ControlDesk

ECU Diagnostics

For ControlDesk 7.4

Release 2021-A – May 2021

dSPACE

How to Contact dSPACE

Mail:	dSPACE GmbH Rathenaustraße 26 33102 Paderborn Germany
Tel.:	+49 5251 1638-0
Fax:	+49 5251 16198-0
E-mail:	info@dspace.de
Web:	http://www.dspace.com

How to Contact dSPACE Support

If you encounter a problem when using dSPACE products, contact your local dSPACE representative:

- Local dSPACE companies and distributors: http://www.dspace.com/go/locations
- For countries not listed, contact dSPACE GmbH in Paderborn, Germany.
  Tel.: +49 5251 1638-941 or e-mail: support@dspace.de

You can also use the support request form: http://www.dspace.com/go/supportrequest. If you are logged on to mydSPACE, you are automatically identified and do not need to add your contact details manually.

If possible, always provide the relevant dSPACE License ID or the serial number of the CmContainer in your support request.

Software Updates and Patches

dSPACE strongly recommends that you download and install the most recent patches for your current dSPACE installation. Visit http://www.dspace.com/go/patches for software updates and patches.

Important Notice

This publication and the contents hereof are subject to change without notice.

Contents

About This Document

Content

This document introduces you to ECU diagnostics with ControlDesk.

Symbols

dSPACE user documentation uses the following symbols:

Symbol	Description
⚠ DANGER	Indicates a hazardous situation that, if not avoided, will result in death or serious injury.
⚠ WARNING	Indicates a hazardous situation that, if not avoided, could result in death or serious injury.
⚠ CAUTION	Indicates a hazardous situation that, if not avoided, could result in minor or moderate injury.
NOTICE	Indicates a hazard that, if not avoided, could result in property damage.
Note	Indicates important information that you should take into account to avoid malfunctions.
Tip	Indicates tips that can make your work easier.
?	Indicates a link that refers to a definition in the glossary, which you can find at the end of the document unless stated otherwise.
📖	Precedes the document title in a link that refers to another document.

Naming conventions

dSPACE user documentation uses the following naming conventions:

%name% Names enclosed in percent signs refer to environment variables for file and path names.

< > Angle brackets contain wildcard characters or placeholders for variable file and path names, etc.

Special folders

Some software products use the following special folders:

Common Program Data folder A standard folder for application-specific configuration data that is used by all users.

`%PROGRAMDATA%\dSPACE\<InstallationGUID>\<ProductName>`

or

`%PROGRAMDATA%\dSPACE\<ProductName>\<VersionNumber>`

Documents folder A standard folder for user-specific documents.

`%USERPROFILE%\Documents\dSPACE\<ProductName>\<VersionNumber>`

Local Program Data folder A standard folder for application-specific configuration data that is used by the current, non-roaming user.

`%USERPROFILE%\AppData\Local\dSPACE\<InstallationGUID>\<ProductName>`

Accessing dSPACE Help and PDF Files

After you install and decrypt dSPACE software, the documentation for the installed products is available in dSPACE Help and as PDF files.

dSPACE Help (local) You can open your local installation of dSPACE Help:

- On its home page via Windows Start Menu
- On specific content using context-sensitive help via **F1**

dSPACE Help (Web) You can access the Web version of dSPACE Help at www.dspace.com/go/help.

To access the Web version, you must have a *mydSPACE* account.

PDF files You can access PDF files via the 📄 icon in dSPACE Help. The PDF opens on the first page.

New Features

New ECU Diagnostics Features (ControlDesk 7.4)

Support of CAN FD via built-in D-PDU API

ControlDesk now supports *CAN with Flexible Data Rate* (CAN FD) in connection with CAN-based ECU diagnostics via a built-in D-PDU API. The built-in D-PDU API supports all the CAN interfaces listed in Supported CAN Interfaces (ControlDesk Platform Management 🕮).

For information on the mandatory communication parameters when using a CAN FD bus, refer to Conventions in Connection with ODX Databases on page 22.

Related topics

Basics

Supported CAN Interfaces (ControlDesk Platform Management 🕮)

Basics and Instructions

Where to go from here

Information in this section

Introduction to ECU Diagnostics

Where to go from here

Information in this section

Basics of ECU Diagnostics with ControlDesk

Introduction

ECU Diagnostics with ControlDesk lets you calibrate and validate ECU diagnostics functions. You can work with an ECU's fault memory, and execute diagnostic services and jobs.

ECU diagnostics based on ODX

The diagnostic support provided by ControlDesk is based on Open Diagnostic Data Exchange (ODX), the ASAM MCD-2 D (ISO 22901) standard. ODX describes communication with ECUs via diagnostic interfaces.

Supported ODX versions ControlDesk supports the following ODX database standards:

- ASAM MCD-2 D V2.0.1
- ASAM MCD-2 D V2.2.0 (ISO 22901-1)

Accessing an ECU

In ControlDesk, you use an ECU Diagnostics device to access an ECU via a diagnostic interface and to configure the connection. For example, you have to select a vehicle and an access path (logical link) to the ECU. An ODX database is used as the central ECU description for working with an ECU Diagnostics device in ControlDesk. For more information on the ODX database and on using the

ECU Diagnostics device, refer to ECU Diagnostics Device Configuration on page 32.

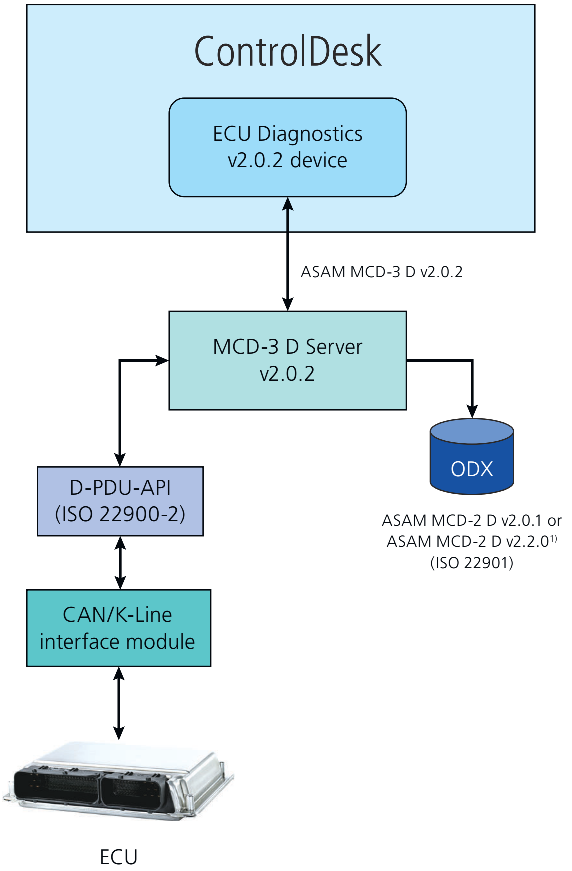
ECU Diagnostics device

ControlDesk provides the *ECU Diagnostics v2.0.2* device, which supports the ASAM MCD-3 D V2.0.2 standard.

COMPARAM identification ControlDesk uses the short name of the parameter to identify a `COMPARAM`. ControlDesk supports the standard communication parameters defined in `D-PDU API` according to ISO 22900-2. `COMPARAMs` that are unknown to ControlDesk are marked.

Architecture overview

The following illustration shows the principle system architecture when performing ECU Diagnostics with ControlDesk.



[1] ControlDesk supports the mandatory functionalities of ASAM MCD-2 D v2.2.0. ControlDesk does not support all the functionalities of ASAM MCD-2 D v2.2.0. For further information, refer to Limitations for ECU Diagnostics on page 147.

Supported diagnostic protocols

ControlDesk's ECU Diagnostics device communicates with ECUs connected to the ControlDesk PC via diagnostic protocols implemented on the ECUs.

Communication is via CAN, K-Line, or Ethernet. ECU access is possible via the following diagnostic protocols:

Diagnostic Protocol	Description
Diagnostics on CAN (ISO 15765)	Diagnostics on CAN / KWP2000 on CAN
KWP2000 on K-Line (ISO 14230)	KWP2000 on K-Line
OBD	On-Board Diagnostics for CAN-based ECU diagnostics
TP 1.6	Transport Protocol TP 1.6 for CAN-based ECU diagnostics
TP 2.0	Transport Protocol TP 2.0 for CAN-based ECU diagnostics
UDS (ISO 14229)	UDS on CAN / UDS on DoIP (Diagnostics over Internet Protocol)

Supported hardware interfaces

For CAN-based diagnostic protocols, ControlDesk supports various interface modules to access the ECU. Refer to Supported CAN Interfaces (ControlDesk Platform Management 🕮).

For K-Line-based logical links, ControlDesk supports various interface modules to access the ECU. Refer to Supported K-Line Interfaces (ControlDesk Platform Management 🕮)

For the Ethernet-based UDS on DoIP protocol, ControlDesk uses available Ethernet interfaces of the host PC to access the ECU. No further interface module is required.

ECU diagnostics functions

To perform ECU diagnostics tasks, ControlDesk provides the following ECU diagnostics functions:

- Reading fault memory (refer to Reading the Fault Memory of an ECU on page 56)
- Reading environment data (refer to Reading Environment Data on page 57)
- Deleting fault memory entries (refer to Clearing Fault Memory Entries on page 60)
- Measuring and calibrating diagnostics variables (refer to Measuring and Calibrating Variables via the ECU Diagnostics Device on page 76)
- Getting security access (refer to Getting Security Access for Parameter Calibration on page 80)
- Measuring diagnostic trouble codes (refer to Performing DTC Measurements on page 81)
- Communicating with an ECU via diagnostic services, diagnostic jobs, and control primitives (refer to Configuring and Executing Diagnostic Communication Objects on page 67)
- Programming the ECU flash memory via diagnostic protocol (refer to How to Program the ECU Flash Memory via a Diagnostic Protocol on page 88)

Instruments

The ControlDesk ECU Diagnostics Module provides two instruments for ECU diagnostics tasks.

Fault Memory Instrument An instrument for reading, clearing, and saving the content of the ECU's fault memory.

Diagnostics Instrument An instrument for communicating with an ECU via the diagnostic protocol using diagnostic services, diagnostic jobs, and control primitives.

Automating ECU diagnostics tasks

ECU diagnostics tasks can be automated in the following ways:

- Via ControlDesk's automation interface. Refer to Automating ECU Diagnostics Tasks on page 136.
- Via AutomationDesk. Refer to AutomationDesk Accessing ControlDesk.
- Via ControlDesk's ASAM MCD-3-compatible interface. Refer to Automating ControlDesk's Diagnostics Features (ControlDesk MCD-3 Automation).

ECU Diagnostics demo project

ControlDesk comes with an ECU Diagnostics demo project which gives you an impression of working with the instruments for ECU diagnostics. Communication with the ECU and with other instruments in the project is limited due to using the CalDemo ECU.

Refer to ECU Diagnostics Demo on page 13.

ControlDesk ECU Diagnostics Module

An optional software module for ControlDesk that facilitates the calibration and validation of ECU diagnostic functions.

- Compliance with ODX database standard
- Support of ISO protocols for CAN/CAN FD (requires a supported PC-based CAN interface such as the DCI-CAN2 or the DCI-CAN/LIN1), Ethernet (requires no further interface), and K-Line (requires a supported K-Line interface (refer to Supported K-Line Interfaces (ControlDesk Platform Management)) such as the DCI-KLine1)
- Instruments for executing diagnostic services and jobs, and for reading and clearing the fault memory of an ECU
- ECU flash memory programming via diagnostic interfaces

Related topics

Basics

ECU Diagnostics Device Configuration.. 32

ECU Diagnostics Demo

Opening demo projects

For instructions on opening demo projects, refer to Opening a demo project (ControlDesk Introduction and Overview).

Description of the demo project

The *DiagDemo* project lets you perform the following tasks even without a real ECU connected to your host PC:

- ECU diagnostics with ControlDesk's diagnostics instruments via the ECU Diagnostics device
- Measurement and calibration of diagnostics variables via the ECU Diagnostics device in connection with ControlDesk's standard instruments

Required products and modules Working with this demo requires:

- *ControlDesk*

Demo setup The *DiagDemo* project contains the *ECU Diagnostics (MCD-3D v2.0.2)* experiment, which contains the following devices:

- *ECU Diagnostics (MCD-3D v2.0.2)* device

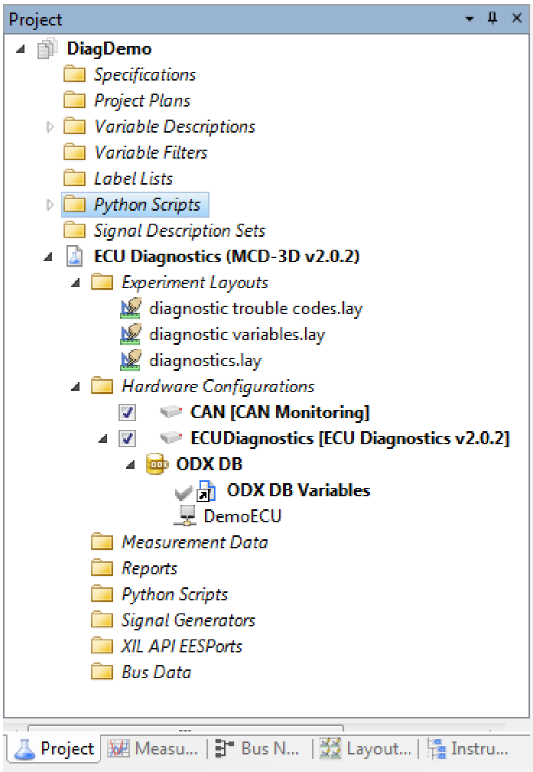
  The device is preconfigured to access the CalDemo ECU via CAN using a virtual CAN channel. ControlDesk starts the CalDemo ECU automatically when you open the DiagDemo project. For instructions on starting the CalDemo ECU manually, for example, if you closed it unintentionally, refer to Starting the CalDemo ECU (ControlDesk Introduction and Overview 🕮).

  ControlDesk automatically loads a demo ODX database to the device when you open the DiagDemo project. A variable description based on the ODX database is added to the device so you can perform measurement and calibration of diagnostic variables via the ECU Diagnostics device.

- *CAN Bus Monitoring* device

  The device is preconfigured to monitor CAN communication between ControlDesk and the CalDemo ECU.

The illustration below shows the demo setup in ControlDesk's Project controlbar:



Project

- DiagDemo
  - *Specifications*
  - *Project Plans*
  - *Variable Descriptions*
  - *Variable Filters*
  - *Label Lists*
  - *Python Scripts*
  - *Signal Description Sets*
  - **ECU Diagnostics (MCD-3D v2.0.2)**
    - *Experiment Layouts*
      - diagnostic trouble codes.lay
      - diagnostic variables.lay
      - diagnostics.lay
    - *Hardware Configurations*
      - ☑ **CAN [CAN Monitoring]**
      - ☑ **ECUDiagnostics [ECU Diagnostics v2.0.2]**
        - **ODX DB**
          - **ODX DB Variables**
          - DemoECU
    - *Measurement Data*
    - *Reports*
    - *Python Scripts*
    - *Signal Generators*
    - *XIL API EESPorts*
    - *Bus Data*

Project | Measu... | Bus N... | Layout... | Instru...

Demo layouts The *ECU Diagnostics (MCD-3D v2.0.2)* experiment contains the following layouts:

- *diagnostic trouble codes* layout

  Contains instruments with DTC measurement variables, and a Fault Memory Instrument. See Measuring diagnostic trouble codes (DTCs) on page 19.

- *diagnostic variables* layout

  Contains instruments to measure and calibrate diagnostics variables via the ECU Diagnostics device. See Measuring and calibrating variables via the ECU Diagnostics device on page 18.

- *diagnostics* layout

  Contains a Diagnostics Instrument for executing diagnostic jobs and services. See Executing diagnostic jobs on page 15 and Configuring and executing diagnostic services on page 16.

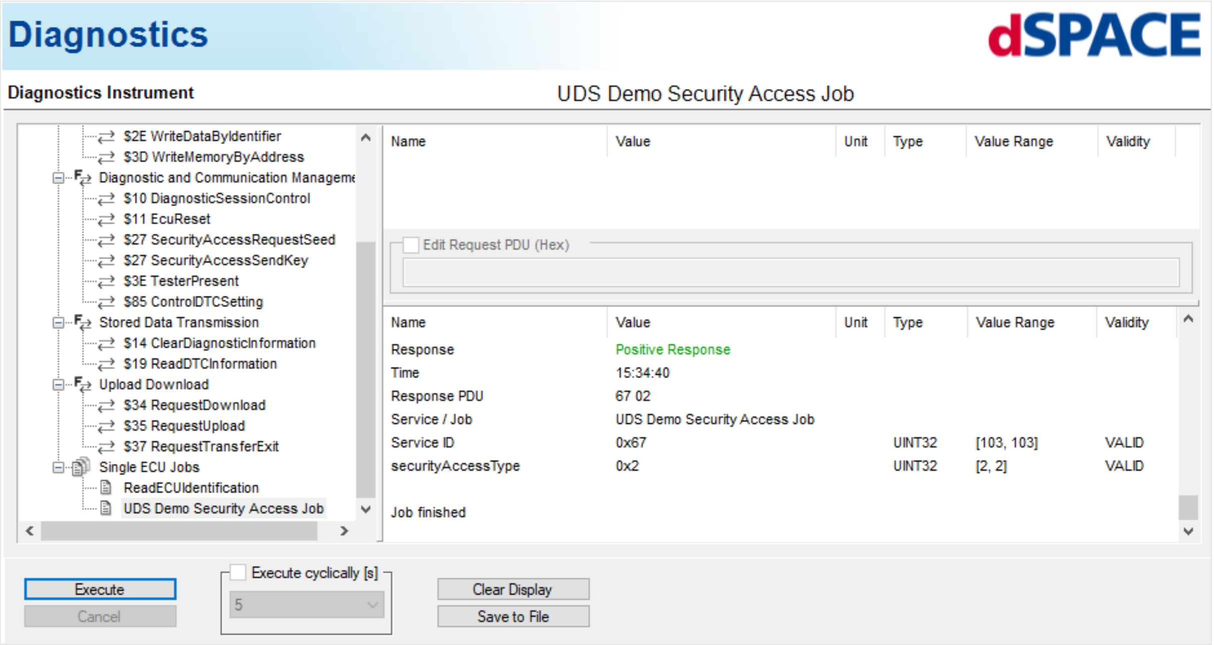
Executing diagnostic jobs

You can execute jobs with the Diagnostics Instrument. For example, you can execute the **UDS Demo Security Access Job** diagnostic job to enable security access, which is required to perform parameter changes on the CalDemo ECU via the ECU Diagnostics device.

> **Tip**
>
> The job is executed for demonstration purposes only. Normally, you do not have to execute it manually. ControlDesk executes it automatically when you start online calibration since this is necessary for write operations and access to protected areas.

To get security access by executing a diagnostic job, perform the following steps:

1. On the Home ribbon, click Status Control – Go Online to start online calibration. ControlDesk connects to the CalDemo ECU. You are now directly accessing the hardware.
2. In the Diagnostics Instrument on the diagnostics layout, select UDS Demo Security Access Job and click Execute.



Name	Value	Unit	Type	Value Range	Validity
Response	Positive Response				
Time	15:34:40				
Response PDU	67 02				
Service / Job	UDS Demo Security Access Job				
Service ID	0x67		UINT32	[103, 103]	VALID
securityAccessType	0x2		UINT32	[2, 2]	VALID

Job finished

The job performs the following steps:

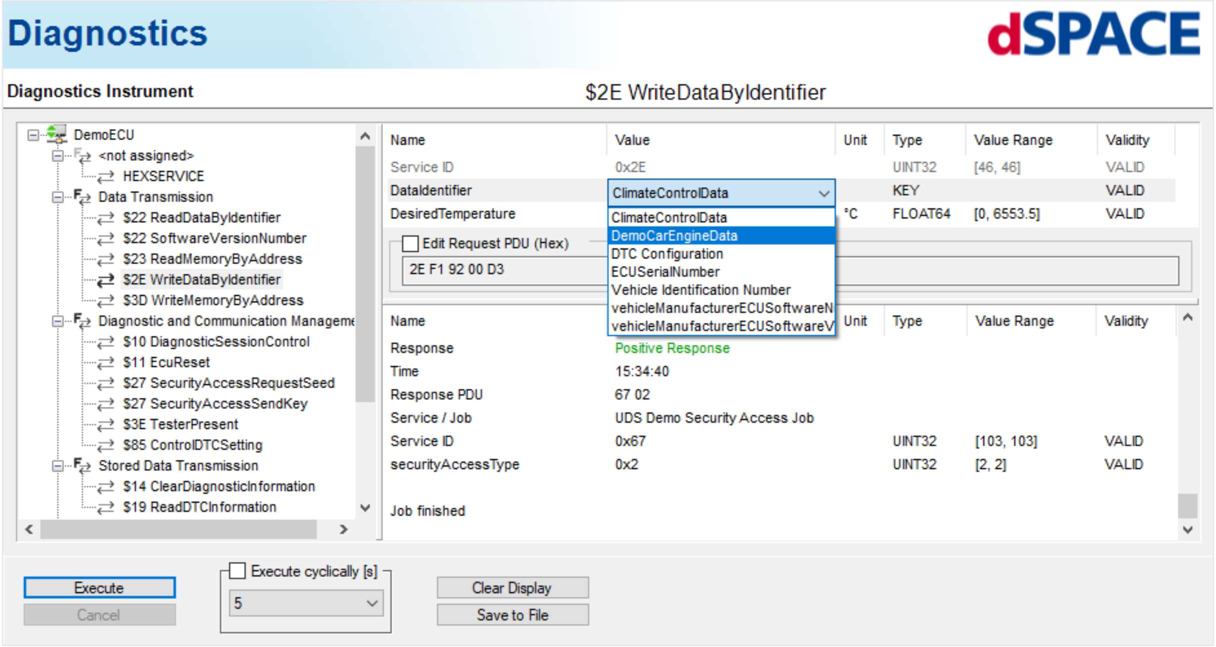
1. It executes the DiagnosticSessionControl service and changes to the programmingSession.
2. It executes the SecurityAccessRequestSeed service to request the seed value from the CalDemo ECU.
3. It executes the SecurityAccessSendKey service to send the key to the CalDemo ECU.

Configuring and executing diagnostic services

You can configure and execute services with the Diagnostics Instrument. For example, you can execute the WriteDataByIdentifier service to configure the current gear value.

To configure the gear value by executing a service, perform the following steps:

1. On the Home ribbon, click Status Control – Go Online to start online calibration. ControlDesk connects to the CalDemo ECU. You are now directly accessing the hardware.
2. In the Diagnostics Instrument on the diagnostics layout, select the WriteDataByIdentifier service.
3. Select DemoCarEngineData as the DataIdentifier.



Diagnostics

Diagnostics Instrument

$2E WriteDataByIdentifier

- DemoECU
  - <not assigned>
    - HEXSERVICE
  - Data Transmission
    - $22 ReadDataByIdentifier
    - $22 SoftwareVersionNumber
    - $23 ReadMemoryByAddress
    - $2E WriteDataByIdentifier
    - $3D WriteMemoryByAddress
  - Diagnostic and Communication Manageme
    - $10 DiagnosticSessionControl
    - $11 EcuReset
    - $27 SecurityAccessRequestSeed
    - $27 SecurityAccessSendKey
    - $3E TesterPresent
    - $85 ControlDTCSetting
  - Stored Data Transmission
    - $14 ClearDiagnosticInformation
    - $19 ReadDTCInformation

Name	Value	Unit	Type	Value Range	Validity
Service ID	0x2E		UINT32	[46, 46]	VALID
DataIdentifier	ClimateControlData		KEY		VALID
DesiredTemperature		°C	FLOAT64	[0, 6553.5]	VALID

ClimateControlData
DemoCarEngineData
DTC Configuration
ECUSerialNumber
Vehicle Identification Number
vehicleManufacturerECUSoftwareN
vehicleManufacturerECUSoftwareV

□ Edit Request PDU (Hex)

2E F1 92 00 D3

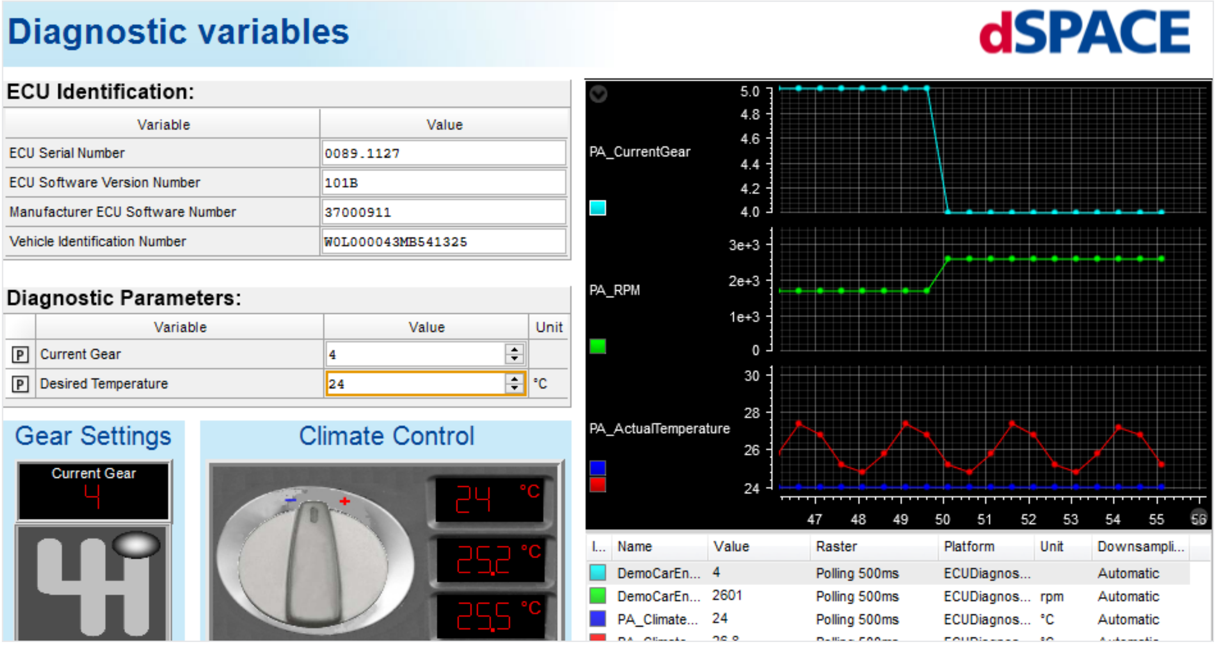
Name	Value	Unit	Type	Value Range	Validity
Response	Positive Response				
Time	15:34:40				
Response PDU	67 02				
Service / Job	UDS Demo Security Access Job				
Service ID	0x67		UINT32	[103, 103]	VALID
securityAccessType	0x2		UINT32	[2, 2]	VALID

Job finished

4. Specify **0x4** as the CurrentGear value.

5. Click Execute to execute the WriteDataByIdentifier service.

   The Time Plotter on the diagnostic variables layout displays the gear change:



Diagnostic variables

ECU Identification:

Variable	Value
ECU Serial Number	0089.1127
ECU Software Version Number	101B
Manufacturer ECU Software Number	37000911
Vehicle Identification Number	W0L000043MB541325

Diagnostic Parameters:

	Variable	Value	Unit
P	Current Gear	4	
P	Desired Temperature	24	°C

Gear Settings

Current Gear 4

Climate Control

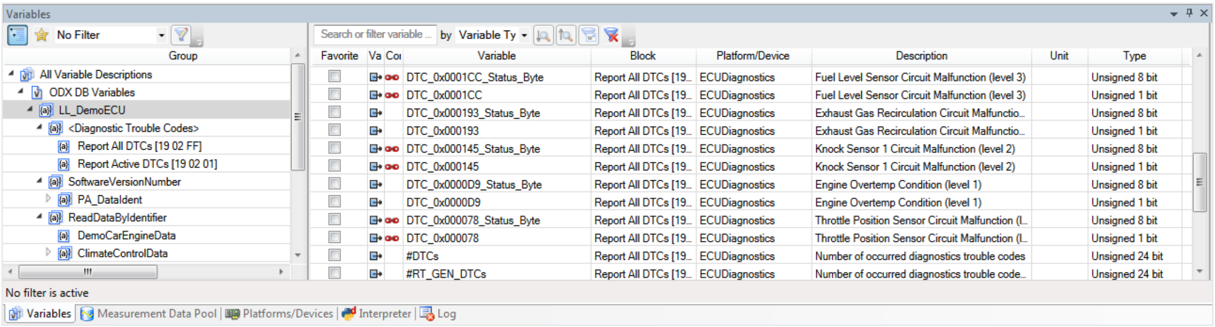
24 °C
25.2 °C
25.5 °C

I...	Name	Value	Raster	Platform	Unit	Downsampli...
🟦	DemoCarEn...	4	Polling 500ms	ECUDiagnos...		Automatic
🟩	DemoCarEn...	2601	Polling 500ms	ECUDiagnos...	rpm	Automatic
🟦	PA_Climate...	24	Polling 500ms	ECUDiagnos...	°C	Automatic

Measuring and calibrating variables via the ECU Diagnostics device

You can measure and calibrate diagnostics variables via the ECU Diagnostics device in connection with ControlDesk's standard instruments. Measurement and calibration via the ECU Diagnostics device uses diagnostic services available from the ODX database. When you calibrate a parameter value, for example, you implicitly configure and execute the service related to that parameter.

ControlDesk's Variables controlbar shows all the available *diagnostics variables*:



Variables

No Filter

Search or filter variable ... by Variable Ty

Group

- All Variable Descriptions
  - ODX DB Variables
    - LL_DemoECU
      - <Diagnostic Trouble Codes>
        - Report All DTCs [19 02 FF]
        - Report Active DTCs [19 02 01]
      - SoftwareVersionNumber
        - PA_DataIdent
      - ReadDataByIdentifier
        - DemoCarEngineData
        - ClimateControlData

Favorite	Va	Coı	Variable	Block	Platform/Device	Description	Unit	Type
☐			DTC_0x0001CC_Status_Byte	Report All DTCs [19...	ECUDiagnostics	Fuel Level Sensor Circuit Malfunction (level 3)		Unsigned 8 bit
☐			DTC_0x0001CC	Report All DTCs [19...	ECUDiagnostics	Fuel Level Sensor Circuit Malfunction (level 3)		Unsigned 1 bit
☐			DTC_0x000193_Status_Byte	Report All DTCs [19...	ECUDiagnostics	Exhaust Gas Recirculation Circuit Malfunctio...		Unsigned 8 bit
☐			DTC_0x000193	Report All DTCs [19...	ECUDiagnostics	Exhaust Gas Recirculation Circuit Malfunctio...		Unsigned 1 bit
☐			DTC_0x000145_Status_Byte	Report All DTCs [19...	ECUDiagnostics	Knock Sensor 1 Circuit Malfunction (level 2)		Unsigned 8 bit
☐			DTC_0x000145	Report All DTCs [19...	ECUDiagnostics	Knock Sensor 1 Circuit Malfunction (level 2)		Unsigned 1 bit
☐			DTC_0x0000D9_Status_Byte	Report All DTCs [19...	ECUDiagnostics	Engine Overtemp Condition (level 1)		Unsigned 8 bit
☐			DTC_0x0000D9	Report All DTCs [19...	ECUDiagnostics	Engine Overtemp Condition (level 1)		Unsigned 1 bit
☐			DTC_0x000078_Status_Byte	Report All DTCs [19...	ECUDiagnostics	Throttle Position Sensor Circuit Malfunction (l...		Unsigned 8 bit
☐			DTC_0x000078	Report All DTCs [19...	ECUDiagnostics	Throttle Position Sensor Circuit Malfunction (l...		Unsigned 1 bit
☐			#DTCs	Report All DTCs [19...	ECUDiagnostics	Number of occurred diagnostics trouble codes		Unsigned 24 bit
☐			#RT_GEN_DTCs	Report All DTCs [19...	ECUDiagnostics	Number of occurred diagnostics trouble code...		Unsigned 24 bit

No filter is active

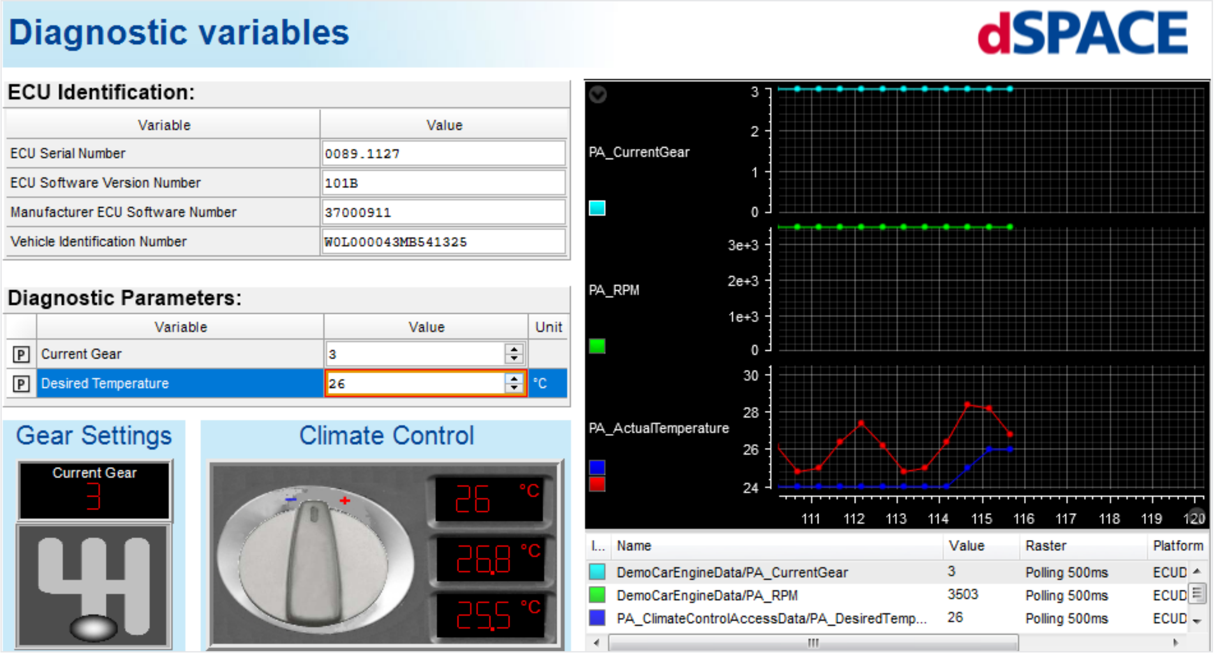
Perform the following steps:

1. On the Home ribbon, click Status Control – Start Measuring to start measuring.

   The Time Plotter on the diagnostics variables layout displays various variables that originate from the ECU Diagnostics device.

2. Use, for example, the Climate Control Multiswitch on the diagnostics variables layout to change the Desired Temperature parameter value.

   You can observe the effect of the changed parameter value in the Time Plotter.



Diagnostic variables

ECU Identification:

Variable	Value
ECU Serial Number	0089.1127
ECU Software Version Number	101B
Manufacturer ECU Software Number	37000911
Vehicle Identification Number	W0L000043MB541325

Diagnostic Parameters:

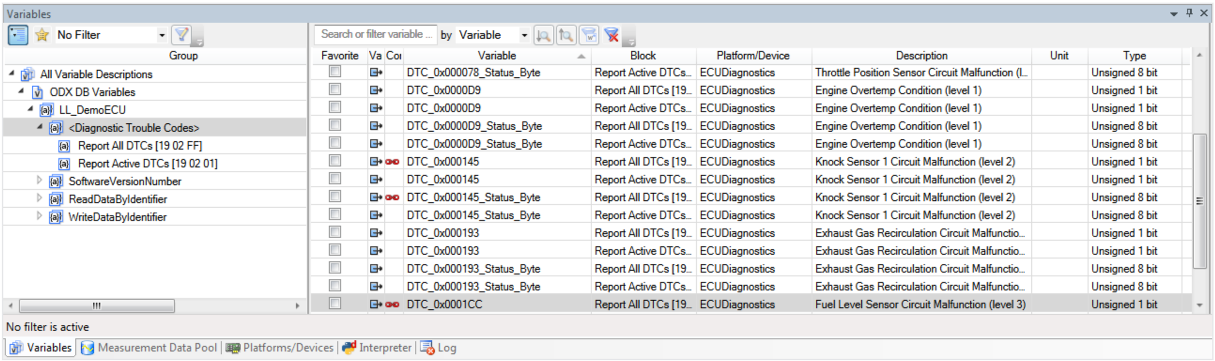
	Variable	Value	Unit
P	Current Gear	3	
P	Desired Temperature	26	°C

I...	Name	Value	Raster	Platform
🟦	DemoCarEngineData/PA_CurrentGear	3	Polling 500ms	ECUD
🟩	DemoCarEngineData/PA_RPM	3503	Polling 500ms	ECUD
🟦	PA_ClimateControlAccessData/PA_DesiredTemp...	26	Polling 500ms	ECUD

Measuring diagnostic trouble codes (DTCs)

The demo provides *DTC measurement variables* generated from the device's ODX database. DTC measurement variables make diagnostic trouble codes visible in ControlDesk instruments other than the Fault Memory and the Diagnostics Instrument.

ControlDesk's Variables controlbar shows all the available *DTC measurement variables* in the `<Diagnostic Trouble Codes>` node:



Variables

No Filter

Group
- All Variable Descriptions
  - ODX DB Variables
    - LL_DemoECU
      - <Diagnostic Trouble Codes>
        - Report All DTCs [19 02 FF]
        - Report Active DTCs [19 02 01]
      - SoftwareVersionNumber
      - ReadDataByIdentifier
      - WriteDataByIdentifier

Search or filter variable ... by Variable

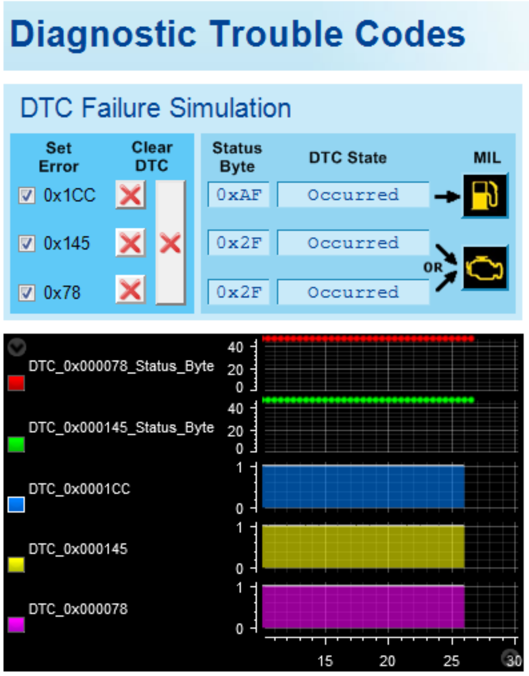
Favorite	Va	Co	Variable	Block	Platform/Device	Description	Unit	Type
□			DTC_0x000078_Status_Byte	Report Active DTCs...	ECUDiagnostics	Throttle Position Sensor Circuit Malfunction (l...		Unsigned 8 bit
□			DTC_0x0000D9	Report All DTCs [19...	ECUDiagnostics	Engine Overtemp Condition (level 1)		Unsigned 1 bit
□			DTC_0x0000D9	Report Active DTCs...	ECUDiagnostics	Engine Overtemp Condition (level 1)		Unsigned 1 bit
□			DTC_0x0000D9_Status_Byte	Report All DTCs [19...	ECUDiagnostics	Engine Overtemp Condition (level 1)		Unsigned 8 bit
□			DTC_0x0000D9_Status_Byte	Report Active DTCs...	ECUDiagnostics	Engine Overtemp Condition (level 1)		Unsigned 8 bit
□			DTC_0x000145	Report All DTCs [19...	ECUDiagnostics	Knock Sensor 1 Circuit Malfunction (level 2)		Unsigned 1 bit
□			DTC_0x000145	Report Active DTCs...	ECUDiagnostics	Knock Sensor 1 Circuit Malfunction (level 2)		Unsigned 1 bit
□			DTC_0x000145_Status_Byte	Report All DTCs [19...	ECUDiagnostics	Knock Sensor 1 Circuit Malfunction (level 2)		Unsigned 8 bit
□			DTC_0x000145_Status_Byte	Report Active DTCs...	ECUDiagnostics	Knock Sensor 1 Circuit Malfunction (level 2)		Unsigned 8 bit
□			DTC_0x000193	Report All DTCs [19...	ECUDiagnostics	Exhaust Gas Recirculation Circuit Malfunctio...		Unsigned 1 bit
□			DTC_0x000193	Report Active DTCs...	ECUDiagnostics	Exhaust Gas Recirculation Circuit Malfunctio...		Unsigned 1 bit
□			DTC_0x000193_Status_Byte	Report All DTCs [19...	ECUDiagnostics	Exhaust Gas Recirculation Circuit Malfunctio...		Unsigned 8 bit
□			DTC_0x000193_Status_Byte	Report Active DTCs...	ECUDiagnostics	Exhaust Gas Recirculation Circuit Malfunctio...		Unsigned 8 bit
□			DTC_0x0001CC	Report All DTCs [19...	ECUDiagnostics	Fuel Level Sensor Circuit Malfunction (level 3)		Unsigned 1 bit

No filter is active

To change the current DTC states (set and clear DTCs) for DTC measurements, perform the following steps:

1. On the Home ribbon, click Status Control – Start Measuring to start measuring.

The instruments on the diagnostic trouble codes layout display various DTC measurement variables.



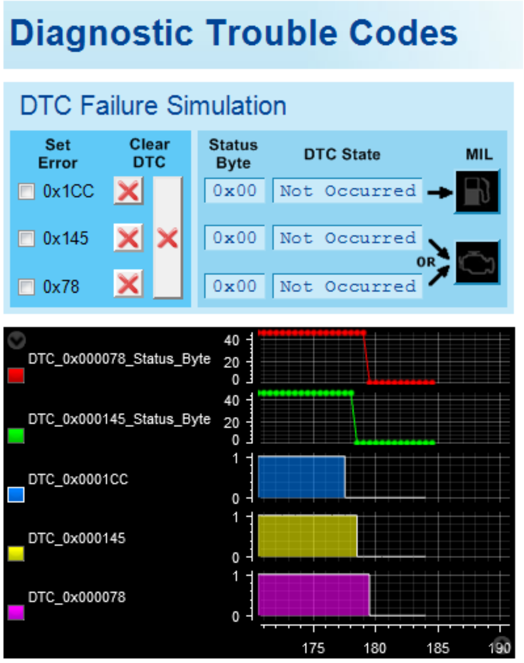
Set Error	Clear DTC		Status Byte	DTC State
☑ 0x1CC	☒		0xAF	Occurred
☑ 0x145	☒	☒	0x2F	Occurred
☑ 0x78	☒		0x2F	Occurred

2. Clear the checkboxes next to 0x1CC, 0x145 and 0x78.

   This deactivates the occurrence of the selected DTCs, i.e., sets the DTC states to 0.

3. Click ☒ next to the three DTCs.

   This clears the DTCs:



Diagnostic Trouble Codes

DTC Failure Simulation

Set Error	Clear DTC		Status Byte	DTC State	MIL
□ 0x1CC	☒		0x00	Not Occurred	
□ 0x145	☒	☒	0x00	Not Occurred	OR
□ 0x78	☒		0x00	Not Occurred	

Note

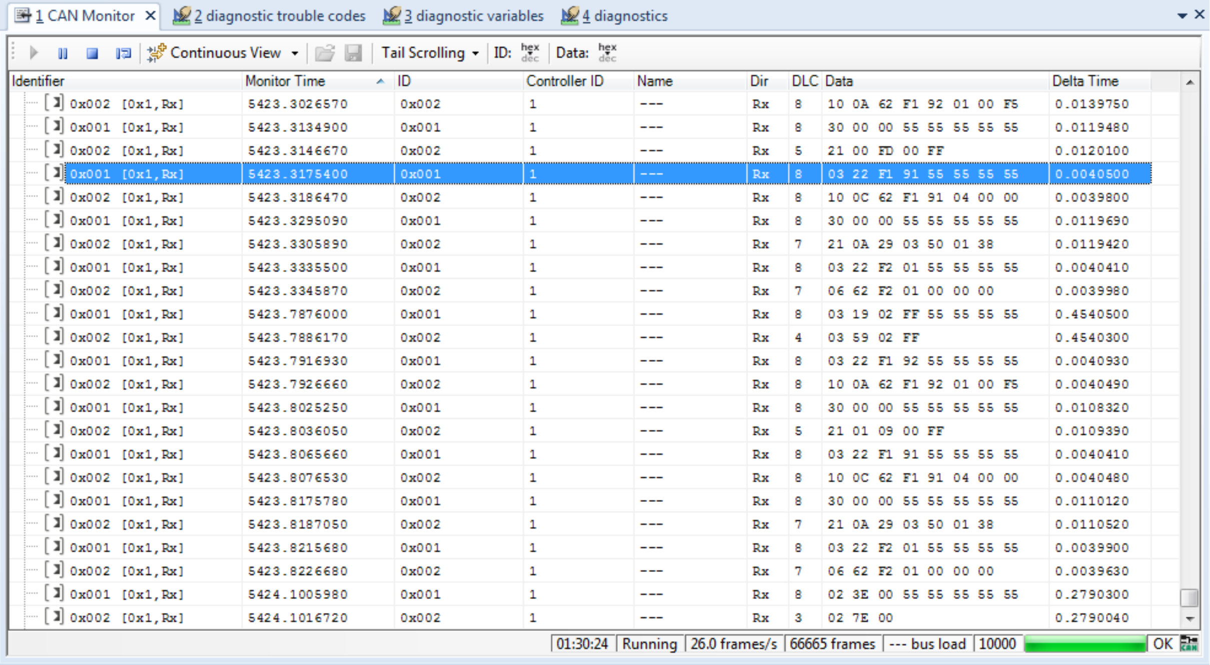
When you clear a DTC *without having deactivated its occurrence*, the DTC reoccurs immediately after you clear it.

Tip

The Fault Memory Instrument on the diagnostic trouble codes layout also displays when DTCs occurred. If you specify an Update Rate [s] value in the instrument, this information is updated cyclically.

Monitoring CAN communication

Via ControlDesk's *Bus Navigator*, you can monitor CAN communication using the CAN Bus Monitoring device. The CAN Monitor monitoring list displays the raw data of the monitored CAN communication:



Identifier	Monitor Time	ID	Controller ID	Name	Dir	DLC	Data	Delta Time
0x002 [0x1,Rx]	5423.3026570	0x002	1	---	Rx	8	10 0A 62 F1 92 01 00 F5	0.0139750
0x001 [0x1,Rx]	5423.3134900	0x001	1	---	Rx	8	30 00 00 55 55 55 55 55	0.0119480
0x002 [0x1,Rx]	5423.3146670	0x002	1	---	Rx	5	21 00 FD 00 FF	0.0120100
0x001 [0x1,Rx]	5423.3175400	0x001	1	---	Rx	8	03 22 F1 91 55 55 55 55	0.0040500
0x002 [0x1,Rx]	5423.3186470	0x002	1	---	Rx	8	10 0C 62 F1 91 04 00 00	0.0039800
0x001 [0x1,Rx]	5423.3295090	0x001	1	---	Rx	8	30 00 00 55 55 55 55 55	0.0119690
0x002 [0x1,Rx]	5423.3305890	0x002	1	---	Rx	7	21 0A 29 03 50 01 38	0.0119420
0x001 [0x1,Rx]	5423.3335500	0x001	1	---	Rx	8	03 22 F2 01 55 55 55 55	0.0040410
0x002 [0x1,Rx]	5423.3345870	0x002	1	---	Rx	7	06 62 F2 01 00 00 00	0.0039980
0x001 [0x1,Rx]	5423.7876000	0x001	1	---	Rx	8	03 19 02 FF 55 55 55 55	0.4540500
0x002 [0x1,Rx]	5423.7886170	0x002	1	---	Rx	4	03 59 02 FF	0.4540300
0x001 [0x1,Rx]	5423.7916930	0x001	1	---	Rx	8	03 22 F1 92 55 55 55 55	0.0040930
0x002 [0x1,Rx]	5423.7926660	0x002	1	---	Rx	8	10 0A 62 F1 92 01 00 F5	0.0040490
0x001 [0x1,Rx]	5423.8025250	0x001	1	---	Rx	8	30 00 00 55 55 55 55 55	0.0108320
0x002 [0x1,Rx]	5423.8036050	0x002	1	---	Rx	5	21 01 09 00 FF	0.0109390
0x001 [0x1,Rx]	5423.8065660	0x001	1	---	Rx	8	03 22 F1 91 55 55 55 55	0.0040410
0x002 [0x1,Rx]	5423.8076530	0x002	1	---	Rx	8	10 0C 62 F1 91 04 00 00	0.0040480
0x001 [0x1,Rx]	5423.8175780	0x001	1	---	Rx	8	30 00 00 55 55 55 55 55	0.0110120
0x002 [0x1,Rx]	5423.8187050	0x002	1	---	Rx	7	21 0A 29 03 50 01 38	0.0110520
0x001 [0x1,Rx]	5423.8215680	0x001	1	---	Rx	8	03 22 F2 01 55 55 55 55	0.0039900
0x002 [0x1,Rx]	5423.8226680	0x002	1	---	Rx	7	06 62 F2 01 00 00 00	0.0039630
0x001 [0x1,Rx]	5424.1005980	0x001	1	---	Rx	8	02 3E 00 55 55 55 55 55	0.2790300
0x002 [0x1,Rx]	5424.1016720	0x002	1	---	Rx	3	02 7E 00	0.2790040

01:30:24 | Running | 26.0 frames/s | 66665 frames | --- bus load | 10000 | OK

The monitored CAN messages originate from communication between the ECU Diagnostics device and the CalDemo ECU:

- CAN messages with the `0x001` ID are requests from the ECU Diagnostics device.
- CAN messages with the `0x002` ID are responses from the CalDemo ECU.

Related topics

Basics

CalDemo ECU Program (ControlDesk Introduction and Overview 🕮)

Conventions in Connection with ODX Databases

Introduction

Performing ECU diagnostics with ControlDesk requires a database compliant with Open Diagnostic Data Exchange (ODX). ControlDesk supports V2.0.1 and V2.2.0 (ISO 22901-1) of the ASAM MCD-2 D (ODX) standard. For proper operation, you have to observe some conventions laid down in the ODX standard.

Structure of the ODX database

The ODX database must contain at least:

- `COMPARAM-SPEC`
- `PROTOCOL`

Preselecting protocol and physical type from ODX database

ControlDesk supports the preselection of the diagnostic protocol and physical connection settings from the ODX database during the configuration of an ECU Diagnostics device if there is a corresponding entry in the ODX database. Depending on the ODX database standard used, the definition in the ODX standard differs:

- For an ODX 2.0.1 database, the `TYPE` attribute of the protocol layer is used.
- For an ODX 2.2.0 database, the `PDU-PROTOCOL-TYPE` in the `PROT-STACK` referenced by the `PROT-STACK-SNREF` specified for the protocol is used.

Preselecting the protocol and physical type from the ODX database is possible if the `TYPE` attribute or the `PDU-PROTOCOL-TYPE` communication parameter is set to one of the following values:

Protocol	Value
KWP2000 on CAN	`ISO_14230_3_on_ISO_15765_2`
KWP2000 on K-Line	`ISO_14230_3_on_ISO_14230_2`
OBD	`ISO_15031_5_on_ISO_15765_4`[1)]
UDS on CAN	`ISO_15765_3_on_ISO_15765_2`
UDS on DoIP	`ISO_14229_5_on_ISO_13400_2`

[1)] Or optionally: `ISO_OBD_on_ISO_15765_4`

Preselection is not possible for TP 1.6 and TP 2.0. The ODX standard does not contain predefined `TYPE` or `PDU-PROTOCOL-TYPE` values for TP 1.6 and TP 2.0.

Identifying communication parameters

In the ODX standard, the communication parameters (`COMPARAMs`) are defined in a separate data structure named `COMPARAM-SPEC`. The `COMPARAM-SPEC` depends on the diagnostic protocol used.

