
Vehicle Communication Software Suite (Powered by Akkodis) User Manual

2024-06-13



Contents

Welcome to the Vehicle Communication Software Suite User Manual.....	4
Before You Begin.....	5
What is the Vehicle Communication Software Suite?.....	6
Vehicle Communication Software Suite Requirements.....	9
New Features and Changes.....	10
Vehicle Communication Software Suite Overview.....	13
Getting Started.....	17
Installing NI-VCOM on Real-Time Targets.....	18
Software Features.....	20
Transmission Modes.....	20
End-to-End Communication Protection.....	25
Secure Onboard Communication.....	26
Signal Splitting.....	28
Network Management.....	28
Custom Device Features.....	29
Exporting and Importing Your Configuration.....	29
Comparing Database Files.....	30
Installing and Using the VCOM Custom Device API.....	31
Configuration Tool.....	32
Bus Monitor.....	32
Configuring NI-VCOM.....	34
Starting the WebUI.....	34
The Database Viewer.....	34
The Configuration Tool.....	35
Adding and Configuring NI-VCOM in VeriStand.....	50
Adding and Configuring NI-VCOM in LabVIEW.....	54
Configuring the Bus Monitor.....	55
Restbus Simulation.....	58
CAN-BUS.....	58
LIN-BUS.....	60
FlexRay.....	62
Automotive Ethernet.....	62

Additional Functions	63
When to Create a Restbus Simulation File	64
Using RBSConfig	65
Configuring a Restbus Simulation File	65
Generating a Configuration	68
Adding Autosignals to the Configuration Through ConfigStepsView	68
Manually Adjusting Parameters	72
Activation and Configuration of Network Management	72
Disabling Network Management and Diagnosing Messages	72
Vehicle Communication Toolkit Add-Ons	74
The Measurement and Calibration Toolkit.....	74
The Diagnostic Toolkit	79
Configuring the Diagnostic Toolkit.....	82
The PROVEtech:TA Log Window	83
Using the VCOM Helper Utility.....	84
Examples.....	85
NI-VCOM Examples.....	85
Configuration Tree Browsing for LabVIEW	87
CAN-FD Example.....	87
LIN Example.....	89
Automotive Ethernet Example	90
Diagnostic Toolkit and Measurement and Calibration Toolkit Examples	91

Welcome to the Vehicle Communication Software Suite User Manual

The Vehicle Communication Software Suite User Manual provides detailed descriptions of product functionality and step-by-step processes for use.

Looking for something else?

For information not found in the User Manual for your product, such as specifications and API reference, browse Related Information.

Related information:

- [NI Software and Driver Downloads](#)
- [NI License Setup and Activation](#)
- [NI Learning Center](#)
- [Vehicle Communication Toolkit Release Notes](#)
- [Vehicle Communication Software Suite Release Notes](#)
- [Vehicle Communication Measurement and Calibration Toolkit Release Notes](#)
- [Vehicle Communication Diagnostic Toolkit Release Notes](#)

Before You Begin

Know the basics and safety requirements for each component in your system before you perform any maintenance, test, or repair procedures. Visit ni.com/docs to review the safety documents that apply to your system.

There are a variety of solution integration options available for your application-specific system requirements. Depending on the configuration of your system and the level of customization by an integrator, some of the procedures and concepts described in this document may not be applicable to your solution. Consult your system integrator for any instructions not provided in this document.

The term ***integrator*** refers to the company or person responsible for customizing the system and software as delivered from NI.



Note Other product and company names listed are trademarks or trade names of their respective companies.

What is the Vehicle Communication Software Suite?

The Vehicle Communication Software Suite, powered by Akkodis, includes the Vehicle Communication Toolkit (NI-VCOM) as well as two optional add-ons: the Vehicle Communication Measurement and Calibration Toolkit and the Vehicle Communication Diagnostic Toolkit. NI-VCOM offers a Base option and a Full option. The add-ons work with the Full version of NI-VCOM to enable NI-XNET-supported devices in LabVIEW to handle restbus simulation for automotive networks and to provide NI-XNET device support for ECU design and validation. This includes but is not limited to reading and writing to internal ECU variables and characteristics through the XCP and CCP protocols. You can also execute diagnostic services, read diagnostic trouble codes, and flash ECUs with PDX files and an MCD-3D server.

Vehicle Communication Software Suite Key Features

The features that set the Vehicle Communication Software Suite apart:

Vehicle Communication Toolkit Key Features:

- Restbus simulation on NI-XNET hardware (CAN, LIN, Automotive Ethernet, FlexRay)
- Multiple restbus simulation transmission modes for `.dbc` and `.ldf` databases (cyclic, spontaneous, event, etc.)
- Signals to activate channel/node/message/PDU
- Automatic calculation of different cyclic redundancy check (CRC) and counter signals
- Manipulation of auto signals (counter, CRC, etc.)
- Automatic calculation of AUTOSAR End2End communication protection profile
- Network management
- Signal multiplexing (explicit and implicit)
- AUTOSAR multiple-PDU-to-container handling
- AUTOSAR secure onboard communication (SecOC)
- Message disassembly

- SOME/IP support (service discovery, SOME/IP services)

Vehicle Communication Measurement and Calibration Toolkit Key Features:

- XCP over CAN/CAN-FD/TCP/UDP/FlexRay

Vehicle Communication Diagnostic Toolkit Key Features:

- UDS on CAN/CAN-FD based on ISO-TP

Components of a Vehicle Communication Software Suite System

The Vehicle Communication Software Suite is designed for use in a test system that includes hardware, drivers, and optional add-ons.

Table 1. Components of a Vehicle Communication Software Suite System

Component Type	Component	Notes
Software	The Vehicle Communication Toolkit supports restbus simulation in VeriStand and LabVIEW.	For additional information, refer to Vehicle Communication Software Suite Overview .
Hardware	The Vehicle Communication Software Suite supports a wide range of hardware.	For additional information, refer to Vehicle Communication Software Suite Overview .
Optional Add-Ons	The Measurement and Calibration Toolkit The Diagnostic Toolkit	For additional information, refer to The Measurement and Calibration Toolkit and The Diagnostic Toolkit .

Related reference:

- [Vehicle Communication Software Suite Overview](#)

- [The Measurement and Calibration Toolkit](#)
- [The Diagnostic Toolkit](#)

Vehicle Communication Software Suite Requirements

Your system must meet the following minimum requirements to run and use the Vehicle Communication Toolkit, Diagnostic Toolkit, and Measurement and Calibration Toolkit.

- Disk Space—Minimum 3 GB
- Operating System

Table 2. Supported Operating Systems

OS	Vehicle Communication Toolkit	Diagnostic Toolkit	Measurement and Calibration Toolkit
Windows 10 (64-bit)	Supported	Supported	Supported
NI Linux® Real-Time (2024 Q2 system image)	Supported	Not Supported	Not Supported

New Features and Changes

Learn about updates—including new features and behavior changes—introduced in each version of the Vehicle Communication Toolkit, Diagnostic Toolkit, and Measurement and Calibration Toolkit.

Vehicle Communication Software Suite 2024 Q2 Changes

Vehicle Communication Software Suite 2024 Q2 includes only bug fixes. Refer to the Release Notes for a list of bug fixes.

Related information:

- [Vehicle Communication Toolkit Release Notes](#)
- [Vehicle Communication Software Suite Release Notes](#)
- [Vehicle Communication Measurement and Calibration Toolkit Release Notes](#)
- [Vehicle Communication Diagnostic Toolkit Release Notes](#)

Vehicle Communication Software Suite 2024 Q1 Changes

Learn about new features, behavior changes, and other updates in Vehicle Communication Software Suite 2024 Q1.

Vehicle Communication Toolkit New Features

- Base and Full Versions:
 - Upgrades supported version of LabVIEW to 2024 Q1
 - Upgrades supported version of VeriStand to 2024 Q1
 - Adds support for a Bus monitor
 - Adds support of byte array data type signals in the NI-VCOM custom device and LabVIEW API

Behavior Changes

- NI-VCOM has a new database viewer for RBS, the Calibration Toolkit, and the Diagnostic Toolkit. The new database viewer allows a more hierarchical and dense view of database content and provides a smoother search experience.
- The Read Signal - State RBS VI in the NI-VCOM LabVIEW API is now named Read (State VCOM).vi. Update the name of the VI if you use this API in your NI-VCOM application.

Vehicle Communication Software Suite 2023 Q3 Changes

Learn about new features, behavior changes, and other updates in Vehicle Communication Software Suite 2023 Q3.

Vehicle Communication Toolkit New Features

- Base and Full Versions:
 - Adds support for J1939 protocol
 - Adds support for CAN and LIN virtual devices
 - Adds support for multiple VCOM custom devices, each with unique configurations, in the same VeriStand project containing multiple controllers. Each controller can be associated with one VCOM custom device.
 - Adds support for LabVIEW API signal selection in the Read and Write VIs.
 - Adds support for deploying configuration files to remote targets from the WebUI rather than manually copying files between machines.
 - Adds performance improvements for importing large `.arxml` files and large numbers of signals.
- Full Version Only:
 - Adds support for optional Diagnostic Toolkit
 - Adds support for AUTOSAR Global Time Synchronization

Behavior Changes

- The VeriStand Custom Device is now called NI:VCOM rather than Provetech:RBS
- The VeriStand Custom Device API is now located on the NI-VCOM:Restbus simulation palette
- The Measurement and Calibration Toolkit is now configured automatically.

- The VCOM custom device now creates channels that are not scalable by default, but you can set them to be scalable if necessary.

Vehicle Communication Toolkit 2023 Q1 Changes

Learn about new features, behavior changes, and other updates in Vehicle Communication Toolkit 2023 Q1.

New Features

- Base and Full Versions:
 - Upgrades supported version of NI-XNET SDK to 2022 Q3
 - Adds support for configuring `CRC_11BIT_ADD` checksum type in RBSSConfig tool for CAN bus
 - Adds support for checking CRC value correctness upon receiving CAN/CAN-FD, LIN, FlexRay, and Automotive Ethernet messages in disassembly module
 - Adds support for checking Secured-PDU security algorithm correctness upon receiving CAN/CAN-FD and Automotive Ethernet messages in disassembly module
 - Extends support for AUTOSAR E2E Profile 4 for CAN/CAN-FD messages
- Full Version Only:
 - Adds support for optional Measurement and Calibration Toolkit
 - Adds support for setting the VLAN Priority field in the Ethernet header
 - Adds support for configuring different port numbers for Automotive Ethernet sender and receiver
 - Adds support for AUTOSAR E2E Profile 4m and 7m for SOME/IP fields in Automotive Ethernet module

Behavior Changes

- Older versions of the NI-VCOM Toolkit Custom Device for VeriStand may not work properly with Vehicle Communication Toolkit 2023 Q1. Delete the Custom Device from your VeriStand project and add a new Custom Device to resolve this issue.

Vehicle Communication Software Suite

Overview

The Vehicle Communication Software Suite, powered by Akkodis, includes the Vehicle Communication Toolkit as well as two optional add-ons: the Vehicle Communication Measurement and Calibration Toolkit and the Vehicle Communication Diagnostic Toolkit.

The Vehicle Communication Toolkit provides a LabVIEW API compatible with Windows and NI Linux Real-Time operating systems, a C/C++ API compatible with Windows, and a custom device for use with VeriStand. You can use the Vehicle Communication Toolkit with communication database files, such as AUTOSAR XML, to send and receive CAN, LIN, FlexRay, and/or automotive Ethernet signals. The signals send to and from NI-XNET-supported PXI, PCI, CompactDAQ, and CompactRIO instruments. The NI-VCOM Toolkit incorporates intellectual properties and products from Akkodis, such as PROVEtech Tool Suite.

The Diagnostic Toolkit and the Measurement and Calibration Toolkit are optional add-ons to the Vehicle Communication Toolkit that expand your ability to communicate with real or simulated ECUs. The Diagnostic Toolkit gives you symbolic access to the ECU diagnostic data and services that are described in ODX description files and PDX files, while the Measurement and Calibration Toolkit enables you to measure or calibrate an ECU through the CCP/XCP protocol by reading from and writing to the internal ECU variables and characteristics represented by ASAM (.A2L) database files.

The Vehicle Communication Toolkit offers Base and Full licensing options. You must have a Full license to use the Diagnostic Toolkit or the Measurement and Calibration Toolkit. The following table lists a comparison of features included in each version of the Vehicle Communication Toolkit as well as the Diagnostic Toolkit and the Measurement and Calibration Toolkit:

Table 3. Vehicle Communication Software Suite Feature Matrix

Features	NI-VCOM (Base)	NI-VCOM (Full)	Diagnostic Toolkit	Measurement and Calibration Toolkit
C/C++ API (Windows only)	Included	Included	Partially Included—Contact NI	Partially Included—Contact NI
LabVIEW API (Windows and NI Linux RT)	Included	Included	Included (Windows only)	Included (Windows only)
VeriStand Custom Device (Windows and NI Linux RT)	Included	Included	—	—
Web UI Database Viewer	Included	Included	Included (.pdx only)	Included (.a21 only)
Web UI Configurator	Included	Included	—	—
CAN and CAN-FD .dbc database support	Included	Included	—	—
LIN .ldf database support	Included	Included	—	—
CAN and CAN-FD .arxml database support	—	Included	—	—
LIN .arxml database support	—	Included	—	—
Automotive Ethernet .arxml database support	—	Included	—	—
FlexRay .arxml database support	—	Included	—	—
CCP over CAN	—	—	—	Included
XCP over CAN/ CAN-FD	—	—	—	Included

Features	NI-VCOM (Base)	NI-VCOM (Full)	Diagnostic Toolkit	Measurement and Calibration Toolkit
XCP over TCP	—	—	—	Included
XCP over UDP	—	—	—	Included
XCP over FlexRay	—	—	—	Partially Included—Contact NI
UDS on CAN/CAN-FD	—	—	Included	—

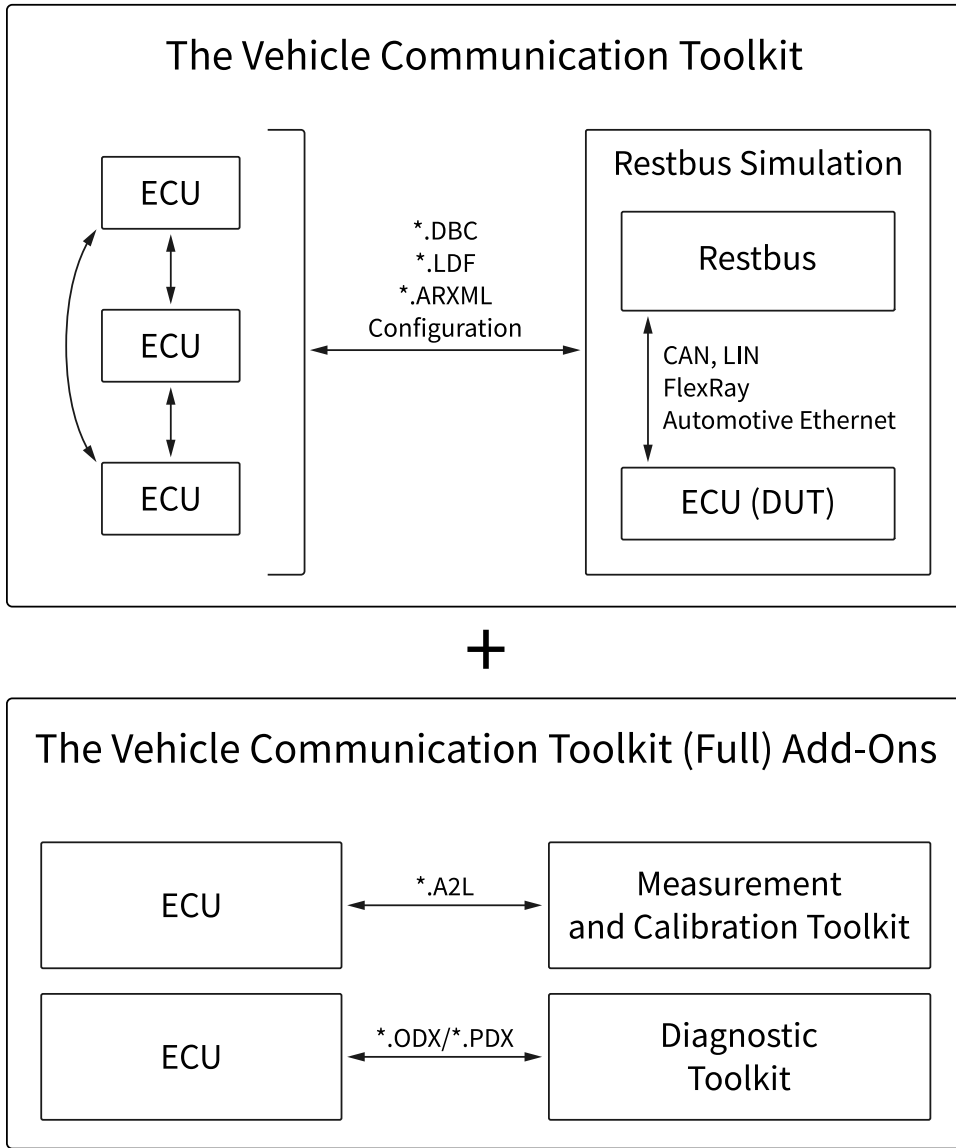
Supported Hardware and Automotive Databases

You can use NI-VCOM with communication database files, such as AUTOSAR XML, to send and receive CAN, LIN, FlexRay, or Automotive Ethernet signals. NI-VCOM supports sending signals to and from NI-XNET-supported PXI, PCI, CompactDAQ, and CompactRIO instruments. Not all the features provided by the NI-XNET API are included in the toolkit.

