECU (has to be unique).

- **Pin Number**

  The ECU pin (has to be unique, but does not have to be a number).

- **Pin Name**

  Possible further definition of the pin, e.g. name of the signal pending at this pin (does not have to be unique).

- **ES4440 Name**

  The name of the ES4440 with which the pin is connected. (Possible names can be found in the "ES4440WireHarnessSignals" table under "ES4440 Device Names").

- **ES4440 Connector**

  This is where you define via which connector the signal gets to the ES4440.
  Possible names can be found in the "ES4440WireHarnessSignals" table under "Connector/Pin/Channel/Electric Type").

- **ES4440 Pin**

  This is where you define with which ES4440 pin of the connector defined above your ECU signal is connected (possible names can be found in the "ES4440WireHarnessSignals" table under "Connector/Pin/Channel/ Electric Type").

Every row of this table now results in a mapping:

<**ECU Name**> + <**Pin Number**> (+ <**Pin Name**>) =
<**ES4440 Name**> + <**ES4440 Connector**> + <**ES4440 Pin**>

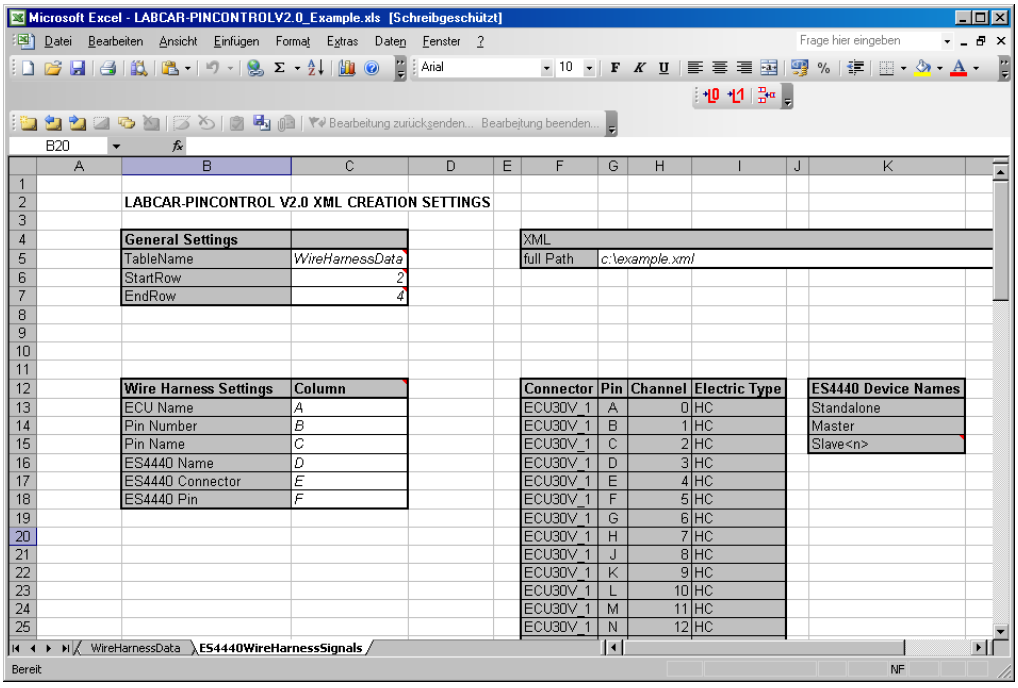
This representation clearly shows that "Pin Name" does not have to be unique – it is only used as an additional description to the "Pin Number".

Once you have connected all the ECU pins to the ES4440.1/.2s to be used, you have specified your wire harness file completely.

A few settings now have to be made so that your data can be transformed into the XML file for LABCAR-PINCONTROL V2.2.

Adaptations

Now select the "ES4440WireHarnessSignals" table – this table contains all data the macro requires to generate the required file from the previously defined wire harness data.



This table contains the following information:

General Settings:

- **TableName**

  This is where you define which table contains the data "ECU Name", "Pin Number", "Pin Name", "ES4440 Name", "ES4440 Connector", "ES4440 Pin".

- **Start Row**

  The first row of the "WireHarnessData" table to be evaluated.

- **End Row**

  The last row of the "WireHarnessData" table to be evaluated.

Wire Harness Settings:

This is where you define which information is to be found in which column.

- **ECU Name**

  The column of the "WireHarnessData" table in which the name of the ECU is defined.

- **Pin Number**

  The column of the "WireHarnessData" table in which the number of the ECU pin is defined.

- **ECU Pin Name**

  The column of the "WireHarnessData" table in which the ECU pin name is defined.

- **ES4440 Name**

  The column of the "WireHarnessData" table in which the name of the ES4440.1/.2 is defined.
  The possible names are listed in the "ES4440 Device Names" Group (see below).

- **ES4440 Connector**

  The column of the "WireHarnessData" table in which the port of the ES4440.1/.2 is defined. The possible ports are defined in the "Connector/Pin/Channel/Electric Type" Group.

- **ES4440 Pin**

  The column of the "WireHarnessData" table in which the port pin is defined.
  "ES4440 Pin" must correspond to the relevant "ES4440 Connector". The possible pins are defined in the "Connector/Pin/Channel/Electric Type" Group (see below).

XML:

- **Full Path**

  This is where you can define the name and path of the wire harness file to be generated by the macro.

ES4440 Device Names:

Definition of the names of all ES4440.1/.2 used.

Connector/Pin/Channel/Electric Type:

Contains the complete description of all ports of an ES4440.1/.2.

Note

Do not change any fields with a gray background. This can lead to a defective file, or no file at all, being generated.

Creating the Wire Harness File

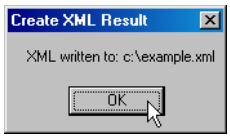
Once all settings have been made, the wire harness file can be generated.

- Activate the "Execute_Example" table.
- Click the **Create WireHarnessFile for ES4440** button.

or

- Select **Extra** → **Macro** → **Macros...**.
- Select the macro "CreateXML" and click **Run**.

  The successful execution of the macro is indicated by the following dialog box:



2.3 Operating LABCAR-PINCONTROL V2.2

This section contains a description of all the important steps for controlling an ES4440.1/.2 Compact Failure Simulation Module from the user interface of LABCAR-PINCONTROL V2.2.

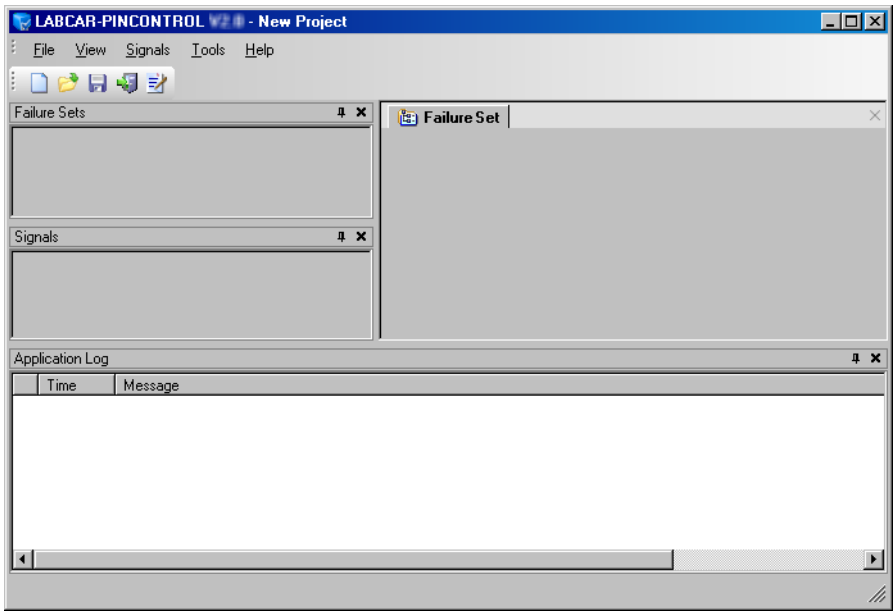
This section contains descriptions of the following user actions:

- "To launch LABCAR-PINCONTROL V2.2" on page 17
- "To import signals from the wire harness file" on page 18
- "To create a new failure set" on page 19
- "To add a signal" on page 20
- "To remove a signal" on page 21
- "To set multiple errors" on page 22
- "To simulate a loose contact" on page 23
- "To simulate a contact resistance" on page 24
- "To measure the current" on page 24
- "To clean up the relay contacts" on page 25
- "To test fuses" on page 26
- "To run a self test" on page 27
- "To specify display options" on page 27

To launch LABCAR-PINCONTROL V2.2

- Select **Programs** → **ETAS** → **LABCAR-PINCONTROL V2.2** → **PinControl.exe** from the Windows Start menu.

  LABCAR-PINCONTROL V2.2 opens.



The LABCAR-PINCONTROL V2.2 user interface consists of the following sections:

- Signals

  This window shows the signals of the project-specific wire harness file (see "To import signals from the wire harness file" on page 18).
- Failure Sets

  This window shows the signals which are grouped into failure sets by the user (with assigned errors) (see "To create a new failure set" on page 19).
- Failure Set

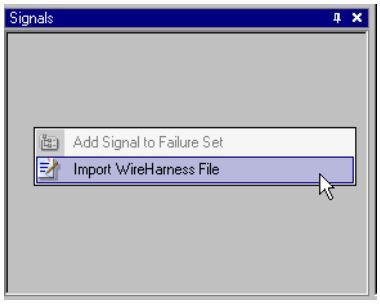
  This section shows the tabs with the various error types. This is where the failure set currently selected is defined, i.e. errors assigned to signals (see "To set multiple errors" on page 22).
- Application Log

  This is where information and messages are issued.

The first step is to import signals by importing the data from your wire harness file.

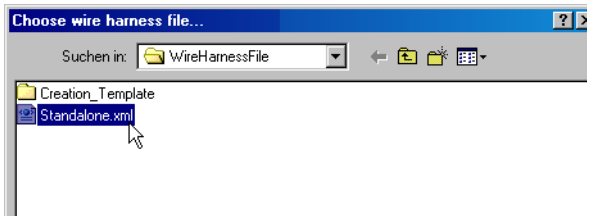
To import signals from the wire harness file

- Right-click in the "Signals" window.
- Select **Import WireHarness File** from the shortcut menu.

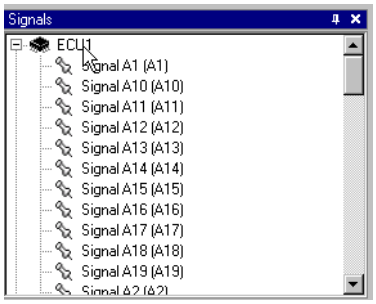


A file selector window opens.

- Select the XML file which contains your wire harness specification.



The signals are imported and displayed in the "Signals" window.

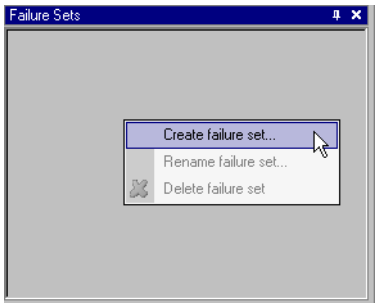


- Save the project with **Save as**.