ControlDesk uses the short name of the parameter to identify a `COMPARAM`. ControlDesk supports the standard communication parameters defined in `D-PDU API` according to ISO 22900-2. `COMPARAMs` that are unknown to ControlDesk are marked.

Diagnostic services for ECU communication

Some diagnostic protocols require specific diagnostic services for establishing communication with the ECU. For example, KWP2000 on K-Line (ISO14230), TP 1.6 and TP 2.0 require the `startCommunication` and `stopCommunication` services to perform the ECU initialization.

startCommunication service To start communication with the ECU, ControlDesk handles the `startCommunication` service like this:

- For identification, the `startCommunication` service must be classified as `DIAGNOSTIC-CLASS = STARTCOMM`. This classification must be unique in the logical link concerned.
- ControlDesk uses this diagnostic service to start communication with the ECU. If this fails, the logical link remains in the 'disconnected' state.

Cyclic sending of TesterPresent messages is started by a service such as the `startCommunication` service. This is a prerequisite for transmitting services for many ECUs, or for keeping an ECU in a specific diagnostics session.

stopCommunication service To stop communication with the ECU, ControlDesk handles the `stopCommunication` service like this:

- For identification, the `stopCommunication` service must be classified as `DIAGNOSTIC-CLASS = STOPCOMM`. This classification must be unique in the logical link concerned.
- ControlDesk uses this diagnostic service to stop communication with the ECU.

Performing ECU connection checks

You can specify, individually for each active logical link, whether to execute a TesterPresent service and/or the `startCommunication` control primitive during the connection check, i.e., during the device state transition from 'disconnected' to 'connected' and during the reconnection check. You can also specify for each active logical link whether to execute a TesterPresent service to check cyclically whether ECU communication is available after the device is connected. ControlDesk lets you specify alternative TesterPresent configurations for single logical links. Refer to Active Logical Links Dialog (ControlDesk Platform Management 🕮).

Note

If the specific diagnostic services needed for the ECU connection check during the device state transition from 'disconnected' to 'connected' are not defined in the ODX database, ControlDesk switches the logical link to the 'connected' state without checking the connection.

If you use the UDS diagnostic protocol, ControlDesk evaluates the `SuppressPositiveResponseBit` flag when executing the services used by `startCommunication` and `stopCommunication`. If the flag is set, an RX error is interpreted as a positive response. However, if a positive response is sent, it is also accepted (even if it is defined as an error in the UDS specification). All other errors except the RX error are treated as usual and cause the connection check to terminate.

Configuring COMPARAMs dynamically

ControlDesk lets you configure communication parameters (`COMPARAMs`) dynamically. This allows you, for example, to enable/disable the cyclic transmission of `TesterPresent` indications to send keep-alive messages, or to change timings (like timeout for response time) to check the ECU behavior.

`COMPARAMs` can be configured in two ways:

- Dynamic configuration of `COMPARAMs` can be done via the **Properties** controlbar. For each active logical link, the relevant communication parameters are displayed with their original values specified in the ODX database, their user-defined values and their currently active values. You can specify user-defined values for single communication parameters, and switch between the original ODX values and the user-defined values. Refer to Logical Link Selection Properties (ControlDesk Platform Management 🕮).

- Dynamic configuration of `COMPARAMs` can be done via the `ProtocolParameterSet` control primitive in the Diagnostics Instrument. After executing the `ProtocolParameterSet` control primitive, the result is displayed in the output field of the Diagnostics Instrument. In case of errors, a negative response is displayed, and the request values of `COMPARAMs` that could not be changed are not reset.

  Before dynamic `COMPARAM` configuration can be used, make sure that the `ProtocolParameterSet` control primitive is activated in the Diagnostics Instrument. This is done via the instrument's Properties controlbar. Refer to Tree View Properties (ControlDesk Instrument Handling 🕮).

  Note

  For logical links whose Physical Connection is set to 'Simulation', dynamic `COMPARAM` configuration is always impossible.

Mandatory communication parameters when using a CAN FD bus

If a CAN FD bus is used, the ODX database of the ECU Diagnostics device must contain CAN FD-specific communication parameters (`COMPARAM`s) so that the CAN interface is initialized with CAN FD. This also applies if the device is to transmit CAN 2.0 (classic CAN) messages on a CAN FD bus.

The following CAN FD-specific `COMPARAM`s are mandatory in the ODX database to initialize the CAN interface with CAN FD:

- `CANFDBaudrate`
- `CANFDBitSamplePoint`
- `CANFDSyncJumpWidth`
- `CANFDTxMaxDataLength`

Depending on the `COMPARAM` settings, the ECU Diagnostics device transmits CAN 2.0 messages or CAN FD messages with or without bit rate switch (BRS) flag:

`CANFDTxMaxDataLength` Setting	**`CANFDBaudrate` Setting**		
	= 0	**≠ 0**	**- (Unavailable)**
Classic CAN	CAN 2.0 (classic CAN)	CAN 2.0 (classic CAN)	CAN 2.0 (classic CAN)
CANFD	CAN FD without BRS flag CAN FD messages are transmitted with the CAN FD arbitration baud rate defined by the `Baudrate` COMPARAM since the BRS is disabled.	CAN FD with BRS flag CAN FD messages are transmitted with the CAN FD data baud rate defined by the	- (no message transmission)

CANFDTxMaxDataLength Setting	CANFDBaudrate Setting		
	= 0	≠ 0	- (Unavailable)
		CANFDBaudrate COMPARAM since the BRS is enabled. **Note** With the ControlDesk ECU Diagnostics device, you cannot transmit CAN FD messages without BRS flag if the CANFDBaudrate is ≠ 0. This means that message transmission at the arbitration rate is not possible in this case.	

If the CAN FD-specific COMPARAMs are not specified in the ODX database, the CAN interface is initialized with CAN 2.0 (classic CAN).

Tip

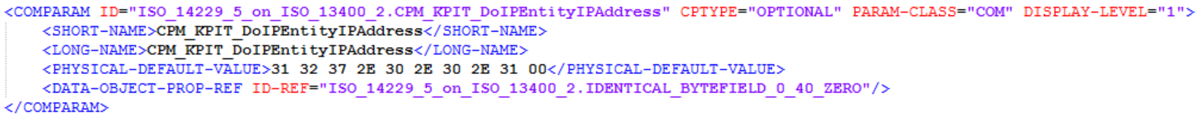
The built-in ODX database template for UDS, which is provided by ControlDesk, supports CAN FD.

Specifics of the DoIP protocol

DoIP (Diagnostics over Internet Protocol) is a transport protocol for UDS, which runs over networks based on TCP/IP. There are some specifics to note when using the DoIP protocol. For example, the IP address must be specified in a proprietary communication parameter in a certain format in the ODX database.

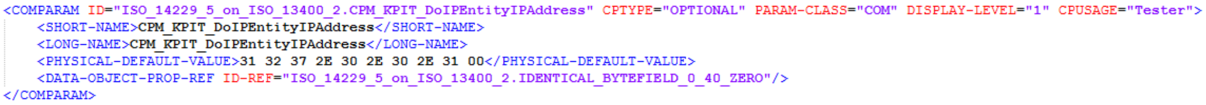
Mandatory communication parameter to specify the IP address To provide the IP address, which is needed for automatic assignment of an Ethernet interface of the host PC to a logical link, the ODX database must contain the CPM_KPIT_DoIPEntityIPAddress communication parameter (short name). The communication parameter must be of the BYTEFIELD type with a maximum length of 40, and the CPTYPE parameter must be set to 'OPTIONAL'. In the BYTEFIELD, the IP address must be specified in hexadecimal representation of ASCII characters with terminating NULL.

The following illustration shows an example of an IP address specification for ODX 2.0.1 databases:



```
<COMPARAM ID="ISO_14229_5_on_ISO_13400_2.CPM_KPIT_DoIPEntityIPAddress" CPTYPE="OPTIONAL" PARAM-CLASS="COM" DISPLAY-LEVEL="1">
    <SHORT-NAME>CPM_KPIT_DoIPEntityIPAddress</SHORT-NAME>
    <LONG-NAME>CPM_KPIT_DoIPEntityIPAddress</LONG-NAME>
    <PHYSICAL-DEFAULT-VALUE>31 32 37 2E 30 2E 30 2E 31 00</PHYSICAL-DEFAULT-VALUE>
    <DATA-OBJECT-PROP-REF ID-REF="ISO_14229_5_on_ISO_13400_2.IDENTICAL_BYTEFIELD_0_40_ZERO"/>
</COMPARAM>
```

For ODX 2.2.0 databases, the CPUSAGE parameter must be set as well as shown in the following illustration:

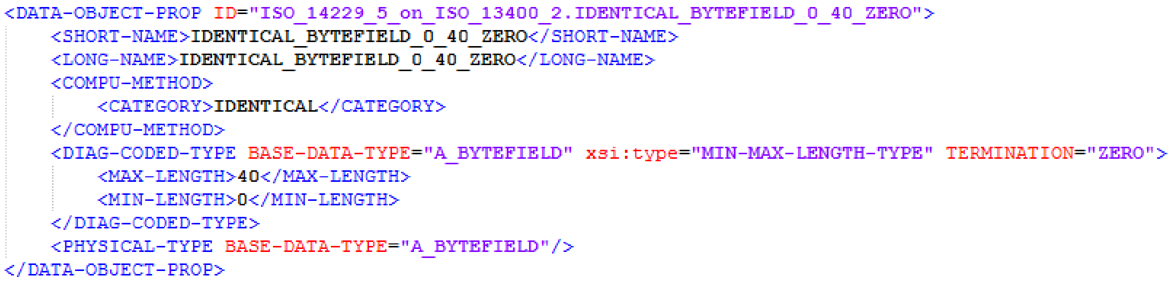


```
<COMPARAM ID="ISO_14229_5_on_ISO_13400_2.CPM_KPIT_DoIPEntityIPAddress" CPTYPE="OPTIONAL" PARAM-CLASS="COM" DISPLAY-LEVEL="1" CPUSAGE="Tester">
    <SHORT-NAME>CPM_KPIT_DoIPEntityIPAddress</SHORT-NAME>
    <LONG-NAME>CPM_KPIT_DoIPEntityIPAddress</LONG-NAME>
    <PHYSICAL-DEFAULT-VALUE>31 32 37 2E 30 2E 30 2E 31 00</PHYSICAL-DEFAULT-VALUE>
    <DATA-OBJECT-PROP-REF ID-REF="ISO_14229_5_on_ISO_13400_2.IDENTICAL_BYTEFIELD_0_40_ZERO"/>
</COMPARAM>
```

In the examples, the IP address 127.0.0.1 is specified. You can see the code translations from the hexadecimal numbers to the ASCII equivalents in the following table:

Numerical Value (hex)	31	32	37	2E	30	2E	30	2E	31	00
ASCII Character	1	2	7	.	0	.	0	.	1	NUL

The description of the corresponding BYTEFIELD data object is shown in the following illustration:



```
<DATA-OBJECT-PROP ID="ISO_14229_5_on_ISO_13400_2.IDENTICAL_BYTEFIELD_0_40_ZERO">
    <SHORT-NAME>IDENTICAL_BYTEFIELD_0_40_ZERO</SHORT-NAME>
    <LONG-NAME>IDENTICAL_BYTEFIELD_0_40_ZERO</LONG-NAME>
    <COMPU-METHOD>
        <CATEGORY>IDENTICAL</CATEGORY>
    </COMPU-METHOD>
    <DIAG-CODED-TYPE BASE-DATA-TYPE="A_BYTEFIELD" xsi:type="MIN-MAX-LENGTH-TYPE" TERMINATION="ZERO">
        <MAX-LENGTH>40</MAX-LENGTH>
        <MIN-LENGTH>0</MIN-LENGTH>
    </DIAG-CODED-TYPE>
    <PHYSICAL-TYPE BASE-DATA-TYPE="A_BYTEFIELD"/>
</DATA-OBJECT-PROP>
```

No automatic connection re-establishment According to ISO 14229-5, with an ECU reset and in some cases with a session switch, the Ethernet connection is lost. ControlDesk does not re-establish the connection automatically, but you must re-establish it manually by disconnecting and subsequently connecting the ECU Diagnostics device. To do so, first select Disconnect Platform/Device and then select Connect Platform/Device from the device's context menu in the Project controlbar.

Automatic reconnect functionality

ControlDesk supports the automatic reconnect functionality for the ECU Diagnostics device. A logical link enters the 'unplugged' state if the logical connection between ControlDesk and the logical link's hardware is lost. When the logical link is reconnected to the hardware, the logical link recovers the state ('connected', 'online calibration started' or 'measuring') it had before the unplugged state was detected.

An ECU Diagnostics device enters the 'unplugged' state if the logical connection between ControlDesk and the device hardware is lost for *all* the logical links specified in the device.

For measurement and calibration use cases, the specific diagnostic job classified as `SEMANTIC = SECURITY` can be executed to resume online calibration and continue measuring, if it is available in the ODX database. You can specify ControlDesk's behavior regarding job execution for security access.

Note

The cyclic diagnostic services for reading and displaying fault memory entries in the Fault Memory Instrument may not be restarted automatically when the ECU Diagnostics device is reconnected to the hardware.

If the automatic reconnect feature is enabled for the ECU Diagnostics device, ControlDesk uses the same settings as specified in the Connection check properties for the connection check that is to be performed during the device state transition from 'disconnected' to 'connected'. Refer to Active Logical Links Dialog (ControlDesk Platform Management 📖).

Note

- Disabling the cyclic connection state check also disables the automatic reconnect functionality for the ECU Diagnostics device.
- You can disable the automatic reconnect functionality for the ECU Diagnostics device via the **Properties** controlbar. Refer to General Settings Properties (ControlDesk Platform Management 📖).

Reading fault memory

The fault memory of an ECU stores diagnostic trouble code entries. ControlDesk's Fault Memory Instrument can read and display these entries.

The ODX database must meet some requirements so that ControlDesk can read and display fault memory entries. Refer to Reading the Fault Memory of an ECU on page 56.

You can also perform DTC measurements. DTC measurements allow you to display diagnostic trouble codes in ControlDesk instruments other than the Fault Memory and the Diagnostics Instrument. DTC measurements can also be used as trigger variables. Refer to Performing DTC Measurements on page 81.

Reading environment data

ControlDesk's Fault Memory Instrument can display additional environment data for each DTC.

The ODX database must meet some requirements so that ControlDesk can read and display the environment data that corresponds to a specific fault memory entry. Refer to Reading Environment Data on page 57.

Clearing fault memory entries

With the Fault Memory Instrument, you can delete entries in the ECU's fault memory. ControlDesk lets you clear an ECU fault memory of all DTCs referenced by the logical links, or delete a single DTC.

The ODX database must meet some requirements so that ControlDesk can clear fault memory entries. Refer to Clearing Fault Memory Entries on page 60.

Performing measurement and calibration via the ECU Diagnostics device

You can measure and calibrate diagnostics variables via the ECU Diagnostics device. This requires that a variable description is generated from the ODX database specified for the device.

The ODX database must meet some requirements so that ControlDesk can create a variable description from it. Refer to Measuring and Calibrating Variables via the ECU Diagnostics Device on page 76.

Related topics

Basics

How to Place an Instrument for Diagnostic Tasks on a Layout

Objective

In ControlDesk you use special instruments to handle the ECU's fault memory or perform diagnostic communication.

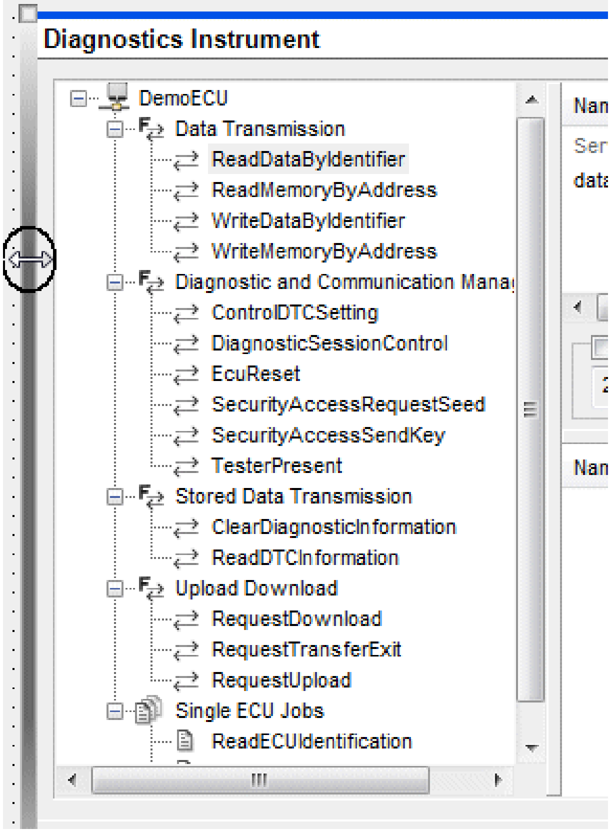
Method

To place an instrument for diagnostic tasks on a layout

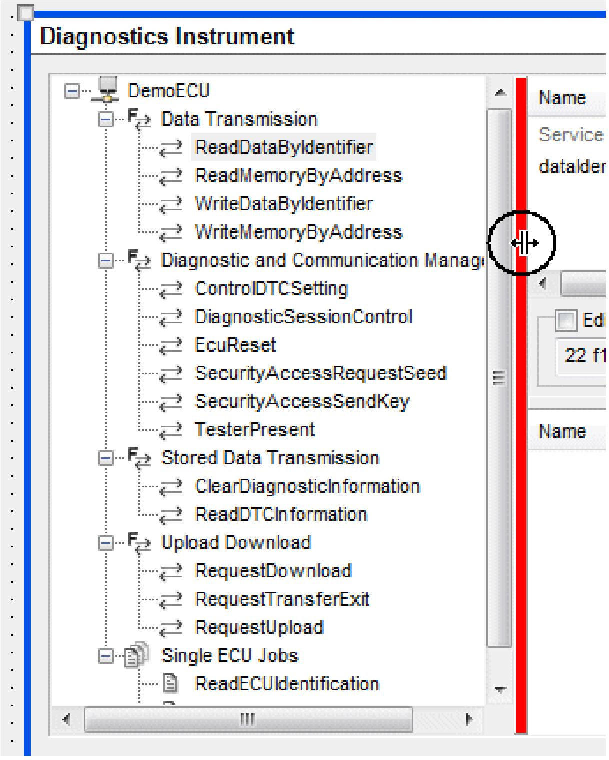
1. On the Layouting ribbon, click Layouts – Insert Layout to create a layout or select an existing layout.
2. In the Instrument Selector, double-click the Fault Memory Instrument or Diagnostics Instrument.

   The selected instrument is placed on the layout. If the experiment does not yet contain an ECU Diagnostics device, the instrument is empty.

3 Configure the size and position of the instrument by dragging the whole instrument or its borders.



4 Configure the size of instrument parts by dragging the splitters. Double-click a splitter to toggle its direction (horizontal – vertical).



Diagnostics Instrument

- DemoECU
  - Data Transmission
    - ReadDataByIdentifier
    - ReadMemoryByAddress
    - WriteDataByIdentifier
    - WriteMemoryByAddress
  - Diagnostic and Communication Manag
    - ControlDTCSetting
    - DiagnosticSessionControl
    - EcuReset
    - SecurityAccessRequestSeed
    - SecurityAccessSendKey
    - TesterPresent
  - Stored Data Transmission
    - ClearDiagnosticInformation
    - ReadDTCInformation
  - Upload Download
    - RequestDownload
    - RequestTransferExit
    - RequestUpload
  - Single ECU Jobs
    - ReadECUIdentification

Result You have added an instrument for diagnostic tasks to a layout.

Related topics

Basics

Instruments for Diagnostic Tasks.. 131
Performing ECU Diagnostic Tasks.. 55

ECU Diagnostics Device Configuration

Where to go from here

Information in this section

Basics of the ECU Diagnostics Device

Introduction

With ControlDesk's ECU Diagnostics device, you can access an ECU for diagnostics purposes, and to program a new software revision or new calibration data to its flash memory. ECU access is via the diagnostic interface.

ECU Diagnostics device

ControlDesk provides the *ECU Diagnostics v2.0.2* device, which supports the ASAM MCD-3 D V2.0.2 standard.

ControlDesk supports the following ODX database standards:

- ASAM MCD-2 D V2.0.1
- ASAM MCD-2 D V2.2.0 (ISO 22901-1)

ECU diagnostics based on ODX

ECU diagnostics with ControlDesk are completely based on Open Diagnostic Data Exchange (ODX), the ASAM MCD-2 D (ISO 22901) diagnostics standard. The ODX database contains the description of the diagnostic services implemented on the ECU, and the settings for communication between ControlDesk and the ECU.

ODX database Abbreviation of Open Diagnostic Data Exchange, a diagnostics database that is the central ECU description for working with an ECU Diagnostics device in ControlDesk. The ODX database contains all the information required to perform diagnostic communication between

ControlDesk and a specific ECU or set of ECUs in a vehicle network. ControlDesk expects the database to be compliant with ASAM MCD-2 D (ODX).

Vehicle information The ODX database can contain information for one or more vehicles. Vehicle information data is used for vehicle identification purposes and for access to vehicles. It references the access paths (logical links) to the ECUs.

Logical link A representation of an ECU specified in the diagnostics database. A logical link contains information on the ECU itself, and all the information required for accessing it, such as the diagnostic protocol used for communication between the ECU and ControlDesk. Each logical link is represented by a unique short name in the ODX database.

Working with several ODX databases

If you work with an ECU Diagnostics device, you can specify several ODX diagnostics databases for it and switch between them. This means you can manage multiple diagnostics databases for one device in ControlDesk, but only one of them is active at a time. In the **Project** controlbar, all the diagnostics databases used in an experiment are displayed below the ECU Diagnostics device. The currently active database is displayed in bold letters with a ✓ODX or ✓BIN symbol (depending on whether it has the ODX data or the binary format) next to it. You can activate a diagnostics database via the **Project** controlbar or the **Properties** controlbar.

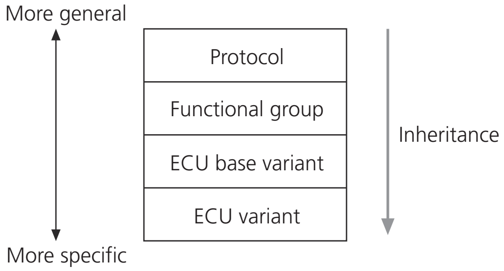
Working with several ODX databases enables you to check a further ODX diagnostics database within an existing experiment without having to modify or copy the experiment. After a successful test, you can easily make the new diagnostics database the only diagnostics database for the device by removing the other databases.

Hierarchical layers of an ODX database

An ODX database contains several layers that are defined in the ASAM MCD-2 D standard:

Layer	Description
Protocol	Describes the information that is usually defined in a vehicle manufacturer's protocol specification or international standard (for example, KWP2000). This layer typically contains all the information that is used by all a vehicle manufacturer's ECUs.
Functional group	Describes the information that is relevant for a group of ECUs with similar functionality. This layer typically contains all the information that is used by a group of a vehicle manufacturer's ECUs.
ECU base variant	Contains data for all the variants of a specific ECU.
ECU variant	Contains data for one specific variant of an ECU.

Inheritance To avoid data redundancy, the layers are structured hierarchically and an inheritance mechanism is supported:



- A functional group inherits data from one or more protocols.
- An ECU base variant can inherit data from one or more protocols and/or one or more functional groups.
- An ECU variant inherits data exclusively from one ECU base variant.

Supported diagnostic protocols

ControlDesk's ECU Diagnostics device communicates with ECUs connected to the ControlDesk PC via diagnostic protocols implemented on the ECUs. Communication is via CAN, K-Line, or Ethernet. ECU access is possible via the following diagnostic protocols:

Diagnostic Protocol	Description
Diagnostics on CAN (ISO 15765)	Diagnostics on CAN / KWP2000 on CAN
KWP2000 on K-Line (ISO 14230)	KWP2000 on K-Line
OBD	On-Board Diagnostics for CAN-based ECU diagnostics
TP 1.6	Transport Protocol TP 1.6 for CAN-based ECU diagnostics
TP 2.0	Transport Protocol TP 2.0 for CAN-based ECU diagnostics
UDS (ISO 14229)	UDS on CAN / UDS on DoIP (Diagnostics over Internet Protocol)

Related topics

Basics

Requirements for Measurement and Calibration via the ECU Diagnostics Device...................... 35

HowTos

How to Configure an ECU Diagnostics Device.. 36

Requirements for Measurement and Calibration via the ECU Diagnostics Device

Introduction

You can measure and calibrate diagnostics variables via the ECU Diagnostics device. This requires that a variable description is generated from the ODX database specified for the device.

Variable description generated from the ODX database

You can enable or disable the generation of a variable description from the ODX database during device configuration via the **Create variable description for selected logical links** checkbox in the **Select Logical Links** dialog, or via the **Properties** controlbar (refer to Logical Link Selection Properties (ControlDesk Platform Management)).

The variable description is saved in the Variable Descriptions folder in the **Project** controlbar. The generated variable description is named after the ODX database, followed by "Variables".

The ODX database must meet some requirements so that ControlDesk can create a variable description from it. For details, refer to Conventions in Connection with ODX Databases on page 22.

Removal or replacement of a generated variable description

Cases in which a generated variable description is removed or replaced A previously generated variable description is removed from the project or replaced by a new variable description in the following cases:

- If generation of the variable description is enabled in the **Select Logical Links** dialog, the following actions cause the generation of a new variable description:
  - Modifying an ODX database (adding, replacing, or removing files)
  - Adding an XML configuration file for variable generation to the ODX database configuration for your device, or replacing or removing the XML configuration file
  - Selecting another vehicle
  - Selecting other or deselecting logical links
  - Modifying the configuration settings for generating the variable description from the ODX database (selecting or clearing the **Use additional protocol information**, **Generate Diagnostic Trouble Code variables**, **Add**

additional measurements for parameters or Use ECU base variant name instead of ECU variant name checkbox)
  - Selecting another diagnostic protocol for an active logical link, if the Use additional protocol information and/or Generate Diagnostic Trouble Code variables options are enabled

  The previously generated variable description is replaced by the new one. The old variable description is removed from the project. Because you have performed an action which causes the removal of an existing variable description, ControlDesk displays a warning.
- If you clear the Create variable description for selected logical links checkbox while reconfiguring the ECU Diagnostics device, a previously generated variable description is removed from the project. ControlDesk displays a warning beforehand.
- You can add multiple diagnostics databases to one ECU Diagnostics device. You can generate a variable description from each individual diagnostics database. Since only one diagnostics database, and therefore at most one variable description, can be active for an ECU Diagnostics device at a time, only the settings from the currently active diagnostics database apply to the project. If you switch to another diagnostics database, the previously active variable description is deactivated. If a variable description was generated from the newly selected diagnostics database, it is activated.

Consequences of removing or replacing of a generated variable description ControlDesk removes or replaces a generated variable description in some cases, which has the following consequences:

- Depending on the size of the ODX data, regenerating the variable description can take some time.
- Data sets created for the removed or replaced variable description are lost.
- After a variable description is replaced due to a modification of the ODX database, connections between variables of the newly generated variable description and instruments are lost if the paths and/or names of the variables have changed.

Related topics

Basics

Basics of the ECU Diagnostics Device........32
Measuring and Calibrating Variables via the ECU Diagnostics Device........76

How to Configure an ECU Diagnostics Device

Objective

ControlDesk can use the ECU Diagnostics device for accessing an ECU via CAN, K-Line, or Ethernet for ECU diagnostics purposes only if the device is configured correctly.

Renaming or modifying the ODX database

If you modify (add, replace or remove) the files of an ODX database, the ODX database is automatically reloaded, its proprietary binary format (if specified) is regenerated, and the variable description (if enabled) is regenerated.

When reloading a database, ControlDesk checks whether the following configuration settings are still valid for the reloaded database:

- Selected vehicle
- Selected logical link(s)
- Configuration of the diagnostic protocol, physical connection, and hardware settings for each selected logical link

Validity is checked hierarchically: A logical link can be valid only if the corresponding vehicle information is still valid. The diagnostic protocol and physical connection settings can be valid only if the corresponding logical link is still valid. The hardware interface settings can be valid only if the corresponding logical link, diagnostic protocol and physical connection are still valid. Settings from the previous configuration that cannot be applied are reset, and you must reconfigure them via the configuration dialogs.

Variable description generated from the ODX database

You can measure and calibrate diagnostics variables via the ECU Diagnostics device. A variable description must be generated from the ODX database specified for the device for this purpose. For further information, refer to Requirements for Measurement and Calibration via the ECU Diagnostics Device on page 35.

Preconditions

- Add the device to the experiment first. Refer to How to Add a Platform/Device to an Experiment (ControlDesk Platform Management).
- You must have a database compliant with ASAM MCD-2 D V2.0.1 or V2.2.0 (ISO 22901). For further information, refer to Conventions in Connection with ODX Databases on page 22.

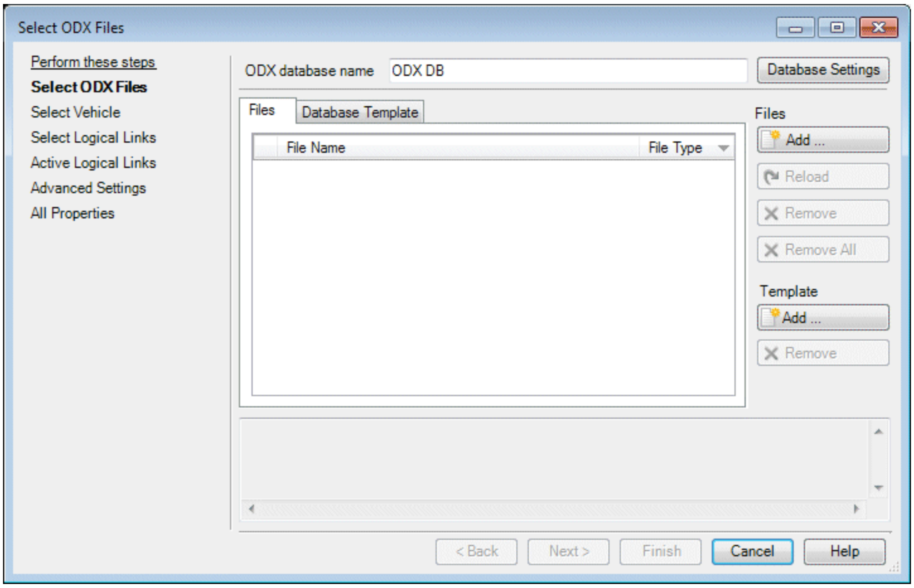
Method

To configure an ECU Diagnostics device

1 Select the ECU Diagnostics device in the **Project** controlbar.

2 From the context menu of the device, select Configure Platform/Device.

The Configure Platform/Device wizard opens, starting with the **Select ODX Files** dialog.



ODX database name: ODX DB

3 Enter a name for the ODX diagnostics database, if desired. If you do not enter a name, ControlDesk uses a default name for the database.

Tip

You can specify several diagnostics databases for an ECU Diagnostics device. You should then change their default names for clarity and better handling.

4 Specify the diagnostics database for the device by importing the files belonging to it. All the files imported so far are displayed in the files list. The list represents the diagnostics database the ECU Diagnostics device works with.

Tip

As an alternative, you can add an ODX database template. An ODX database template is a valid ODX database with fixed contents that cannot be modified. ControlDesk provides built-in ODX database templates for different diagnostic protocols and also supports custom ODX database templates. Built-in templates let you access your ECU via the ECU Diagnostics device without having to provide an ODX database yourself. Custom templates let you use an ODX database that is available on your file system.

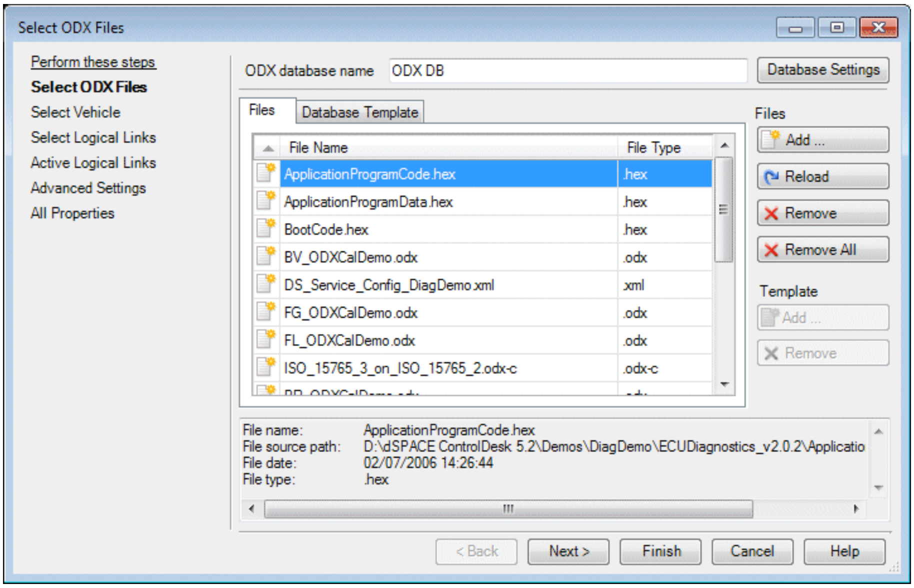
The built-in ODX database templates make no claim to completeness. They can be used for commissioning, for example, but are not suitable for release tests.

For further information on adding ODX database templates, refer to Select ODX Files Dialog (ControlDesk Platform Management 📖).

ControlDesk supports different ODX versions. However, the diagnostics database for your device must contain only files belonging to one of the supported ODX version. If you import files of different or unsupported ODX versions to the diagnostics database, an error message is displayed when you click **Next** or **Finish**.

Tip

ControlDesk can detect the ODX version automatically using the files you imported for the diagnostics database for your ECU Diagnostics device. Alternatively, you can specify the ODX version to be used with your ECU Diagnostics device manually. Refer to Select ODX Files Dialog (ControlDesk Platform Management 🕮).



Select ODX Files

ODX database name: ODX DB

File Name	File Type
ApplicationProgramCode.hex	.hex
ApplicationProgramData.hex	.hex
BootCode.hex	.hex
BV_ODXCalDemo.odx	.odx
DS_Service_Config_DiagDemo.xml	.xml
FG_ODXCalDemo.odx	.odx
FL_ODXCalDemo.odx	.odx
ISO_15765_3_on_ISO_15765_2.odx-c	.odx-c

File name: ApplicationProgramCode.hex
File source path: D:\dSPACE ControlDesk 5.2\Demos\DiagDemo\ECUDiagnostics_v2.0.2\Applicatio
File date: 02/07/2006 14:26:44
File type: .hex

You can add further files to the database via the **Add Files** button.

To remove one or more files from the diagnostics database, select them from the files list and click **Remove** or press the `Delete` key. To remove all the files from the diagnostics database in one step, click **Remove All**.

If you want ControlDesk to reload files in the diagnostics database, select them from the files list and press the **Reload** button. ControlDesk reloads the files from their original file source paths, if possible.

Modifying (adding, replacing, or removing) the files belonging to an ODX database that was already configured for the device causes a reload of the ODX database.

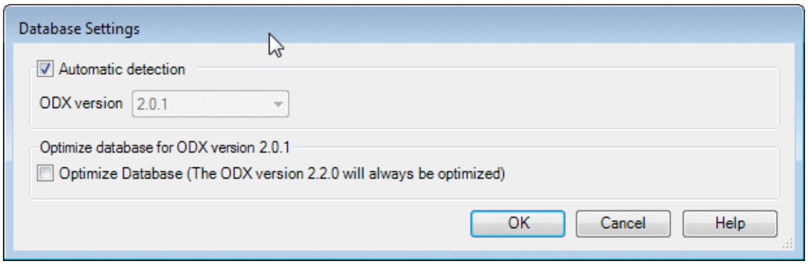
5 You can specify to use a proprietary binary format (generated from the ODX data) instead of ODX data for the diagnostics database. Using the binary files allows faster experiment loading and lets you work with large ODX databases since memory usage is reduced. However, transforming the ODX data into the binary format takes some time, so you should think carefully whether to use ODX data or the binary format with your experiment.

Tip

Using the binary format is useful in the following cases:

- Experiments that are frequently loaded, and whose diagnostics database is modified only rarely
- Experiments that require large ODX databases

To generate the binary format for the ODX 2.0.1 diagnostics database and use it with your experiment, click the **Database Settings** button at top right in the **Select ODX Files** dialog and select the **Optimize database** checkbox in the **Database Settings** dialog.



Database Settings

☑ Automatic detection

ODX version: 2.0.1

Optimize database for ODX version 2.0.1

□ Optimize Database (The ODX version 2.2.0 will always be optimized)

Note

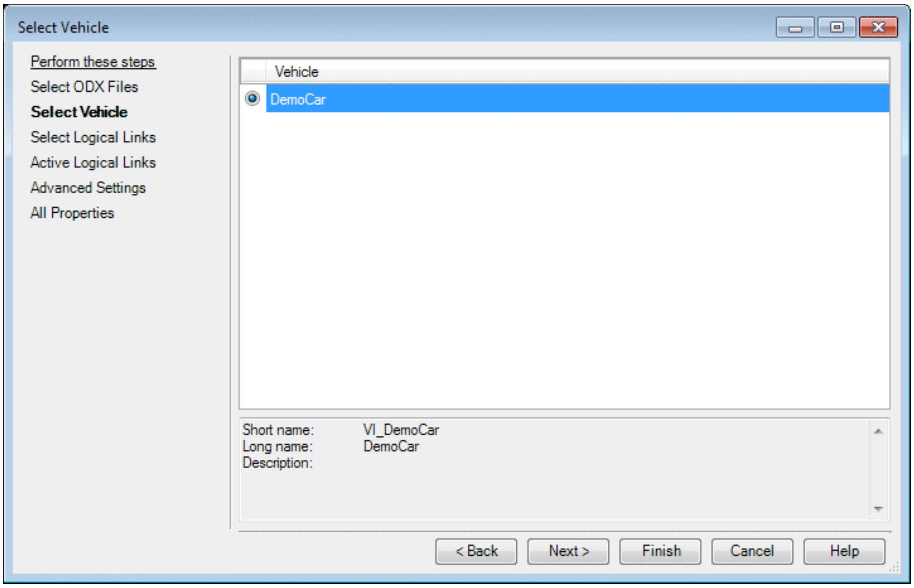
For ODX 2.2.0 diagnostics databases, database optimization is always performed. You cannot change this setting. So if you work with an ODX 2.2.0 database, ControlDesk always uses the binary format instead of ODX data for the diagnostics database.

For further information on specifying the diagnostics database, refer to Select ODX Files dialog (ControlDesk Platform Management 📖).

6 Click **Next >**.

ControlDesk analyzes the specified ODX diagnostics database, and then displays it as the active diagnostics database in the **Project** controlbar (indicated by the ✓ODX or ✓BIN symbol, depending on whether the ODX data or the binary format is used for the database). It also adds a folder with the name of the ODX database to the project's folder structure on your file system, containing all the files belonging to the specified ODX database and, if needed, the generated binary files. Do not make any changes to the files on the file system. Changes to the ODX database or the generated binary files must be made exclusively via the **Select ODX Files** dialog.

The **Select Vehicle** dialog opens, displaying all the vehicles contained in the specified ODX database.



	Vehicle
●	DemoCar

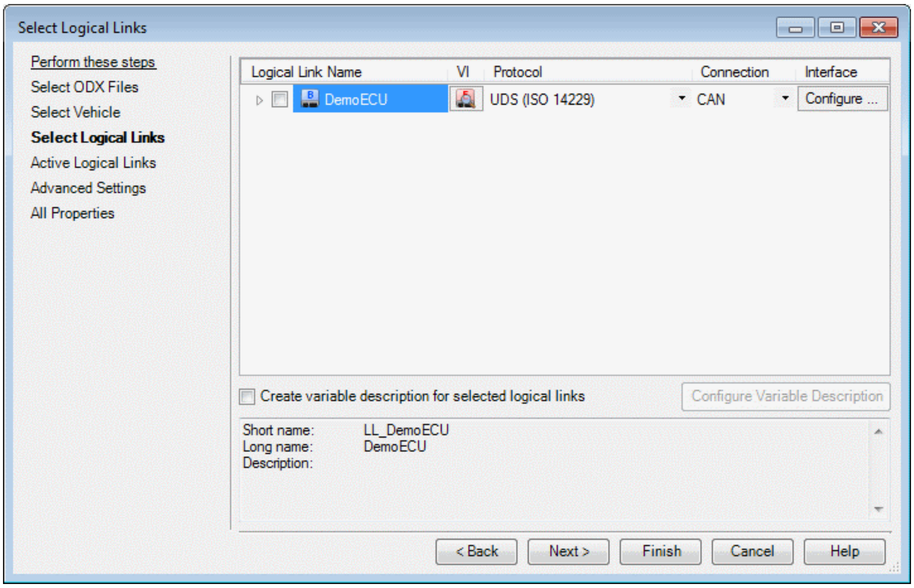
Short name: VI_DemoCar
Long name: DemoCar
Description:

If no vehicle information is specified in the ODX database, a default vehicle named 'VI_GlobalVehicleInfor' (short name) or 'Global Vehicle Information' (long name) is generated automatically for further use.

7 Select the vehicle to be used with the ECU Diagnostics device from the vehicles list.

8 Click **Next >**.

The **Select Logical Links** dialog opens, displaying the logical links available in the ODX database for the selected vehicle.



Logical Link Name	VI	Protocol	Connection	Interface
□ DemoECU		UDS (ISO 14229)	CAN	Configure ...

□ Create variable description for selected logical links

Short name: LL_DemoECU
Long name: DemoECU
Description:

The logical links are displayed hierarchically according to the structure of the hierarchical layers (protocol, functional group, ECU base variant, ECU variant) of the ODX database. For clarity, you can reduce the display to the logical links to ECU base variants and ECU variants via the **Hide Protocol and Functional Group** command (available from the context menu).

If the ODX database contains no vehicle information specification and the 'VI_GlobalVehicleInfor' (short name) or 'Global Vehicle Information' (long name) vehicle is selected for use instead, the logical link list contains only automatically generated logical links. Logical links are generated according to a specified pattern (described here with the logical links' short names):

- GeneratedLogicalLink_ProtocolShortName
- GeneratedLogicalLink_FunctionalGroupShortName_ProtocolShortName
- GeneratedLogicalLink_BaseVariantShortName_FunctionalGroupShortName_ProtocolShortName
- EcuVariantDiagLayerShortName_GeneratedLogicalLink_BaseVariantShortName_FunctionalGroupShortName

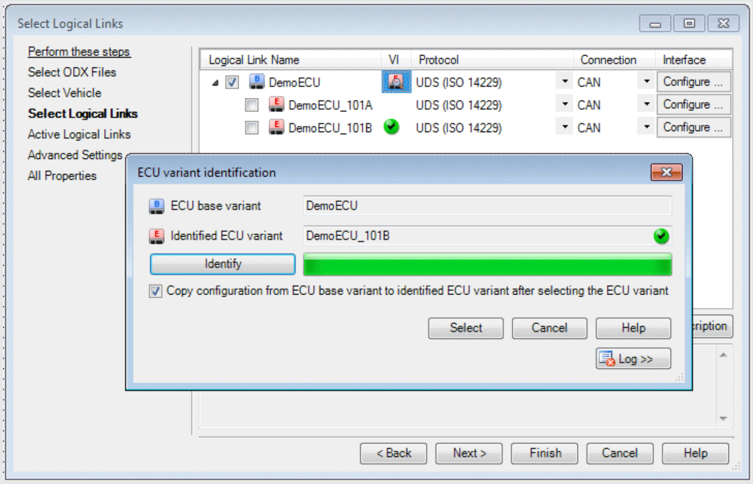
Note

The pattern depends on the structure of the underlying database.

A logical link is generated for each possible combination. Thus, the number of generated logical links depends on the layer definitions available in the ODX database.

Tip

ControlDesk supports logical link selection via ODX variant identification. If the ODX database contains one or more ECU variants for an ECU base variant (indicated by a button next to the ECU base variant), you can open the **ECU Variant Identification** dialog via the button and start the identification process. If an ECU variant is identified, you can select it for use with the ECU Diagnostics device.
Additionally, you can specify to apply the configuration of the ECU base variant to the selected identified ECU variant.



Logical Link Name	VI	Protocol	Connection	Interface
☑ DemoECU		UDS (ISO 14229)	CAN	Configure ...
□ DemoECU_101A		UDS (ISO 14229)	CAN	Configure ...
□ DemoECU_101B	✓	UDS (ISO 14229)	CAN	Configure ...

ECU base variant: DemoECU

Identified ECU variant: DemoECU_101B ✓

☑ Copy configuration from ECU base variant to identified ECU variant after selecting the ECU variant

For further information, refer to Select Logical Links Dialog (ControlDesk Platform Management 📖).

9 Specify whether a variable description is to be generated from the ODX database by selecting or clearing the **Create variable description for selected logical links** checkbox. If you want to measure or calibrate variables via the ECU Diagnostics device later on, the checkbox must be selected.

If the checkbox is selected, you can specify further variable description configuration settings. Click **Configure Variable Description** to open the **Diagnostic Variable Description Configuration** dialog. For example, you can specify to generate DTC measurement variables which allow you to measure changes on diagnostic trouble codes. For further information on the configuration settings, refer to Select Logical Links Dialog (ControlDesk Platform Management 📖).

10 Specify the logical links to be used with the ECU Diagnostics device by selecting or clearing the checkboxes to the left of the **Logical Link Names**. An activated checkbox indicates that the logical link is selected for ECU diagnostics.

Each logical link that is selected for ECU diagnostics must then be configured as described in the following steps.

11 Choose a selected logical link and specify its diagnostic protocol. The following diagnostic protocols are available:

Diagnostic Protocol	Description
Diagnostics on CAN (ISO 15765)	Diagnostics on CAN / KWP2000 on CAN
KWP2000 on K-Line (ISO 14230)	KWP2000 on K-Line
OBD	On-Board Diagnostics for CAN-based ECU diagnostics
TP 1.6	Transport Protocol TP 1.6 for CAN-based ECU diagnostics
TP 2.0	Transport Protocol TP 2.0 for CAN-based ECU diagnostics
UDS (ISO 14229)	UDS on CAN / UDS on DoIP (Diagnostics over Internet Protocol)

The diagnostic protocol setting is preset with data from the ODX database the first time you open the **Select Logical Links** dialog during device configuration, provided that the ODX database contains appropriate data. If no protocol setting can be associated, the diagnostic protocol is set to 'None' and you must specify it manually.

12 Select the physical connection that is to be used for the logical link. The available physical connections depend on the selected diagnostic protocol.

The physical connection is preset with data from the ODX database the first time you open the **Select Logical Links** dialog during device configuration, provided that the ODX database contains appropriate data. If no physical connection setting can be associated, the physical connection is set to 'None' and you must specify it manually.

Note

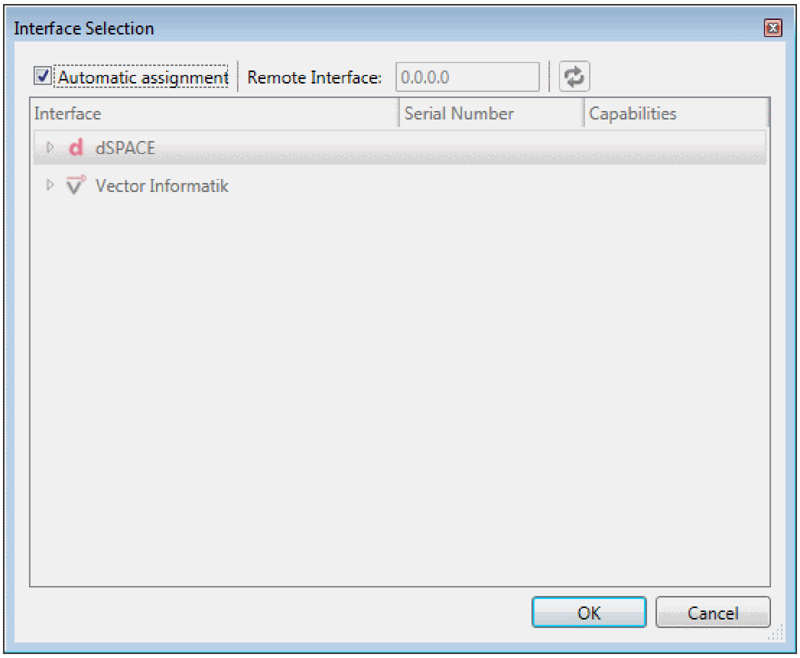
Your logical link selection must not contain more than one simulated logical link. If you select two or more logical links for which 'Simulation' is specified as the physical connection, the selection is invalid and ControlDesk opens a dialog for you to define a valid logical link selection.

The selected ECU connection requires further interface configuration settings, corresponding to the selected physical connection:

- For a CAN-based logical link, see next step.
- For a K-Line-based logical link, skip to step 16.
- For an Ethernet-based logical link or the 'Simulation' physical connection, no further settings are required; skip to step 19.

13 Click Configure.

The Interface Selection dialog opens.



Interface Selection

☑ Automatic assignment Remote Interface: 0.0.0.0

Interface	Serial Number	Capabilities
dSPACE		
Vector Informatik		

14 Select the physical CAN interface to be assigned to the logical link. ControlDesk provides various options. What to do also depends on the interface you use:

Interface Type	Interface Assignment
CAN channel of SCALEXIO, MicroAutoBox III, or VEOS	CAN channels of a SCALEXIO or MicroAutoBox III system or VEOS must be made available to the device before they can be assigned to the device. There are two ways to do this: ▪ Add a SCALEXIO, MicroAutoBox III, or VEOS platform that has an application with suitable CAN channels loaded to the current experiment. The CAN channels of all the SCALEXIO, MicroAutoBox III, or VEOS platforms in the experiment are available. ▪ Automatic assignment: Select Automatic assignment to automatically select a CAN channel of a SCALEXIO or MicroAutoBox III system or on VEOS. ▪ Manual assignment: Clear the Automatic assignment checkbox if you wish to select a listed CAN channel manually. ▪ CAN channels of a SCALEXIO or MicroAutoBox III system or VEOS that are not in the current experiment can be made available in the Interface Selection dialog: Clear the Automatic assignment checkbox. Next to it, in the Remote Interface field, enter the IP address of the SCALEXIO or MicroAutoBox III system or VEOS platform whose CAN channel you want to assign to the device. If the desired CAN channel is not displayed at this point, press the Scan Interfaces button. If the CAN channel of SCALEXIO or MicroAutoBox III that you want to use is also used in the model underlying the application loaded to the SCALEXIO or MicroAutoBox III platform, you must enter the IP address that is specified in the model.
Connected PC-based interface (e.g., DCI-CAN/LIN1)	▪ Automatic assignment: Select Automatic assignment to automatically select a connected PC-based interface. ▪ Manual assignment: Clear the Automatic assignment checkbox and select one of the connected PC-based interfaces from the list to be assigned to the device.

Interface Type	Interface Assignment
Currently not connected interface	Clear the Automatic assignment checkbox and click the ⚙ button beside Offline Configuration to assign an interface to the device while the interface is currently not connected to the host PC. ControlDesk then prompts you to supply an interface name in the User Identifier edit field and (optionally) a channel number in the Controller Identifier edit field.
Virtual interface (for testing purposes)	Clear the Automatic assignment checkbox and select a virtual interface that is simulated on the host PC (possible for CAN interfaces only). **Note** Do not confuse the 'Virtual' CAN interface with CAN channels on VEOS.