The Measurement and Calibration Toolkit is compatible with CAN and Automotive Ethernet signals. The Diagnostic Toolkit is compatible with CAN signals.

Vehicle Communication Software Suite Architecture

Figure 1. Vehicle Communication Software Suite Architecture



Getting Started

This section contains information about installing and activating the NI-VCOM Toolkit.

Software Installation and Activation

The Vehicle Communication Software Suite

Download the Vehicle Communication Software Suite installer from ni.com/downloads. NI software includes NI Package Manager to handle the installation.

Refer to the ***NI Package Manager Documentation*** at ni.com/r/nipmmanual for more information about installing, removing, and upgrading NI software using Package Manager.



Note If your system uses PROVEtech:RBS, you must uninstall it and install the Vehicle Communication Toolkit instead since PROVEtech:RBS is no longer supported. The NI-VCOM palette is similar to that of PROVEtech:RBS, with the exception of two sets of VIs that are specific to the type of Vehicle Communication Toolkit license you hold.

On Windows, NI-VCOM installs in the following locations:

- %Program Files%\National Instruments\LabVIEW\vi.lib\Vehicle Communication Toolkit
- %Public%\Documents\National Instruments\NI-VCOM

After installing your software, you will need to activate it. Activating your software verifies the licenses associated with your NI account, so you can start using your software. You have different options for activation, based on your software and how it was purchased. Please visit ni.com/activate for more information on activation methods.

The Vehicle Communication Diagnostic Toolkit

If you are using the Vehicle Communication Diagnostic Toolkit, you must install and activate version 9.03 of the Softing Smart Diagnostic Engine (SDE), which includes an MCD-3D server. Refer to the Softing Automotive website at automotive.softing.com for information about installing and activating the SDE.

Related information:

- [NI Software and Driver Downloads](#)
- [NI Package Manager Documentation](#)
- [NI License Setup and Activation](#)
- automotive.softing.com

Installing NI-VCOM on Real-Time Targets

1. Open NI Measurement and Automation Explorer (MAX) and expand your real-time target.
2. Right-click the **Software** menu, and select **Add/Remove Software**.
3. Select the Add tab of the Real-Time Software Installation Wizard.
4. Select both **NI Vehicle Communication Toolkit Library** and **NI Vehicle Communication Toolkit WebUI** from the list.
5. Follow the Real-Time Software Installation Wizard prompts to install the software.
6. (Optional) To uninstall the toolkit from your real-time target:
 - a. Repeat steps 1 and 2.
 - b. Select the Remove tab of the Real-Time Software Installation Wizard.
 - c. Select both **NI Vehicle Communication Toolkit Library** and **NI Vehicle Communication Toolkit WebUI** from the list.
 - d. Follow the Real-Time Software Installation Wizard prompts to uninstall the software.

RT Target Troubleshooting

If you encounter network issues while using remote targets, try the following steps:

1. Check the quality and integrity of your network cables.

2. Contact your IT team, and check if your RT target is correctly configured on your company network.

Software Features

This section describes features of the NI-VCOM Toolkit, including transmission modes, end-to-end communication, secure onboard communication, signal splitting, and network management.

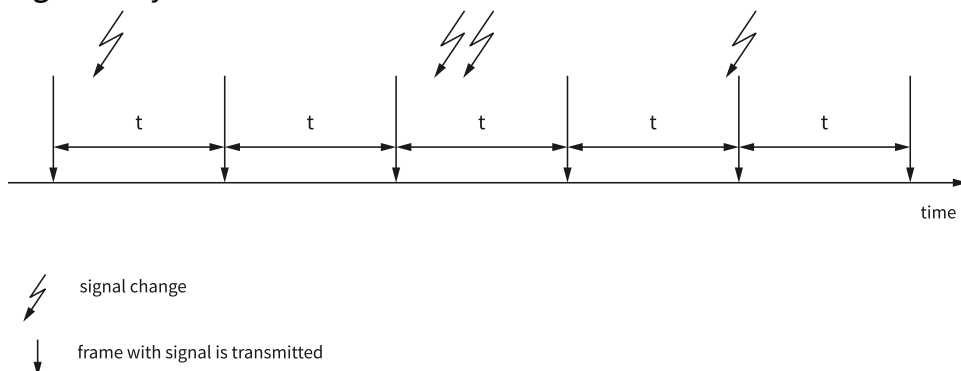
Transmission Modes

NI-VCOM restbus simulation supports different transmission modes. This is already defined in AUTOSAR files, but you must perform additional configuration using the RBS Config Tool if you are using `.dbc` or `.ldf` files.

The interaction layer converts the signal-based access of higher applications to the message-based transmission method of the bus system. Every message in a bus system is assigned a specific transmission mode, which is listed in the `.dbc` or AUTOSAR-XML file. The transmission mode defines the intervals and conditions the message transmits under. Messages can send with different transmission modes. NI-VCOM supports the following transmission modes:

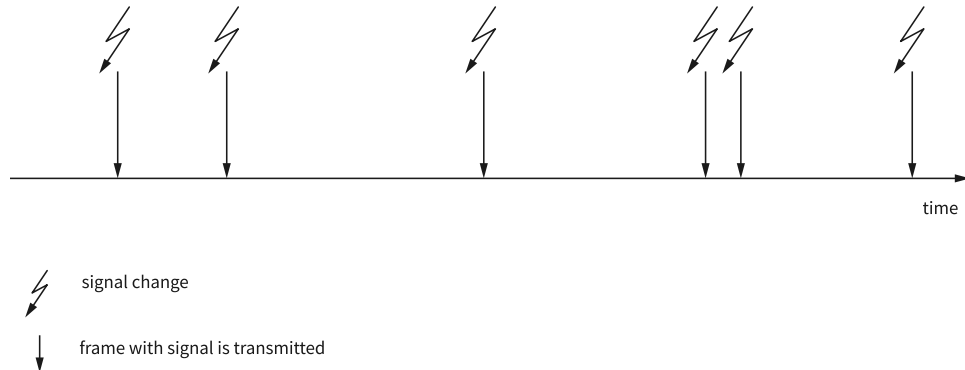
- **Cyclic**—The message sends periodically.

Figure 2. Cyclic Transmission Mode



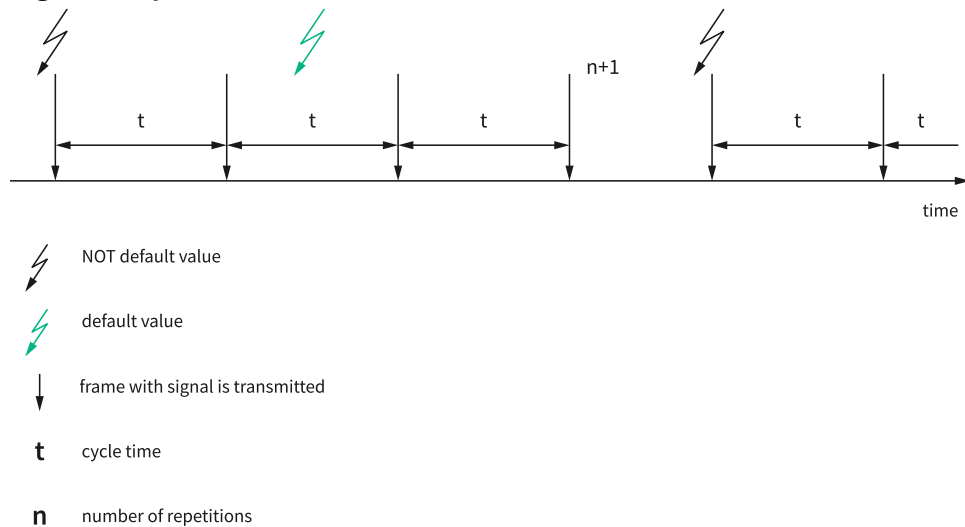
- **Spontaneous**—The message always sends when at least one of its signals changes its value.

Figure 3. Spontaneous Transmission Mode



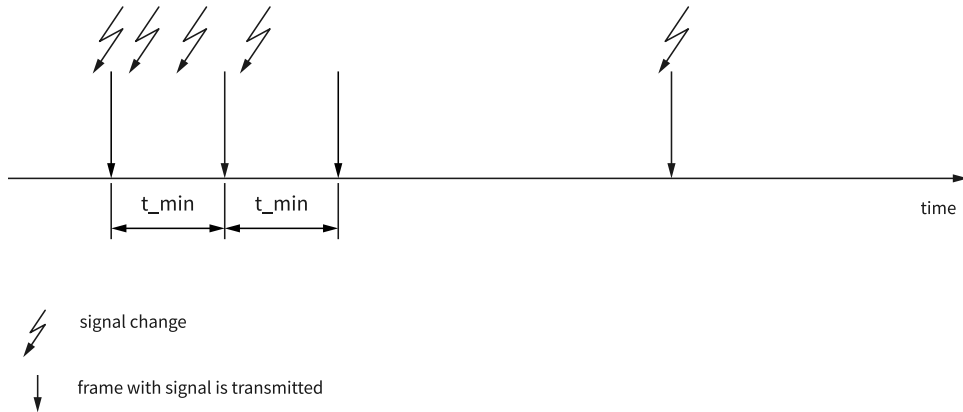
- **Cyclic if active**—The message sends periodically if at least one of its signals differs from its inactive value. It sends again $n+1$ times as soon as the last signal is inactive (stop messages). n is defined when all signals become default.

Figure 4. Cyclic if Active Transmission Mode



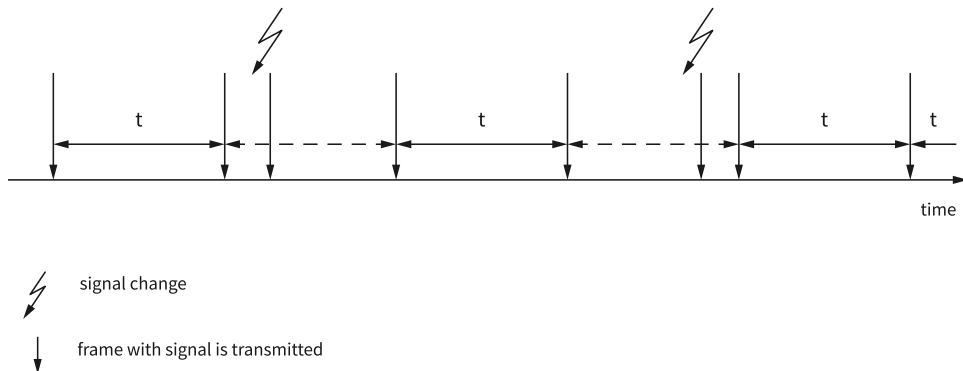
- **Spontaneous with minimum delay**—The message always sends under the following conditions:
 - At least one of its signals changes its value
 - The last transmission was sent at least the specified interval ago

Figure 5. Spontaneous with Minimum Delay Transmission Mode



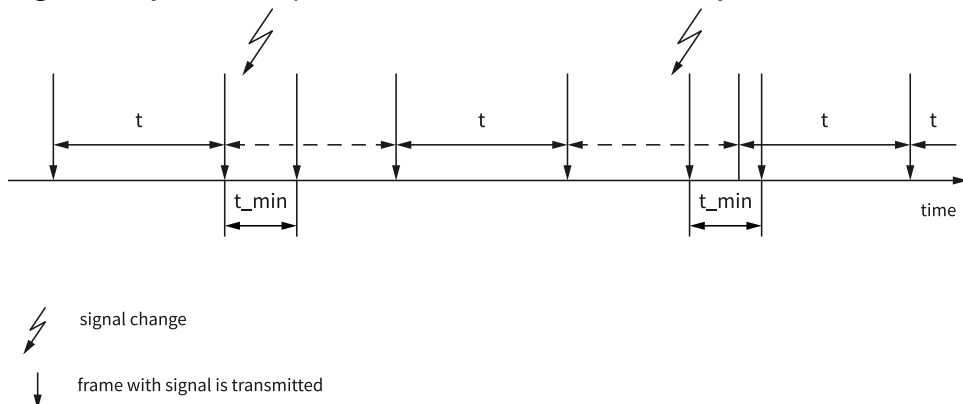
- **Cyclic and spontaneous**—This transmission type is a combination of the **cyclic** and **spontaneous** transmission types.

Figure 6. Cyclic and Spontaneous Transmission Mode



- **Cyclic and spontaneous with minimum delay**—In principle, this transmission mode is the same type as **cyclic and spontaneous** but with a minimum delay. Fulfilling the delay time has a higher priority than fulfilling the cycle time.

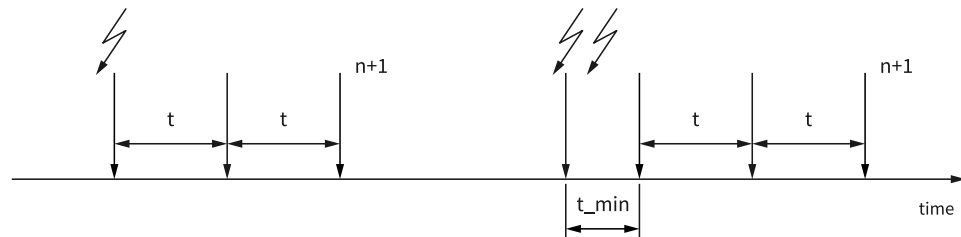
Figure 7. Cyclic and Spontaneous with Minimum Delay Transmission Mode





- **Spontaneous with repetition and minimum delay**—If a signal of a message with this transmission type changes, the message sends with a minimum delay of t_{min} . After that, the message repeats for n times with a cycle time of t . The

complete number of sent messages is $n+1$. Fulfilling the delay time has a higher priority than fulfilling the repetition cycle time.

Figure 8. Spontaneous with Repetition and Minimum Delay Transmission Mode



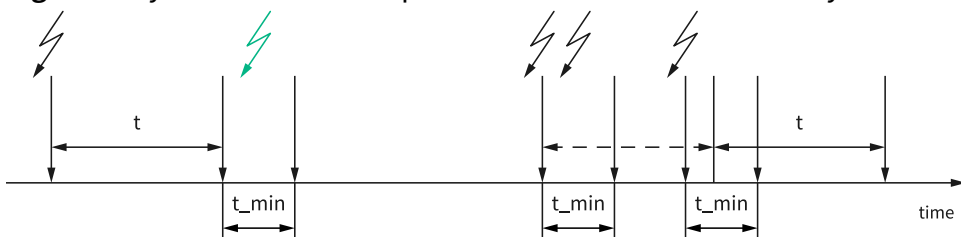
-  signal value change in application
-  frame with signal is transmitted
- t** cycle time




t_min minimum delay time

n number of repetitions

- **Cyclic if active and spontaneous with minimum delay**—As soon as at least one signal from this message is active, the message sends periodically. If a signal changes, the message sends immediately, taking into account the minimum interval. Fulfilling the delay time has a higher priority than fulfilling the cycle time.

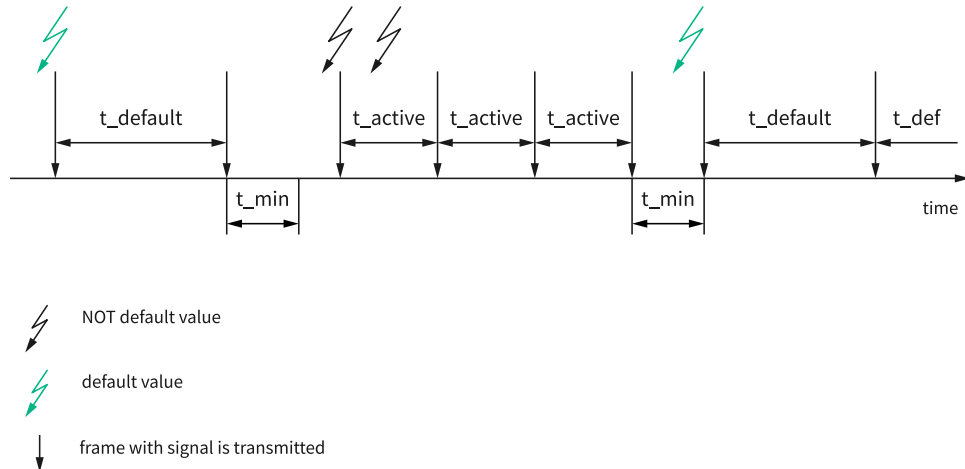
Figure 9. Cyclic if Active and Spontaneous with Minimum Delay Transmission Mode



-  signal changes NOT to default value
-  signal changes TO default value
-  frame with signal is transmitted

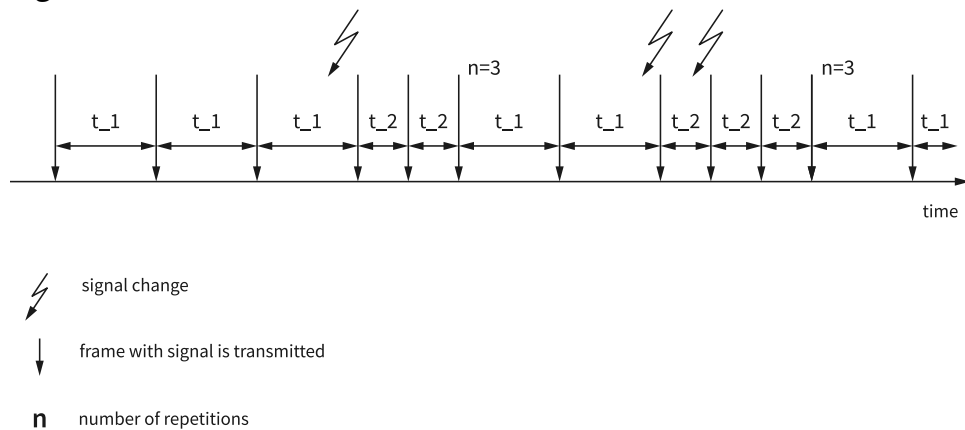
- **Cyclic if active fast**—The message sends periodically; however, there are two different cycle times. One cycle time is for the inactive state and one is for the active state (at least one signal must be active). The minimum delay time is taken into account while the transmission goes from the inactive to active state. The same is true when the transmission goes from the active to inactive state. Fulfilling the delay time has a higher priority than fulfilling the cycle time.

