CAN

ECU Measurement and Calibration Toolkit User Manual

July 2005
371601A-01

NATIONAL INSTRUMENTS™

Important Information

Contents

Chapter 6 ECU M&C API for C

About This Manual

This manual provides instructions for using the ECU Measurement & Calibration (ECU M&C) Toolkit. It contains information about installation, configuration, and troubleshooting, and also contains ECU M& C function references for LabVIEW-based and C-based APIs.

Use the ECU M&C Toolkit Installation Guide in the jewel case of the program CD to install the ECU M&C Toolkit software. Use this manual to learn the basics of ECU Measurement and Calibration, as well as how to develop an application.

Conventions

The following conventions appear in this manual:

»	The » symbol leads you through nested menu items and dialog box options to a final action. The sequence **File»Page Setup»Options** directs you to pull down the **File** menu, select the **Page Setup** item, and select **Options** from the last dialog box.
💡	This icon denotes a tip, which alerts you to advisory information.
📝	This icon denotes a note, which alerts you to important information.
bold	Bold text denotes items that you must select or click in the software, such as menu items and dialog box options. Bold text also denotes parameter names.
italic	Italic text denotes variables, emphasis, a cross reference, or an introduction to a key concept. This font also denotes text that is a placeholder for a word or value that you must supply.
`monospace`	Text in this font denotes text or characters that you should enter from the keyboard, sections of code, programming examples, and syntax examples. This font is also used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, functions, operations, variables, filenames, and extensions.
`monospace italic`	Italic text in this font denotes text that is a placeholder for a word or value that you must supply.

Related Documentation

The following documents contain information that you might find helpful as you read this manual:

- *ANSI/ISO Standard 11898-1993, Road Vehicles—Interchange of Digital Information—Controller Area Network (CAN) for High-Speed Communication*
- *CAN Specification Version 2.0*, 1991, Robert Bosch GmbH., Postfach 106050, D-70049 Stuttgart 1
- *CiA Draft Standard 102, Version 2.0*, CAN Physical Layer for Industrial Applications
- *CAN Calibration Protocol Specification, Version 2.1*, ASAP Arbeitskreis zur Standardisierung von Applikationssystemen Standardization of Application/Calibration Systems task force
- *Interface Specification Interface 2 (ASAM MCD 2MC/ASAP2) Version 1.51 Release 2003-03-11*, Applications Systems Standardization Working Group
- *NI-CAN Hardware and Software Manual*

1 Introduction

The ECU Measurement and Calibration (ECU M&C) Toolkit contains a development system for electronic control units (ECU) based on existing ASAM standards. The function set of the ECU M&C Toolkit enables engineers to optimize and verify the functionality of electronic controller devices. Most ECUs interact with other ECUs, external sensors, and actuators in a Controller Area Network (CAN). During the development and verification phase of an ECU, engineers access the ECU for acquired data (Measurement), or to adjust parameters inside the ECU itself (calibration). Since the bandwidth and number of identifiers for a CAN network is limited, the Association for Standardization of Automation and Measuring Systems (ASAM e.V.) has specified the CAN Calibration Protocol (CCP), a protocol layer based on CAN to access the measurement and calibration data in an ECU.

The ECU M&C Toolkit is particularly suited to the automotive industry and their component suppliers. It provides a function set that can be used in the development or verification phase of an ECU. Access to the data inside an ECU takes place based on information stored in ASAM MCD 2MC (`*.A2L`) database files that are provided by the ECU suppliers. Selecting the signals by their names provides convenient access to the data inside an ECU. The ECU M&C Toolkit uses the CCP protocol as the fundamental communication protocol and for support of ECU database (`*.A2L`) files.

CAN Calibration Protocol (CCP) Overview

The CAN Calibration Protocol is a CAN-based master-slave protocol for calibration and data acquisition. A single master device (host) can be connected to one or more slave devices. Before a slave device may accept commands from the host, the host has to establish a logical point-to-point connection to the slave device. After this connection has been established, the slave device has to acknowledge each command received from the host within a specified time.

CCP defines two function sets—one for control/memory transfer, and one for data acquisitions that are independent of each other and may run asynchronously. The control commands are used to carry out functions in the slave device, and may use the slave to perform tasks on other devices. The data acquisition commands are used for continuous data acquisition from a slave device. The devices continuously transmit internal data according to a list that has been configured by the host. Data acquisition is initiated by the host, then executed by the slave device, and may be based on a fixed sampling rate or be event-driven.

The communication of controllers with a master device through CCP is based on the CAN 2.0B standard (11-bit and 29-bit identifier), which includes 2.0A (11-bit identifier) for data acquisition from the controllers, memory transfers to the controllers, and control functions in the controllers for calibration.

The ECU M&C Toolkit abstracts the CCP communication layer so that it is transparent to the user. For most cases it is sufficient that the underlying CCP communication is handled by the toolkit kernel itself. Nevertheless, the ECU M&C Toolkit offers direct access to the low level CCP commands if a non-standard timing behavior or independent user defined command sequence is needed.

CCP Protocol Version

The ECU M&C Toolkit supports the CAN Calibration Protocol specification, version 2.1.

Measurement and Calibration Databases

The ASAP description file (ASAP2 or ASAM MCD 2MC) is used to describe the ECU internal memory configuration. An ASAM MCD 2MC description file with the file extension `.A2L` contains information and access location for the relevant data objects in the ECU, such as:

- Project relevant information
- ECU data structure
- Conversion procedures for representation in physical units
- Descriptions of the available Measurement channels inside the ECU
- Descriptions of the available Characteristics inside the ECU
- Descriptions of how to access the ECU over CAN

Use of the ECU M&C Toolkit requires an existing ASAM MCD 2MC database file. These files can be generated by various third-party utilities. A database editor for ASAM MCD 2MC databases is not part of the ECU M&C Toolkit.

ECU Measurements

The ECU M&C Toolkit gives the user access to ECU internal physical values defined by their names in the ASAM MCD 2MC database file. Based on this information, the ECU M&C Toolkit communicates through CCP to the ECU. A DAQ (data acquisition) list can be set up, which sends ECU internal data synchronously or asynchronously to the CCP master. The ECU M&C Toolkit provides a way to configure several Measurement channels into a single Measurement task. The term *task* refers to a list of measurements (channels) read or written together. A common use of the task concept is to read DAQ channels available on the ECU.

ECU Characteristics

ECU *characteristics* are maps of ECU internal variables, which may be used as calibration information or set-point information. The ECU memory content of characteristics can be read or even changed with the help of the ECU M&C Toolkit.

2 Installation and Configuration

This chapter explains how to install and configure the ECU M&C Toolkit.

Installation

This section discusses the installation of the ECU M&C Toolkit for Microsoft Windows.

Note You need administrator rights to install ECU M&C Toolkit on your computer.

1. Insert the ECU M&C Toolkit CD into the CD-ROM drive.
2. Open Windows Explorer.
3. Access the CD-ROM drive.
4. Double-click on `autorun.exe`. This will launch the software interface.
5. Start the installation. The installation program will guide you through the rest of the installation process.
6. When installation is complete, the National Instruments License Manager will launch automatically to activate your license.

License Management Overview

License management is the process of controlling access to products based on an explicit license agreement. The ECU M&C Toolkit requires an activated license in order to launch, so a license must be acquired and activated before the product can be used. The activation process involves using the Activation Wizard to send the following information to National Instruments:

- The product you are activating: *ECU Measurement and Calibration Toolkit 1.0*
- The serial number of the product
- The version of the product
- Your name

- Your organization
- A computer ID that uniquely identifies your system

National Instruments uses this information to generate an activation code, which is used to activate the ECU M&C Toolkit on your system. National Instruments does not use this information for any other purpose. Refer to the *Privacy Policy* section for information on the National Instruments privacy policy regarding your personal information.

The Activation Wizard offers a variety of options you can use to obtain an activation code from National Instruments, including an automatic option through an Internet connection, or through email, by telephone, or by fax.

Activate ECU M&C Toolkit

ECU M&C Toolkit must be activated before using it, in accordance with its license agreement. To activate the ECU M&C Toolkit, you must first purchase a license. For information on purchasing licenses, contact your local National Instruments sales representative or visit `ni.com`.

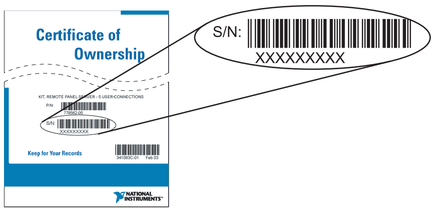
Once you have purchased a license, you can activate your product using the Activation Wizard.

Your National Instruments software then requires activation in order to access all features. Activation is simple and you can activate your software 24 hours a day, 7 days a week.

Complete the following steps to activate the ECU M&C Toolkit.

1. Locate your serial number.

   Your serial number uniquely identifies your purchase of NI software. You can find it on the *Certificate of Ownership* included in your software kit. If you subscribe to NI Developer Suite or Academic Software Solutions, use the original serial number you received with your initial purchase.



2. Install your software.

3. Launch the License Activation Wizard.

If you installed your software for the first time, the installer may launch the License Activation Wizard for you. Otherwise, launch your software and choose to activate when prompted.

OR

Perform the following steps if you do not receive a prompt:

   a. Launch the NI License Manager by selecting **Start»Programs» National Instruments»NI License Manager**.
   b. Click the **Activate** button on the toolbar.

The wizard will guide you through the activation process.

4. Save your activation code for future use (optional).

You can reactivate your software at any time. The activation wizard provides you with the option to receive an email confirmation of your activation code. To apply this activation code in the future, launch the Activation Wizard and choose to apply a 20-character activation code. If you reinstall your software on the same computer, the same activation code will work.

For more information on activation, refer to your product documentation, or visit `ni.com/activate`.

National Instruments uses activation to better support evaluation of our software, to enable additional software features, and to support license management in large organizations. To find out more about National Instruments software licensing, visit `ni.com/activate` to find frequently asked questions, resources, and technical support.

Terms

Table 1. Definition of Activation Terms

Serial Number	A 9-character, alphanumeric string that uniquely identifies your purchase of a single copy of software, included in your software kit on your *Certificate of Ownership*. The serial number for hardware products is printed either on the product box or on the device.
Computer ID or **Device ID**	A 16-character ID that uniquely identifies your computer or NI hardware, generated during the activation process.
Activation Code	A 20-character code that enables NI software to run on your computer, based on your serial number and computer ID. You generate and install an activation code by completing the activation process.

Moving Software After Activation

To transfer your software to another computer, install and reactivate it on the second computer. You are not prohibited from transferring your software from one computer to another. Because activation codes are unique to each computer, you will need a new activation code. Follow the steps on the previous page to acquire a new activation code and reactivate your software.

Volume License Program

National Instruments offers volume licenses through the NI Volume License Program. The NI Volume License Program makes managing software licenses and maintenance easy. For more information, refer to `ni.com/vlp`.

Online Activation

Activation is available on `ni.com/activate` 24 hours a day, 7 days a week. You can retrieve an activation code from any computer that has an Internet connection. NI does not require that the computer on which you run NI software have Internet or email access.

Home Computer Use

National Instruments permits you to use this software at home. Refer to the NI License Manager help file or the software end-user license agreement in the installer or online at `ni.com/legal/license` for more information.

Privacy Policy

National Instruments respects your privacy. For more information about the National Instruments activation information privacy policy, go to `ni.com/activate/privacy`.

Upon successful activation, you can use the product immediately.

Note If the ECU M&C Toolkit was in use before you began the activation process, you may need to restart it for the change to take effect.

Tip In the NI License Manager, products that have not been activated are denoted either by a yellow stoplight or a red stoplight, depending whether the product is in evaluation mode or is unusable. Activated products are denoted by a green stoplight.

LabVIEW Real-Time (RT) Configuration

LabVIEW Real-Time (RT) combines easy-to-use LabVIEW programming with the power of real-time systems. When you use a National Instruments PXI controller as a LabVIEW RT system, you can install a PXI CAN card and use the NI-CAN APIs to develop real-time applications. As with any other NI product for LabVIEW RT, you must download the ECU M&C Toolkit software to the LabVIEW RT system using the Remote Systems branch in MAX. For more information, refer to the LabVIEW RT documentation.

After you have installed the PXI CAN cards and downloaded the ECU M&C Toolkit software to the LabVIEW RT system, you must verify the installation.

To use the ECU M&C Toolkit on the LabVIEW RT system, you must also download the ASAM MCD 2MC database file to the RT target. The LabVIEW Real-Time Engine that runs on the PXI LabVIEW Real-Time controller supports a File Transfer Protocol (FTP) server. You can access the LabVIEW RT target FTP server using any standard FTP utility for transferring files to and from the hard drive or compact flash. The following sections demonstrate how to transfer files from and to your LabVIEW Real-Time target using various FTP clients.

DOS command prompt

You can run a native FTP client from the DOS command prompt on a Windows PC. To open the FTP client, click **Start»Run** to open the user-command dialog box. Type `command`, and click **Enter**. This opens a window with a DOS prompt.

Then use the following table to enter a sequence of commands that may be used to access the FTP server of your RT target.

Note *w.x.y.z* represents the IP address of the RT target in this document.

Table 2-1. Example of FTP Transfer

Command	Result
ftp	Open a connection to the FTP server.
open *w.x.y.z*	
(username)	Enter your username and password here or press the Enter key twice if these security settings have not been applied.
(password)	
help	View a list of commands.
cd ni-rt\system\www	Change to the desired directory.
dir	View the files present.
get index.htm c:\index.htm	Copy the file.
cd \	Change directory back to the root (c:\).
cd d:	Change directories to the external compact flash.
put c:\index.htm index.htm	Copy the file from the FTP client machine to the target.
dir	Verify the copied file on the target.
cd c:	Change directory back to the internal compact flash or hard drive.
quit	Disconnect from the FTP server.