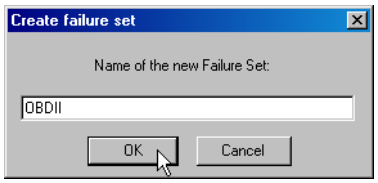
You now have all the information on how the ECU is connected to the ES4440.1/.2 Compact Failure Simulation Module. The next step is to group specific signals which are interesting for your current project to create one (or more) failure set(s).

To create a new failure set

- Right-click in the "Failure Sets" window.
- From the shortcut menu, select **Create failure set...**.



- Enter a name for the failure set to be created.

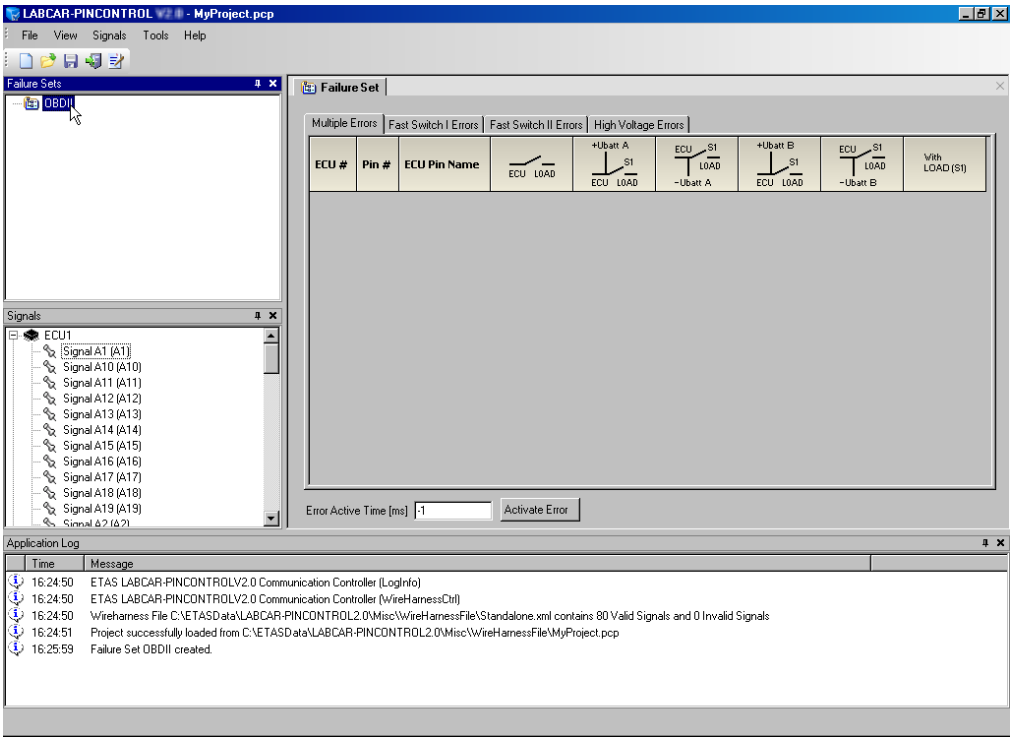


- Click **OK**.

  The "OBDII" failure set is created.

- Click the newly created failure set.

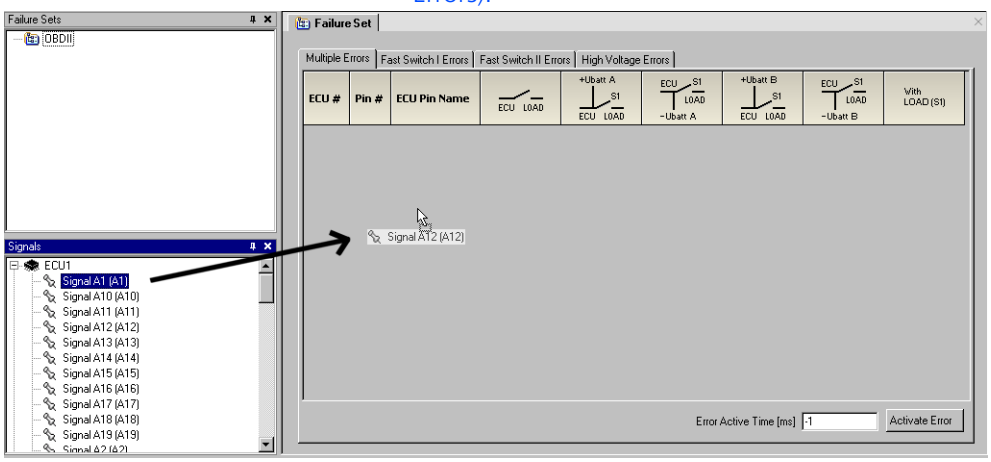
  All types of available error are now shown (in various tabs) in the right-hand section of the user interface.



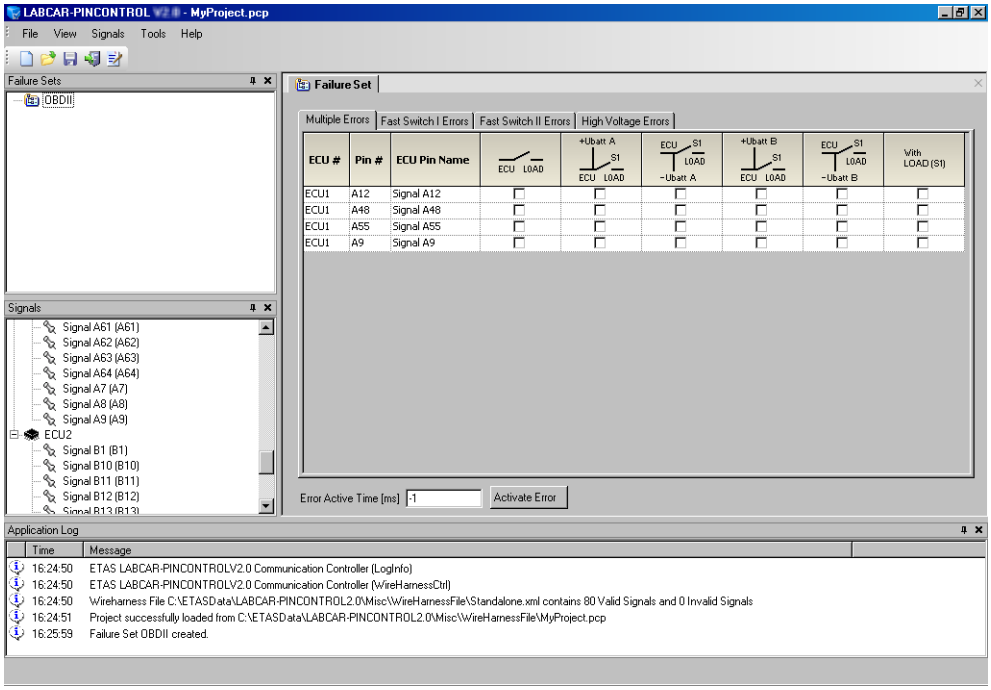
Now you will complete the "OBDII" failure set by adding signals.

To add a signal

- Select a signal in the "Signals" window.
- Drag it (by keeping the mouse button pressed down) to the tab you selected before (e.g. Multiple Errors).



- Do exactly the same with other signals.



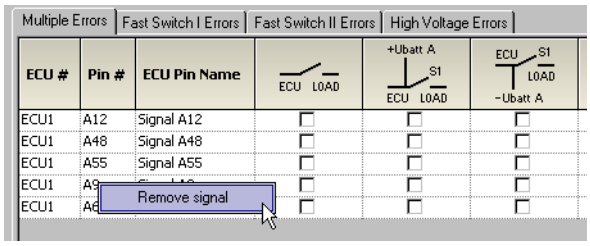
- Save the project.

  The failure set is now part of your project.

  If necessary, you can now create further failure sets as described above.

To remove a signal

- If you want to remove a signal which you have added to the failure set by mistake, click it using the right-hand mouse button.
- Select **Remove Signal**.

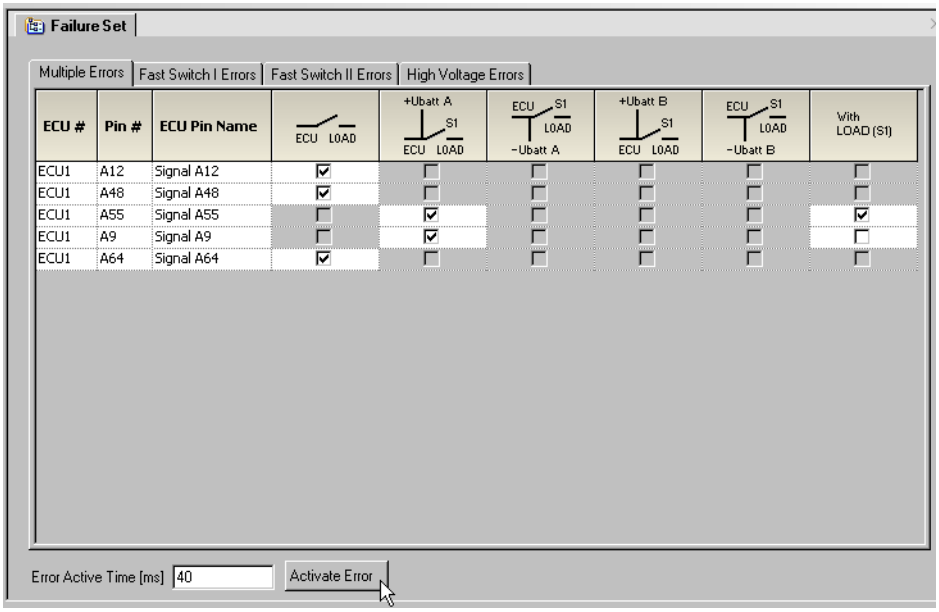


The signal is removed from your failure set.

To set multiple errors[1] for specific signals of the failure set, proceed as follows.

To set multiple errors

- To simulate a short for the signals "Signal A12", "Signal A48" and "Signal A64", select the relevant box.
- For the signal "Signal A55", set the error "Short against +UBatt_A". As the load is to remain connected with this error, activate the relevant box in the column "With Load (S1)".
- Then select a short against +UBatt_A for the signal "Signal A9".



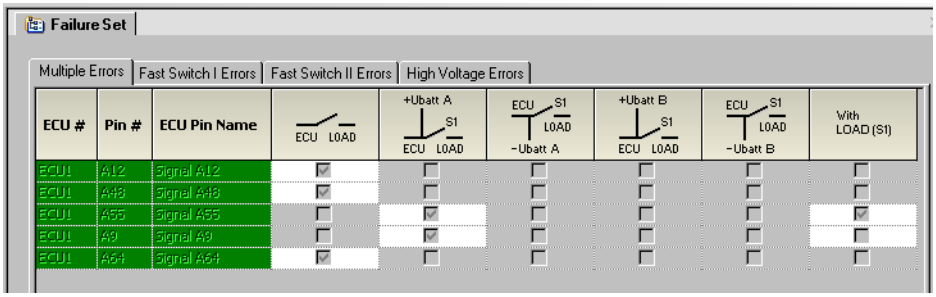
- At the bottom of the tab, enter the time for which the errors are to be active.