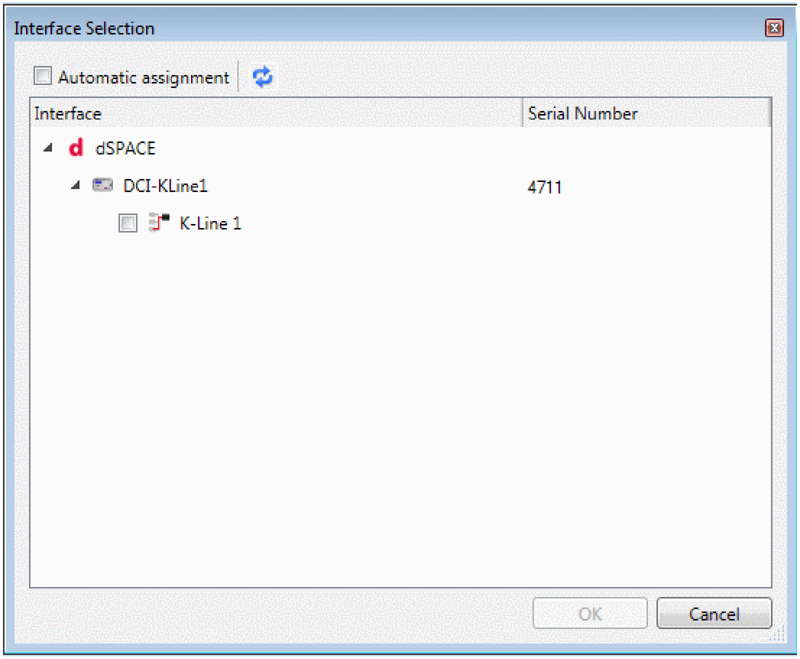
For further information on selecting physical CAN interfaces, refer to Interface Selection (CAN- and LIN-based devices) dialog (ControlDesk Platform Management 📖).

15 Click OK.

The Interface Selection dialog is closed; skip to step 19.

16 Click Configure.

The Interface Selection dialog opens.



Interface Selection

☐ Automatic assignment

Interface	Serial Number
dSPACE	
DCI-KLine1	4711
☐ K-Line 1	

17 Select the physical K-Line interface to be assigned to the logical link. ControlDesk provides the following options:

- Select Automatic assignment to select a connected interface automatically. If only one K-Line interface is connected to the host PC, ControlDesk assigns it to the device, regardless of the serial number of the interface. If you have several K-Line interfaces connected to the host PC, ControlDesk assigns the ECU Diagnostics device to the interface that has the smallest serial number.

- Clear the **Automatic assignment** checkbox and select one of the connected interfaces from the list to be assigned to the logical link. All the K-Line interfaces that are connected to the host PC are listed with their serial numbers.

Note

ControlDesk can use only one (physical) K-Line interface at a time.

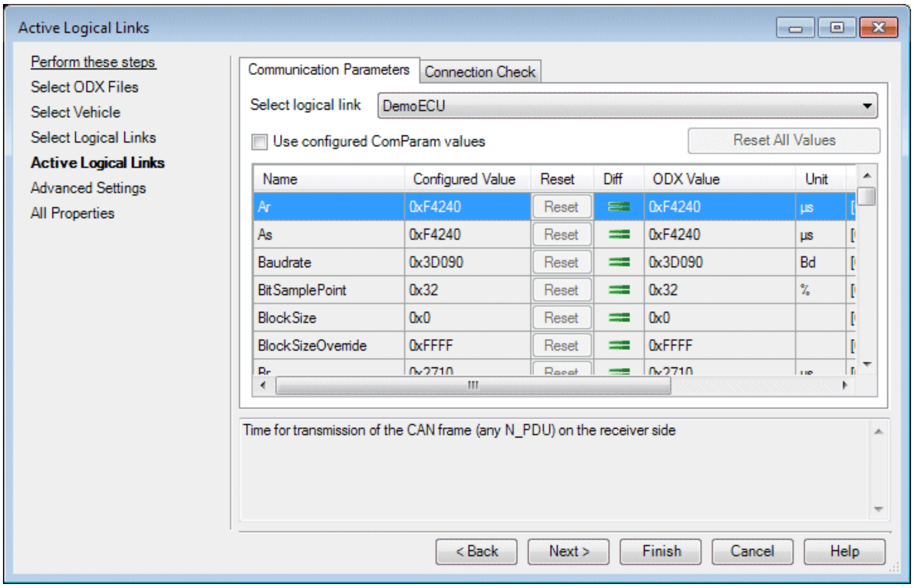
For further information on selecting physical K-Line interfaces, refer to Interface Selection dialog (ControlDesk Platform Management 🕮).

18 Click **OK** to close the **Interface Selection** dialog.

19 Repeat steps 11 … 18 for all the logical links that are selected for use with the ECU Diagnostics device.

20 Click **Next >**.

The **Active Logical Links** dialog opens, displaying the **Communication Parameters** page.



Select logical link: DemoECU

☐ Use configured ComParam values

Name	Configured Value	Reset	Diff	ODX Value	Unit
Ar	0xF4240	Reset	=	0xF4240	µs
As	0xF4240	Reset	=	0xF4240	µs
Baudrate	0x3D090	Reset	=	0x3D090	Bd
BitSamplePoint	0x32	Reset	=	0x32	%
BlockSize	0x0	Reset	=	0x0	
BlockSizeOverride	0xFFFF	Reset	=	0xFFFF	
Br	0x2710	Reset	=	0x2710	µs

Time for transmission of the CAN frame (any N_PDU) on the receiver side

For each logical link selected to be used with the ECU Diagnostics device, the relevant communication parameters are displayed with the original values specified for them in the ODX database and their manually configured values. By default, the configured values are equal to the ODX values. You can switch between the active logical links via the drop-down list at the top of the dialog.

Note

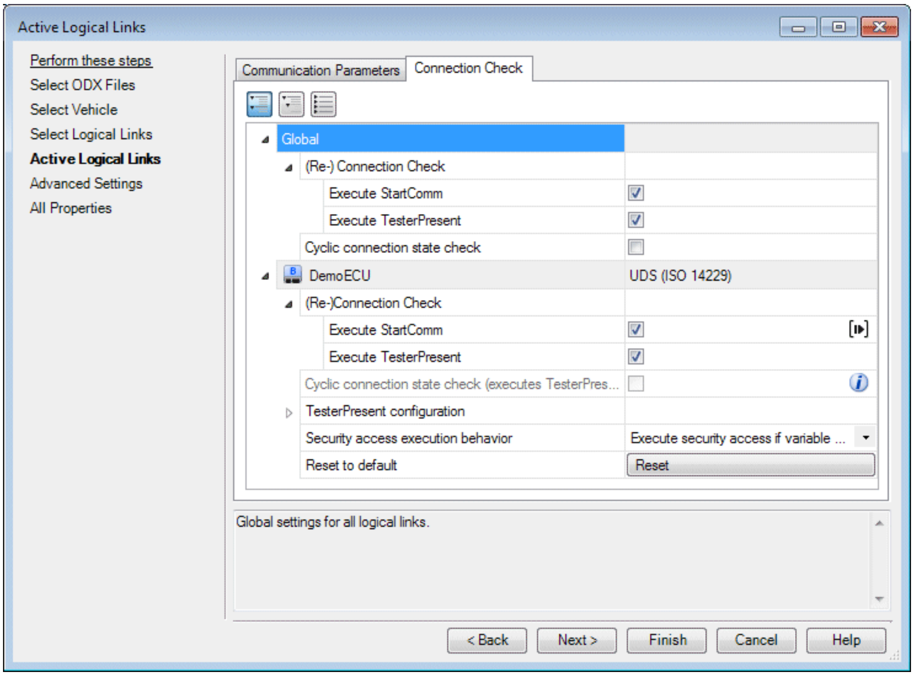
- If a CAN FD bus is used, the ODX database of the ECU Diagnostics device must contain CAN FD-specific communication parameters, such as `CANFDBaudrate`, `CANFDBitSamplePoint`, `CANFDSyncJumpWidth`, and `CANFDTxMaxDataLength` so that the CAN interface is initialized with CAN FD. This also applies if the device is to transmit classic CAN messages on a CAN FD bus. Refer to Conventions in Connection with ODX Databases on page 22.
- If you use the DoIP protocol, the `DoIPEntityIPAddress` communication parameter is mandatory for each logical link selected to be used with the ECU Diagnostics device. Its value specifies the IP address of the ECU to be accessed. For more information, refer to Conventions in Connection with ODX Databases on page 22.

21 If necessary, specify user-defined values for single communication parameters by overwriting the appropriate entries in the **Configured Value** column. To make the **Configured Value** fields editable, the **Use configured ComParam values** checkbox must be selected. To reset the configured value of a communication parameter to its original ODX value, you can click the appropriate **Reset** button.

Via the **Use configured ComParam values** checkbox, you can specify to use the configured values for the communication parameters. If the checkbox is cleared, the original ODX values are used.

You can modify the user-defined values or switch between the ODX values and user-defined values later on via the **Properties** controlbar. Refer to Logical Link Selection Properties (ControlDesk Platform Management 🕮).

22 Open the **Connection Check** page in the **Active Logical Links** dialog to specify, for each individual active logical link, whether and how to perform the (re)connection check and the cyclic ECU communication check after the device is connected.



Global	
(Re-) Connection Check	
Execute StartComm	☑
Execute TesterPresent	☑
Cyclic connection state check	☐
DemoECU	UDS (ISO 14229)
(Re-)Connection Check	
Execute StartComm	☑
Execute TesterPresent	☑
Cyclic connection state check (executes TesterPres...	☐
TesterPresent configuration	
Security access execution behavior	Execute security access if variable ...
Reset to default	Reset

Global settings for all logical links.

The connection check settings are made of two steps:

The *global settings* let you activate or deactivate the logical-link-specific settings for executing the `StartCommunication` control primitive and/or the TesterPresent service during the connection/reconnection check and for executing the cyclic ECU communication check when the device is connected. If an option is globally enabled, the associated logical-link-specific properties are activated and evaluated. You can then specify the option individually for each active logical link (see below). If an option is globally disabled, it is disabled for all logical links, and any associated logical-link-specific settings are always deactivated and do not take effect.

Via the *logical-link-specific* settings, you can disable the checks for single active logical links or specify how to perform the enabled checks for single active logical links.

- Execution of `StartCommunication` control primitive during the connection check:
  - To let ControlDesk execute a `StartCommunication` control primitive during the connection check, ensure that the **Execute StartComm** checkbox is selected for the logical link.

Enabling the **Execute StartComm** option ensures that ControlDesk activates the cyclic transmission of TesterPresent messages to the ECU if specified in the ODX database. You can disable the cyclic transmission via the configuration of the `COMPARAMs`.

- Clear the **Execute StartComm** checkbox to disable the execution of the `StartCommunication` control primitive during the connection check for the selected logical link.

> **Note**
>
> For K-Line-based diagnostic protocols, the `StartCommunication` control primitive must be executed to initialize the bus. So if you want to access an ECU via K-Line, you should not disable this option, because otherwise communication with an ECU via K-Line might be impossible.

- Execution of a TesterPresent service during the connection check:
  - To let ControlDesk execute a TesterPresent service with a specific semantic, ensure that the **Execute TesterPresent** checkbox is selected for the logical link, select **Execute a service with the specified semantic** from the **TesterPresent behavior** list and, in the **Semantic for service identification** edit field, select the semantic that classifies the service to be used for the connection check (default setting: TESTERPRESENT).
  - To let ControlDesk execute a TesterPresent service that matches the selected diagnostic protocol, make sure that the **Execute TesterPresent** checkbox is selected for the logical link, and select **Send the protocol-specific TesterPresent service** from the **TesterPresent behavior** list. For details on the used request PDU and response PDU values, refer to Active Logical Links - Connection Check page (ControlDesk Platform Management 🕮).
  - To let ControlDesk send a custom request PDU to the ECU and react to the ECU's response in a defined manner, ensure that the **Execute TesterPresent** checkbox is selected for the logical link, select the suitable **Send a custom PDU, accept ...** entry from the **TesterPresent behavior** list and specify the request PDU to be executed via the hex service in the **Request PDU** edit field. If only one specific response PDU means a

successful connection check, specify that PDU in the **Response PDU** edit field.

- Clear the **Execute TesterPresent** checkbox to disable the execution of a TesterPresent service during the connection check for the selected logical link.

Note

> For CAN-based diagnostic protocols, the connection check that is performed during the state transition from 'disconnected' to 'connected' is done by executing a TesterPresent service for the logical links. So if you want to check the access to an ECU via CAN, you should enable this option and specify the **TesterPresent configuration** settings.

- Cyclic connection state check:
  - To let ControlDesk execute a TesterPresent service to check the ECU communication cyclically after the logical link is connected, ensure that the **Cyclic connection state check (executes TesterPresent)** checkbox is selected for the logical link. ControlDesk then transmits a message (according to the **TesterPresent Configuration** you specified for the

logical link) cyclically to the ECU to check whether ECU communication is available.

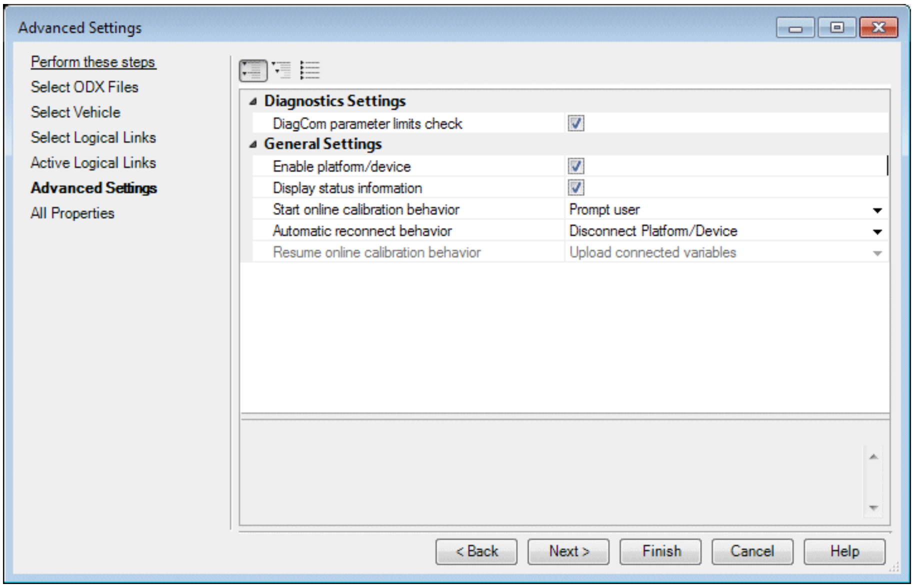
Tip

- ControlDesk comes with default configuration settings for the connection check according to the diagnostic protocol used. Via the **Reset to default** button you can reset the logical-link-specific configuration settings for the connection check behavior to their default values.
- In some cases, ControlDesk displays symbols that mark specific configurations. The tool tips of the symbols provide corresponding information.
  - A ⓘ symbol next to a property indicates that you can get more information on a certain setting. For example, if an option is globally disabled, ControlDesk displays the symbol next to the associated logical-link-specific properties.
  - `StartCommunication` control primitives with the `NoOperation` property set to `True` are indicated by a [▶] symbol, indicating that the control primitive is not suitable for the connection check since it does not send a request to the ECU.
  - A ⚠ symbol indicates that a recommended setting has been switched off or modified, which can result in deactivated functionality. For example, if you disable **Execute StartComm**, but there are `StartCommunication` control primitives with the `NoOperation` property set to `FALSE`, ControlDesk displays the ⚠ symbol.

23 Specify whether and when the diagnostic job for preparing the ECU for security access is to be executed (with every online calibration start, or only if a variable description was generated from the ODX database for the ECU Diagnostics device). To do so, select the appropriate value from the **Security access execution behavior** list.

24 Click Next >.

The Advanced Settings dialog opens.



Advanced Settings

Perform these steps
Select ODX Files
Select Vehicle
Select Logical Links
Active Logical Links
Advanced Settings
All Properties

Diagnostics Settings

DiagCom parameter limits check	☑

General Settings

Enable platform/device	☑
Display status information	☑
Start online calibration behavior	Prompt user
Automatic reconnect behavior	Disconnect Platform/Device
Resume online calibration behavior	Upload connected variables

25 Select whether diagnostics-specific checks are to be performed when working with the ECU Diagnostics device and specify general settings for the device.

For further information, refer to Advanced Settings dialog (ControlDesk Platform Management 📖).

26 Click Next >.

The All Properties dialog opens.

27 In the dialog, view and specify the settings:

Purpose	Refer to
To specify common properties of the platform/device.	Common Properties (ControlDesk Platform Management 📖)
To specify diagnostics settings for the selected device.	Diagnostics Settings Properties (ControlDesk Platform Management 📖)
To specify general settings of the selected platform/device.	General Settings Properties (ControlDesk Platform Management 📖)
To select logical links for ECU diagnostics, to display the active logical links of the ECU Diagnostics device with their configurations, and to specify connection check settings and configure communication parameters individually for each active logical link.	Logical Link Selection Properties (ControlDesk Platform Management 📖)
To display and configure the ODX database specified for the selected ECU Diagnostics device.	Active ODX Database / ODX Database Properties (ControlDesk Platform Management 📖)
To display the vehicle that is selected to be used with the selected ECU Diagnostics device and to select another vehicle.	Vehicle Selection Properties (ControlDesk Platform Management 📖)

28 Click Finish to close the dialog.

Whenever you finish the configuration of a platform/device, ControlDesk checks whether the platform/device is in the 'connected' state. If the connection cannot be established, ControlDesk opens the Platform/Device

State Overview dialog for the platform/device. Click **OK** or **Cancel** to close the **Platform/Device State Overview** dialog.

Result

You have now configured the ECU Diagnostics device. The logical links which are selected for use with this device are displayed in the **Project** controlbar.

> **Tip**
>
> ControlDesk displays the settings you specified during device configuration hierarchically in the device's **Properties** controlbar. You can also (re)configure an ECU Diagnostics device via the **Properties** controlbar. Refer to Platform/Device-Related Properties (ControlDesk Platform Management 🕮).

Next steps

You can now place instruments on a layout and perform ECU diagnostics tasks, or, if a variable description was generated from the ODX database for the device, measure and calibrate scalar diagnostics variables via the ECU Diagnostics device. Refer to

- ControlDesk ECU Diagnostics 🕮 on page 1
- ControlDesk Measurement and Recording 🕮
- Calibrating Parameters (ControlDesk Calibration and Data Set Management 🕮)
- How to Program the ECU Flash Memory via a Diagnostic Protocol on page 88

You can specify multiple diagnostics databases for the device. Only one database can be active at a time. Refer to

- Add ODX Database (ControlDesk Platform Management 🕮)
- Activate ODX Database (ControlDesk Platform Management 🕮)
- Configure ODX Database (ControlDesk Platform Management 🕮)

Related topics

Basics

Basics of the ECU Diagnostics Device.. 32

References

Configure Platform/Device (ControlDesk Platform Management 🕮)
Platform/Device State Overview Dialog (ControlDesk Platform Management 🕮)

Performing ECU Diagnostic Tasks

Where to go from here

Information in this section

Reading the Fault Memory of an ECU

Introduction

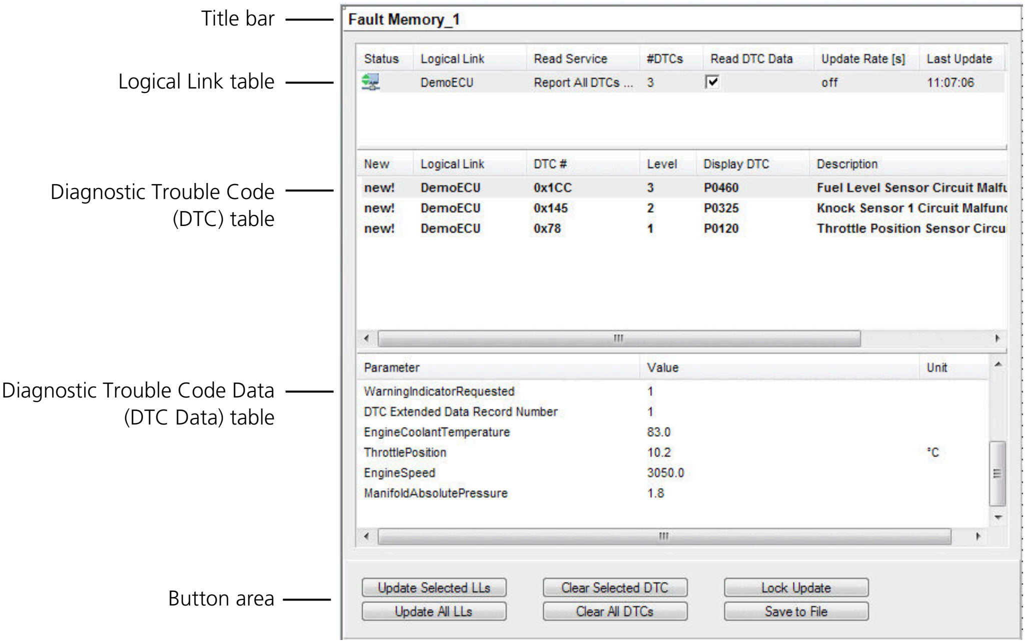
ControlDesk provides the Fault Memory Instrument to read, clear and save the content of the fault memory of an ECU. To read and display entries of the ECU's fault memory, ControlDesk uses specific diagnostic services specified in ODX.

Fault memory

The fault memory of an ECU stores diagnostic trouble code (DTC) entries with status and environment information. With ControlDesk, you can read these entries, save them to a file, and delete them. The entries can be updated manually or automatically at defined intervals.

Fault Memory Instrument

The Fault Memory Instrument consists of several parts.



Fault Memory_1

Status	Logical Link	Read Service	#DTCs	Read DTC Data	Update Rate [s]	Last Update
	DemoECU	Report All DTCs ...	3	☑	off	11:07:06

New	Logical Link	DTC #	Level	Display DTC	Description
new!	**DemoECU**	**0x1CC**	**3**	**P0460**	**Fuel Level Sensor Circuit Malfu**
new!	**DemoECU**	**0x145**	**2**	**P0325**	**Knock Sensor 1 Circuit Malfunc**
new!	**DemoECU**	**0x78**	**1**	**P0120**	**Throttle Position Sensor Circu**

Parameter	Value	Unit
WarningIndicatorRequested	1	
DTC Extended Data Record Number	1	
EngineCoolantTemperature	83.0	
ThrottlePosition	10.2	°C
EngineSpeed	3050.0	
ManifoldAbsolutePressure	1.8	

Logical Link table Displays all the logical links activated for the instrument, the selected read service, the number of received DTCs and the selected update rate.

Diagnostic Trouble Code (DTC) table Lists the diagnostic trouble codes that have been received for all the logical links in the instrument.

Diagnostic Trouble Code Data (DTC Data) table Displays the status and environment data of the DTC that is selected in the diagnostic trouble code table. The table is empty if no DTC or more than one is selected.

Default diagnostic service for reading fault memory according to protocol-specific service identification

You can read and display fault memory entries in ControlDesk's Fault Memory Instrument using a default diagnostic service according to the protocol-specific identification of services.

If your ODX database contains suitable data, ControlDesk enables reading the fault memory for a specific diagnostic service, regardless of whether an XML configuration for reading the fault memory is available or not. This diagnostic service is identified via the service ID.

If possible, two fault memory services are provided: one using the default DTC status byte, the other using status byte = 0xFF. You can recognize a protocol-specific default service by its name, where the associated PDU is displayed in brackets.

Configuring further services for reading fault memory

If you want to use a fault memory read service different from the default services in ControlDesk, you can specify further read services in an XML configuration file. For further information, refer to Configuring Services for Reading Fault Memory in the XML Configuration File on page 100.

Related topics

Basics

Basics of the XML Configuration File..93
Conventions in Connection with ODX Databases...22

HowTos

How to Display Fault Memory Entries..61

Reading Environment Data

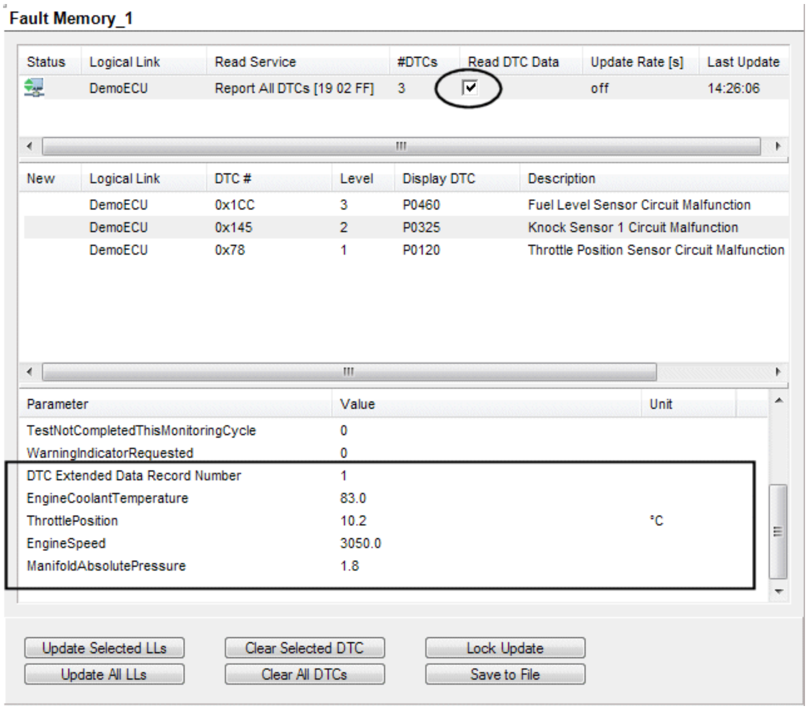
Introduction

The Fault Memory Instrument can display environment data that corresponds to a specific fault memory entry. To read environment data of diagnostic trouble codes, ControlDesk uses specific diagnostic services specified in ODX.

Basics on reading environment data

The fault memory of an ECU stores diagnostic trouble code (DTC) entries with status and environment information. The Fault Memory Instrument can read and display the environment data of a diagnostic trouble code in addition to its status data.

The additional display of environment data must be activated individually for each logical link via the **Read DTC Data** checkbox. If activated, updating the instrument takes more time, because not only the service for reading the ECU's fault memory is executed, but also the service for reading the environment data for each DTC.



Fault Memory_1

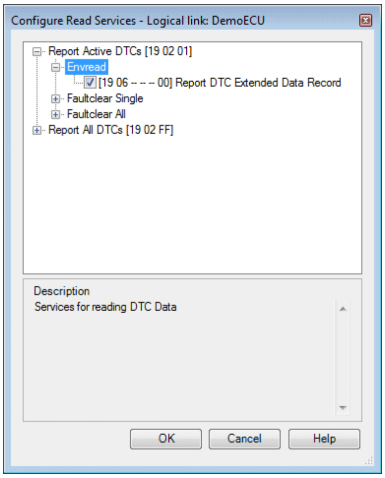
Status	Logical Link	Read Service	#DTCs	Read DTC Data	Update Rate [s]	Last Update
	DemoECU	Report All DTCs [19 02 FF]	3	☑	off	14:26:06

New	Logical Link	DTC #	Level	Display DTC	Description
	DemoECU	0x1CC	3	P0460	Fuel Level Sensor Circuit Malfunction
	DemoECU	0x145	2	P0325	Knock Sensor 1 Circuit Malfunction
	DemoECU	0x78	1	P0120	Throttle Position Sensor Circuit Malfunction

Parameter	Value	Unit
TestNotCompletedThisMonitoringCycle	0	
WarningIndicatorRequested	0	
DTC Extended Data Record Number	1	
EngineCoolantTemperature	83.0	
ThrottlePosition	10.2	°C
EngineSpeed	3050.0	
ManifoldAbsolutePressure	1.8	

The service to be used for reading environment data (just like the services for clearing single and all fault memory entries) is always specified in connection with the selected service for reading the fault memory. This means that the read service you select for a logical link in the Fault Memory Instrument actually stands for a combination of services for reading the fault memory, reading environment data, clearing single fault memory entries and clearing all fault memory entries. If no service for reading the fault memory is available, there is also no service for reading environment data.

The **Configure Read Services** dialog displays the available services for you to select a combination. You can call this selection dialog via the **Properties** controlbar of the Fault Memory Instrument. Refer to Logical Links Properties (ControlDesk Instrument Handling 📖).



- Report Active DTCs [19 02 01]
  - Envread
    - ☑ [19 06 -- -- -- 00] Report DTC Extended Data Record
  - Faultclear Single
  - Faultclear All
- Report All DTCs [19 02 FF]

Description
Services for reading DTC Data

Default diagnostic service for reading environment data according to protocol-specific service identification

You can read and display the environment data that corresponds to a specific fault memory entry in ControlDesk's Fault Memory Instrument using a default diagnostic service according to the protocol-specific identification of services.

Note

Protocol-specific default services are not available for the OBD diagnostic protocol.

If your ODX database contains suitable data, ControlDesk enables reading environment data for a specific diagnostic service, regardless of whether an XML configuration for reading environment data is available or not. This diagnostic service is identified via the service ID and the byte position.

Configuring further services for reading environment data

If you want to use a service different from the default services for reading environment data in ControlDesk, you can specify further diagnostic services in an XML configuration file. For further information, refer to Configuring Services for Reading Environment Data in the XML Configuration File on page 104.

Related topics

Basics

Basics of the XML Configuration File..93
Conventions in Connection with ODX Databases..........................22

HowTos

How to Display Fault Memory Entries..61

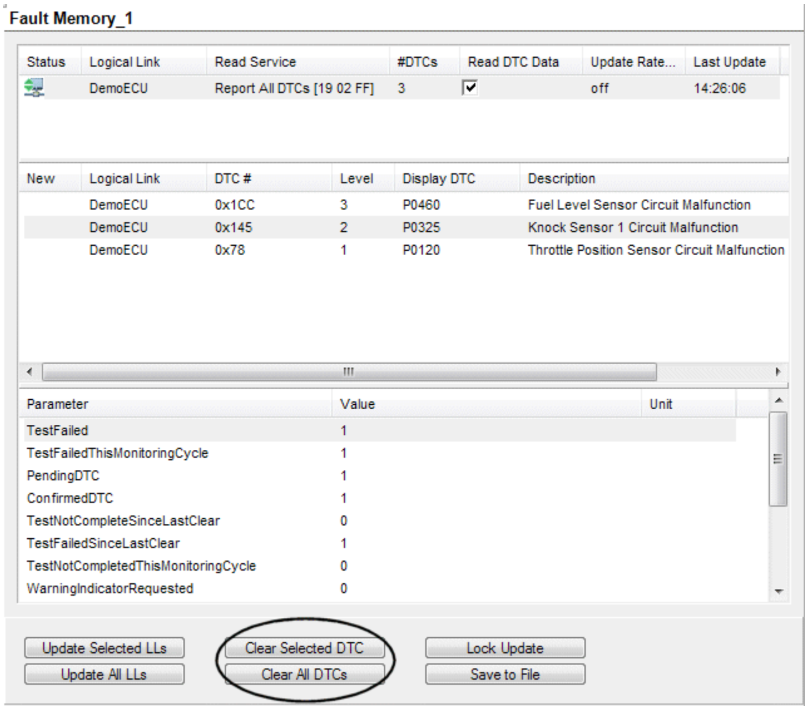
Clearing Fault Memory Entries

Introduction

The Fault Memory Instrument lets you delete single or all trouble code entries from the fault memory. To clear fault memory entries, ControlDesk uses specific diagnostic services specified in the ODX database.

Basics on clearing fault memory entries

The fault memory of an ECU stores diagnostic trouble code (DTC) entries with status and environment information. The Fault Memory Instrument lets you delete the trouble code entries referenced by the logical links from the fault memory of the ECUs. You can clear all trouble code entries in one step, or a single trouble code entry if the ECU allows the deletion of a selected DTC.



Fault Memory_1

Status	Logical Link	Read Service	#DTCs	Read DTC Data	Update Rate...	Last Update
	DemoECU	Report All DTCs [19 02 FF]	3	☑	off	14:26:06

New	Logical Link	DTC #	Level	Display DTC	Description
	DemoECU	0x1CC	3	P0460	Fuel Level Sensor Circuit Malfunction
	DemoECU	0x145	2	P0325	Knock Sensor 1 Circuit Malfunction
	DemoECU	0x78	1	P0120	Throttle Position Sensor Circuit Malfunction

Parameter	Value	Unit
TestFailed	1	
TestFailedThisMonitoringCycle	1	
PendingDTC	1	
ConfirmedDTC	1	
TestNotCompleteSinceLastClear	0	
TestFailedSinceLastClear	1	
TestNotCompletedThisMonitoringCycle	0	
WarningIndicatorRequested	0	

The services to be used for clearing single fault memory entries and clearing all fault memory entries (just like the service for reading environment data) are always specified in connection with the selected service for reading the fault memory. This means that the read service you select for a logical link in the Fault Memory Instrument actually stands for a combination of services for reading the fault memory, reading environment data, clearing single fault memory entries and clearing all fault memory entries. If no service for reading the fault memory is available, there is also no service for clearing fault memory entries.

ControlDesk lets you select the services for clearing single fault memory entries and all fault memory entries, if several services are available. You can call the selection dialog via the **Properties** controlbar of the Fault Memory Instrument. Refer to Logical Links Properties (ControlDesk Instrument Handling 🕮).

Default diagnostic service for clearing fault memory entries according to protocol-specific service identification

You can clear fault memory entries using a default diagnostic service according to the protocol-specific identification of services.

If your ODX database contains suitable data, ControlDesk enables clearing single fault memory entries and clearing all fault memory entries for specific diagnostic services, regardless of whether XML configurations for clearing fault memory entries are available or not. These protocol-specific diagnostic services are identified via service IDs and byte positions.

Configuring further services for deleting fault memory entries

If you want to use services others than the default services for clearing fault memory entries in ControlDesk, you can specify further diagnostic services in an XML configuration file. For further information, refer to Configuring Services for Clearing Single Fault Memory Entries in the XML Configuration File on page 106 and Configuring Services for Clearing All Fault Memory Entries in the XML Configuration File on page 108.

Related topics

Basics

Basics of the XML Configuration File.. 93
Conventions in Connection with ODX Databases.. 22

HowTos

How to Display Fault Memory Entries.. 61

How to Display Fault Memory Entries

Objective

With the Fault Memory Instrument you can read, clear, and save entries of the ECU's fault memory.

Basics

For basic information on reading fault memory via the Fault Memory Instrument, refer to Reading the Fault Memory of an ECU on page 56.

Lock update

The fault memory information is automatically synchronized for each logical link. If you update the information on a logical link in one Fault Memory Instrument, it is also updated in all other Fault Memory Instruments that reference the same logical link and use the same read service.

You can 'freeze' the content of a selected Fault Memory Instrument with Lock Update. Updates of logical links are then not displayed in this Fault Memory Instrument until you click Unlock Update.

Preconditions

- When performing ECU diagnostics with ControlDesk, you have to observe some conventions in connection with the ODX database you use. Refer to Conventions in Connection with ODX Databases on page 22.
- If the default diagnostic service definitions (i.e., the default services according to the protocol-specific service identification) are unsuitable or if no default diagnostic services can be generated, you must use an XML configuration file containing the service configurations matching your requirements. The XML file must meet some requirements. Refer to Basics of the XML Configuration File on page 93.
- A Fault Memory Instrument must be placed on a layout. Refer to How to Place an Instrument for Diagnostic Tasks on a Layout on page 29.
- To access the fault memory of an ECU, online calibration must be started. Refer to How to Start and Stop Online Calibration (ControlDesk Calibration and Data Set Management 🕮).

Method

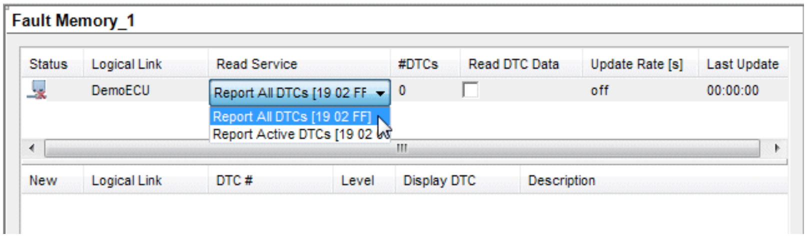
To display fault memory entries

1 Select a Fault Memory Instrument on a layout.

2 If the desired logical link is not visible in the instrument, select Instrument Properties from the context menu and activate it on the Logical Links page.

3 Select the logical links you want to use.

4 Select the read service to be used for reading the fault memory entries. The list contains all the available read services (default read services and/or read services specified in the XML configuration file).



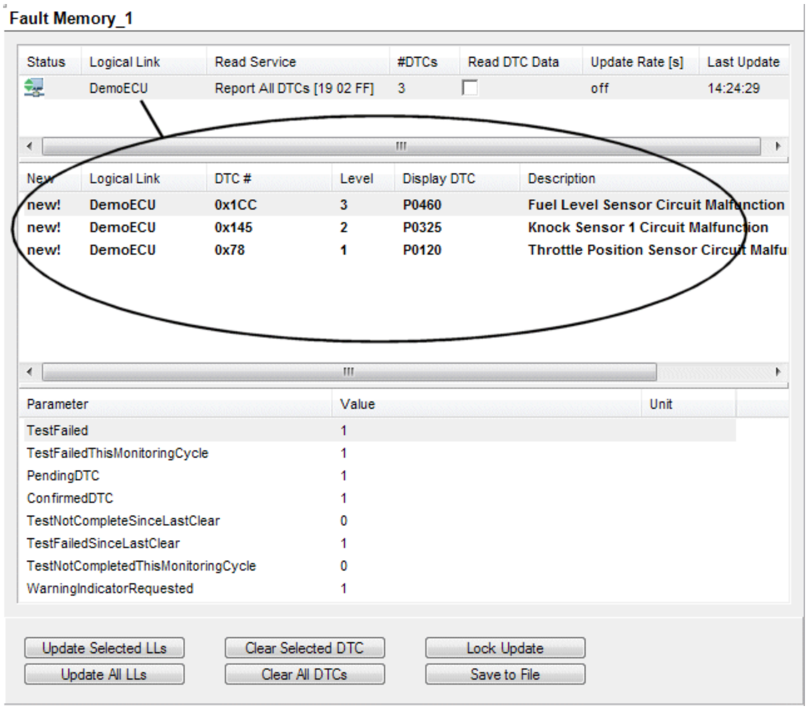
Fault Memory_1

Status	Logical Link	Read Service	#DTCs	Read DTC Data	Update Rate [s]	Last Update
	DemoECU	Report All DTCs [19 02 FF	0	□	off	00:00:00

New	Logical Link	DTC #	Level	Display DTC	Description

5 Click Update Selected LLs to read the DTC information for the selected logical links.

ControlDesk executes the selected read service. If the Read DTC Data checkbox is selected, it also executes the service selected for reading the environment data for each DTC. The DTC table displays the received diagnostic trouble codes. New DTCs are displayed in bold letters and marked by new! entries in the DTC table.



Fault Memory_1

Status	Logical Link	Read Service	#DTCs	Read DTC Data	Update Rate [s]	Last Update
	DemoECU	Report All DTCs [19 02 FF]	3	□	off	14:24:29

New	Logical Link	DTC #	Level	Display DTC	Description
new!	DemoECU	0x1CC	3	P0460	Fuel Level Sensor Circuit Malfunction
new!	DemoECU	0x145	2	P0325	Knock Sensor 1 Circuit Malfunction
new!	DemoECU	0x78	1	P0120	Throttle Position Sensor Circuit Malfu

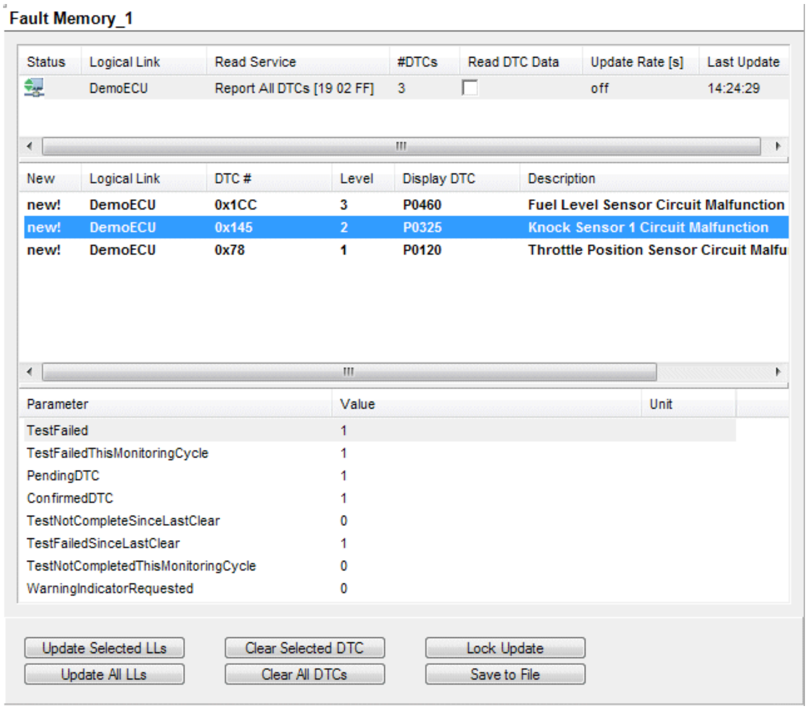
Parameter	Value	Unit
TestFailed	1	
TestFailedThisMonitoringCycle	1	
PendingDTC	1	
ConfirmedDTC	1	
TestNotCompleteSinceLastClear	0	
TestFailedSinceLastClear	1	
TestNotCompletedThisMonitoringCycle	0	
WarningIndicatorRequested	1	

Tip

If you want to read all DTC entries for all the logical links in the instrument, click Update All LLs.

6 In the DTC table, select a received diagnostic trouble code.

The DTC Data table displays status information on the selected diagnostic trouble code.



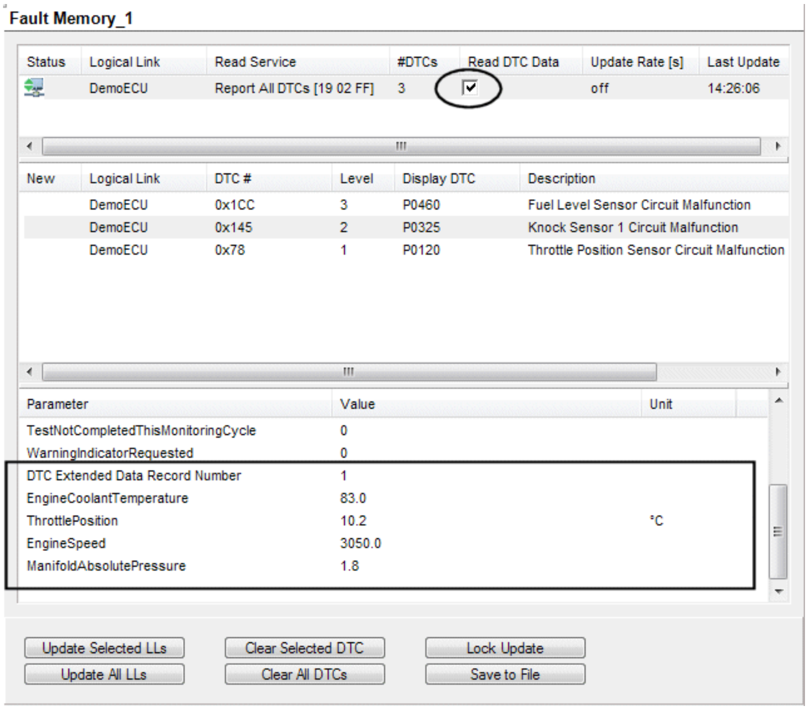
Fault Memory_1

Status	Logical Link	Read Service	#DTCs	Read DTC Data	Update Rate [s]	Last Update
	DemoECU	Report All DTCs [19 02 FF]	3	□	off	14:24:29

New	Logical Link	DTC #	Level	Display DTC	Description
new!	DemoECU	0x1CC	3	P0460	Fuel Level Sensor Circuit Malfunction
new!	DemoECU	0x145	2	P0325	Knock Sensor 1 Circuit Malfunction
new!	DemoECU	0x78	1	P0120	Throttle Position Sensor Circuit Malfu

Parameter	Value	Unit
TestFailed	1	
TestFailedThisMonitoringCycle	1	
PendingDTC	1	
ConfirmedDTC	1	
TestNotCompleteSinceLastClear	0	
TestFailedSinceLastClear	1	
TestNotCompletedThisMonitoringCycle	0	
WarningIndicatorRequested	0	

If **Read DTC Data** in the logical link table is activated, additional environment data is also displayed. If activated, updating the instrument takes more time, because not only the service for reading the ECU's fault memory is executed, but also the service for reading the environment data for each DTC.



Fault Memory_1

Status	Logical Link	Read Service	#DTCs	Read DTC Data	Update Rate [s]	Last Update
	DemoECU	Report All DTCs [19 02 FF]	3	☑	off	14:26:06

New	Logical Link	DTC #	Level	Display DTC	Description
	DemoECU	0x1CC	3	P0460	Fuel Level Sensor Circuit Malfunction
	DemoECU	0x145	2	P0325	Knock Sensor 1 Circuit Malfunction
	DemoECU	0x78	1	P0120	Throttle Position Sensor Circuit Malfunction

Parameter	Value	Unit
TestNotCompletedThisMonitoringCycle	0	
WarningIndicatorRequested	0	
DTC Extended Data Record Number	1	
EngineCoolantTemperature	83.0	
ThrottlePosition	10.2	°C
EngineSpeed	3050.0	
ManifoldAbsolutePressure	1.8	

Tip

ControlDesk lets you select the service for reading environment data via the **Properties** controlbar related to the instrument (see Logical Links Properties (ControlDesk Instrument Handling 🕮)), if several services are available.

7 If you want to store the content of the Fault Memory Instrument, click **Save to File**.

8 If you want to delete all DTCs from the fault memory of the ECUs that are referenced by the logical links, click **Clear All DTCs**.

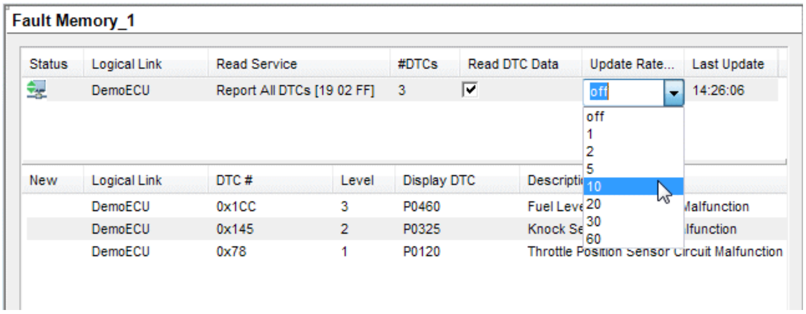
Some ECUs allow the deletion of a selected DTC. In this case you can select a DTC from the DTC table and delete it via **Clear Selected DTC**.

ControlDesk first executes the fault clear service to delete the trouble code entries from the fault memory and then the fault read service to update the display in the instrument.

Tip

ControlDesk lets you select the services for clearing single fault memory entries and all fault memory entries via the **Properties** controlbar (see Logical Links Properties (ControlDesk Instrument Handling 🕮)), if several services are available.

9 You can select an update rate to update a logical link repeatedly.



Fault Memory_1

Status	Logical Link	Read Service	#DTCs	Read DTC Data	Update Rate...	Last Update
	DemoECU	Report All DTCs [19 02 FF]	3	☑	off	14:26:06

New	Logical Link	DTC #	Level	Display DTC	Descripti
	DemoECU	0x1CC	3	P0460	Fuel Leve alfunction
	DemoECU	0x145	2	P0325	Knock Se lfunction
	DemoECU	0x78	1	P0120	Throttle Position Sensor Circuit Malfunction

10 If you want to 'freeze' the content of the instrument click **Lock Update**. Updates of DTC information are then not shown in this instrument until you click **Unlock Update**.

Result

You have displayed fault memory entries.

Tip

- ControlDesk can automatically set a bookmark during a measurement or recording if the number of DTCs for a logical link changes. These 'diagnostic trouble code' bookmarks can then be visualized in a Time Plotter. Refer to Edit Bookmark Settings (ControlDesk Measurement and Recording 🕮).
- You can use the diagnostic trouble code bookmarks as predefined trigger conditions, for example, to start a triggered recording with a specified pretriggering time when the bookmark is set. Refer to How to Define a Trigger Rule for Triggered Recording (ControlDesk Measurement and Recording 🕮).
- You can use DTC measurements as trigger variables. Compared with diagnostic trouble code bookmarks, DTC measurements allow you to define more specific or individual DTC trigger rules. Refer to Performing DTC Measurements on page 81. DTC measurements and DTC bookmarks are not synchronized.
- ControlDesk comes with an ECU Diagnostics demo project, which lets you simulate communicating with an ECU via diagnostic interface. Refer to ECU Diagnostics Demo on page 13.

Related topics

Basics

Reading the Fault Memory of an ECU..56

References

Copy (Fault Memory Instrument) (ControlDesk Instrument Handling 📖)
Fault Memory Instrument (ControlDesk Instrument Handling 📖)

Configuring and Executing Diagnostic Communication Objects

Introduction

ECU diagnostics with ControlDesk are completely based on Open Diagnostic Data Exchange (ODX), the ASAM MCD-2 D (ISO 22901) diagnostics standard. The ODX database contains the description of the diagnostic services implemented on the ECU, and the settings for communication between ControlDesk and the ECU.

ControlDesk provides the Diagnostics Instrument to communicate with the ECU via diagnostic protocol.

Communication objects

The ODX database describes different diagnostic communication objects to communicate with an ECU.

Diagnostic service A service implemented on the ECU as a basic diagnostic communication element. Communication is performed by selecting a service, configuring its parameters, executing it, and receiving the ECU results. When a service is executed, a defined request is sent to the ECU and the ECU answers with a specific response.

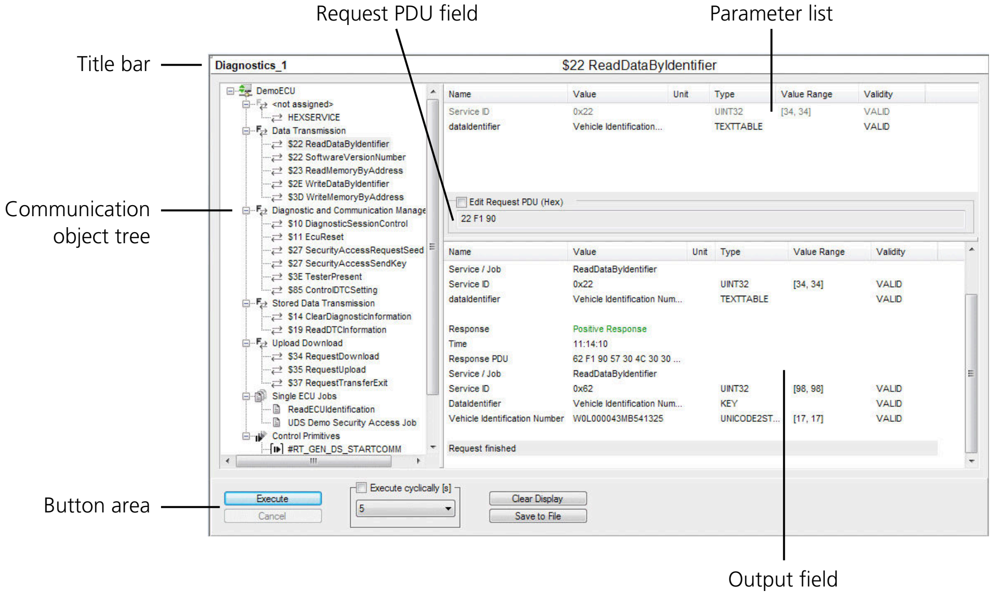
Diagnostic job (often called Java job) Programmed sequence that is usually built from a sequence of the diagnostic service. A diagnostic job is either a single-ECU job or a multiple-ECU job, depending on whether it communicates with one ECU or multiple ECUs.

Diagnostic jobs can load DLL files, for example, Seed&Key DLL files to get security access for ECU parameter calibration. ControlDesk only supports 64-bit DLL files in connection with ECU diagnostic jobs.

Control primitive A special diagnostic communication object for changing communication states or protocol parameters, or for identifying (ECU) variants.

Diagnostics Instrument

The Diagnostics Instrument consists of several parts.