Web Browsers

You can also use Internet Explorer or Netscape Navigator to ftp files to and from the controller. This is an easier method of transfer, since there is no need to learn ftp commands—instead the files are simply copied and pasted as they would be in a Windows Explorer window. The disadvantage of this method is that Internet Explorer sometimes caches old information, so you will need to refresh occasionally.

If *w.x.y.z* is the IP address of your RT target, open Internet Explorer to access the hard drive or internal compact flash, or type the following in the address field:

```
ftp://w.x.y.z/
```

If a username and password are required, then use the following format:

```
ftp://username:password@w.x.y.z/
```

To access the external compact flash, open Internet Explorer and type the following in the address field:

```
ftp://w.x.y.z/d:/
```

To enter a directory, double-click on its icon. Right-click on a file or folder and choose cut, copy, paste or delete to perform those actions.

LabVIEW Real-Time Graphical File Transfer Utility

LabVIEW Real-Time Module versions 7.0 and later include a File Transfer Utility that can be used to access your RT target. This method helps you avoid the caching problem encountered when using web browsers. You can find this utility in the Measurement and Automation Explorer (MAX). To open the utility, right-click on the desired RT target under the **Remote Systems** list and choose **File Transfer**, as shown in Figure 2-2.

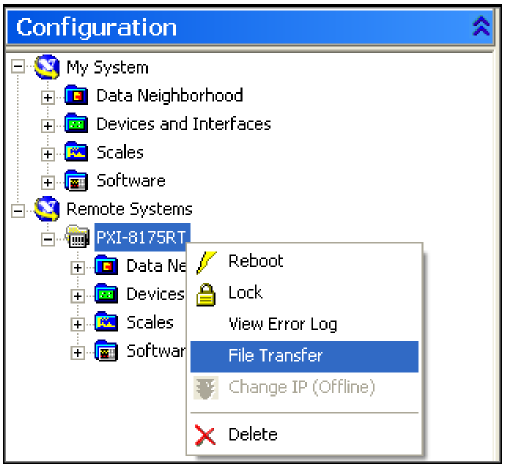
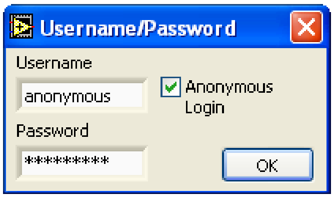


Figure 2-2. FTP Utility Access in MAX

At this point, you will be prompted for a username and password. If these security features have not been enabled, check the **Anonymous Login** box as shown in Figure 2-3.

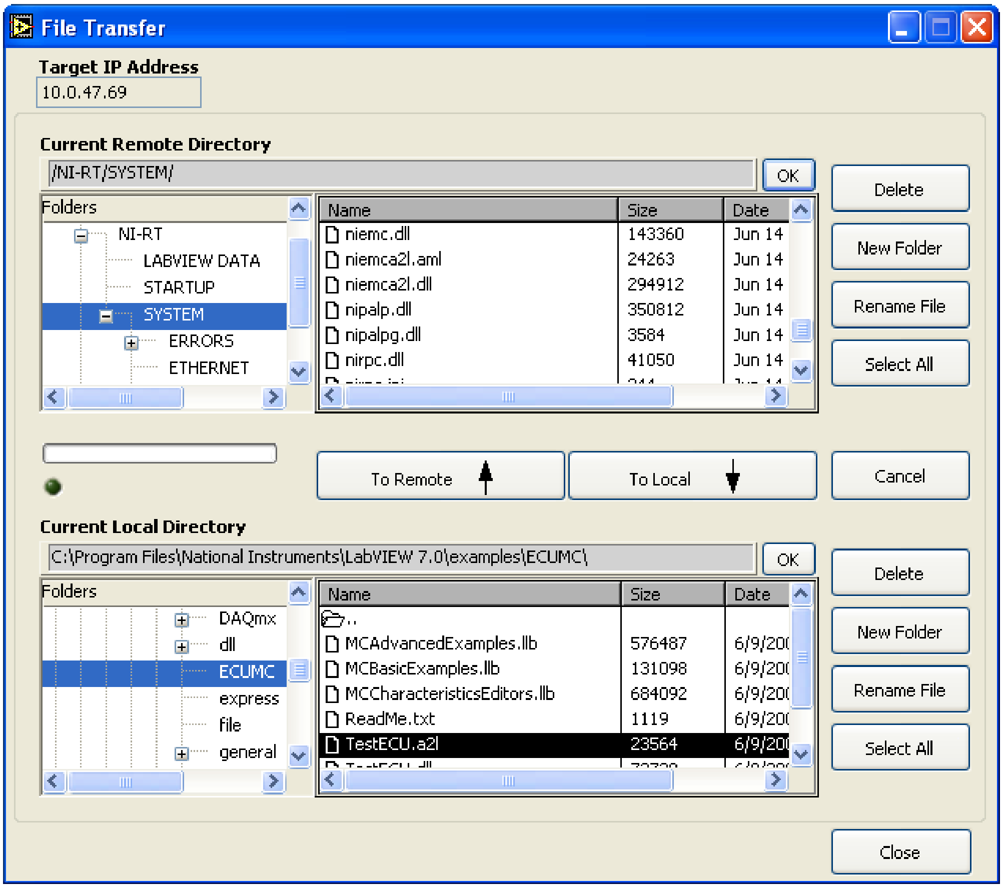


Username: anonymous
[✓] Anonymous Login
Password: *********

Figure 2-3. FTP Login Dialog Box

The upper section of the utility interface shows the current directory and contents on the remote RT target, while the lower section gives information for the host or local machine. To copy a file (`TestECU.a2l`, for instance) to the RT target, complete the following steps, referring to Figure 2-4 for details.

1. In the Current Directory section, navigate through the tree structure to the System folder.
2. In the local directory section, navigate through the tree structure to the location of the file you want to transfer and highlight the file.
3. Click the **To Remote** button to copy the file.



Target IP Address
10.0.47.69

Current Remote Directory
/NI-RT/SYSTEM/

NI-RT
LABVIEW DATA
STARTUP
SYSTEM
ERRORS
ETHERNET

Name	Size	Date
niemc.dll	143360	Jun 14
niemca2l.aml	24263	Jun 14
niemca2l.dll	294912	Jun 14
nipalp.dll	350812	Jun 14
nipalpg.dll	3584	Jun 14
nirpc.dll	41050	Jun 14

Current Local Directory
C:\Program Files\National Instruments\LabVIEW 7.0\examples\ECUMC\

DAQmx
dll
ECUMC
express
file
general

Name	Size	Date
..		
MCAdvancedExamples.llb	576487	6/9/20
MCBasicExamples.llb	131098	6/9/20
MCCharacteristicsEditors.llb	684092	6/9/20
ReadMe.txt	1119	6/9/20
TestECU.a2l	23564	6/9/20

Figure 2-4. Transferring Files With the FTP Utility

LabVIEW

You also can use LabVIEW to programmatically access the FTP server of a LabVIEW Real-Time target.

The **DataSocket Read** function has the ability to read raw text, tabbed text, and `.wav` files from an FTP server. For more information on this, refer to the *LabVIEW User Manual*.

The *LabVIEW Internet Developers Toolkit* allows you to send files or raw data to an FTP server, as well as sending emails and adding security to your web-based applications.

Hardware and Software Requirements

The ECU M&C Toolkit requires National Instruments NI-CAN hardware Series 1 or 2 and the NI-CAN driver software version 2.3 or later installed.

3 Application Development

This chapter explains how to develop an application using the ECU M&C API.

Choose the Programming Language

The programming language you use for application development determines how to access the ECU M&C Toolkit APIs.

LabVIEW

ECU M&C Toolkit functions and controls are available in the LabVIEW palettes. In LabVIEW, the **ECU M&C Toolkit** palette is located within the top-level **NI Measurements** palette.

The reference for each ECU M&C Toolkit Channel API function is in Chapter 5, *ECU M&C API for LabVIEW*. To access the reference for a function from within LabVIEW, press <Ctrl-H> to open the Help window, click the appropriate ECU M&C function, and then follow the link. The ECU M&C Toolkit software includes a full set of examples for LabVIEW. These examples teach programming basics as well as advanced topics. The example help describes each example and includes a link you can use to open the VI.

LabWindows/CVI

Within LabWindows/CVI™, the ECU M&C Toolkit function panel is in **Libraries»ECU Measurement and Calibration Toolkit**. Like other LabWindows/CVI function panels, the ECU M&C Toolkit function panel provides help for each function and the ability to generate code. The reference for each API function is located in Chapter 6, *ECU M&C API for C*. You can access the reference for each function directly from within the function panel. The header file for the ECU M&C Toolkit APIs is `niemc.h`. The library for the ECU M&C Toolkit APIs is `niemc.lib`. The toolkit software includes a full set of examples for LabWindows/CVI. The examples are installed in the LabWindows/CVI directory under `samples\ecumc`. Each example provides a complete LabWindows/CVI project (`.prj` file).

A description of each example is provided in comments at the top of the `.c` file.

Visual C++ 6

The ECU M&C Toolkit software supports Microsoft Visual C/C++ 6. The header file and library for Visual C/C++ 6 are in the `MS Visual C` folder of the ECU M&C Toolkit folder. The typical path to this folder is `\ProgramFiles\National Instruments\ECU Measurement and Calibration Toolkit\MS Visual C`. To use the ECU M&C API, include the `niemc.h` header file in the code, then link with the `niemc.lib` library file.

For C applications (files with a `.c` extension), include the header file by adding a `#include` to the beginning of the code, like this:

```
#include "niemc.h"
```

For C++ applications (files with a `.cpp` extension), define `_cplusplus` before including the header, like this:

```
#define _cplusplus

#include "niemc.h"
```

The `_cplusplus` define enables the transition from C++ to the C language functions.

The reference for each API function is in Chapter 6, *ECU M&C API for C*. You can find examples for the C language in the `MS Visual C` subfolder of the ECU M&C Toolkit folder. Each example is in a separate folder. A description of each example is in comments at the top of the `.c` file. At the command prompt, after setting MSVC environment variables (such as with MS `vcvars32.bat`), you can build each example using a command such as:

```
cl /I.. measure.c ..\niemc.lib
```

Other Programming Languages

The ECU M&C Toolkit software does not provide formal support for programming languages other than those described in the preceding sections. If the programming language provides a mechanism to call a Dynamic Link Library (DLL), you can create code to call ECU M&C Toolkit functions. All functions for the ECU M&C API are located in `niemc.dll`. If the programming language supports the Microsoft Win32

APIs, you can load pointers to ECU M&C Toolkit functions in the application. The following text demonstrates use of the Win32 functions for C/C++ environments other than Visual C/C++ 6. For more detailed information, refer to Microsoft documentation.

The following C language code fragment illustrates how to call Win32 `LoadLibrary` to load the DLL for the ECU M&C API:

```
#include <windows.h>

#include "niemc.h"

HINSTANCE NiMcLib = NULL;

NiMcLib = LoadLibrary("niemc.dll");
```

Next, the application must call the Win32 `GetProcAddress` function to obtain a pointer to each ECU M&C Toolkit function that the application will use. For each function, you must declare a pointer variable using the prototype of the function. For the prototypes of each ECU M&C Toolkit function, refer to Chapter 6, *ECU M&C API for C*.

Before exiting the application, you must unload the ECU M&C Toolkit DLL as follows:

```
FreeLibrary (NiMcLib);
```

Debugging An Application

The NI-Spy tool monitors function calls to the ECU M&C API, to help in the debugging of the application. To launch this tool, open the **Software** branch of the MAX configuration tree, right-click **NI Spy**, and select **Launch NI Spy**.

If you have more than one National Instruments driver installed on your computer, you can specify which APIs you want to monitor at any time. By default, all installed APIs are enabled. To select the APIs to monitor, select **Spy»Options**, select the **View Selections** tab, and select the desired APIs under **Installed API Choices**.

Using the ECU M&C API

This chapter helps you get started with the Support ECU M&C API.

Structure of the ECU M&C API

The ECU M&C API is divided into two main categories of API functions, high-level Channel-based functions and generic low-level CCP functions. The ECU M&C Channel API functions provide an easy way to access ECU internal data through named channels. The ECU M&C CCP API functions provide direct access to the CCP command on a very low programming level. Figure 4-1 outlines the two ECU M&C API categories.