  "-1" means that the errors are pending until **Activate Error** is pressed again.

[1] For information on the different error types, refer to the User's Guide of the ES4440.1/.2 Compact Failure Simulation Module.

- Click **Activate Error**.

  The errors are activated – when they are run, the signal definitions are shown on a green background.



To simulate a loose contact

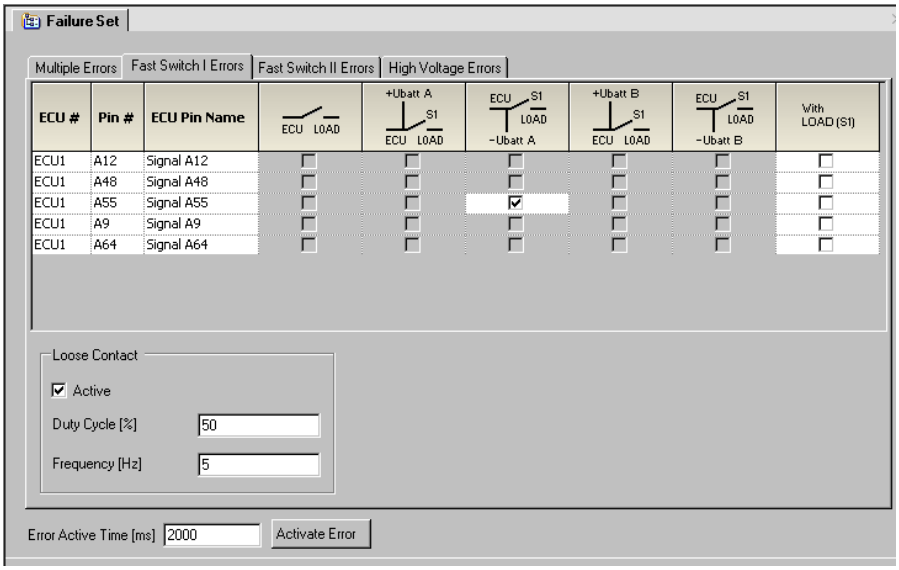
It is also possible to simulate loose contacts in the tabs "Fast Switch I Errors" and "Fast Switch II Errors", i.e. a specific error is not pending for a defined period of time but is switched with a specific frequency and duty cycle.

- Toggle, for example, to the "Fast Switch I Errors" tab.

Note

When toggling to another tab, the errors selected in the current tab are deleted!

- Select an error.
- In the "Loose Contact" field, activate the option "Activate".
- Enter the required duty cycle under "Duty Cycle".
- Enter the required frequency under "Frequency".



- Enter the required duration of loose contact simulation under "Error Active Time".
- To run the error, click **Activate Error**.

To simulate a contact resistance

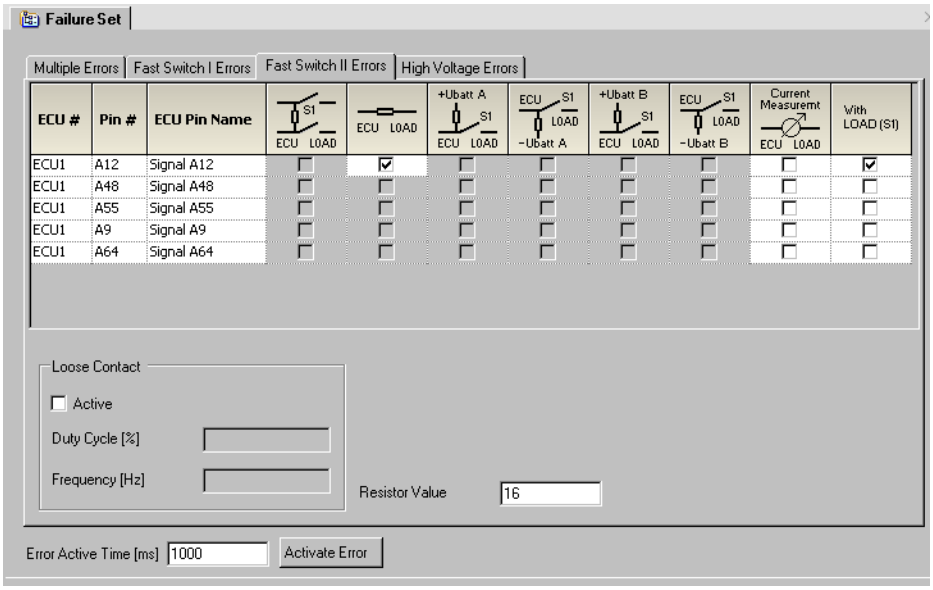
The errors with contact resistances (Inline, Pin-to-Pin, leakage currents to battery voltages) can be found in the tab "Fast Switch II Errors".

- Select an error in this tab (e.g. a line resistance).

Note

Please note that you must assign two signals to a Pin-to-Pin error!

- Enter the required resistor in the "Resistor Value" field.

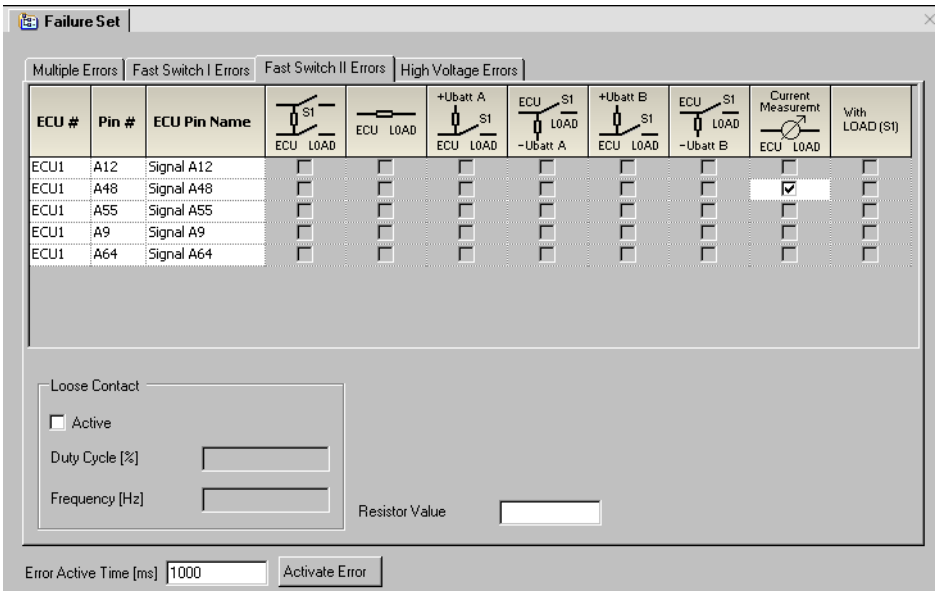


- Enter the required duration of the error simulation under "Error Active Time".
- To run the error, click **Activate Error**.

To measure the current

In the "Fast Switch II Errors" tab, it is possible to measure the current flowing in a signal line by routing this signal via the "Current" front-panel connectors.

- Select the relevant signal.
- Enter the required duration of the measurement under "Error Active Time".



- To run the error, click **Activate Error**.
- The selected signal is now routed via the "Current" ports on the front panel of the ES4440.1/.2 Compact Failure Simulation Module at which the current flowing via this channel can then be measured.

To clean up the relay contacts

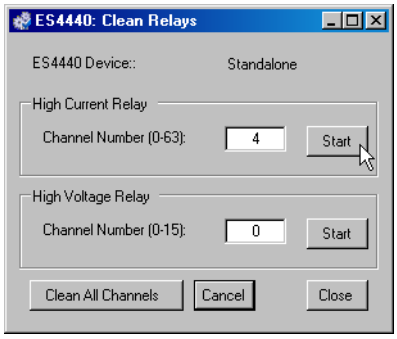
To clean oxidized relay contacts, proceed as follows:

- Select **Tools** → **ES4440 System Configuration**.

  The "ES4440 System Configuration" dialog box opens.
- In the "Devices found" field, select the hardware to which the following action should refer.
- In the "Device Tests" field, click **Clean Up Relays**.

  A warning is issued stating that you should disconnect all connectors of type "LOAD" and "ECU" before cleaning takes place.

- In the following window, you can now select relays of individual channels or clean all relays.



.

Note

As switching takes place ten times each time a relay is cleaned, cleaning all relays can take a few minutes' time!

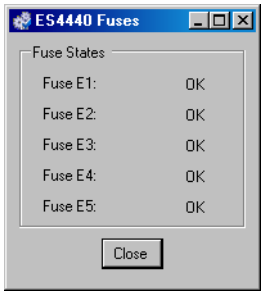
To test fuses

Please proceed as follows to test the five fuses of the error rails:

- Select **Tools** → **ES4440 System Configuration**.

  The "ES4440 System Configuration" dialog box opens.
- In the "Devices found" field, select the hardware to which the following action should refer.
- In the "Device Tests" field, click **Check Fuses**.

  The fuse test is run and the result displayed.



Note

The position of the fuses in the device and their specification are described in the ES4440.1/.2 Compact Failure Simulation Module manual.

To run a self test

PLD, the CAN controller and the EEPROM are tested for their integrity in a self test. To run a self test, proceed as follows:

- Select **Tools** → **ES4440 System Configuration**.

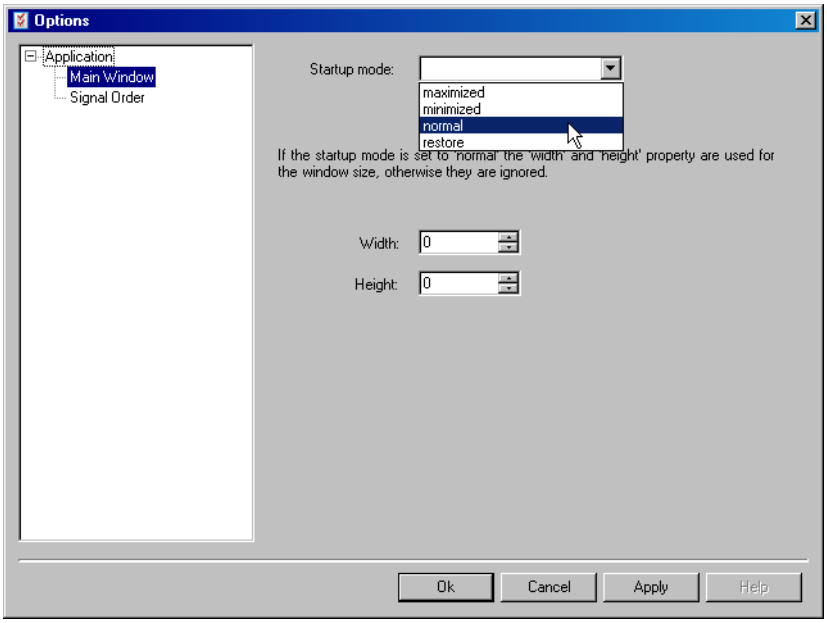
  The "ES4440 System Configuration" dialog box opens.
- In the "Devices found" field, select the hardware to which the following action should refer.
- In the "Device Tests" field, click **Self Test**.