Name	Value	Unit	Type	Value Range	Validity
Service ID	0x22		UINT32	[34, 34]	VALID
dataIdentifier	Vehicle Identification...		TEXTTABLE		VALID

□ Edit Request PDU (Hex)

22 F1 90

Name	Value	Unit	Type	Value Range	Validity
Service / Job	ReadDataByIdentifier				
Service ID	0x22		UINT32	[34, 34]	VALID
dataIdentifier	Vehicle Identification Num...		TEXTTABLE		VALID
Response	Positive Response				
Time	11:14:10				
Response PDU	62 F1 90 57 30 4C 30 30 ...				
Service / Job	ReadDataByIdentifier				
Service ID	0x62		UINT32	[98, 98]	VALID
DataIdentifier	Vehicle Identification Num...		KEY		VALID
Vehicle Identification Number	W0L000043MB541325		UNICODE2ST...	[17, 17]	VALID

Request finished

□ Execute cyclically [s]: 5

Communication object tree Displays all the diagnostic communication objects (diagnostic services, diagnostic jobs, and control primitives) that are available for the logical links, sorted alphabetically by their names in ascending order. The service IDs are displayed as prefixes to the names of the communication objects, if this is specified in the **Properties** controlbar of the instrument (refer to Tree View Properties (ControlDesk Instrument Handling 🕮)).

Parameter list Displays the parameters that are available for the selected communication object.

Request PDU field Displays the request PDU that results from the service selection in the communication object tree and the parameter configuration in the parameter list, or lets you edit the request PDU before the service is executed.

Output field Displays the results (and intermediate results) of a communication object call.

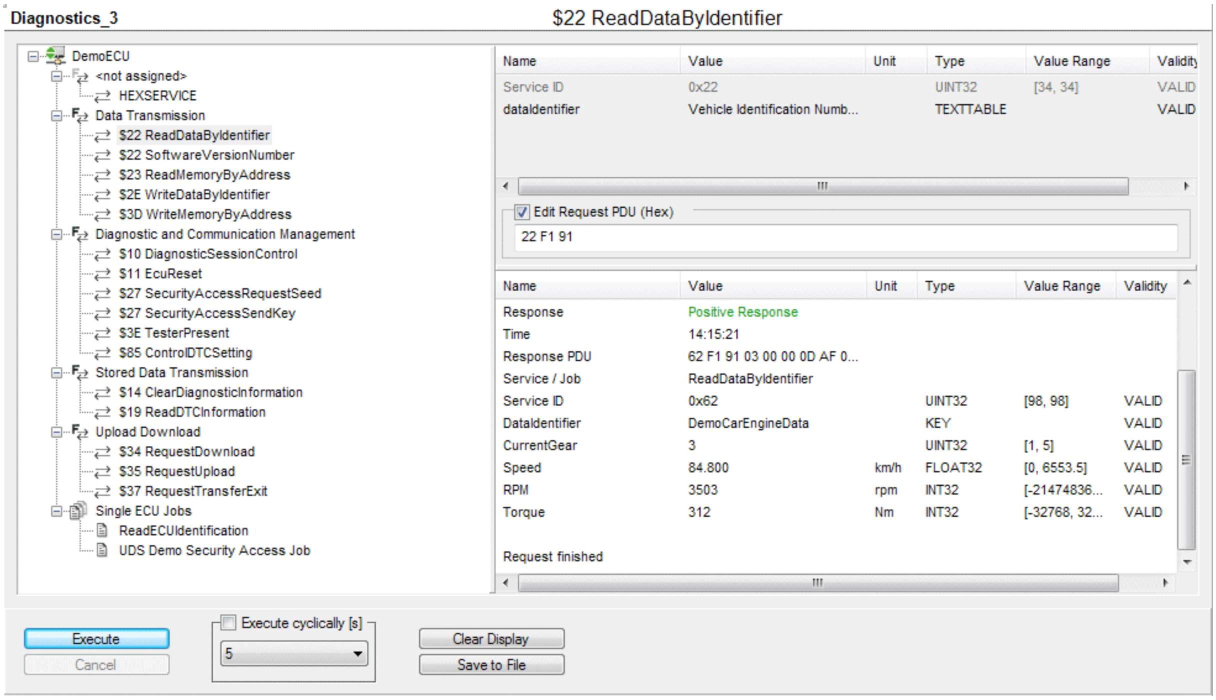
ECU communication via PDU information

Via the Diagnostics Instrument, you can perform communication with the ECU based on raw protocol data units (PDUs). When you select a service and configure its parameters, the Diagnostics Instrument displays the resulting request PDU. If desired, you can edit the request PDU before sending a request to the ECU. The ECU response is taken as the response to the service, that is, ControlDesk not only displays the response PDU in hexadecimal notation, but also interprets the ECU response, if the ODX database contains the relevant information.

Tip

Editing the request PDU allows you, for example, to send a PDU to the ECU even if it is not configured in the parameter list. This lets you violate value limits and computation methods for parameters, for example.

As an example, the illustration below shows the Diagnostics Instrument displaying an interpreted ECU response.



Diagnostics_3 — $22 ReadDataByIdentifier

- DemoECU
  - <not assigned>
    - HEXSERVICE
  - Data Transmission
    - $22 ReadDataByIdentifier
    - $22 SoftwareVersionNumber
    - $23 ReadMemoryByAddress
    - $2E WriteDataByIdentifier
    - $3D WriteMemoryByAddress
  - Diagnostic and Communication Management
    - $10 DiagnosticSessionControl
    - $11 EcuReset
    - $27 SecurityAccessRequestSeed
    - $27 SecurityAccessSendKey
    - $3E TesterPresent
    - $85 ControlDTCSetting
  - Stored Data Transmission
    - $14 ClearDiagnosticInformation
    - $19 ReadDTCInformation
  - Upload Download
    - $34 RequestDownload
    - $35 RequestUpload
    - $37 RequestTransferExit
  - Single ECU Jobs
    - ReadECUIdentification
    - UDS Demo Security Access Job

Name	Value	Unit	Type	Value Range	Validity
Service ID	0x22		UINT32	[34, 34]	VALID
dataIdentifier	Vehicle Identification Numb...		TEXTTABLE		VALID

☑ Edit Request PDU (Hex)

22 F1 91

Name	Value	Unit	Type	Value Range	Validity
Response	Positive Response				
Time	14:15:21				
Response PDU	62 F1 91 03 00 00 0D AF 0...				
Service / Job	ReadDataByIdentifier				
Service ID	0x62		UINT32	[98, 98]	VALID
DataIdentifier	DemoCarEngineData		KEY		VALID
CurrentGear	3		UINT32	[1, 5]	VALID
Speed	84.800	km/h	FLOAT32	[0, 6553.5]	VALID
RPM	3503	rpm	INT32	[-21474836...	VALID
Torque	312	Nm	INT32	[-32768, 32...	VALID

Request finished

☐ Execute cyclically [s]: 5

ControlDesk also provides a generic diagnostic hex service for ECU communication via raw PDU information. The hex service lets you configure the request in hexadecimal notation, and the hex service response is also in hexadecimal notation. However, ControlDesk does not interpret the response PDU of the hex service as a response to any service. Refer to How to Configure and Execute Diagnostic Communication Objects on page 70.

Related topics

Basics

Conventions in Connection with ODX Databases.. 22

HowTos

How to Configure and Execute Diagnostic Communication Objects... 70

How to Configure and Execute Diagnostic Communication Objects

Objective

With the Diagnostics Instrument, you can communicate with an ECU via diagnostic services, diagnostic jobs, and control primitives.

Basics

For basic information on communicating with an ECU via diagnostic interface using the Diagnostics Instrument, refer to Configuring and Executing Diagnostic Communication Objects on page 67.

Changing views

Via the context menu, you can change the views in the communication object tree (hierarchical vs. flat view) and in the output field (automatic clearing, displaying results only) of the Diagnostics Instrument. These settings are also available in the instrument's properties dialog.

In addition, this dialog lets you specify the communication object types to be displayed in the instrument.

Default parameter values

Via the context menu of the parameter list and the request PDU field, you can activate default values for the parameters. The default value of a parameter can be defined in the diagnostics database. If not, standard default values are used (0 for numbers, empty string for strings, for example).

Display of complex data structures

The parameter list of the Diagnostics Instrument displays all the parameters that are available for the selected communication object. Since parameter names must be unique only within a hierarchy level, identifying parameters via their names can cause problems. To facilitate parameter handling, you can select to display the superordinate complex data of parameters together with the parameter names. For example, if a parameter is part of a structure, field or multiplexer (= complex data type), the name of this complex data is displayed with the parameter name in the Diagnostics Instrument's parameter list and output field (if available). If the parameter is part of a hierarchy of multiple complex data elements, the entire hierarchy structure is displayed.

You can enable or disable the display of complex data structures in the ControlDesk Options dialog. Refer to Diagnostics Management Page on page 127.

The whole path (short name path) is used for identification purposes, for example, when setting values in AutomationDesk or via ControlDesk tool automation. The Diagnostics Instrument lets you copy the short name path of parameters. Refer to Copy (Diagnostics Instrument) (ControlDesk Instrument Handling 🕮).

Generic diagnostic hex service

ControlDesk provides the generic diagnostic hex service named '#RT_GEN_DS_HEXSERVICE' in the Diagnostics Instrument. The service can be used to communicate with the ECU via raw PDU (protocol data unit) information. It is not defined in the ODX database, and it is not assigned to a functional class. The hex service is generated by the diagnostic server for each logical link.

The generic hex service contains one request and one response parameter, each of BYTEFIELD type. The maximum length of the parameter depends on the protocol and physical connection used:

Physical Connection	Maximum Length
K-Line	255 bytes
CAN	4096 bytes
Simulation	1024 bytes

You can configure the request in hexadecimal notation, and the service response is also in hexadecimal notation.

Preconditions

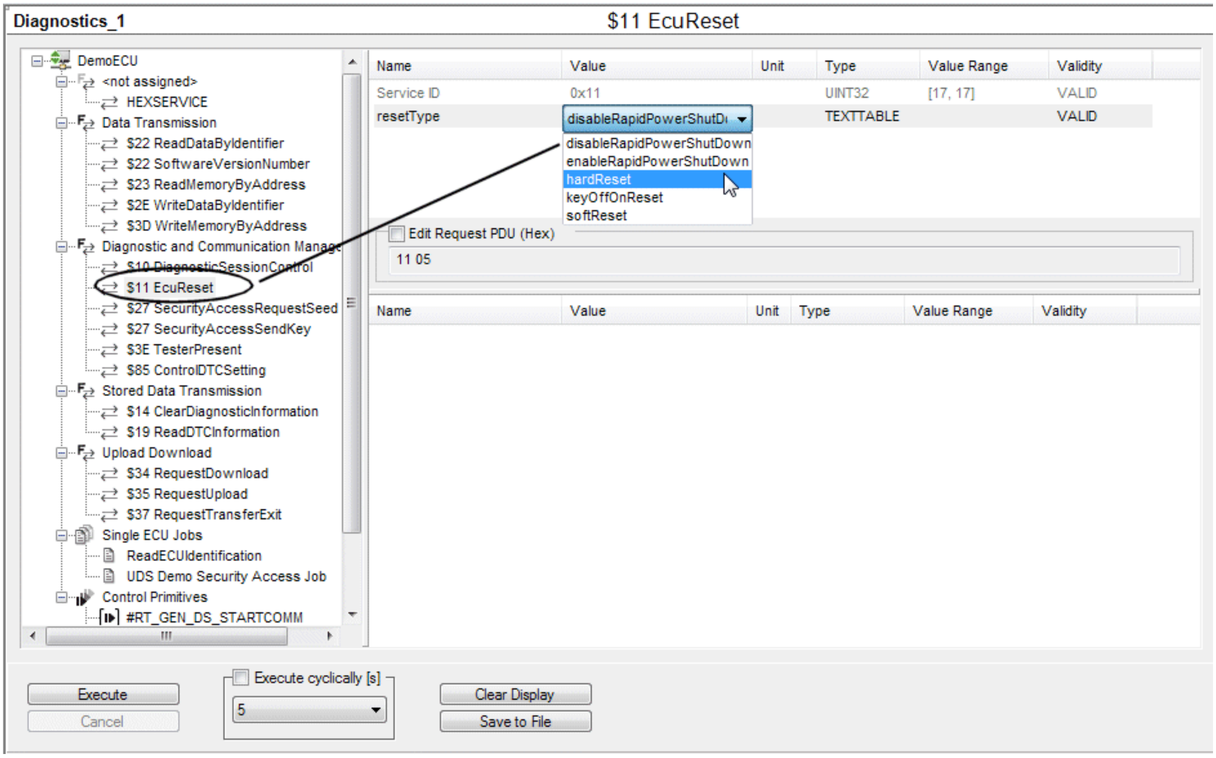
- When performing ECU diagnostics with ControlDesk, you have to observe some conventions in connection with the ODX database you use. Refer to Conventions in Connection with ODX Databases on page 22.
- A Diagnostics Instrument must be placed on a layout. Refer to How to Place an Instrument for Diagnostic Tasks on a Layout on page 29.
- To execute communication objects, online calibration must be started. Refer to How to Start and Stop Online Calibration (ControlDesk Calibration and Data Set Management 📖).

Method

To communicate with an ECU via diagnostic interface

1. Select a Diagnostics Instrument on a layout.
2. If the desired logical link is not visible in the communication object tree, select Instrument Properties from the context menu and activate it on the Logical Links page.
3. Select a communication object in the tree and configure its parameters (if available) in the parameter list.

The resulting request PDU is displayed in the request PDU field.



Diagnostics_1

$11 EcuReset

Name	Value	Unit	Type	Value Range	Validity
Service ID	0x11		UINT32	[17, 17]	VALID
resetType	disableRapidPowerShutDown		TEXTTABLE		VALID

□ Edit Request PDU (Hex)

11 05

Name	Value	Unit	Type	Value Range	Validity

□ Execute cyclically [s]: 5

Tip

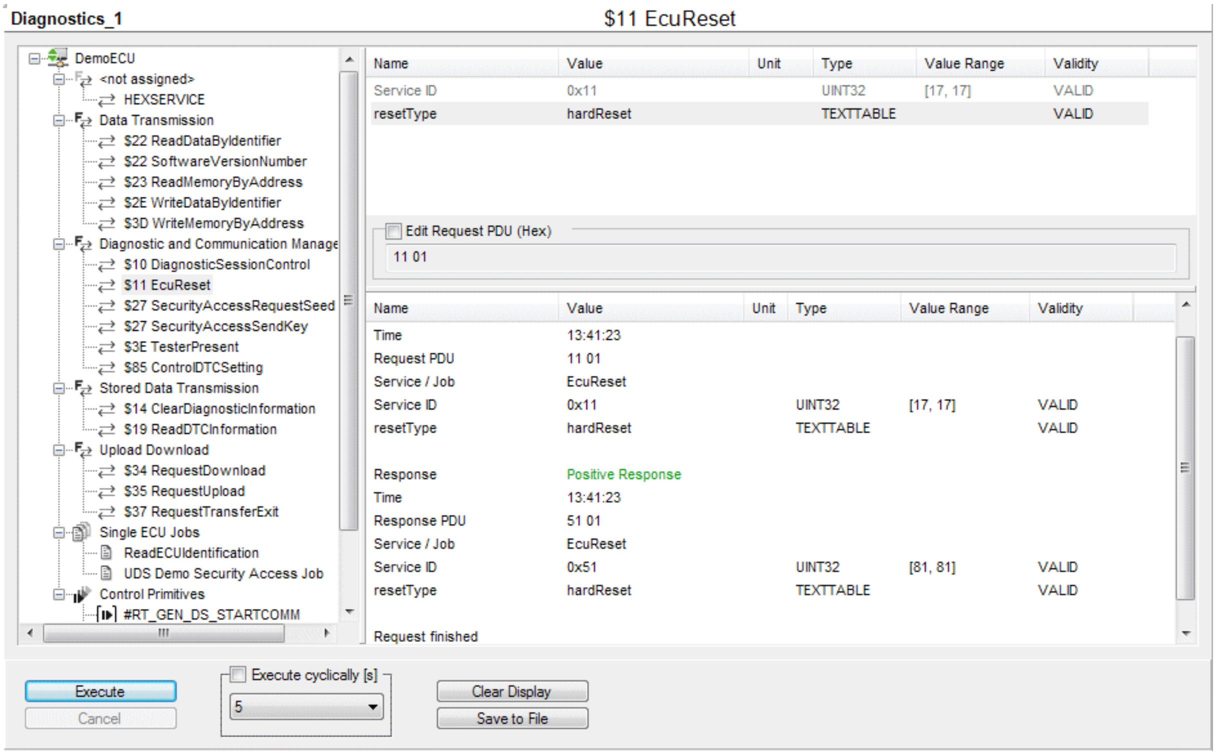
If a service is selected in the tree, you can alternatively enter a request PDU in the Request PDU edit field. Select the Edit Request PDU (Hex) checkbox to make the request PDU field writable, and specify the request PDU manually. Press **Enter** to confirm your PDU settings.

In cases where editing the request PDU is not possible (for example, if the service is executed cyclically or if an object other than service is selected in the tree), the Edit Request PDU (Hex) checkbox is disabled.

4 Click Execute to execute the communication object once.

For diagnostic services, you can alternatively select a cycle time and activate Execute cyclically [s].

The results are displayed in the instrument's output field.



Name	Value	Unit	Type	Value Range	Validity
Service ID	0x11		UINT32	[17, 17]	VALID
resetType	hardReset		TEXTTABLE		VALID

☐ Edit Request PDU (Hex)

11 01

Name	Value	Unit	Type	Value Range	Validity
Time	13:41:23				
Request PDU	11 01				
Service / Job	EcuReset				
Service ID	0x11		UINT32	[17, 17]	VALID
resetType	hardReset		TEXTTABLE		VALID
Response	Positive Response				
Time	13:41:23				
Response PDU	51 01				
Service / Job	EcuReset				
Service ID	0x51		UINT32	[81, 81]	VALID
resetType	hardReset		TEXTTABLE		VALID
Request finished					

☐ Execute cyclically [s]

5

The output field displays validity, data type, and value range information for the single response parameters if available. Validity information on the valid value ranges is defined in the ODX database. Response parameters with values marked as invalid in the ODX database are displayed in red.

The response types are colored to indicate whether the ECU sends a positive response (green) or negative response (red). There is one exception: When the generic diagnostic hex service is used, the response type is always displayed in green. However, ControlDesk does not interpret the response PDU of the hex service as a response to any service.

Tip

For DTCs, the output field shows additional information specified in the ODX database: the display name of the DTC, the description text of the DTC, and the DTC level.

The Diagnostics Instrument does not require an appropriate ODX database or XML configuration. This allows you to get DTC information even if you work with an ODX database which is not yet fully adjusted, for example, during a start-up in a hardware-in-the-loop (HIL) environment.

5 If you want to store the content of the output field, click **Save to File**.

6 If you want to remove all responses from the output field, click **Clear Display**.

7 You can copy entries in the communication object tree, parameter list or output field, or specific information belonging to the entries, to the Clipboard. This can be useful to get the name and path of a parameter for tool automation purposes, for example.

ControlDesk provides several copy commands (see Copy (Diagnostics Instrument) (ControlDesk Instrument Handling 🕮)). Below are some examples:

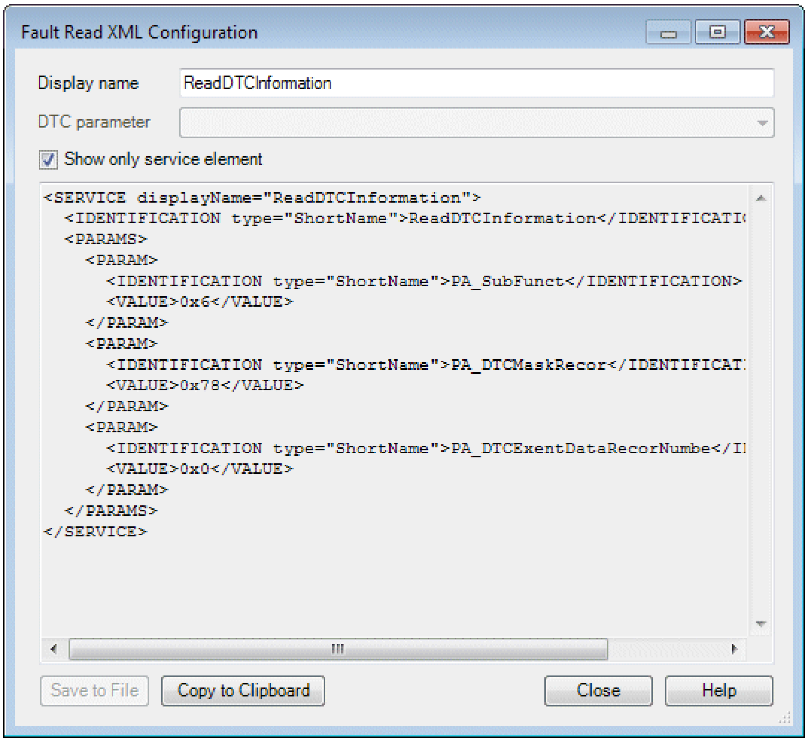
- If you want to copy the request parameters belonging to the executed communication object for further use with tool automation or in AutomationDesk, select **Copy – Tool Automation Format (Python)** or **Copy – AutomationDesk Format** from the parameter list's context menu. ControlDesk then copies the short names and values of the writable request parameters to the Clipboard in the appropriate format.
- If you want to copy the whole contents of the output field to the Clipboard, select **Copy – Entire Output Field** from the output field's context menu.
- If you want to copy a part of the output field contents, select the rows to be copied in the output field, and then call the **Copy – Selected Row(s)** command from the context menu.

Tip

You can use **`Shift`** or **`Ctrl`** for multiselection.

8 ControlDesk allows you to make a fault memory service that is parameterized in the Diagnostics Instrument available as an XML configuration to be used in an XML configuration file in ControlDesk projects. After you configured the service in the parameter list, select the appropriate **Create XML Configuration** command from the parameter list's context menu. ControlDesk provides commands for creating XML service configurations for reading fault memory, reading environment data, and clearing single or all fault memory entries. Their availability depends on the diagnostic service type the **Create XML Configuration** command is invoked for.

ControlDesk opens an XML Configuration dialog for you to specify the configuration settings that are relevant for the content and usage of the created XML configuration.



Display name: ReadDTCInformation

DTC parameter:

☑ Show only service element

```
<SERVICE displayName="ReadDTCInformation">
  <IDENTIFICATION type="ShortName">ReadDTCInformation</IDENTIFICATI
  <PARAMS>
    <PARAM>
      <IDENTIFICATION type="ShortName">PA_SubFunct</IDENTIFICATION>
      <VALUE>0x6</VALUE>
    </PARAM>
    <PARAM>
      <IDENTIFICATION type="ShortName">PA_DTCMaskRecor</IDENTIFICAT
      <VALUE>0x78</VALUE>
    </PARAM>
    <PARAM>
      <IDENTIFICATION type="ShortName">PA_DTCExentDataRecorNumbe</I
      <VALUE>0x0</VALUE>
    </PARAM>
  </PARAMS>
</SERVICE>
```

For example, you can specify a name to display in the Fault Memory Instrument for the service, and must select whether to create the XML configuration for the service element only or to create a ready-to-use XML configuration file containing the service XML configuration. You can then copy the service configurations to the Clipboard or save them to a file.

Result

You have performed a diagnostic communication by parameterizing and executing a communication object and monitoring the ECU responses.

Tip

ControlDesk comes with an ECU Diagnostics demo project, which lets you simulate communicating with an ECU via diagnostic interface. Refer to ECU Diagnostics Demo on page 13.

Related topics

Basics

Basics of ECU Diagnostics with ControlDesk 10
Configuring and Executing Diagnostic Communication Objects 67

Workflow for Creating an XML Configuration via the Diagnostics Instrument........................ 120

References

Copy (Diagnostics Instrument) (ControlDesk Instrument Handling 🕮)
Create XML Configuration (ControlDesk Instrument Handling 🕮)
Diagnostics Instrument (ControlDesk Instrument Handling 🕮)

Measuring and Calibrating Variables via the ECU Diagnostics Device

Introduction

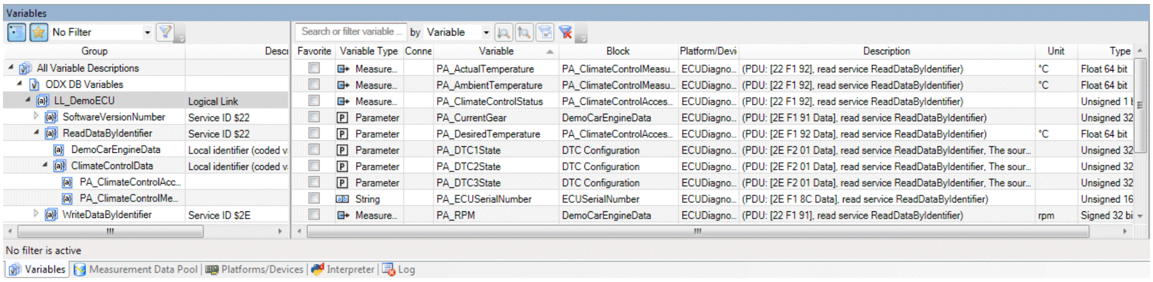
You can measure and calibrate diagnostics variables via the ECU Diagnostics device. This means that diagnostic protocols can be used for measurement and calibration. A variable description has to be generated from the ODX database specified for the device for this.

You can perform measurement and calibration via the ECU Diagnostics device in the same way as for other devices in ControlDesk, using ControlDesk's measurement and calibration instruments. For details, refer to Measuring Data (ControlDesk Measurement and Recording 🕮) and Calibrating Parameters (ControlDesk Calibration and Data Set Management 🕮).

Diagnostics variables

To measure and calibrate ECU variables via ControlDesk's ECU Diagnostics device, you need a variable description. The ControlDesk-internal variable description for most application interfaces is generated from a standard data format such as A2L. For the ECU Diagnostics device, it is generated from the ODX database. All the measurement variables and parameters generated from an ODX database are called *diagnostics variables*.

After generation, ControlDesk displays the diagnostics variables in the **Variables** controlbar.



Group	Desc
All Variable Descriptions	
ODX DB Variables	
LL_DemoECU	Logical Link
SoftwareVersionNumber	Service ID $22
ReadDataByIdentifier	Service ID $22
DemoCarEngineData	Local identifier (coded v
ClimateControlData	Local identifier (coded v
PA_ClimateControlAcc...	
PA_ClimateControlMe...	
WriteDataByIdentifier	Service ID $2E

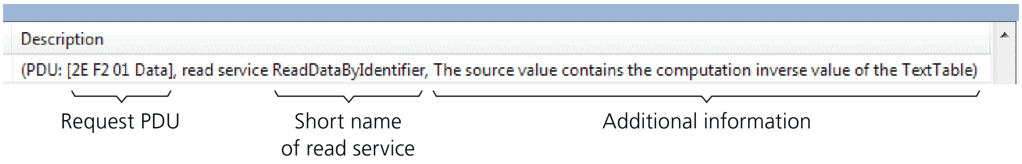
Favorite	Variable Type	Conne	Variable	Block	Platform/Devi	Description	Unit	Type
□	Measure...		PA_ActualTemperature	PA_ClimateControlMeasu...	ECUDiagno...	(PDU: [22 F1 92], read service ReadDataByIdentifier)	°C	Float 64 bit
□	Measure...		PA_AmbientTemperature	PA_ClimateControlMeasu...	ECUDiagno...	(PDU: [22 F1 92], read service ReadDataByIdentifier)	°C	Float 64 bit
□	Measure...		PA_ClimateControlStatus	PA_ClimateControlAcces...	ECUDiagno...	(PDU: [22 F1 92], read service ReadDataByIdentifier)		Unsigned 1 b
□	Parameter		PA_CurrentGear	DemoCarEngineData	ECUDiagno...	(PDU: [2E F1 91 Data], read service ReadDataByIdentifier)		Unsigned 32
□	Parameter		PA_DesiredTemperature	PA_ClimateControlAcces...	ECUDiagno...	(PDU: [2E F1 92 Data], read service ReadDataByIdentifier)	°C	Float 64 bit
□	Parameter		PA_DTC1State	DTC Configuration	ECUDiagno...	(PDU: [2E F2 01 Data], read service ReadDataByIdentifier, The sour...		Unsigned 32
□	Parameter		PA_DTC2State	DTC Configuration	ECUDiagno...	(PDU: [2E F2 01 Data], read service ReadDataByIdentifier, The sour...		Unsigned 32
□	Parameter		PA_DTC3State	DTC Configuration	ECUDiagno...	(PDU: [2E F2 01 Data], read service ReadDataByIdentifier, The sour...		Unsigned 32
□	String		PA_ECUSerialNumber	ECUSerialNumber	ECUDiagno...	(PDU: [2E F1 8C Data], read service ReadDataByIdentifier)		Unsigned 16
□	Measure...		PA_RPM	DemoCarEngineData	ECUDiagno...	(PDU: [22 F1 91], read service ReadDataByIdentifier)	rpm	Signed 32 bi

The type of the generated diagnostics variable depends on whether it is defined in a diagnostics read service and write service:

- Diagnostics variables that are defined only in a read service are represented as *measurement variables* in ControlDesk.

- Diagnostics variables that are defined in a read service and in a write service are represented as *parameters* in ControlDesk.

In the variable list of the **Variables** controlbar, the description text of a diagnostics variable provides information on the associated request PDU, the name of the read service (only for parameters), and further additional descriptive information, if available. This information is displayed in parentheses. You can use it to filter the variable list.

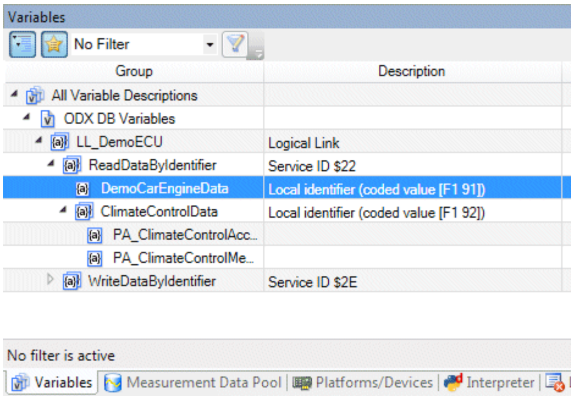


Description

(PDU: [2E F2 01 Data], read service ReadDataByIdentifier, The source value contains the computation inverse value of the TextTable)

The tree view of the **Variables** controlbar displays the variables according to the structure in the related variable description. Descriptions for some block group types help you identify the variables. These are some examples:

- For a logical link block group, the description text is 'Logical link'.
- The description of a service block group displays the associated service ID.
- The description of the block group of a local identifier in a diagnostic service shows the coded value of the associated service parameter.



Group	Description
All Variable Descriptions	
ODX DB Variables	
LL_DemoECU	Logical Link
ReadDataByIdentifier	Service ID $22
DemoCarEngineData	Local identifier (coded value [F1 91])
ClimateControlData	Local identifier (coded value [F1 92])
PA_ClimateControlAcc...	
PA_ClimateControlMe...	
WriteDataByIdentifier	Service ID $2E

No filter is active

You can also perform DTC measurements, which makes diagnostic trouble codes visible in ControlDesk instruments other than the Fault Memory and the Diagnostics Instrument. Refer to Performing DTC Measurements on page 81.

Basics on measuring and calibrating diagnostics variables

To perform measurement and calibration via the ECU Diagnostics device, the following preconditions must be met:

- The ODX database, which is used to describe the ECU, must contain a read service for measurement and a write service for calibration. There are certain conventions in connection with the ODX database for this purpose (see Default diagnostic services for measuring and calibrating variables according to ODX semantics on page 78).
- The ODX database with configured read service and write service must be added to the ECU Diagnostics device. For instructions, refer to How to Configure an ECU Diagnostics Device on page 36.

- You must enable ControlDesk to generate the variable description from the ODX database. Select the **Create variable description for selected logical links** checkbox during configuration of the ECU Diagnostics device for this purpose. Refer to Select Logical Links Dialog (ControlDesk Platform Management 🕮).

  You can specify to also use information from the protocol specification for variable generation. Select the **Use additional protocol information** checkbox for this. If enabled, ControlDesk analyzes all read services and write services in the ODX database on the basis of the protocol-specific identification of services and generates all possible variables.

ECU diagnostics mode To perform parameter calibration, the ECU must be in a diagnostics mode which allows parameters to be written (for example, in ECU development mode). For further information, refer to Getting Security Access for Parameter Calibration on page 80.

Default diagnostic services for measuring and calibrating variables according to ODX semantics

You can perform measurements and calibration in ControlDesk using default diagnostic services identified by ODX semantics.

Note

However, the default diagnostic services according to ODX semantics described below are available in ControlDesk only if the following condition is fulfilled:
Both the file list in the **Select ODX Files** dialog specified during configuration of your ECU Diagnostics device and the `.\Config` folder of your ControlDesk installation do not contain an XML configuration file with service configuration settings for the `MEASUREMENT` and `CALIBRATION` functions. For information on the conventions when working with a local and/or global XML configuration file, refer to Basics of the XML Configuration File on page 93.

There are some conventions to meet when defining a read/write service for the diagnostics device:

- The read service must be classified with the `SEMANTIC` attribute with the value '`CURRENTDATA`'.
- The write service must be classified with the `SEMANTIC` attribute with the value '`CALIBRATION`'.
- The read service and the write service must both contain a request parameter and a response parameter classified with the `SEMANTIC` attribute with the value '`DATA-ID`' as the local ID.
- Each measurement variable/parameter that corresponds to a response parameter in a read service or to a request parameter in a write service must be classified with the `SEMANTIC` attribute with the value '`DATA`'.

If several measurement variables/parameters are defined in a service and if they are united, for example, in a TABLE-STRUCT element, it is sufficient to specify the `SEMANTIC` attribute for the element, as all underlying measurement variables/parameters inherit this attribute automatically. However, if you want to use individual diagnostics variables of a structure element in ControlDesk, make sure that the structure element itself does not contain the `SEMANTIC` attribute.

- Each parameter in a write service must have a corresponding measurement variable in a read service. The read service and the write service must have the same structure. The request parameter and the response parameter that are classified with the `SEMANTIC` attributes '`DATA-ID`' or '`DATA`' must have identical names in the read service and the write service.

Tip

For information on the default diagnostic services for DTC measurements, refer to Performing DTC Measurements on page 81.

Default diagnostic services for measuring and calibrating variables according to protocol-specific service identification

You can perform measurements and calibration in ControlDesk using default diagnostic services according to the protocol-specific identification of services.

If your ODX database contains suitable data, ControlDesk enables measurement and calibration for specific diagnostic services, regardless of whether XML configurations for measurement and calibration are available or not. These diagnostic services are identified via service IDs and byte positions.

To make the default services according to the protocol-specific service identification available in ControlDesk, you must specify to use additional information from the protocol specification for generating variables. Refer to Select Logical Links Dialog (ControlDesk Platform Management 📖).

Configuring further services for measurement and calibration

If you want to use services others than the default services for performing measurements and calibration in ControlDesk, you can specify further diagnostic services in an XML configuration file. For further information, refer to Configuring Services for Measurement and Calibration in the XML Configuration File on page 110.

Related topics

Basics

Basics of the XML Configuration File	93
Conventions in Connection with ODX Databases	22
Requirements for Measurement and Calibration via the ECU Diagnostics Device	35

Getting Security Access for Parameter Calibration

Introduction

To perform parameter calibration, the ECU must be in a diagnostics mode which allows parameters to be written. To get security access, ControlDesk uses a specific diagnostic job specified in the ODX database.

ECU diagnostics mode

To perform parameter calibration, the ECU must be in a diagnostics mode which allows parameters to be written (for example, in ECU development mode). If writing parameters is not permitted for an ECU in the standard diagnostics mode or if an additional security access (Seed&Key) is needed, the ECU must be prepared for calibration access by changing the ECU mode or providing security access.

Default diagnostic job for getting security access according to ODX semantics

ControlDesk uses a default diagnostic job identified by ODX semantics.

> **Note**
>
> The default diagnostic job according to ODX semantics described below is available in ControlDesk only if the following condition is fulfilled:
> Both the file list in the **Select ODX Files** dialog specified during configuration of your ECU Diagnostics device and the `.\Config` folder of your ControlDesk installation do not contain an XML configuration file with service configuration settings for the `SECURITY_ONLINE` function. For information on the conventions when working with a local and/or global XML configuration file, refer to Basics of the XML Configuration File on page 93.

The default diagnostic job according to ODX semantics is identified as follows:

To get security access, the ODX database must contain a specific diagnostic job classified with the SEMANTIC attribute. For identification, the job must be classified as `SEMANTIC = SECURITY`. It must contain all the information needed for parameter calibration.

When online calibration is started, ControlDesk checks whether such a diagnostic job is contained in the ODX database. If it is, the diagnostic job is executed. If the job unlocks the ECU, calibrating parameters is possible. If the ODX database does not contain an appropriate diagnostic job, a warning is displayed.

Note

Whether and when the check is performed (with every online calibration start for the device, or only if a variable description was generated from the ODX database for the ECU Diagnostics device) depends on the settings made on the **Connection Check** page in the **Active Logical Links** dialog. Refer to Active Logical Links Dialog (ControlDesk Platform Management 📖).

Configuring further jobs for getting security access

If you want to use a job different from the default job for getting security access when performing calibration in ControlDesk, you can specify a further diagnostic job in an XML configuration file. For further information, refer to Configuring Services for Getting Security Access in the XML Configuration File on page 114.

Related topics

Basics

Basics of the XML Configuration File.. 93
Conventions in Connection with ODX Databases.. 22

Performing DTC Measurements

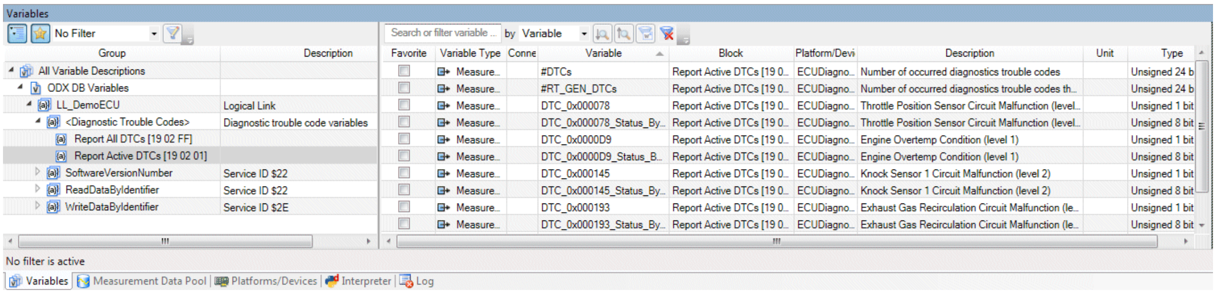
Introduction

Performing diagnostic trouble code (DTC) measurements makes diagnostic trouble codes visible in ControlDesk instruments other than the Fault Memory and the Diagnostics Instrument. This allows you to use changes on DTCs as trigger inputs for recordings, for example. The variable description of the ECU Diagnostics device must contain DTC measurement variables for this. ControlDesk uses fault memory read services specified in the ODX database to perform DTC measurements.

DTC variables

To perform diagnostic trouble code (DTC) measurements, the variable description of the ECU Diagnostics device must contain DTC measurement variables. DTC variables are available if you enabled their generation in the **Select Logical Links** dialog during device configuration, or via the **Properties** controlbar (refer to Logical Link Selection Properties (ControlDesk Platform Management 📖)).

The generated DTC variables are displayed in the **Variables** controlbar in the `<Diagnostic Trouble Codes>` node located below the logical link nodes. There is one specific node for each configured fault read service. The configured services are the services that are displayed as fault memory read services in ControlDesk's Fault Memory instrument. Every diagnostic trouble code is read by every read service. If no read service is available, no DTC variables are generated.



Variables

No Filter

Group	Description
All Variable Descriptions	
ODX DB Variables	
LL_DemoECU	Logical Link
<Diagnostic Trouble Codes>	Diagnostic trouble code variables
Report All DTCs [19 02 FF]	
Report Active DTCs [19 02 01]	
SoftwareVersionNumber	Service ID $22
ReadDataByIdentifier	Service ID $22
WriteDataByIdentifier	Service ID $2E

Search or filter variable ... by Variable

Favorite	Variable Type	Conne	Variable	Block	Platform/Devi	Description	Unit	Type
□	Measure...		#DTCs	Report Active DTCs [19 0...	ECUDiagno...	Number of occurred diagnostics trouble codes		Unsigned 24 b
□	Measure...		#RT_GEN_DTCs	Report Active DTCs [19 0...	ECUDiagno...	Number of occurred diagnostics trouble codes th...		Unsigned 24 b
□	Measure...		DTC_0x000078	Report Active DTCs [19 0...	ECUDiagno...	Throttle Position Sensor Circuit Malfunction (level...		Unsigned 1 bit
□	Measure...		DTC_0x000078_Status_By...	Report Active DTCs [19 0...	ECUDiagno...	Throttle Position Sensor Circuit Malfunction (level...		Unsigned 8 bit
□	Measure...		DTC_0x0000D9	Report Active DTCs [19 0...	ECUDiagno...	Engine Overtemp Condition (level 1)		Unsigned 1 bit
□	Measure...		DTC_0x0000D9_Status_B...	Report Active DTCs [19 0...	ECUDiagno...	Engine Overtemp Condition (level 1)		Unsigned 8 bit
□	Measure...		DTC_0x000145	Report Active DTCs [19 0...	ECUDiagno...	Knock Sensor 1 Circuit Malfunction (level 2)		Unsigned 1 bit
□	Measure...		DTC_0x000145_Status_By...	Report Active DTCs [19 0...	ECUDiagno...	Knock Sensor 1 Circuit Malfunction (level 2)		Unsigned 8 bit
□	Measure...		DTC_0x000193	Report Active DTCs [19 0...	ECUDiagno...	Exhaust Gas Recirculation Circuit Malfunction (le...		Unsigned 1 bit
□	Measure...		DTC_0x000193_Status_By...	Report Active DTCs [19 0...	ECUDiagno...	Exhaust Gas Recirculation Circuit Malfunction (le...		Unsigned 8 bit

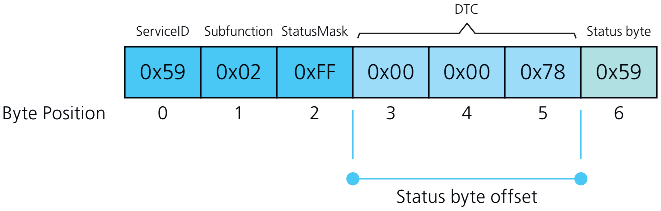
No filter is active

ControlDesk generates two DTC variables for each diagnostic trouble code: the DTC number and the DTC status byte (exception: no status byte is generated for the OBD diagnostic protocol). The DTC number is a verbal computation table indicating whether the DTC has occurred or not. The status byte parameter has a length of 1 byte and displays the state of the DTC. Additionally, the `#DTCs` variable (showing the number of all DTCs that occurred) and the `#RT_GEN_DTCs` variable (showing the number of the DTCs that occurred and that are not defined in the ODX database) are generated. The latter can be used to check if there are any DTCs that are not defined in the ODX database (see Custom DTCs).

Status byte offset In most cases, the status byte of a DTC directly follows the DTC. Its exact position is determined by the status byte offset, which begins with the DTC's first byte and ends with the byte before the status byte.

Usually, the default status byte offset according to the internal protocol-specific logic is used (KWP protocol: status byte offset = 2, UDS protocol: status byte offset = 3). However, you can use a different status byte offset value. This can be necessary if the DTC length differs from the default offset value, or if the status byte does not directly follow the DTC. To use a different status byte offset, you have to configure a service for DTC measurements in the XML configuration file. Refer to Configuring Services for DTC Measurements in the XML Configuration File on page 115.

The following illustration shows an example of a response PDU using the UDS diagnostic protocol. The status byte offset has a value of 3, so the status byte location is determined by adding this offset to the start position of the first byte position of the DTC.

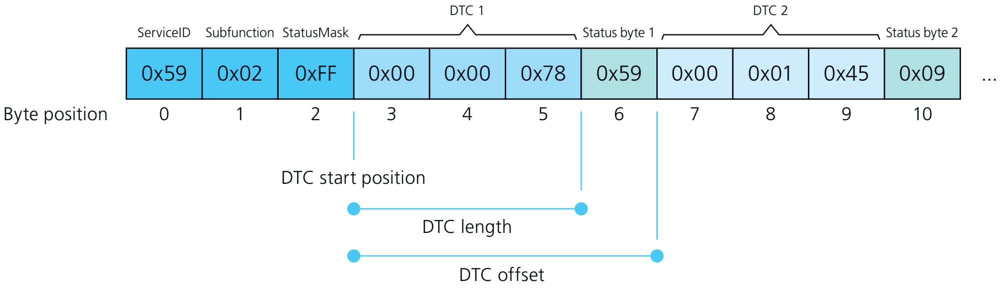


				DTC		
	ServiceID	Subfunction	StatusMask			Status byte
	0x59	0x02	0xFF	0x00	0x00	0x78	0x59
Byte Position	0	1	2	3	4	5	6

Extracting DTCs from a response PDU ControlDesk can extract DTC measurements directly from a response PDU. This is done on the basis of DTC information that is specified in the service configuration for DTC measurements. This DTC information contains three information types: the length (in bytes) of a DTC, the start byte position of the first DTC in the PDU, and the offset (in bytes) between two DTCs within the PDU. The DTC information data allows ControlDesk to evaluate the DTCs from a PDU.

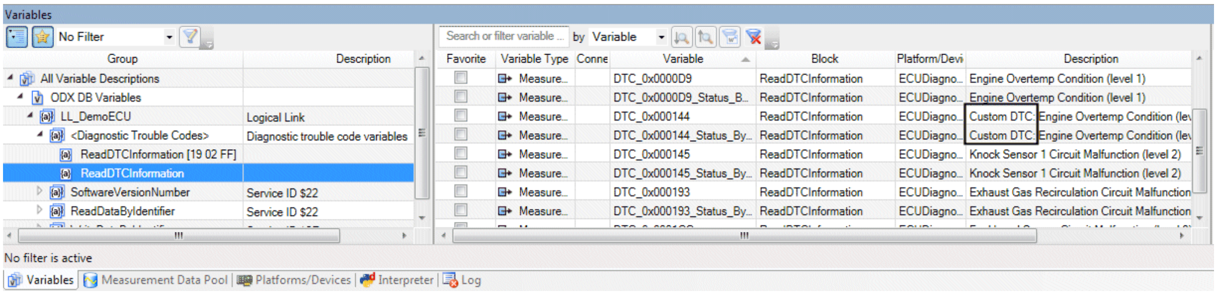
The following illustration shows an example of a response PDU using the UDS diagnostic protocol. In the example, the following DTC information is used to extract the DTCs from the PDU:

- DTC length = 3
- Start byte position of the first DTC in the PDU = 3
- DTC offset = 4



Byte position	0	1	2	3	4	5	6	7	8	9	10	
	0x59	0x02	0xFF	0x00	0x00	0x78	0x59	0x00	0x01	0x45	0x09	...

Custom DTCs ControlDesk allows you to define custom DTCs. Custom DTCs are defined outside the ODX database and are handled in ControlDesk in the same way as DTCs which are defined within the ODX database. Defining custom DTCs can be useful if the ODX database does not contain the required DTCs and the corresponding DTC variables would not be available. You can distinguish custom DTCs and DTCs from the ODX via their descriptions.



Variables

No Filter

Group	Description
All Variable Descriptions	
ODX DB Variables	
LL_DemoECU	Logical Link
<Diagnostic Trouble Codes>	Diagnostic trouble code variables
ReadDTCInformation [19 02 FF]	
ReadDTCInformation	
SoftwareVersionNumber	Service ID $22
ReadDataByIdentifier	Service ID $22

Search or filter variable ... by Variable

Favorite	Variable Type	Conne	Variable	Block	Platform/Devi	Description
□	Measure...		DTC_0x0000D9	ReadDTCInformation	ECUDiagno...	Engine Overtemp Condition (level 1)
□	Measure...		DTC_0x0000D9_Status_B...	ReadDTCInformation	ECUDiagno...	Engine Overtemp Condition (level 1)
□	Measure...		DTC_0x000144	ReadDTCInformation	ECUDiagno...	Custom DTC: Engine Overtemp Condition (lev
□	Measure...		DTC_0x000144_Status_By...	ReadDTCInformation	ECUDiagno...	Custom DTC: Engine Overtemp Condition (lev
□	Measure...		DTC_0x000145	ReadDTCInformation	ECUDiagno...	Knock Sensor 1 Circuit Malfunction (level 2)
□	Measure...		DTC_0x000145_Status_By...	ReadDTCInformation	ECUDiagno...	Knock Sensor 1 Circuit Malfunction (level 2)
□	Measure...		DTC_0x000193	ReadDTCInformation	ECUDiagno...	Exhaust Gas Recirculation Circuit Malfunction
□	Measure...		DTC_0x000193_Status_By...	ReadDTCInformation	ECUDiagno...	Exhaust Gas Recirculation Circuit Malfunction

No filter is active

However, if a custom DTC is also specified in the ODX database, ControlDesk only uses the definition from the ODX database to generate the DTC variables. No additional DTC variables are generated for the custom DTC in this case.

Restricting the number of DTC variables ControlDesk generates DTC variables for all diagnostic trouble codes that are defined in the ODX database and for the custom DTCs defined in the XML configuration by default. You can restrict the number of DTC variables to be generated by defining a range of DTCs. This can be useful if only a certain range of DTCs is relevant for your work.

Custom DTCs and the DTC range can be specified in the XML configuration file. Refer to Configuring Services for DTC Measurements in the XML Configuration File on page 115.

Basics on DTC measurements

Performing DTC measurements makes diagnostic trouble codes visible in ControlDesk instruments other than the Fault Memory and the Diagnostics Instrument. This allows you to record a DTC's behavior in a plotter, to use changes to DTCs as trigger inputs for measurements etc., to use DTC variables as input signals for calculated variables, etc.

The service configurations for performing DTC measurements are part of the service configurations for reading fault memory.

Default diagnostic service settings for performing DTC measurements according to protocol-specific service identification

ControlDesk generates default service settings for the DTC status byte offset and the DTC information (DTC start position, DTC offset, DTC length) according to the protocol-specific identification of services. These default values are provided with the protocol-specific default diagnostic services for reading fault memory.

Configuring further services for DTC measurements

If you want to use DTC information or a DTC status byte offset value different from the default settings in ControlDesk, or if you want to specify DTC ranges or custom DTCs to be used in ControlDesk, you can specify a specific diagnostic service for fault memory reading in an XML configuration file. For further information, refer to Configuring Services for DTC Measurements in the XML Configuration File on page 115.

Related topics

Basics

Specifics of the OBD Support

Introduction

There are some specifics to note when using the OBD diagnostic protocol.

ODX database template

ControlDesk provides an ODX database template for OBD2 (ISO15031). The database is provided in binary format.

Note

The template makes no claim to completeness. It can be used for commissioning, for example, but is not suitable for release tests.

Supported modes

According to the OBD specification, a *mode* is a service used for a specific task such as requesting current powertrain diagnostic data from the ECU.

ControlDesk supports the following modes:

Supported Mode	Description
Mode 01[1]	Request current powertrain diagnostic data
Mode 02 Frame 00[1]	Request powertrain freeze frame data
Mode 03[2]	Request emission-related diagnostic trouble codes
Mode 04[2]	Clear/reset emission-related diagnostic information
Mode 06[1]	Request onboard monitoring test results for specific monitored systems
Mode 07[2]	Request emission-related diagnostic trouble codes detected during current or last completed driving cycle
Mode 08[1]	Request control of onboard system, test or component
Mode 09[1]	Request vehicle information

[1] PID analysis is performed for this mode. See below.