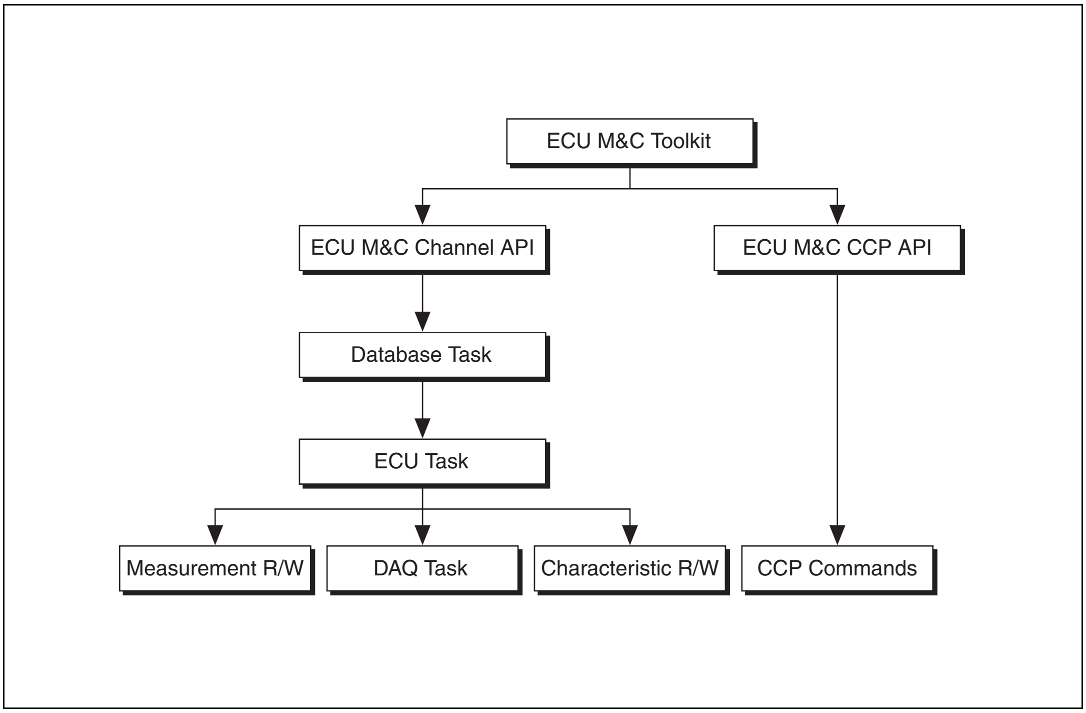


Figure 4-1. ECU Architectural Overview

ECU M&C Channel API

Within the ECU M&C Channel API there are a number of ways to access memory content in an ECU. The starting point is always the creation of a database task, which is the link to a valid ASAM MCD 2MC database file (`*.A2L` file). With the database task reference it is possible to create an ECU task reference, which links to the selected ECU. Depending on the application scenario, the ECU task reference can be used for the following:

- Creation of a Measurement task to measure ECU internal data continuously or on demand
- Direct read/write of 0- to 2-dimensional characteristics
- Read/write of single Measurement values on demand

What is an ECU Measurement?

An ECU Measurement, called *ECU Data Acquisition (DAQ)* in the CCP specification, is a definition of specific procedures and CAN messages sent from the slave device (ECU) to the master device for fast data acquisition (DAQ).

What is an ECU Characteristic?

An ECU Characteristic represents an ECU internal memory range with defined access methods through the CCP protocol. The memory range of a single Characteristic can be structured in three ways:

- 0-dimensional—a single value
- 1-dimensional—a curve of values
- 2-dimensional—a field of values

A Characteristic may be defined as read-only or read and write accessible.

ECU M&C CCP API

The ECU M&C Channel API does not expose the method used for ECU memory access. However, some applications may need specific CCP command sequences or custom designed commands which are not supported by the CCP protocol. For these applications, the ECU M&C CCP API functions provides the functionality to access ECU information at a very low level.

Basic Programming Model

The flowchart in Figure 4-2 illustrates the process to initiate communication to an ECU with the ECU M&C Channel API. A description of each step in the decision process follows the flowchart.

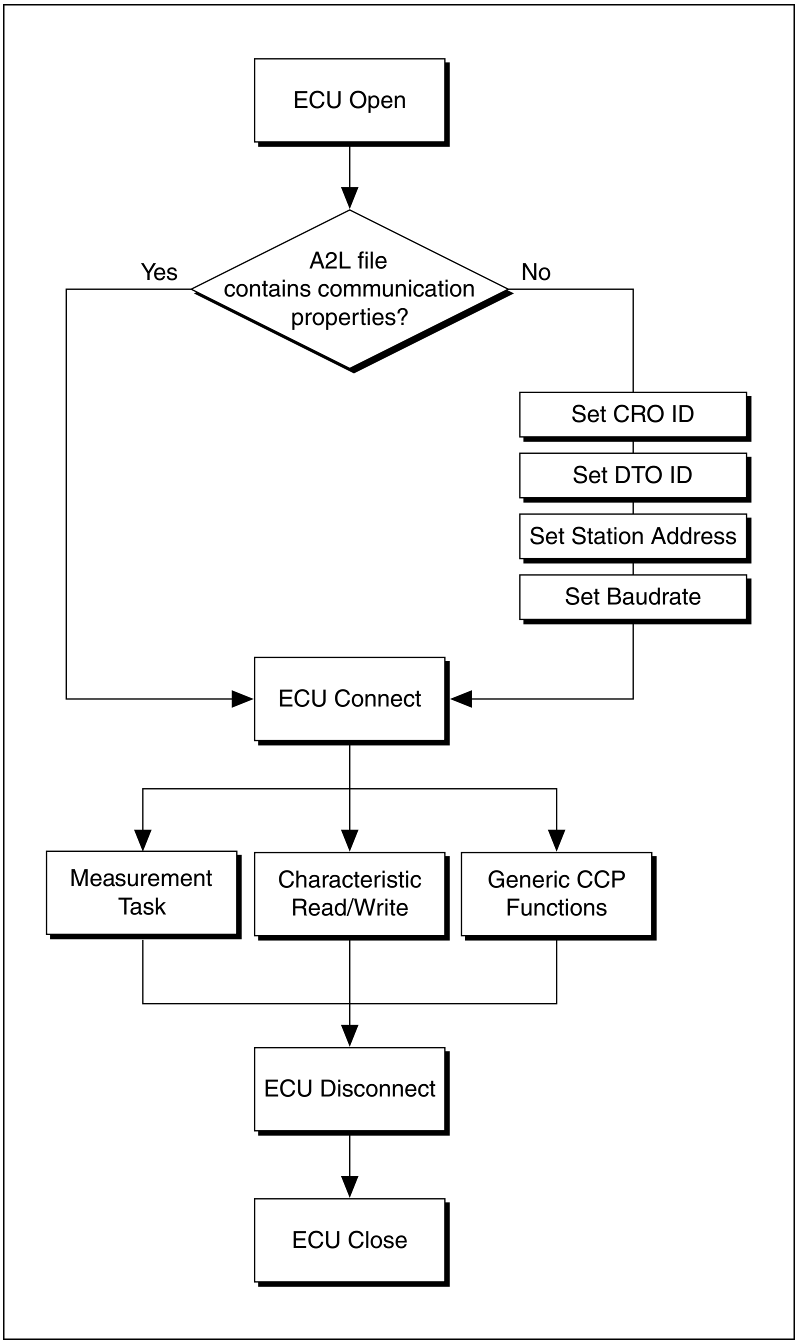


Figure 4-2. ECU Communication Decision Chart

ECU Open

The **ECU Open** function combines the opening of a selected ASAM MCD 2MC database file with the `.A2L` file extension and the selection of a stored ECU name. The required parameters are the ASAM MCD 2MC database path and filename, and the dedicated CAN interface. The CAN interface is used for communication with the ECU.

The function to open and select an ECU is **MC ECU Open.vi** in LabVIEW, and `mcDatabaseOpen` followed by `mcECUSelect` in C.

Note The import of ASAM MCD 2MC database files into MAX is not supported.

ASAM MCD 2MC Communication Properties

If your ASAM MCD 2MC database file already contains communication properties, you can directly open the communication to your selected ECU.

If the CCP communication properties are not stored in the ASAM MCD 2MC file, the communication properties must be manually set. To establish communication through CCP, the target ECU slave should be addressed by setting the following properties.

CRO ID

The **CRO ID** (**C**ommand **R**eceive **O**bject) is used to send commands and data from the host to the slave device.

DTO ID

The **DTO ID** (**D**ata **T**ransmission **O**bject) is used by the ECU to respond to CCP commands, and send data and status information to the CCP master.

Station Address

CCP is based on the idea that several ECUs can share the same CAN Arbitration IDs for CCP communication. To avoid communication conflicts, CCP defines a **Station Address** that has to be unique for all ECUs sharing the same CAN Arbitration IDs. Unless an ECU has been addressed by its **Station Address**, the ECU must not react to CCP commands sent by the CCP master.

Baudrate

The **baudrate** property may be missing in an A2L database file and can be set explicitly within the application. This property provides the baud rate at which communication will occur, and applies to all tasks initialized with the interface. You can specify one of the predefined baud rates, or specify advanced baud rates which refer to the settings of the Bit Timing Register 0 (BTR0) and 1 (BTR1). For more information, refer to the **Interface Properties** dialog in MAX, or the *NI-CAN Hardware and Software Manual*. The baud rate is originally set within MAX.

ECU Connect

The **ECU Connect** function establishes communication to the selected ECU through CCP using the CCP CONNECT command. It establishes a logical connection to an ECU. Unless a slave device (ECU) is unconnected, it must not execute or respond to any command sent by the application. The only exception to this rule is the Test command, to which the CCP slave with the specific address may return an acknowledgement. Only a single CCP slave can be connected to the application at a time. After a successful **ECU Connect** you can create a Measurement Task or read/write a Characteristic.

The function to open and select an ECU is **MC ECU Connect.vi** in LabVIEW, and `mcECUConnect` in C.

ECU Disconnect

The **ECU Disconnect** function permanently disconnects the specified CCP slave and ends the measurement and calibration session. When the measurement and calibration session is terminated, all CCP DAQ lists of the device will be stopped and cleared, and the protection masks of the device will be set to their default values.

The function to disconnect an ECU is **MC ECU Disconnect.vi** in LabVIEW, and `mcECUDisconnect` in C.

ECU Close

The **MC ECU Close** function deselects the ECU and closes the remaining database reference handle. **MC ECU Close** must always be the final M&C function call. If you do not use **MC ECU Close**, the remaining task configurations can cause problems in the execution of subsequent M&C applications.

The function to close an ECU is **MC ECU Close.vi** in LabVIEW. To deselect the ECU and close the database reference handle in C, call the function `mcECUDeselect` followed by `mcDatabaseClose`.

Characteristic Read and Write

Access Characteristics

To access the characteristics of an ECU you must select and connect to the specific ECU through the procedure given above. The function to open and select an ECU is **MC ECU Open.vi** in LabVIEW, or `mcDatabaseOpen` followed by `mcECUSelect` in C. Once the ECU has been connected, pass an ECU Reference handle (**ECU ref out** in LabVIEW, `ECURefNum` in C) before any additional actions can be performed.

Characteristic Read

The application must call the **Read Characteristic** function to obtain scaled floating point samples. The application typically calls **Read Characteristic** on demand. Calling **Read Characteristic** in a loop can cause significant CAN network traffic, as Characteristics may contain large amounts of data.

The function to read 0- to 2-dimensional characteristics is **MC Characteristic Read.vi** in LabVIEW and `mcCharacteristicRead` in C. The function to read single double values as characteristics is **MC Characteristic Read Single Value.vi** in LabVIEW and `mcCharacteristicReadSingleValue` in C.

Before reading a Characteristic it may be helpful to verify the dimension of the Characteristic based on the definition in the ASAM MCD 2MC database file. Depending on the dimension of the Characteristic, use the appropriate **Read** function for reading a double, a 1D array of doubles, or a 2D array of doubles.

The function to verify a dimension of a named Characteristic is **MC Get Property.vi** with the parameter **Characteristic/Dimension** in LabVIEW and `mcGetProperty` with the parameter `mcPropChar_Dimension` in C.

Characteristic Write

The application must call the **Write Characteristic** function to output-scaled floating-point samples. The application typically calls **Write Characteristic** on demand. Calling **Write Characteristic** in a loop can cause significant CAN network traffic, as Characteristics may contain large amounts of data.

The function to write a Characteristic is **MC Characteristic Write.vi** in LabVIEW and `mcCharacteristicWrite` in C.

Before writing a Characteristic it may be helpful to verify the dimension of the Characteristic based on the definition in the ASAM MCD 2MC database file. Depending on the dimension of the Characteristic, use the appropriate **Write** function for writing a double, a 1D array of doubles, or a 2D array of doubles.

The function to verify a dimension of a named Characteristic is **MC Get Property.vi** with the parameter **Characteristic/Dimension** in LabVIEW and `mcGetProperty` with the parameter `mcPropChar_Dimension` in C.

Measurement Task

To create a Measurement task you need to select available Measurement signals from an ASAM MCD 2MC database file. Create a valid ECU Reference handle as described in the *Access Characteristics* section.

The flowchart in Figure 4-3 shows the process to perform an ECU Measurement task. A description of each step in the decision process follows the flowchart.

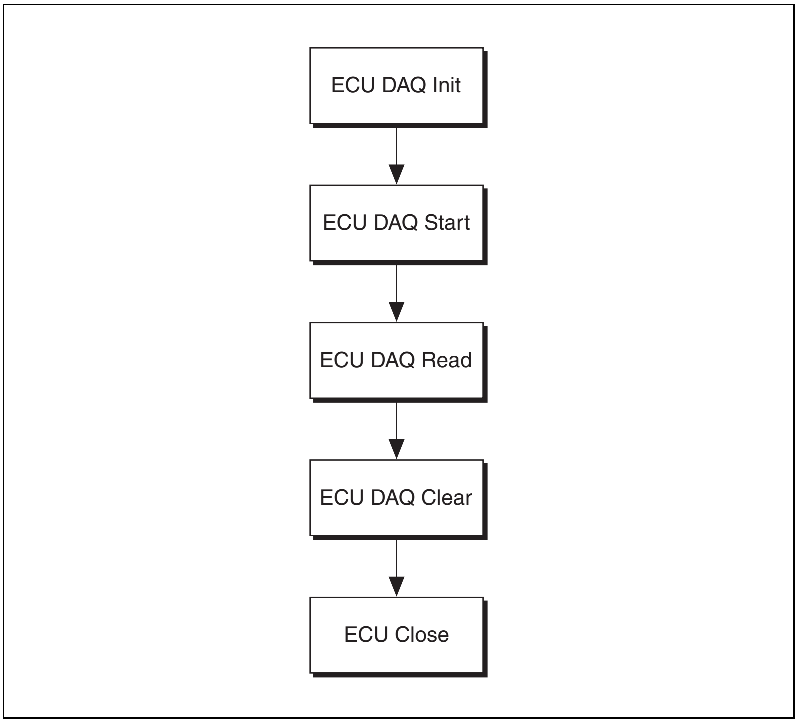


Figure 4-3. ECU Measurement Setup Flowchart

DAQ Initialize

The **DAQ Initialize** function initializes a list of Measurement channels as a single Measurement task. The communication for that Measurement task is started by the first **DAQ Read** function. The **DAQ Initialize** function is **MC DAQ Initialize.vi** in LabVIEW and `mcDAQInitialize` in other languages.