  The self test is executed and the result shown in the log window.

To specify display options

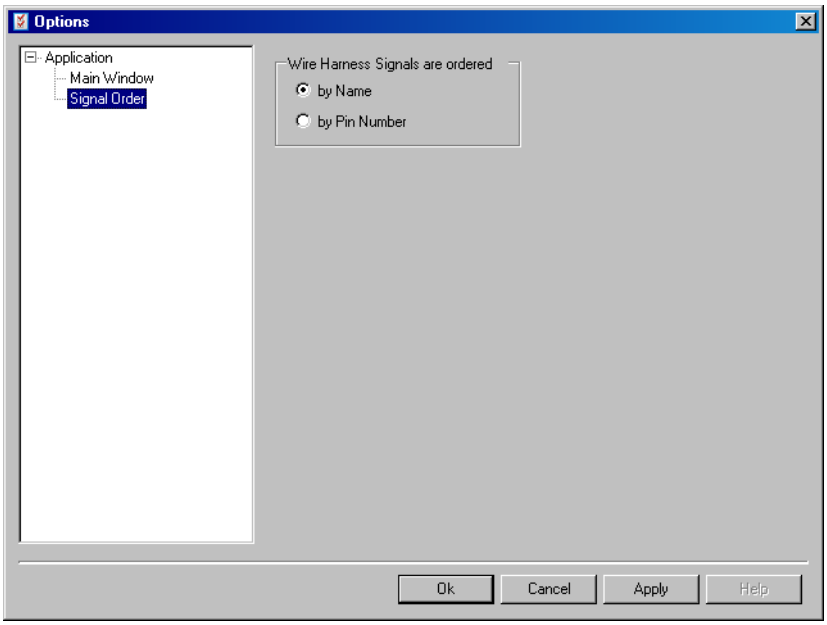
You can influence the display of the user interface of LABCAR-PINCONTROL V2.2 when the program is launched as well as the sorting of the signals in the "Signals" window.

- Select **Tools** → **Options...**.
- Select "Main window" in the left-hand section of the "Options" window.
- Select the required display of the main window under "Startup Mode".



- If you select the option "normal", you can specify the width and height of the window in "Width" and "Height".

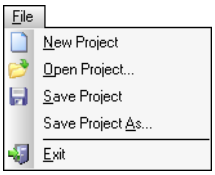
- With the option "Signal Order", you can specify whether the signals (in the "Signals" window) are ordered according to their name or "Pin Number".



2.4 The Main Menu

This section contains a description of the PINCONTROL main menu.

The "File" Menu



- **File → New Project**

  Creates a new project

- **File → Open Project...**

  Opens a project saved previously

- **File → Save Project**

  Saves the project currently loaded. In addition to configuration data, data of the failure set (Pin Name, Pin Number and ECU Name) is stored.

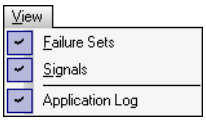
- **File → Save Project As...**

  Saves the project under a new name

- **File → Exit**

  Ends LABCAR-PINCONTROL V2.2

The "View" Menu



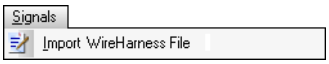
- **View → Failure Sets**

  Shows/hides the area in which the failure sets of the project are shown
- **View → Signals**

  Shows/hides the area in which the signals of the wire harness file imported into the project are shown.
- **View → Application Log**

  Shows/hides the log window

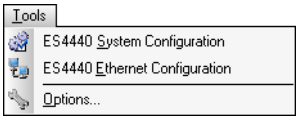
The "Signals" Menu



- **Signals → Import Wire Harness File**

  Enables the selection of a wire harness file

The "Tools" Menu



- **Tools → ES4440 System Configuration**

  Opens the dialog box in which the system settings are made
- **Tools → ES4440 Ethernet Configuration**

  Opens the dialog window for the configuration of the Ethernet interface
- **Tools → Options...**

  Opens the dialog box for defining the display options

The "Help" Menu

- **Help → About**

  Opens a window containing version information
- **Help → Support**

  Opens a window containing ETAS contact addresses.

3 API Documentation

This chapter contains information on the automatic operation of the ES4440.1/.2 Compact Failure Simulation Module with the COM controller of LABCAR-PINCONTROL V2.2 or via CAN.

This chapter contains the following information:

- "Introduction" on page 31
- "Configurations and Sequence of the CAN Messages" on page 32
- "Initialization" on page 39
- "Detailed Description of the Commands" on page 40

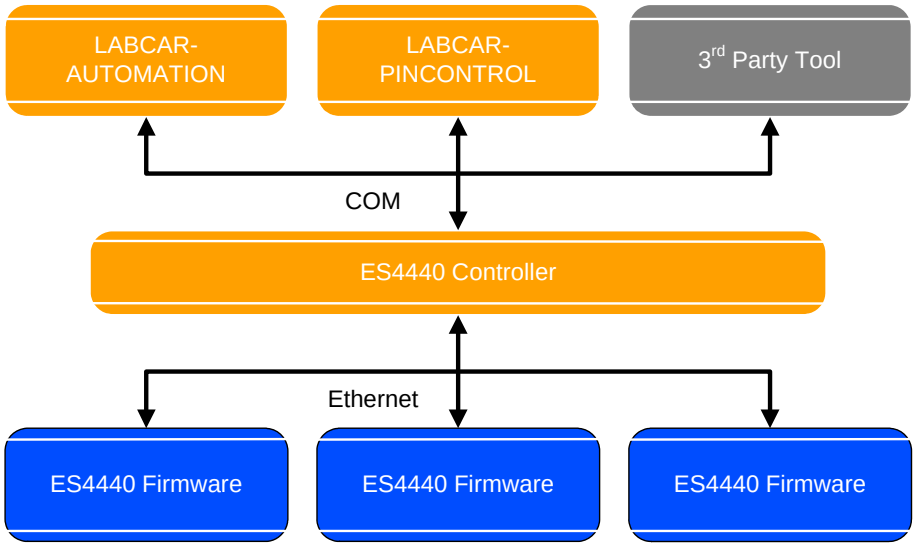
3.1 Introduction

The ES4440.1/.2 Compact Failure Simulation Module can be controlled both via Ethernet and via CAN. The Ethernet protocol is supported by the COM controller provided. The CAN protocol is to be used for the direct connection of a real-time simulation target to one or more ES4440.1/.2s using a CAN board.

This API description contains the method calls of the COM controller and the data structure of the CAN messages.

3.1.1 Tasks of the COM Controller

The following figure shows how the COM controller works.



Input values of the COM controller are the ECU name and ECU port. Using the wire harness file, the COM controller assigns an ES4440.1/.2 and the number of the relevant pin to these values.

In addition, the COM controller is responsible for the transfer of the COM methods to the Ethernet protocol. In a master/slave configuration, the COM controller is also responsible for the distribution of the commands to the correct ES4440 modules.

3.1.2 Using the CAN-API

The CAN messages are sent directly to the relevant ES4440.1/.2. As it is not possible for the ES4440.1/.2 modules to communicate with each other, the user must ensure

- the mapping of ECU name and pins to the ES4440 modules and
- the distribution of the commands to the ES4440 modules in master/slave operation.

3.1.3 Data Content of the COM- and CAN-API

The data content of the COM-API and the CAN-API is identical apart from one exception. The COM controller expects ECU name and pin name as input data, the CAN-API pin numbers of the ES4440 modules.

3.2 Configurations and Sequence of the CAN Messages

This section describes the distribution of the CAN messages to the ES4440 modules used. All errors are listed below and the order in which the CAN messages have to be sent to the various ES4440 modules is also shown.

The following rules generally apply:

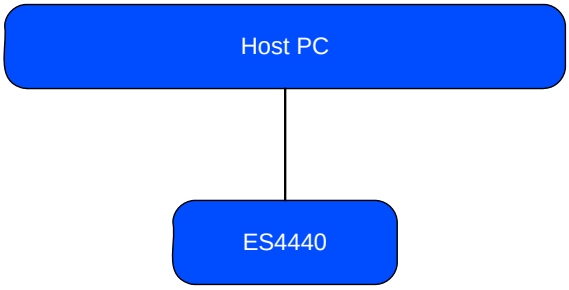
- The relay errors are always activated by the master module. This ensures that the error is switched simultaneously by all ES4440 modules.
- For errors realized with MOSFETs, activation takes place on the module on which the error was configured.
- As soon as an error has been configured on an ES4440 module, it also requires a reset command so that the error can be recovered.
- Pin-to-Pin errors are a kind of exception – these are dealt with separately.

Note

The scenarios shown of master/slave configuration are examples – if a command is shown as being executed on "Slave 1", it could equally well be the master or another slave!

3.2.1 Single Errors in Stand-Alone Applications

The program runs described in this section are valid for single errors which are executed on an ES4440.1/.2 Compact Failure Simulation Module.



Program Run For Errors Generated with a Relay

1. **Error configuration**

   High-current errors:

   – `Open_Load`

   or

   – `ShortCut_xUBATTy_20A`

   or

   – `Pin2PinFirstChWithoutLoad` / `Pin2PinSecondChWithoutLoad`

   High-voltage errors:

   – `Open_Load_400V`

   or

   – `ShortCut_xUBATTy_400V`

   or

   – `Pin_2_Pin_400V`

2. **Error activation**

   – `Activate_relay`

3. **Resetting the error**

   – `Reset_all_errors`

Program Run for Errors Realized with MOSFETs

1. **Error configuration**

   – `Open_Load_realtime`

   or

   – `ShortCut_xUBATTy_20A_realtime`

   or

   – `Pin2PinFirstChRelatimeWithLoad`

   or

   – `Pin2PinSecondChRealtimeWithLoad`

   or

- `RInline_realtime`

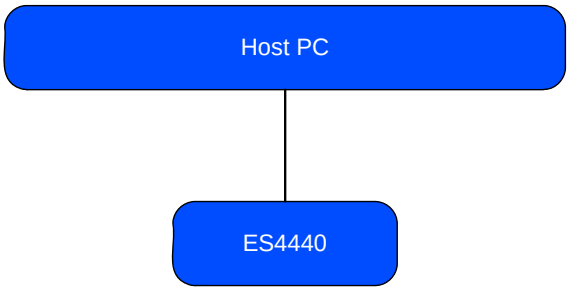
  or

- `Pullup_Pulldown_xUBATTy_20A_realtime`

2. **Error activation**
   - `Activate_realtime_switch`
3. **Resetting the error**
   - `Reset_all_errors`

3.2.2 Multiple Errors in Stand-Alone Applications

The program runs described in this section are valid for multiple errors which are executed on an ES4440.1/.2 Compact Failure Simulation Module.