Figure 10. Cyclic if Active Fast Transmission Mode



- **Event and periodic**—The message repeats n times on any event. If no event exists, the message is sent by t_1 . If something changes, the message is sent by t_2 and repeated three times.

Figure 11. Event and Periodic Transmission Mode



The following table shows which values are relevant to the different types of transmission:

Table 4.

SendType	CycleTime	CycleTimeActive	DelayTime	NumOfRepetitions
Cyclic	Relevant	—	—	—
Spontaneous	—	—	—	—
SpontaneousWD	—	—	Relevant	—
CyclicAndSpontaneous	Relevant	—	—	—
CyclicAndSpontaneousWD	Relevant	—	Relevant	—
SpontaneousWithRepetitionWD	Relevant	—	Relevant	Relevant

SendType	CycleTime	CycleTimeActive	DelayTime	NumOfRepetitions
CyclicIfActiveAndSpontaneousWD	—	Relevant	Relevant	—
CyclicIfActive	—	Relevant	—	Relevant
CyclicIfActiveFast	Relevant	Relevant	Relevant	—
EventAndPeriodic	Relevant	Relevant	—	Relevant

End-to-End Communication Protection

The End-to-End (E2E) protection mechanism protects the safety-related data exchange at run time against the effects of faults within the communication link. By using E2E communication protection mechanisms, faults in the communication link can be detected and handled at run time.

AUTOSAR-Specific E2E Protection

AUTOSAR specific E2E information is present inside an `.arxml` file. NI-VCOM supports E2EP01, E2EP02, and E2EP05. On Automotive Ethernet bus systems, NI-VCOM also supports E2EP0M, E2EP04, E2EP04M, E2EP06, and E2EP07. The following table describes which bus systems support the different AUTOSAR E2E mechanisms:

Table 5. AUTOSAR E2E Supported Bus Systems

E2E Profile Type	NI-XNET CAN	NI-XNET LIN	Automotive Ethernet
E2EP01A	Supported	—	Supported
E2EP01B	Supported	—	Supported
E2EP01C	Supported	—	Supported
E2EP02	Supported	Supported	Supported
E2EP04	Supported	—	Supported
E2EP04M	—	—	Supported
E2EP05	Supported	Supported	Supported
E2EP06	—	—	Supported
E2EP07	—	—	Supported
E2EP07M	—	—	Supported

Checksum Protection

NI-VCOM supports additional checksum mechanisms. You can configure the checksum mechanisms using the RBSConfig tool. The following table describes which bus systems support the different checksums:

Table 6. Checksum Protection Supported Bus Systems

Cyclic Redundancy Check (CRC) Mechanism Type	NI-XNET CAN	NI-XNET LIN	NI-XNET FlexRay
J1850	Supported	Supported	Supported
J1850 User	Supported	Supported	Supported
Custom	Supported	Supported	Supported
AddAndComplementToOne	Supported	—	Supported
AddWithCarry	Supported	Supported	—
Honda (OEM Specific)	Supported	—	—
Mazda (OEM Specific)	Supported	—	—
Gac (OEM Specific)	Supported	—	—
11BitAdd (OEM specific)	Supported	—	—
XOR	Supported	Supported	Supported
XOR4Bit	Supported	Supported	—
CCITT	Supported	Supported	Supported

Secure Onboard Communication

Secure Onboard Communication provides a feature to verify the authenticity and freshness of PDUs. Only PDUs of type Secured are relevant for this feature.

NI-VCOM supports the generation of Authentication and Freshness Value, which is based on the AUTOSAR standards. NI-VCOM also supports an OEM-specific secured communication protocol. You must declare the OEM definition to make this feature work properly. Obtain the appropriate security information from the ECU manufacturer, and contact NI if you would like to define your own parameters. Refer to ***CAN Configuration*** for more information about how to configure this feature in NI-

VCOM.

Supported Bus Interfaces

CAN and Automotive Ethernet support Secure Onboard Communication.

Signals Related to Secure Onboard Communication

For each Secured PDU, the following two signals are created:

- **AuthInfo**—Provides the Message Authentication Code (MAC) for the respective Secured PDU.
- **Freshness**—Provides the Freshness Value at the time of generation of the Authentication code. The Freshness Value is generated by timestamps or individual Freshness counters.

The local tick-count of an ECU determines the Freshness value. As a result, two specific messages are implemented to synchronize tick-count for every ECU in a cluster.

Vehicle Security Master (VSM) sends the following messages to all other ECUs in a cluster, which all Automotive Ethernet and CAN instruments support through NI-VCOM:

- **Distributing the secured tick count**—VSM sends this message upon its start with a cycle time of 100 ms for a duration of one second. After one second, the secured tick-count message sends with the cycle time. The NI-VCOM Configuration Tool (using **SecTickCountCycleTime** tag) provides the cycle time as long as NI-VCOM is running. If no value for **SecTickCountCycleTime** is set, the cycle time remains 100 ms.
- **Distributing the real-time offset**—This contains the reference date and tick-count that ECUs use to calculate the current date and time. It is a cyclic message that VSM sends every 10 seconds.
- **Distributing the Vehicle Identification Number**—VSM sends this every 1,000 ms to distribute the vehicle identification number (VIN). To specify a VIN in the NI-VCOM Configuration Tool, you must use the **VehicleIdentificationNumber** tag. The VIN is a string of 17 characters.
- **Distributing the authentication broadcast information**—VSM sends this every 1,000 ms.

Related reference:

- [CAN Configuration](#)

Signal Splitting

NI-VCOM automatically splits signals defined in the bus description files referenced by the communication port. For example, all 64-bit signals can be split into 8- by 8-bit signals so that NI-VCOM can handle the original signal as an array of bytes. This allows you to work with all the partitions of the original signal independently.

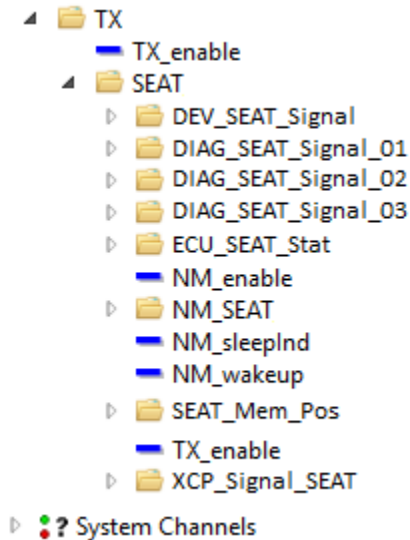
Network Management

Network management (NM) controls the transition of ECU between Bus Sleep Mode and Network Mode. This feature is implemented in CAN and Ethernet buses based on AUTOSAR specifications and is automatically configured for CAN networks with NM capability. To enable or disable this feature, you must specify the **ConfigureNM** tag for the port in the NI-VCOM Configuration Tool. By default, NM is enabled for CAN and disabled for Automotive Ethernet.

Three NM signals appear in the signal tree for each configured ECU:

- **NM_enable**—This signal is added for the Channel and ECU. When this signal is on, the ECU is enabled to send network management frames.
- **NM_sleepInd**—This signal triggers network release and prepares the ECU to go to Bus Sleep Mode. When this signal is on, the ECU does not send any network management frames.
- **NM_wakeUp**—This signal triggers network requests. When this signal is on, the ECU starts to send network management frames again.

Figure 12. Network Management Signals Available in the Signal Tree



Not all the ECUs in a CAN or Ethernet network have network management frames. For these ECUs, network management signals do not appear in the signal tree. When the `NM_sleepInd` signal is turned on, the ECU stops sending network management frames immediately. However, other frames send until the Bus Sleep timeout is reached, as indicated in Network Descriptor file.

To access more information about a CAN port state, set the **EnableBusState** parameter to `true`, which creates an additional folder with the same name as the hardware port as well as an additional signal, `ResetTransceiver`.

Custom Device Features

The following features are only available on the NI-VCOM Toolkit for VeriStand.

Exporting and Importing Your Configuration

You can deploy some projects on more than one test system. To facilitate the project deployment, an Export/Import function is available in the toolkit.

Configure your project.

1. Right-click **VCOM** in your system tree and select **Export Config**.

A new window appears.

2. Enter a name and a file path for the ZIP file you want to create.
3. Click **Export** to create the file.
4. Import the file into a new project by right-clicking **VCOM** and selecting **Import Config**.
5. Enter the file path.
6. Click **Import**.
7. Save your project.

Related reference:

- [Adding and Configuring NI-VCOM in VeriStand](#)

Comparing Database Files

Complete the following steps to compare the differences between two database files set one after the other one on the same project.

1. Update the **Network Descriptor** parameter in the NI-VCOM Configuration Tool to match the second database file name.
2. Save the configuration.

A new window is displayed that shows which channels must be updated. While importing the signals into the corresponding channels, the tool displays the following four different states for each signal:

- **Same:** This signal represents no difference between the previous and new database file.
 - **New:** This signal was not present in the previous database file and is new to the configuration.
 - **Different:** This signal has been updated in the new database file.
 - **Missing:** This signal has been removed from the new database file and will be removed from the configuration.
3. Review the signal states tally displayed at the bottom of the Add/Update Channel Selection window.



Note If you do not see the Add/Update Channel Selection window, select **Import Manually** in the Add/Update Sections window.

4. Select the desired signals for your project.
5. (Optional) Click **Export Report** to export this information in a text file.
6. Click **Done** to apply the changes.

Installing and Using the VCOM Custom Device API

The NI:VCOM Custom Device API allows you to programmatically create an instance of the add-on from a previously exported configuration. This API consists of:

- An example VI
- The NI-VCOM LabVIEW API
- A VI library

Default Installation Path

By default, the API installs in the following locations:

Table 7. Custom Device API Default Data Path for Windows

File Type	Custom Device API Data Path
Example file (.vi)	%Program Files (x86)%\National Instruments\LabVIEW\examples\NI-VCOM\NI-VCOM Custom Device API or %Program Files%\National Instruments\LabVIEW\examples\NI-VCOM\NI-VCOM Custom Device API
Function palette (.lvlibp)	%Program Files (x86)%\National Instruments\LabVIEW\vi.lib\NI-VCOM or %Program Files%\National Instruments\LabVIEW\vi.lib\NI-VCOM

How to Use the LabVIEW VCOM Custom Device API

Start the example from the default installation location or launch the example from the LabVIEW Example Finder.

Before running the example, set the input variable and paths to the projects. Based on

these variables, two example scenarios are available:

- The first scenario demonstrates how to add an NI:VCOM custom device to an existing project or replace the custom device in an existing project.
- The second scenario creates a completely new system definition file and adds the custom device into the `.nivssdf` file.

After you execute this example VI, the NI:VCOM custom device is added to the new or existing VeriStand project with all the configurations and dependencies. The VeriStand project is ready to use.

Configuration Tool

You can configure a variety of different function modules in NI-VCOM, such as restbus simulation (RBS) and disassembly. Configuration takes place through an XML file, which includes parameters such as:

- Path of the communication file
- Name of the NI MAX hardware port
- Parameters for the restbus simulation:
 - Baud Rate
 - Channel name
 - Specific functions to be activated

Use the Configuration Tool embedded in the WebUI to configure this XML file. This allows you to configure communication bus modules, such as CAN, FlexRay, LIN, and Automotive Ethernet, with their dedicated hardware interfaces. Refer to ***The Configuration Tool*** for more information.

Related concepts:

- [The Configuration Tool](#)

Bus Monitor

The Bus Monitor allows you to view, analyze, and log the CAN, LIN, FlexRay, and

Ethernet data that are transmitted or received in NI-VCOM.

When you enable the Bus Monitor feature, the NI-VCOM application streams the communication data to the WebUI bus monitor. Using the Bus Monitor allows you to do the following:

- Monitor the communication from the same machine or different machines that your NI-VCOM application (VeriStand, LabVIEW, etc.) runs.
- Display network data in either a most recent or historical data view.
- Assign the configuration file that is used in your NI-VCOM application to the Bus Monitor. By assigning a configuration file, you can identify more detailed frame information, identify decoded signal values, or plot the signal values in a graph.
- Stream all received network data to a file on the disk for offline data analysis.

Related tasks:

- [Configuring the Bus Monitor](#)

Configuring NI-VCOM

Configure NI-VCOM according to the requirements of your application.

Starting the WebUI

1. Launch the NI-VCOM WebUI.
 - To start the NI-VCOM WebUI from the desktop, double-click the desktop icon.
 - To start the NI-VCOM WebUI from LabVIEW, open a VI front panel and select **Tools » Vehicle Communication Toolkit » Vehicle Communication Toolkit WebUI**.
2. Enter your login information after the WebUI opens in your default browser. The default username is `admin` and the default password is `admin`.

Related concepts:

- [The Database Viewer](#)
- [The Configuration Tool](#)

The Database Viewer

Use the Database Viewer to select and view database files.

Viewing a Restbus Simulation Database File

Complete the following steps to view a restbus simulation database file:

1. Open the WebUI.
2. Click **Database » RestBus**.
3. Select a database file.
4. Click **View Database File** for a detailed view.
5. Select a bus cluster before selecting a specific cluster for that bus.



Note The information displayed depends on the content from your file and the bus type displayed. You see different information depending on whether you are looking at a CAN, LIN, FlexRay, or Automotive Ethernet cluster. This example procedure shows what you can find in an Automotive Ethernet cluster.

6. Choose one Virtual LAN (VLAN) under the **Cluster Channels** list.

Two new tabs appear:

- PDUs: Select a PDU name to view included signals. Select a signal to display its information on the right side of the window.
- ECUs: Select the ECU tab to view ECUs included in this part of the communication. Select an ECU name to display **TX** and **RX** headings, where all the PDUs that are transmitted (TX) and received (RX) are listed.

The Configuration Tool

Use the Configuration Tool to configure a variety of different function modules in NI-VCOM, such as restbus simulation (RBS) and disassembly.

Configuration takes place through an XML file, which includes parameters such as:

- Path of the communication database file
- Name of the NI MAX hardware port
- Parameters for the restbus simulation:
 - Baud rate
 - Channel name
 - Specific functions to be activated

Use the Configuration Tool embedded in the WebUI to configure this XML file. This allows you to configure communication bus modules, such as CAN, FlexRay, LIN, and Automotive Ethernet, with their dedicated hardware interfaces.



Note Some parameters are only available for specific XNET releases. NI recommends always keeping the NI-XNET drivers up to date.

General Settings

While running, NI-VCOM generates three different log files that report each warning or error to help you analyze operational issues:

- `Offline Core.log`
- `Online Core.log`
- `RBS.log`

The content of these log files depends on the value of the **Debug Level** set in the Configuration Tool.

- **Debug Level**—
 - 0: All
 - 1: Warnings and errors
 - 2: Only errors
- **Event Source**—A default event source is already available in the tool with a time period of 10,000 μ s. The `SysClock` event source contains the tag `TimerPeriod(μ s)` that you must set to the length, in microseconds, of one operation cycle. In a typical system, this value is set to 10,000 μ s, which ensures a good run time performance in most usage scenarios. Typically, only one event source exists for all modules/peers in one configuration file. Using more event sources for different modules/peers is only necessary if you face performance issues at the specified timer period, such as timer task overrunning at run time.



Note For all functional modules, the default event source is `SysClock` unless you modify it.

You can add a new event source by clicking + on the right side of **Event Source** under General Settings.

For every additional event source, the `SysClock` name is incremented. You can change this name.

Function Modules Configuration

Select a function module from the WebUI function modules list.

CAN Configuration

CAN configurations are available for NI-XNET CAN and CAN-FD instruments that NI MAX recognizes. You can configure more than one instrument.

You can modify the name of each instrument or add a specific **Clock Source** to this instrument if multiple are present in your configuration.

You can add one or more hardware ports to your configuration.