The **DAQ Initialize** function uses the following input parameters:

Measurement list

Specifies the list of channels for the task, with one string for each channel.

ECU Reference handle

Typically, the **ECU Reference handle** is created by opening the ASAM MCD 2MC database using the **ECU Open** function, then connecting to an ECU using the **ECU Connect** function.

Mode

Specifies the input mode to use for the task. This determines the data transfer for the task (that is, Polling or DAQ List).

SampleRate

Specifies the sampling rate for a specific DAQ List. The sample rate is specified in Hertz (samples per second). For more information, refer to the *DAQ Read* section.

DTO ID

The **DTO ID** (**D**ata **T**ransmission **O**bject) is used by the ECU to respond to CCP commands and send data and status information to the CCP master.

DAQ Start

The optional function **DAQ Start** starts the transmission of the DAQ lists for an M&C Measurement task. If you do not specify **MC DAQ Start.vi** before your first **DAQ Read** function, **MC DAQ Start.vi** will be implicitly performed by the first **DAQ Read call**. After you start the transmission of the DAQ lists, you can no longer change the configuration of the Measurement task with **Set Property**.

The function to start a DAQ List is **MC DAQ Start.vi** in LabVIEW, and `mcDAQStart` in C.

DAQ Read

The application must call the **DAQ Read** function to obtain floating-point samples. The application typically calls **DAQ Read** in a loop until done. The **Read** function is **MC DAQ Read.vi** in LabVIEW (all types that do not end in **Time & Dbl**) and `mcDAQRead` in other languages.

The behavior of **Read** depends on the initialized sample rate and the selected mode.

sample rate = 0

DAQ Read returns a single sample from the most recent message(s) received from the network. One sample is returned for every channel in the **DAQ Initialize** list.

Figure 4-4 shows an example of **DAQ Read** with a sample rate = 0. *A*, *B*, and *C* represent messages for the initialized channels. *def* represents the default value 0. If no message is received since the start of the application, the default value 0 is returned, along with a warning.

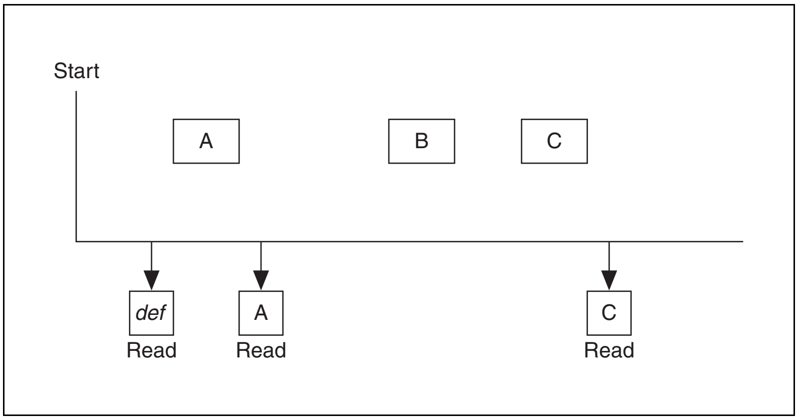


Figure 4-4. Example of Read With Sample Rate = 0

sample rate > 0

DAQ Read returns an array of samples for every channel in the **DAQ Initialize** list. Each time the clock ticks at the specified rate, a sample from the most recent message(s) is inserted into the arrays. In other words, the samples are repeated in the array at the specified rate until a new message is received. By using the same sample rate with NI-DAQ Analog Input channels or NI-DAQmx Analog Input channels, you can compare ECU DAQ and NI-DAQ/NI-DAQmx samples over time.

Figure 4-5 shows an example of **DAQ Read** with a sample rate > 0. *A*, *B*, and *C* represent messages for the initialized channels. *delta-t* represents the time between samples as specified by the sample rate. *def* represents the default value 0.

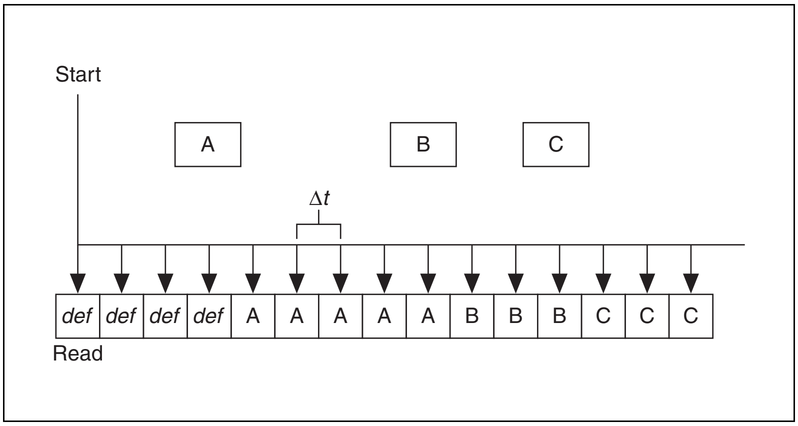


Figure 4-5. Example of Read With Sample Rate > 0

DAQ Clear

DAQ Clear must always be the final function called for a specific Measurement task. If you do not use **DAQ Clear**, the remaining Measurement task configuration can cause problems in the execution of subsequent ECU M&C applications. Because this function clears the Measurement task, the Measurement task reference will be transferred into an ECU reference task handle. To change properties of a running Measurement task, use **DAQ Stop** to stop the task, **Set Property** to change the desired DAQ property, and then **DAQ Start** to restart the Measurement task again.

The function to clear a DAQ list is **MC DAQ Clear.vi** in LabVIEW, and `mcDAQClear` in C.

Additional Programming Topics

The following sections provide information you can use to extend the basic programming model.

Get Names

If you are developing an application that another person will use, you may not want to specify a fixed channel list for a Measurement task, or a fixed channel for a Characteristic in the application. Ideally, you want the end-user to select the channels of interest from user interface controls, such as list boxes. The **Get Names** function queries an ASAM MCD 2MC database and returns a list of all channels in that database regarding the selected query mode. You can use this list to populate user-interface controls. The end-user can then select channels from these controls, avoiding the need to type in each name. Once the user makes the selections, the application can pass the resulting list to the appropriate function, such as **DAQ Initialize**, for an ECU Measurement channel list. The **Get Names** function is **MC Get Names.vi** in LabVIEW and `mcGetNames` in C.

Set/Get Properties

If you need to change particular parameters within an application, such as the **DTO ID**, use the following calling sequence:

1. Initialize the Measurement task as stopped. The **Initialize** function is **MC DAQ Initialize.vi** in LabVIEW and `mcDAQInitialize` in C.
2. Use **Set Property** to specify the new value for the **DTO_ID** property. The **Set Property** function is **MC Set Property.vi** in LabVIEW and `mcSetProperty` in C.
3. Start the Measurement task with the **Start** function. The **Start** function is **MC DAQ Start.vi** in LabVIEW and `mcDAQStart` in C.

After the task is started, you may need to change properties again. To change properties within the application, use the **DAQ Stop** function to stop the Measurement task, **Set Property** to change properties, and then start the task again.

You also can use the **Get Property** function to get the value of any property. The **Get Property** function returns values whether the task is running or not. The **Get Property** function is **MC Get Property.vi** in LabVIEW and `mcGetProperty` in C.

Generic CCP Functions

The generic ECU M&C CCP API functions provide direct access to the CCP command on a very low programming level. For further information for the use and parameters of the CCP commands, refer to the *CAN Calibration Protocol Specification, Version 2.1*. Table 4-1 gives an overview of which CCP command is provided by which LabVIEW VI or C function.

Table 4-1. Overview of the CCP Commands with Related VIs and C Functions

CCP Command	LabVIEW VI Name	C Function Name
ACTION_SERVICE	**MC CCP Action Service.vi**	`mcCCPActionService`
BUILD_CHKSUM	**MC CCP Build Checksum.vi**	`mcCCPBuildChecksum`
CLEAR_MEMORY	**MC CCP Clear Memory.vi**	`mcCCPClearMemory`
DIAG_SERVICE	**MC CCP Diag Service.vi**	`mcCCPDiagService`
DNLOAD	**MC CCP Download.vi**	`mcCCPDownload`
GET_ACTIVE_CAL_PAGE	**MC CCP Get Active Cal Page.vi**	`mcCCPGetActiveCalPage`
GET_S_STATUS	**MC CCP Get Session Status.vi**	`mcCCPGetSessionStatus`
GET_CCP_VERSION	**MC CCP Get Version.vi**	`mcCCPGetVersion`
MOVE	**MC CCP Move Memory.vi**	`mcCCPMoveMemory`
PROGRAM	**MC CCP Program.vi**	`mcCCPProgram`
SELECT_CAL_PAGE	**MC CCP Select Cal Page.vi**	`mcCCPSelectCalPage`
SET_S_STATUS	**MC CCP Set Session Status.vi**	`mcCCPSetSessionStatus`
UPLOAD	**MC CCP Upload.vi**	`mcCCPUpload`

5 ECU M&C API for LabVIEW

This chapter lists the LabVIEW VIs for the ECU M&C API and describes the format, purpose, and parameters for each VI. The VIs in this chapter are listed alphabetically. Unless otherwise stated, each VI suspends execution of the calling thread until it completes.

Section Headings

The following are section headings found in the ECU M&C API for LabVIEW VIs.

Purpose

Each VI description includes a brief statement of the purpose of the VI.

Format

The format section describes the format of each VI.

Input and Output

The input and output parameters for each VI are listed.

Description

The description section gives details about the purpose and effect of each VI.

List of VIs

The following table is an alphabetical list of the ECU M&C Toolkit VIs.

Table 5-1. ECU M&C API VIs for LabVIEW

Function	Purpose
MC CCP Action Service.vi	Calls an implementation-specific action service on the ECU.
MC CCP Build Checksum.vi	Calculates a checksum over a defined memory range.
MC CCP Calc Checksum.vi	Calculates the checksum of a data block.

Table 5-1. ECU M&C API VIs for LabVIEW (Continued)

Function	Purpose
MC CCP Clear Memory.vi	Clears the contents of a specified memory block.
MC CCP Diag Service.vi	Calls a diagnostic service on the ECU.
MC CCP Download.vi	Downloads data to an ECU.
MC CCP Generic.vi	Sends a generic CCP command.
MC CCP Get Active Cal Page.vi	Retrieves the ECU Memory Transfer Address pointer to the calibration data page.
MC CCP Get Result.vi	Uploads requested data.
MC CCP Get Session Status.vi	Retrieves the current calibration status of the ECU.
MC CCP Get Version.vi	Retrieves the version of the CCP implemented in the ECU.
MC CCP Move Memory.vi	Moves a memory block on the ECU.
MC CCP Program.vi	Programs a memory block on the ECU.
MC CCP Select Cal Page.vi	Sets the beginning of the calibration data page.
MC CCP Set Session Status.vi	Updates the ECU with the current state of the calibration session.
MC CCP Upload.vi	Uploads data from an ECU.
MC Characteristic Read.vi	Reads data from a named Characteristic on the ECU which is identified by the ECU Reference handle. The Poly VI returns a specific double, 1D, or 2D double array.
MC Characteristic Read Single Value.vi	Reads a value from a named Characteristic on the ECU which is identified by the ECU Reference handle.
MC Characteristic Write.vi	Writes the value(s) of a named Characteristic to an ECU identified by the ECU ref handle. The Poly VI writes the selected type double, 1D or 2D array.
MC Characteristic Write Single Value.vi	Writes a value to a named Characteristic on the ECU.
MC DAQ Clear.vi	Stops communication for the Measurement task and then clears the configuration.
MC DAQ Initialize.vi	Initializes a Measurement task for the specified Measurement channel list.

Table 5-1. ECU M&C API VIs for LabVIEW (Continued)

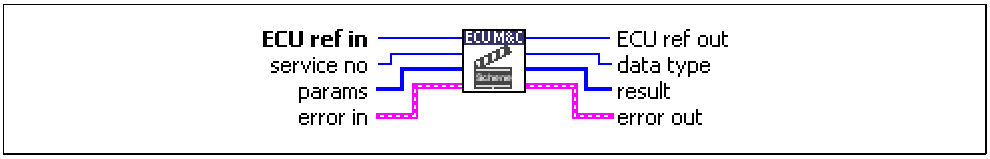
Function	Purpose
MC DAQ Read.vi	Reads samples from a Measurement task. Samples are obtained from received CAN messages.
MC DAQ Start.vi	Starts transmission of the DAQ lists for the specified Measurement task.
MC DAQ Stop.vi	Stops transmission of the DAQ lists for the specified Measurement task.
MC Database Close.vi	Closes a specified A2L Database.
MC Database Open.vi	Opens a specified A2L Database.
MC ECU Close.vi	Closes the selected ECU and the associated A2L database.
MC ECU Connect.vi	Establishes the communication to the selected ECU through the CCP protocol. After a successful ECU Connect you can create a Measurement Task or read/write a Characteristic.
MC ECU Deselect.vi	Deselects an ECU and invalidates the ECU reference handle.
MC ECU Disconnect.vi	Permanently disconnects the CCP communication to the selected ECU and ends the calibration session.
MC ECU Open.vi	Opens a specified A2L database and selects the first ECU found in the database. If there are several ECUs stored in the A2L database use the Database Open and ECU Select VIs.
MC ECU Select.vi	Selects an ECU from the names stored in an A2L database.
MC Get Names.vi	Gets an array of ECU names, Measurement names, Characteristic names, or Event names from a specified A2L database file.
MC Get Property.vi	Gets a property for the object referenced by the **reference in** terminal. The poly VI selection determines the property to get.
MC Measurement Read.vi	Reads a single Measurement value from the ECU.
MC Measurement Write.vi	Writes a single Measurement value to the ECU.
MC Set Property.vi	Sets a property for the specified A2L database file, Measurement task or Characteristic. The poly VI selection determines the property to set.