Program Run for Errors Generated with a Relay

1. **Error configuration**

   (max. 10 commands are sent in sequence to the ES4440)
   - `Open_Load`

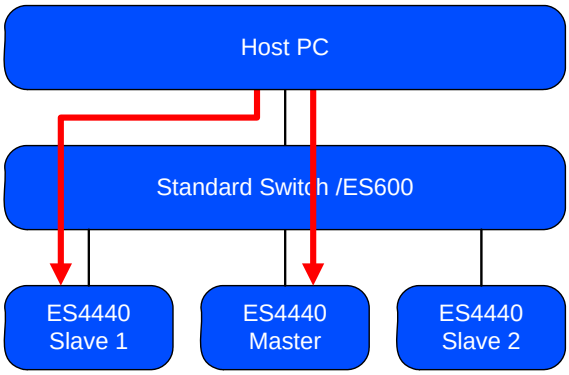
     and/or

   - `ShortCut_xUBATTy_20A`
2. **Error activation**
   - `Activate_relay`
3. **Resetting the errors**
   - `Reset_all_errors`

3.2.3 Single Errors in Master/Slave Applications

The program runs described in this section are valid for single errors which are executed on a master/slave system.

Program Run for Errors Generated with a Relay



1. **Error configuration (on Slave 1)**

   Max. 10 high-current errors:

   – `Open_Load`

   and/or

   – `ShortCut_xUBATTy_20A`

2. **Error activation (on the master)**

   – `Activate_relay`

3. **Resetting the errors (on Slave 1) ***

   – `Reset_all_errors`

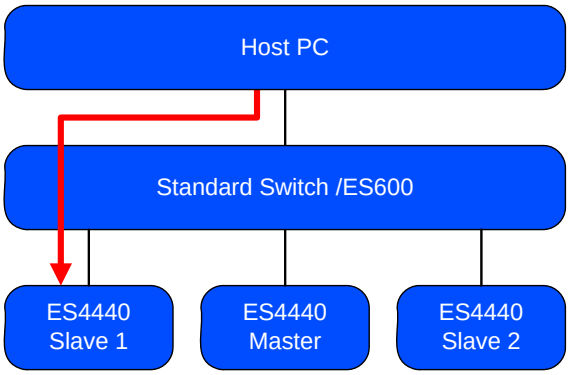
4. **Resetting the errors (on the master) ****

   – `Reset_all_errors`

* The command "Reset_all_errors" on a slave is simply saved initially.

** The command "Reset_all_errors" on the master results in synchronous resetting of the errors on the master and all slaves which have previously saved this command.

Program Run for Errors Generated with MOSFETs



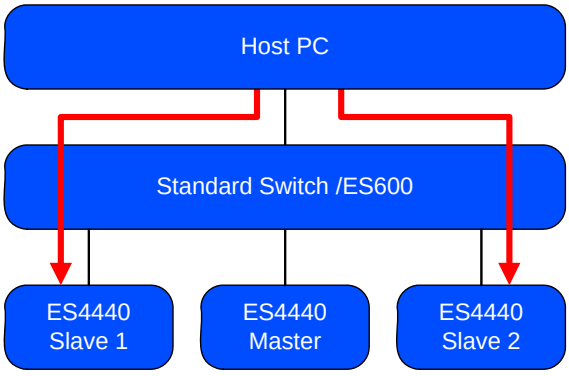
1. **Error configuration**
   - `Open_Load_realtime`

     or
   - `ShortCut_xUBATTy_20A_realtime`

     or
   - `RInline_realtime`

     or
   - `Pullup_Pulldown_xUBATTy_20A_realtime`
2. **Error activation**
   - `Activate_realtime_switch`
3. **Resetting the error**
   - `Reset_all_errors`

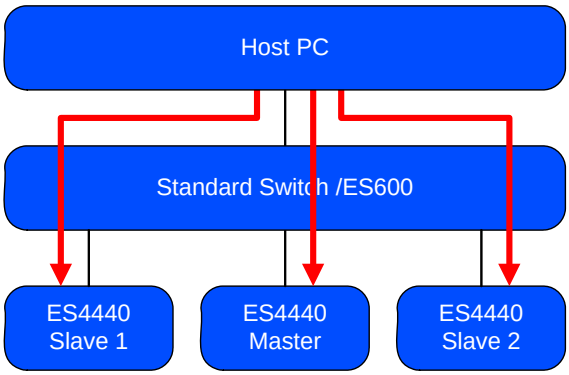
Special Case: Pin-to-Pin Error with Load



1. **Error configuration for the first pin (on Slave 1)**
   - `Pin2PinFirstChRelatimeWithLoad`
2. **Error configuration for the second pin (on Slave 2)**
   - `Pin2PinSecondChRealtimeWithLoad`

3. **Error activation (on Slave 1)**
   - `Activate_realtime_switch`
4. **Resetting the error (on Slave 1)**
   - `Reset_all_errors`
5. **Resetting the error (on Slave 2)**
   - `Reset_all_errors`

Special Case: Pin-to-Pin without Load



1. **Error configuration for the first pin (on Slave 1)**
   - `Pin2PinFirstChWithoutLoad`
2. **Error configuration for the second pin (on Slave 2)**
   - `Pin2PinSecondChWithoutLoad`
3. **Error activation (on the master)**
   - `Activate_relay`
4. **Resetting the error (on Slave 1)**
   - `Reset_all_errors`
5. **Resetting the error (on Slave 2)**
   - `Reset_all_errors`
6. **Resetting the error (on the master)**
   - `Reset_all_errors`

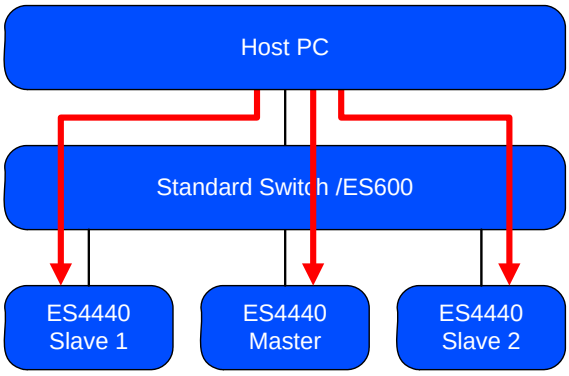
Note

This error is switched with a relay – there is no fuse in the error path between Pin 1 and Pin 2!

3.2.4 Multiple Errors in Master/Slave Applications

The program runs described in this section are valid for multiple errors which are executed on a master/slave system.

Program Run for Errors Generated with a Relay



1. **Error configuration (on Slave 1)**
   Max. 10 high-current errors:
   - `Open_Load`
     and/or
   - `ShortCut_xUBATTy_20A`
2. **Error configuration (on Slave 2)**
   Max. 10 high-current errors:
   - `Open_Load`
     and/or
   - `ShortCut_xUBATTy_20A`
3. **Error activation (on the master)**
   - `Activate_relay`
4. **Resetting the error (on Slave 1)**
   - `Reset_all_errors`
5. **Resetting the error (on Slave 2)**
   - `Reset_all_errors`
6. **Resetting the error (on the master)**
   - `Reset_all_errors`

3.3 Initialization

3.3.1 Initialization of the COM Controller

The initialization procedure of the COM controller is as follows:

```
Dim returnValue As Integer
Dim WireharnessPath As String
'access CommCtrlAccess
Set ctrla = CreateObject
   ("ETAS.PTS.PINCONTROLV2.CommCtrl.CommCtrlAccess")
'using CommCtrlAccess class you can try to access the
singleton instance
Set ctrl = ctrla.CommCtrlInstance
returnValue = ctrl.InitErrorSimulationUsingFile(Wire-
harnessPath)
```

The "Exit" procedure is as follows:

```
'free CommCtrl Singleton instance
    Call ctrla.FreeCommCtrlInstance
```

Note

The COM controller can only be used by one application at a time!

3.3.2 Initialization for CAN

No initialization is necessary for CAN.

3.4 Detailed Description of the Commands

This section contains the complete syntax description of all commands.

These are:

- "The IDN Command" on page 45
- "Open_Load" on page 46
- "Open_Load_realtime" on page 47
- "ShortCut_xUBATTy_20A" on page 48
- "ShortCut_xUBATTy_20A_realtime" on page 49
- "Pin2PinFirstChWithoutLoad" on page 50
- "Pin2PinSecondChannelWithoutLoad" on page 51
- "Pin2PinFirstChRealtimeWithLoad" on page 52
- "Pin2PinSecondChRealtimeWithLoad" on page 53
- "RInline_realtime" on page 54
- "Pullup_Pulldown_xUBATTy_20A_realtime" on page 56
- "Open_Load_400V" on page 57
- "ShortCut_xUBATTy_400V" on page 58
- "ShortCut_xUBATTy_400V_Ex" on page 59
- "Pin_2_Pin_400V" on page 60
- "Pin_2_Pin_400V_Ex" on page 61
- "Reset_all_errors" on page 62
- "Activate_relay" on page 63
- "Activate_realtime_switch" on page 65
- "Test Fuses" on page 67
- "CurrentMeasurement" on page 69

3.4.1 General Command Structure

The following is a description of the general structure of the CAN send and CAN receive messages.

CAN Send Message

***COM_command_name*(*arguments*)**		
1st byte	Command ID	*Value*
2nd byte	Parameter 0	*Value*
3rd byte	Parameter 1	*Value*
4th byte	Parameter 2	*Value*
5th byte	Parameter 3	*Value*
6th byte	Parameter 4	*Value*
7th byte	Parameter 5	*Value*
8th byte	Parameter 6	*Value*

Tab. 3-1 Structure of a Send Message

CAN Receive Message

Answer		
1st byte	Command ID	*Value*
2nd byte	Parameter 0	*Value*
3rd byte	Parameter 1	*Value*
4th byte	Parameter 2	*Value*
5th byte	Parameter 3	*Value*
6th byte	Parameter 4	*Value*
7th byte	Parameter 5	*Value*
8th byte	Parameter 6	*Value*

Tab. 3-2 Structure of the Receive Message

Note

To avoid a buffer overflow in the ES4440.1/.2, CAN messages may only be sent in single-shot mode!

3.4.2 Definitions for all Functions

The information in this section is valid for all commands.

Channel Numbers

High-current channels are counted from 0 - 63; high-voltage channels from 0 - 15.

Definition for Parameter 1 of the CAN Send Message:

Parameter 1 (3rd byte) has the same structure for all functions:

Bit 0	**load**	load = 1: error simulation with load load = 0: error simulation without load
Bit 1 **Bit 2** **Bit 3**	**xUBatty**	xUBatty = 0: +UBatt_A xUBatty = 1: -UBatt_A xUBatty = 2: +UBatt_B xUBatty = 3: -UBatt_B xUBatty = 4:+UBatt_C xUBatty = 5: -UBatt_C
Bit 4	**current**	current = 0: current measuring off current = 1: current measuring on
Bit 5	**set**	set = 1: error set set = 0: error reset
Bit 6	**duration_flag**	duration_flag = 0: error pending infinitely (i.e. until reset). duration_flag = 1: error duration determined by "duration_time".
Bit 7	Not used	

3.4.3 Error Codes

Error codes are transmitted in byte 8 of a receive message – the API command GetLastErrorSimulationAnswer() also transmits the error code in byte 8.