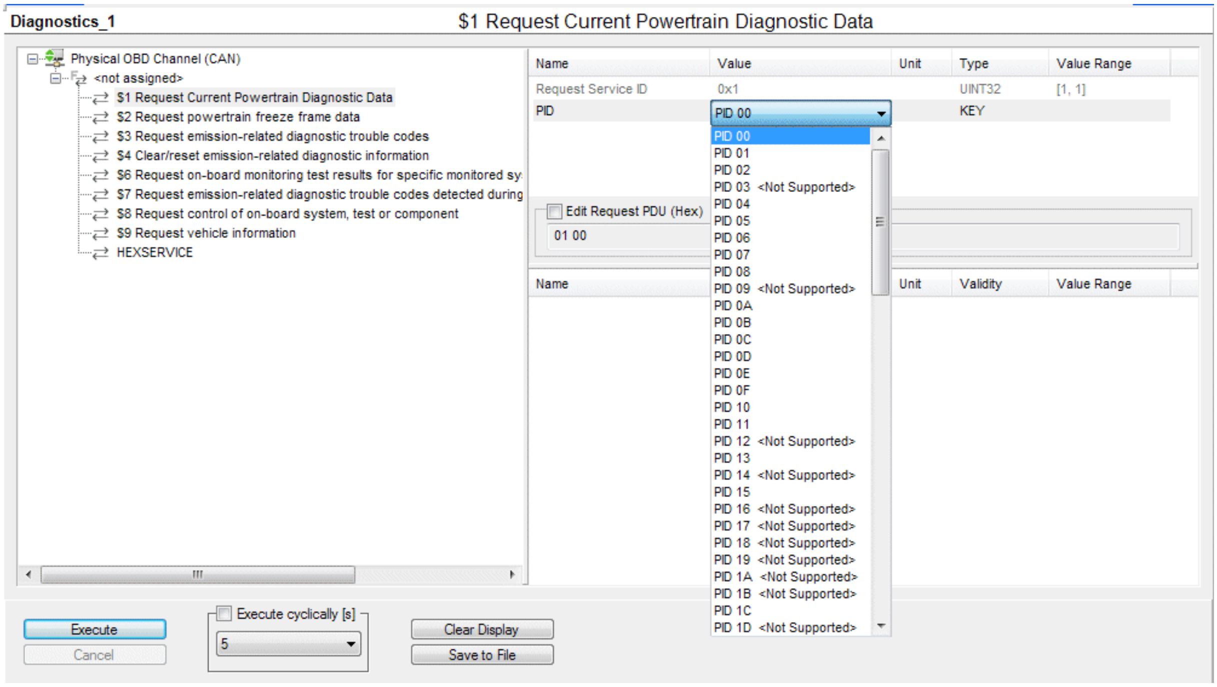
[2] Provided as a fault memory read service. This mode does not provide PIDs.

PID analysis

ControlDesk performs a *parameter identifier (PID) analysis* for logical links during device connection to check which PIDs are supported by the ECU/vehicle. PIDs are bit-encoded parameters defined in OBD specification ISO 15031-5. Since not all ECUs support all PIDs, ControlDesk lets you check which PIDs a specific ECU supports. If there are any logical links for which no supported PIDs were found during device connection, ControlDesk performs a second PID analysis for them when online calibration is started.

PID analysis is performed individually for each mode that supports PIDs.

The following illustration shows an example of the Diagnostics Instrument with the PID analysis for mode 01:



Diagnostics_1 $1 Request Current Powertrain Diagnostic Data

- Physical OBD Channel (CAN)
  - <not assigned>
    - $1 Request Current Powertrain Diagnostic Data
    - $2 Request powertrain freeze frame data
    - $3 Request emission-related diagnostic trouble codes
    - $4 Clear/reset emission-related diagnostic information
    - $6 Request on-board monitoring test results for specific monitored sy
    - $7 Request emission-related diagnostic trouble codes detected during
    - $8 Request control of on-board system, test or component
    - $9 Request vehicle information
    - HEXSERVICE

Name	Value	Unit	Type	Value Range
Request Service ID	0x1		UINT32	[1, 1]
PID	PID 00		KEY	

PID 00
PID 01
PID 02
PID 03 <Not Supported>
PID 04
PID 05
PID 06
PID 07
PID 08
PID 09 <Not Supported>
PID 0A
PID 0B
PID 0C
PID 0D
PID 0E
PID 0F
PID 10
PID 11
PID 12 <Not Supported>
PID 13
PID 14 <Not Supported>
PID 15
PID 16 <Not Supported>
PID 17 <Not Supported>
PID 18 <Not Supported>
PID 19 <Not Supported>
PID 1A <Not Supported>
PID 1B <Not Supported>
PID 1C
PID 1D <Not Supported>

□ Edit Request PDU (Hex)
01 00

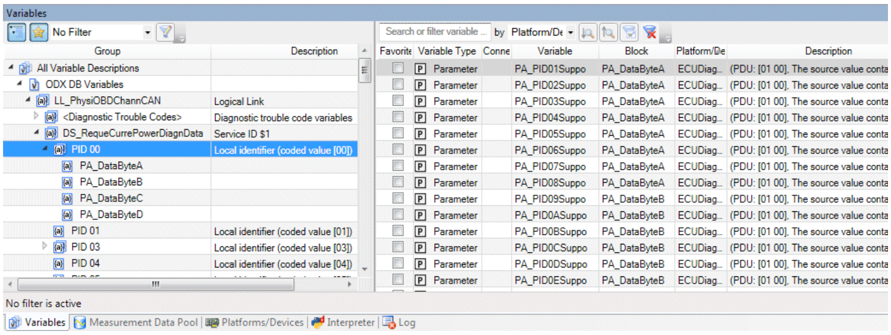
Name		Unit	Validity	Value Range

Variables generated for PIDs

If generation of the variable description from the ODX database is enabled in the **Select Logical Links** dialog (with selected **Use additional protocol information** checkbox), ControlDesk generates the following variables:

- Read-only parameters for each PID of each mode containing PIDs. After online calibration is started, the parameter values indicate whether a specific PID is supported by the ECU.

  The illustration below shows an example of the **Variable** controlbar displaying these variables for mode 01 (**DS_RequeCurrePowerDiagnData**).



Variables

No Filter

Group	Description
All Variable Descriptions	
ODX DB Variables	
LL_PhysiOBDChannCAN	Logical Link
<Diagnostic Trouble Codes>	Diagnostic trouble code variables
DS_RequeCurrePowerDiagnData	Service ID $1
PID 00	Local identifier (coded value [00])
PA_DataByteA	
PA_DataByteB	
PA_DataByteC	
PA_DataByteD	
PID 01	Local identifier (coded value [01])
PID 03	Local identifier (coded value [03])
PID 04	Local identifier (coded value [04])

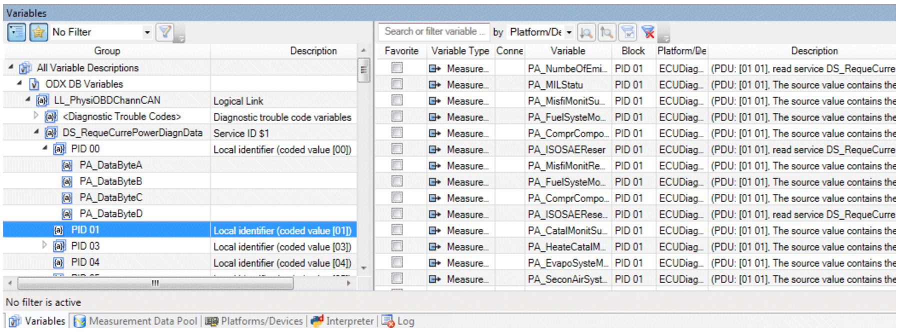
Search or filter variable ... by Platform/De

Favorite	Variable Type	Conne	Variable	Block	Platform/De	Description
□	Parameter		PA_PID01Suppo	PA_DataByteA	ECUDiag...	(PDU: [01 00]. The source value conta
□	Parameter		PA_PID02Suppo	PA_DataByteA	ECUDiag...	(PDU: [01 00]. The source value conta
□	Parameter		PA_PID03Suppo	PA_DataByteA	ECUDiag...	(PDU: [01 00]. The source value conta
□	Parameter		PA_PID04Suppo	PA_DataByteA	ECUDiag...	(PDU: [01 00]. The source value conta
□	Parameter		PA_PID05Suppo	PA_DataByteA	ECUDiag...	(PDU: [01 00]. The source value conta
□	Parameter		PA_PID06Suppo	PA_DataByteA	ECUDiag...	(PDU: [01 00]. The source value conta
□	Parameter		PA_PID07Suppo	PA_DataByteA	ECUDiag...	(PDU: [01 00]. The source value conta
□	Parameter		PA_PID08Suppo	PA_DataByteA	ECUDiag...	(PDU: [01 00]. The source value conta
□	Parameter		PA_PID09Suppo	PA_DataByteB	ECUDiag...	(PDU: [01 00]. The source value conta
□	Parameter		PA_PID0ASuppo	PA_DataByteB	ECUDiag...	(PDU: [01 00]. The source value conta
□	Parameter		PA_PID0BSuppo	PA_DataByteB	ECUDiag...	(PDU: [01 00]. The source value conta
□	Parameter		PA_PID0CSuppo	PA_DataByteB	ECUDiag...	(PDU: [01 00]. The source value conta
□	Parameter		PA_PID0DSuppo	PA_DataByteB	ECUDiag...	(PDU: [01 00]. The source value conta
□	Parameter		PA_PID0ESuppo	PA_DataByteB	ECUDiag...	(PDU: [01 00]. The source value conta

No filter is active

- Variables for each PID of a specific mode according to the ODX database

The illustration below shows an example of the **Variables** controlbar displaying these variables for PID 01.



Group	Description
All Variable Descriptions	
ODX DB Variables	
LL_PhysiOBDChannCAN	Logical Link
<Diagnostic Trouble Codes>	Diagnostic trouble code variables
DS_RequeCurrePowerDiagnData	Service ID $1
PID 00	Local identifier (coded value [00])
PA_DataByteA	
PA_DataByteB	
PA_DataByteC	
PA_DataByteD	
PID 01	Local identifier (coded value [01])
PID 03	Local identifier (coded value [03])
PID 04	Local identifier (coded value [04])

Favorite	Variable Type	Conne	Variable	Block	Platform/De	Description
□	Measure...		PA_NumbeOfEmi...	PID 01	ECUDiag...	(PDU: [01 01], read service DS_RequeCurre
□	Measure...		PA_MILStatu	PID 01	ECUDiag...	(PDU: [01 01], The source value contains the
□	Measure...		PA_MisfiMonitSu...	PID 01	ECUDiag...	(PDU: [01 01], The source value contains the
□	Measure...		PA_FuelSysteMo...	PID 01	ECUDiag...	(PDU: [01 01], The source value contains the
□	Measure...		PA_ComprCompo...	PID 01	ECUDiag...	(PDU: [01 01], The source value contains the
□	Measure...		PA_ISOSAEReser	PID 01	ECUDiag...	(PDU: [01 01], read service DS_RequeCurre
□	Measure...		PA_MisfiMonitRe...	PID 01	ECUDiag...	(PDU: [01 01], The source value contains the
□	Measure...		PA_FuelSysteMo...	PID 01	ECUDiag...	(PDU: [01 01], The source value contains the
□	Measure...		PA_ComprCompo...	PID 01	ECUDiag...	(PDU: [01 01], The source value contains the
□	Measure...		PA_ISOSAERese...	PID 01	ECUDiag...	(PDU: [01 01], read service DS_RequeCurre
□	Measure...		PA_CatalMonitSu...	PID 01	ECUDiag...	(PDU: [01 01], The source value contains the
□	Measure...		PA_HeateCatalM...	PID 01	ECUDiag...	(PDU: [01 01], The source value contains the
□	Measure...		PA_EvapoSysteM...	PID 01	ECUDiag...	(PDU: [01 01], The source value contains the
□	Measure...		PA_SeconAirSyst...	PID 01	ECUDiag...	(PDU: [01 01], The source value contains the

ControlDesk lets you generate diagnostics variables for the following modes:

- Mode 01
- Mode 06

> **Note**
>
> Only variables that indicate whether a specific PID is supported are generated for this mode.

- Mode 08
- Mode 09

> **Note**
>
> Mode 09 can provide several ECU responses, but only the first response is evaluated.

Variables from unsupported PIDs

Diagnostics variables are generated during device configuration. Since PID analysis is performed later on, ControlDesk generates diagnostics variables for all the PIDs regardless of whether the ECU supports them.

This has the following consequences:

- Parameters from unsupported PIDs are ignored during upload when starting online calibration.
- Measurement variables from unsupported PIDs are ignored during measurement.

Related topics

Basics

Basics of ECU Diagnostics with ControlDesk... 10

Programming the ECU Flash Memory via a Diagnostic Protocol

Introduction

You can program the flash memory of ECUs. This allows you to store a modified data set or a new ECU software revision on an ECU permanently. The modified data set also remains active after you stop using a calibration tool.

How to Program the ECU Flash Memory via a Diagnostic Protocol

Objective

With the ControlDesk ECU Diagnostics Module, the flash memory of ECUs can be programmed via diagnostic protocols. ControlDesk uses the flash programming information contained in the ODX database for this.

Preconditions

- An ECU Diagnostics device must be available in the active experiment.
- The appropriate ECU must be connected to the host PC.
- The device must be configured correctly. Refer to How to Configure an ECU Diagnostics Device on page 36.
- The device must be in the disconnected state.
- The flash programming information in the ODX database must be valid, and the compiled Java file (`CLASS` or `JAR` files) containing the flash job must be available in the database.
- You can import a PDX package to specify an ODX diagnostics database. If you do so, note the following: To execute diagnostic jobs and/or perform ECU flash memory programming, it is not sufficient that the related files (such as `CLASS`, `JAR`, or `HEX` files) are in the PDX package. You have to import these files in addition to the PDX package.

Restriction

`DiagnosticsManagementEvents / IXaDiagnosticsManagementEvents <<EventInterface>>` events *are not triggered when you perform ECU flash programming as described below*. The events are triggered only when you execute an ECU flash programming session via automation.

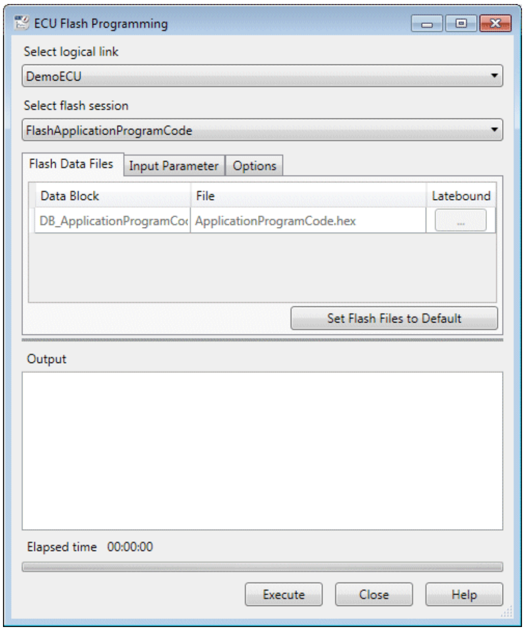
Method

To program the ECU flash memory via a diagnostic protocol

1 Select the ECU Diagnostics device in the **Project** controlbar.

2 From the context menu of the device, select **Flash ECU**.

The **ECU Flash Programming** dialog opens.



ECU Flash Programming

Select logical link: DemoECU

Select flash session: FlashApplicationProgramCode

Data Block	File	Latebound
DB_ApplicationProgramCo(	ApplicationProgramCode.hex	...

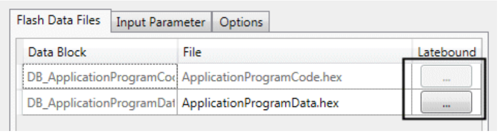
Elapsed time 00:00:00

3 Specify the ECU to be flashed by selecting the appropriate logical link from the list. The list contains all the logical links that were selected for use with the device during device configuration.

4 Select a flash session from the list of all the flash sessions defined in the ODX database for the selected logical link.

The flash data files belonging to the selected flash session are displayed on the **Flash Data Files** tab.

5 You can replace a flash data file on the list by any other ECU Image file that matches the selected ECU, provided that the flash data file's latebound settings in the ODX database allow this. To specify a flash data file, type text in the **File** edit field, or click the **Browse** button and select the ECU Image file in the **Flash File** dialog. If the flash data file selection is not modifiable, the **Browse** button is disabled.



Data Block	File	Latebound
DB_ApplicationProgramCo(	ApplicationProgramCode.hex	...
DB_ApplicationProgramDat	ApplicationProgramData.hex	...

6 The **Input Parameter** tab displays the input parameters of the flash job belonging to the selected flash session. You can change the settings of writable input parameters before the flash operation is started.

7 On the **Options** tab, you can specify whether the flash job contains the `StartCommunication` control primitive or ControlDesk must execute the control primitive before the flash job is started.

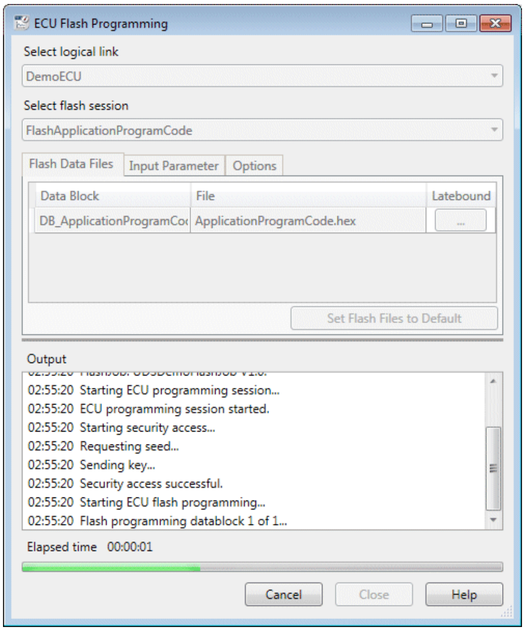
- With the checkbox selected, ControlDesk starts the flash job (which is expected to execute the `StartCommunication` control primitive) without having to execute any control primitives beforehand.

- With the checkbox cleared, ControlDesk executes the `StartCommunication` control primitive before it starts the flash job.

Note

For K-Line-based diagnostic protocols, the `StartCommunication` control primitive must be executed first to initialize the bus. Otherwise, communication with an ECU via K-Line is impossible.

8 Click **Execute** to start the flash operation.



Select logical link: DemoECU

Select flash session: FlashApplicationProgramCode

Data Block	File	Latebound
DB_ApplicationProgramCo	ApplicationProgramCode.hex	...

Output

```
02:55:20  Starting ECU programming session...
02:55:20  ECU programming session started.
02:55:20  Starting security access...
02:55:20  Requesting seed...
02:55:20  Sending key...
02:55:20  Security access successful.
02:55:20  Starting ECU flash programming...
02:55:20  Flash programming datablock 1 of 1...
```

Elapsed time 00:00:01

Information on the current status, warnings, error messages, etc., about the flashing operation are displayed in the **Output** field according to the implementation of the flash job. The progress of the ECU flash programming operation is shown at the bottom of the dialog, and the elapsed time of the running operation is displayed.

You can cancel the ECU flash programming operation by clicking the **Cancel** button.

NOTICE

Cancelling an ECU flash programming operation may lead to unpredictable results or conflicts, depending on the point in time the ECU flash programming process is aborted. For example, aborting a flashing operation while ECU boot code is being flashed to the ECU may make the ECU completely unflashable. If you abort a flashing operation while ECU application code is being programmed to the ECU, the application might not run.

9 When ECU flash programming is completed, click **Close**.

ControlDesk closes the **ECU Flash Programming** dialog.

Result

You have supplied the ECU's flash memory with the latest software revision and/or new calibration data.

Related topics

Basics

Automating ECU Diagnostics Tasks.. 136

References

DiagnosticsManagementEvents / IXaDiagnosticsManagementEvents <<EventInterface>> (ControlDesk Automation 📖)
Flash ECU (ECU Diagnostics) (ControlDesk Platform Management 📖)

Identifying Services and Parameters for ControlDesk Functions

Where to go from here

Information in this section

Basics of the XML Configuration File

Introduction

ControlDesk uses specific diagnostic services specified in the ODX database to perform the following actions:

- Reading and displaying fault memory entries and the corresponding environment data
- Clearing fault memory entries
- Performing measurement and calibration via the ECU Diagnostics device (including getting the required security access)
- Performing DTC measurements

Services and parameters are identified by ODX semantics, short names, service IDs and/or byte positions. The services and parameters must be mapped to the ControlDesk functions, such as reading the fault memory.

You can use an XML configuration file to set up services different from the default diagnostic services.

XML configuration file

You can use an XML configuration file to set up services different from the default diagnostic services, for example, for reading the fault memory and environment data, clearing fault memory entries, or performing measurement and calibration. These services can then be used in ControlDesk. No entries in the XML configuration file are required for ControlDesk functions that use the default diagnostic services. ControlDesk supports two types of XML configuration files.

Local XML configuration file The local XML configuration file lets you specify project-specific service configurations. You must add the local

XML configuration file to the file list in the **Select ODX Files** dialog when configuring your ECU Diagnostics device. Refer to How to Configure an ECU Diagnostics Device on page 36.

Global XML configuration file The global XML configuration file lets you specify service configurations that apply to all ControlDesk projects. The global XML configuration file must be located in the `.\Config` folder of your ControlDesk installation. It is made available to all ControlDesk projects without further action. Using a global XML configuration file is useful if you work with an ODX that must adhere to guidelines for authors, for example.

Note

- If the `.\Config` folder contains several XML configuration files that meet the naming requirements, ControlDesk uses the first XML file it finds as the global XML configuration file. To avoid errors, do not save more than one XML configuration file to the `.\Config` folder.
- If you migrate to another ControlDesk version, you must copy the global XML configuration file manually to the `.\Config` folder of the other ControlDesk installation.

Using XML configuration files is optional, which means that you can work with a global and/or local XML configuration file, or without an XML configuration file. An XML configuration file must meet some requirements (see below).

Mapping of ODX services and parameters to ControlDesk functions

The way in which mapping to ControlDesk functions is done depends on whether the global and/or local XML configuration files are available.

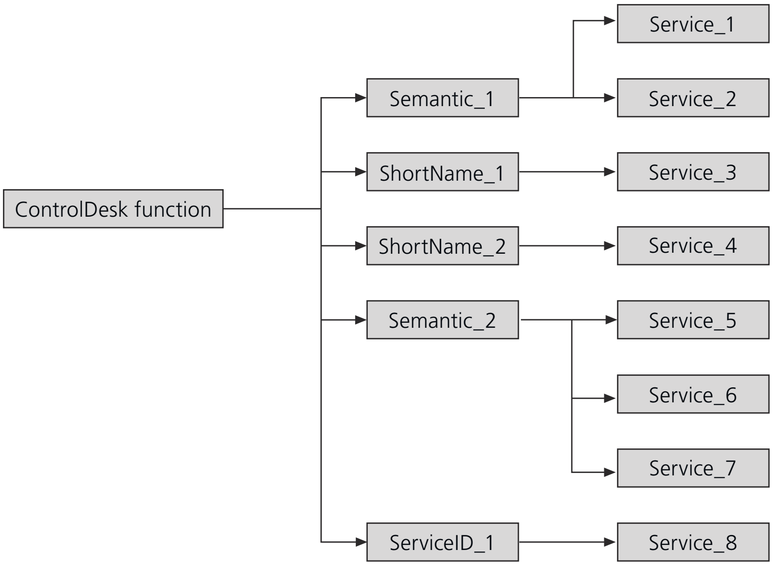
Available XML Files	Description
No XML configuration file (neither global nor local)	The default diagnostic services are taken. Services can be according to the internal protocol-specific logic and, in some cases, identified by ODX semantics. They are mapped to the ControlDesk functions according to the conventions described for the default diagnostic services in Performing ECU Diagnostic Tasks on page 55 and according to the conventions described in Conventions in Connection with ODX Databases on page 22.
Global XML configuration file only	Mapping is done via the global XML configuration file for functions with service configuration settings in the XML file. For all other functions, the default diagnostic services are taken, that is, services can be according to the internal protocol-specific logic and, in some cases, identified by ODX semantics.
Local XML configuration file only	Mapping is done via the local XML configuration file for functions with service configuration settings in the XML file. For all other functions, the default diagnostic services are taken, that is, services can be according to the internal protocol-specific logic and, in some cases, identified by ODX semantics.
Global and local XML configuration files	The XML configuration files are evaluated in a hierarchical order: ▪ The service configuration settings in the global XML configuration file overrule the default settings. The service configuration settings in the local

Available XML Files	Description
	XML configuration file overrule the settings in the global XML configuration file. ▪ For functions with service configuration settings in the local XML configuration file, mapping is done via the local XML configuration file. ▪ For functions with service configuration settings in the global XML configuration file but not in the local XML file, mapping is done via the global XML configuration file. ▪ For functions with no service configuration settings in the local and global XML files, the default diagnostic services are taken, that is, services can be according to the internal protocol-specific logic and, in some cases, identified by ODX semantics. ▪ Service configurations that are specified for a specific logical link (and therefore are valid for this logical link only) do not overrule service configurations that are specified for all logical links (elements that do not have the optional `logicalLink` attribute (see below)).

Using several ODX services in parallel

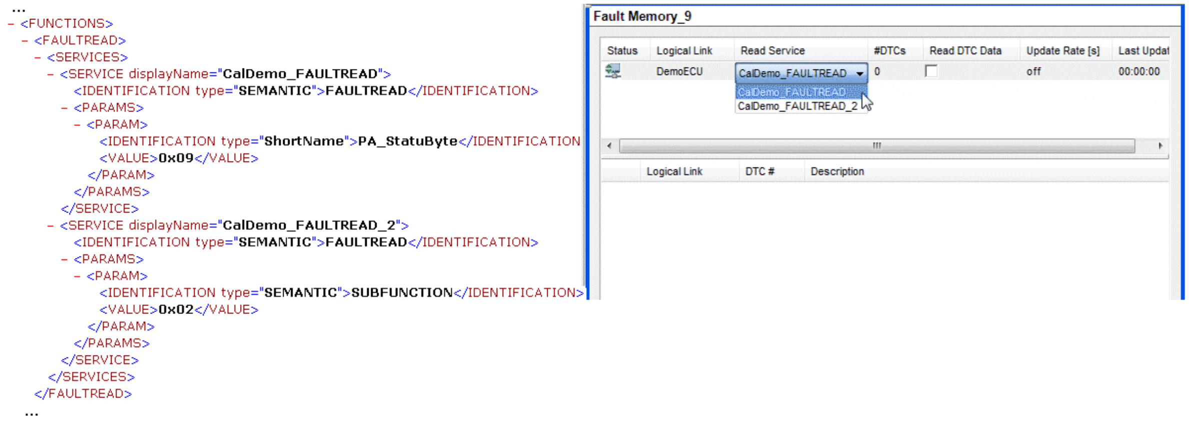
ControlDesk allows you to use several ODX services for the same ControlDesk function in parallel. Each ControlDesk function uses one or more semantics, short names, service IDs and/or combinations of them for identification purposes. Each semantic references one or more ODX services. A short name references a single ODX service (because the short names of the services must be unique within each logical link). A service ID references one or more ODX services of the same type. The mapping of each ControlDesk function to a set of ODX semantics, short names and/or service IDs is configured via an XML file which must meet some requirements (see below).

The following illustration shows an example of specifying several ODX services for one ControlDesk function. Service identification via ODX semantics, via short names and via service ID is displayed.



To make the different services available in ControlDesk, you must use the local and/or global XML configuration file(s). As a result, all the services found by semantics, short names and service IDs are offered as services in a ControlDesk instrument such as the Fault Memory Instrument. ControlDesk lets you select one of the read services for each logical link and for each instance of the Fault Memory Instrument.

Example: Two services for reading the fault memory The following illustration shows an excerpt from an XML configuration file where two services for reading the fault memory are configured, and ControlDesk's Fault Memory Instrument offering the two read services found for selection:



```
 ...
- <FUNCTIONS>
  - <FAULTREAD>
    - <SERVICES>
      - <SERVICE displayName="CalDemo_FAULTREAD">
          <IDENTIFICATION type="SEMANTIC">FAULTREAD</IDENTIFICATION>
        - <PARAMS>
          - <PARAM>
              <IDENTIFICATION type="ShortName">PA_StatuByte</IDENTIFICATION
              <VALUE>0x09</VALUE>
            </PARAM>
          </PARAMS>
        </SERVICE>
      - <SERVICE displayName="CalDemo_FAULTREAD_2">
          <IDENTIFICATION type="SEMANTIC">FAULTREAD</IDENTIFICATION>
        - <PARAMS>
          - <PARAM>
              <IDENTIFICATION type="SEMANTIC">SUBFUNCTION</IDENTIFICATION>
              <VALUE>0x02</VALUE>
            </PARAM>
          </PARAMS>
        </SERVICE>
      </SERVICES>
    </FAULTREAD>
   ...
```

Fault Memory_9

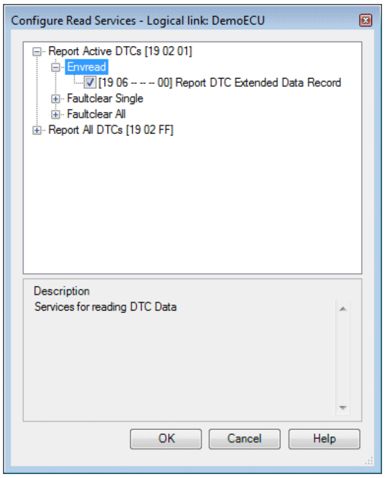
Status	Logical Link	Read Service	#DTCs	Read DTC Data	Update Rate [s]	Last Updat
	DemoECU	CalDemo_FAULTREAD CalDemo_FAULTREAD CalDemo_FAULTREAD_2	0	□	off	00:00:00

	Logical Link	DTC #	Description

The fault memory of an ECU usually contains different types of information, for example, floating and confirmed DTCs. Configuring several read services in the XML file allows you to read the different types of information with ControlDesk's Fault Memory Instruments. After the XML file is added to the file list in the Select ODX Files dialog during configuration of your ECU Diagnostics device, ControlDesk lets you select one of the read services found for each logical link and for each instance of the Fault Memory Instrument.

Example: Several services for reading environment data The following illustration shows the Configure Read Services dialog offering a service for reading environment data selection.

You can reach the selection dialog via the instrument's **Properties** controlbar (**Logical Links** properties).



Configure Read Services - Logical link: DemoECU

- Report Active DTCs [19 02 01]
  - Envread
    - ☑ [19 06 -- -- -- 00] Report DTC Extended Data Record
  - Faultclear Single
  - Faultclear All
- Report All DTCs [19 02 FF]

Description
Services for reading DTC Data

The services to be used for reading environment data and clearing single or all fault memory entries are always specified in connection with the selected service for reading the fault memory. This means that the Read service you select in the Fault Memory Instrument for a logical link actually stands for a combination of services for reading the fault memory, reading environment data, clearing single fault memory entries and clearing all memory entries (as configured via the **Properties** controlbar).

Supported ControlDesk functions

Identification of services and parameters for ControlDesk functions via the XML configuration file is supported for the following functions:

- Reading fault memory (FAULTREAD)
- Reading environment data (ENVREAD)
- Clearing single fault memory entries (FAULTCLEAR_SINGLE)
- Clearing all fault memory entries (FAULTCLEAR_ALL)
- Measuring variables (MEASUREMENT)
- Calibrating parameters (CALIBRATION)
- Getting security access for parameter calibration (SECURITY_ONLINE)

Note

If you want to use non-default diagnostic services in ControlDesk, you must create and use a local and/or global XML configuration file containing the relevant configuration settings for the ControlDesk functions. No entries in an XML configuration file are required for ControlDesk functions that use the default diagnostic services.

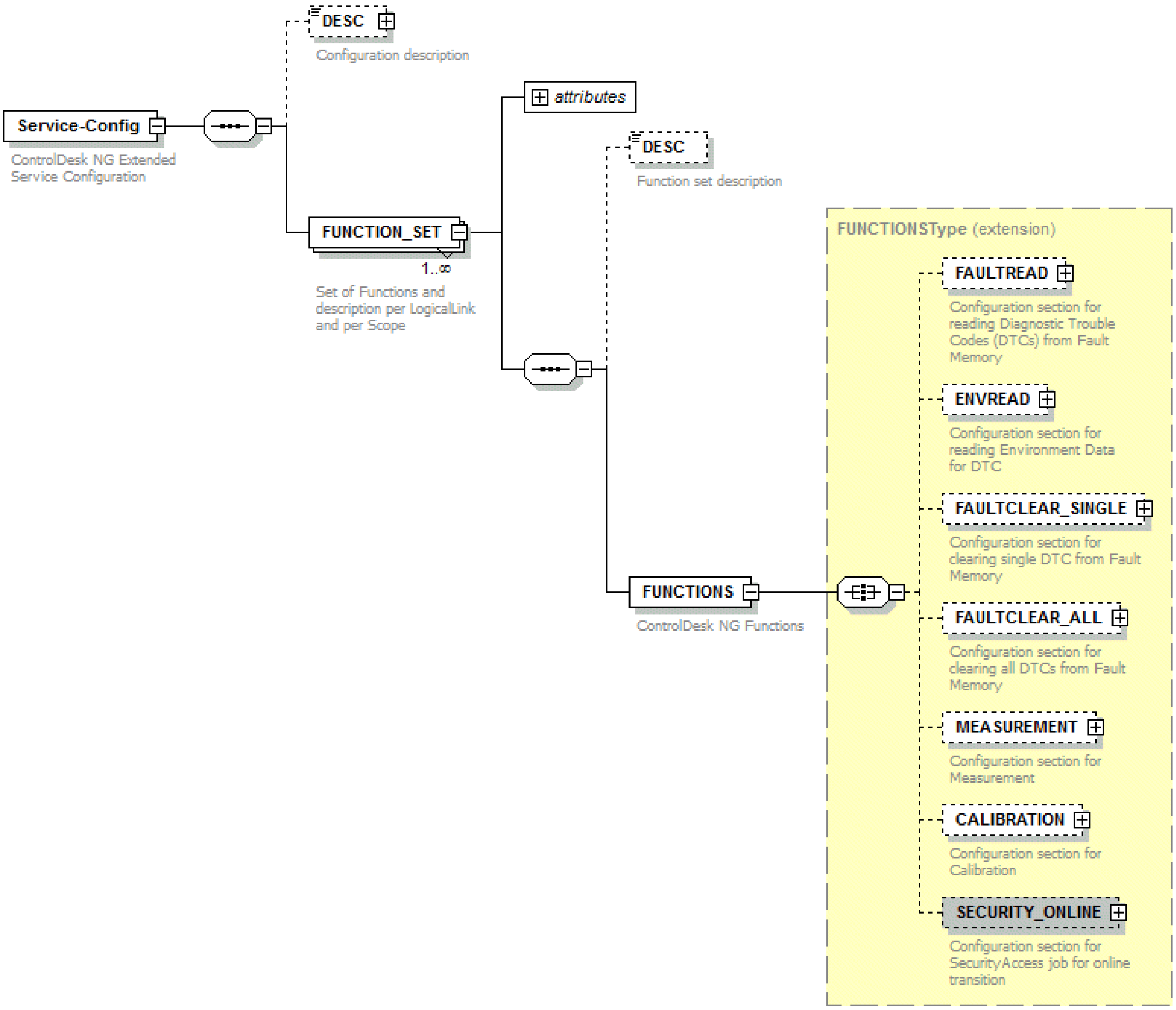
Principle structure of the XML configuration file

The XML configuration file comprises the service configurations for the logical links. Each configuration is specified in a separate `FUNCTION_SET` element. The XML configuration file can contain one or more `FUNCTION_SET` elements.

Service configurations can be specified individually for each logical link. The optional `logicalLink` attribute of the `FUNCTION_SET` element lets you define the logical link the configuration is to be applied to. You must specify the logical link's short name as the `logicalLink` value. The `FUNCTION_SET` element for which the `logicalLink` attribute is not defined is used as the default service configuration. The default service configuration is applied to all the logical links for which the XML configuration file does not contain individual `FUNCTION_SET` elements.

A `FUNCTION_SET` element should contain configuration settings for all the supported ControlDesk functions (`FAULTREAD`, `ENVREAD`, `FAULTCLEAR_SINGLE`, `FAULTCLEAR_ALL`, `MEASUREMENT`, `CALIBRATION`, `SECURITY_ONLINE`) which are to be used in ControlDesk in connection with the appropriate logical link(s).

The following illustration shows the elementary structure of an XML configuration file. The structure is displayed on the basis of the `DS_Service_Config.xsd` validation schema:


DESC
Configuration description
Service-Config
ControlDesk NG Extended Service Configuration
attributes
DESC
Function set description
FUNCTION_SET
1..∞
Set of Functions and description per LogicalLink and per Scope
FUNCTIONSType (extension)
FAULTREAD
Configuration section for reading Diagnostic Trouble Codes (DTCs) from Fault Memory
ENVREAD
Configuration section for reading Environment Data for DTC
FAULTCLEAR_SINGLE
Configuration section for clearing single DTC from Fault Memory
FUNCTIONS
ControlDesk NG Functions
FAULTCLEAR_ALL
Configuration section for clearing all DTCs from Fault Memory
MEASUREMENT
Configuration section for Measurement
CALIBRATION
Configuration section for Calibration
SECURITY_ONLINE
Configuration section for SecurityAccess job for online transition

For information on the XSD file, refer to XML file validation on page 99.

For details on the necessary configuration settings for the different ControlDesk functions, refer to Performing ECU Diagnostic Tasks on page 55.

Naming convention for the XML configuration file

The file name of the XML configuration file must begin with `DS_Service_Config`. You can expand the file name, if necessary.

> **Note**
>
> - The ODX database can contain one XML configuration file that meets the naming requirements only.
> - If the `.\Config` folder contains several XML configuration files that meet the naming requirements, ControlDesk uses the first XML file it finds as the global XML configuration file. To avoid errors, you should not save more than one XML configuration file to the `.\Config` folder.

XML file validation

When a local XML configuration file is imported into the ODX database, it is validated against the `DS_Service_Config.xsd` validation schema. ControlDesk shows an occurred error message in the Message Viewer after validation. The XML file is always imported in the ODX database, regardless of whether validation is successful. ControlDesk uses the valid values, and the invalid values are discarded.

If you specified a global XML configuration file, ControlDesk also validates this global XML configuration file against the `DS_Service_Config.xsd` validation schema each time an ECU diagnostics project is created or loaded.

You can find the XSD file in the `.\Config` folder of your ControlDesk installation.

> **Note**
>
> Do not change the path and name of the XSD validation schema.

The `DS_Service_Config.xsd` schema might also be of help when you create your individual XML configuration file. Refer to Principle structure of the XML configuration file on page 97.

Related topics

Basics

Workflow for Creating an XML Configuration via the Diagnostics Instrument........................ 120

Examples

Example of an XML Configuration File.. 117

References

Logical Links Properties (ControlDesk Instrument Handling 🕮)

Configuring Services for Reading Fault Memory in the XML Configuration File

Introduction

In the XML file, you can specify several services for reading the fault memory. For example, you can specify different read services or configure different parameterizations for one read service. The specified read services are available for selection in the Fault Memory Instruments in ControlDesk.

FAULTREAD services

Each read service configuration is specified in a separate `SERVICE` element. The different `SERVICE` elements are collected in the `SERVICES` element, which is included in the `FAULTREAD` element.

Each `SERVICE` element for reading the fault memory must be configured as follows:

- It must contain the `IDENTIFICATION` element containing the identification string, either semantic, short name, or service ID, to search for read services. The `IDENTIFICATION` element has a `type` attribute to specify whether it is a semantic, a short name, or a service ID.
- It contains one or more `PARAM` elements specifying the necessary parameters. Each `PARAM` element must contain the `IDENTIFICATION` and `VALUE` elements. The `IDENTIFICATION` element has a `type` attribute to specify whether it is a semantic, a short name, or a byte position. The `VALUE` entry must be specified in hexadecimal notation, for example, 0x10.
- You can use the `EXT_PARAM` element to specify extended parameters beyond the ODX specification. It lets you identify a parameter in the FAULTREAD response whose value is to be used in an ENVREAD request. The `EXT_PARAM` element contains the `IDENTIFICATION` element. Since extended parameters are applied dynamically during run time, they do not have a `VALUE` element. Like the handling of DTC parameters, you can use an `EXT_PARAM` element to set additional information to a request.
- You can use the optional `displayName` attribute to specify the name to be displayed in the Read Service drop-down list in ControlDesk's Fault Memory Instrument for the service. The `displayName` value should be unique.

Services for which the `displayName` is not defined are displayed in ControlDesk according to the following pattern: `<ServiceShortName>:<Parameter1_ShortName>=<Value1>:<Parameter2_ShortName>=<Value2>`.

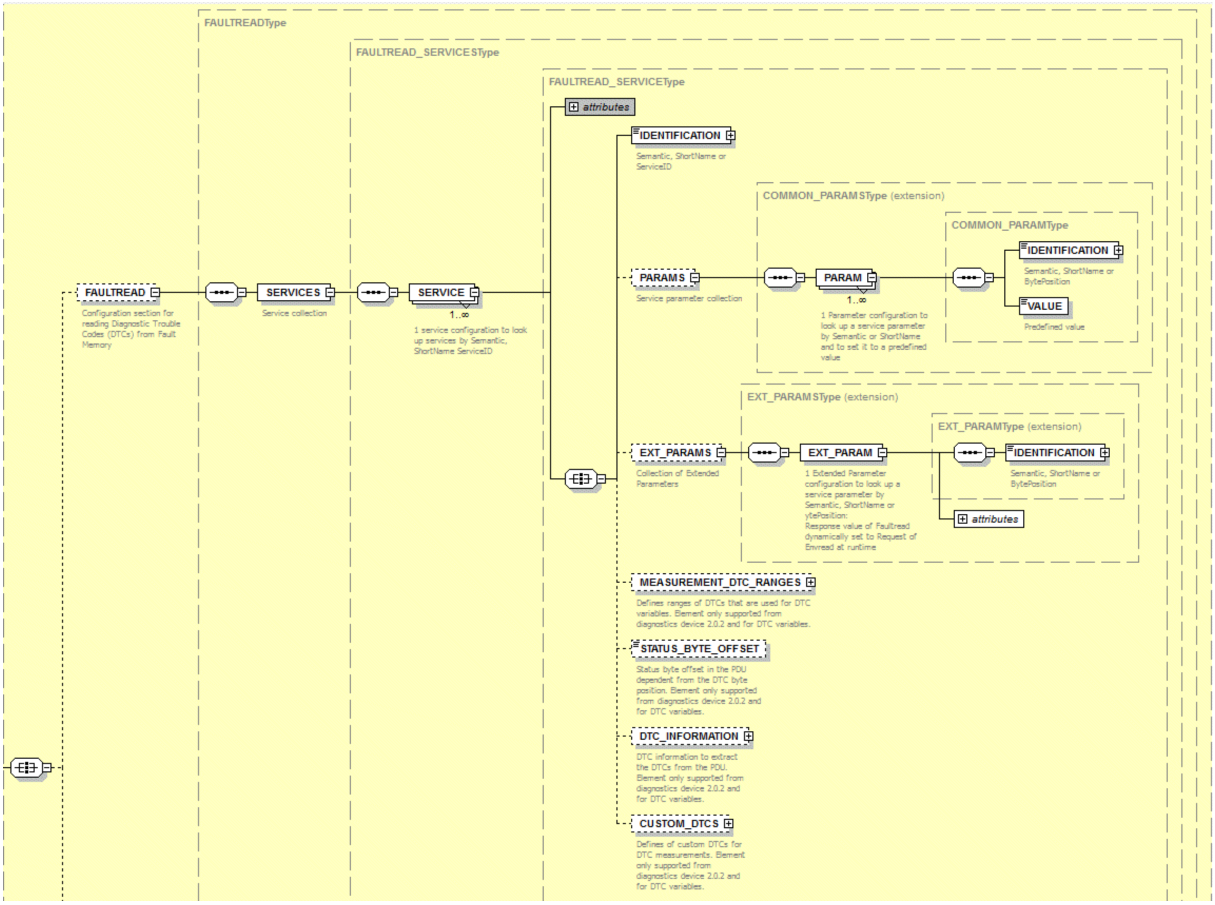
For services with identical `displayName` values, ControlDesk adds the postfix '#<number>' to the displayed service names to make the services distinguishable in the instruments.

> **Tip**
>
> To simplify service handling in ControlDesk, it is recommended to use the `displayName` attribute.

- You can use the `default` attribute to select a service for use as the default read service in ControlDesk's Fault Memory Instruments. The service must be classified with `default = "true"` for this. If several services are marked with `default = "true"`, ControlDesk uses the first service it finds.
- You can use one or more of the optional elements `MEASUREMENT_DTC_RANGES`, `STATUS_BYTE_OFFSET`, `DTC_INFORMATION`, and `CUSTOM_DTCS`. They are relevant only if you want to perform measurements on diagnostic trouble codes. They let you specify non-default service settings for DTC generation, i.e., settings different from the defaults according to the protocol-specific service identification. For further information, refer to Performing DTC Measurements on page 81.

Structure of the FAULTREAD element

The following diagram shows the required and optional elements for the `FAULTREAD` service configurations. Optional elements are represented by dashed lines.


FAULTREADType
FAULTREAD_SERVICESType
FAULTREAD_SERVICEType
attributes
FAULTREAD
Configuration section for reading Diagnostic Trouble Codes (DTCs) from Fault Memory
SERVICES
Service collection
SERVICE
1..∞
1 service configuration to look up services by Semantic, ShortName ServiceID
IDENTIFICATION
Semantic, ShortName or ServiceID
PARAMS
Service parameter collection
COMMON_PARAMSType (extension)
PARAM
1..∞
1 Parameter configuration to look up a service parameter by Semantic or ShortName and to set it to a predefined value
COMMON_PARAMType
IDENTIFICATION
Semantic, ShortName or BytePosition
VALUE
Predefined value
EXT_PARAMSType (extension)
EXT_PARAMS
Collection of Extended Parameters
EXT_PARAM
1 Extended Parameter configuration to look up a service parameter by Semantic, ShortName or ytePosition:
Response value of Faultread dynamically set to Request of Envread at runtime
EXT_PARAMType (extension)
IDENTIFICATION
Semantic, ShortName or BytePosition
attributes
MEASUREMENT_DTC_RANGES
Defines ranges of DTCs that are used for DTC variables. Element only supported from diagnostics device 2.0.2 and for DTC variables.
STATUS_BYTE_OFFSET
Status byte offset in the PDU dependent from the DTC byte position. Element only supported from diagnostics device 2.0.2 and for DTC variables.
DTC_INFORMATION
DTC information to extract the DTCs from the PDU. Element only supported from diagnostics device 2.0.2 and for DTC variables.
CUSTOM_DTCS
Defines of custom DTCs for DTC measurements. Element only supported from diagnostics device 2.0.2 and for DTC variables.

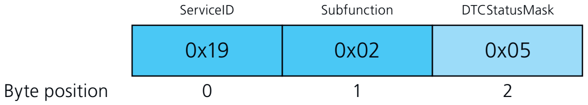
Examples

Service classified via service ID, parameters classified via byte position The following listing shows an example of a service configuration via service ID and byte position. It is an example for reading the fault memory using the UDS diagnostic protocol.

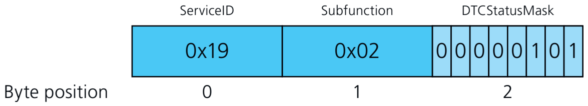
The following settings are used: service ID 0x19 = fault read service, byte position 1 = subfunction, byte position 2 = status byte.

```
...
  <FAULTREAD>
    <SERVICES>
      <SERVICE>
        <IDENTIFICATION type="ServiceID">0x19</IDENTIFICATION>
        <PARAMS>
          <PARAM>
            <IDENTIFICATION type="BytePosition">1</IDENTIFICATION>
             <VALUE>0x02</VALUE>
          </PARAM>
          <PARAM>
            <IDENTIFICATION type="BytePosition">2</IDENTIFICATION>
             <VALUE>0x05</VALUE>
          </PARAM>
        </PARAMS>
      </SERVICE>
    ...
```

Parameter identification via the byte position means using all the parameters that are located at or start at the specified byte position. If there is only one parameter whose length is 1 byte, the specified value is applied to it.



If more than one parameter is defined within the byte at the specified byte position, the value is applied to the parameters according to the bit positions.



For a parameter whose length is more than 1 byte, the specified value is set accordingly over the bytes.

Service and parameters classified via the SEMANTIC attribute For an example of a service configuration for reading fault memory information via ODX semantics, refer to Example of an XML Configuration File on page 117.

Related topics

Basics

Reading the Fault Memory of an ECU.. 56

Configuring Services for Reading Environment Data in the XML Configuration File

Introduction

In the XML file, you can specify several services for reading environment data that corresponds to a specific fault memory entry. The specified services are available for selection via the **Properties** controlbar of a Fault Memory instrument in ControlDesk. Refer to Logical Links Properties (ControlDesk Instrument Handling 🕮).

ENVREAD services

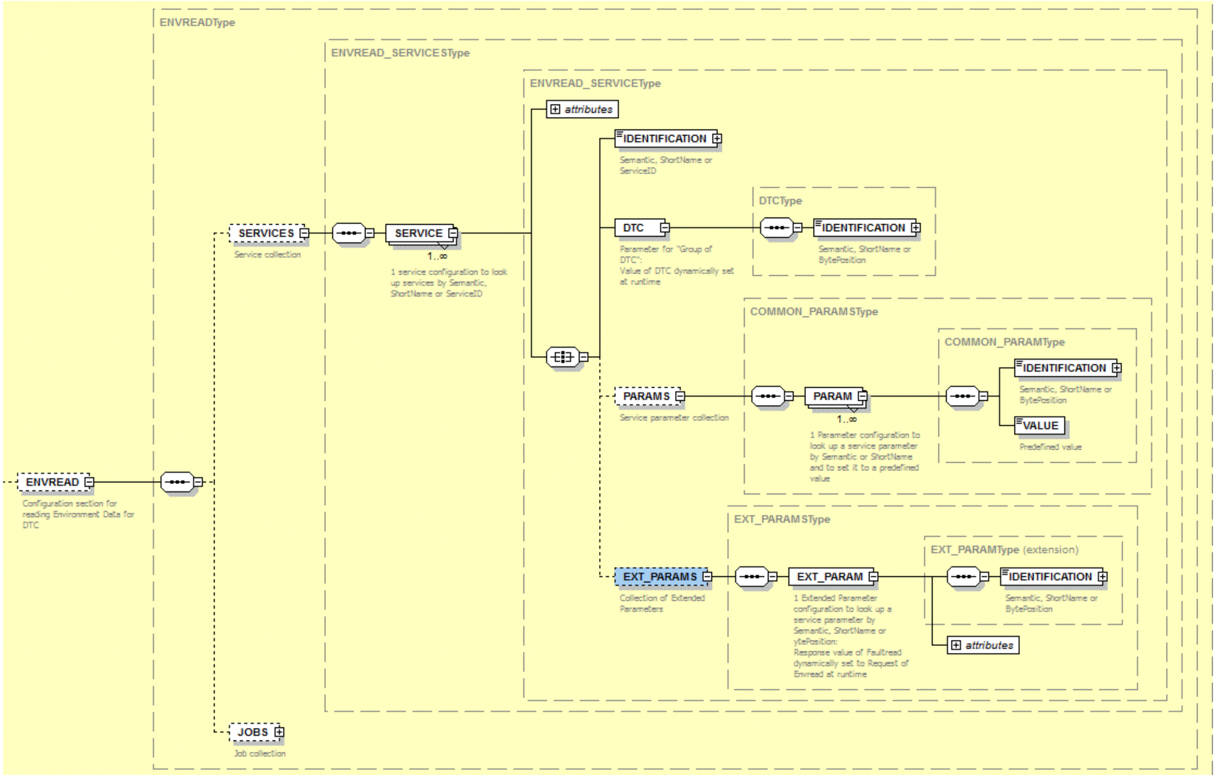
Each service is configured in a separate SERVICE element. The different SERVICE elements are collected in the SERVICES element, which is included in the ENVREAD element.

Each SERVICE element for reading environment data must be configured as follows:

- It must contain the IDENTIFICATION element containing the identification string, either semantic, short name, or service ID, to search for environment data read services. The IDENTIFICATION element has a type attribute to specify whether it is a semantic, a short name, or a service ID.
- It must contain the DTC element. The service is used to read the environment data for a specific fault memory entry. The connection to the fault memory entry is established via the DTC entry. The DTC element must contain the IDENTIFICATION element containing the identification string. The IDENTIFICATION element has a type attribute to specify whether it is a semantic, a short name, or a byte position.
- It contains one or more PARAM elements specifying the necessary parameters. Each PARAM element must contain the IDENTIFICATION and VALUE elements. The IDENTIFICATION element has a type attribute to specify whether it is a semantic, a short name, or a byte position. The VALUE entry must be specified in hexadecimal notation.
- You can include the EXT_PARAM element to specify extended parameters beyond the DTC specification. This allows you to transfer an additional parameter besides the DTC when reading environment data, if the EXT_PARAM element is also included in the respective read service definition. The EXT_PARAM element contains the IDENTIFICATION element. Since extended parameters are applied dynamically during run time, they do not have a VALUE element. Like the handling of DTC parameters, you can use an EXT_PARAM element to add additional information.