MC CCP Action Service.vi

Purpose

Calls an implementation-specific action service on the ECU.

Format



Input

U32 **ECU ref in** is the task reference which links to the selected ECU. This reference is originally returned from **MC ECU Open.vi** or **MC ECU Select.vi**, and then wired through subsequent VIs.

U16 **Service No** determines the service that will be executed inside the ECU. For more information about the services that are implemented in the ECU, refer to the documentation for the ECU.

U8 **Params** passes an array to the ECU that might be needed by the ECU to run the service. Since this VI has no knowledge about how the data will be interpreted by the ECU, you are responsible for providing the data in the correct byte ordering.

Error in is a cluster which describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

TF **status** is `TRUE` if an error occurred. This VI is not executed when **status** is `TRUE`.

I32 **code** is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

abc **source** identifies the VI where the error occurred.

Output

ECU ref out is the same as **ECU ref in**. Wire the task reference to subsequent VIs for this task.

Data type is a data type qualifier that determines the data format of the result.

Result indicates the amount of data that can be uploaded from the ECU as a result of the execution of the service.

Error out describes error conditions. If the **Error in** cluster indicated an error, the **Error out** cluster contains the same information. Otherwise, **Error out** describes the error status of this VI.

status is `TRUE` if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

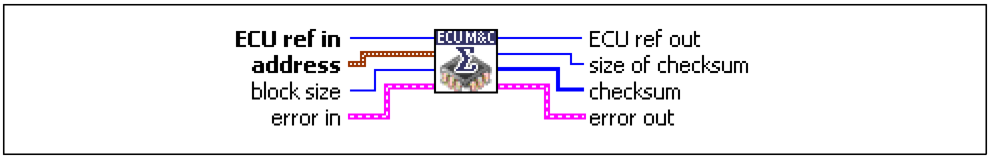
MC CCP Action Service.vi implements the CCP command ACTION_SERVICE. The ECU carries out the requested service and automatically uploads the requested action service return information.

MC CCP Build Checksum.vi

Purpose

Calculates a checksum over a defined memory range.

Format



Input

U32 **ECU ref in** is the task reference which links to the selected ECU. This reference is originally returned from **MC ECU Open.vi** or **MC ECU Select.vi**, and then wired through subsequent VIs.

Address is a cluster which contains the following values.

U32 **Address** specifies the address part of the source address.

U8 **Extension** contains the extension part of the source address.

U32 **Block size** determines the size of the block for which the checksum has to be calculated.

Error in is a cluster which describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

TF **status** is `TRUE` if an error occurred. This VI is not executed when **status** is `TRUE`.

I32 **code** is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

abc **source** identifies the VI where the error occurred.

Output

ECU ref out is the same as **ECU ref in**. Wire the task reference to subsequent VIs for this task.

Size of checksum returns the size, in bytes, of the calculated checksum.

Checksum is the calculated checksum.

Error out describes error conditions. If the **Error in** cluster indicated an error, the **Error out** cluster contains the same information. Otherwise, **Error out** describes the error status of this VI.

status is `TRUE` if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

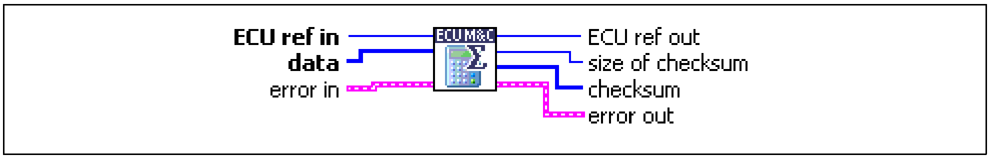
MC CCP Build Checksum.vi is used to calculate the checksum of the specified memory block inside the ECU starting at the selected Memory Transfer Address. The checksum algorithm is not specified by CCP and the checksum algorithm may be different on different devices.

MC CCP Calc Checksum.vi

Purpose

Calculates the checksum of a data block.

Format



Input

U32 **ECU ref in** is the task reference which links to the selected ECU. This reference is originally returned from **MC ECU Open.vi** or **MC ECU Select.vi**, and then wired through subsequent VIs.

[U8] **Data** is a byte array over which the checksum calculation will be performed.

Error in is a cluster which describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

TF **status** is `TRUE` if an error occurred. This VI is not executed when **status** is `TRUE`.

I32 **code** is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

abc **source** identifies the VI where the error occurred.

Output

U32 **ECU ref out** is the same as **ECU ref in**. Wire the task reference to subsequent VIs for this task.

U8 **Size of checksum** returns the size, in bytes, of the calculated checksum.

Checksum is the calculated checksum.

Error out describes error conditions. If the **Error in** cluster indicated an error, the **Error out** cluster contains the same information. Otherwise, **Error out** describes the error status of this VI.

status is `TRUE` if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

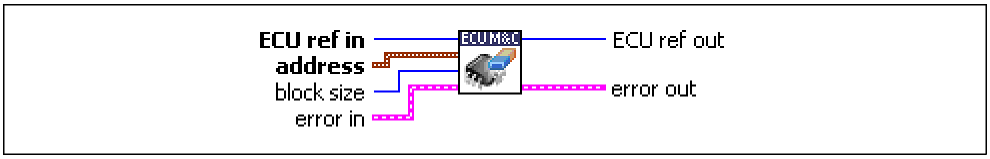
MC CCP Calc Checksum.vi implements a checksum calculation over a given data block. The checksum algorithm will be performed over a dedicated checksum function provided by a specific DLL. The Checksum DLL is defined in the A2L data base and can be changed by the application by the **MC Set Property.vi** using the **Checksum DLL Name** property.

MC CCP Clear Memory.vi

Purpose

Clears the contents of a specified memory block.

Format



Input

U32 **ECU ref in** is the task reference which links to the selected ECU. This reference is originally returned from **MC ECU Open.vi** or **MC ECU Select.vi**, and then wired through subsequent VIs.

Address is a cluster which contains the following values.

U32 **Address** specifies the address part of the source address.

U8 **Extension** contains the extension part of the source address.

U32 **Block size** determines the size of the block that has to be cleared.

Error in is a cluster which describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

TF **status** is `TRUE` if an error occurred. This VI is not executed when **status** is `TRUE`.

I32 **code** is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

abc **source** identifies the VI where the error occurred.

Output

ECU ref out is the same as **ECU ref in**. Wire the task reference to subsequent VIs for this task.

Error out describes error conditions. If the **Error in** cluster indicated an error, the **Error out** cluster contains the same information. Otherwise, **Error out** describes the error status of this VI.

status is `TRUE` if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

MC CCP Clear Memory.vi can be used to erase the FLASH EPROM prior to reprogramming. The CCP Memory Transfer address (MTA0) pointer will be set to the memory location to be erased specified by the parameters **Address** and **Extension**.

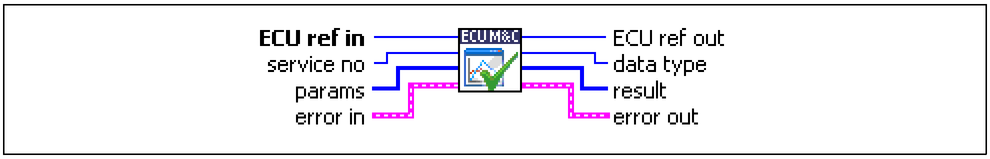
MC CCP Clear Memory.vi implements the CCP CLEAR_MEMORY command defined by the CCP specification.

MC CCP Diag Service.vi

Purpose

Calls a diagnostic service on the ECU.

Format



Input

ECU ref in is the task reference which links to the selected ECU. This reference is originally returned from **MC ECU Open.vi** or **MC ECU Select.vi**, and then wired through subsequent VIs.

Service no determines the diagnostic service that will be executed inside the ECU. For more information about the services that are implemented in the ECU, refer to the documentation for the ECU.

Params passes an array to the ECU that might be needed by the ECU to run the service. Since this VI has no knowledge about how the data will be interpreted by the ECU, you are responsible for providing the data in the correct byte ordering.

Error in is a cluster which describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is `TRUE` if an error occurred. This VI is not executed when **status** is `TRUE`.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

ECU ref out is the same as **ECU ref in**. Wire the task reference to subsequent VIs for this task.

Data Type returns a Data Type Qualifier which provides information about the data type of the result of the diagnostic service.

Result contains the information returned from the diagnostic service, uploaded from the ECU by the CCP master.

Error out describes error conditions. If the **Error in** cluster indicated an error, the **Error out** cluster contains the same information. Otherwise, **Error out** describes the error status of this VI.

status is `TRUE` if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

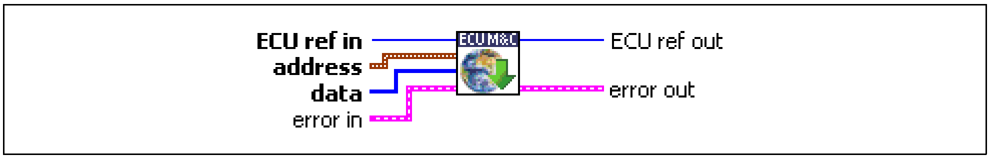
MC CCP Diag Service.vi implements the CCP command DIAG_SERVICE, which starts a diagnostic service on the ECU and waits until it is finished. The selected **Service no** specifies the diagnostic service that will be executed inside the ECU. For more information about the available services that are implemented in the ECU, refer to the documentation for the ECU.

MC CCP Download.vi

Purpose

Downloads data to an ECU.

Format



Input

U32 **ECU ref in** is the task reference which links to the selected ECU. This reference is originally returned from **MC ECU Open.vi** or **MC ECU Select.vi**, and then wired through subsequent VIs.

Address is a cluster which contains the following values.

U32 **Address** specifies the address part of the destination address.

U8 **Extension** contains the extension part of the destination address.

[U8] **Data** contains the information to be downloaded. If **data** contains more than 5 bytes of information, the download will be performed in segments up to 5 bytes in size.

Error in is a cluster which describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

TF **status** is `TRUE` if an error occurred. This VI is not executed when **status** is `TRUE`.

I32 **code** is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

abc **source** identifies the VI where the error occurred.

Output

ECU ref out is the same as **ECU ref in**. Wire the task reference to subsequent VIs for this task.

Error out describes error conditions. If the **Error in** cluster indicated an error, the **Error out** cluster contains the same information. Otherwise, **Error out** describes the error status of this VI.

status is `TRUE` if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

MC CCP Download.vi is used to download data to an ECU. The data will be stored starting at the location specified by the **Address** and **Extension** parameters.

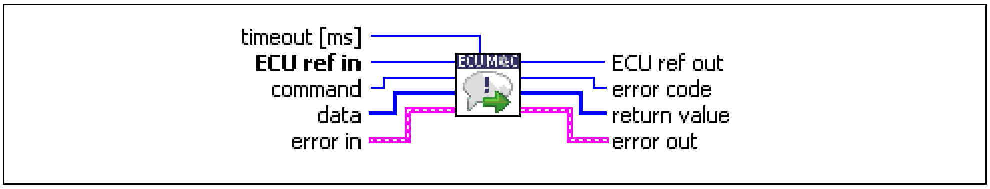
MC CCP Download.vi implements the CCP command DNLOAD defined by the CCP specification.

MC CCP Generic.vi

Purpose

Sends a generic CCP command.

Format



Input

U32 **Timeout** is the time limit, in milliseconds, during which a specified command must complete.

U32 **ECU ref in** is the task reference which links to the selected ECU. This reference is originally returned from **MC ECU Open.vi** or **MC ECU Select.vi**, and then wired through subsequent VIs.

U8 **Command** is the CCP command to be sent to the ECU.

[U8] **Data** contains a 1-dimensional array of byte information to send to the ECU.

Error in is a cluster which describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

- TF **status** is `TRUE` if an error occurred. This VI is not executed when **status** is `TRUE`.
- I32 **code** is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.
- abc **source** identifies the VI where the error occurred.

Output

ECU ref out is the same as **ECU ref in**. Wire the task reference to subsequent VIs for this task.

Error code describes the error returned from the ECU during the communication.

Return value may contain a data array bytes returned from the ECU as a response to the sent CCP command.

Error out describes error conditions. If the **Error in** cluster indicated an error, the **Error out** cluster contains the same information. Otherwise, **Error out** describes the error status of this VI.

status is `TRUE` if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

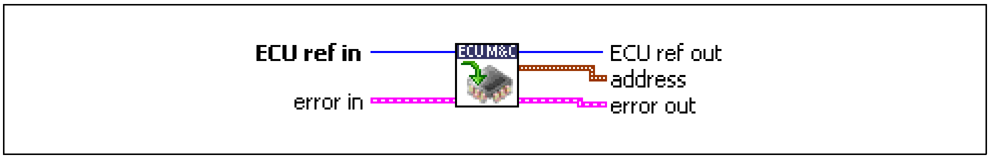
MC CCP Generic.vi implements any generic CCP command that can be used to execute user-defined commands that are not defined in the CCP standard.

MC CCP Get Active Cal Page.vi

Purpose

Retrieves the ECU Memory Transfer Address pointer to the calibration data page.