The Vehicle Communication Toolkit also supports virtual CAN on Windows and NI Linux RT, which enables you to simulate a CAN port without physical hardware. Configure a virtual CAN port in the same way as a physical CAN port, except that you must define the `<HWPort>` tag in the format `1, 1`, where the first number defines the port ID and the second number defines the bus ID. The port ID must be unique, but the bus ID can be shared by different ports. The default bus ID is `0`. Refer to the NI-VCOM Virtual Device Example for more information.

Configuration of the Peers

You can add up to two peers on a configured port. Configure peers under the hardware port. While generating the configuration XML file, the peer takes the port name of the hardware port that the peer is defined under.

The configurable parameters are highly dependent on the **Class** of the peer.

- **Name**—Arbitrary, but you cannot use the same name multiple times.
- **Class**—
 - `RBS`: Enables the sending part of the residual bus simulation.
 - `Disassembly`: Enables the receiving and disassembly part of the residual bus simulation.
- **ClockSource**—Selects your event source. The default is `SysClock`.
- **Change BaudRate (optional)**—Set to `true` to be able to change the baud rate. The default value is `false`. For more information, refer to ***Additional Functions***.
- **Change BaudRateFD (optional)**—Set to `true` to be able to change the data baud rate. The default value is `false`. For more information, refer to ***Additional Functions***.

- **Rename Signal TX_enable (optional)**—Renames the signal for the activation of the restbus simulation.
- **Rename Signal TX_cycletime (optional)**—Renames the signal for the specification of the cycle time (TX_cycletime exists only for cyclic messages).
- **Rename Signal TX_kickout (optional)**—Renames the signal for manual message sending.

The following signals are configured as **auto signals** (for more information, refer to **Configuring a Restbus Simulation File**).



Note **Active** signals (applies to CRC and ramp signals), **Control** signals (applies to CRC and ramp signals), and **Increment** signals (applies to ramp signals) are automatically added to the corresponding CRC and ramp signals. If **Active=0**, NI-VCOM automatically calculates the value of the auto signal and the value you provide in the **Increment** signal changes the increment of an automatically generated ramp signal. The default increment is 1. If **Active=1**, the value you provide in the **Control** signal overwrites the automatically generated signal.

- **Rename Signal Active (optional)**—Overrides default name of **Active** signal.
- **Rename Signal Control (optional)**—Overrides default name of **Control** signal.
- **Rename Signal Increment (optional)**—Overrides default name of **Increment** signal.
- **AutoSignal Toggle Bit Key (optional)**—Flags to configure a signal as a toggle bit signal in case the key value given with this flag is found in the signal name.
- **AutoSignal Parity Bit Key (optional)**—Flags to configure a signal as a parity bit signal in case the key value given with this flag is found in the signal name.

Initial load distribution is available to avoid TX buffer overflow. NI recommends turning this feature on for restbus simulations with FlexCard instruments over a CAN. Other options to reduce the chance of TX buffer overflow are to restbus simulate fewer ECUs, or to only turn on the relevant messages. The load distribution is implemented for classical message frames but not for Container PDUs.

- **Use Load Distribution (optional)**—The use of initial load distribution adds a delay to the cyclic messages to avoid TX buffer overflow. Possible values are `yes` or `no`. The default value is `no`.
- **EnableBusLoad**—Set to `true` to activate the bus load calculation. For more information, refer to **Additional Functions**.

Repeat these operations to add more hardware ports to the configuration.



Note You can add LIN, FlexRay, and Automotive Ethernet modules at the same time depending on your test system needs.

Configuration of the Secure Onboard Communication Parameters (Optional)

NI-VCOM retrieves this information from the ARXML file, but you can reconfigure it by adding the secure onboard communication (SecOC) parameters to your hardware port configuration.

You can configure the following SecOC parameters:

- **OEM**—Set to MB_00 or VW_00.



Note If you set the **OEM** parameter to VW_00, you must add and set **SOKKey** parameters for each specific Secured PDU. These parameters include the following:

- **DataID**—A 4-digit hex string value from the secured PDU.
 - **Key**—Hex string value of the key.
- **VehicleIdentificationNumber (optional)**—Distributes the vehicle identification number (VIN), which VSM sends every 1,000 ms. This parameter is a string of 17 characters.
 - **ProductionDate (optional)**—Distributes the real-time offset, which contains the reference date and tick-count that ECUs use to calculate the current date and time. This is a cyclic message that VSM sends every 10 seconds. To specify the **Reference** date, use this tag. This parameter is a string of 8 characters in the format YYYYMMDD.
 - **CarSharedSecret (optional)**—Defines a custom Car Shared Secret key for CAN and Automotive Ethernet. This parameter replaces the key with the user-provided value, which can be a 32-byte string or a 64-byte hexadecimal representation of it.
 - **SecTickCountCycleTime (optional)**—Distributes the secured tick-count, which VSM sends upon its start with a cycle time of 100 ms for a duration of one second. After one second, the secured tick-count message sends with the cycle time, which is provided in the configuration (using the `SecTickCountCycleTime` tag) as

long as the toolkit is running. If you do not set a value for this parameter, the cycle time remains 100 ms.

Related concepts:

- [CAN-BUS](#)
- [Network Management](#)
- [Additional Functions](#)
- [Configuring a Restbus Simulation File](#)

LIN Configuration

LIN configurations are available for LIN NI-XNET instruments that NI MAX recognizes. You can configure more than one instrument.

You can modify the name of each instrument and add a specific **Clock Source** to this instrument if multiple clock sources are present in your configuration.

You can add one or more hardware ports to your configuration.

The Vehicle Communication Toolkit also supports virtual LIN on Windows and NI Linux RT, which enables you to simulate a LIN port without physical hardware. Configure a virtual LIN port in the same way as a physical LIN port, except that you must define the `<HWPort>` tag in the format `1, 1`, where the first number defines the port ID and the second number defines the bus ID. The port ID must be unique, but the bus ID can be shared by different ports. The default bus ID is 0. Refer to the NI-VCOM Virtual Device Example for more information.

Configuration of the Hardware Port

You can specify the following values as channel properties for every LIN port:

- **Name**—For example, `LIN_Master`.
- **HW Port**—You must set the same hardware port name as the one displayed in NI MAX.
- **BitRate**—Bit rate defined in base unit bits per second (bps).
- **Mode**—Set to `Master` or `Slave`.
- **Protocol**—Set to `1.3`, `2.0`, or `2.1`.
- **Network Descriptor**—Enter the file path of the `.ldf` or `ARXML` file to be used. For

more information, refer to **LIN-BUS**.

- **RBS Descriptor (optional)**—For the RBS configuration to be used. For more information, refer to **LIN-BUS**.
- **Same Network Descriptor on Host (optional)**—Set this value to `true` to reuse the path of **Network Descriptor** within PROVEtech:TA so that the NI-VCOM Toolkit and PROVEtech:TA use the same descriptor file. Only use this setting in combination with PROVEtech:TA 2015 or later; older versions do not support this setting.
- **OEM (optional)**—Customer-specific flag.
- **Language (optional)**—Sets the description of network elements in the signal tree of the toolkit in a preferred language (such as DE for German or EN for English).
- **EnableBusTermination (optional)**—Enables or disables the terminating resistor through software. The NI-XNET instrument must support this function. The default value is `false`.
- **EnableDiagnosticOnlyMode (optional)**—An optional Boolean flag for the master node to enable diagnostic only mode, in which only the diagnostic schedules, rather than normal communication schedules, are executed. The default is the diagnostic interleaved mode. Do not switch to any diagnostic schedule table for diagnostics. The NI-VCOM LIN master simulation automatically handles the schedule table for diagnostics.
- **Cluster Name (optional)**—Name of the cluster from an AUTOSAR file you want to simulate. If you do not configure a cluster name, NI-VCOM uses the first cluster found in the AUTOSAR XML File.
- **Channel Name (optional)**—Name of the channel from an AUTOSAR file you want to simulate. If you do not configure a channel name, NI-VCOM uses the first channel found in the AUTOSAR XML File.

There are two different ways a LIN master can schedule diagnostic frames. In diagnostic only mode, it abandons the current schedule table while doing diagnostics. It first sends all the necessary master request messages and then requests all necessary slave response messages. Then it switches back to the currently set schedule table. In interleaved mode, the LIN master still schedules the current schedule table while doing diagnostics. However, every time it finishes the current schedule table, it inserts a diagnostic frame before restarting the schedule table. First, it sends all master request messages and then requests all slave response messages. Then it returns to normal scheduling.

Configuration of the Peers

You can add up to two peers on a configured port. Configure peers under the hardware port. While generating the configuration XML file, the peer takes the port name of the hardware port that the peer is defined under.

The configurable parameters are dependent on the **Class** of the peer:

- **Name**—Arbitrary, but you cannot use the same name multiple times.
- **Class**—
 - **RBS**: Enables the sending part of the residual bus simulation.
 - **Disassembly**: Enables the receiving and disassembly part of the residual bus simulation.
- **Clock Source**—Selects your event source.
- **Rename Signal TX_enable (optional)**—Renames the signal for the activation of the restbus simulation.
- **Rename Signal TX_cycletime (optional)**—Renames the signal for the specification of the cycle time (TX_cycletime exists only for cyclic messages).
- **Rename Signal TX_kickout (optional)**—Renames the signal for manual message sending.
- **Use Load Distribution (optional)**—The use of initial load distribution adds a delay to the cyclic messages to avoid TX buffer overflow. Possible values are `yes` or `no`. The default value is `no`.

The following signals are configured as **auto signals** (for more information, refer to **Configuring a Restbus Simulation File**).



Note **Active** signals (applies to CRC and ramp signals), **Control** signals (applies to CRC and ramp signals), and **Increment** signals (applies to ramp signals) are automatically added to the corresponding CRC and ramp signals. If **Active**=0, NI-VCOM automatically calculates the value of the auto signal and the value you provide in the **Increment** signal changes the increment of an automatically generated ramp signal. The default increment is 1. If **Active**=1, the value you provide in the **Control** signal overwrites the automatically generated signal.

- **Rename Signal Active (optional)**—Overrides default name of **Active** signal.

- **Rename Signal Control (optional)**—Overrides default name of ***Control*** signal.
- **Rename Signal Increment (optional)**—Overrides default name of ***Increment*** signal.

Related concepts:

- [LIN-BUS](#)
- [Configuring a Restbus Simulation File](#)

FlexRay Configuration

You can modify the name of each instrument or add a specific **Clock Source** to this instrument if multiple are present in your configuration.

You can add one or more hardware ports to your configuration.

Configuration of the Channel and Peers

After you configure the port, you can configure your desired channel and then add up to two peers on it. Configure peers under the hardware port. While generating the configuration XML file, the peer takes the name of the hardware port that the peer is defined under.

A cluster has up to two channels. You must map each `ChannelName` (from AUTOSAR) to a physical channel.

Next, you must set two parameters:

- **Channel Name**—For example, `CHAUIS_CHANNEL_A`
- **HW Channel**—For example, `A`.

You can specify `Disassembler` and `RBS` peers. The configurable parameters depend on the **Class** of the peer.

You can configure the following disassembly parameters:

- **Name**—Arbitrary, but you cannot use the same name multiple times.
- **Class**—
 - `RBS`: Enables the sending part of the residual bus simulation.

- **Disassembly**: Enables the receiving and disassembly part of the residual bus simulation.
- **ClockSource**—Selects your event source. The default is `SysClock`. For more information, refer to **The Configuration Tool**.
- **Channel**—For every restbus simulation or disassembly peer, you must specify the given FlexRay port and the corresponding FlexRay channel.

You can configure the following RBS parameters:

- **Name**—Arbitrary, but you cannot use the same name multiple times.
- **Class**—
 - **RBS**: Enables the sending part of the residual bus simulation.
 - **Disassembly**: Enables the receiving and disassembly part of the residual bus simulation.
- **ClockSource**—Selects your event source. The default is `SysClock`. For more information, refer to **The Configuration Tool**.
- **Channel**—For every restbus simulation or disassembly peer, you must specify the given FlexRay port and the corresponding FlexRay channel.
- **Rename Signal TX_enable (optional)**—Renames the signal for the activation of the restbus simulation.
- **Rename Signal TX_cycletime (optional)**—Renames the signal for the specification of the cycle time. (`TX_cycletime` exists only for cyclic messages).
- **Rename Signal TX_kickout (optional)**—Renames the signal for manual message sending.
- **Use Load Distribution (optional)**—The use of initial load distribution adds a delay to the cyclic messages to avoid TX buffer overflow. Possible values are `yes` or `no`. The default value is `no`.

The following signals are configured as **auto signals** (for more information, refer to **Configuring a Restbus Simulation File**).



Note **Active** signals (applies to CRC and ramp signals), **Control** signals (applies to CRC and ramp signals), and **Increment** signals (applies to ramp signals) are automatically added to the corresponding CRC and ramp signals. If **Active**=0, NI-VCOM automatically calculates the value of the auto signal and the value you provide in the **Increment** signal changes the increment of an automatically generated ramp signal. The default increment is 1. If

Active=1, the value you provide in the **Control** signal overwrites the automatically generated signal.

- **Rename Signal Active (optional)**—Overrides default name of **Active** signal.
- **Rename Signal Control (optional)**—Overrides default name of **Control** signal.
- **Rename Signal Increment (optional)**—Overrides default name of **Increment** signal.
- **AutoSignal Toggle Bit Key (optional)**—Flags to configure a signal as a toggle bit signal in case the key value given with this flag is found in the signal name.
- **AutoSignal Parity Bit Key (optional)**—Flags to configure a signal as a parity bit signal in case the key value given with this flag is found in the signal name.

You can add a cold start `ECU` or `Frame` to an RBS peer.

- **ECU (optional)**—Use this parameter if you are simulating a particular ECU, and it must contain the name of the ECU.
- **Frame (optional)**—Allows you to simulate the needed messages only instead of all messages of a particular ECU. When restbus simulating only the Rx-messages of one ECU, each message name needs to appear in this tag.



Note If an `ECU` is provided, all `Frame` entries are ignored.

Related concepts:

- [The Configuration Tool](#)
- [Additional Functions](#)
- [Configuring a Restbus Simulation File](#)

Automotive Ethernet Configuration

Automotive Ethernet configurations are available for NI-XNET instruments that NI MAX recognizes. You can configure more than one instrument. To configure Windows correctly, follow the steps described in **Automotive Ethernet**.

To start the configuration, add `Ethernet` as a function module. Next, add `Automotive Ethernet` under the `Ethernet` module. Choose `NI-XNET (TAP)` to configure the port in TAP Mode. For more information, refer to **Automotive**

Ethernet Tap Mode.

- **Name**—The tool sets this parameter.



Note Simulating Automotive Ethernet networks is only possible with the PXI Automotive Ethernet Interface Module (PXIe-8521/8522/8523) and allows deployment on Windows and NI Linux RT targets.

Configuration of the Hardware Port

You can specify the following values for every Automotive Ethernet port:

- **Name**—You can choose a name from your Ethernet bus.
- **Cluster Name**—Comes from the ARXML file.
- **Network Descriptor**—The AUTOSAR XML file you are using. You can also use this for signal disassembly.
- **Same Network Descriptor on Host (optional)**—Set this value to `true` to reuse the path of **Network Descriptor** within PROVEtech:TA so that the NI-VCOM Toolkit and PROVEtech:TA use the same descriptor file. Only use this setting in combination with PROVEtech:TA 2015 or later. Older versions do not support this setting.
- **Language (optional)**—Sets the description of network elements in the signal tree of the toolkit in a preferred language (such as DE for German or EN for English).
- **Enable Event Multicast (optional)**—Configures sending events on multicast or unicast nodes. The default is to send events to a multicast node.
- **ConfigureNM (optional)**—Activates network management if set to `true`. The default value is `false`.
- **DeviceName (optional)**—The hardware port name should be the same as the name displayed in NI MAX.
- **EnableHWTimeSynchronisation (optional)**—Enables the time synchronization protocol.
- **HWTimeSynchronisationBMCAEnabled (optional)**—Enables the Best Master Clock Algorithm (BMCA) of the time synchronization protocol.
- **HWTimeSynchronisationPortStateConfigured (optional)**—Provides the propagation delay for the Ethernet cable between the master clock and the slave neighboring clock.
- **HWTimeSynchronisationPropagationDelayThreshold (optional)**—For IEEE Std

802.1AS, if the Propagation Delay exceeds the threshold in this property, the protocol assumes that a switch or router that is not 802.1AS-capable exists between this clock and the neighboring 802.1AS-capable clock.

- **Triggering (optional)**—Activates the hardware synchronization between PXI-Cards if set to `Master` or `Slave`.
 - • **None**—Synchronization is not activated.
 - • **Master**—The configured port acts as a master.
 - • **Slave**—The configured port acts as a slave.

The following additional parameters are displayed if triggering is activated:

- **Trigger**—Choose a trigger line from 0 to 7.
- **TriggerDelay (optional)**—Future Timestamp (in seconds) for a Time Trigger Event.
- **AdjustLocalTime (optional)**—Phase adjustment (in nanoseconds) to align the timestamp with another device.



Note More information on both **Triggering** and **HWTimeSynchronisation** is available in the ***NI-XNET Hardware and Software Help*** on ni.com/docs.

You can add a channel to configured hardware ports. The channel configuration for Automotive Ethernet is the same as the VLAN configuration for Ethernet instruments. Configure the used channels, VLANs, and IP addresses in Windows for the existing network adapter. To configure Windows correctly, follow the steps described in ***Automotive Ethernet***.