Result	Meaning
0x0	Command OK
0x21	Wrong parameter for slave address (> 16)
0x22	Unknown command
0x23	Wrong data type when writing to flash
0x24	Wrong parameter for LED test
0x25	Wrong number in IP address (must be < 256
0x26	Wrong parameter for CAN baud rate
0x27	Wrong parameter for CAN termination
0x28	Wrong parameter for type of CAN identifier
0x29	Parameter for cascade channel too large (must be < 15)
0x2a	Wrong parameter for resistor cascade
0x2b	not used
0x2c	Wrong address for flash read access (must be < 513)
0x2d	Wrong data length for flash read access (must be < 17)
0x2e	Wrong address for flash write access (must be < 513)
0x2f	Wrong data length for flash write access (must be < 17)
0x30	PLD error
0x31	Error EEPROM checksum
0x32	CAN controller unreachable
0x41	Simulation command returns plausibility error
0x42	Reference relay not detected
0x43	Value of "duration_time" not equal "0xffff", although error should be pending infinitely.
0x44	Simulation command was not recognized
0x45	PLD error: Command could not be executed
0x46	Value of "duration time“ exceeds valid range (1 to 5000 or 0xFFFF)
0x47	Recent error simulation still active, end with `Reset_all_errors`
0x48	Maximum number of relays reached
0x49	Error with multi-error flag
0x4a	Specified channel number exceeds valid range
0x4b	Frequency or duty cycle exceeds valid range
0x4c	System temperature > 60 °C
0x4d	Temperature of resistor cascade > 60 °C
0x4e	MOSFET temperature > 60 °C

Result	Meaning
0x4f	Sensor for system temperature broken
0x50	Sensor for resistor cascade temperature broken
0x51	Sensor for MOSFET temperature broken
0x52	Rail voltage erroneous (possible short-circuit)
0x53	Invalid value for resistance

3.4.4 The IDN Command

This command is used to identify the relevant ES4440.1/.2.

Command for COM Controller

```
int ctrl.CommandIDN(string deviceKey);
```

Possible values of "deviceKey": Standalone, Master, Slave1, ..., Slave14

Return Value from the COM Controller

```
byte[] Answer =
          ctrl.GetLastSystemConfigurationAnswer();
```

CAN Send Message

IDN Command		
1st byte	Command ID	0x0
2nd byte	Parameter 0	Not used
3rd byte	Parameter 1	Not used
4th byte	Parameter 2	Not used
5th byte	Parameter 3	Not used
6th byte	Parameter 4	Not used
7th byte	Parameter 5	Not used
8th byte	Parameter 6	Not used

CAN Receive Message

Answer		
1st byte	Command ID	0x0
2nd byte	Parameter 0	Device Config Byte 1
3rd byte	Parameter 1	Device Config Byte 0
4th byte	Parameter 2	Not used
5th byte	Parameter 3	Not used
6th byte	Parameter 4	Not used
7th byte	Parameter 5	Not used
8th byte	Parameter 6	Command result

Values for "Device Config":

- Standalone: 255
- Master: 0
- Slave 1 ... Slave 14: 1...14

3.4.5 Open_Load

Interrupts a line between ECU and load. This error is switched with a relay – up to ten errors can be switched simultaneously. The value of "channels left" in the command answer specifies how many channels are available for further errors.

Command for COM Controller

```
int ctrl.OpenLoad(string ecu, string ecuPin,
                  int durationType, int set);
```

Return Value from the COM Controller

```
byte[] Answer = ctrl.GetLastErrorSimulationAnswer();
```

CAN Send Message

Open_Load(channel_nr, duration_flag, set)		
1st byte	Command ID	0x1
2nd byte	Parameter 0	ES4440 channel number
3rd byte	Parameter 1	set, duration_flag (see page 42)
4th byte	Parameter 2	Not used
5th byte	Parameter 3	Not used
6th byte	Parameter 4	Not used
7th byte	Parameter 5	Not used
8th byte	Parameter 6	Not used

CAN Receive Message

Answer		
1st byte	Command ID	0x1
2nd byte	Parameter 0	ES4440 channel number
3rd byte	Parameter 1	channels left
4th byte	Parameter 2	Not used
5th byte	Parameter 3	Not used
6th byte	Parameter 4	Not used
7th byte	Parameter 5	Not used
8th byte	Parameter 6	Command result

3.4.6 Open_Load_realtime

Interrupts a line between ECU and load. This error is switched with MOSFETs and can only be realized as a single error.

Command for COM Controller

```
int ctrl.OpenLoad_RealTime(string ecu,
                string ecuPin, int durationType);
```

Return Value from the COM Controller

```
byte[] Answer = ctrl.GetLastErrorSimulationAnswer();
```

Structure of the CAN Message

Open_Load_realtime (channel_nr, duration_flag)		
1st byte	Command ID	0x2
2nd byte	Parameter 0	ES4440 channel number
3rd byte	Parameter 1	duration_flag (see page 42)
4th byte	Parameter 2	Not used
5th byte	Parameter 3	Not used
6th byte	Parameter 4	Not used
7th byte	Parameter 5	Not used
8th byte	Parameter 6	Not used

CAN Receive Message

Answer		
1st byte	Command ID	0x2
2nd byte	Parameter 0	ES4440 channel number
3rd byte	Parameter 1	Not used
4th byte	Parameter 2	Not used
5th byte	Parameter 3	Not used
6th byte	Parameter 4	Not used
7th byte	Parameter 5	Not used
8th byte	Parameter 6	Command result

3.4.7 ShortCut_xUBATTy_20A

Generates a short of a line against a battery voltage with a high-current channel. This error is switched with a relay and can be realized several times.

Command for COM Controller

```
int ctrl.Shortcut_xUBATTy_20A(string ecu,
            string ecuPin, int load, int xUBATTy,
                      int durationType, int set);
```

Return Value from the COM Controller

```
byte[] Answer = ctrl.GetLastErrorSimulationAnswer();
```

CAN Send Message

ShortCut_xUBATTy_20A (channel_nr, load, xUBATTy, duration_flag, set)		
1st byte	Command ID	0x3
2nd byte	Parameter 0	ES4440 channel number
3rd byte	Parameter 1	load, xUBatty, set, duration_flag (see page 42)
4th byte	Parameter 2	Not used
5th byte	Parameter 3	Not used
6th byte	Parameter 4	Not used
7th byte	Parameter 5	Not used
8th byte	Parameter 6	Not used

CAN Receive Message

Answer		
1st byte	Command ID	0x3
2nd byte	Parameter 0	ES4440 channel number
3rd byte	Parameter 1	channels left
4th byte	Parameter 2	Not used
5th byte	Parameter 3	Not used
6th byte	Parameter 4	Not used
7th byte	Parameter 5	Not used
8th byte	Parameter 6	Command result

3.4.8 ShortCut_xUBATTy_20A_realtime

Generates a short of a line against a battery voltage with a high-current channel. This error is switched with MOSFETs and can thus only be realized as a single error.

Command for COM Controller

```
int ctrl.Shortcut_xUBATTy_20A_RealTime
        (string ecu, string ecuPin, int load,
                  int xUBATTy, int durationType);
```

Return Value from the COM Controller

```
byte[] Answer = ctrl.GetLastErrorSimulationAnswer();
```

CAN Send Message

ShortCut_xUBATTy_20A_realtime (channel_nr, load, xUBATTy, duration_flag)		
1st byte	Command ID	0x4
2nd byte	Parameter 0	ES4440 channel number
3rd byte	Parameter 1	load, xUBatty, duration_flag (see page 42)
4th byte	Parameter 2	Not used
5th byte	Parameter 3	Not used
6th byte	Parameter 4	Not used
7th byte	Parameter 5	Not used
8th byte	Parameter 6	Not used

CAN Receive Message

Answer		
1st byte	Command ID	0x4
2nd byte	Parameter 0	ES4440 channel number
3rd byte	Parameter 1	Not used
4th byte	Parameter 2	Not used
5th byte	Parameter 3	Not used
6th byte	Parameter 4	Not used
7th byte	Parameter 5	Not used
8th byte	Parameter 6	Command result

3.4.9 Pin2PinFirstChWithoutLoad

Defines the first line for a short between two lines – the second line is defined with the command "Pin2PinSecondChannelWithoutLoad" (see page 51).

Note

This error is switched with a relay and realized without a load and without resistance. There is no fuse between Pin 1 and Pin 2!

Command for COM Controller

```
int ctrl.Pin2PinChannel1_WithoutLoad(string ecu,
                  string ecuPin, int durationType);
```

Return Value from the COM Controller

```
byte[] Answer = ctrl.GetLastErrorSimulationAnswer();
```

CAN Send Message

Pin2PinFirstChWithoutLoad (channel_nr1, duration_flag)		
1st byte	Command ID	0x5
2nd byte	Parameter 0	ES4440 channel number 1
3rd byte	Parameter 1	duration_flag (see page 42)
4th byte	Parameter 2	Not used
5th byte	Parameter 3	Not used
6th byte	Parameter 4	Not used
7th byte	Parameter 5	Not used
8th byte	Parameter 6	Not used

CAN Receive Message

Answer		
1st byte	Command ID	0x5
2nd byte	Parameter 0	ES4440 channel number 1
3rd byte	Parameter 1	Not used
4th byte	Parameter 2	Not used
5th byte	Parameter 3	Not used
6th byte	Parameter 4	Not used
7th byte	Parameter 5	Not used
8th byte	Parameter 6	Command result

3.4.10 Pin2PinSecondChannelWithoutLoad

Defines the second line for a short between two lines.

Note

This error is switched with a relay and realized without a load and without resistance. There is no fuse between Pin 1 and Pin 2!

Command for COM Controller

```
int ctrl.Pin2PinChannel2_WithoutLoad(string ecu,
                  string ecuPin, int durationType);
```

Return Value from the COM Controller

```
byte[] Answer = ctrl.GetLastErrorSimulationAnswer();
```

CAN Send Message

Pin2PinSecondChannelWithoutLoad (channel_nr1, duration_flag)		
1st byte	Command ID	0x6
2nd byte	Parameter 0	ES4440 channel number 2
3rd byte	Parameter 1	duration_flag (see page 42)
4th byte	Parameter 2	Not used
5th byte	Parameter 3	Not used
6th byte	Parameter 4	Not used
7th byte	Parameter 5	Not used
8th byte	Parameter 6	Not used

CAN Receive Message

Answer		
1st byte	Command ID	0x6
2nd byte	Parameter 0	ES4440 channel number 2
3rd byte	Parameter 1	Not used
4th byte	Parameter 2	Not used
5th byte	Parameter 3	Not used
6th byte	Parameter 4	Not used
7th byte	Parameter 5	Not used
8th byte	Parameter 6	Command result

3.4.11 Pin2PinFirstChRealtimeWithLoad

Defines the first line for a short between two lines. This error enables a simulation with load and finite resistance between the two lines.