Format



Input

ECU ref in is the task reference which links to the selected ECU. This reference is originally returned from **MC ECU Open.vi** or **MC ECU Select.vi**, and then wired through subsequent VIs.

Error in is a cluster which describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is `TRUE` if an error occurred. This VI is not executed when **status** is `TRUE`.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

ECU ref out is the same as **ECU ref in**. Wire the task reference to subsequent VIs for this task.

Address is a cluster which contains the following values.

Address specifies the address part of the active calibration page address.

Extension contains the extension part of the active calibration page address.

Error out describes error conditions. If the **Error in** cluster indicated an error, the **Error out** cluster contains the same information. Otherwise, **Error out** describes the error status of this VI.

status is `TRUE` if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

MC CCP Get Active Cal Page.vi retrieves the ECU Memory Transfer Address pointer of the active calibration data page.

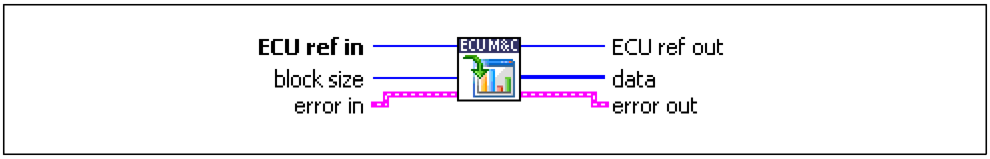
MC CCP Get Active Cal Page.vi implements the CCP command GET_ACTIVE_CAL_PAGE defined by the CCP specification.

MC CCP Get Result.vi

Purpose

Uploads requested data.

Format



Input

ECU ref in is the task reference which links to the selected ECU. This reference is originally returned from **MC ECU Open.vi** or **MC ECU Select.vi**, and then wired through subsequent VIs.

Block size is the size of the data block, in bytes, to be uploaded.

Error in is a cluster which describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is `TRUE` if an error occurred. This VI is not executed when **status** is `TRUE`.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

ECU ref out is the same as **ECU ref in**. Wire the task reference to subsequent VIs for this task.

Data is a byte array which receives the uploaded data information from the ECU.

Error out describes error conditions. If the **Error in** cluster indicated an error, the **Error out** cluster contains the same information. Otherwise, **Error out** describes the error status of this VI.

status is `TRUE` if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

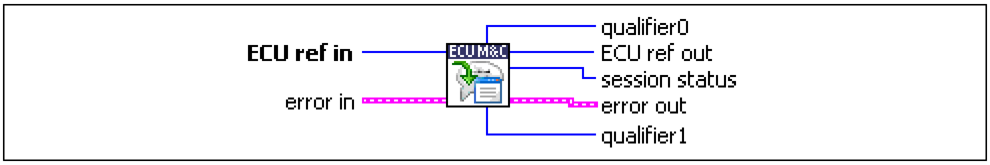
MC CCP Get Result.vi uploads data bytes from the ECU. It is assumed that the Memory Transfer Address 0(MTA0) has been set by a previous VI like **MC CCP Action Service.vi** or **MC CCP Diag Service.vi**.

MC CCP Get Session Status.vi

Purpose

Retrieves the current calibration status of the ECU.

Format



Input

U32 **ECU ref in** is the task reference which links to the selected ECU. This reference is originally returned from **MC ECU Open.vi** or **MC ECU Select.vi**, and then wired through subsequent VIs.

Error in is a cluster which describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

TF **status** is `TRUE` if an error occurred. This VI is not executed when **status** is `TRUE`.

I32 **code** is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

abc **source** identifies the VI where the error occurred.

Output

U8 **Qualifier0** describes an additional status qualifier.

U32 **ECU ref out** is the same as **ECU ref in**. Wire the task reference to subsequent VIs for this task.

U8 **Session status** is the actual session status which is returned from the ECU.

Error out describes error conditions. If the **Error in** cluster indicated an error, the **Error out** cluster contains the same information. Otherwise, **Error out** describes the error status of this VI.

status is `TRUE` if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Qualifier1 describes an additional status qualifier.

Description

MC CCP Get Session Status.vi retrieves the session status of the ECU. The return value **Session status** is a bit mask that represents several session states inside the ECU. **Qualifier0** and **Qualifier1** contain additional status information. The content of these parameters is project-specific and not defined by CCP. For more information about these parameters refer to the documentation for the ECU.

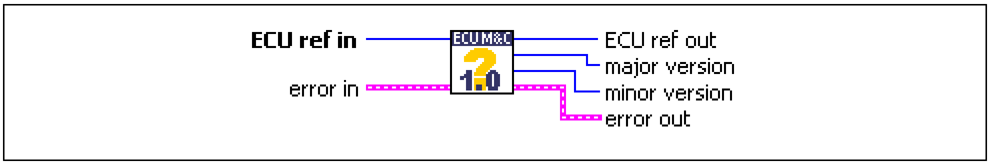
MC CCP Get Session Status.vi implements the CCP command GET_S_STATUS defined by the CCP specification.

MC CCP Get Version.vi

Purpose

Retrieves version of the CCP implemented in the ECU.

Format



Input

U32 **ECU ref in** is the task reference which links to the selected ECU. This reference is originally returned from **MC ECU Open.vi** or **MC ECU Select.vi**, and then wired through subsequent VIs.

Error in is a cluster which describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

TF **status** is `TRUE` if an error occurred. This VI is not executed when **status** is `TRUE`.

I32 **code** is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

abc **source** identifies the VI where the error occurred.

Output

U32 **ECU ref out** is the same as **ECU ref in**. Wire the task reference to subsequent VIs for this task.

U8 **Major version** returns the major version number of the CCP implementation.

U8 **Minor version** returns the minor version number of the CCP implementation.

Error out describes error conditions. If the **Error in** cluster indicated an error, the **Error out** cluster contains the same information. Otherwise, **Error out** describes the error status of this VI.

status is `TRUE` if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

MC CCP Get Version.vi can be used to query the CCP version implemented in the ECU. This command performs a mutual identification of the protocol version in the slave device to agree on a common protocol version.

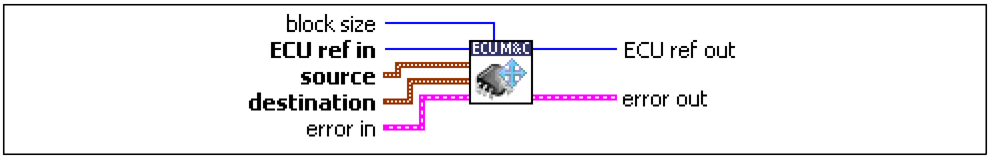
MC CCP Get Version.vi implements the CCP command GET_CCP_VERSION defined by the CCP specification.

MC CCP Move Memory.vi

Purpose

Moves a memory block on the ECU.

Format



Input

U32 **Block size** determines the size of the block on which the checksum has to be calculated.

U32 **ECU ref in** is the task reference which links to the selected ECU. This reference is originally returned from **MC ECU Open.vi** or **MC ECU Select.vi**, and then wired through subsequent VIs.

Source is a cluster which contains the following values.

- U32 **Address** specifies the address part of the source address from which the memory block will be copied.
- U8 **Extension** specifies the extension part of the source address.

Destination is a cluster which contains the following values.

- U32 **Address** specifies the address part of the destination address to which the memory block will be copied.
- U8 **Extension** specifies the extension part of the destination address.

Error in is a cluster which describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

- TF **status** is `TRUE` if an error occurred. This VI is not executed when **status** is `TRUE`.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

ECU ref out is the same as **ECU ref in**. Wire the task reference to subsequent VIs for this task.

Error out describes error conditions. If the **Error in** cluster indicated an error, the **Error out** cluster contains the same information. Otherwise, **Error out** describes the error status of this VI.

status is `TRUE` if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

MC CCP Move Memory.vi is used to move the memory contents of an ECU from one memory location to another. This function sets the Memory Transfer Address pointers MTA0 as defined in the source cluster and MTA1 as defined in the destination cluster to appropriate values.

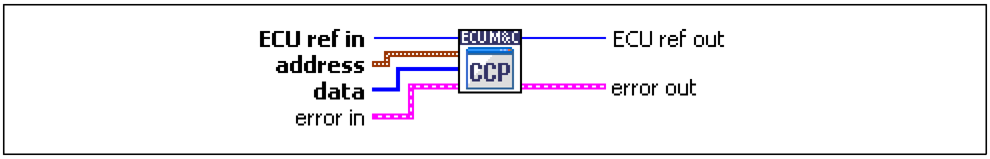
MC CCP Move Memory.vi implements the CCP command MOVE defined by the CCP specification.

MC CCP Program.vi

Purpose

Programs a memory block on the ECU.

Format



Input

ECU ref in is the task reference which links to the selected ECU. This reference is originally returned from **MC ECU Open.vi** or **MC ECU Select.vi**, and then wired through subsequent VIs.

Address is a cluster which contains the following values.

Address specifies the address part of the destination address.

Extension contains the extension part of the destination address.

Data contains the byte array to be transmitted to the ECU.

Error in is a cluster which describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is `TRUE` if an error occurred. This VI is not executed when **status** is `TRUE`.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

ECU ref out is the same as **ECU ref in**. Wire the task reference to subsequent VIs for this task.

Error out describes error conditions. If the **Error in** cluster indicated an error, the **Error out** cluster contains the same information. Otherwise, **Error out** describes the error status of this VI.

status is `TRUE` if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

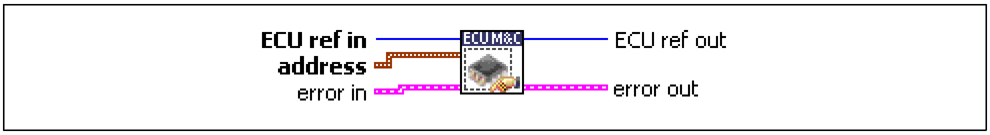
MC CCP Program.vi implements the CCP command PROGRAM. The command is used to program the specified data into nonvolatile ECU memory (Flash, EEPROM, etc). Programming starts at the selected MTA0 address and extension defined in the **Address** cluster.

MC CCP Select Cal Page.vi

Purpose

Sets the beginning of the calibration data page.

Format



Input

U32 **ECU ref in** is the task reference which links to the selected ECU. This reference is originally returned from **MC ECU Open.vi** or **MC ECU Select.vi**, and then wired through subsequent VIs.

Address is a cluster which contains the following values.

U32 **Address** specifies the address part of the address.

U8 **Extension** contains the extension part of the address.

Error in is a cluster which describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

TF **status** is `TRUE` if an error occurred. This VI is not executed when **status** is `TRUE`.

I32 **code** is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

abc **source** identifies the VI where the error occurred.

Output

ECU ref out is the same as **ECU ref in**. Wire the task reference to subsequent VIs for this task.

Error out describes error conditions. If the **Error in** cluster indicated an error, the **Error out** cluster contains the same information. Otherwise, **Error out** describes the error status of this VI.

status is `TRUE` if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

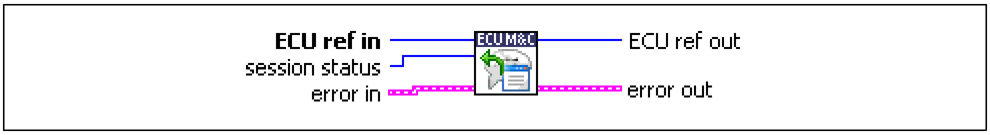
MC CCP Select Cal Page.vi implements the CCP command SELECT_CAL_PAGE. The operation of the command depends on the ECU implementation.

MC CCP Set Session Status.vi

Purpose

Updates the ECU with the current state of the calibration session.

Format



Input

ECU ref in is the task reference which links to the selected ECU. This reference is originally returned from **MC ECU Open.vi** or **MC ECU Select.vi**, and then wired through subsequent VIs.

Session status is the new status to be set in the ECU.

Error in is a cluster which describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is `TRUE` if an error occurred. This VI is not executed when **status** is `TRUE`.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

ECU ref out is the same as **ECU ref in**. Wire the task reference to subsequent VIs for this task.

Error out describes error conditions. If the **Error in** cluster indicated an error, the **Error out** cluster contains the same information. Otherwise, **Error out** describes the error status of this VI.

status is `TRUE` if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

This VI implements the CCP SET_S_STATUS command and is used to keep the ECU informed about the current state of the calibration session. The session status bits of an ECU can be read and written. Possible conditions are: reset on power-up, session log-off, and in applicable error conditions. The calibration session status is organized as a bit mask with the following assignment.

Table 5-2. Bit Mask Assignment for Calibration Session Status

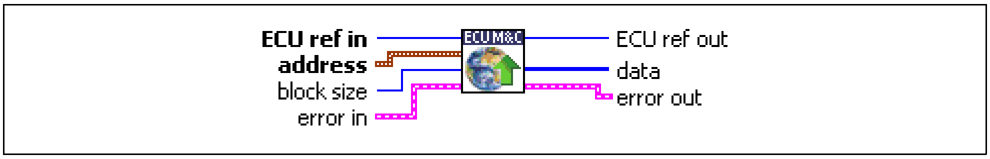
Bit	Name	Description
0	CAL	Calibration data initialized.
1	DAQ	DAQ list(s) initialized.
2	RESUME	Request to save DAQ set-up during shutdown in CCP slave. CCP slave automatically restarts DAQ after start-up.
3	Reserved	
4	Reserved	
5	Reserved	
6	STORE	Request to save calibration data during shut-down in CCP slave.
7	RUN	Session in progress.

MC CCP Upload.vi

Purpose

Uploads data from an ECU.

Format



Input

U32 **ECU ref in** is the task reference which links to the selected ECU. This reference is originally returned from **MC ECU Open.vi** or **MC ECU Select.vi**, and then wired through subsequent VIs.

Address is a cluster which contains the following values.