You must use the same VLAN name for your channel as the one written in your ARXML database file. You can add ECUs to configured channels/VLANs.

You can configure the following ECU parameters:

- **Name**—You must use the same ECU name written in your ARXML file.
- **IsECUUnderTest (optional)**—Set this parameter to `true` to configure this ECU as a DUT. Setting this property to `true` only configures Automotive Ethernet services from the other simulated ECUs that are relevant to this DUT ECU. Additionally, this tag creates a DUT ECU node with Consumed Service Events and Remote Procedure Call (RPC) Methods inside in the NI-VCOM signal tree. This helps with monitoring if the Provided Service Events and RPC Methods from other simulated ECUs are sent

to DUT ECUs on time and disassembled properly.

- **IPAddress**— IP address of the ECU for the corresponding VLAN/channel.
- **SdIPMulticastAddress (optional)**—Parameter for the Service Discovery module. If used, you must set the multicast IP Address.
- **SdTransportProtocolPort (optional)**—Parameter for the Service Discovery module. If used, you must set the port used for the multicast.
- **UdpDynamicPort (optional)**—Parameter to set a range of UDP Ports. For example, 30491;32490.
- **TdpDynamicPort (optional)**—Parameter to set a range of TCP Ports.

Configuration of the Peers

After you configure the port, you must add the following three peers:

- **TX Peer:** Transmitting peer.
- **RX Peer:** Peer that disassembles the received messages.
- **AEthernet Peer:** Mandatory peer for Automotive Ethernet simulators.

Next, you must set the name of each peer.

- **Name**—Name of the peer.



Note The **AEthernet** peer name defines the beginning of each signal name that you find in your configuration.

You can set the following additional parameters for a **TX Peer**:

- **Rename Signal TX_enable (optional)**—Renames the signal to activate the restbus simulation.
- **Rename Signal TX_cycletime (optional)**—Renames the signal to specify the cycle time (`TX_cycletime` exists only for cyclic messages).
- **Rename Signal TX_kickout (optional)**—Renames the signal to manually send a message.
- **Use Load Distribution (optional)**—The use of initial load distribution adds a delay to the cyclic messages to avoid TX buffer overflow. Possible values are `yes` or `no`. The default value is `no`.
- The following signals are configured as **auto signals** (for more information, refer to **Configuring a Restbus Simulation File**).



Note *Active* signals (applies to CRC and ramp signals), *Control* signals (applies to CRC and ramp signals), and *Increment* signals (applies to ramp signals) are automatically added to the corresponding CRC and ramp signals. If **Active**=0, NI-VCOM automatically calculates the value of the auto signal and the value you provide in the *Increment* signal changes the increment of an automatically generated ramp signal. The default increment is 1. If **Active**=1, the value you provide in the *Control* signal overwrites the automatically generated signal.

- ◦ **Rename Signal Active (optional)**—Overrides default name of *Active* signal.
- ◦ **Rename Signal Control (optional)**—Overrides default name of *Control* signal.
- ◦ **Rename Signal Increment (optional)**—Overrides default name of *Increment* signal.

Configuration of a Service Discovery Delay

The following peer is optional, and you do not need to configure it to be able to simulate a residual bus simulation. The peer allows you to overwrite the Automotive Ethernet Service Discovery **OFFER-CYCLIC-DELAY** parameter given inside the network description file.

Complete the following steps to configure a service discovery delay:

1. Add `Service Discovery Config` under the **Hardware Port** parameter.
2. Set the **Offer_Cyclic_Delay** parameter on the right side of the window. Setting a value overwrites the **Offer-Cyclic-Delay** parameter given inside the network description file.

Related concepts:

- [Automotive Ethernet](#)

Automotive Ethernet Tap Mode

The NI-VCOM Toolkit supports XNET Tap Mode for the XNET Automotive Ethernet

cards. Review the XNET documentation and become familiar with the corresponding settings in NI MAX before you set up a Tap Mode scenario.

Ensure you have the following hardware and software before beginning:

- Windows or NI Linux system with XNET Automotive Ethernet card
- XNET version 21.5 or later

Configure the settings and wiring of your hardware in NI MAX.

1. Add `Ethernet` in **Function Modules**.
2. Add `NI-XNET` to `Ethernet`.
3. (Optional) Update **Name** after you add `NI-XNET`.
4. Add a **Hardware Port** to `NI-XNET`.
5. At a minimum, specify the **Name**, **Cluster Name**, **Network Descriptor**, and **DeviceName** for the hardware port.
6. Add a **Channel** to the hardware port and specify **Name**.
7. Add an **RX Peer** and specify its **Name**.
8. Add a second **AEthernet Peer** and specify its **Name**.

To get started, NI recommends viewing the Tap Mode Example Project, which is included in the VeriStand installation package.

Adding and Configuring NI-VCOM in VeriStand

Adding the Custom Device

After you install the toolkit on your computer, you can add the toolkit to an existing project or create a new one. If you want to add the NI-VCOM Toolkit to an existing project, ignore information about creating a project in VeriStand.

The VeriStand project window displays all recently launched projects. It allows you to import a project by clicking **Browse**. You can also create a new project.

Create a new project in VeriStand using one of the following methods:

- Press `<CTRL+N>`.
- Click **New NI VeriStand Project**.

- Go to the **File** menu and click **New Project**.

You can set a project name, author, and description.

After you open your project, complete the following steps to edit its system definition file (.nivssdf):

1. Click **Configure** in the VeriStand Editor window to open the System Explorer.
2. Click **Controller** in the System Explorer tree.
3. Change the **Target Rate** value to 1,000 Hz. You may need to update the **Timing Source Timeout** value because the project contains thousands of signals.
4. Right-click **Custom Device** in the System Explorer tree and select **NI...VCOM**. This adds the VCOM custom device.
5. Refer to the instructions on the VCOM custom device main page to configure the VCOM custom device.

Custom Device Main Interface

You can select the following system signals from your restbus simulation under **Status** in the System Explorer tree:

- **State:** Displays the state of the custom device: Idle, Starting, Running, Stopping, or Error.
- **Connect:** Allows the custom device to start if Connect on Deploy is not activated.
- **Error Code:** Displays error code.
- **Execution Time:** Execution time of the custom device, when connected.
- **Execution Time Mean:** Base 8 floating average of Execution Time.
- **Overrun Count:** Number of overruns since the restbus simulation started.
- **RE_ReadSignals:** Time to write data from RBS Core to the custom device.
- **RE_StepRead:** Time of receive processing.
- **RE_StepWrite:** Time of transmit processing.
- **RE_WriteSignals:** Time to write data from the custom device to RBS Core.
- **VS_ReadSignals:** Time to acquire data from VeriStand engine to the custom device.
- **VS_WriteSignals:** Time to provide data from the custom device to the VeriStand Engine.

Custom Device Channel Structure in the System Explorer Tree

If your project uses the same configuration file as the NI CAN-FD example project, then all the channels will become folders after you close the configuration tool.

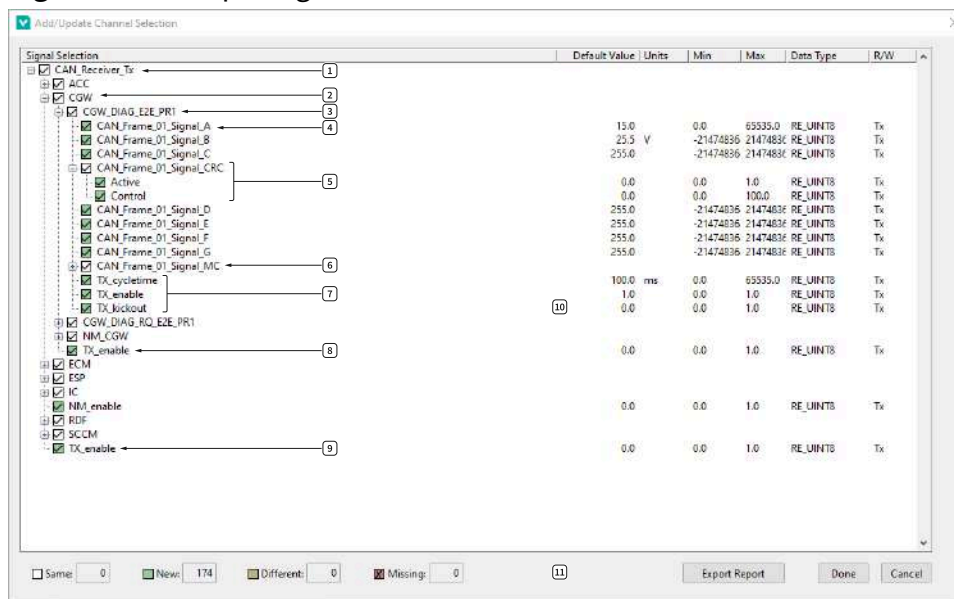
- **CAN_Receiver_Rx:** Folder **Rx** from a CAN channel. This folder displays all the signals received by the node **CAN_Receiver**.
- **CAN_Receiver_Tx:** Folder **Tx** from a CAN channel. This folder displays all the signals transmitted by the node **CAN_Receiver**.
- **CAN_Sender_Rx:** Folder **Rx** from a CAN channel. This folder displays all the signals received by the node **CAN_Sender**.
- **CAN_Sender_Tx:** Folder **Tx** from a CAN channel. This folder displays all the signals transmitted by the node **CAN_Sender**.

Importing a Channel

When you click one of the VCOM folders in the System Explorer tree, a new window appears that allows you to import all of the signals from that folder by clicking **Import Channels**.

The following window appears. The folders in the signal tree will change depending on the peers you configure for each hardware port. You can update the names of some frequently occurring signals to make them easier to locate.

Figure 13. Example Signal Tree



1. CAN_Receiver_Tx: Name of the active channel. Clicking the tick box selects all the signals of the channel.
2. CGW: Name of one of the nodes included in your CAN Network file.
3. CGW_DIAG_E2E_PR1: Name of the frame. This frame is included in the node CGW.
4. CAN_Frame_01_Signal_A: Name of the signal. This signal is included in the frame CGW_DIAG_E2E_PR1. When you start the restbus simulation, its value can dynamically change.
5. CAN_Frame_01_Signal_CRC: Name of a checksum signal included in the frame CAN_FRAME_01. The restbus simulation automatically calculates its value. A CRC is composed of two other parameters:
 - Active: Boolean value in which you can control NI:VCOM. If set to 0, the value of the signal automatically calculates. If set to 1, the value of the signal is set under Control.
 - Control: Integer value that you can set. If you set Active to 1, the signal gets this value.
6. CAN_Frame_01_Signal_MC: Name of a sequence counter signal included in the frame CAN_Frame_01. The restbus simulation automatically calculates its value. A sequence counter is composed of three other parameters:
 - Active: Boolean value in which you can control NI:VCOM. If set to 0, the value of the signal automatically calculates. If set to 1, the value of the signal is set under Control.
 - Control: Integer value that you can set. If you set Active to 1, the sequence counter gets this value.
 - Increment: Integer value that defines the counting step of the counter. It can change dynamically.
7. Parameters of a CAN frame:
 - TX_cycletime: Signal that changes the cycle time of a message in milliseconds. This signal exists only for cyclic messages.
 - TX_kickout: Boolean signal that allows sending a spontaneous frame. If set to 0, the frame does not send. If set to 1, the frame sends.
 - TX_enable (on a frame level): Boolean signal that allows the transmission of the frame. If set to 0, the frame does not send. If set to 1, the frame sends with the defined cycle time.
8. TX_enable (on a node level): Boolean signal that allows transmission on a node level. If set to 0, none of the frame of the node sends. If set to 1, the CAN channel frame sends if you set the frame-level TX_enable to 1.
9. TX_enable (on a channel level): Boolean signal that allows transmission on a

channel level. If set to 0, none of the frame of the CAN channel sends. If set to 1, the CAN channel frame sends if you set the frame-level and node-level TX_enable to 1.

10. General information regarding the signals.
11. The custom device compares the new signal list with a previously configured one if it exists. It is then possible to view the differences between two versions of a database file (for more information, refer to ***Comparing Database Files***).

Setting Your Target and Running the Project

Select **Controller** in the System Explorer tree and specify the **Operating System** and **IP Address** parameters for your target.



Note To deploy the project on a different target, update the **Operating System** and **IP Address** parameters.

Save your project, then close the System Explorer window and deploy your project in VeriStand.

Related concepts:

- [Using RBSSConfig](#)
- [Configuration Tool](#)

Related tasks:

- [Comparing Database Files](#)

Adding and Configuring NI-VCOM in LabVIEW

NI-VCOM API

The NI-VCOM API provides a signals single-point mode. Using this mode, the signals update with every call of the Write VI. The signals then continue maintaining the cycle time specified in the database or TX_cycletime channel. Event triggered signals send with every value change or by toggling the TX_kickout channel from 0 to 1. To get a message on the wire, you must set all TX_enable channels from the desired node up its

hierarchy (PDU » Service » ECU » Bus) to 1. You must use either the Write VI or the Enable VI.

RBSConfig and the Configuration Tool

Two legacy tools are installed with the NI-VCOM Toolkit API:

- **Configuration Tool**—Create and edit RBS configurations (select NI-XNET interfaces and databases to use). You can also use the WebUI to complete this configuration. For more information, refer to ***Configuration Tool***.
- **LDF/DBC RBS Descriptor Tools**—For LIN `.ldf` and CAN `.dbc` databases, you must configure mapping certain signal endings to checksum and counter operations. AUTOSAR `.arxml` databases do not need this additional configuration. For more information, refer to ***Using RBSConfig***.

Related concepts:

- [Configuration Tool](#)
- [Using RBSConfig](#)

Configuring the Bus Monitor

1. Navigate to the Configuration Tool within NI-VCOM.
2. Select **Enable** from the Bus Monitor drop-down menu.
3. Save the Bus Monitor configuration file.
4. Deploy the Bus Monitor configuration file to your target.
5. Run your NI-VCOM application with the Bus Monitor configuration file through VeriStand or LabVIEW.


Related concepts:

- [The Configuration Tool](#)

Using the Bus Monitor to View Communication Data

Before you begin, ensure the Bus Monitor is enabled in your NI-VCOM application's configuration file.

1. Navigate to the Bus Monitor page within the NI-VCOM WebUI.
2. Specify the **Server Address** and **Server Port** for the Bus Monitor.
 - **Server Address**—the IP address of the machine running the NI-VCOM application (VeriStand, LabVIEW, etc.). The default server address is "localhost".
 - **Server Port**—the Ethernet port that the NI-VCOM application uses to establish the connection with the Bus Monitor tool. The default server port is "50012". The server port should not be occupied by other applications, so if the default port number is used by other applications, you must specify an available port number in the configuration file and in the server port. If the port number you specify in the server port doesn't match the port number in the configuration file, the Bus Monitor won't connect to the NI-VCOM application.
3. **Optional:** Specify the Bus Monitor configuration file if you want to view decoded signal values or plot signal values in a graph. Skip this step if you only need to view raw frames.
 - a. Click **Settings**.
 - b. Click **Change Configuration File**.
 - c. Select a configuration file and click **Load**.

The configuration file you choose appears in the Loaded Configuration File dialog. Each port and their respective network descriptors populate the Bus Monitor Settings table. If the tool cannot load a network descriptor, an error () appears in the table.
4. Select the protocol you want to monitor.

NI-VCOM supports simulating more than one protocol in the same application, and the Bus Monitor can monitor all the protocols simulated in a NI-VCOM application. However, you can only view one type at a time, so you need to select the protocol you want to monitor.
5. Click the following tabs to monitor results.
 - **Monitor**—displays the most recent network frame data. Each row in the Monitor table shows the frames with the same port and direction.
 - **ID Logger**—displays historical network frame data in order of appearance.
 - **Graph**—displays a graph that monitors all signals. Click + to add a monitoring signal and - to remove a monitoring signal.



Note The Ethernet protocol does not have an ID Logger tab; its Monitor tab functions like the ID Logger tab functions for other protocols.

6. **Optional:** Click **Filter** to filter the Monitor, ID Logger, or Graph tables.
7. Click **Stop** to stop the monitor.

After you stop the monitor, you can expand the resultant Monitor and ID Logger table rows to show the signals of each frame. Click **Log to Disk** to store all received frames in a plain text file.

Related tasks:

- [Starting the WebUI](#)

Restbus Simulation

Restbus simulation enables testing simulated and real ECU nodes in a CAN, LIN, FlexRay, or Automotive Ethernet network. The simulation range varies from an individual network composing a specific bus system to an entire car network comprising several bus systems.

CAN-BUS

CAN RBS

The CAN RBS Module provides a complete CAN restbus simulation. NI-VCOM supports CAN instruments that have their own time management and instruments that depend on software controlled timing provided by NI-VCOM. The first category has a more precise timing for the restbus simulation. Additionally, it supports the simulation of CAN network management (NM).

Complete a CAN RBS based on the information provided in such an AUTOSAR-XML or .dbc file, which is specified in the **Network Descriptor** tag of the NI-VCOM configuration.

When using a .dbc file, RBS descriptor XML files provide more RBS information. Create RBS descriptor XML files with the RBSConfig tool (For more information, refer to **Using RBSConfig**).