Structure of the ENVREAD element

The following diagram shows the required and optional elements for the ENVREAD service configurations. Optional elements are represented by dashed lines.


ENVREADType
ENVREAD_SERVICESType
ENVREAD_SERVICEType
attributes
IDENTIFICATION
Semantic, ShortName or ServiceID
DTCType
DTC
IDENTIFICATION
Semantic, ShortName or BytePosition
Parameter for "Group of DTC":
Value of DTC dynamically set at runtime
SERVICES
Service collection
SERVICE
1..∞
1 service configuration to look up services by Semantic, ShortName or ServiceID
COMMON_PARAMSType
COMMON_PARAMType
IDENTIFICATION
Semantic, ShortName or BytePosition
PARAMS
Service parameter collection
PARAM
1..∞
VALUE
Predefined value
1 Parameter configuration to look up a service parameter by Semantic or ShortName and to set it to a predefined value
ENVREAD
Configuration section for reading Environment Data for DTC
EXT_PARAMSType
EXT_PARAMType (extension)
EXT_PARAMS
EXT_PARAM
IDENTIFICATION
Collection of Extended Parameters
Semantic, ShortName or BytePosition
1 Extended Parameter configuration to look up a service parameter by Semantic, ShortName or ytePosition:
Response value of Faultread dynamically set to Request of Envread at runtime
attributes
JOBS
Job collection

Tip

In addition to or as an alternative to services, you can specify diagnostic jobs for reading environment data in the XML configuration file.

Each job is configured in a separate `JOB` element. The different `JOB` elements are collected in the `JOBS` element, which is included in the `ENVREAD` element.

A `JOB` element for reading environment data must be configured in the same way as a `SERVICE` element. (There are two exceptions: the `IDENTIFICATION` element containing the identification string to search for jobs for reading environment data cannot be of `serviceID` type, and the `IDENTIFICATION` elements of the `DTC`, `PARAM` and `EXT_PARAM` elements cannot be of `BytePosition` type.)

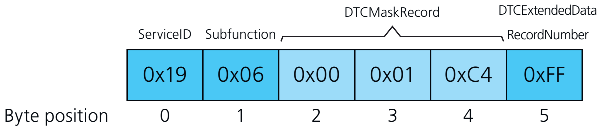
Examples

Service classified via service ID, parameters classified via byte position The following listing shows an example of a service configuration via service ID and byte position. It is an example for reading environment data using the UDS diagnostic protocol.

```
...
  <ENVREAD>
    <SERVICES>
      <SERVICE>
        <IDENTIFICATION type="ServiceID">0x19</IDENTIFICATION>
        <PARAMS>
          <PARAM>
            <IDENTIFICATION type="BytePosition">1</IDENTIFICATION>
             <VALUE>0x06</VALUE>
          </PARAM>
          <PARAM>
            <IDENTIFICATION type="BytePosition">2</IDENTIFICATION>
             <VALUE>0x0001C4</VALUE>
          </PARAM>
        </PARAMS>
      </SERVICE>
    ...
```

Parameter identification via the byte position means using all the parameters that are located or start at the specified byte position. (The indices on byte positions start at 0.) If there is only one parameter whose length is 1 byte, the specified value is applied to it. If more than one parameter is defined within the byte at the specified byte position, the value is applied to the parameters according to the bit positions. For a parameter which is longer than 1 byte, the specified value is set accordingly over the bytes.

In the example, the `DTCMaskRecord` parameter has a length of 3 bytes, so the specified value is set over the three bytes.



Service and parameters classified via the SEMANTIC attribute For an example of a service configuration via ODX semantics for reading environment data, refer to Example of an XML Configuration File on page 117.

Related topics

Basics

Configuring Services for Clearing Single Fault Memory Entries in the XML Configuration File

Introduction

In the XML file, you can specify several services for clearing *single* fault memory entries. ControlDesk uses them when you click **Clear Selected DTC** in a Fault Memory instrument to delete the selected trouble code entry from the fault memory. The specified services are available for selection via the **Properties** controlbar of a Fault Memory instrument in ControlDesk. Refer to Logical Links Properties (ControlDesk Instrument Handling 🕮).

FAULTCLEAR_SINGLE services

Each service is configured in a separate `SERVICE` element. The different `SERVICE` elements are collected in the `SERVICES` element, which is included in the `FAULTCLEAR_SINGLE` element.

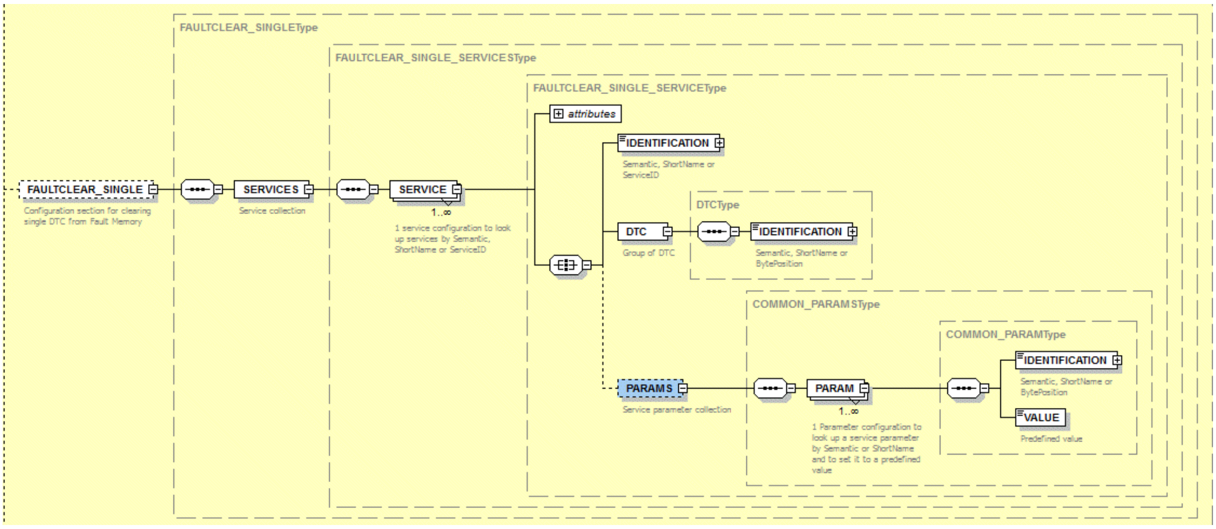
> **Note**
>
> If the XML file does not contain a service configuration for clearing a single fault memory entry, ControlDesk uses the service configuration for clearing all fault memory entries.

Each `SERVICE` element for clearing single fault memory entries must be configured as follows:

- It must contain the `IDENTIFICATION` element containing the identification string used for identifying services for clearing single fault memory entries. The `IDENTIFICATION` element has a `type` attribute to specify whether it is a semantic, a short name, or a service ID.
- It must contain the `DTC` element for DTC identification purposes. The `DTC` element must contain the `IDENTIFICATION` element containing the identification string. The `IDENTIFICATION` element has a `type` attribute to specify whether it is a semantic, a short name, or a byte position.
- It can contain `PARAM` elements specifying the necessary parameters. Each `PARAM` element must contain the `IDENTIFICATION` and `VALUE` elements. The `IDENTIFICATION` element has a `type` attribute to specify whether it is a semantic, a short name, or a byte position. The `VALUE` entry must be specified in hexadecimal notation, for example, 0xFF.

Structure of the FAULTCLEAR_SINGLE element

The following diagram shows the required and optional elements for the `FAULTCLEAR_SINGLE` service configurations. Optional elements are represented by dashed lines.


FAULTCLEAR_SINGLEType
FAULTCLEAR_SINGLE_SERVICESType
FAULTCLEAR_SINGLE_SERVICEType
attributes
FAULTCLEAR_SINGLE
Configuration section for clearing single DTC from Fault Memory
SERVICES
Service collection
SERVICE
1..∞
1 service configuration to look up services by Semantic, ShortName or ServiceID
IDENTIFICATION
Semantic, ShortName or ServiceID
DTCType
DTC
Group of DTC
IDENTIFICATION
Semantic, ShortName or BytePosition
COMMON_PARAMSType
COMMON_PARAMType
PARAMS
Service parameter collection
PARAM
1..∞
1 Parameter configuration to look up a service parameter by Semantic or ShortName and to set it to a predefined value
IDENTIFICATION
Semantic, ShortName or BytePosition
VALUE
Predefined value

Example

For an example of a service configuration for clearing single fault memory entries, refer to Example of an XML Configuration File on page 117.

Related topics

Basics

Basics of the XML Configuration File.. 93
Clearing Fault Memory Entries.. 60
Configuring Services for Clearing All Fault Memory Entries in the XML Configuration File.. 108

Configuring Services for Clearing All Fault Memory Entries in the XML Configuration File

Introduction

In the XML file, you can specify several services for clearing *all* fault memory entries. ControlDesk uses them when you click Clear All DTCs in a Fault Memory instrument to delete all trouble code entries from the fault memory. The specified services are available for selection via the Properties controlbar of a Fault Memory instrument in ControlDesk. Refer to Logical Links Properties (ControlDesk Instrument Handling 🕮).

FAULTCLEAR_ALL services

Each service is configured in a separate `SERVICE` element. The different `SERVICE` elements are collected in the `SERVICES` element, which is included in the `FAULTCLEAR_ALL` element.

Note

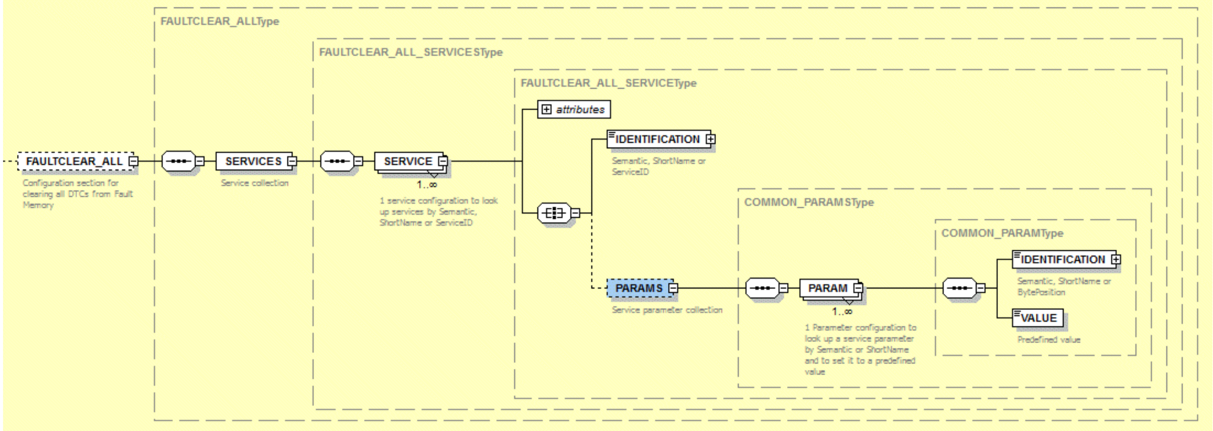
If the XML file does not contain a service configuration for clearing all fault memory entries, ControlDesk uses the service configuration for clearing a single fault memory entry.

Each `SERVICE` element for clearing all fault memory entries must be configured as follows:

- It must contain the `IDENTIFICATION` element containing the identification string used for identifying services for clearing all fault memory entries. The `IDENTIFICATION` element has a `type` attribute to specify whether it is a semantic, a short name, or a service ID.
- It can contain `PARAM` elements specifying the necessary parameters. Each `PARAM` element must contain the `IDENTIFICATION` and `VALUE` elements. The `IDENTIFICATION` element has a `type` attribute to specify whether it is a semantic, a short name, or a byte position. The `VALUE` entry must be specified in hexadecimal notation, for example, 0xFF.

Structure of the FAULTCLEAR_ALL element

The following diagram shows the required and optional elements for the `FAULTCLEAR_ALL` service configurations. Optional elements are represented by dashed lines.


FAULTCLEAR_ALLType
FAULTCLEAR_ALL_SERVICESType
FAULTCLEAR_ALL_SERVICEType
attributes
FAULTCLEAR_ALL
Configuration section for clearing all DTCs from Fault Memory
SERVICES
Service collection
SERVICE
1..∞
1 service configuration to look up services by Semantic, ShortName or ServiceID
IDENTIFICATION
Semantic, ShortName or ServiceID
PARAMS
Service parameter collection
COMMON_PARAMSType
PARAM
1..∞
1 Parameter configuration to look up a service parameter by Semantic or ShortName and to set it to a predefined value
COMMON_PARAMType
IDENTIFICATION
Semantic, ShortName or BytePosition
VALUE
Predefined value

Example

For an example of a service configuration for clearing all fault memory entries, refer to Example of an XML Configuration File on page 117.

Related topics

Basics

Configuring Services for Measurement and Calibration in the XML Configuration File

Introduction

In the XML file, you can specify several services for reading diagnostics variables or writing diagnostics variables via the ECU Diagnostics device.

MEASUREMENT and CALIBRATION services

Each service is configured in a separate `SERVICE` element. The different `SERVICE` elements are collected in the `SERVICES` elements, which are included in the `MEASUREMENT` or `CALIBRATION` elements.

Each `SERVICE` element must be configured as follows:

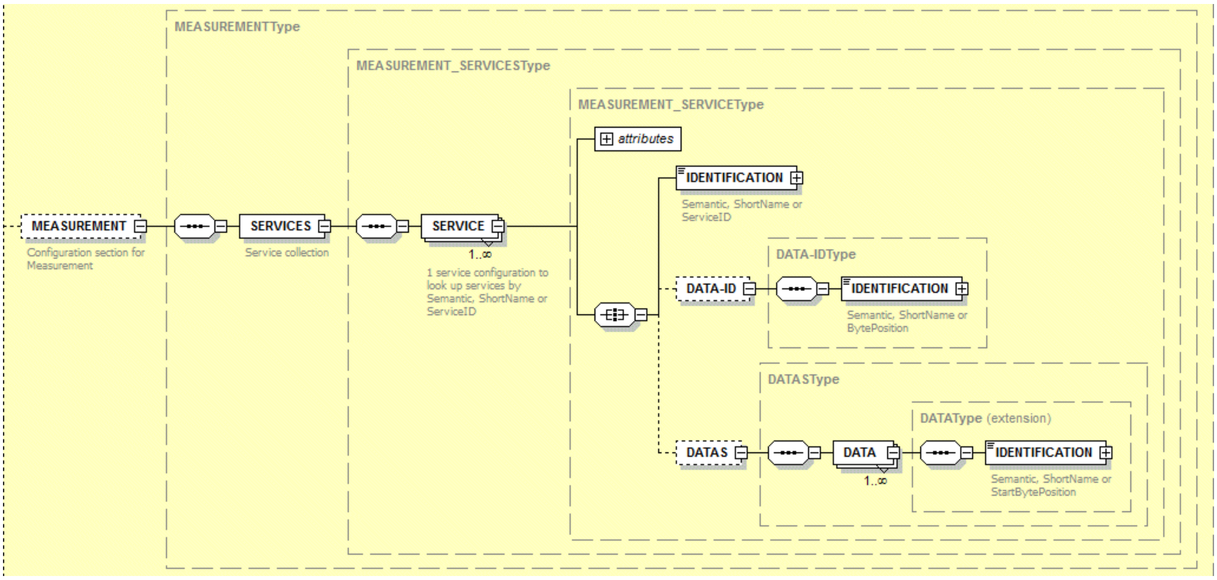
- It must contain the `IDENTIFICATION` element containing the identification string, either semantic, short name, or service ID, to search for read or write services. The `IDENTIFICATION` element has a `type` attribute to specify whether it is a semantic, a short name, or a service ID.
- It must contain the `DATA-ID` element specifying the service's request parameter and response parameter containing the data identifier. The `DATA-ID` element must contain the `IDENTIFICATION` element containing the identification string. The `IDENTIFICATION` element has a `type` attribute to specify whether it is a semantic, a short name, or a byte position.
- It must contain one or more `DATA` elements to specify the diagnostics variables. One `DATA` element must be configured for each measurement variable/parameter that corresponds to a response parameter in a read service or to a request parameter in a write service. Each `DATA` element must contain the `IDENTIFICATION` element. The `IDENTIFICATION` element has a `type` attribute to specify whether it is a semantic, a short name, or a start byte position.

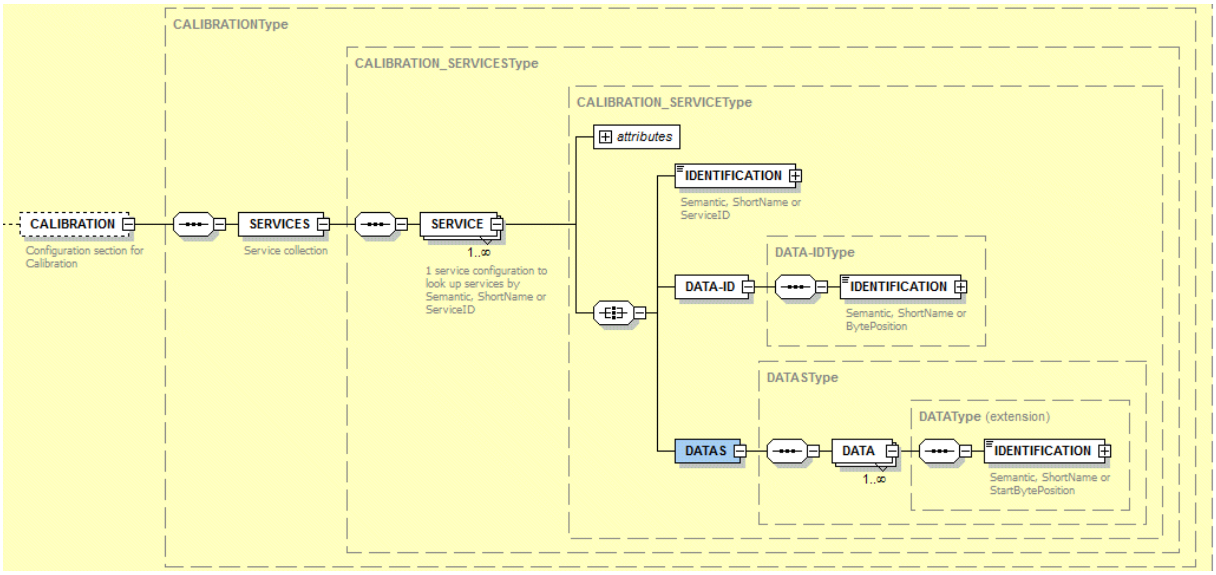
  If several measurement variables/parameters are defined in a service and if they are united, for example, in a TABLE-STRUCT element, it is sufficient to specify the `IDENTIFICATION` attribute for the element, as all underlying measurement variables/parameters inherit this attribute automatically. However, if you want to use individual diagnostics variables of a structure element in ControlDesk, make sure that the structure element itself does not contain the `IDENTIFICATION` attribute.
- Each parameter in a write service must have a corresponding measurement variable in a read service. The read service and the write service must hold `DATA-ID`- and `DATA`-related information having the same structure. The request parameter and the response parameter that are classified with the `DATA-ID` or `DATA` elements must have identical names in the read service and the write service.

  If there is no matching measurement variable in a read service, the corresponding parameter in the write service is discarded.

Structure of the MEASUREMENT and CALIBRATION elements

The following diagrams show the elements for the `MEASUREMENT` and `CALIBRATION` service configurations.


MEASUREMENTType
MEASUREMENT_SERVICESType
MEASUREMENT_SERVICEType
attributes
MEASUREMENT
Configuration section for Measurement
SERVICES
Service collection
SERVICE
1..∞
1 service configuration to look up services by Semantic, ShortName or ServiceID
IDENTIFICATION
Semantic, ShortName or ServiceID
DATA-IDType
DATA-ID
IDENTIFICATION
Semantic, ShortName or BytePosition
DATASType
DATAType (extension)
DATAS
DATA
1..∞
IDENTIFICATION
Semantic, ShortName or StartBytePosition


CALIBRATIONType
CALIBRATION_SERVICESType
CALIBRATION_SERVICEType
attributes
CALIBRATION
Configuration section for Calibration
SERVICES
Service collection
SERVICE
1..∞
1 service configuration to look up services by Semantic, ShortName or ServiceID
IDENTIFICATION
Semantic, ShortName or ServiceID
DATA-IDType
DATA-ID
IDENTIFICATION
Semantic, ShortName or BytePosition
DATASType
DATAType (extension)
DATAS
DATA
1..∞
IDENTIFICATION
Semantic, ShortName or StartBytePosition

Examples

Service classified via service ID, data ID classified via byte position, measurement variables/parameters classified via start byte position Each diagnostic variable that corresponds to a response parameter in a read service or to a request parameter in a write service is specified in one DATA element. Identifying a DATA element via the start byte position means using all the parameters that are located in the PDU from the start byte position onwards as measurement variables/parameters in ControlDesk.

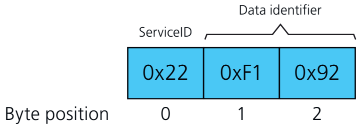
- Identifying diagnostics variables from a read service

  The following listing shows an example of a service configuration for reading diagnostics variables where the start byte position is used for identifying measurement variables.

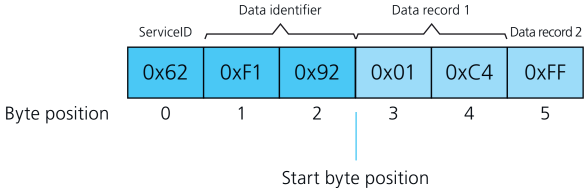
```
...
  <MEASUREMENT>
    <SERVICES>
      <SERVICE>
        <IDENTIFICATION type="ServiceID">0x22</IDENTIFICATION>
        <DATA-ID>
          <IDENTIFICATION type="BytePosition">1</IDENTIFICATION>
        </DATA-ID>
        <DATAS>
          <DATA>
            <IDENTIFICATION type="StartBytePosition">3</IDENTIFICATION>
          </DATA>
        </DATAS>
      </SERVICE>
    ...
```

  Suppose you work with the UDS diagnostic protocol, and you have specified to use this service configuration in ControlDesk for performing measurements via the ECU Diagnostics device. The start byte position is set to 3 in the service configuration, so the response parameters located in the data records starting at byte position 3 in the response PDU are used as the measurement variables.

  Request PDU:



  Response PDU:



  For example, if you execute the `ReadDataByIdentifier` service with 'ClimateControlData' selected as the data identifier in the Diagnostics instrument, all the measurement variables belonging to this data identifier are returned for use in ControlDesk.

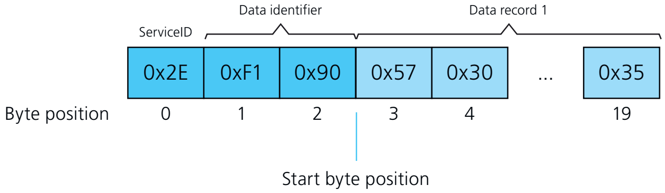
- Identifying parameters from a write service

  The following listing shows an example of a service configuration for writing diagnostics variables that uses the start byte position for identification purposes.

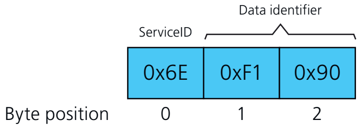
```
...
  <CALIBRATION>
    <SERVICES>
      <SERVICE>
        <IDENTIFICATION type="ServiceID">0x2E</IDENTIFICATION>
        <DATA-ID>
          <IDENTIFICATION type="BytePosition">1</IDENTIFICATION>
        </DATA-ID>
        <DATAS>
          <DATA>
            <IDENTIFICATION type="StartBytePosition">3</IDENTIFICATION>
          </DATA>
        </DATAS>
      </SERVICE>
    ...
```

Suppose you work with the UDS diagnostic protocol, and you have specified to use this service configuration in ControlDesk for performing calibration via the ECU Diagnostics device. The start byte position is set to 3 in the service configuration, so you can find the parameters for calibration in the data records starting at byte position 3 in the request PDU. In the example, there is only one parameter (with a length of 17 bytes) in the write service.

Request PDU:



Response PDU:



For example, if you execute the `WriteDataByIdentifier` service with 'Vehicle Identification Number' selected as the data identifier in the Diagnostics instrument, all the parameters belonging to this data identifier are used in ControlDesk.

Services and parameters classified via the SEMANTIC attribute For an example of a service configuration via ODX semantics for measuring and calibrating diagnostics variables, refer to Example of an XML Configuration File on page 117.

Related topics

Basics

Configuring Services for Getting Security Access in the XML Configuration File

Introduction

In the XML file, you can specify one diagnostic job for preparing the ECU for calibration access. The configured element can be executed automatically when online calibration is started for the ECU Diagnostics device, letting you calibrate parameters in ControlDesk when the device is online. You can specify whether the job is executed each time online calibration is started for the device, or only if a variable description was generated from the ODX database specified for the device.

SECURITY_ONLINE services

The job for preparing the ECU for calibration access must be specified in one `SERVICE` element. The `SERVICE` element typically configures the security access job which performs both changing the ECU mode and security access.

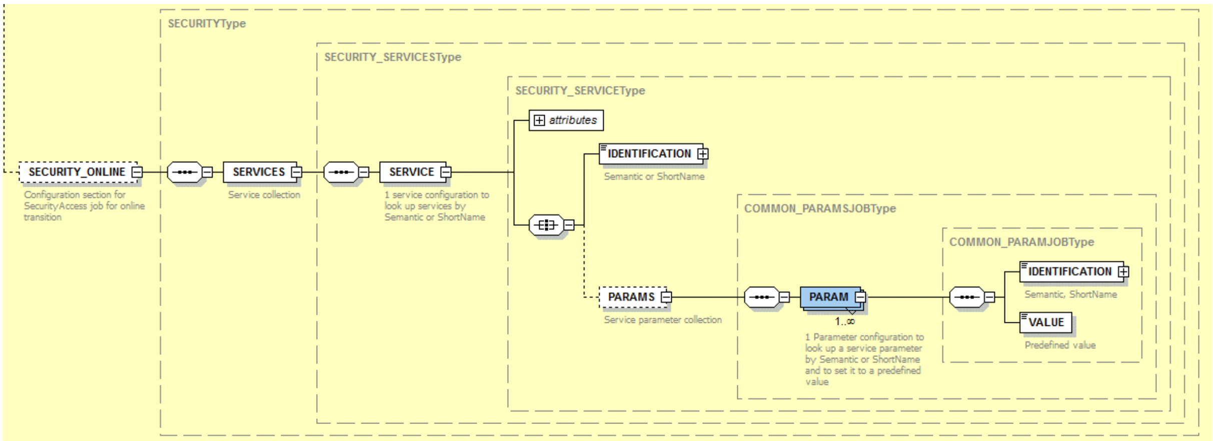
The `SERVICE` element is included in the `SERVICES` element, which is included in the `SECURITY_ONLINE` element.

The `SERVICE` element must be configured as follows:

- It must contain the `IDENTIFICATION` element containing the identification string, either semantic or short name, to search for a service to change the diagnostics mode or a job providing security access. The `IDENTIFICATION` element has a `type` attribute to specify whether it is a semantic or a short name.
- It can contain `PARAM` elements specifying the necessary parameters. Each `PARAM` element must contain the `IDENTIFICATION` and `VALUE` elements. The `IDENTIFICATION` element has a `type` attribute to specify whether it is a semantic or a short name.

Structure of the SECURITY_ONLINE element

The following diagram shows the required and optional elements for the `SECURITY_ONLINE` service configurations. Optional elements are represented by dashed lines.


SECURITYType
SECURITY_SERVICESType
SECURITY_SERVICEType
attributes
IDENTIFICATION
Semantic or ShortName
SECURITY_ONLINE
Configuration section for SecurityAccess job for online transition
SERVICES
Service collection
SERVICE
1 service configuration to look up services by Semantic or ShortName
COMMON_PARAMSJOBType
COMMON_PARAMJOBType
IDENTIFICATION
Semantic, ShortName
PARAMS
Service parameter collection
PARAM
1..∞
1 Parameter configuration to look up a service parameter by Semantic or ShortName and to set it to a predefined value
VALUE
Predefined value

Example

For an example of a service configuration for getting security access, refer to Example of an XML Configuration File on page 117.

Related topics

Basics

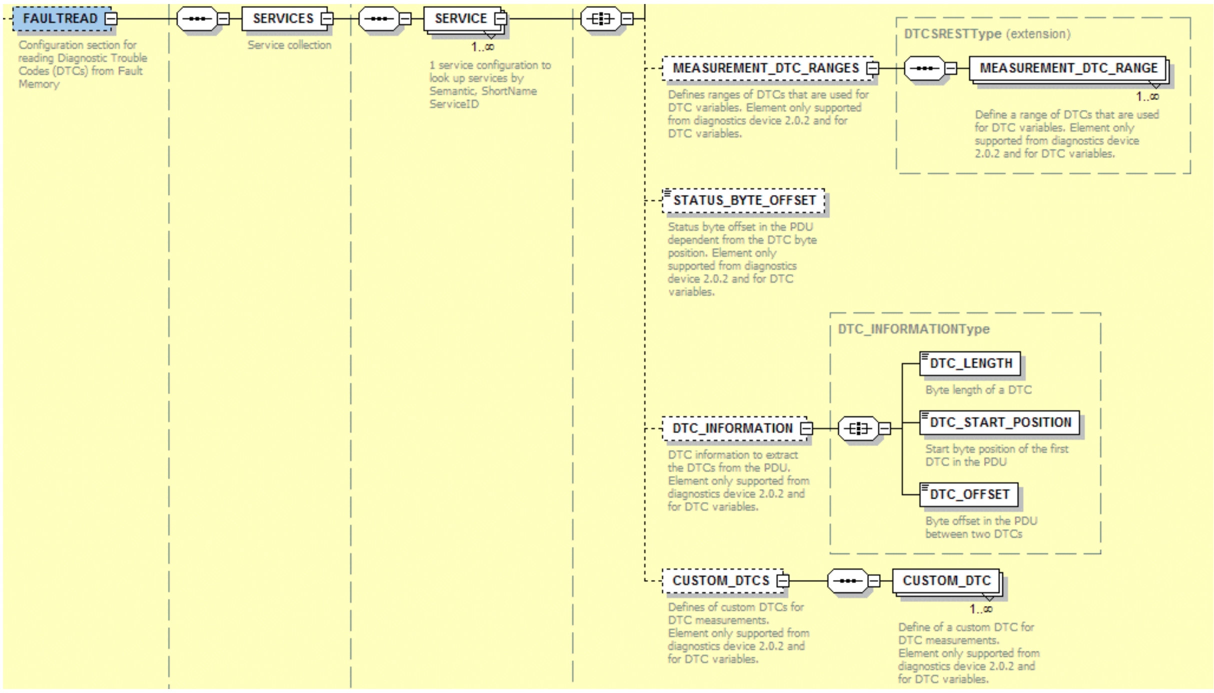
Configuring Services for DTC Measurements in the XML Configuration File

Introduction

The service configurations for performing DTC measurements are part of the service configurations for reading fault memory. So if you want to use DTC information or a DTC status byte offset value different from the default service settings specified according to the standard protocol-specific service identification, or if you want to define DTC ranges or custom DTCs to be used in ControlDesk, you must extend the `SERVICE` element for reading the fault memory.

Structure of the elements relevant for DTC measurements

The following diagram shows the required and optional elements for the service configurations for DTC measurements. Optional elements are represented by dashed lines.



FAULTREAD: Configuration section for reading Diagnostic Trouble Codes (DTCs) from Fault Memory

SERVICES: Service collection

SERVICE: 1 service configuration to look up services by Semantic, ShortName ServiceID

MEASUREMENT_DTC_RANGES: Defines ranges of DTCs that are used for DTC variables. Element only supported from diagnostics device 2.0.2 and for DTC variables.

MEASUREMENT_DTC_RANGE: Define a range of DTCs that are used for DTC variables. Element only supported from diagnostics device 2.0.2 and for DTC variables.

STATUS_BYTE_OFFSET: Status byte offset in the PDU dependent from the DTC byte position. Element only supported from diagnostics device 2.0.2 and for DTC variables.

DTC_LENGTH: Byte length of a DTC

DTC_INFORMATION: DTC information to extract the DTCs from the PDU. Element only supported from diagnostics device 2.0.2 and for DTC variables.

DTC_START_POSITION: Start byte position of the first DTC in the PDU

DTC_OFFSET: Byte offset in the PDU between two DTCs

CUSTOM_DTCS: Defines of custom DTCs for DTC measurements. Element only supported from diagnostics device 2.0.2 and for DTC variables.

CUSTOM_DTC: Define of a custom DTC for DTC measurements. Element only supported from diagnostics device 2.0.2 and for DTC variables.

MEASUREMENT_DTC_RANGES

In the XML file, you can restrict the number of DTC variables to be generated. You must specify a range for the DTC variables that are to be generated by specifying a range of DTC numbers, DTC levels, etc.

Use one or more `MEASUREMENT_DTC_RANGE` elements to specify ranges of DTCs for which DTC variables are to be generated.

STATUS_BYTE_OFFSET

ControlDesk generates a status byte variable for each DTC. The offset is the number of bytes beginning with the first byte of the DTC parameter and ending with the byte before the status byte (see Performing DTC Measurements on page 81).

There are default status byte offsets according to the standard protocol specification (see above). However, if the default setting does not match your requirements, you can specify a non-default offset for the DTC status byte in the XML file.

DTC_INFORMATION

ControlDesk uses DTC information to extract DTCs from response PDUs (see Performing DTC Measurements on page 81).

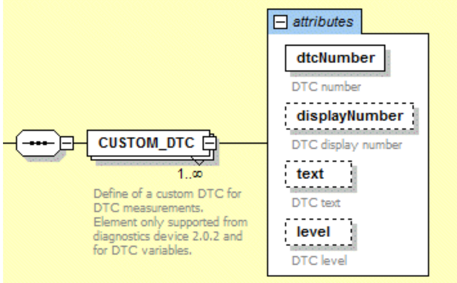
There are default DTC information according to the standard protocol specification. However, if this default information does not match your requirements, you can specify non-default DTC information in the XML file.

Use the `DTC_INFORMATION` elements to specify the DTC information to be used. The `DTC_INFORMATION` element must contain the `DTC_LENGTH`, `DTC_START_POSITION`, and `DTC_OFFSET` elements.

CUSTOM_DTCS

Custom DTCs are DTCs that are defined outside the ODX database. You can define custom DTCs in the XML configuration file. Custom DTCs can be used for DTC measurements like DTCs from the ODX database.

Use one or more `CUSTOM_DTC` elements to specify the required diagnostic trouble codes outside the ODX database. Each `CUSTOM_DTC` element must have the `dtcNumber` attribute to specify the DTC number. Further optional attributes (`displayNumber`, `text`, `level`) are available.


attributes
dtcNumber
DTC number
displayNumber
DTC display number
text
DTC text
level
DTC level
CUSTOM_DTC
1..∞
Define of a custom DTC for DTC measurements. Element only supported from diagnostics device 2.0.2 and for DTC variables.

For further information on custom DTCs, refer to Performing DTC Measurements on page 81.

Related topics

Basics

Basics of the XML Configuration File..93
Performing DTC Measurements..81

Example of an XML Configuration File

Example

The following listing is an example of an XML configuration file for use with ECU Diagnostics v2.0.2 devices. It contains one service specification each for reading the fault memory, reading environment data, clearing single fault memory entries, clearing all fault memory entries, reading diagnostics variables, writing diagnostics variables, and changing to the ECU diagnostics mode. It comprises the service configurations for all logical links in one single configuration (`FUNCTION_SET` element). It shows how services and parameters are identified for the corresponding ControlDesk functions.

```
<?xml version="1.0" encoding="UTF-8"?>
<Service-Config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <DESC name="ControlDesk NG Service-Config Default"></DESC>
  <FUNCTION_SET>
    <FUNCTIONS>
      <FAULTREAD>
        <SERVICES>
          <SERVICE displayName="DiagDemo_FAULTREAD">
            <IDENTIFICATION type="SEMANTIC">FAULTREAD</IDENTIFICATION>
            <PARAMS>
              <PARAM>
                <IDENTIFICATION type="SEMANTIC">SUBFUNCTION</IDENTIFICATION>
                <VALUE>0x02</VALUE>
              </PARAM>
            </PARAMS>
          </SERVICE>
        </SERVICES>
      </FAULTREAD>
      <ENVREAD>
        <SERVICES>
          <SERVICE displayName="DiagDemo_ENVREAD">
            <IDENTIFICATION type="SEMANTIC">FAULTREAD</IDENTIFICATION>
            <DTC>
              <IDENTIFICATION type="SEMANTIC">DATA</IDENTIFICATION>
            </DTC>
            <PARAMS>
              <PARAM>
                <IDENTIFICATION type="SEMANTIC">SUBFUNCTION</IDENTIFICATION>
                <VALUE>0x06</VALUE>
              </PARAM>
            </PARAMS>
          </SERVICE>
        </SERVICES>
      </ENVREAD>
      <FAULTCLEAR_SINGLE>
        <SERVICES>
          <SERVICE displayName="DiagDemo_FAULTCLEAR_SINGLE">
            <IDENTIFICATION type="SEMANTIC">FAULTCLEAR</IDENTIFICATION>
            <DTC>
              <IDENTIFICATION type="ShortName">PA_groupOfDTC</IDENTIFICATION>
            </DTC>
          </SERVICE>
        </SERVICES>
      </FAULTCLEAR_SINGLE>
      <FAULTCLEAR_ALL>
        <SERVICES>
          <SERVICE displayName="DiagDemo_FAULTCLEAR_ALL">
            <IDENTIFICATION type="SEMANTIC">FAULTCLEAR</IDENTIFICATION>
            <PARAMS>
              <PARAM>
                <IDENTIFICATION type="ShortName">PA_groupOfDTC</IDENTIFICATION>
                <VALUE>0xFFFFFF</VALUE>
              </PARAM>
            </PARAMS>
          </SERVICE>
        </SERVICES>
      </FAULTCLEAR_ALL>
      <MEASUREMENT>
        <SERVICES>
          <SERVICE displayName="DefaultSemantics">
            <IDENTIFICATION type="SEMANTIC">CURRENTDATA</IDENTIFICATION>
            <DATA-ID>
              <IDENTIFICATION type="SEMANTIC">DATA-ID</IDENTIFICATION>
            </DATA-ID>
            <DATAS>
              <DATA>
                <IDENTIFICATION type="SEMANTIC">DATA</IDENTIFICATION>
              </DATA>
            </DATAS>
          </SERVICE>
        </SERVICES>
      </MEASUREMENT>
      <CALIBRATION>
        <SERVICES>
          <SERVICE displayName="DefaultSemantics">
            <IDENTIFICATION type="SEMANTIC">CALIBRATION</IDENTIFICATION>
            <DATA-ID>
              <IDENTIFICATION type="SEMANTIC">DATA-ID</IDENTIFICATION>
            </DATA-ID>
            <DATAS>
              <DATA>
                <IDENTIFICATION type="SEMANTIC">DATA</IDENTIFICATION>
              </DATA>
            </DATAS>
          </SERVICE>
        </SERVICES>
      </CALIBRATION>
      <SECURITY_ONLINE>
        <SERVICES>
          <SERVICE displayName="DefaultSemantics">
            <IDENTIFICATION type="SEMANTIC">SECURITY</IDENTIFICATION>
          </SERVICE>
        </SERVICES>
      </SECURITY_ONLINE>
    </FUNCTIONS>
  </FUNCTION_SET>
</Service-Config>
```

Using the example as a template

You can create an XML configuration file out of the example shown above, and then use the file as a template when creating your own user-specific XML configuration file. You have to perform the following steps:

- Copy the example to an empty XML file.
- Save the file according to the XML file name convention (see Naming convention for the XML configuration file on page 99).
- Adapt the XML file contents to your requirements.

Tip

The example above can be used with the ECU Diagnostics demo projects which come with ControlDesk. Import the XML file into the ODX diagnostics database of the ECU Diagnostics device for this.

Related topics

Basics

Workflow for Creating an XML Configuration via the Diagnostics Instrument

Introduction

When you work with the Fault Memory Instrument, you might want to use a different fault memory service than the default diagnostic services. ControlDesk enables you to make a fault memory service that is parameterized in the Diagnostics Instrument available as XML configuration for use in an XML configuration file in ControlDesk projects. After the XML configuration file has been added to the ECU Diagnostics device configuration, the newly configured service is available in the Fault Memory Instrument in addition to the default services.

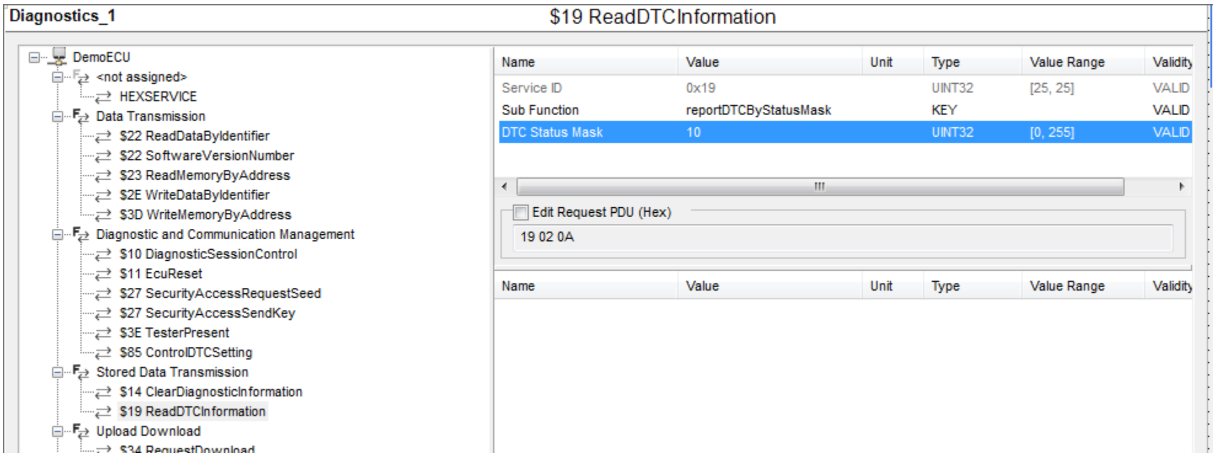
Workflow

The following workflow shows the required steps for creating an XML configuration via the Diagnostics Instrument. The workflow is described using the example of an XML service configuration for reading the fault memory. It is assumed that no XML configuration file was previously added to the device configuration.

1. Configure the service that you want to use in addition to the default services in the Fault Memory Instrument.

   To do so, open the Diagnostics Instrument and select the service that you want to use in the Fault Memory Instrument and configure it appropriately in the parameter list.

   The following illustration shows the configuration of the ReadDTCByStatusMask service as an example:



Diagnostics_1

$19 ReadDTCInformation

Name	Value	Unit	Type	Value Range	Validity
Service ID	0x19		UINT32	[25, 25]	VALID
Sub Function	reportDTCByStatusMask		KEY		VALID
DTC Status Mask	10		UINT32	[0, 255]	VALID

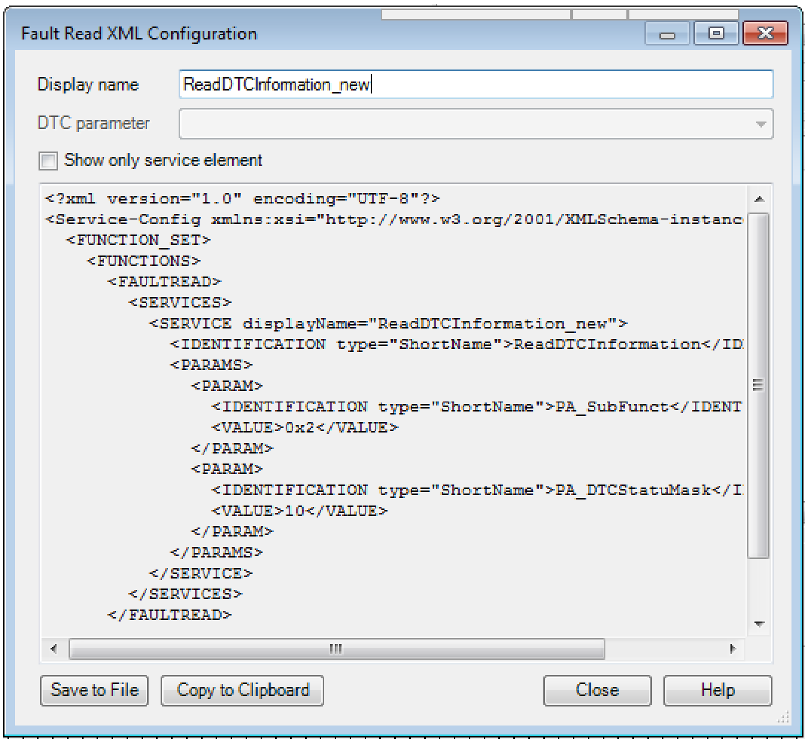
□ Edit Request PDU (Hex)

19 02 0A

Name	Value	Unit	Type	Value Range	Validity

2. Create the XML configuration file that contains the newly configured service XML configuration.

   From the context menu of the parameter list, select the **Create XML Configuration – Fault Read** command. ControlDesk opens the **Fault Read XML Configuration** dialog, displaying the current configuration settings.

   Display name: ReadDTCInformation_new

   DTC parameter:

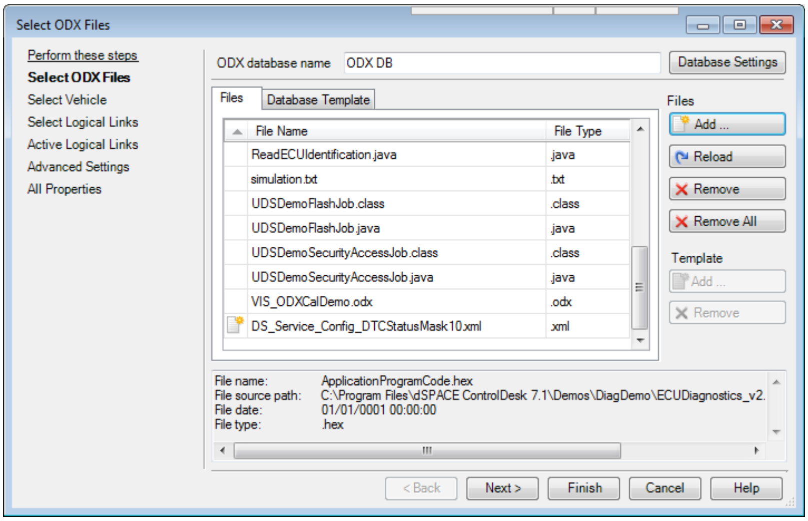
   □ Show only service element

   ```
   <?xml version="1.0" encoding="UTF-8"?>
   <Service-Config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instanc
     <FUNCTION_SET>
       <FUNCTIONS>
         <FAULTREAD>
           <SERVICES>
             <SERVICE displayName="ReadDTCInformation_new">
               <IDENTIFICATION type="ShortName">ReadDTCInformation</ID
               <PARAMS>
                 <PARAM>
                   <IDENTIFICATION type="ShortName">PA_SubFunct</IDENT
                   <VALUE>0x2</VALUE>
                 </PARAM>
                 <PARAM>
                   <IDENTIFICATION type="ShortName">PA_DTCStatuMask</I
                   <VALUE>10</VALUE>
                 </PARAM>
               </PARAMS>
             </SERVICE>
           </SERVICES>
         </FAULTREAD>
   ```

   You can specify the name to display in the Fault Memory Instrument for the configured service. Then you can create the ready-to-use XML configuration file containing the service XML configuration by clicking **Save to File** and specifying the file name. The file name must begin with `DS_Service_Config`.

3. Add the generated XML configuration file to the ECU Diagnostics device configuration.

   Select the ECU Diagnostics device in the **Project** controlbar and select **Configure Platform/Device** from the context menu of the device. In the **Select ODX Files** dialog, press the **Add** button. A standard **Open** dialog opens for you to add your XML configuration file to the diagnostics database for the ECU Diagnostics device.



Select ODX Files

Perform these steps
Select ODX Files
Select Vehicle
Select Logical Links
Active Logical Links
Advanced Settings
All Properties

ODX database name: ODX DB

File Name	File Type
ReadECUIdentification.java	.java
simulation.txt	.txt
UDSDemoFlashJob.class	.class
UDSDemoFlashJob.java	.java
UDSDemoSecurityAccessJob.class	.class
UDSDemoSecurityAccessJob.java	.java
VIS_ODXCalDemo.odx	.odx
DS_Service_Config_DTCStatusMask10.xml	.xml

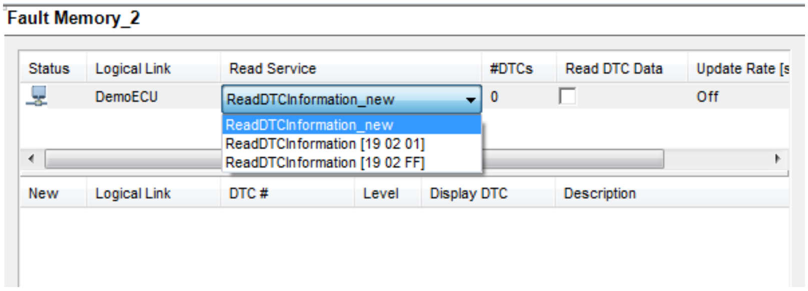
File name: ApplicationProgramCode.hex
File source path: C:\Program Files\dSPACE ControlDesk 7.1\Demos\DiagDemo\ECUDiagnostics_v2.
File date: 01/01/0001 00:00:00
File type: .hex

In the **Select ODX Files** dialog, click **Finish** to finish the device configuration and close the dialog.

4. Reload the layout with the Fault Memory Instrument in ControlDesk, if required.

   If, after adding the XML configuration file to the device configuration, the newly configured service is not immediately available in the Fault Memory Instrument, you must reopen the layout containing the Fault Memory Instrument.

   The following illustration shows the Fault Memory Instrument that provides the new service and the default read services:



Fault Memory_2

Status	Logical Link	Read Service	#DTCs	Read DTC Data	Update Rate [s
	DemoECU	ReadDTCInformation_new	0	□	Off

ReadDTCInformation_new
ReadDTCInformation [19 02 01]
ReadDTCInformation [19 02 FF]

New	Logical Link	DTC #	Level	Display DTC	Description

Result

The newly configured service for reading fault memory is available in the Fault Memory Instrument.

Related topics

Basics

Basics of the XML Configuration File..93

HowTos

How to Configure and Execute Diagnostic Communication Objects..70

References

Create XML Configuration (ControlDesk Instrument Handling 🕮)

Reference Information

Where to go from here

Information in this section

ECU Diagnostics Device

Introduction

ECU Diagnostics device A device that provides access to ECUs connected to the ControlDesk PC via CAN or K-Line for diagnostics or flash programming purposes.

ControlDesk provides the *ECU Diagnostics v2.0.2* device, which supports the ASAM MCD-3 D V2.0.2 standard.

ControlDesk supports the following ODX database standards:

- ASAM MCD-2 D V2.0.1
- ASAM MCD-2 D V2.2.0 (ISO 22901-1)

Managing tasks with the ECU Diagnostics device

For information on managing tasks with the ECU Diagnostics device, refer to Introduction to ECU Diagnostics on page 10.

Configuring the device

For instructions on configuring the device, refer to ECU Diagnostics Device Configuration on page 32.