U32 **Address** specifies the address part of the source address in the ECU from which the memory block will be copied.

U8 **Extension** specifies the extension part of the source address.

U32 **Block size** is the size of the data block, in bytes, to be uploaded.

Error in is a cluster which describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

TF **status** is `TRUE` if an error occurred. This VI is not executed when **status** is `TRUE`.

I32 **code** is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

abc **source** identifies the VI where the error occurred.

Output

ECU ref out is the same as **ECU ref in**. Wire the task reference to subsequent VIs for this task.

Data is a byte array which receives the uploaded data from the ECU.

Error out describes error conditions. If the **Error in** cluster indicated an error, the **Error out** cluster contains the same information. Otherwise, **Error out** describes the error status of this VI.

status is `TRUE` if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

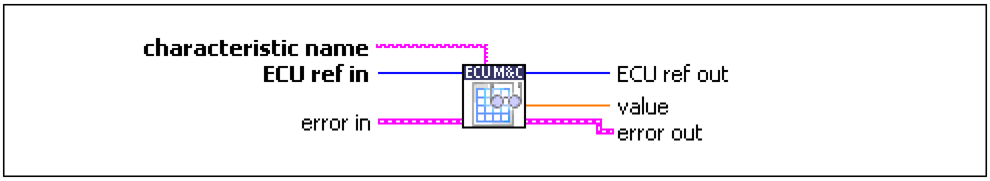
MC CCP Upload.vi implements the UPLOAD command. A data block of the specified length, starting at the specified address, will be uploaded from the ECU. **MC CCP Upload.vi** will set the Memory Transfer Address pointer MTA0 to the appropriate value as defined in the **Address** cluster.

MC Characteristic Read.vi

Purpose

Reads data from a named Characteristic on the ECU which is identified by the ECU Reference handle. The Poly VI returns a specific double, 1D, or 2D double array.

Format



Input

Characteristic name is the name of the Characteristic defined in the A2L database.

ECU ref in is the task reference which links to the selected ECU. This reference is originally returned from **MC ECU Open.vi** or **MC ECU Select.vi**, and then wired through subsequent VIs.

Error in is a cluster which describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is `TRUE` if an error occurred. This VI is not executed when **status** is `TRUE`.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

ECU ref out is the same as **ECU ref in**. Wire the task reference to subsequent VIs for this task.

Value is a poly output value which represents the data read from the ECU. The type of the poly output is determined by the poly VI selection. For information on the different poly VI types provided by **MC Characteristic Read.vi**, refer to the *Poly VI Types* section.

To select the data type, right-click the VI, go to **Select Type**, and select the type by name.

Error out describes error conditions. If the **Error in** cluster indicated an error, the **Error out** cluster contains the same information. Otherwise, **Error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

Poly VI Types

Table 5-3. Poly VI Types for the Value Parameter

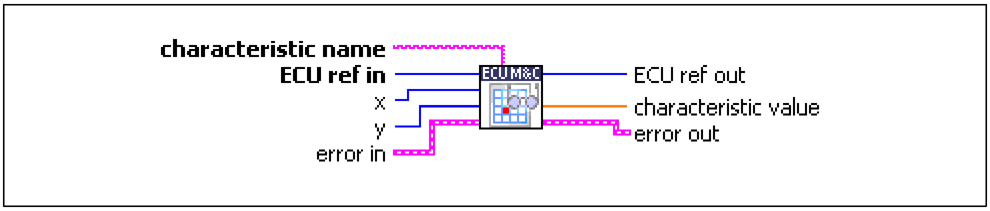
VI Type	Description
Parameter (DBL)	Returns a single double value for the selected Characteristic name.
Curve (1D)	Returns a 1-dimensional array of double values for the selected Characteristic name.
Field (2D)	Returns a 2-dimensional array of double values for the selected Characteristic name.

MC Characteristic Read Single Value.vi

Purpose

Reads a value from a named Characteristic on the ECU which is identified by the ECU Reference handle.

Format



Input

Characteristic name is the name of the Characteristic defined in the A2L database.

ECU ref in is the task reference which links to the selected ECU. This reference is originally returned from **MC ECU Open.vi** or **MC ECU Select.vi**, and then wired through subsequent VIs.

x is the horizontal index if the Characteristic consists of 1 or 2 dimensions.

y is the vertical index if the Characteristic consists of 2 dimensions.

Error in is a cluster which describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is `TRUE` if an error occurred. This VI is not executed when **status** is `TRUE`.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

ECU ref out is the same as **ECU ref in**. Wire the task reference to subsequent VIs for this task.

Characteristic value returns a single sample for the specified Characteristic.

Error out describes error conditions. If the **Error in** cluster indicated an error, the **Error out** cluster contains the same information. Otherwise, **Error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

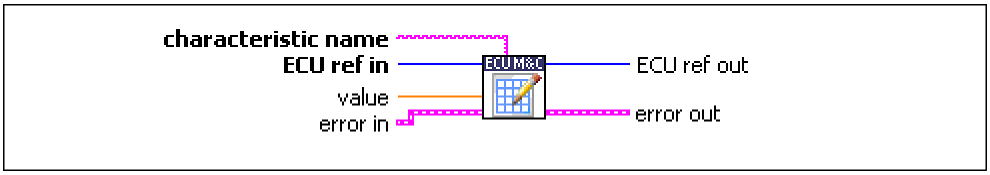
MC Characteristic Read Single Value.vi reads a value from a specified Characteristic on the ECU which is identified by the ECU Reference handle. The value to be read is identified by the **x** and **y** indices. If the Characteristic array has 0 or 1 dimensions, **y** and/or **x** can be left unwired.

MC Characteristic Write.vi

Purpose

Writes the value(s) of a named Characteristic to an ECU identified by the ECU ref handle. The Poly VI writes the selected type double, 1D or 2D array.

Format



Input

Characteristic name is the name of a Characteristic stored in the A2L database file to which one or more values may be written.

ECU ref in is the task reference which links to the selected ECU. This reference is originally returned from **MC ECU Open.vi** or **MC ECU Select.vi**, and then wired through subsequent VIs.

Value writes the data for the Characteristic channel initialized by **Characteristic name**. **Value** values are listed in the *Poly VI Types* section.

Error in is a cluster which describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is `TRUE` if an error occurred. This VI is not executed when **status** is `TRUE`.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

ECU ref out is the same as **ECU ref in**. Wire the task reference to subsequent VIs for this task.

Error out describes error conditions. If the **Error in** cluster indicated an error, the **Error out** cluster contains the same information. Otherwise, **Error out** describes the error status of this VI.

status is `TRUE` if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

Poly VI Types

Table 5-4. Poly VI Types for the Characteristic Parameter

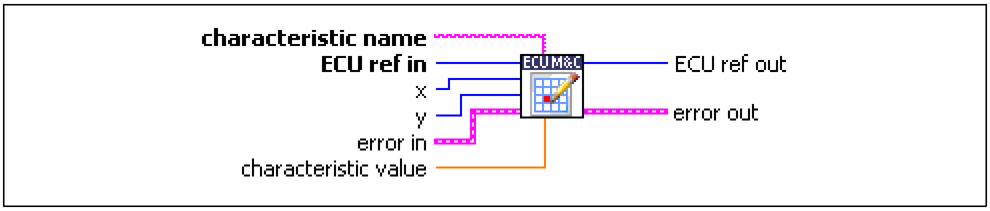
VI Type	Description
Parameter (DBL)	Writes a single double value to the selected Characteristic name.
Curve (1D)	Writes a 1-dimensional array of double values to the selected Characteristic name.
Field (2D)	Writes a 2-dimensional array of double values to the selected Characteristic name.

MC Characteristic Write Single Value.vi

Purpose

Writes a value to a named Characteristic on the ECU.

Format



Input

Characteristic name is the name of a Characteristic stored in the A2L database file to which one or more values may be written.

ECU ref in is the task reference which links to the selected ECU. This reference is originally returned from **MC ECU Open.vi** or **MC ECU Select.vi**, and then wired through subsequent VIs.

x is an optional input that refers to the array offset if the Characteristic is defined in the A2L database file as 1- or 2-dimensional. If the Characteristic is defined as having 0 dimensions, the input can be left unwired.

y is an optional input that refers to the array offset if the Characteristic is defined in the A2L database file as 2-dimensional. If the Characteristic is defined as having 0 or 1 dimensions, the input can be left unwired.

Error in is a cluster which describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is `TRUE` if an error occurred. This VI is not executed when **status** is `TRUE`.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Characteristic value is the value to be set for the Characteristic.

Output

ECU ref out is the same as **ECU ref in**. Wire the task reference to subsequent VIs for this task.

Error out describes error conditions. If the **Error in** cluster indicated an error, the **Error out** cluster contains the same information. Otherwise, **Error out** describes the error status of this VI.

status is `TRUE` if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

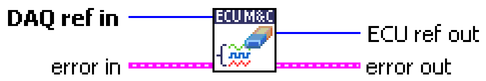
MC Characteristic Write Single Value.vi writes a value to a defined Characteristic on the ECU which is identified by the ECU Reference handle. The location to which the value is written is identified by the **x** and **y** indices. If the Characteristic array has 0 or 1 dimensions, **y** and/or **x** can be left unwired.

MC DAQ Clear.vi

Purpose

Stops communication for the Measurement task and then clears the configuration.

Format



Input

DAQ ref in is the task reference which links to the Measurement task. This reference is originally returned from **MC DAQ Initialize.vi**, and then wired through subsequent VIs.

Error in is a cluster which describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is `TRUE` if an error occurred. Unlike other VIs, this VI will execute when **status** is `TRUE`.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

ECU ref out is the ECU reference upon which **MC DAQ Initialize.vi** was called. Wire this to subsequent ECU operations.

Error out describes error conditions. If the **Error in** cluster indicated an error, the **Error out** cluster contains the same information. Otherwise, **Error out** describes the error status of this VI.

status is `TRUE` if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

MC DAQ Clear.vi must always be the final ECU M&C VI called for a Measurement task. If you do not use the **MC DAQ Clear.vi**, the remaining task configurations can cause problems in execution of subsequent ECU M&C applications.

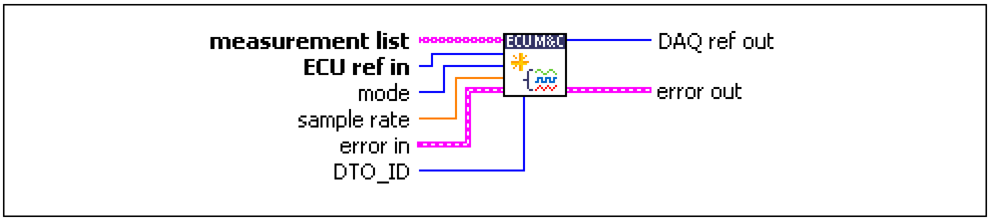
Because this VI clears the Measurement task, the Measurement task reference is not wired as an output but will be transferred into an ECU reference task handle. To change properties of a running Measurement task, use **MC DAQ Stop.vi** to stop the task, **MC Set Property.vi** to change the desired DAQ property, and then **MC DAQ Start.vi** to restart the Measurement task again.

MC DAQ Initialize.vi

Purpose

Initializes a Measurement task for the specified Measurement channel list.

Format



Input

Measurement list is the array of channel names to initialize as a Measurement task. Each channel name is provided in an array entry.

ECU ref in is the task reference which links to the selected ECU. This reference is originally returned from **MC ECU Open.vi** or **MC ECU Select.vi**, and then wired through subsequent VIs.

Mode specifies the I/O mode for the task. For an overview of the I/O modes, including figures, refer to the *Basic Programming Model* section of Chapter 4, *Using the ECU M&C API*.

DAQ List

The data is transmitted from the ECU in equidistant time intervals as defined in the A2L database. The data can be read back with the **MC DAQ Read.vi** as Single point data using a sample rate = 0 or as waveform using a sample rate > 0. Input channel data are received from the DAQ messages. Use **MC DAQ Read.vi** to obtain input samples as single-point, array, or waveform.

Polling

In this mode the data from the Measurement task are acquired from the ECU whenever the **MC DAQ Read.vi** is called.

Sample rate specifies the timing to use for samples of the task. The sample rate is specified in Hertz (samples per second). A sample rate of zero means to sample immediately. If the **Mode** is defined as DAQ List, a sample rate of zero means that **MC DAQ Read.vi** returns a single point from the most recent message received, and greater than zero means that **MC DAQ Read.vi** returns samples timed at the specified rate. If the **Mode** is defined as Polling, the sample rate is ignored.

Error in is a cluster which describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is `TRUE` if an error occurred. This VI will not execute when **status** is `TRUE`.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

DTO_ID is the CAN identifier for the **D**ata **T**ransmission **O**bject (**DTO**) used by the ECU to transmit the DAQ list data to the host. If the **DTO_ID** terminal is unwired the ECU will use the same identifier for sending the DAQ list data as for the normal CCP communication.

Output

DAQ ref out is a task reference for the Measurement task created. Wire this task reference to subsequent VIs for this Measurement task.

Error out describes error conditions. If the **Error in** cluster indicated an error, the **Error out** cluster contains the same information. Otherwise, **Error out** describes the error status of this VI.

status is `TRUE` if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

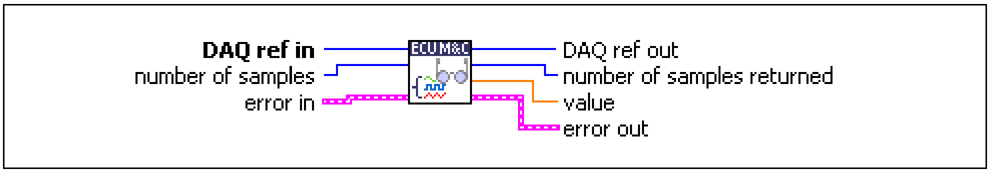
MC DAQ Initialize.vi does not start the transmission of the DAQ lists on the ECU through CCP. This enables you to use **MC Set Property.vi** to change the properties of a Measurement task. After you change properties use **MC DAQ Start.vi** to start the communication for the Measurement task.