Configure the following signals using the RBSConfig tool:

- Autosignals: Automatically calculated signal values. To configure autosignals using RBSConfig tool, refer to **Configuring a Restbus Simulation File**. CANs support the following autosignals:
 - Ramp (multiple bits): The bits contain a message counter.
 - Checksum (multiple bits):
 - ChecksumJ1850 (8 Bit)

- ChecksumJ1850 with User Bytes (8 Bit)
- Custom Check sum CRC-8
- ChecksumXOR (8 Bit)
- ChecksumXOR4Bit (4 Bit)
- ChecksumAddWithCarry (8 Bit)
- ChecksumAddAndComplementToOne (8 Bit)
- Parity (1 bit)
- Toggle (1 bit)
- Signal TX_enable: This signal can activate the RBS for message, ECU, and CAN channel.
- Signal TX_kickout: This signal can send a CAN message immediately.
- Signal TX_cycletime: This signal can change the cycle time of a message in milliseconds (ms).
- Signal NM_enable: This signal can activate the NM simulation for the ECU and CAN channel level.
- Signal NM_sleepInd: This signal can set the sleep indication bit.
- Signal NM_wakeUp: This signal can wake up the ECU.

It is possible to configure the NM capabilities in the RBSConfig tool. NM signals are only available if the RBSConfig tool enabled NM. Refer to **Transmission Modes** for information on the different transmission types signals can be sent with.



Note The RBSDescriptor file is not necessarily needed for the CAN RBS. For example, the RBSConfig tool does not support the new AUTOSAR-XML files with the Container-PDU concept (from the AUTOSAR Release v4.2.1).

CAN Message Disassembly with DBC and AUTOSAR XML File

The `.dbc` and AUTOSAR XML files contain the information for interpreting the raw data from CAN messages received (Rx from the perspective of NI-VCOM) for converting the signal names, values and units. These signal values are available in NI-VCOM for the user interface as **ReadOnly** signals. You can specify a file for disassembling every CAN port in the **Network Descriptor** tag. For more information, refer to **CAN Configuration**.



Note The disassembly of each function module (CAN, LIN, FlexRay, and

Automotive Ethernet) adds the following signals as a message level in the NI-VCOM signal tree:

- `RX_time`: The signal value corresponds to the current module software time, in seconds, which is configured through the **`clock_src`** event source.
- `RX_deltatime`: The signal value is calculated from the hardware time stamps, in seconds, of the last two consecutive messages on receive.

CAN-FD

CAN-FD allows you to increase the data bit rate by transferring up to 64 payload bytes in a single message. To activate this function, you must set the ***Data BitRate*** parameter in the Configuration Tool. For more information, refer to ***CAN Configuration***.

Related concepts:

- [Transmission Modes](#)
- [Using RBSConfig](#)
- [Configuring a Restbus Simulation File](#)

Related reference:

- [CAN Configuration](#)

LIN-BUS

LIN RBS

For a restbus simulation sending LIN messages, the LIN card controls the simulated control units. The use from bus interfaces, which have their own microcontroller, enable a PC to cyclically transmit messages independent of timing.

RBS descriptor XML files, created by the RBSSConfig tool, provide more RBS information when using a `.ldf` file (for more information, refer to **Using RBSSConfig**).

Configure the XML file for RBS with the RBSSConfig tool. Specify the XML file in the NI-VCOM configuration. The following functions are available:

- Simulating Master and/or Slaves
- Calculating values known as **autosignals** automatically for an RBS. Example autosignals can consist of the following elements:
 - Ramp (multiple bits): The bits contain a numerical value, such as the number of certain messages (Messagecounter).
 - ChecksumJ1850 (1 Byte)
 - ChecksumXOR (1 Byte)
 - Parity (1 bit)
 - Toggle (1 bit)

A LIN RBS with NI-VCOM also supports autosignals. With a passive card, such as NI-XNET, NI-VCOM must compute the autosignals. Because LIN slave responses send immediately upon a master request, NI-VCOM must prepare such responses before a master request comes. As a result, autosignals are not always correct if there are other dynamic values in the transmitted messages (typically sequence counters).

NI-VCOM can automatically detect whether an ARXML has CRC signals, which use AUTOSAR End-to-End Profile 1, Profile 2, or Profile 5. NI-VCOM automatically configures such signals as CRC signals. In such cases, it is not necessary to explicitly configure these signals with RBSSConfig. For more information, refer to **End-to-End Communication Protection**.

LIN Message Disassembly with LDF and ARXML Files

An `.ldf` or `.arxml` file contains the information for interpreting the raw data from the LIN messages received (Rx from the perspective of NI-VCOM). This is used to convert messages to signal names, values, and units. These signal values are available in NI-VCOM for both the user interface and the test automation as **ReadOnly** signals.

You can specify an `.ldf` or `.arxml` file for the disassembly of every LIN port in the **Network Descriptor** tag. For more information, refer to **LIN Configuration**.

Related concepts:

- [Using RBSConfig](#)
- [End-to-End Communication Protection](#)

Related reference:

- [LIN Configuration](#)

FlexRay

NI-VCOM also supports restbus simulation for FlexRay Bus. This requires an ARXML database.

Supported Hardware

NI-VCOM supports the NI PXI-8517 FlexRay interface card. Container-PDUs are not supported when using NI FlexRay interfaces.

Configuration in FlexRay Interfaces

Refer to ***FlexRay Configuration*** for information about setting up for a restbus simulation for FlexRay.

Related reference:

- [FlexRay Configuration](#)

Automotive Ethernet

Overview

Automotive Ethernet is a data communication module. You can use Automotive Ethernet to simulate and test car components using flexible test configurations. Use the SOME/IP Service Discovery Module to manage the availability of services in communication between ECUs on runtime, and control the send behavior of event

messages.

Complete a restbus simulation (RBS) on the Automotive Ethernet network based on the information provided in the AUTOSAR-XML file. For more information on common RBS features, refer to ***Software Features***.

The channel configuration for the Automotive Ethernet means that you need the VLAN configuration for Ethernet instruments.



Note Simulating Automotive Ethernet networks is only possible with the PXI Automotive Ethernet Interface Module (PXIe-8521, PXIe-8522, PXIe-8523) and allows deployment on Windows and NI Linux RT targets.

Related concepts:

- [Software Features](#)

Additional Functions

Modify CAN/CAN-FD Baud Rate at Run Time

For CAN ports, you can change the defined baud rate, even at run time, using the **BaudRate** signal. To make this signal visible in the NI-VCOM signal tree, configure the tag **Change BaudRate** in the NI:VCOM configuration tool to the corresponding restbus simulation peer. Then, set its value to `true`.

When dealing with instruments that support CAN-FD, there is an additional tag available called **Change-BaudRateFD**. When its value is set to `true`, the **DataBaudRate** signal is visible in the NI-VCOM signal tree.

Bus Load Measurement

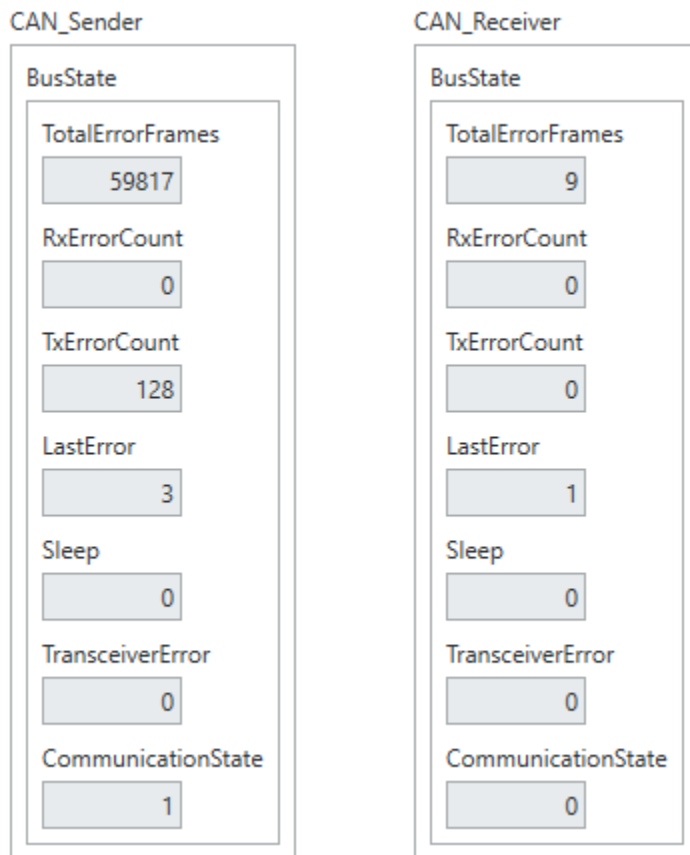
You can use NI-VCOM to measure the bus load of CAN and FlexRay instruments.

Set **EnableBusLoad** to `true` to enable the bus load measurement. Bus measurements are only an approximation and can differ from the real value. For CAN, a new signal called **Busload** is created, which shows the percentage of the load.

Bus State

- **CommunicationState**—Communicates state.
- **LastError**—Specifies the status of the last attempt to receive or transmit a frame.
- **RxErrorCount**—Counts Rx errors.
- **Sleep**—Indicates whether the transceiver and communication controller are in their sleep state. `False` indicates normal operation (awake) and `true` indicates sleep.
- **TotalErrorFrames**—Counts error frames since the start of the bus.
- **TransceiverError**—Indicates whether an error condition exists on the physical transceiver. This is typically referred to as the transceiver chip NERR pin. `False` indicates normal operation (no error), and `true` indicates an error.
- **TxErrorCount**—Counts Tx errors.

Figure 14. Example Bus State After Disconnecting the Loop Between Two CAN Ports



When to Create a Restbus Simulation File

You must create an **RBSDescriptor File** when your communication database file has a `.dbc` or `.ldf` extension. The `.dbc` and `.ldf` files do not contain all the information required to perform a full restbus simulation with features like automatic counter and calculated checksums.

Using RBSConfig

In a restbus simulation (RBS), control units that do not exist in the network are simulated. For these simulated control units, the test system sends the CAN, LIN, FlexRay, or Automotive Ethernet messages instead of the real electronic control unit. An RBS on a CAN, LIN, or FlexRay channel can use an RBS descriptor XML file. Generate the XML file from a network description file (`.ldf`, `.dbc` or AUTOSAR-XML) using the RBSConfig tool. For more information, refer to **Configuring a Restbus Simulation File**. The generated RBS descriptor XML file path has the XML tag `<RBSDescriptor>` inside the configuration file.

The RBSConfig tool does not support the new AUTOSAR-XML files with the Container-PDU concept (from the AUTOSAR Release v4.2.1). RBS in the NI-VCOM Toolkit runs based on the information provided directly in an AUTOSAR-XML file, which is specified in the XML tag Network Descriptor file. This means that there is no need for an RBS descriptor file if you are using an AUTOSAR-XML file.

Related concepts:

- [Configuring a Restbus Simulation File](#)

Configuring a Restbus Simulation File

The RBSConfig software tool can generate an RBS descriptor XML file, which you can specify for an RBS in the Configuration tool. For more information, refer to **CAN BUS** and **LIN BUS**.

Create RBS Descriptor File

1. Generate Networks

Generate networks to provide a basis for restbus simulation. Generate a new network in the Datapool pane using the following steps:

- a. Right-click the pane and select **New Network » CAN or FlexRay Network** or **New Network » LIN Network**, depending on your bus protocol.
- b. Enter a name for the new network in the dialog that appears. This name can only occur once in the datapool.
- c. Select whether you want to import multiple `.dbc` files or a single file. You can import multiple `.dbc` files in this step for a CAN network.

After you create a new network, the Datapool pane displays the empty network. If you import multiple `.dbc` files, then all the `.dbc` files placed in the selected folder import in a single step. In this case, the Datapool pane contains all the imported `.dbc` files.

2. Import Data Sources

Complete this step to import data sources into the new network.

- a. Select and right-click the network.
- b. Select **Add Data to Network**.
- c. Select the file that contains the data you want to import into the network in the Open File dialog. Select a `.dbc` or AUTOSAR XML file for a CAN. Select a `.ldf` file for a LIN. The program imports the data source and updates the Datapool pane.

After adding the data to the network, you can change the default RBS values by right-clicking the network and selecting **Change RBS Default Values**.

The default values are enabled `true` for all messages and disabled `false` for all nodes. If you restart the RBS Config Tool, the value properties reset to the default values. Every time you change the default values, right-click the network and select **Copy2Config**. If the Datapool pane has multiple `.dbc` files, then **Copy2Config** copies all the `.dbc` files to the configuration panel.

3. Select Messages

To select the messages for the restbus simulation, use the **Copy to Config ConfigSteps** as a resource. Generate these using one of the following options:

- Using the ConfigStepsView
- Dragging and dropping from the DatapoolView to the ConfigurationView
- Right-clicking the DatapoolView

A **Copy to Config *ConfigStep*** is an abstract description of which messages you should select for the configuration.

- Type
 - Copy to Config
- Property
 - Name
 - Transmitter
 - Receiver
 - Source
- Filter Expression
 - Case Sensitive

For example, you can select all of the messages received by a specific control unit:

Table 8. Selection of Messages in RBSCConfig

Type	Filter Property	Filter Expression
Copy to Config	Receiver	Name of the control unit

View and run messages with **Execute** through **Preview** in the ConfigStepsView.

You can also move individual messages or message lists to the ConfigurationView by right-clicking a message or message list and selecting **Copy2Config**.

After executing the **ConfigStep**, view the selected messages in the ConfigurationView.

4. Configure Restbus Simulation

Add autosignals using one of the following options:

- Right-click menu for a signal in the configuration (select **Change to Autosignal**)
- Define an autosignal using the **ConfigStepView**

Table 9. Filter for Autosignals in RBSCConfig

Ramp/Message Counter	^MC_
Toggle	Tgl\$
Checksum	^CRC_
Parity	^Prty_

The naming convention for the signals is listed above. The filter expressions can vary. Use the **regular expressions** known in computer science for Filter Expression to search for specific names. For example:

- ◦ **^ character at the beginning of a string**—The name begins with the specified string.
- ◦ **\$ character at the beginning of a string**—The name ends with the specified string.
- ◦ **String without ^ or \$ characters**—The name contains the specified string.

5. Exporting the Configuration

After you configure the complete restbus simulation, you can export the simulation as an XML file. Export the configuration using the **Save as XML** command in the network right-click menu. You can export the complete configuration with multiple networks. You can also export the individual configuration for a single network if you only select one network. To generate the XML file for restbus simulation, you must select **Selected Network**.

Related concepts:

- [CAN-BUS](#)
- [LIN-BUS](#)

Generating a Configuration

To generate a configuration, left-click the network in the Datapool pane and drag it to the Configuration pane.

This process applies to all of the CAN, LIN, and FlexRay messages and nodes in a `.dbc` or `.ldf` file.

Adding Autosignals to the Configuration Through ConfigStepsView

You can add autosignals to the configuration through the Configuration Steps pane for a CAN database.



Note Autosignals are uncommon, but possible, for LIN.

In the Configuration Steps pane, select **Type** » **Change to AutoSignal**.

- **Message Counters**—

1. Define the filter expression as `^MC_` (or a filter to filter all of the signals with message counters).
2. Check through **Preview** and select the `Ramp` type for **AutoSignal Type**.
3. Add selected signals to the autosignals through **Execute**.

The signals are now present in the Configuration Steps pane as **AutoSignals**.

Configure the autosignal parameters in the Details pane.

- **Toggle (refer to Message Counters)**—

- Filter Expression: `Tgl$`
- AutoSignal Type: `Toggle`

To apply this autosignal type to selected signals, click **Execute**.

- **Parity (refer to Message Counters)**—

- Filter Expression: `^Prty_`
- AutoSignal Type: `ParityBit`

To apply this autosignal type to selected signals, click **Execute**.

- **Checksum (refer to Message Counters)**—To configure **Checksum** autosignals, you must provide additional information, including `InitialValue`, `FinalXORValue`, `CRCMask`, `UserByteBefore`, and `UserByteAfter`. The following is a list of configurable checksums and additional parameter information:

- `ChecksumJ1850 (8 Bit)`
- `ChecksumUserJ1850 (8 Bit)`
 - ◦ **InitialValue**—CRC initial value. The default is `0xFF (255)`.
 - ◦ **FinalXORValue**—XOR value at the end of the CRC algorithm. The default is `0xFF (255)`.
 - ◦ **CRCMask**—Determines the range of data bytes that the CRC calculation uses. For example, `CRCMask 0x7F (127)` means that the checksum calculation uses Byte 0 to Byte 6 of the data bytes.
 - ◦ **UserByteBefore**—The bytes used before the data bytes for the checksum calculation.
 - ◦ **UserByteAfter**—The bytes used after the data bytes for the checksum

calculation.