Device properties

The device provides the following properties and settings:

Purpose	Refer to
To specify common properties of the platform/device.	Common Properties (ControlDesk Platform Management 🕮)
To specify diagnostics settings for the selected device.	Diagnostics Settings Properties (ControlDesk Platform Management 🕮)
To specify general settings of the selected platform/device.	General Settings Properties (ControlDesk Platform Management 🕮)
To select logical links for ECU diagnostics, to display the active logical links of the ECU Diagnostics device with their configurations, and to specify connection check settings and configure communication parameters individually for each active logical link.	Logical Link Selection Properties (ControlDesk Platform Management 🕮)
To display and configure the ODX database specified for the selected ECU Diagnostics device.	Active ODX Database / ODX Database Properties (ControlDesk Platform Management 🕮)
To display the vehicle that is selected to be used with the selected ECU Diagnostics device and to select another vehicle.	Vehicle Selection Properties (ControlDesk Platform Management 🕮)

Related commands

The ECU Diagnostics device provides the following commands:

Purpose	Refer to
To add a new diagnostics database to the selected ECU Diagnostics device.	Add ODX Database (ControlDesk Platform Management 🕮)
To insert a platform/device into the active experiment.	Insert Platform / Add Platform/Device (ControlDesk Platform Management 🕮)
To configure the platform/device selected in ControlDesk and assign it to the available hardware or VEOS.	Configure Platform/Device (ControlDesk Platform Management 🕮)
To establish a logical connection between ControlDesk and the selected platform/device hardware.	Connect Platform/Device (ControlDesk Platform Management 🕮)
To deactivate the platform/device in the experiment.	Disable Platform/Device (ControlDesk Platform Management 🕮)
To stop communication between ControlDesk and the hardware or VEOS that belongs to the selected platform/device.	Disconnect Platform/Device (ControlDesk Platform Management 🕮)
To activate the platform/device in the experiment.	Enable Platform/Device (ControlDesk Platform Management 🕮)
To execute an ECU flash programming operation based on an ODX database and diagnostic protocol.	Flash ECU (ECU Diagnostics) (ControlDesk Platform Management 🕮)
To view and edit the properties of the selected element.	Properties (Controlbar) (ControlDesk User Interface Handling 🕮)
To remove an item from the currently loaded project.	Remove (from Project) (ControlDesk Project and Experiment Management 🕮)
To change the name of the selected platform/device in the experiment.	Rename Platform/Device (ControlDesk Platform Management 🕮)
To start online calibration and measurement for the currently selected platform/device.	Start Calibration & Measurement (for Single Platform/Device) (ControlDesk Measurement and Recording 🕮)

Purpose	Refer to
To start measurement for the selected platform/device.	Start Measurement (for Single Platform/Device) (ControlDesk Measurement and Recording 📖)
To start online calibration for the selected platform/device.	Start Online Calibration (for Single Platform/Device) (ControlDesk Calibration and Data Set Management 📖)
To stop measurement and online calibration for the currently selected platform/device.	Stop Calibration & Measurement (for Single Platform/Device) (ControlDesk Measurement and Recording 📖)
To stop measurement for the currently selected platform/device.	Stop Measurement (for Single Platform/Device) (ControlDesk Measurement and Recording 📖)
To stop online calibration for the selected platform/device.	Stop Online Calibration (for Single Platform/Device) (ControlDesk Calibration and Data Set Management 📖)

Related topics

Basics

Basics of the ECU Diagnostics Device.. 32
Requirements for Measurement and Calibration via the ECU Diagnostics Device........................ 35

HowTos

How to Configure an ECU Diagnostics Device.. 36

Diagnostics Management Page

Access

This page is part of the ControlDesk Options dialog.

The dialog can be opened via the **Options Command** (ControlDesk User Interface Handling 📖).

Purpose

To specify general settings used in diagnostic instruments and dialogs.

Dialog settings

Select display name type Lets you specify whether the short name or the long name of an item (for example, a logical link or a diagnostic service) is displayed in instruments and in the Properties controlbar.

Display Option	Description
Use short name	Short identifier. Unique for items within one object collection.

Display Option	Description
Use long name	Long, more detailed identifier. Not necessarily unique within one object collection.

Show complex data structure Lets you select whether to display all superordinate complex data that a parameter of a diagnostic communication object is part of together with the parameter name in the Diagnostics Instrument. For example, if a parameter is part of a structure, field or multiplexer (= complex data type), the name of this complex data is displayed with the parameter name in the Diagnostics Instrument's parameter list and output field (if available). If the parameter is part of a hierarchy of multiple complex data elements, the complete hierarchy structure is displayed.

Select default display mode Lets you select whether to display values of integer parameters in the Diagnostics Instrument in hexadecimal (HEX), decimal (DEC), or binary (BIN) notation. DTC values in the Fault Memory Instrument are strings and therefore not affected by this setting.

Custom ODX database template source path Lets you specify a path for custom ODX database templates. You can enter a path or click the Browse button to select one.

A custom ODX database template is a valid ODX database whose files are located in one folder in the specified custom ODX database template source path. The custom ODX database template is named after the folder that contains the files.

Advanced Settings dialog

Opens an Advanced Settings dialog for you to specify default configuration settings for the ECU Diagnostics device and global diagnostics server settings:

Purpose	Property Page
To specify default configuration settings for the ECU Diagnostics device.	Default Configurations page
To specify configuration options for database optimization and the diagnostics server.	Diagnostics Server Settings page

Default Configurations page

To specify default configuration settings for ECU Diagnostics devices. Changes take effect in future ControlDesk experiments.

Automatic ODX version detection Lets you select the ODX version used with your ECU Diagnostics device automatically. If the checkbox is selected, ControlDesk finds the ODX version by using the files imported for the ODX diagnostics database for your ECU Diagnostics device. If the checkbox is cleared, you must select the ODX version manually.

Note

Disable automatic ODX version detection only if you are very familiar with ODX databases and the used database files.

This setting is the default setting for the Automatic detection option for newly created ECU Diagnostics devices.

ODX version for manual selection Lets you specify the default setting for the ODX version option for newly created ECU Diagnostics devices or newly generated ODX databases when automatic ODX version detection is disabled. You can select one of the ODX versions supported by ControlDesk from the list, or select 'Undefined' to specify that no ODX version should be preset. In that case you must set the ODX version manually later when configuring your ODX database.

Optimize database for ODX version 2.0.1 Lets you specify to convert the diagnostics database (ODX version 2.0.1) to binary format. The binary format allows faster experiment loading and lets you work with large ODX databases since memory usage is reduced. However, transforming the ODX data into the binary format takes some time.

Note

For ODX 2.2.0 diagnostics databases, database optimization is always performed. You cannot change this setting. So if you work with an ODX 2.2.0 database, ControlDesk always uses the binary format instead of ODX data for the diagnostics database.

This setting is the default setting for the Optimize database option for newly created ECU Diagnostics devices.

Copy ECU base variant configuration Lets you specify to apply the configuration of an ECU base variant to an identified ECU variant when the ECU variant is selected for use with the ECU Diagnostics device.

This setting is the default setting for the Copy configuration from ECU base variant to identified ECU variant after selecting the ECU variant option for newly created ECU Diagnostics devices.

Create variable description for selected logical links Lets you specify the default setting for the Create variable description for selected logical links option for newly created ECU Diagnostics devices. You can select to generate a variable description from the ODX database for the selected logical links. A variable description is necessary for measuring and calibrating diagnostics variables via the ECU Diagnostics device.

To enable ControlDesk to create a variable description from the ODX database, the ODX database must meet some requirements. For details, refer to Conventions in Connection with ODX Databases on page 22.

Use additional protocol information Lets you specify to also use information from the protocol specification for variable generation. If the checkbox is selected, ControlDesk generates variables not only according to the default ODX semantics and service configurations in the XML configuration file(s), but also on the basis of the protocol-specific identification of services. However, selecting this checkbox can significantly increase the number of variables to be generated. As a result, the generation of the variable description and also later uploads may take some time. If the checkbox is cleared, no variables are generated on the basis of the protocol information.

> **Note**
>
> Fault read services and DTC variables always use the information from the protocol specification, regardless of whether the Use additional protocol information option is enabled or not.

This setting is the default setting for newly created ECU Diagnostics devices.

Generate Diagnostic Trouble Code variables Lets you specify to generate diagnostic trouble code (DTC) measurement variables. DTC variables can be used to measure changes on DTCs, and for triggering measurements. You can find the DTC variables in the Variables controlbar in the `<Diagnostic Trouble Codes>` nodes located below the logical link nodes.

This setting is the default setting for the Generate DTC variables option for newly created ECU Diagnostics devices.

Add additional measurements for parameters Lets you specify to generate a measurement variable for each generated parameter. The generated measurement variables are displayed in the Variables controlbar in the read service nodes located below the logical link nodes. This allows quick and easy access to the measurement variables of a read service. A parameter and the associated generated measurement variable are not synchronized. If the checkbox is cleared, no additional measurement variables are generated for parameters.

This setting is the default setting for the Add additional measurements for parameters option for newly created ECU Diagnostics devices.

Use ECU base variant name instead of ECU variant name Lets you specify to use the name of an ECU base variant as the name for all the variants of the ECU. Using the same name prevents the connection to the variables being lost when a variable description is reloaded after you switch to another ECU variant.

This setting is the default setting for the Use ECU base variant name instead of ECU variant name option for newly created ECU Diagnostics devices.

Flash job contains start communication Lets you specify the default setting for the Flash job contains start communication option for new flash programming sessions based on an ODX database and diagnostic protocol. The setting indicates whether the diagnostic job used for flashing the ECU memory contains the `StartCommunication` control primitive.

- With the checkbox selected, ControlDesk starts the flash job (which is expected to execute the `StartCommunication` control primitive) without having to execute any control primitives beforehand.
- With the checkbox cleared, ControlDesk executes the `StartCommunication` control primitive before it starts the flash job.

Note

For K-Line-based diagnostic protocols, the `StartCommunication` control primitive must be executed first to initialize the bus. Otherwise, communication with an ECU via K-Line is impossible.

Diagnostics Server Settings page

To specify global configuration settings for the diagnostics server.

Select the heap size for the diagnostics server Lets you specify how much heap memory in byte is reserved for executing the diagnostic server within the ControlDesk process. If Execute diagnostic server in the process of ControlDesk is selected, the diagnostic server is started with the specified heap size at maximum.

Note

After changing the memory size, you must restart ControlDesk to let the change take effect.

Related topics

References

Options Command (ControlDesk User Interface Handling 📖)

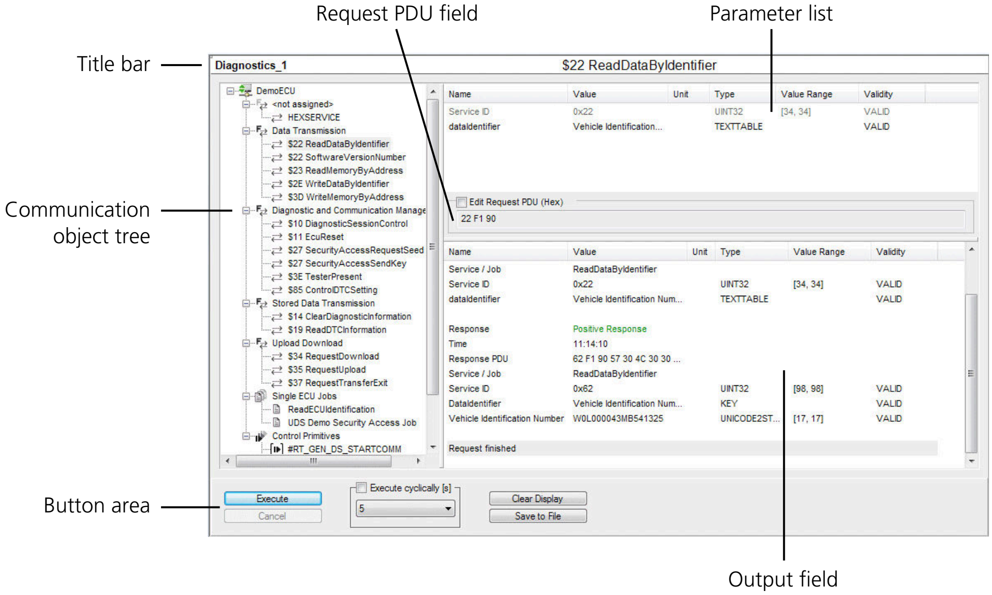
Instruments for Diagnostic Tasks

Introduction

ControlDesk provides several instruments that let you perform diagnostic tasks.

Diagnostics Instrument

To communicate with the ECU via the diagnostic protocol using diagnostic services, diagnostic jobs, and control primitives.



Diagnostics_1 $22 ReadDataByIdentifier

- DemoECU
  - <not assigned>
    - HEXSERVICE
  - Data Transmission
    - $22 ReadDataByIdentifier
    - $22 SoftwareVersionNumber
    - $23 ReadMemoryByAddress
    - $2E WriteDataByIdentifier
    - $3D WriteMemoryByAddress
  - Diagnostic and Communication Manage
    - $10 DiagnosticSessionControl
    - $11 EcuReset
    - $27 SecurityAccessRequestSeed
    - $27 SecurityAccessSendKey
    - $3E TesterPresent
    - $85 ControlDTCSetting
  - Stored Data Transmission
    - $14 ClearDiagnosticInformation
    - $19 ReadDTCInformation
  - Upload Download
    - $34 RequestDownload
    - $35 RequestUpload
    - $37 RequestTransferExit
  - Single ECU Jobs
    - ReadECUIdentification
    - UDS Demo Security Access Job
  - Control Primitives
    - #RT_GEN_DS_STARTCOMM

Name	Value	Unit	Type	Value Range	Validity
Service ID	0x22		UINT32	[34, 34]	VALID
dataIdentifier	Vehicle Identification...		TEXTTABLE		VALID

□ Edit Request PDU (Hex)

22 F1 90

Name	Value	Unit	Type	Value Range	Validity
Service / Job	ReadDataByIdentifier				
Service ID	0x22		UINT32	[34, 34]	VALID
dataIdentifier	Vehicle Identification Num...		TEXTTABLE		VALID
Response	Positive Response				
Time	11:14:10				
Response PDU	62 F1 90 57 30 4C 30 30 ...				
Service / Job	ReadDataByIdentifier				
Service ID	0x62		UINT32	[98, 98]	VALID
DataIdentifier	Vehicle Identification Num...		KEY		VALID
Vehicle Identification Number	W0L000043MB541325		UNICODE2ST...	[17, 17]	VALID

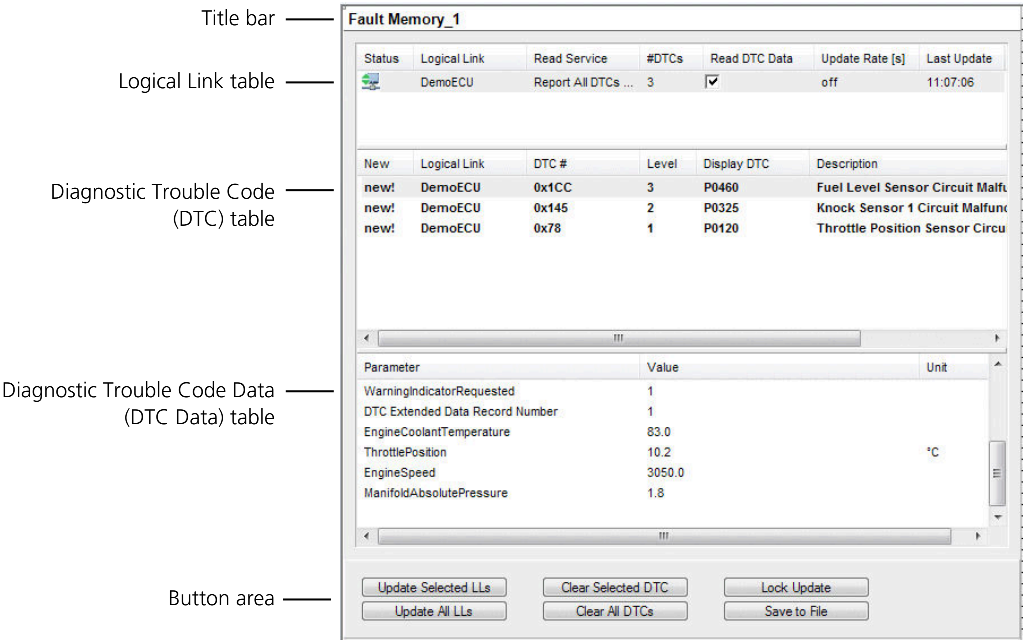
Request finished

Execute | Cancel | □ Execute cyclically [s] 5 | Clear Display | Save to File

For further reference information, refer to Diagnostics Instrument (ControlDesk Instrument Handling 🕮).

Fault Memory Instrument

To read, clear, and save the content of the ECU's fault memory (diagnostic trouble codes and corresponding environment data).


Title bar
Logical Link table
Diagnostic Trouble Code (DTC) table
Diagnostic Trouble Code Data (DTC Data) table
Button area

Fault Memory_1

Status	Logical Link	Read Service	#DTCs	Read DTC Data	Update Rate [s]	Last Update
	DemoECU	Report All DTCs ...	3	☑	off	11:07:06

New	Logical Link	DTC #	Level	Display DTC	Description
new!	**DemoECU**	**0x1CC**	**3**	**P0460**	**Fuel Level Sensor Circuit Malfu**
new!	**DemoECU**	**0x145**	**2**	**P0325**	**Knock Sensor 1 Circuit Malfunc**
new!	**DemoECU**	**0x78**	**1**	**P0120**	**Throttle Position Sensor Circu**

Parameter	Value	Unit
WarningIndicatorRequested	1	
DTC Extended Data Record Number	1	
EngineCoolantTemperature	83.0	
ThrottlePosition	10.2	°C
EngineSpeed	3050.0	
ManifoldAbsolutePressure	1.8	

Update Selected LLs | Update All LLs | Clear Selected DTC | Clear All DTCs | Lock Update | Save to File

For further reference information, refer to Fault Memory Instrument (ControlDesk Instrument Handling 📖).

Related topics

Basics

HowTos

Automation

Where to go from here

Information in this section

Programming ControlDesk Automation

Where to go from here

Information in this section

Information in other sections

Tool Automation Demos (ControlDesk Automation 📖)
Demonstrate how to automate ControlDesk and use ControlDesk events.

Automating ECU Diagnostics Tasks

Introduction

In ControlDesk, you add an ECU Diagnostics device to an experiment to access an ECU via a diagnostic interface. You can automate adding and configuring an ECU Diagnostics device. You can also automate communicating with an ECU via diagnostic services, diagnostic jobs, and control primitives.

The program listing below consists of excerpts from the `ECUDiagnosticsHandling.py` demo script.

Adding an ECU Diagnostics device

The following listing shows you how to add an ECU Diagnostics (MCD-3D v2.0.2) device.

```
class MainDemoController(object):
    (...)
    def __init__(self):
        # The platform used in this demo.
        self.ECUDiagnostics = None
    def Initialize(self):
        (...)
        # Define Enums object.
        self.enums = Enums(self.ControlDeskApplication)
        (...)
    def CreatePlatform(self):
        # Add ECU Diagnostics platform.
        self.ECUDiagnostics = self.ControlDeskApplication.ActiveExperiment.Platforms.Add(
                                      self.enums.PlatformType.Diagnostic2)
```

Importing ODX database files

The following listing shows you how to import ODX database files.

```
# Define ODXCalDemo ODX database folder.
ODXDBPATH = os.path.abspath(os.path.join(SCRIPTPATH, "..\..\..\DiagDemo\ECUDiagnostics_v2.0.2"))
(...)
class MainDemoController(object):
    (...)
     def ImportODXDatabaseFiles(self, databasePath, databaseName = "", optimizeDatabaseFlag=False):
        # Import ODX Database Files.
        self.ECUDiagnostics.ActiveDiagnosticsDatabase.DatabaseFiles.AddFilesFromDirectory(databasePath)
        # Activate ODX Database files.
        self.ECUDiagnostics.ActiveDiagnosticsDatabase.Update()
```

Adding and activating another ODX database

The following listing shows you how to add another ODX database to the device and then activate it.

```
# Define a custom name for the second ODX Database.
OTHERODXDBNAME = "Another ODX Database"
(...)
class MainDemoController(object):
    (...)
   def AddAndActivateDatabase(self, databaseName):
      # Add a new database.
      newDatabase = self.ECUDiagnostics.DiagnosticsDatabases.Add(databaseName)
      # Activate the new database.
      newDatabase.Activate()
```

Note

After you add another ODX database, you can import ODX database files to it. Refer to Importing ODX database files.

Selecting a vehicle and a logical link

The following listing shows you how to select a vehicle and a logical link.

```
class MainDemoController(object):
    (...)
    def __init__(self):
        (...)
        # Define enums object.
        self.enums = Enums(self.ControlDeskApplication)
        (...)
    def SelectVehicle(self)
        # Get the vehicle from the vehicles collection by name (by index is also possible).
        vehicle = self.ECUDiagnostics.ActiveDiagnosticsDatabase.VehicleSelection.Vehicles.Item(
                  VEHICLENAME)
        # Select the vehicle
        selectedVehicle = vehicle.Select()
        (...)
    def ConfigureLogicalLink(self):
        # Get Logical Link selection.
        logicalLinkSelection = self.ECUDiagnostics.ActiveDiagnosticsDatabase.VehicleSelection.\
                               selectedVehicle.logicalLinkSelection
        # Get the logical link by name (by index is also possible).
        logicalLink = logicalLinkSelection.LogicalLinks.Item(LOGICALLINKNAME)
        # Select the logical link.
        logicalLink.Select()
        # Set the protocol(UDS) which is used by the logical link.
        logicalLink.Protocol = self.Enums.ECUDiagnosticsProtocol.ISO_14229_UDS
        # Set the physical connection (CAN) which is used by the logical link.
        logicalLink.PhysicalConnection = self.enums.ECUDiagnosticsPhysicalConnection.CAN
        # Get the CANInterfaceSelection of the logical link.
        canInterfaceSelection = logicalLink.InterfaceSelection
        # Get dSPACE vendor.
        dspaceVendor = canInterfaceSelection.Vendors.Item(DSPACEVENDORNAME)
        # Get Virtual CAN Interface.
        virtualInterface = DspaceVendor.AvailableInterfaces.Item(VIRTUALCANINTERFACENAME)
        # Select the first channel of the virtual interface.
        virtualInterface.Channels.Item(0).Select()
        # Deactivate creating of variable description.
        logicalLinkSelection.CreateVariableDescriptionForSelectedLogicalLinks = False
        # Activate the settings (protocol, physical connection, interface selection) of the
        # selected Logical Link.
        logicalLinkSelection.Update()
        self.ActiveLogicalLink = logicalLinkSelection.ActiveLogicalLinks.Item(LOGICALLINKNAME)
```

Starting online calibration with the CalDemo ECU

The following listing shows how to connect to the CalDemo ECU via the ECU Diagnostics (MCD-3D v2.0.2) device and then start online calibration.

```
class MainDemoController(object):
    (...)
    def StartOnlineCalibrationCalDemo(self):
        # Start CalDemo ECU.
        os.startfile(CALDEMOPATH)
        # Wait for the CalDemo to completly start
        time.sleep(3)
        # Connect device.
        self.ECUDiagnostics.Connect()
        # Start online calibration
        self.ControlDeskApplication.CalibrationManagement.StartOnlineCalibration()
```

Executing a service (specified by its name) using parameter values

The following listing shows how to execute a service (specified by its name) using parameter values.

```
class MainDemoController(object):
    (...)
    def ExecuteService(self, serviceName, requestParameterPaths = [], requestParameterValues = []):
        (...)
        # Get the services collection
        serviceCollection = self.ActiveLogicalLink.Services
        # Get the service to be executed
        service = serviceCollection.Item(serviceName)
        # Set the request parameter values.
        self.SetRequestParameters(service.RequestParameters, requestParameterPaths, requestParameterValues)
        # Execute the service
        responses = service.Execute()
```

Executing a service (specified by its name) using a PDU value

The following listing shows how to execute a service (specified by its name) using a PDU value.

```
class MainDemoController(object):
    (...)
    def ExecuteServiceWithPDU(self, serviceName, pdu):
        (...)
        # Get the services collection
        serviceCollection = self.ActiveLogicalLink.Services
        # Get the service to be executed
        service = serviceCollection.Item(serviceName)
        # Execute the service
        responses = service.ExecuteUsingCustomPDU(pdu)
```

The following listing shows how to use the method defined above in a main function. The PDU value is entered directly in the method.

```
    def ExecuteDemo():
        (...)
        # Execute Diagnostic Service by entering the request PDU directly
        demoController.ExecuteServiceWithPDU("ReadDataByIdentifier", (0x22, 0xf1, 0x90))
```

Executing the hex service using a PDU

The following listing shows how to execute the hex service using a PDU value.

```
class MainDemoController(object):
    (...)
    def ExecuteHexService(self, pdu):
        (...)
        # Execute the service
        responses = self.ActiveLogicalLink.ExecuteHexService(pdu)
```

Executing a job specified by its name

The following listing shows how to execute a job specified by its name.

```
class MainDemoController(object):
    (...)
    def ExecuteJob(self, jobName):
        # Get the jobs collection
        jobsCollection = self.ActiveLogicalLink.SingleECUJobs
        # Get the job to be executed.
        job = JobsCollection.Item(jobName)
        # Execute the job
        responses = job.Execute()
```

Executing an ECU flash programming session

The following listings show two examples of how to execute a flash session by using the `ExecuteAsync` method of the `ECUDiagnostics2FlashSession / IPmECUDiagnostics2FlashSession <<Interface>>` interface. In both examples, the flash session is executed *asynchronously*, i.e., ControlDesk is not blocked during the execution of the session.

Executing a flash session without events The following listing shows how to execute a flash session without using events.

```
class MainDemoController(object):
    (...)
    def ExecuteFlashSession(self, flashSessionName):
        # Show information in a dialog.
        USERDIALOG.Show("Execute Flash Session", DiagResources.ExecuteFlashSession % flashSessionName)
        # Get the flash sessions collection.
        flashSessionsCollection = self.ActiveLogicalLink.FlashSessions
        # Get the flash session that will be executed.
        flashSession = flashSessionsCollection.Item(flashSessionName)
        # Execute the flash session.
        flashSession.ExecuteAsync()
        (...)
        # Wait for the flash session to finish.
        while flashSession.State == self.Enums.ECUDiagnosticsPrimitiveState.Executing:
            time.sleep(1)
        # Display the response.
        self.ShowFlashSessionResponseCollection(flashSession.responses)
```

Executing a flash session by using events The following listing shows how to execute a flash session by using `DiagnosticsManagementEvents / IXaDiagnosticsManagementEvents <<EventInterface>>` events.

Using these events lets you perform the following actions:

- Get progress information during a running flash session.
- Get responses during a running flash session.
- Get informed on the termination of a flash session.

Note

ControlDesk must be able to process messages while it is waiting for events. To make this possible, use the `PumpWaitingMessages` method in a loop as long as ControlDesk waits for events. The method pumps all the waiting messages for the current thread. As an alternative, you can also use the `Wait` method defined in the `DemoUtilities.py` script.
Do not use the `Sleep` method, because this blocks the processing of messages when ControlDesk is waiting for events.

```
class MainDemoController(object):
    (...)
    def ExecuteFlashSessionWithEvents(self, flashSessionName):
        # Show information in a dialog.
        USERDIALOG.Show("Execute Flash Session", DiagResources.ExecuteFlashSessionWithEvents % flashSessionName)
        # Get the flash sessions collection.
        flashSessionsCollection = self.ActiveLogicalLink.FlashSessions
        # Get the flash session that will be executed.
        flashSession = flashSessionsCollection.Item(flashSessionName)
        # Connect the DiagnosticsManagement event sink.
        diagnosticsManagement = DispatchWithEvents(self.ControlDeskApplication.DiagnosticsManagement,\
        DiagnosticsManagementEvents)
        # Initialize the DiagnosticsManagement events
        diagnosticsManagement.InitializeDiagnosticsManagementEvents(FLASH_SESSION_STATUS_DIALOG, self.enums)
        # Execute the flash session.
        flashSession.ExecuteAsync()
        # Show the Status dialog. This is a blocking method, but it pumps all the waiting messages
        # for the current thread.
        dlgResult = FLASH_SESSION_STATUS_DIALOG.ShowDialog("Flash Session Status", "Results", 250, 300)
        # Cancel the flash session if the user clicked the 'Cancel' button.
        if not dlgResult:
            flashSession.Cancel()
```

The following listing shows the event sink definition for ECU Diagnostics flash session events.

```
class DiagnosticsManagementEvents(object):
    """Defines the event sink for the Diagnostics flash session events.
    Syntax       : OBJ = DiagnosticsManagementEvents()
    parameters : -
    Description: Defines the event sink for the Diagnostics flash session events.
    """
    #----------------------------------------------------------------------------------------------
    # Method : InitDiagnosticsManagementEvents
    #      Call this method to initialize the DiagnosticsManagementEvents object.
    #----------------------------------------------------------------------------------------------
    def InitializeDiagnosticsManagementEvents(self, statusDialog, enums):
        """Call this method to initialize the DiagnosticsManagementEvents object.
        Syntax      : Obj.InitDiagnosticsManagementEvents(LogDialog)
        parameters  : statusDialog       - object  - The dialog to show the event information.
                      enums              - object  - The ControlDesk Enums object.
        Description : Call this method to initialize the DiagnosticsManagementEvents object.
        Return Value: -
        """
        self.statusDialog = statusDialog
        self.enums = enums
    #----------------------------------------------------------------------------------------------
    # Method : OnFlashSessionTerminated
    #      This method is called if a flash session is terminated.
    #----------------------------------------------------------------------------------------------
    def OnFlashSessionTerminated(self, platform, flashSession):
        """This method is called, if a flash session is terminated.
        Syntax      : Obj.OnFlashSessionTerminated(platform, flashSession)
        parameters  : platform       - object  - The platform object the flash session belongs to.
                      flashSession   - object  - The flash session object which is terminated.
        Description : This method is called if a flash session is terminated.
        Return Value: -
        """
        flashSession = Dispatch(flashSession)
        if self.statusDialog is not None:
            Message = DiagResources.FlashSessionTerminatedString % (flashSession.ShortName, \
            self.enums.ECUDiagnosticsRequestStatus(flashSession.Responses.RequestState))
            # Display the message in the Status dialog.
            self.statusDialog.AddMessage(Message)
            # Signal the dialog that the flash session is finished and the demo can continue.
            self.statusDialog.SetTaskFinished()
    #----------------------------------------------------------------------------------------------
    # Method : OnFlashSessionResponseReceived
    #      This method is called if a flash session received a response.
    #----------------------------------------------------------------------------------------------
    def OnFlashSessionResponseReceived(self, platform, flashSession, response):
        """This method is called if a flash session received a response.
        Syntax      : Obj.OnFlashSessionResponseReceived(platform, flashSession, response)
        parameters  : platform       - object  - The platform object the flash session belongs to.
                      flashSession   - object  - The flash session object which received a response.
                      response       - object  - The response object with the response information.
        Description : This method is called if a flash session received a response.
        Return Value: -
        """
        response = Dispatch(response)
        if self.statusDialog is not None:
            for outputParameter in response.OutputParameters:
                output = "'%s'" %  outputParameter.Value
                # Display the output in the Status dialog.
                self.statusDialog.AddMessage(output)
    #---------------------------------------------------------------------------------------------
    # Method : OnFlashSessionProgressInfoReceived
    #      This method is called if a flash session received progress information.
    #---------------------------------------------------------------------------------------------
    def OnFlashSessionProgressInfoReceived(self, platform, flashSession, progress):
        """This method is called if a flash session received progress information.
        Syntax      : Obj.OnFlashSessionProgressInfoReceived(platform, flashSession, progress)
        parameters  : platform       - object  - The platform object the flash session belongs to.
                      flashSession   - object  - The flash session object which received progress information.
                      progress       - integer - The percent by progress information.
        Description : This method is called if a flash session received progress information.
        Return Value: -
        """
        flashSession = Dispatch(flashSession)
        if self.statusDialog is not None:
            # Show the progress in the Status dialog.
            self.statusDialog.SetProgress(progress)
```

Reading DTCs from the ECU fault memory

The following listing shows how to read DTCs from the ECU fault memory.

```
class MainDemoController(object):
    (...)
    def ReadDTCs(self):
       # Get the configured faultread services collection.
       configuredFaultreadServicesCollection = self.ActiveLogicalLink.ConfiguredFaultreadServices
       # Get the first item in the collection.
       configuredFaultreadService = configuredFaultreadServicesCollection.Item(0)
       # Execute read DTCs
       responses = ConfiguredFaultreadService.ReadDTCs()
```

Reading environment data for a specific DTC

The following listing shows how to read environment data for a specific DTC.

```
class MainDemoController(object):
   (...)
   def ReadEnvironmentData(self, DTCNumber):
      # Get the configured faultread services collection.
      configuredFaultreadServicesCollection = self.ActiveLogicalLink.ConfiguredFaultreadServices
      # Get the first item in the collection.
      configuredFaultreadService = configuredFaultreadServicesCollection.Item(0)
      # Read environment data for a DTC
      responses = ConfiguredFaultreadService.ReadEnvironmentData(dtcNumber)
```

Related topics

Basics

Basics of ECU Diagnostics with ControlDesk... 10
Tool Automation Demos (ControlDesk Automation 🕮)

ECU Diagnostics Handling

ECU Diagnostics-Related Interfaces

Introduction

In ControlDesk, you add an ECU Diagnostics device to an experiment to access an ECU via a diagnostic interface. You can automate adding and configuring an ECU Diagnostics (MCD-3D v2.0.x) device.

If you use the ECU Diagnostics (MCD-3D v2.0.2) device, you can also automate ECU communication via diagnostic services, diagnostic jobs, and control primitives.

Description

In ControlDesk, you add an ECU Diagnostics device to an experiment to access an ECU via a diagnostic interface. The resulting platform object implements the *IPmECUDiagnosticsPlatform* or *IPmECUDiagnostics2Platform* interface.

Related interfaces

Interface	Description
IPmECUDiagnosticsPlatform (refer to ECUDiagnosticsPlatform / IPmECUDiagnosticsPlatform <<Interface>> (ControlDesk Automation 📖))	This interface is to access a ECU diagnostic platform.
IPmECUDiagnostics2Platform (refer to ECUDiagnostics2Platform / IPmECUDiagnostics2Platform <<Interface>> (ControlDesk Automation 📖))	This interface is to access a ECU diagnostic platform version 2.0.2.

Related documentation

Topic	Description
Automating ECU Diagnostics Tasks on page 136	In ControlDesk, you add an ECU Diagnostics device to an experiment to access an ECU via a diagnostic interface. You can automate adding and configuring an ECU Diagnostics device. You can also automate communicating with an ECU via diagnostic services, diagnostic jobs, and control primitives.

Troubleshooting

Diagnostic Service Execution Hangs Sporadically

Problem

The execution of a diagnostic service can hang sporadically when you use *UDS on Diagnostics over Internet Protocol* (UDS on DoIP) to communicate with an ECU for ECU diagnostics.

Description

ControlDesk can be specified to cyclically perform a connection state check according to the logical-link-specific TesterPresent Configuration. This might cause connection problems when you use *UDS on Diagnostics over Internet Protocol* (UDS on DoIP) for that logical link. As a result, the execution of a diagnostic service can hang sporadically.

Solution

Disable the Cyclic connection state check for the logical link that communicates via UDS on DoIP.

Refer to Active Logical Links Dialog (ControlDesk Platform Management).

Related topics

Basics

Active Logical Links Dialog (ControlDesk Platform Management)

Limitations

Limitations for ECU Diagnostics

Introduction

There are some limitations for using the ControlDesk ECU Diagnostics Module.

General limitations for the ECU Diagnostics device

Only one ECU Diagnostics device in an experiment There can be only one active ECU Diagnostics device in an experiment.

Communication with only one ECU of the functional group layer You can communicate with only one ECU of the *functional group* layer at the same time.

Resume online calibration behavior is applied to all logical links The ECU Diagnostics device applies the selected Resume online calibration behavior such as "Upload" to *all* the logical links specified in it even if only one logical link has been reconnected.

ECU diagnostics via CAN

Limited number of CAN controllers When performing ECU diagnostics via CAN, you can use only six CAN controllers (physical and/or virtual) simultaneously at the most.

Limited number of logical links When performing ECU diagnostics via CAN, the number of logical links accessible via one CAN controller is limited:

- 30 logical links if you work with UDS or KWP2000 on CAN
- 5 logical links if you work with TP2.0
- 1 logical link if you work with TP1.6

Transmitting CAN FD messages without BRS flag if CANFDBaudrate ≠ 0 not possible With the ControlDesk ECU Diagnostics device, you cannot transmit CAN FD messages without BRS flag if the `CANFDBaudrate` is ≠ 0. This means that message transmission at the arbitration rate is not possible in this case.

For more information, refer to Conventions in Connection with ODX Databases on page 22.

ECU diagnostics via K-Line

Only one (physical) K-Line interface When performing ECU diagnostics via K-Line, you can use only one (physical) K-Line interface.

Communication with only one ECU When performing ECU diagnostics via K-Line, you communicate with only one ECU (point-to-point communication).

K-Line for a logical link of the functional group layer Do not select K-Line as the physical connection for a logical link of the *functional group* layer.

ECU diagnostics via Ethernet

IPv6 not supported Internet Protocol version 6 (IPv6) is not supported in connection with UDS on DoIP (Diagnostics over Internet Protocol).

Limitations for COMPARAMs

CAN-based diagnostic protocols The following limitations for COMPARAMs apply to CAN-based diagnostic protocols:

- Do not specify the baud rate via communication parameters (`COMPARAMs`) when you perform ECU diagnostics via CAN and when the CAN interface is already used by another device.
- The following COMPARAMs are not supported for CAN-based diagnostic protocols:
  - `CP_BitSamplePoint`
  - `CP_SyncJumpWidth`

  As a consequence, you cannot specify bit timing parameters for CAN-based ECU diagnostics.

K-Line-based diagnostic protocols The DCI-KLine1 does not support the `CP_P4Min` COMPARAM, which is used to specify the minimum interbyte time.

UDS diagnostic protocol The `CP_Cs` COMPARAM is not supported for the UDS diagnostic protocol.

Functional limitations

The following functional limitations apply:

- `Dynamically defined Local Identifier`s are not supported.
- `DIAG-VARIABLE`s are not supported.
- `MULTIPLE-ECU-JOB`s are not supported.
- For `TABLE`s, `TABLE-ENTRY`s are not supported.
- The `A_UTF8STRING` BASE-DATA-TYPE is not supported.
- Only default encodings are supported for the following simple data object properties (DOPs):

Simple DOP[1]	Supported Default Encoding
`A_ASCIISTRING`[2]	ISO-8859-1
`A_BYTEFIELD`	NONE
`A_UNICODE2STRING`	UCS-2

[1] Data object property

[2] The limitation applies to the request only.

- For parameters that reference a complex data object property (DOP), the bit length of a complex parameter must be an integer multiple of 8. The parameter must always start at a byte limit.

Limitations for the ASAM MCD-2 D v2.2.0 support

ControlDesk supports the ASAM MCD-2 D v2.2.0 ODX database standard. However, only the mandatory functionalities of ASAM MCD-2 D v2.2.0 are supported.

The following limitations apply to the functionalities of ASAM MCD-2 D v2.2.0 supported by ControlDesk:

- The ECU base variant and ECU variant layers inherit `COMPARAM` modifications made in the functional group layer. This behavior, however, is not compatible with ODX 2.2.0.
- `PROT-STACK-SNREF` specified for `COMPARM-REF` and for `LOGICAL-LINK` are not supported. `PROT-STACK-SNREF` must always be specified for the protocol.
- The `CP_UniqueRespIdTable` `COMPLEX-COMPARAM` is the only `COMPLEX-COMPARAM` supported by ControlDesk.
- The `CP_UniqueRespIdTable` `COMPLEX-COMPARAM` cannot be modified multiple times in one layer since the `ALLOW-MULTIPLE-VALUES` attribute is not supported.

SINGLE-ECU-JOBs

`SINGLE-ECU-JOB`s must be implemented as Java CLASS files or as JAR files in the ODX database.

Limitation for importing PDX packages

You can import a PDX package to specify an ODX diagnostics database. To execute diagnostic jobs and/or perform ECU flash memory programming, it is not sufficient that the related files (such as `CLASS`, `JAR`, or `HEX` files) are in the PDX package. You have to import these files in addition to the PDX package.

Limitations for measurement and calibration via the ECU Diagnostics device

You have to observe some conventions for performing measurement and calibration via the ECU Diagnostics device. Refer to Conventions in Connection with ODX Databases on page 22.

There are some limitations for performing measurement and calibration via the ECU Diagnostics device.

Local identifier in a diagnostic service Each diagnostic service must have only one request parameter and one response parameter for the definition of a local identifier. Otherwise, the diagnostic service cannot be identified.

Diagnostic variable names The names (`ShortName`) of diagnostics variables in a diagnostic service must be unique. That means if you specify the `DATA` semantic attribute for a request/response parameter of a single diagnostic variable, no other diagnostic variables can have the same name (`ShortName`) and the same semantic attribute within the same service.

This limitation does not apply if the local identifier of each diagnostic variable within a service is unique.

No display of read-only strings as measurement variables ControlDesk cannot identify read-only diagnostic variables which have the string type (character strings, byte fields etc.) as measurement variables. These read-only measurement variables from a read service are not displayed as measurement variables, but as parameters. You can measure them via a predefined polling raster, but you cannot calibrate them since there is no corresponding write service.

Limitations for selecting measurement rasters There are some limitations for selecting measurement rasters when you perform measurement and calibration via the ECU Diagnostics device:

- Rasters for measurement variables
  - Do not select *hardware polling rasters* in the Measurement Configuration controlbar for measuring measurement variables via the ECU Diagnostics device. Select *polling rasters* instead since they have a higher performance.
- Rasters for parameters
  - Do not select *hardware polling rasters* in the Measurement Configuration controlbar for measuring parameters via the ECU Diagnostics device. Select *polling rasters* instead since they have a higher performance.

Note

It is not recommended to measure parameters via the ECU Diagnostics device.

For details on measurement rasters, refer to Basics on Measurement Rasters (ControlDesk Measurement and Recording 🕮).

No source value display for diagnostic variables In ControlDesk instruments, you can display only the converted values of diagnostic variables. It is not possible to get the source values of diagnostic variables. The source value of diagnostic variables is identical to the converted value. Diagnostic variables that correspond to text tables are an exception.

Calibration and computation methods In an ODX database, computation methods can be defined for diagnostic variables. Such a method converts the calibrated physical value of a parameter to its source value on the ECU. If the `LINEAR`, `SCALE-LINEAR` or `TAB-INTP` computation methods are used, calibration using default increments may have no effect on the source value due to rounding effects. As a result, the previous value of the parameter is shown.

To solve this problem, you can configure the instrument's increment in the Properties controlbar or change the parameter value manually in the calibration instrument.

String variables with special character CR This limitation applies to variables of the string type. Suppose the variable description created from the ODX database contains variables of the string type. If such a variable contains the special character `CR` (carriage return) = `0x0D`, ControlDesk replaces this character by the "." character when you export the variable description to a data set. There are technical reasons for this.

Note

Keep in mind that when you download the data set, you also download the automatically changed string variable.

No termination of the up/download If you start online calibration and the ECU does not respond to a service request during up/download (for example, because the diagnostic interface has been disconnected), ControlDesk does not terminate up/downloading but processes all service requests as scheduled (including a timeout for each service request without a response). This can take some time, depending on the number of service requests and the configured timeout in the `COMPARAM-SPEC`.

The up/download is terminated, however, if the service request cannot be transmitted to the ECU (TX error).

Limitations in connection with the Fault Memory instrument

Changing a read service Using the XML configuration file allows you to select an individual read service for each logical link. Read services of the same logical links on different fault memory instruments are synchronized.

If you change the read service for one of the synchronized instruments, ControlDesk resets the Update Rate [s] to **`Off`** for the remaining ones. The instrument with the new selected read service takes the settings for Update Rate [s] and Read DTC Data from the other instruments with the same logical link and the same read service (or, if there are no others, the instrument gets the default settings: Update Rate [s] is set to **`Off`** and Read DTC Data is deactivated).

Automation interface: No multi-client support

ControlDesk's ASAM MCD-3D automation interface for diagnostics has the following limitations:

- Only one client (automation system) can access the ASAM MCD-3D automation interface for diagnostics at the same time. The automation interface cannot handle multiple clients.
- Only one run-time system object (D3System202) can be created by the client.
- An ECU Diagnostics device cannot be reconfigured via ControlDesk's ASAM MCD-3 D compatible interface. As a consequence, only the vehicles and logical links configured in the ControlDesk experiment are accessible to the interface.

  There is one exception: *All* the ECU variants that belong to a base variant using ControlDesk's ASAM MCD-3 D compatible interface are available, regardless of the configuration of the related ECU Diagnostics device. However, you cannot use these ECU variants via ASAM MCD-3 D.

Conventions in connection with ODX databases

When performing ECU diagnostics with ControlDesk, you have to observe some conventions in connection with the ODX database you use. Refer to Conventions in Connection with ODX Databases on page 22.

Glossary

Introduction

Briefly explains the most important expressions and naming conventions used in the ControlDesk documentation.

Where to go from here

Information in this section

Numerics

3-D Viewer An instrument for displaying items in a 3-D environment.

A

A2L file A file that contains all the relevant information on measurement and calibration variables in an ECU application ⧉ and the ECU's communication interface(s). This includes information on the variables' memory addresses and conversion methods, the memory layout and data structures in the ECU as well as interface description data (IF_DATA) ⧉.

Acquisition An object in the Measurement Configuration ⧉ controlbar that specifies the variables to be measured and their measurement configuration.

Active variable description The variable description that is currently active for a platform/device. Multiple variable descriptions can be assigned to one platform/device, but only one of them can be active at a time.

Additional write variable A scalar parameter or writable measurement variable that can be connected to an instrument in addition to the main variable ⧉. When the value of the main variable changes, the changed value is also applied to all the additional write variables connected to the instrument.

Airspeed Indicator An instrument for displaying the airspeed of a simulated aircraft.

Altimeter An instrument for displaying the altitude of a simulated aircraft.

Animated Needle An instrument for displaying the value of a connected variable by a needle deflection.

Application image An image file that contains all the files that are created when the user builds a real-time application. It particularly includes the variable description (SDF) file. To extend a real-time application, ControlDesk lets the user create an updated application image from a data set. The updated application image then contains a real-time application with an additional set of parameter values.

Artificial Horizon An instrument displaying the rotation on both the lateral and the longitudinal axis to indicate the angle of pitch and roll of a simulated aircraft. The Artificial Horizon has a pitch scale and a roll scale.

Automatic Reconnect Feature for automatically reconnecting to platform/device hardware, for example, when the ignition is turned off and on, or when the physical connection between the ControlDesk PC and the ECU is temporarily interrupted.

If the feature is enabled for a platform/device and if the platform/device is in the 'unplugged' state, ControlDesk tries to re-establish the logical connection to the platform/device hardware. After the logical connection is re-established, the platform/device has the same state as before the unplugged state was detected. A measurement started before the unplugged state was detected is resumed.

Automation A communication mechanism that can be used by various programming languages. A client can use it to control a server by calling methods and properties of the server's automation interface.

Automation script A script that uses automation to control an automation server.

Axis point object Common axis

B

Bar An instrument (or a value cell type of the Variable Array) for displaying a numerical value as a bar deflection on a horizontal or vertical scale.

Bitfield A value cell type of the Variable Array for displaying and editing the source value of a parameter as a bit string.

Bookmark A marker for a certain event during a measurement or recording.

Browser An instrument for displaying HTML and TXT files. It also supports Microsoft Internet Explorer© plug-ins that are installed on your system.

Bus communication replay A feature of the Bus Navigator that lets you replay logged bus communication data from a log file. You can add replay nodes to the Bus Navigator tree for this purpose. You can specify filters to replay selected parts of the logged bus communication.

Bus configuration A configuration of all the controllers, communication matrices, and messages/frames/PDUs of a specific communication bus such as CAN. ControlDesk lets you display and experiment with bus configurations in the Bus Navigator.

Bus connection A mode for connecting dSPACE real-time hardware to the host PC via bus. The list below shows the possible bus connections:

- dSPACE real-time hardware installed directly in the host PC
- dSPACE real-time hardware installed in an expansion box connected to the host PC via dSPACE link board

Bus Instrument An instrument available for the Bus Navigator. It can be configured for different purposes, for example, to display information on received messages (RX messages) or to manipulate and transmit messages (TX messages). The instrument is tailor-made and displays only the message- and signal-specific settings which are enabled for display and/or manipulation by ControlDesk during run time.

Bus logging A feature of the Bus Navigator that lets you log raw bus communication data. You can add logger nodes on different hierarchy levels of the Bus Navigator tree for this purpose. You can specify filters to log filtered bus communication. The logged bus communication can be replayed.

Bus monitoring A feature of the Bus Navigator that lets you observe bus communication. You can open monitoring lists and add monitor nodes on different hierarchy levels of the Bus Navigator tree for this purpose. You can specify filters to monitor filtered bus communication.

Bus Navigator A controlbar for handling bus messages, such as CAN messages, LIN frames, and Ethernet packets.

Bus statistics A feature of the Bus Navigator that lets you display and log statistical information on the bus load during bus monitoring.

Bypassing A method for replacing an existing ECU function by running a new function.

C

Calculated variable A scalar variable that can be measured and recorded, and that is derived from one or more *input variables*.

The following input variable types are supported:

- Measurement variables
- Single elements of measurement arrays or value blocks
- Scalar parameters, or existing calculated variables

The value of a calculated variable is calculated via a user-defined *computation formula* that uses one or more input variables.

Calculated variables are represented by the symbol.

CalDemo ECU A demo program that runs on the same PC as ControlDesk. It simulates an ECU on which the Universal Measurement and Calibration (XCP) protocol and the Unified Diagnostic Services (UDS) protocol are implemented.

The CalDemo ECU allows you to perform parameter calibration, variable measurement, and ECU diagnostics with ControlDesk under realistic conditions, but without having to have a real ECU connected to the PC. Communication between the CalDemo ECU and ControlDesk can be established via XCP on CAN or XCP on Ethernet, and UDS on CAN.

> **Tip**
>
> If communication is established via XCP on Ethernet, the CalDemo ECU can also run on a PC different from the PC on which ControlDesk is running.

The memory of the CalDemo ECU consists of two areas called memory page. Each page contains a complete set of parameters, but only one page is accessible by the CalDemo ECU at a time. You can easily switch the memory pages of the CalDemo ECU to change from one parameter to another in a single step.

Two ECU tasks run on the CalDemo ECU:

- ECU task #1 runs at a fixed sample time of 5 ms. In ControlDesk's **Measurement Configuration**, ECU task #1 is related to the time-based **5 ms**, **10 ms**, **50 ms** and **100 ms** measurement rasters of the CalDemo ECU.
- ECU task #2 has a variable sample time. Whenever the CalDemo ECU program is started, the initial sample time is 5 ms. This can then be increased or decreased by using the **dSPACE CalDemo** dialog.

  ECU task #2 is related to the **extEvent** measurement raster of the CalDemo ECU.

The CalDemo ECU can also be used to execute diagnostic services and jobs, handle DTCs and perform measurement and calibration via ECU diagnostics.

The CalDemo ECU program is run by invoking `CalDemo.exe`. The file is located in the `.\Demos\CalDemo` folder of the ControlDesk installation.

Calibration Changing the parameter values of real-time applications or ECU applications.

Calibration memory segment Part of the memory of an ECU containing the calibratable parameters. Memory segments can be defined as `MEMORY_SEGMENT` in the A2L file. ControlDesk can use the segments to evaluate the memory pages of the ECU.

ControlDesk lets you perform the calibration of:

- Parameters inside memory segments
- Parameters outside memory segments
- Parameters even if no memory segments are defined in the A2L file.

CAN Bus Monitoring device A device that monitors the data stream on a CAN bus connected to the ControlDesk PC.

The CAN Bus Monitoring device works, for example, with PC-based CAN interfaces such as the DCI-CAN2 or the DCI-CAN/LIN1.

The device supports the following variable description file types:

- DBC
- FIBEX
- AUTOSAR system description (ARXML)

CANGenerator A demo program that simulates a CAN system, that is, it generates signals that can be measured and recorded with ControlDesk. The program runs on the same PC as ControlDesk.

The CANGenerator allows you to use the CAN Bus Monitoring device under realistic conditions, but without having to have any device hardware connected to the PC.