This error is switched with MOSFETs.

Command for COM Controller

```
int ctrl.Pin2PinChannel1_RealTime_WithLoad
     (string ecu, string ecuPin, int resistor,
                    int current, int durationType);
```

Return Value from the COM Controller

```
byte[] Answer = ctrl.GetLastErrorSimulationAnswer();
```

CAN Send Message

Pin2PinFirstChRealtimeWithLoad (channel_nr1, duration_flag)		
1st byte	Command ID	0x7
2nd byte	Parameter 0	ES4440 channel number 1
3rd byte	Parameter 1	current, duration_flag (see page 42)
4th byte	Parameter 2	Not used
5th byte	Parameter 3	Resistance byte 0
6th byte	Parameter 4	Resistance byte 1
7th byte	Parameter 5	Resistance byte 2
8th byte	Parameter 6	Resistance byte 3

CAN Receive Message

Answer		
1st byte	Command ID	0x7
2nd byte	Parameter 0	ES4440 channel number 1
3rd byte	Parameter 1	Not used
4th byte	Parameter 2	Not used
5th byte	Parameter 3	Not used
6th byte	Parameter 4	Not used
7th byte	Parameter 5	Not used
8th byte	Parameter 6	Command result

3.4.12 Pin2PinSecondChRealtimeWithLoad

Defines the second line for a short between two lines. This error enables a simulation with load and finite resistance between the two lines.

This error is switched with MOSFETs.

Command for COM Controller

```
int ctrl.Pin2PinChannel2_RealTime_WithLoad
     (string ecu, string ecuPin, int durationType);
```

Return Value from the COM Controller

```
byte[] Answer = ctrl.GetLastErrorSimulationAnswer();
```

CAN Send Message

Pin2PinSecondChRealtimeWithLoad (channel_nr2, duration_flag)		
1st byte	Command ID	0x8
2nd byte	Parameter 0	ES4440 channel number 2
3rd byte	Parameter 1	duration_flag (see page 42)
4th byte	Parameter 2	Not used
5th byte	Parameter 3	Not used
6th byte	Parameter 4	Not used
7th byte	Parameter 5	Not used
8th byte	Parameter 6	Not used

CAN Receive Message

Answer		
1st byte	Command ID	0x8
2nd byte	Parameter 0	ES4440 channel number 2
3rd byte	Parameter 1	Not used
4th byte	Parameter 2	Not used
5th byte	Parameter 3	Not used
6th byte	Parameter 4	Not used
7th byte	Parameter 5	Not used
8th byte	Parameter 6	Not used

3.4.13 RInline_realtime

Switches a line resistance with MOSFETs.

Command for COM Controller

```
int ctrl.RInline_RealTime(string ecu1,
            string ecuPin1, int resistor,
                   int current, int durationType);
```

Return Value from the COM Controller

```
byte[] Answer = ctrl.GetLastErrorSimulationAnswer();
```

CAN Send Message

RInline_realtime (channel_nr1, resistor, duration_flag, current)		
1st byte	Command ID	0x9
2nd byte	Parameter 0	ES4440 channel number
3rd byte	Parameter 1	current, duration_flag (see page 42)
4th byte	Parameter 2	Not used
5th byte	Parameter 3	Resistance byte 0 (LSB) *
6th byte	Parameter 4	Resistance byte 1 *
7th byte	Parameter 5	Resistance byte 2 *
8th byte	Parameter 6	Resistance byte 3 (MSB) *

* The byte order corresponds to Motorola format:

```
Byte n    (LSB)
Byte n+1
Byte n+2
Byte n+3  (MSB)
```

A resistance value of 0x1234 thus has the following byte order:

```
Byte n    0x34 (LSB)
Byte n+1  0x12
Byte n+2
Byte n+3       (MSB)
```

CAN Receive Message

Answer		
1st byte	Command ID	0x9
2nd byte	Parameter 0	ES4440 channel number
3rd byte	Parameter 1	Not used
4th byte	Parameter 2	Not used
5th byte	Parameter 3	Not used
6th byte	Parameter 4	Not used
7th byte	Parameter 5	Not used
8th byte	Parameter 6	Command result

3.4.14 Pullup_Pulldown_xUBATTy_20A_realtime

Applies a channel to a battery voltage via a resistor (pull-up/pull-down).

Command for COM Controller

```
int ctrl.PullUp_PullDown_20A_RealTime
   (string ecu, string ecuPin, int load, int xUBATTy,
     int resistor, int current, int durationType);
```

Return Value from the COM Controller

```
byte[] Answer = ctrl.GetLastErrorSimulationAnswer();
```

CAN Send Message

Pullup_Pulldown_xUBATTy_20A_realtime (channel_nr, load, xUBATTy, resistor, duration_flag, current)		
1st byte	Command ID	0xB
2nd byte	Parameter 0	ES4440 channel number
3rd byte	Parameter 1	xUBatty, current, duration_flag, load (see page 42)
4th byte	Parameter 2	Not used
5th byte	Parameter 3	Resistance byte 0 (LSB) *
6th byte	Parameter 4	Resistance byte 1 *
7th byte	Parameter 5	Resistance byte 2 *
8th byte	Parameter 6	Resistance byte 3 (MSB) *

* For information on the representation of the bytes see "RInline_realtime" on page 54.

CAN Receive Message

Answer		
1st byte	Command ID	0xB
2nd byte	Parameter 0	ES4440 channel number
3rd byte	Parameter 1	Not used
4th byte	Parameter 2	Not used
5th byte	Parameter 3	Not used
6th byte	Parameter 4	Not used
7th byte	Parameter 5	Not used
8th byte	Parameter 6	Command result

3.4.15 Open_Load_400V

Interrupts a high-voltage line between ECU and load.

This error is switched with a relay and is only possible as a single error.

Command for COM Controller

```
int ctrl.OpenLoad_400V(string ecu, string ecuPin,
                       int durationType, int set);
```

Return Value from the COM Controller

```
byte[] Answer = ctrl.GetLastErrorSimulationAnswer();
```

CAN Send Message

Open_Load_400V (channel_nr, set, duration_flag)		
1st byte	Command ID	0xD
2nd byte	Parameter 0	ES4440 channel number
3rd byte	Parameter 1	duration_flag, set (see page 42)
4th byte	Parameter 2	Not used
5th byte	Parameter 3	Not used
6th byte	Parameter 4	Not used
7th byte	Parameter 5	Not used
8th byte	Parameter 6	Not used

CAN Receive Message

Answer		
1st byte	Command ID	0xD
2nd byte	Parameter 0	ES4440 channel number
3rd byte	Parameter 1	Not used
4th byte	Parameter 2	Not used
5th byte	Parameter 3	Not used
6th byte	Parameter 4	Not used
7th byte	Parameter 5	Not used
8th byte	Parameter 6	Command result

3.4.16 ShortCut_xUBATTy_400V

This command is used to short a high-voltage channel with a battery voltage. This error is switched with a relay and is only possible as a single error and without load.

Command for COM Controller

```
int ctrl.ShortCut_xUBATTy_400V
        (string ecu, string ecuPin, int xUBATTy,
                        int durationType, int set);
```

Return Value from the COM Controller

```
byte[] Answer = ctrl.GetLastErrorSimulationAnswer();
```

CAN Send Message

ShortCut_xUBATTy_400V (channel_nr, xUBATTy, set, duration_flag)		
1st byte	Command ID	0xE
2nd byte	Parameter 0	ES4440 channel number
3rd byte	Parameter 1	xUBATTy, duration_flag, set (see page 42)
4th byte	Parameter 2	Not used
5th byte	Parameter 3	Not used
6th byte	Parameter 4	Not used
7th byte	Parameter 5	Not used
8th byte	Parameter 6	Not used

CAN Receive Message

Answer		
1st byte	Command ID	0xE
2nd byte	Parameter 0	ES4440 channel number
3rd byte	Parameter 1	Not used
4th byte	Parameter 2	Not used
5th byte	Parameter 3	Not used
6th byte	Parameter 4	Not used
7th byte	Parameter 5	Not used
8th byte	Parameter 6	Command result

3.4.17 ShortCut_xUBATTy_400V_Ex

This command is used to short a high-voltage channel with a battery voltage. This error is switched with a relay and is only possible as a single error. It can be applied with or without a load.

Command for COM Controller

```
int ctrl.ShortCut_xUBATTy_400V_Ex
        (string ecu, string ecuPin, int xUBATTy,
                  int load, int durationType, int set);
```

Return Value from the COM Controller

```
byte[] Answer = ctrl.GetLastErrorSimulationAnswer();
```

CAN Send Message

ShortCut_xUBATTy_400V (channel_nr, load, xUBATTy, set, duration_flag)		
1st byte	Command ID	0xE
2nd byte	Parameter 0	ES4440 channel number
3rd byte	Parameter 1	load, xUBATTy, duration_flag, set (see page 42)
4th byte	Parameter 2	Not used
5th byte	Parameter 3	Not used
6th byte	Parameter 4	Not used
7th byte	Parameter 5	Not used
8th byte	Parameter 6	Not used

CAN Receive Message

Answer		
1st byte	Command ID	0xE
2nd byte	Parameter 0	ES4440 channel number
3rd byte	Parameter 1	Not used
4th byte	Parameter 2	Not used
5th byte	Parameter 3	Not used
6th byte	Parameter 4	Not used
7th byte	Parameter 5	Not used
8th byte	Parameter 6	Command result

3.4.18 Pin_2_Pin_400V

Generates a short between two high-voltage channels (without a load).

This error is switched with a relay and is only possible as a single error and without load.

Command for COM Controller

```
int ctrl.Pin2Pin_400V
    (string ecu1, string ecuPin1, string ecu2,
                string ecuPin2, int durationType);
```

Return Value from the COM Controller

```
byte[] Answer = ctrl.GetLastErrorSimulationAnswer();
```

CAN Send Message

Pin_2_Pin_400V (channel_nr1, channel_nr2, duration_flag)		
1st byte	Command ID	0xF
2nd byte	Parameter 0	ES4440 channel number 1
3rd byte	Parameter 1	duration_flag (see page 42)
4th byte	Parameter 2	ES4440 channel number 2
5th byte	Parameter 3	Not used
6th byte	Parameter 4	Not used
7th byte	Parameter 5	Not used
8th byte	Parameter 6	Not used

CAN Receive Message

Answer		
1st byte	Command ID	0xF
2nd byte	Parameter 0	ES4440 channel number 1
3rd byte	Parameter 1	Not used
4th byte	Parameter 2	ES4440 channel number 2
5th byte	Parameter 3	Not used
6th byte	Parameter 4	Not used
7th byte	Parameter 5	Not used
8th byte	Parameter 6	Command result

3.4.19 Pin_2_Pin_400V_Ex

Generates a short between two high-voltage channels (without a load). This error is switched with a relay and is only possible as a single error. It can be applied with or without a load.