- Custom Checksum CRC-8: Select the `CustomChecksum` type as **AutoSignal Type** for a custom CRC-8 checksum. The following is a list of additional parameters that you can provide through this tool:
 - ◦ **CRCPolynomial**—A hexadecimal representation of the CRC-8 polynomial without the most significant bit. The following polynomial is represented as 0x14D:

$$x^8 + x^6 + x^3 + x^2 + 1$$

In this case, enter 0x4D in the `CRCPolynom` field (instead of 0x14D).
 - ◦ **InitialValue**—CRC initial value.
 - ◦ **FinalXORValue**—XOR value applied at the end of the CRC algorithm.
 - ◦ **CRCMask**—Determines the data bytes of the CAN messages that the CRC calculation uses. For example, enter 127 (decimal representation of 0x7F) to use bytes 0 to 6.
 - ◦ **ReflectedInput**—Reverses the order of the message bytes before the CRC calculation.
 - ◦ **ReflectedOutput**—Reverses the order of the computed CRC bits before output.
- ChecksumXOR (8 Bit)
 - ◦ **FinalXORValue**—XOR value at the end of the CRC algorithm.
 - ◦ **CRCMask**—Determines the data bytes of the CAN messages that the CRC calculation uses. For example, 0x7F (127) means that the `CRCMask` uses bytes 0 to 6.
- ChecksumXOR4Bit (4 Bit)
 - ◦ **CRCMask**—Determines the data bytes of the CAN messages that the CRC calculation uses. For example, 0x7F (127) means that bytes 0 to 6 are used.
 - ◦ **Start Bit**—Determines the bit where you can find the CRC checksum.

The following example shows how this algorithm works:

```
Checksum = Byte1XORByte2XORByte3XORByte4
Checksum = (higher_nibble(Checksum))XOR(lower_nibble(Checksum))
Checksum=ChecksumXORMSG_CNT
```

- ChecksumAddWithCarry
 - ◦ **FinalXORValue**—XOR value at the end of the CRC algorithm.

The following example shows how this algorithm works:

```

if(Byte1 + Byte2 > 255)then(C = 1)else(C = 0)
Checksum = Byte1 + Byte2 + C
if(Checksum + Byte3 > 255)then(C = 1)else(C = 0)
Checksum = Checksum + Byte3 + C
if(Checksum + Byte4 > 255)then(C = 1)else(C = 0)
Checksum = Checksum + Byte4 + C
if(Checksum + Byte5 > 255)then(C = 1)else(C = 0)
Checksum = Checksum + Byte5 + C
if(Checksum + Byte6 > 255)then(C = 1)else(C = 0)
Checksum = Checksum + Byte6 + C
if(Checksum + Byte7 > 255)then(C = 1)else(C = 0)
Checksum = Checksum + Byte7 + C
Checksum = ChecksumAND255
Checksum = ChecksumXOR126

```

- ChecksumAddAndComplementToOne (8 Bit)

The following example shows how this algorithm works:

```

Checksum = (Byte1 + Byte2 + Byte3 + Byte4 + Byte5 + Byte6 + Byte7)
Checksum = ChecksumXOR255

```

- ChecksumGAC (8 bit)

The following example shows how this algorithm works. Assume that the checksum position is byte 4 (*Byte4*):

```

Checksum = (Checksum + Byte0 + Byte1 + Byte2 + Byte3)
Checksum = (Checksum + Byte5 + Byte6 + Byte7)
Checksum = (((Checksum & 0xFF)XOR255) + 1)& 0xFF

```

- ChecksumMazda (8 Bit)

The following example shows how this algorithm works. Assume that the checksum position is byte 4 (Byte4):

```
Checksum=uFrameID & 0xFF
Checksum = Checksum + (uFrameID >> 8)& 0xFF
Checksum = Checksum + FrameLength
Checksum = (Checksum + Byte0 + Byte1 + Byte2 + Byte3)
Checksum = (Checksum + Byte5 + Byte6 + Byte7)
```

To apply one of these autosignal types to the selected signals, click **Execute**.



Note For information about which bus systems support which checksums, refer to ***End-to-End Communication Protection***.

Related concepts:

- [End-to-End Communication Protection](#)

Manually Adjusting Parameters

You can manually adjust parameters in a LIN RBS configuration using RBSConfig. Specify the **DefaultScheduleTable** for the master node by clicking the field next to **DefaultScheduleTable** and selecting the desired schedule table from the drop-down menu. The best option is usually to select the normal schedule table.

Activation and Configuration of Network Management

For a CAN, configure the network management parameters in Details pane in the section for configuration of **Nodes**. To enable NM simulation by default, set `IsNmEnable` to `true`. All the NM nodes are listed in the Node Details window.

To activate or deactivate a node as an NM node, set `IsNmNode`. By default, all nodes are activated as NM nodes.

Disabling Network Management and Diagnosing Messages

For a CAN, NI recommends disabling the network management and diagnosing messages if they are not relevant for your system. Use the Configuration Steps pane of RBSConfig to enable or disable these messages. To confirm your changes, click **Preview**.

Generally, the network management messages start with `NM_`, and the diagnosing messages start with `D_R`. Pre-configured filter expressions exist for these messages, but you can change the filter expression at any time. You can also enable or disable any other type of message.

Vehicle Communication Toolkit Add-Ons

The Measurement and Calibration Toolkit and the Diagnostic Toolkit are optional add-ons to the Vehicle Communication Toolkit (Full version) that expand your ability to communicate with real or simulated ECUs.

The Measurement and Calibration Toolkit

The Vehicle Communication Measurement and Calibration Toolkit is an optional add-on to the Vehicle Communication Toolkit (Full version) that expands your ability to communicate with real or simulated ECUs.

If your system uses the Measurement and Calibration Toolkit for NI-VCOM, refer to the following topics for more information.

Viewing a Measurement and Calibration Database File

Complete the following steps to view a measurement and calibration database file:

1. Launch the WebUI.
2. Select **Calibration**.
3. Upload or select a calibration database file.
4. Select **View Database File** to access a detailed view.

The following tabs appear.

- **Protocols**: Displays database file protocols and related event channels.
- **Calibration Pages**: Displays database file segments and pages.
- **Objects**: Displays database file objects, including measurements, characteristics, functions, and groups.

Using Measurement and Calibration APIs

You can develop with the Measurement and Calibration Toolkit using the LabVIEW API. Refer to your in-product help or contact NI for more information about specific VIs.

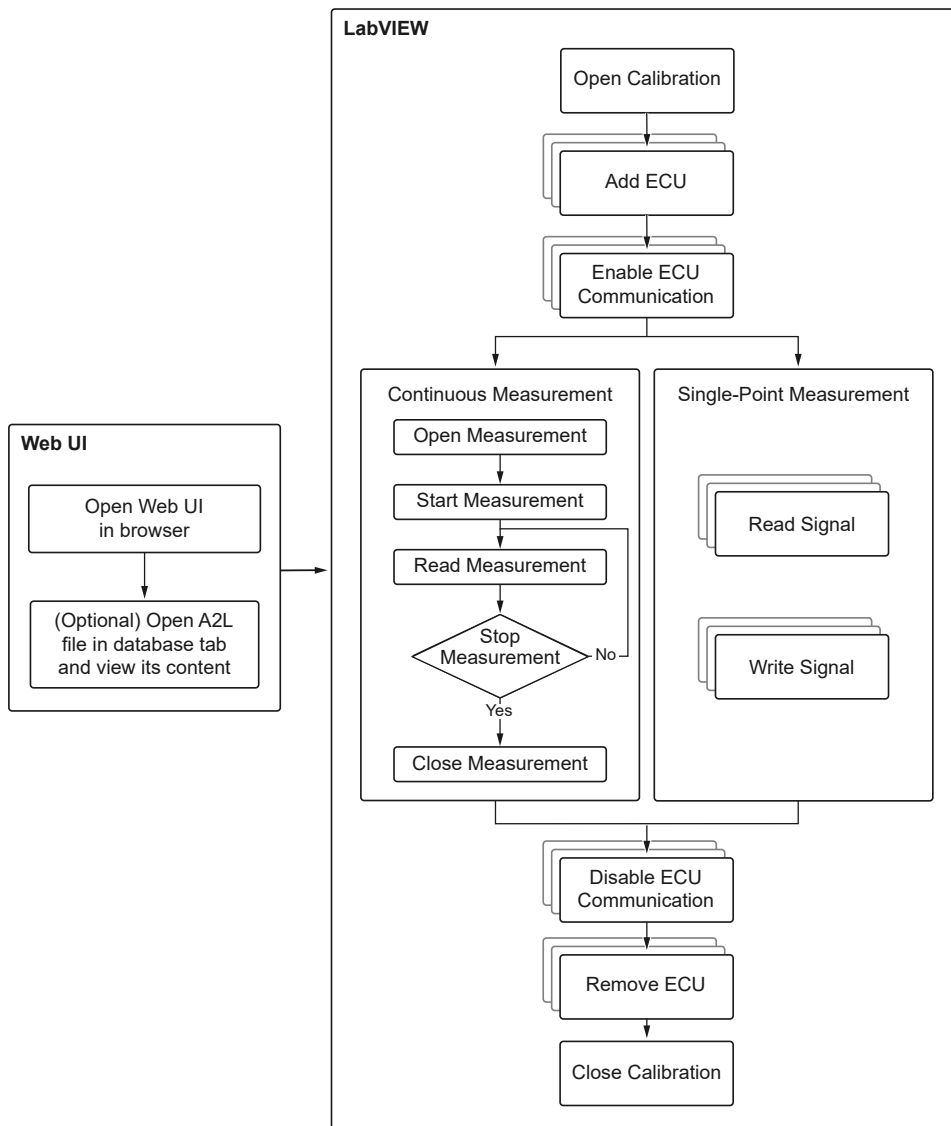
The C API is available as an alternative development option upon request. Contact NI for more information.

Contact NI to access a beta version of the following VI:

- Add ECU for FlexRay

The following diagram represents a typical measurement and calibration API workflow in LabVIEW:

Figure 15. Measurement and Calibration LabVIEW API Workflow



1. The Open VI initializes a measurement and calibration session and establishes a connection with the calibration server.



Note When initializing a measurement and calibration session, the connection fails if the specified port is already in use. If this occurs, you must specify a different port to connect to the calibration server. The default host value is `localhost` and the default port value is `9876`.

Complete the following steps to change the port value:

- a. Open the INI file, located at `%Public%\Documents\PROVEtechTA\cfg\TM.ini`.
- b. Locate the `[gRPC1]` tag and change the last four digits of the `Listen` line to your desired port value.
- c. Save the INI file.

2. The Add ECU VI adds ECU configurations to the calibration server. You must add each ECU separately with its configuration parameters.
 - Use the `ECU reference out` parameter to set optional ECU configurations with the property node.
3. The Enable ECU Communication VI enables communication between the calibration server and individual ECUs.
4. Choose from continuous measurement or single-point measurement.
 - Continuous measurement enables you to read signal data from the measurement signal list using either polling mode or DAQ list mode. Measurement continues until you stop it or an error occurs. In polling mode, data is acquired through cyclic polling of each signal, where a cycle takes longer the more signals it has. In DAQ list mode, data is acquired in equidistant time intervals as defined in the A2L database. Refer to **Measurement Execution Workflow** for more information.
 - The Open Measurement VI adds the measurement signal list and the measurement settings to the measurement task. The `measurement settings` parameter includes `measurement mode`, `buffer size`, and `update cycle time`.



Note This VI only supports opening and running a single measurement task at once. Do not use multiple instances of the Open Measurement VI at the same time. Since only one instance of measurement can be active, you can use the measurement operation to measure multiple signals with different sub-sampling rates.

- The Start Measurement VI starts the measurement for the task and starts transmission of the measurement signal(s) from the ECU(s).
- The Read Measurement VI returns the measurement data for the signal list added to the Open Measurement VI.
- The Stop Measurement VI stops the measurement for the task and stops transmission of the measurement signal(s) from the ECU(s).
- The Close Measurement VI clears the measurement task configuration.
- Single-point measurement enables you to read and write single samples of data to and from individual ECUs through cyclic polling of each selected signal, where a cycle takes longer the more signals it has.
 - The Read Signal VI returns a single measurement value from the specified signal name. Only DBL, U64, and INT64 datatypes are supported. Any other datatype that is 32-bit or less is read as a DBL datatype. Use the String datatype to read a string value from the signal.
 - The Write Signal VI writes a single value to the specified signal name. Only DBL, U64, and INT64 datatypes are supported. Any other datatype that is 32-bit or less is read as a DBL datatype. Use the String datatype to write a string value to the signal.
- 5. The Disable ECU Communication VI disables communication between the calibration server and individual ECUs.
- 6. The Remove ECU VI removes ECU configurations from the calibration server.
- 7. The Close VI closes the connection to the calibration server.

Measurement Execution Workflow

The Measurement and Calibration Toolkit reads measurement values from an ECU using DAQ list mode or polling mode.

DAQ List Mode

In DAQ list mode, data is transmitted from the ECU to the calibration engine in equidistant time intervals based on the specified `event_channel` and `signal_list` parameters. You can use DAQ list mode to monitor ECU data without permanently polling the values of these measurements. After a DAQ list has been set up and initialized, the ECU transmits the data autonomously without any further action. The number of available DAQ lists and their sizes depend on the implementation of the ECU.

The calibration engine adds the signals in the measurement list of the Open Measurement VI to the ECU DAQ list using data acquisition commands. After you start the measurement, the ECU continues sending measurement values for each signal at the rate decided by their event channel to the calibration engine until you stop the measurement. After this measurement data is available in the calibration engine, it is streamed to the data buffer at the rate defined by the `update cycle time` parameter, the value of which is calculated, in seconds, as

$$\min \left(fBufferSizeInSeconds / 2, 4 \right)$$

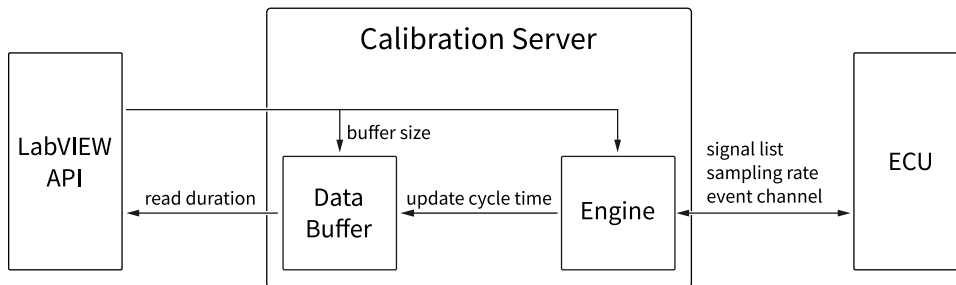
. The data buffer controls the throughput of the data sent from the calibration engine to the LabVIEW API.

The Read Measurement VI `read duration` parameter dictates how long a client reads streaming data from the calibration engine. The buffer size is determined by the `buffer size` parameter. If the data buffer fills up, the calibration engine generates an error and stops streaming data.



Note Your parameter selection and settings affect the CPU usage of your system and may need to be optimized.

Figure 16. DAQ List Mode



Polling Mode

In polling mode, data is transmitted from the ECU to the calibration engine through cyclic polling of each signal using the `UPLOAD XCP` command. This operation takes place as fast as possible based on the specified `sampling rate` parameter.

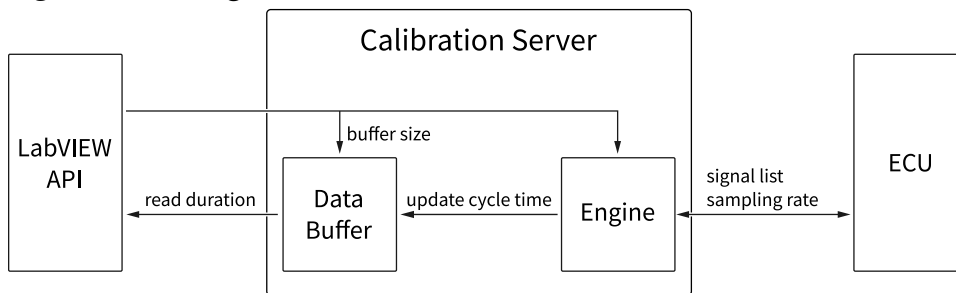
The measurement value for each signal added to the measurement list with the Open Measurement VI is read from the ECU. After the Start Measurement VI is called, the calibration engine requests the measurement value for each signal using the short

upload command (0x $F4$) and then the ECU responds to the calibration engine with the requested data. Since each signal measurement requires both a request and a response, the more signals are in the measurement list, the longer it takes to complete a single measurement of all signals. This operation continues until the Stop Measurement VI is called.



Note Your parameter selection and settings affect the CPU usage of your system and may need to be optimized.

Figure 17. Polling Mode



The Diagnostic Toolkit

If your system uses the Diagnostic Toolkit for NI-VCOM, refer to the following topics for more information and contact NI for additional support as needed.

Related reference:

- [Vehicle Communication Software Suite Overview](#)
- [The Measurement and Calibration Toolkit](#)

Viewing a Diagnostic Database File

Complete the following steps to view a diagnostic database file:

1. Launch the WebUI.
2. Select **Diagnostic**.
3. Upload or select a diagnostic database file.
4. Select **View Database File** to access a detailed view.
The following tabs appear.

- **Project:** Displays a list of available database files.
- **Services:** Displays available services with accompanying requests, responses, and parameters. Select an ECU variant to view its content.
- **Comm Parameters:** Displays communication parameters for ECU variants. Select an ECU variant to view its content.
- **DIDs:** Displays read/write data IDs with detailed information. Select an ECU variant to view its content.
- **DTC:** Displays diagnostic trouble codes for ECU variants. Select an ECU variant to view its content.