The CAN (Controller Area Network) protocol is used for communication between the CANGenerator and ControlDesk. However, since the CANGenerator runs on the same PC as ControlDesk, ControlDesk does not communicate with the device via a real CAN channel, but via a *virtual CAN channel* implemented on the host PC.

You can start the CAN generator program by running `CANGenerator.exe`. The file is located in the `.\Demos\CANGenerator` folder of the ControlDesk installation.

Capture A data packet of all the measurement variables assigned to a measurement raster. The packet comprises the data that results from a single triggering of the raster.

CCP Abbreviation of CAN Calibration Protocol. This protocol can be implemented on electronic control units (ECUs) and allows users to access ECUs with measurement and calibration systems (MCS) such as ControlDesk.

The basic features of CCP are:

- Read and write access to the ECU memory, i.e., providing access for calibration
- Synchronous data acquisition
- Flash programming for ECU development purposes

The CCP protocol was developed by ASAM e.V. (Association for Standardization of Automation and Measuring Systems e.V.). For the protocol specification, refer to http://www.asam.net.

The following device supports ECUs with an integrated CCP service:

- CCP device

CCP device A device that provides access to an ECU with CCP connected to the ControlDesk PC via CAN, for example, for measurement and calibration purposes via CCP (CAN Calibration Protocol).

Check Button An instrument (or a cell type of the Variable Array) for displaying whether the value of a connected variable matches predefined values or for writing a predefined value to a connected variable.

cmdloader A command line tool for handling applications without using the user interface of an experiment software.

Common axis A parameter that consists of a 1-dimensional array containing axis points. A common axis can be referenced by one or more curves and/or maps. Calibrating the data points of a common axis affects all the curves and/or maps referencing the axis.

Common axes are represented by the symbol.

Common Program Data folder A standard folder for application-specific configuration data that is used by all users.

`%PROGRAMDATA%\dSPACE\<InstallationGUID>\<ProductName>`

or

`%PROGRAMDATA%\dSPACE\<ProductName>\<VersionNumber>`

Computation method A formula or a table that defines the transformation of a source value into a converted value (and vice versa). In addition to the computation methods defined in the variable description file, ControlDesk provides the *__Identity* computation method which means the converted and the source value are equal.

Connected A platform/device state defined by the following characteristics:

- A continuous logical connection is established between ControlDesk and the platform/device hardware.
- A platform/device must be in the 'connected' state before it can change to the 'measuring/recording' or 'online calibration started' state.
- Online calibration is impossible. ControlDesk did not yet adjust the memory segments containing calibration data in the platform/device and on the corresponding hardware. Offline calibration is possible.
- Platform/device configuration is not possible. However, you can invoke platform/device configuration for a platform/device that is in the connected state. ControlDesk temporarily sets the platform/device to the disconnected state.

The 'connected' platform/device state is indicated by the icon.

Connection mode dSPACE real-time systems can be installed within the host PC or connected to the host via a bus interface and/or via Ethernet. When the Ethernet is being used, different network clients might exist. The connection type being used and, in the case of Ethernet, the network client being used, determine the dSPACE systems that can be accessed.

Control primitive A special diagnostic communication object for changing communication states or protocol parameters, or for identifying (ECU) variants.

Controlbar A window or pane outside the working area. Can be docked to an edge of the main window or float in front of it. A controlbar can contain a document, such as a layout, or a tool, such as the Bus Navigator. It can be grouped with other controlbars in a window with tabbed pages.

ControlDesk The main version of ControlDesk for creating and running experiments, and for accessing dSPACE real-time hardware and VEOS. The functionality can be extended by optional software modules.

ControlDesk - Operator Version A version of ControlDesk that provides only a subset of functionality for running existing experiments. The functionality can be extended by optional software modules.

ControlDesk Bus Navigator Module An optional software module for ControlDesk for handling bus messages, such as CAN, LIN, and FlexRay messages, frames, and PDUs and Ethernet packets.

ControlDesk ECU Diagnostics Module An optional software module for ControlDesk that facilitates the calibration and validation of ECU diagnostic functions.

ControlDesk ECU Interface Module An optional software module for ControlDesk for calibration and measurement access to electronic control units (ECUs). The module is also required for calibration and measurement access to virtual ECUs (V-ECUs) used in SIL testing scenarios.

ControlDesk Signal Editor Module An optional software module for ControlDesk for the graphical definition and execution of signal generators for stimulating model variables of real-time/offline simulation applications.

Controller board Single-board hardware computing the real-time application. Contains a real-time processor for fast calculation of the model and I/O interfaces for carrying out the control developments.

Conversion table A table that specifies the value conversion of a source value into a converted value. In the case of verbal conversion, the converted value is a string that represents one numerical value or a range of numerical values.

Conversion type The type of a computation method, for example a linear function or a verbal computation method.

Curve A parameter that consists of

- A 1-dimensional array containing the axis points for the x-axis. This array can also be specified by a reference to a common axis.
- Another 1-dimensional array containing data points. The curve assigns one data point to each axis point.

Curves are represented by the ⊞ symbol.

D

DAQ module A hardware module for the acquisition of physical quantities

Data Cursor One or two cursors that are used to display the values of selected chart positions in a Time Plotter or an Index Plotter.

Data logger An object in the Measurement Configuration controlbar that lets you configure a data logging.

Data logger signal list A list that contains the variables to be included in subsequent data loggings on real-time hardware.

Data logging The recording of data on dSPACE real-time hardware that does not require a physical connection between the host PC and the real-time hardware. In contrast to flight recording, data logging is configured in ControlDesk.

Data set A set of the parameters and their values of a platform/device derived from the variable description of the platform/device. There are different types of data sets:

- Reference data set
- Sub data set
- Unassigned data set
- Working data set

DCI-CAN/LIN1 A dSPACE-specific interface between the host PC and the CAN/CAN FD bus and/or LIN bus. The DCI-CAN/LIN1 transfers messages between the CAN-/LIN-based devices and the host PC via the universal serial bus (USB).

DCI-CAN2 A dSPACE-specific interface between the host PC and the CAN bus. The DCI-CAN2 transfers CAN and CAN FD messages between the CAN-based devices and the host PC via the universal serial bus (USB).

DCI-GSI2 Abbreviation of *dSPACE Communication Interface - Generic Serial Interface 2*. A dSPACE-specific interface for ECU calibration, measurement and ECU interfacing.

DCI-GSI2 device A device that provides access to an ECU with DCI-GSI2 connected to the ControlDesk PC for measurement, calibration, and bypassing purposes via the ECU's debug interface.

DCI-KLine1 Abbreviation of *dSPACE Communication Interface - K-Line Interface*. A dSPACE-specific interface between the host PC and the diagnostics bus via K-Line.

Debug interface An ECU interface for diagnostics tasks and flashing.

Default raster A platform-/device-specific measurement raster that is used when a variable of the platform/device is connected to a plotter or a recorder, for example.

Deposition definition A definition specifying the sequence in which the axis point values of a curve or map are deposited in memory.

Device A software component for carrying out calibration and/or measurement, bypassing, ECU flash programming, or ECU diagnostics tasks.

ControlDesk provides the following devices:

- Bus devices:
  - CAN Bus Monitoring device
  - Ethernet Bus Monitoring device
  - LIN Bus Monitoring device
- ECU Diagnostics device
- GNSS device
- Measurement and calibration devices:
  - CCP device
  - DCI-GSI2 device
  - XCP on CAN device
  - XCP on Ethernet device

Each device usually has a variable description that specifies the device's variables to be calibrated and measured.

Diagnostic interface Interface for accessing the fault memory of an ECU.

Diagnostic job (often called Java job) Programmed sequence that is usually built from a sequence of the diagnostic service. A diagnostic job is either a single-ECU job or a multiple-ECU job, depending on whether it communicates with one ECU or multiple ECUs.

Diagnostic protocol A protocol that defines how an ECU communicates with a connected diagnostic tester. The protocol must be implemented on the ECU and on the tester. The diagnostics database specifies the diagnostic protocol(s) supported by a specific ECU.

ControlDesk's ECU Diagnostics device supports CAN and K-Line as the physical layers for communication with an ECU connected to the ControlDesk PC. For information on the supported diagnostic protocols with CAN and K-Line, refer to Basics of ECU Diagnostics with ControlDesk on page 10.

Diagnostic service A service implemented on the ECU as a basic diagnostic communication element. Communication is performed by selecting a service, configuring its parameters, executing it, and receiving the ECU results. When a service is executed, a defined request is sent to the ECU and the ECU answers with a specific response.

Diagnostic trouble code (DTC) A hexadecimal index for the identification of vehicle malfunctions. DTCs are stored in the fault memory of ECUs and can be read by diagnostic testers.

Diagnostics database A database that completely describes one or more ECUs with respect to diagnostics communication. ControlDesk supports the ASAM MCD-2 D ODX database format, which was standardized by ASAM e.V. (Association for Standardisation of Automation and Measuring Systems e.V.). For the format specification, refer to http://www.asam.net.

Proprietary diagnostics database formats are not supported by ControlDesk.

Diagnostics Instrument An instrument for communicating with an ECU via the diagnostic protocol using diagnostic services, diagnostic jobs, and control primitives.

Disabled A platform/device state defined by the following characteristics:

- No logical connection is established between ControlDesk and the platform/device hardware.
- When a platform/device is disabled, ControlDesk does not try to establish the logical connection for that platform/device. Any communication between the platform/device hardware and ControlDesk is rejected.
- Online calibration is impossible. Offline calibration is possible.
- Platform/device configuration is possible.

The 'disabled' platform/device state is indicated by the 🔒 icon.

Disconnected A platform/device state defined by the following characteristics:

- No logical connection is established between ControlDesk and the platform/device hardware.
- When a platform/device is in the disconnected state, ControlDesk does not try to re-establish the logical connection for that platform/device.
- Online calibration is impossible. Offline calibration is possible.
- Platform/device configuration is possible.

The 'disconnected' platform/device state is indicated by the ✗ icon.

Display An instrument (or a value cell type of the Variable Array) for displaying the value of a scalar variable or the text content of an ASCII variable.

Documents folder A standard folder for user-specific documents.

`%USERPROFILE%\Documents\dSPACE\<ProductName>\`
`<VersionNumber>`

DS1006 Processor Board platform A platform that provides access to a DS1006 Processor Board connected to the host PC for HIL simulation and function prototyping purposes.

DS1007 PPC Processor Board platform A platform that provides access to a single multicore DS1007 PPC Processor Board or a DS1007 multiprocessor system consisting of two or more DS1007 PPC Processor Boards, connected to the host PC for HIL simulation and function prototyping purposes.

DS1104 R&D Controller Board platform A platform that provides access to a DS1104 R&D Controller Board installed in the host PC for function prototyping purposes.

DS1202 MicroLabBox platform A platform that provides access to a MicroLabBox connected to the host PC for function prototyping purposes.

DsDAQ service A service in a real-time application or offline simulation application (OSA) that provides measurement data from the application to the host PC. Unlike the host service, the DsDAQ service lets you perform, for example, triggered measurements with complex trigger conditions.

The following platforms support applications that contain the DsDAQ service:

- DS1007 PPC Processor Board platform
- DS1202 MicroLabBox platform
- MicroAutoBox III platform
- SCALEXIO platform
- VEOS platform
- XIL API MAPort platform

dSPACE Calibration and Bypassing Service An ECU service for measurement, calibration, bypassing, and ECU flash programming. The dSPACE Calibration and Bypassing Service can be integrated on the ECU. It provides access to the ECU application and the ECU resources and is used to control communication between an ECU and a calibration and/or bypassing tool.

With the dSPACE Calibration and Bypassing Service, users can run measurement, calibration, bypassing, and flash programming tasks on an ECU via the DCI-GSI2. The service is also designed for bypassing ECU functions using dSPACE prototyping hardware by means of the RTI Bypass Blockset in connection with DPMEM PODs. The dSPACE Calibration and Bypassing Service allows measurement, calibration, and bypassing tasks to be performed in parallel.

dSPACE Internal Bypassing Service An ECU service for on-target prototyping. The dSPACE Internal Bypassing Service can be integrated in the ECU application. It lets you add additional functions to be executed in the context of the ECU application without the need for recompiling the ECU application.

dSPACE Log A collection of errors, warnings, information, questions, and advice issued by all dSPACE products and connected systems over more than one session.

dSPACE system A hardware system such as a MicroAutoBox III or SCALEXIO system on which the real-time application runs.

Duration trigger A trigger that defines a duration. Using a duration trigger, you can, for example, specify the duration of data acquisition for a measurement raster. A duration trigger can be used as a stop trigger.

E

ECU Abbreviation of *electronic control unit*.

ECU application A sequence of operations executed by an ECU. An ECU application is mostly represented by a group of files such as ECU Image files, MAP files, A2L files and/or software module description files.

ECU calibration interface Interface for accessing an ECU by either emulating the ECU's memory or using a communication protocol (for example, XCP on CAN).

ECU diagnostics Functions such as:

- Handling the ECU fault memory: Entries in the ECU´s fault memory can be read, cleared, and saved.
- Executing diagnostic services and jobs: Users can communicate with an ECU via a diagnostic protocol using diagnostic services, diagnostic jobs, and control primitives.

ControlDesk provides the ECU Diagnostics device device to access ECUs for diagnostic tasks. Communication is via diagnostic protocols implemented on the ECUs.

ECU diagnostics with ControlDesk are completely based on Open Diagnostic Data Exchange (ODX), the ASAM MCD-2 D diagnostics standard.

ControlDesk provides the Fault Memory Instrument and the Diagnostics Instrument for ECU diagnostics tasks.

ECU Diagnostics device A device that provides access to ECUs connected to the ControlDesk PC via CAN or K-Line for diagnostics or flash programming purposes.

ControlDesk provides the *ECU Diagnostics v2.0.2* device, which supports the ASAM MCD-3 D V2.0.2 standard.

ControlDesk supports the following ODX database standards:

- ASAM MCD-2 D V2.0.1
- ASAM MCD-2 D V2.2.0 (ISO 22901-1)

ECU flash programming A method by which new code or data is stored in ECU flash memory.

ECU Image file A binary file that is part of the ECU application. It usually contains the code of an ECU application and the data of the parameters within the application. It can be stored as an Intel Hex (HEX) or Motorola S-Record (MOT or S19) file.

EESPort Configurations controlbar A controlbar for configuring error configurations.

Electrical error simulation Simulating electrical errors such as loose contacts, broken cables, and short-circuits, in the wiring of an ECU. Electrical error simulation is performed by the failure simulation hardware of an HIL simulator.

Electrical Error Simulation port (EESPort) An *Electrical Error Simulation port* (EESPort) provides access to a failure simulation hardware for simulating electrical errors in an ECU wiring according to the ASAM AE XIL API standard.

The configuration of the EESPort is described by a hardware-dependent *port configuration* and one or more *error configurations*.

Environment model A model that represents a part or all of the ECU's environment in a simulation scenario.

The environment model is a part of the simulation system.

Environment VPU The executable of an environment model built for the VEOS platform. An environment VPU is part of an offline simulation application (OSA).

Error An electrical error that is specified by:

- An error category
- An error type
- A load type

Error category The error category defines how a signal is disturbed. Which errors you can create for a signal depends on the connected failure simulation hardware.

Error configuration An XML file that describes a sequence of errors you want to switch during electrical error simulation. Each error configuration comprises error sets with one or more errors.

Error set An error set is used to group errors (pin failures).

Error type The error type specifies the way an error category – i.e., an interruption or short circuit of signals – is provided. The error type defines the disturbance itself.

Ethernet Bus Monitoring device A device that monitors the data stream on an Ethernet network connected to the ControlDesk PC.

The device supports the following variable description file type:

- AUTOSAR system description (ARXML)

Ethernet connection A mode for connecting dSPACE real-time hardware to the host PC via Ethernet. The list below shows the possible Ethernet connections:

- dSPACE real-time hardware installed in an expansion box connected to the host PC via Ethernet.
- MicroAutoBox II/III and MicroLabBox connected via Ethernet.

Ethernet decoding A feature of the Bus Navigator that lets you view protocol data and raw data of an Ethernet frame.

Event An event that is triggered by an action performed in ControlDesk.

Event context The scope of validity of event sources and events. There is one event handler code area for each event context.

Event handler Code that is executed when the related event occurs.

Event management Functionality for executing custom code according to actions triggered by ControlDesk.

Event source An object providing and triggering events.
LayoutManagement is an example of an event source.

Event state State of an event. ControlDesk provides the following event states:

- No event handler is defined
- Event handler is defined and enabled
- Event handler is defined and disabled
- Event handler is defined, but no Python code is available
- Event handler is deactivated because a run-time error occurred during the execution of the Python code

Expansion box A box that hosts dSPACE boards. It can be connected to the host PC via bus connection or via network.

Experiment A container for collecting and managing information and files required for a parameter calibration and/or measurement task. A number of experiments can be collected in a project but only one of them can be active.

Extension script A Python script (PY or PYC file) that is executed each time ControlDesk starts up. An extension script can be executed for all users or user-specifically.

F

Failure insertion unit Hardware unit used with dSPACE simulators to simulate failures in the wiring of an ECU, such as broken wire and short circuit to ground.

Fault memory Part of the ECU memory that stores diagnostic trouble code (DTC) entries with status and environment information.

Fault Memory Instrument An instrument for reading, clearing, and saving the content of the ECU's fault memory.

Firmware update An update for the firmware installed in the board's flash memory. Firmware should be updated if it is older than required by the real-time application to be downloaded.

Fixed axis An axis with data points that are not deposited in the ECU memory. Unlike a common axis, a fixed axis is specified within a curve or map. The parameters of a fixed axis cannot be calibrated.

Fixed parameter A parameter that has a fixed value during a running simulation. Changing the value of a fixed parameter does not immediately affect the simulation results. The affect occurs only after you stop the simulation and start it again. A fixed parameter is represented by an added pin in its symbol, for example: P.

Flash job A specific diagnostic job for flashing the ECU memory. A flash job implements the process control for flashing the ECU memory, such as initialization, security access, writing data blocks, etc.

Flight recording The recording of data on dSPACE real-time hardware that does not require a physical connection between the host PC and the real-time hardware. In contrast to data logging, flight recording is not configured in ControlDesk but via RTI and RTLib.

Frame An instrument for adding a background frame to a layout, for example, to visualize an instrument group.

G

Gauge An instrument for displaying the value of the connected variable by a needle deflection on a circular scale.

Gigalink module A dSPACE board for connecting several processor boards in a multiprocessor system. The board allows high-speed serial data transmission via fiber-optic cable.

GNSS data Positioning and timing data that is transmitted by a Global Navigation Satellite System (GNSS), such as GPS, GLONASS, or Galileo. GNSS receivers use this data to determine their location.

GNSS device A device that provides positioning data from a GNSS receiver (e.g., a serial GPS mouse) in ControlDesk.

ControlDesk provides the *GNSS (GPS, GLONASS, Galileo, ...)* device that supports various global navigation satellite systems.

GPX file An XML file that contains geodata, such as waypoints, routes, or tracks. In ControlDesk, you can import GPX files to visualize GNSS positioning data in a Map instrument.

Group A collection of variables that are grouped according to a certain criterion.

H

Heading Indicator An instrument displaying the heading direction of a simulated aircraft on a circular scale.

Host service A service in a real-time application that provides measurement data from the application to the host PC.

The following platforms support applications that contain the host service:

- DS1006 Processor Board platform
- DS1104 R&D Controller Board platform
- MicroAutoBox platform
- Multiprocessor System platform

I

Index Plotter A plotter instrument for displaying signals that are measured in an event-based raster (index plots).

Input quantity A measurement variable that is referenced by a common axis and that provides the input value of that axis.

Instrument An on-screen representation that is designed to monitor and/or control simulator variables interactively and to display data captures. Instruments can be arranged freely on layouts.

The following instruments can be used in ControlDesk:

- 3-D Viewer
- Airspeed Indicator
- Altimeter
- Animated Needle
- Artificial Horizon
- Bar
- Browser
- Bus Instrument
- Check Button
- Diagnostics Instrument
- Display
- Fault Memory Instrument
- Frame
- Gauge
- Heading Indicator
- Index Plotter
- Invisible Switch
- Knob
- Multistate Display
- Multiswitch
- Numeric Input
- On/Off Button

- Push Button
- Radio Button
- Selection Box
- Slider
- Sound Controller
- Static Text
- Steering Controller
- Table Editor
- Time Plotter
- Variable Array
- XY Plotter

Instrument Navigator A controlbar that displays a tree with all the instruments of the active layout and all the variables that are connected to them. The Instrument Navigator's main function is easy selection of instruments in complex layouts.

Instrument script A Python script used to extend the functionality of an instrument.

Instrument Selector A controlbar that provides access to ControlDesk's instruments. The instruments can be placed on a layout via double-click or drag & drop.

Interface description data (IF_DATA) An information structure, mostly provided by an A2L file, describing the type, features and configuration of an implemented ECU interface.

Internal Interpreter ControlDesk's built-in programming interface for editing, running and importing Python scripts. It contains an Interpreter controlbar where the user can enter Python commands interactively and which displays output and error messages of Python commands.

Interpreter controlbar A controlbar that can be used to execute line-based commands. It is used by the Internal Interpreter to print out Python standard error messages and standard output during the execution or import of Python scripts.

Invisible Switch An instrument for defining an area that is sensitive to mouse operations.

IOCNET IOCNET (I/O carrier network) is a dSPACE-specific high-speed serial communication bus that connects all the real-time hardware in a SCALEXIO system. IOCNET can also be used to build a multiprocessor system that consists of multiple SCALEXIO processor hardware components.

K

Knob An instrument for displaying and setting the value of the connected variable by means of a knob on a circular scale.

L

Label list A list of user-defined variables that can be used for saving connected variables, etc.

Layout A window with instrument↗s connected to variables of one or more simulation models.

Layout Navigator A controlbar↗ that displays all opened layout↗s. It can be used for switching between layouts.

Layout script A Python script used to extend the functionality of a layout↗.

Leading raster The measurement raster↗ that specifies the trigger↗ settings for the Time Plotter↗ display. The leading raster determines the time range that is visible in the plotter if a start and stop trigger is used for displaying the signals.

LIN Bus Monitoring device A device that monitors the data stream on a LIN bus connected to the ControlDesk PC.

The LIN Bus Monitoring device works, for example, with PC-based LIN interfaces.

The device supports the following variable description file types:

- LDF
- FIBEX
- AUTOSAR system description (ARXML)

Load type The load type specifies the option to disturb a signal with or without load rejection.

Local Program Data folder A standard folder for application-specific configuration data that is used by the current, non-roaming user.

`%USERPROFILE%\AppData\Local\dSPACE\<InstallationGUID>\<ProductName>`

Logical link A representation of an ECU specified in the diagnostics database. A logical link contains information on the ECU itself, and all the information required for accessing it, such as the diagnostic protocol↗ used for communication between the ECU and ControlDesk. Each logical link is represented by a unique short name in the ODX database.

Look-up table A look-up table maps one or more input values to one output value. You have to differentiate between the following look-up table types:

- A 1-D look-up table maps one input value to one output value.
- A 2-D look-up table maps two input values to one output value.
- An n-D look-up table maps multidimensional table data with 3 or more input values to one output value.

Look-up table is a generic term for curves and maps.

M

Main variable A scalar variable that is visualized in an instrument that can be used to change parameter values. In addition to the main variable, additional write variables can also be connected to (but not visualized in) the same instrument. When you change the value of the main variable in an instrument, the changed value is also applied to all the additional write variables connected to that instrument.

Map A parameter that consists of

- A 1-dimensional array containing the axis points for the x-axis. This array can also be specified by a reference to a common axis.
- A 1-dimensional array containing the axis points for the y-axis. This array can also be specified by a reference to a common axis.
- A 2–dimensional array containing data points. The map assigns one data point of the array to each pair of x-axis and y-axis points.

Maps are represented by the symbol.

Map file A file that contains symbols (symbolic names) and their physical addresses. It is generated during a build process of an ECU application.

Map instrument A customized Browser instrument. It uses an instrument script to open a web map and connect positioning data to the map. The Map instrument offers prepared connection nodes to connect variables with GNSS data.

Measurement Viewing and analyzing the time traces of variables, for example, to observe the effects of ECU parameter changes.

ControlDesk provides various instruments for measuring variables.

Measurement (variable type) A scalar variable that can be measured, including individual elements of a measurement array.

Measurement variables are represented by the symbol.

Measurement array A 1-, 2-, or 3-dimensional array of measurement variables. In variable lists, ControlDesk displays entries for the measurement array itself and for each array element.

Measurement arrays are represented by the ▦ symbol.

Measurement buffer A ring buffer that buffers measurement data at the start of a measurement. The measurement buffer size determines the amount of data that can be buffered. Earlier values are overwritten by later values when the buffer capacity is exceeded (buffer overflow).

Measurement Configuration A controlbar that allows you to configure measurement, recording and data logging.

Measurement Data API Application programming interface for accessing measurement data. The API lets the user access measurement data without having to use ControlDesk.

Measurement Data Pool A controlbar that provides access to measurement data recorded in measurement data files.

Measurement raster Specification of how often a value of a variable is updated during a measurement. A measurement raster can be derived from a measurement service.

Measurement service The generic term for the following services:

- CCP service
- DsDAQ service
- Host service
- XCP service

Measurement signal list A list containing the variables to be included in subsequent measurements and recording. The list is global for all platforms/devices of the current experiment. The measurement signal list is available in the configuration area of the **Measurement Configuration** controlbar.

Measurement variable Any variable type that can be measured but not calibrated.

Measuring/recording A platform/device state defined by the following characteristics:

- A continuous logical connection is established between ControlDesk and the platform/device hardware.
- Online calibration is possible. Parameter values can be changed directly on the platform/device hardware.
- A measurement (or recording) is running.
- Platform/device configuration is not possible.

The 'measuring' / 'recording' platform/device state is indicated by the ▸ icon.

Memory page An area of a calibration memory. Each page contains a complete set of parameters of the platform/device hardware, but only one of the pages is “visible” to the microcontroller of the ECU or the real-time processor (RTP) of the platform hardware at a time.

ControlDesk supports platform/device hardware with up to two memory pages. These are usually the working page and the reference page. The parameter values on the two memory pages usually are different. ControlDesk lets you switch from one page to the other, so that when parameters are changed on one page, the changes can be made available to the ECU or prototyping hardware via a single page switch.

Messages controlbar A controlbar displaying a history of all error and warning messages that occur during work with ControlDesk.

MicroAutoBox III platform A platform that provides access to a MicroAutoBox III connected to the host PC for function prototyping purposes such as Bypassing.

MicroAutoBox platform A platform that provides access to a MicroAutoBox II connected to the host PC for function prototyping purposes such as bypassing.

Mirrored memory A memory area created by ControlDesk on the host PC that mirrors the contents of the available memory pages of calibration and prototyping hardware. For hardware with two memory pages, the mirrored memory is divided into a reference and a working page, each of them containing a complete set of parameters. When a calibration or prototyping platform/device is added to an experiment, ControlDesk initially fills the available memory pages of the mirrored memory with the contents of the ECU Image file (initial filling for calibration devices) or with the contents of the SDF file (initial filling for platforms).

- Mirrored memory for offline calibration

  Parameter values can even be changed offline. Changes to parameter values that are made offline affect only the mirrored memory.
- Offline-to-online transition for online calibration

  For online calibration, an offline-to-online transition must be performed. During the transition, ControlDesk compares the memory pages of the hardware of each platform/device with the corresponding pages of the mirrored memory. If the pages differ, the user has to equalize them by uploading them from the hardware to the host PC, or downloading them from the host PC to the hardware.
- Mirrored memory for online calibration

  When ControlDesk is in the online mode, parameter value changes become effective synchronously on the memory pages of the hardware and in the mirrored memory. In other words, parameter values on the hardware and on the host PC are always the same while you are performing online calibration.

Modular system A dSPACE processor board and one or more I/O boards connected to it.

Multi-capture history The storage of all the captures acquired during a triggered measurement. The amount of stored data depends on the measurement buffer.

Multi-pin error A feature of the SCALEXIO concept for electrical error simulation that lets you simulate a short circuit between three or more signal channels and/or bus channels. The channels can be located on the same or different boards or I/O units. You can simulate a short circuit between:

- Channels of the same signal category (e.g., four signal generation channels)
- Channels of different signal categories (e.g., three signal generation channels and two signal measurement channels)
- Signal channels and bus channels (e.g., two signal generation channels, one signal measurement channel, and one bus channel)

Multiple electrical errors A feature of the SCALEXIO concept for electrical error simulation that lets you switch electrical errors at the same time or in succession. For example, you can simulate an open circuit for one channel and a short circuit for another channel at the same time, without deactivating the first error.

Multiprocessor System platform A platform that provides access to:

- A multicore application running on a multicore DS1006 board
- A multiprocessor application on a multiprocessor system consisting of two or more DS1006 processor boards interconnected via Gigalink.

ControlDesk handles a multiprocessor/multicore system as a unit and uses one system description file (SDF file) to load the applications to all the processor boards/cores in the system.

Multistate Display An instrument for displaying the value of a variable as an LED state and/or as a message text.

Multistate LED A value cell type of the Variable Array for displaying the value of a variable as an LED state.

Multiswitch An instrument for changing variable values by clicking sensitive areas in the instrument and for visualizing different states depending on the current value of the connected variable.

N

Numeric Input An instrument (or a value cell type of the Variable Array) for displaying and setting the value of the connected variable numerically.

O

Observing variables Reading variable values cyclically from the dSPACE real-time hardware and displaying their current values in ControlDesk, even if no measurement is running. Variable observation is performed without using a measurement buffer, and no value history is kept.

For platforms that support variable observation, variable observation is available for parameters and measurement variables that are visualized in single-shot instruments (all instruments except for a plotter). If you visualize a variable in a single-shot instrument, the variable is not added to the measurement signal list. Visualizing a parameter or measurement variable in a plotter automatically adds the variable to the measurement signal list.

ControlDesk starts observing variables if one of the following conditions is true:

- Online Calibration is started for the platform.

  All the parameters and measurement variables that are visualized in single-shot instruments are observed.

- Measurement is started for the platform.

  All the visualized parameters and measurement variables that are not activated for measurement in the measurement signal list are observed. Data of the activated parameters and measurement variables is acquired using measurement rasters.

ODX database Abbreviation of Open Diagnostic Data Exchange, a diagnostics database that is the central ECU description for working with an ECU Diagnostics device in ControlDesk. The ODX database contains all the information required to perform diagnostic communication between ControlDesk and a specific ECU or set of ECUs in a vehicle network. ControlDesk expects the database to be compliant with ASAM MCD-2 D (ODX).

Offline State in which the parameter values of platform/device hardware in the current experiment cannot be changed. This applies regardless of whether or not the host PC is physically connected to the hardware.

The mirrored memory allows parameter values to be changed even offline.

Offline simulation A PC-based simulation in which the simulator is not connected to a physical system and is thus independent of the real time.

Offline simulation application (OSA) An offline simulation application (OSA) file is an executable file for VEOS. After the build process with a tool such as the VEOS Player, the OSA file can be downloaded to VEOS.

An OSA contains one or more VPUs, such as V-ECUs and/or environment VPUs.

On/Off Button An instrument (or a value cell type of the Variable Array) for setting the value of the connected parameter to a predefined value when the button is pressed (On value) and released (Off value).

Online calibration started A platform/device state defined by the following characteristics:

- A continuous logical connection is established between ControlDesk and the platform/device hardware.
- Online calibration is possible. Parameter values can be changed directly on the platform/device hardware.
- Platform/device configuration is not possible.

Before starting online calibration, ControlDesk lets you compare the memory pages on the platform/device hardware with the corresponding pages of the mirrored memory. If the parameter values on the pages differ, they must be equalized by uploading the values from the hardware to ControlDesk, or downloading the values from ControlDesk to the hardware. However, a page cannot be downloaded if it is read-only.

The 'online calibration started' platform/device state is indicated by the ⇕ symbol.

Operation signal A signal which represents the result of an arithmetical operation (such as addition or multiplication) between two other signals.

Operator mode A working mode of ControlDesk in which only a subset of the ControlDesk functionality is provided. You can work with existing experiments but not modify them, which protects them from unintentional changes.

Output parameter A parameter or writable measurement whose memory address is used to write the computed value of a calculated variable to.

P

Parameter Any variable type that can be calibrated.

Parameter (variable type) A scalar parameter, as well as the individual elements of a value block.

Scalar parameters are represented by the **P** symbol.

Parameter limits Limits within which parameters can be changed. Parameters have hard and weak limits.

- Hard limits

  Hard limits designate the value range of a parameter that you *cannot* cross during calibration.

  The hard limits of a parameter originate from the corresponding variable description and cannot be edited in ControlDesk.
- Weak limits

  Weak limits designate the value range of a parameter that you *should not* cross during calibration. When you cross the value range defined by the weak limits, ControlDesk warns you.

  In ControlDesk, you can edit the weak limits of a parameter within the value range given by the parameter's hard limits.

PHS (Peripheral High Speed) bus A dSPACE-specific bus for communication between a processor board and the I/O boards in a modular system. It allows direct I/O operations between the processor board (bus master) and I/O boards (bus slaves).

PHS-bus-based system A modular dSPACE system consisting of a processor board such as the DS1006 Processor Board and I/O boards. They communicate with each other via the PHS (Peripheral High Speed) bus.

Pitch variable A variable connected to the pitch scale of an Artificial Horizon.

Platform A software component representing a simulator where a simulation application is computed in real-time (on dSPACE real-time hardware) or in non-real-time (on VEOS).

ControlDesk provides the following platforms:

- DS1006 Processor Board platform
- DS1007 PPC Processor Board platform
- DS1104 R&D Controller Board platform
- DS1202 MicroLabBox platform
- MicroAutoBox platform
- MicroAutoBox III platform
- Multiprocessor System platform
- SCALEXIO platform
- VEOS platform
- XIL API MAPort platform

Each platform usually has a variable description that specifies its variables.

Platform trigger A trigger that is available for a platform and that is evaluated on the related dSPACE real-time hardware or VEOS.

Platforms/Devices controlbar A controlbar that provides functions to handle devices, platforms, and the applications assigned to the platforms.

Plotter instrument ControlDesk offers three plotter instruments with different main purposes:

- The Index Plotter displays signals in relation to events.
- The Time Plotter displays signals in relation to measurement time.
- The XY Plotter displays signals in relation to other signals.

Port configuration To interface the failure simulation hardware, an EESPort needs the hardware-dependent *port configuration file* (PORTCONFIG file). The file's contents must fit the connected HIL simulator architecture and its failure simulation hardware.

Postprocessing The handling of measured and recorded data by the following actions:

- Displaying measured or recorded data
- Zooming into measured or recorded signals with a plotter
- Displaying the values of measurement variables and parameters as they were at any specific point in time

Processor board A board that computes real-time applications. It has an operating system that controls all calculations and communication to other boards.

Project A container for collecting and managing the information and files required for experiment/calibration/modification tasks in a number of experiments. A project collects the experiments and manages their common data.

Project controlbar A controlbar that provides access to projects and experiments and all the files they contain.

Project root directory The directory on your file system to which ControlDesk saves all the experiments and documents of a project. Every project is associated with a project root directory, and several projects can use the same project root directory. The user can group projects by specifying several project root directories.

ControlDesk uses the Documents folder as the default project root directory unless a different one is specified.

Properties controlbar A controlbar providing access to the properties of, for example, platforms/devices, layouts/instruments, and measurement/recording configurations.

Proposed calibration A calibration mode in which the parameter value changes that the user makes do not become effective on the hardware until they are applied. This allows several parameter changes to be written to the hardware together. Being in proposed calibration mode is like being in the offline calibration mode temporarily.

Push Button An instrument (or a value cell type of the Variable Array) for setting the value of the connected parameter by push buttons.

Python Editor An editor for opening and editing PY files.

Q

Quick start measurement A type of measurement in which all the ECU variables configured for measurement are measured and recorded, starting with the first execution of an ECU task. ControlDesk supports quick start measurements on ECUs with DCI-GSI2, CCP, and XCP (except for XCP on Ethernet with the TCP transmission protocol).

Quick start measurement can be used to perform cold start measurements. Cold start means that the vehicle and/or the engine are cooled down to the temperature of the environment and then started. One reason for performing cold start measurements is to observe the behavior of an engine during the warm-up phase.

R

Radio Button An instrument for displaying and setting the value of the connected parameter by radio buttons.

Real-time application An application that can be executed in real time on dSPACE real-time hardware. A real-time application can be built from a Simulink model containing RTI blocks, for example.

Record layout A record layout is used to specify a data type and define the order of the data in the memory of the target system (ECU, for example). For scalar data types, a record layout allows you to add an address mode (direct or indirect). For structured (aggregated) data types, the record layout specifies all the structure elements and the order they appear in.

The `RECORD_LAYOUT` keyword in an A2L file is used to specify the various record layouts of the data types in the memory. The structural setup of the various data types must be described in such a way that a standard application system will be able to process all data types (reading, writing, operating point display etc.).

Record layout component A component of a record layout. A structured record layout consists of several components according to the ASAP2 specification. For example, the AXIS_PTS_X component specifies the x-axis points, and the FNC_VALUES component describes the function values of a map or a curve.

Recorder An object in the Measurement Configuration controlbar that specifies and executes the recording of variables according to a specific measurement configuration.

Recorder signal list A list that contains the variables to be included in subsequent recordings.

Recording Saving the time traces of variables to a file. Both measurement variables and parameters can be recorded. Recorded data can be postprocessed directly in ControlDesk.

A recording can be started and stopped immediately or via a trigger:

- Immediate recording

  The recording is started and stopped without delay, without having to meet a trigger condition.

- Triggered recording

  The recording is not started or stopped until certain trigger conditions are met. These conditions can be defined and edited in ControlDesk.

Reduction data Additional content in an MF4 file that allows for visualizing the MF4 file data depending on the visualization resolution. Reduction data therefore improves the performance of the visualization and postprocessing of measurement data.

Reference data set A read-only data set assigned to the reference page of a device that has two memory pages. There can be only one reference data set for each device. The reference data set is read-only.

Reference page Memory area containing the parameters of an ECU. The reference page contains the read-only reference data set.

> **Note**
>
> Some platforms/devices provide only a working page. You cannot switch to a reference page in this case.

Resynchronization Mechanism to periodically synchronize the drifting timers of the platform/device hardware ControlDesk is connected to. Resynchronization means adjustment to a common time base.

Roll variable A variable connected to the roll scale of an Artificial Horizon.

S

Sample count trigger A trigger that specifies the number of samples in a data capture.

A sample count trigger can be used as a stop trigger.

SCALEXIO platform A platform that provides access to a single-core, multicore or multiprocessor SCALEXIO system connected to the host PC for HIL simulation and function prototyping purposes.

SCALEXIO system A dSPACE hardware-in-the-loop (HIL) system consisting of at least one processing hardware component, I/O boards, and I/O units. They communicate with each other via the IOCNET. In a SCALEXIO system, two types of processing hardware can be used, a DS6001 Processor Board or a real-time industry PC as the SCALEXIO Processing Unit. The SCALEXIO system simulates the environment to test an ECU. It provides the sensor signals for the ECU, measures the signals of the ECU, and provides the power (battery voltage) for the ECU and a bus interface for restbus simulation.

SDF file The system description file that describes the files to be loaded to the individual processing units of a simulation platform. It also contains the variable description of the relevant simulation application.

The SDF file is generated automatically when the TRC file is built.

Segment The minimum part a segment signal can consist of.

There are different kinds of segments to be used in segment signals:

- Segments to form synthetic signal shapes (sine, sawtooth, ramp, etc.)
- Segments to perform arithmetical operations (addition, multiplication) with other segments
- Segments to represent numerical signal data (measured data)

Segment signal A signal consisting of one or more segments.

Selection Box An instrument for selecting a text-value entry and setting the respective numerical value for the connected variable.

Signal

- Representation of a variable measured in a specific measurement raster.
- Generic term for segment signals and operation signals.

  A signal is part of a signal description set which can be displayed and edited in the working area.

Signal description set A group of one or more signals.

A signal description set and its signals can be edited in the working area by means of the Signal Editor. Each signal description set is stored as an STZ file either in the **Signal Description Sets** folder or in the **Signal Generators** folder.

Signal Editor A software component to create, configure, display, and manage signals in signal description sets.

Signal file A file that contains the wiring information of a simulator and that is part of the standard dSPACE documentation of dSPACE Simulator Full-Size. Normally, dSPACE generates this file when designing the simulator. Before using a failure simulation system, users can adapt the signal file to their needs.

Signal generator An STZ file containing a signal description set and optional information about the signal mapping, the description of variables, and the real-time platform.

The file is located in the **Signal Generators** folder and used to generate, download, and control Real-Time Testing sequences, which are executed on the real-time platform to stimulate model variables in real time.

Signal Mapping A controlbar of the Signal Editor to map model variables to signals and variable aliases of a signal generator.

Signal Selector A controlbar of the Signal Editor. The Signal Selector provides signals and segments for arranging and configuring signal description sets in the working area.

SIL testing Abbreviation of *software-in-the-loop testing*.

Simulation and testing of individual software functions, complete virtual ECUs (V-ECUs), or even V-ECU networks on a local PC or highly parallel in the cloud independently of real-time constraints and real hardware.

Simulation application The generic term for offline simulation application (OSA) and real-time application.

Simulation system A description of the composition of V-ECU models, environment models, real ECUs, and their interconnections required for simulating the behavior of a system. A simulation system is the basis for the generation of a simulation application for a given simulator platform.

Simulation time group Group of platforms/devices in an experiment whose simulation times are synchronized with each other. If resynchronization is enabled, ControlDesk synchronizes a simulation time group as a whole, not the single members of the group individually.

Simulator A system that imitates the characteristics or behaviors of a selected physical or abstract system.

Single-processor system A system that is based on one dSPACE processor or controller board.

Single-shot instrument An instrument that displays an instantaneous value of a connected variable without keeping a value history. In ControlDesk, all instruments except for a plotter are single-shot instruments. For platforms that support the variable observer functionality, you can use single-shot instruments to observe variables.

Slave application An application assigned to the slave DSP of a controller or I/O board. It is usually loaded and started together with the real-time application running on the corresponding main board.

Slave DSP A DSP subsystem installed on a controller or I/O board. Its slave application can be loaded together with the real-time application or separately.

Slider An instrument (or a value cell type of the Variable Array) for displaying and setting the value of the connected variable by means of a slide.

Sound Controller An instrument for generating sounds to be played.

Standard axis An axis with data points that are deposited in the ECU memory. Unlike a common axis, a standard axis is specified within a curve or map. The parameters of a standard axis can be calibrated, which affects only the related curve or map.

Start trigger A trigger that is used, for example, to start a measurement raster. A platform trigger can be used as a start trigger.

Static Text An instrument for displaying explanations or inscriptions on the layout.

Steering Controller An instrument for changing variable values using a game controller device such as a joystick or a steering wheel.

Stimulation Writing signals to variables in real-time models during a simulation run.

Stop trigger A trigger that is used, for example, to stop a measurement raster.

String A text variable in ASCII format.

Strings are represented by the symbol.

Struct A variable with the struct data type. A struct contains a structured list of variables that can have various data types. In ControlDesk, a struct variable can contain either parameters and value blocks or measurement variables and measurement arrays. ControlDesk supports nested structs, i.e., structs that contain further structs and struct arrays as elements.

Structs are represented by the ▦ symbol.

Struct array An array of homogeneous struct variables.

Struct arrays are represented by the ▦ symbol.

STZ file A ZIP file containing signal descriptions in the STI format. The STZ file can also contain additional MAT files to describe numerical signal data.

Sub data set A data set that does not contain the complete set of the parameters of a platform/device.

Symbol A symbolic name of a physical address in a MAP file. A symbol can be associated to a variable in the Variable Editor, for example, to support an address updates.

System variable A type of variable that represents internal variables of the device or platform hardware and that can be used as measurement signals in ControlDesk to give feedback on the status of the related device or platform hardware. For example, an ECU's power supply status or the simulation state of a dSPACE board can be visualized via system variables.

T

Table Editor An instrument for displaying and setting values of a connected curve, map, value block, or axis in a 2-D, 3-D, and grid view. The Table Editor can also display the values of a measurement array.

The Table Editor can be used for the following variable types:

- Common axis (↑↑↑)
- Curve (⊞)
- Map (⊞)
- Measurement array (⊞)
- Value block (▥)

Time cursor A cursor which is visible at the same time position in the following instruments:

- In all Time Plotters
- In all XY Plotters
- In all bus monitoring lists

You can use the time cursor to view signal values at a specific point in time. If you move the time cursor, all measured signals and the respective parameters are updated. Instruments and bus monitoring lists display the values that are available at the selected time position.

Time Plotter A plotter instrument for displaying signals that are measured in a time-based raster (time plots).

Topology A description of the processor boards belonging to a multiprocessor system and their interconnections via Gigalinks. The topology also contains information on which Gigalink port of each processor board is connected to the Gigalink ports of other processor boards in the multiprocessor system.

Topology information is contained in the real-time application (PPC/x86/RTA) files of the multiprocessor system's processor boards.

TRC file A variable description file with information on the variables available in an environment model running on a dSPACE platform.

Trigger A condition for executing an action such as starting and stopping a measurement raster or a recorder.

The generic term for the following trigger types:

- Duration trigger
- Platform trigger
- Sample count trigger

Trigger condition A formula that specifies the condition of a trigger mathematically.

Triggered measurement The measurement of a measurement raster started by a platform trigger. The data flow between the dSPACE real-time hardware or VEOS and the host PC is not continuous.

U

Unassigned data set A data set that is assigned neither to the working page nor to the reference page of a platform/device. An unassigned data set can be defined as the new working or reference data set. It then replaces the “old“ working or reference data set and is written to the corresponding memory page, if one is available on the platform/device.

Unplugged A platform/device state defined by the following characteristics:

- The logical connection between ControlDesk and the hardware was interrupted, for example, because the ignition was turned off or the ControlDesk PC and the hardware were disconnected.
- Before the state of a platform/device changes to 'unplugged', the platform/device was in one of the following states:
  - 'Connected'
  - 'Online calibration started'
  - 'Measuring' / 'Recording'

> **Tip**
>
> A device for which the connection between ControlDesk and the device hardware currently is interrupted is also set to the 'unplugged' state when you start online calibration if both the following conditions are fulfilled:
> - The device's **Start unplugged** property is enabled.
> - The **Start online calibration behavior** property is set to 'Ignore differences'.
>
> This is possible for CCP and XCP devices. For details on the two properties listed above, refer to General Settings Properties (ControlDesk Platform Management).

- If the Automatic Reconnect feature is enabled for a platform/device and if the platform/device is in the 'unplugged' state, ControlDesk periodically tries to re-establish the logical connection for that platform/device.
- Online calibration is impossible. Offline calibration is possible.
- Platform/device configuration is possible.

The 'unplugged' platform/device state is indicated by the ⚠ icon.

Untriggered measurement The measurement of a measurement raster not started by a platform trigger. The data flow between the dSPACE real-time hardware or VEOS and the host PC is continuous.

User function An external function or program that is added to the ControlDesk user interface for quick and easy access during work with ControlDesk.

User Functions Output A controlbar that provides access to the output of external tools added to the **Automation** ribbon.

V

Value block A parameter that consists of a 1- or 2-dimensional array of scalar parameters.

In variable lists, ControlDesk displays entries for the value block itself and for each array element.

Value blocks are represented by the ▥ symbol.

Value conversion The conversion of the original *source values* of variables of an application running on an ECU or dSPACE real-time hardware into the corresponding scaled *converted values*.

Variable Any parameter or measurement variable defined in a variable description. ControlDesk provides various instruments to visualize variables.

Variable alias An alias name that lets the user control the property of a segment by a model parameter of a real-time application.

Variable Array An instrument for calibrating parameters and displaying measurement variable values.

The Variable Array can be used for the following variable types:

- Measurement (⊟)
- Measurement array (⊞)
- String (▭)
- Struct (▦)
- Struct array (▦)
- Value (P)
- Value block (◫)

Variable connection The connection of a variable to an instrument. Via the variable connection, data is exchanged between a variable and the instrument used to measure or calibrate the variable. In other words, variable connections are required to visualize variables in instrument.

Variable description A file describing the variables in a simulation application, which are available for measurement, calibration, and stimulation.

Variable Editor A tool for viewing, editing, and creating variable descriptions in the ASAM MCD-2MC (A2L) file format. The Variable Editor allows you to create A2L files from scratch, or to import existing A2L files for modification.

Variable Filter A variable filter contains the filter configuration of a combined filter, which is used to filter the variable list in the **Variables** controlbar using a combination of filter conditions.

Variables controlbar A controlbar that provides access to the variables of the currently open experiment.

V-ECU Abbreviation of *virtual ECU*.

ECU software that can be executed in a software-in-the-loop (SIL) testing environment such as a local PC or highly parallel in the cloud independently of real-time constraints and real ECU hardware.

Vehicle information The ODX database can contain information for one or more vehicles. Vehicle information data is used for vehicle identification purposes and for access to vehicles. It references the access paths (logical links) to the ECUs.

VEOS A simulator which is part of the PC and allows the user to run an offline simulation application (OSA) without relation to real time.

VEOS Player is the graphical user interface for VEOS.

VEOS platform A platform that configures and controls the offline simulation application (OSA) running in VEOS and that also provides access to the application's environment VPU.

VEOS Player An application running on the host PC for editing, configuring and controlling an offline simulation application (OSA) running in VEOS.

Verbal conversion A conversion in which a conversion table is used to specify the computation of numerical values into strings. The verbal conversion table is used when you switch the value representation from source to converted mode and vice versa.

Verbal conversion range A conversion in which a conversion table is used to specify the computation of a range of numerical values into strings. The verbal conversion range table is used when you switch the value representation from source to converted mode and vice versa.

View set A named configuration of the controlbars of ControlDesk. A view set has a default state and a current state that can differ from the default state. The configuration includes the geometry, visibility, and docking or floating state of controlbars.

Visualization The representation of variables in instruments:

- Measurement variables are visualized in instruments to view and analyze their time traces.
- Calibration parameters are visualized in instruments to change their values.

VPU Abbreviation of *virtual processing unit*. A VPU is part of an offline simulation application in VEOS. Each VPU runs in a separate process of the PC.

VPU is also the generic term for:

- V-ECUs
- Environment VPUs
- Controller VPUs
- Bus VPUs

W

Working data set The data set currently residing in the memory of a platform/device hardware. There can be only one working data set for each calibration platform/device. The working data set is read/write.

Working page Memory area containing the parameters of an ECU or prototyping hardware (memory page). The working page contains the read/write working data set.

If the platform/device also provides a reference page, ControlDesk lets you switch between both pages.

Writable measurement A scalar variable that can be measured and calibrated.

X

XCP Abbreviation of *Universal Measurement and Calibration Protocol*. A protocol that is implemented on electronic control units (ECUs) and provides access to ECUs with measurement and calibration systems (MCS) such as ControlDesk.

XCP is based on the *master-slave principle*:

- The ECU is the slave.
- The measurement and calibration system is the master.

The "X" stands for the physical layers for communication between the ECU and the MCS, such as CAN (Controller Area Network) and Ethernet.

The basic features of XCP are:

- ECU parameter calibration (CAL)
- Synchronous data acquisition (DAQ)
- Synchronous data stimulation (STIM), i.e., for bypassing
- ECU flash programming (PGM)

The XCP protocol was developed by ASAM e.V. (Association for Standardisation of Automation and Measuring Systems e.V.). For the protocol specification, refer to http://www.asam.net.

The following ControlDesk devices support ECUs with an integrated XCP service:

- XCP on CAN device
- XCP on Ethernet device

XCP on CAN device A device that provides access to an ECU with XCP connected to the ControlDesk PC via CAN. Using the XCP on CAN device, you can access the ECU for measurement and calibration purposes via XCP (*Universal Measurement and Calibration Protocol*).

XCP on Ethernet device A device that provides access to an ECU or V-ECU with XCP connected to the ControlDesk PC via Ethernet. The XCP on Ethernet device provides access to the ECU/V-ECU via XCP (*Universal Measurement and Calibration Protocol*) for measurement and calibration purposes.

XIL API EESPort Electrical Error Simulation port (EESPort)

XIL API MAPort platform A platform that provides access to a simulation platform via the ASAM XIL API implementation that is installed on your host PC.

XY Plotter A plotter instrument for displaying signals as functions of other signals.