Command for COM Controller

```
int ctrl.Pin2Pin_400V_Ex
    (string ecu1, string ecuPin1, string ecu2,
          string ecuPin2, int load, int durationType);
```

Return Value from the COM Controller

```
byte[] Answer = ctrl.GetLastErrorSimulationAnswer();
```

CAN Send Message

Pin_2_Pin_400V_Ex (channel_nr1, channel_nr2, load, duration_flag)		
1st byte	Command ID	0xF
2nd byte	Parameter 0	ES4440 channel number 1
3rd byte	Parameter 1	load, duration_flag (see page 42)
4th byte	Parameter 2	ES4440 channel number 2
5th byte	Parameter 3	Not used
6th byte	Parameter 4	Not used
7th byte	Parameter 5	Not used
8th byte	Parameter 6	Not used

CAN Receive Message

Answer		
1st byte	Command ID	0xF
2nd byte	Parameter 0	ES4440 channel number 1
3rd byte	Parameter 1	Not used
4th byte	Parameter 2	ES4440 channel number 2
5th byte	Parameter 3	Not used
6th byte	Parameter 4	Not used
7th byte	Parameter 5	Not used
8th byte	Parameter 6	Command result

3.4.20 Reset_all_errors

This command is used to reset all errors on an ES4440.1/.2 Compact Failure Simulation Module.

Command for COM Controller

```
int ctrl.ResetAllErrors();
```

Return Value from the COM Controller

```
byte[] Answer = ctrl.GetLastErrorSimulationAnswer();
```

CAN Send Message

Reset_all_errors ()		
1st byte	Command ID	0x10
2nd byte	Parameter 0	Not used
3rd byte	Parameter 1	Not used
4th byte	Parameter 2	Not used
5th byte	Parameter 3	Not used
6th byte	Parameter 4	Not used
7th byte	Parameter 5	Not used
8th byte	Parameter 6	Not used

CAN Receive Message

Answer		
1st byte	Command ID	0x10
2nd byte	Parameter 0	Not used
3rd byte	Parameter 1	Not used
4th byte	Parameter 2	Not used
5th byte	Parameter 3	Not used
6th byte	Parameter 4	Not used
7th byte	Parameter 5	Not used
8th byte	Parameter 6	Command result

3.4.21 Activate_relay

This command is used to close the relay for a specific time.

If "duration_flag" (bit 6 in the 3rd byte) is set in the previous error command (= 1), "duration_time" can be selected as being between 20 ms and 5 s. If "duration_flag" = 0, "duration_time" has to be selected as = -1 or = 65535 (0xFFFF).

Note

When setting multiple errors, the "duration_flag" parameters of all errors must have the same value!

The switch times measured on the reference relay are transferred in the command answer.

Command for COM Controller

```
int ctrl.ActivateRelay(int durationTime);
```

Return Value from the COM Controller

```
byte[] Answer = ctrl.GetLastErrorSimulationAnswer();
```

CAN Send Message

Activate_relay (duration_ time)		
1st byte	Command ID	0x12
2nd byte	Parameter 0	Not used
3rd byte	Parameter 1	duration_time in ms (Byte 0)
4th byte	Parameter 2	duration_time in ms (Byte 1)
5th byte	Parameter 3	Not used
6th byte	Parameter 4	Not used
7th byte	Parameter 5	Not used
8th byte	Parameter 6	Not used

The "duration_time" parameter can be specified in intervals of 20 ms – the smallest possible value is 20 ms, the greatest possible value 5000 ms.

Continuous errors are generated with the value 0xFFFF.

The "channel_type" parameter can have the following values:

- channel_type = 0: high-current channel
- channel_type = 1: high-voltage channel

CAN Receive Message

Answer		
1st byte	Command ID	0x12
2nd byte	Parameter 0	delay_time0 NO contact* 20 A closes (in 100 µs intervals)
3rd byte	Parameter 1	delay_time1 NO contact* 20 A closes (in 100 µs intervals)
4th byte	Parameter 2	delay_time0 NC contact** 20 A opens (in 100 µs intervals)
5th byte	Parameter 3	delay_time1 NC contact** 20 A opens (in 10 µs intervals)
6th byte	Parameter 4	delay_time0 NC contact** 400 V closes (in 100 µs intervals)
7th byte	Parameter 5	NC contact** 400 V closes (in 100 µs intervals)
8th byte	Parameter 6	Command result

* NO = normally open
** NC = normally closed

3.4.22 Activate_realtime_switch

This command is used to close an error switched with MOSFETs for a certain length of time.

If "duration_flag" (bit 6 in the 3rd byte) is set in the previous error command (= 1), "duration_time" can be selected as being between 1 ms and 5 s. If "duration_flag" = 0, "duration_time" has to be selected as = -1 or = 65535 (0xFFFF).

Command for COM Controller

```
int ctrl.ActivateRealTimeSwitch(int mode,
     int durationTime, int dutyCycle, int frequency);
```

Return Value from the COM Controller

```
byte[] Answer = ctrl.GetLastErrorSimulationAnswer();
```

CAN Send Message

Activate_realtime_switch (mode, duration_time, dutycycle, frequency)		
1st byte	Command ID	0x13
2nd byte	Parameter 0	mode
3rd byte	Parameter 1	duration_time (Byte 0)
4th byte	Parameter 2	duration_time (Byte 1)
5th byte	Parameter 3	Not used
6th byte	Parameter 4	Duty cycle with a loose contact
7th byte	Parameter 5	Frequency with a loose contact (Byte 0)
8th byte	Parameter 6	Frequency with a loose contact (Byte 1)

The "duration_time" parameter can be specified in intervals of 1 ms – the smallest possible value is 1 ms, the greatest possible value 5000 ms. Continuous errors are generated with the value 0xFFFF.

In order to deactivate loose contact simulation the parameters 4, 5, and 6 must be set to 0xFF.

The "mode" parameter can have the following values:

- mode = 0: static errors the duration of which is defined by "duration_time".
- mode = 1: loose contact simulation

  There are the following limitations for duty cycle and frequency: 1 - 99% at 3 Hz to 100 Hz, 50% at 2 Hz

CAN Receive Message

Answer		
1st byte	Command ID	0x13
2nd byte	Parameter 0	mode
3rd byte	Parameter 1	duration_time (Byte 0)
4th byte	Parameter 2	duration_time (Byte 1)
5th byte	Parameter 3	duration_time (Byte 2)
6th byte	Parameter 4	duration_time (Byte 3)
7th byte	Parameter 5	Not used
8th byte	Parameter 6	Command result

3.4.23 Test Fuses

Determines the status of the fuses for the error rails.

Command for COM Controller

```
int ctrl.TestFuses(string deviceKey,
                           out int[] fuses);
```

The status of the fuses is stored in the array `fuses`: `fuses[n]` contains the status of the fuse n+1 (n=0 ... 4). 1 means "fuse OK", 0 means "fuse defective".

Possible values for "deviceKey": Standalone, Master, Slave1, ..., Slave14

Return Value from the COM Controller

```
byte[] Answer = ctrl.GetLastErrorSimulationAnswer();
```

CAN Send Message

1st byte	Command ID	0x14
2nd byte	Parameter 0	Not used
3rd byte	Parameter 1	Not used
4th byte	Parameter 2	Not used
5th byte	Parameter 3	Not used
6th byte	Parameter 4	Not used
7th byte	Parameter 5	Not used
8th byte	Parameter 6	Not used

CAN Receive Message

Answer		
1st byte	Command ID	0x14
2nd byte	Parameter 0	Fuse_status
3rd byte	Parameter 1	Not used
4th byte	Parameter 2	Not used
5th byte	Parameter 3	Not used
6th byte	Parameter 4	Not used
7th byte	Parameter 5	Not used
8th byte	Parameter 6	Command result

The fuse status is determined via the relevant bit (0 = fuse defective, 1 = fuse OK):

MSB							LSB
Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1
-	-	-	E5	E1	E3	E4	E2

3.4.24 CurrentMeasurement

Routes a channel via the "Current" connectors on the front panel of the ES4440.1/.2 Compact Failure Simulation Module to measure the current.

Command for COM Controller

```
int ctrl.CurrentMeasurement
          (string ecu, string ecuPin, byte set);
```

Return Value from the COM Controller

```
byte[] Answer = ctrl.GetLastErrorSimulationAnswer();
```

CAN Send Message

CurrentMeasurement (channel_nr)		
1st byte	Command ID	0x15
2nd byte	Parameter 0	channel_nr
3rd byte	Parameter 1	Not used
4th byte	Parameter 2	Not used
5th byte	Parameter 3	Not used
6th byte	Parameter 4	Not used
7th byte	Parameter 5	Not used
8th byte	Parameter 6	Not used

CAN Receive Message

Answer		
1st byte	Command ID	0x15
2nd byte	Parameter 0	channel_nr
3rd byte	Parameter 1	Not used
4th byte	Parameter 2	Not used
5th byte	Parameter 3	Not used
6th byte	Parameter 4	Not used
7th byte	Parameter 5	Not used
8th byte	Parameter 6	Command result

4 ETAS Contact Addresses

ETAS HQ

ETAS GmbH

Borsigstraße 14 Phone: +49 711 3423-0

70469 Stuttgart Fax: +49 711 3423-2106

Germany WWW: www.etas.com

ETAS Subsidiaries and Technical Support

For details of your local sales office as well as your local technical support team and product hotlines, take a look at the ETAS website:

ETAS subsidiaries	WWW:	www.etas.com/en/contact.php
ETAS technical support	WWW:	www.etas.com/en/hotlines.php

Index

A
API documentation 31

C
CAN command
  Activate_realtime_switch 65
  Activate_relay 63
  CurrentMeasurement 69
  general command structure 41
  Open_Load 46
  Open_Load_400V 57
  Open_Load_realtime 47
  Pin_2_Pin_400V 60
  Pin_2_Pin_400V_Ex 61
  Pin2PinFirstChRealtimeWithLoad 52
  Pin2PinFirstChWithoutLoad 50
  Pin2PinSecondChannelWithoutLoad 51
  Pin2PinSecondChRealtimeWithLoad 53
  Pullup_Pulldown_xUBATTy_20A_realtime 56
  Reset_all_errors 62
  RInline_realtime 54
  ShortCut_xUBATTy_20A 48
  ShortCut_xUBATTy_20A_realtime 49
  ShortCut_xUBATTy_400V 58
  ShortCut_xUBATTy_400V_Ex 59
  test fuses 67
CAN interface
  configuration 11
channels left 46
Configuration
  CAN interface 11
  ES4440.1 10
Contact resistance
  simulating 24
Current
  measuring 24

D
Display options
  specifying 27

E
ES4440.1
  configuration 10
ETAS Contact Addresses 71
Ethernet
  settings 9

F
Failure set
adding a signal 20
creating 19
removing a signal 21
Fuses
testing 26

I
Introduction 5

L
LABCAR-PINCONTROL
launching 17
Loose contact
simulating 23

M
Main menu 28
Master/Slave
configuration 10
Multiple errors
setting 22

O
Operation
conventions 6
use-case 6

R
Relay contacts
cleaning 25

S
Self test
running 27
Signals
importing from the wire harness file 18

T
TCP/IP
configuring 7

U
User profile 5

W
Wire harness file 12
creating 12