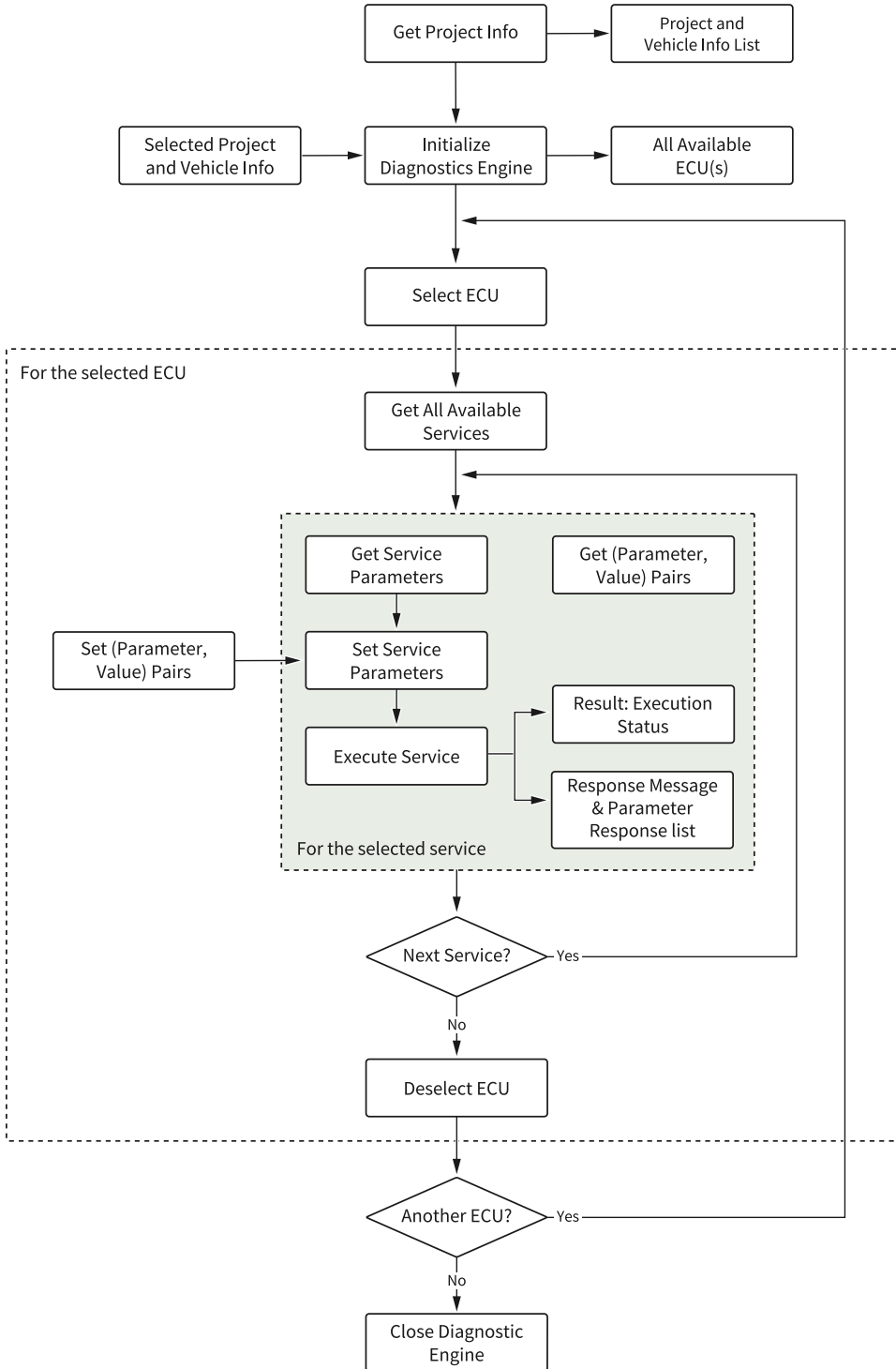
Using Diagnostic APIs

You can develop with the Diagnostic Toolkit using the LabVIEW API. Refer to your in-product help or contact NI for more information about specific VIs.

The C API is available as an alternative development option upon request. Contact NI for more information.

The following diagram represents a typical diagnostic API workflow in LabVIEW:

Figure 18. Diagnostic LabVIEW API Workflow



Note When initializing a diagnostic session, the connection fails if the specified port is already in use. If this occurs, you must specify a different port to connect to the diagnostic server. The default host is `localhost` and the default port is `9876`. Complete the following steps to specify a different

port:

1. Open `TM.ini`, located in the `%Public%\Documents\PROVEtechTA\cfg` directory.
2. Locate the `[gRPC1]` tag and change the last four digits of the `Listen` line to your desired port value.
3. Save the INI file.

Configuring the Diagnostic Toolkit

Complete the following steps to configure the Diagnostic Toolkit for your application. Install and activate the Softing Smart Diagnostic Engine (SDE).



Note Refer to documentation from Softing for detailed information about installing and configuring the SDE and DTS.

1. Configure the DTS interface.
2. Configure an MCD-3D configuration INI file.

Related reference:

- [Software Installation and Activation](#)

Configuring the DTS Interface

Refer to the readme file located at `%Program Files%\PROVEtechDPDU\README.txt` for detailed instructions about configuring the DTS interface with the D-PDU API.

Configuring an MCD-3D File

1. Open the default MCD-3D configuration file from the following location: `%Public Documents%\PROVEtechTA\cfg\DiagnosticsMCD3D.ini`.
2. Copy the following configuration values into the INI file underneath the `[Config.MyConfig]` heading:
 - `DServerPath=<Path to the DTS Server DLL>`



Note Example file path: `%Program Files%\Softing\Smart Diagnostic Engine\9.03\bin\ComApi.dll`

- `FunctionalGroups=0`
- `Project=<Project_Name created in System Configurator>`



Note If you created a DTS project in the System Configurator, specify that project name here. Otherwise, refer to the System Configurator documentation to create a new project in the System Configurator. To run the Diagnostic Toolkit LabVIEW examples, ensure that your DTS project references the PDX file in the following location:

`%Public%\Documents\National Instruments\NI-VCOM\Diagnostic Toolkit\NIECU.pdx.`

- `VIName=<Vehicle_Information_Name created in System Configurator>`
3. Copy the following configuration value into the INI file underneath the `[ConfigAssignments]` heading:
 - `MyConfig=VcomToolkitCalAndDiag`
 4. Save the INI file.
 5. Restart the diagnostic server.
 - a. Open LabVIEW.
 - b. Select **Tools » Vehicle Communication Toolkit » VCOM Helper Utility**. The VCOM Helper Utility launches.
 - c. Ensure that the Server Host Name is set to `localhost` and the Port Number is set to `9876`.
 - d. Click **Restart Server**.



The PROVEtech:TA Log Window

The PROVEtech:TA Log window provides additional status and error information about the calibration and diagnostic server, which runs in the background to process core operations anytime the Measurement and Calibration Toolkit or the Diagnostic Toolkit is running.

Select **PROVEtech:TA Log** from the system tray to open the PROVEtech:TA Log window. This logging takes place through a separate process that is independent of the

Measurement and Calibration Toolkit and Diagnostic Toolkit, so the calibration and diagnostic server continues to log core operations in the background even if the measurement and calibration or diagnostic operation crashes. The last output in the log displays the operation with which a crash is most likely associated. The log can only hold a limited number of messages, so it deletes old messages when it reaches this limit.

Each message is displayed as follows:

- **Standard Status Messages**—Displayed in black.
- **Warnings**—Displayed in color and identified by the  symbol. Warnings indicate a status that may result in a future issue.
- **Errors**—Displayed in color and identified by the  symbol. Errors usually cause the current function to be aborted.

The location of the log file path is `%Public%\Documents\AKKA\PROVEtechTA\traces\Trace.txt`. PROVEtech:TA creates a new version of this file each time it starts and renames the existing one within the same folder for backup purposes.

Using the VCOM Helper Utility

Use the VCOM Helper Utility to restart or check the status of the calibration and diagnostic server. Complete the following steps to run the VCOM Helper Utility:

1. Launch the VCOM Helper Utility by opening LabVIEW and then selecting **Tools** » **Vehicle Communication Toolkit** » **VCOM Helper Utility**.
2. Verify the server host name and port number values.
3. Click **Check Server Status** to get the status of the server. Status is shown in green if the server is running properly.
4. (Optional) Click **Restart Server** to force the server to restart or to launch the server if it is not already running.
5. Click **Exit** to stop the VCOM Helper Utility.

Examples

The following examples provide more information to help you utilize NI-VCOM to meet your application requirements.

NI-VCOM Examples

To locate examples, launch the NI Example Finder for LabVIEW and search for VCOM or navigate to the following location: `%Program Files%\National Instruments\LabVIEW\examples\Vehicle Communication Toolkit`.

The examples require you to activate the NI-VCOM Toolkit with an appropriate license.

Demo ECU

Navigate to the following path to locate the Demo ECU Simulator:

`%Public%\Documents\National Instruments\NI-VCOM\Examples\Demo ECU`.

The simulator is a LabVIEW application that demonstrates the following features of the NI-VCOM Toolkit:

- Read and write operations for signals and messages with various automotive communication protocols.
- Support of CAN, LIN and Automotive Ethernet protocols.
- Support of AUTOSAR XML, DBC and LDF Network Descriptors.

To launch the Demo ECU Simulator, double-click `ECU Simulator.exe`.

Running C Examples

Navigate to the following path to locate the C example: `%Public%\Documents\National Instruments\NI-VCOM\Examples\MS Visual C`.

You must install the following tools:

- CMake for Windows
- C++ Compiler and Build Toolchain – For Windows, you can use Build Tools for Visual Studio. The version most recently tested is Build Tools for Visual Studio 2022. This version comes by default if you have already installed the appropriate Visual Studio version. If you choose to use VS Code instead, you must install the build toolchain manually.

Additionally, if you are using VS Code, NI recommends installing the following VS Code extensions published by Microsoft:

- C/C++
- CMake Tools
- Complete the following steps to build NI-VCOM C examples using Visual Studio:
 1. Open Visual Studio and select **Continue without code**.
 2. Navigate to **File » Open » CMake**.
 3. Navigate to `%Public%\Documents\National Instruments\NI-VCOM\Examples\MS Visual C` and select the `CMakeLists.txt` file.
 4. Select **Build All** from the **Build** menu after the configuration loads.
- Complete the following steps to build NI-VCOM C examples using Visual Studio Code:
 1. Open Visual Studio Code and select **Open Folder**.
 2. Navigate to `%Public%\Documents\National Instruments\NI-VCOM\Examples\MS Visual C` and open that folder in Visual Studio Code.
 3. Press `<Ctrl+Shift+P>` after the folder opens.
 4. Type `CMake: Build`, then select this command to start the build. You can also use `<F7>` for this step.
 5. Choose the appropriate x64 architecture compiler toolkit in Visual Studio Code. The NI-VCOM C examples only build the x64 target, so you must select an x64 compiler toolkit.
 6. Ensure the build starts and eventually completes with no errors. This may take a few minutes.

Find the result of the build in `%Public%\Documents\National Instruments\NI-VCOM\Examples\MS Visual C\bin64`. Two applications

are built:

- • **Demo ECU Communication**—This example shows how to communicate with the shipping Demo ECU in the NI-VCOM Toolkit using the C++ API.
- • **Restbus Communication**—This example shows how to perform simple restbus communication between two peers/nodes from the same application using the C++ API.

To get help using the examples, execute an example application with the argument `--help` from the command line.

Configuration Tree Browsing for LabVIEW

After you install the toolkit, navigate to the following path to locate the example:

```
%Program Files%\National Instruments\LabVIEW\examples\  
Vehicle Communication Toolkit\VCOM Full - Configuration  
Tree Browsing.vi.
```

This example demonstrates how to process and visualize all signals from an RBS configuration. You can copy the signal names for use with Write and Read VIs.

CAN-FD Example

This CAN-FD sample project calculates the CRC according to AUTOSAR End-to-End-Profile 2. In this profile, special Data-IDs are included in the `.dbc` file. These are used to calculate the CRC depending on the counter value and the Data-ID.

To deploy this example, you must connect automotive CAN Flexible Data-Rate hardware to your system.

You must use two interconnected CAN ports. Configure their names in NI-MAX as `CAN1` and `CAN2`.

You can find a list of CAN interface instruments on the NI store page for CAN hardware at ni.com/shop.

In VeriStand

How to Import the Project

After you install the toolkit, navigate to the following path to locate the example folder:
`%Public%\Documents\National Instruments\NI VeriStand\Examples.`



Note The directory depends on the VeriStand version already installed. If more than one version of VeriStand is installed on your computer and you installed the toolkit for all VeriStand versions, the example is available for each version.

After VeriStand starts, import the example project to add it to VeriStand. Use one of the following methods to locate and import your project:

- Click **Browse**.
- Navigate to **File » Open Project**.
- Press `<Ctrl+O>`.

Look for the project file `NI-VCOM_CAN-FD_Example.nivsprj` in the example folder.

Double-click the project to open the Project Explorer window. To run the project, press `<F5>` or click **Deploy**.

In the UI Manager, some signals are already available to show some of the features the toolkit has. You can import more signals by pressing `<CTRL+M>` and selecting a signal in the System Definition panel.



Note Starting in VeriStand 2019, you can import a complete folder by dragging it from the System Definition panel into the Workspace.

Select a Different Target

Specify the **Operating System** and **IP Address** parameters in VeriStand for your target.



Note To deploy the project on a different target, update the **Operating**

System and IP Address parameters.

In LabVIEW

How to Open the Project

After you install the toolkit, navigate to the following path to locate the example folder:
`%Program Files%\National Instruments\LabVIEW\examples\
Vehicle Communication Toolkit\VCOM Full Examples.`

This example demonstrates how to use the API for CAN in signals single point mode. You can open the `Configuration Tree Browsing.vi` example in parallel to have better access to all signals. For more information, refer to ***Configuration Tree Browsing for LabVIEW***.

A simulated ACC ECU sends the message ACC_ESP_E2E_PR2 on port CAN1 to a simulated ESP ECU on port CAN2.

To view all traffic, run the NI-XNET Bus Monitor as a subordinate session on either CAN1 or CAN2.

Related concepts:

- [Configuration Tree Browsing for LabVIEW](#)

LIN Example

This LIN sample project simulates communication between a LIN Master and a LIN Slave.

To deploy this example, you must connect LIN hardware to your system.

You must use two interconnected LIN ports that are each connected to an external power supply. Configure their names in NI-MAX as LIN1 and LIN2.

You can find a list of LIN interface instruments on the NI store page for LIN hardware at

ni.com/shop.

In VeriStand

After you install the toolkit, you can find an example folder through the following path:

```
%Public%\Documents\National Instruments\NI VeriStand\  
Examples.
```



Note The directory depends on the VeriStand version already installed. If you have more than one version of VeriStand on your computer and you installed the toolkit for all VeriStand versions, the example is available for each version.

As soon as the example loads, you can deploy your project and start working with your LIN hardware and the example.

In LabVIEW

After you install the toolkit, navigate to the following path to locate the example:

```
%Program Files%\National Instruments\LabVIEW\examples\  
Vehicle Communication Toolkit\VCOM Full Examples\VCOM Full  
- Signals Single Point LIN.vi.
```

Automotive Ethernet Example

This Automotive Ethernet sample project shows the configuration of an automotive network restbus simulation.

To deploy this example, you must connect Automotive Ethernet hardware to your system and enable both the transmitter and receiver.

You must use two interconnected Ethernet ports. Configure their names in NI-MAX as ENET1 and ENET2.

In VeriStand

After you install the toolkit, you can find an example folder through the following path:

%Public%\Documents\National Instruments\NI VeriStand\
Examples.



Note The directory depends on the VeriStand version already installed. If you have more than one version of VeriStand installed on your computer and you installed the toolkit for all VeriStand versions, the example is available for each version.

As soon as the example loads, you can deploy your project and start working with your Automotive Ethernet hardware and the example.

In LabVIEW

After you install the toolkit, navigate to the following path to locate the example:

%Program Files%\National Instruments\LabVIEW\examples\
Vehicle Communication Toolkit\VCOM Full Examples\VCOM Full
- Signals Single Point Ethernet.vi.

This example demonstrates how to use the API for Automotive Ethernet signals in single-point mode.

Diagnostic Toolkit and Measurement and Calibration Toolkit Examples

Table 10. Diagnostic Toolkit and Measurement and Calibration Toolkit Examples

Toolkit	Examples	Demo ECU
Measurement and Calibration Toolkit	Launch the NI Example Finder for LabVIEW and browse or search for VCOM-MC.	Navigate to %Public%\Documents\National Instruments\NI-VCOM\Examples\Measurement and Calibration Toolkit\Demo ECU. Launch the Demo ECU and select Help for more information about how to use it.

Toolkit	Examples	Demo ECU
Diagnostic Toolkit	Launch the NI Example Finder for LabVIEW and browse or search for VCOM-DI.	Navigate to %Public%\Documents\National Instruments\NI-VCOM\Examples\Diagnostic Toolkit\Demo ECU. Launch the Demo ECU and select Help for more information about how to use it.