MC DAQ Read.vi

Purpose

Reads samples from a Measurement task. Samples are obtained from received CAN messages.

Format



Input

U32 **DAQ ref in** is the task reference from the previous Measurement task VI. The task reference is originally returned from **MC DAQ Initialize.vi**, and then wired through subsequent Measurement task VIs.

U32 **Number of samples** specifies the number of samples to read for the Measurement task. For single-sample Poly VI types, **MC DAQ Read.vi** always returns one sample, so this input is ignored.

Error in is a cluster which describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

TF **status** is `TRUE` if an error occurred. This VI will not execute when **status** is `TRUE`.

I32 **code** is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

abc **source** identifies the VI where the error occurred.

Output

U32 **DAQ ref out** is the same as **DAQ ref in**. Wire the task reference to subsequent VIs for this task.

Number of samples returned indicates the number of samples returned in the samples output.

Value is a poly output that returns the samples read from the received CAN messages of the DAQ list. The type of the poly output is determined by the poly VI selection. For information on the different poly VI types provided by **MC DAQ Read.vi**, refer to the *Poly VI Types* section.

To select the data type, right-click the VI, go to **Select Type**, and select the type by name.

Error out describes error conditions. If the **Error in** cluster indicated an error, the **Error out** cluster contains the same information. Otherwise, **Error out** describes the error status of this VI.

status is `TRUE` if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

Poly VI Types

The name of each Poly VI type uses the following conventions:

- The first term is either *1Chan* or *NChan*. This indicates whether the type returns data for a single channel or multiple channels. *NChan* types return an array of analogous *1Chan* types, one entry for each channel initialized in channel list of **MC DAQ Initialize.vi**. *1Chan* types are convenient because no array indexing is required, but you are limited to reading only one channel.
- The second term is either *1Samp* or *NSamp*. This indicates whether the type returns a single sample, or an array of multiple samples. *1Samp* types are often used for single point control applications, such as within LabVIEW RT.
- The third term indicates the data type used for each sample. The type *Dbl* indicates double-precision (64-bit) floating point. The type *Wfm* indicates the waveform data type. The types *1D* and *2D* indicate one and two-dimensional arrays, respectively.

1Chan 1Samp Dbl

Returns a single sample for the first channel initialized in channel list. If the initialized sample rate is greater than zero, this poly VI type waits for the next sample time, and then returns a single sample. This enables you to execute a control loop at a specific rate. If the initialized sample rate is zero, this poly VI immediately returns a single sample. The samples output returns a single sample from the most recent message received. If no message has been received since you started the task, the value of 0 is returned in samples. You can use error out to determine whether a new message has been received since the previous call to **MC DAQ Read.vi** (or **MC DAQ Start.vi**). If no message has been received, the warning code 3FF60009 hex is returned in **error out**. If a new message has been received, the success code 0 is returned in **error out**. Unless an error occurs, **number of samples returned** is one.

NChan 1Samp 1D Dbl

Returns an array, one entry for each channel initialized in channel list. Each entry consists of a single sample. The order of channel entries in samples is the same as the order in the original channel list. If the initialized sample rate is greater than zero, this poly VI type waits for the next sample time, then returns a single sample for each channel. This enables you to execute a control loop at a specific rate. If the initialized sample rate is zero, this poly VI immediately returns a single sample for each channel. The samples output returns a single sample for each channel from the most recent message received. If no message has been received for a channel since you started the task a 0 is returned in samples. You can specify channels in channel list that span multiple messages. A sample from the most recent message is returned for all channels. You can use error out to determine whether a new message has been received since the previous call to **MC DAQ Read.vi** (or **MC DAQ Start.vi**). If no message has been received for one or more channels, the warning code 3FF60009 hex is returned in **error out**. If a new message has been received for all channels, the success code 0 is returned in **error out**. Unless an error occurs, **number of samples returned** is one. The samples array is indexed by channel, and the entry for each channel contains a single sample. If you need to determine the number of channels in the task after initialization, get the **Number of Channels** property for the task reference.

1Chan NSamp 1D Dbl

Returns an array of samples for the first channel initialized in channel list. The initialized sample rate must be greater than zero for this poly VI, because each sample in the array indicates the value of the CAN channel at a specific point in time. In other words, the sample rate specifies a virtual clock that copies the most recent value from CAN messages for each sample time. The changes in sample values from message to message enable you to view the CAN channel over time, such as for comparison with other CAN or DAQ input channels. This VI waits until all samples arrive in time before returning.

If the initialized sample rate is zero, this poly VI returns an error. If the intent is simply to read the most recent sample for a task, use the *1Chan 1Samp Dbl* type. If no message has been received since you started the task a 0 is returned in all entries of the samples array. You can use **error out** to determine whether a new message has been received since the previous call to **MC DAQ Read.vi** (or **MC DAQ Start.vi**). If no message has been received, the warning code 3FF60009 hex is returned in **error out**. If a new message has been received, the success code 0 is returned in **error out**. Unless an error occurs, the **number of samples returned** is equal to **number of samples** to read.

NChan NSamp 2D Dbl

Returns an array, one entry for each channel initialized in channel list. Each entry consists of an array of **value**. The order of channel entries in **value** is the same as the order in the original channel list. The initialized sample rate must be greater than zero for this poly VI, because each sample in the array indicates the value of each CAN channel at a specific point in time. In other words, the sample rate specifies a virtual clock that copies the most recent value from CAN messages for each sample time. The changes in sample values from message to message enable you to view the CAN channels over time, such as for comparison with other CAN or DAQ input channels. This VI waits until all samples arrive in time before returning.

If the intent is simply to read the most recent samples for a task, use the *NChan 1Samp 1D Dbl* type. If no message has been received for a channel since you started the task, the Default Value of the channel is returned in **value**. You can specify channels in channel list that span multiple messages. At each point in time, a sample from the most recent message is returned for all channels. You can use error out to determine whether a new message has been received since the previous call to **MC DAQ Read.vi** (or **MC DAQ Start.vi**). If no message has been received for one or more channels, the warning code 3FF60009 hex is returned in **error out**. If a new message has been received for all channels, the success code 0 is returned in **error out**. Unless an error occurs, the **number of samples returned** is equal to **number of samples** to read. If you need to determine the number of channels in the task after initialization, get the **Number of Channels** property for the task reference.

1Chan NSamp Wfm

Returns a single waveform for the first channel initialized in *channel list*. The initialized sample rate must be greater than zero for this poly VI, because each sample in the array indicates the value of the CAN channel at a specific point in time. In other words, the sample rate specifies a virtual clock that copies the most recent value from CAN messages for each sample time. The changes in sample values from message to message enable you to view the CAN channel over time, such as for comparison with other CAN or DAQ input channels. This VI waits until all samples arrive in time before returning. The start time of a waveform indicates the time of the first CAN sample in the array. The delta time of a waveform indicates the time between each sample in the array, as determined by the original sample rate. If the initialized sample rate is zero, this poly VI returns an error.

If the intent is to simply read the most recent sample for a task, use the *1Chan 1Samp Dbl* type. If no message has been received since you started the task a 0 is returned in all entries of the **value** waveform. You can use **error out** to determine whether a new message has been received since the previous call to **MC DAQ Read.vi** (or **MC DAQ Start.vi**). If no message has been received, the warning code 3FF60009 hex is returned in **error out**. If a new message has been received, the success code 0 is returned in **error out**. Unless an error occurs, the **number of samples returned** is equal to **number of samples** to read.

NChan NSamp 1D Wfm

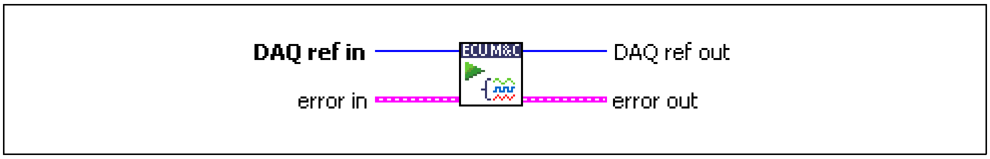
Returns an array, one entry for each channel initialized in channel list. Each entry consists of a single waveform. The order of channel entries in **value** is the same as the order in the original channel list. The initialized sample rate must be greater than zero for this poly VI, because each sample in the array of a waveform indicates the value of the CAN channel at a specific point in time. In other words, the sample rate specifies a virtual clock that copies the most recent value from CAN messages for each sample time. The changes in sample values from message to message enable you to view the M&C DAQ channel over time, such as for comparison with other CAN or DAQ input channels. This VI waits until all samples arrive in time before returning. The start time for each waveform indicates the time of the first CAN sample in the array. The delta time of a waveform indicates the time between each sample in the array, as determined by the original sample rate. If the initialized sample rate is zero, this poly VI returns an error. If the intent is simply to read the most recent samples for a task, use the *NChan 1Samp 1D Dbl* type. If no message has been received for a channel since you started the task a 0 is returned in **value**. You can specify channels in channel list that span multiple messages. At each point in time, a sample from the most recent message is returned for all channels. You can use **error out** to determine whether a new message has been received since the previous call to **MC DAQ Read.vi** (or **MC DAQ Start.vi**). If no message has been received for one or more channels, the warning code 3FF60009 hex is returned in **error out**. If a new message has been received for all channels, the success code 0 is returned in **error out**. Unless an error occurs, the **number of samples returned** is equal to **number of samples** to read. If you need to determine the number of channels in the task after initialization, get the **Number of Channels** property for the task reference.

MC DAQ Start.vi

Purpose

Starts transmission of the DAQ lists for the specified Measurement task.

Format



Input

U32 **DAQ ref in** is the task reference from the previous Measurement task VI. The task reference is originally returned from **MC DAQ Initialize.vi**, and then wired through subsequent Measurement task VIs.

Error in is a cluster which describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

TF **status** is `TRUE` if an error occurred. This VI will not execute when **status** is `TRUE`.

I32 **code** is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

abc **source** identifies the VI where the error occurred.

Output

U32 **DAQ ref out** is the same as **DAQ ref in**. Wire the task reference to subsequent VIs for this task.

Error out describes error conditions. If the **Error in** cluster indicated an error, the **Error out** cluster contains the same information. Otherwise, **Error out** describes the error status of this VI.

TF **status** is `TRUE` if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

MC DAQ Start.vi is optional to start transmission of the DAQ lists for an M&C Measurement task to use **MC DAQ Read.vi**. If you do not specify **MC DAQ Start.vi** before your first Read VI, **MC DAQ Start.vi** will be implicitly performed by the first **MC DAQ Read.vi** call.

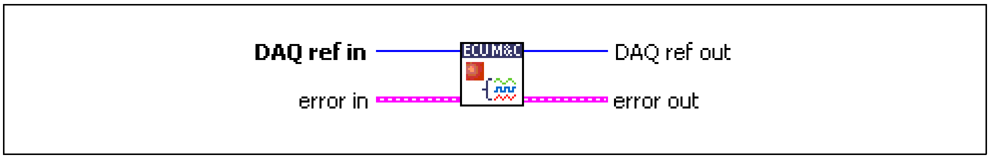
After you start the transmission of the DAQ lists, you can no longer change the configuration of the task with **MC Set Property.vi**.

MC DAQ Stop.vi

Purpose

Stops transmission of the DAQ lists for the specified Measurement task.

Format



Input

DAQ ref in is the task reference from the previous Measurement task VI. The task reference is originally returned from **MC DAQ Initialize.vi**, and then wired through subsequent Measurement task VIs.

Error in is a cluster which describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is TRUE if an error occurred. This VI will not execute when **status** is TRUE.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

DAQ ref out is the same as **DAQ ref in**. Wire the task reference to subsequent VIs for this task.

Error out describes error conditions. If the **Error in** cluster indicated an error, the **Error out** cluster contains the same information. Otherwise, **Error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

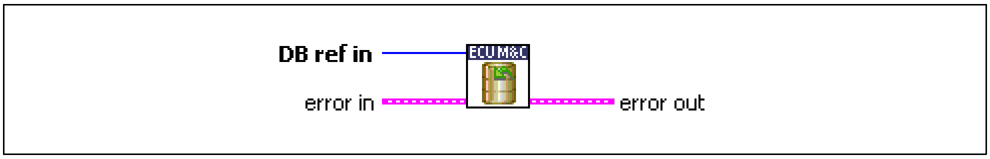
MC DAQ Stop.vi stops the transmission of the DAQ lists for the Measurement task so that you can change the configuration of the task, such as by using **MC Set Property.vi**. After you change the configuration, use **MC DAQ Start.vi** to start again. This VI does not clear the configuration for the task, so don't use it as the last M&C VI in the application. **MC DAQ Clear.vi** must always be the last M&C VI for each task.

MC Database Close.vi

Purpose

Closes a specified A2L Database.

Format



Input

DB reference in is the task reference from the initial database task VI. The task reference is originally returned from **MC Database Open.vi**.

Error in is a cluster which describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is `TRUE` if an error occurred. Unlike other VIs, this VI will execute when **status** is `TRUE`.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

Error out describes error conditions. If the **Error in** cluster indicated an error, the **Error out** cluster contains the same information. Otherwise, **Error out** describes the error status of this VI.

status is `TRUE` if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

MC Database Close.vi must always be the final M&C VI called for each communication task. If you do not use **MC Database Close.vi**, the remaining task configurations can cause problems in the execution of subsequent Measurement and Calibration applications.

MC Database Close.vi is an advanced function for database handling. In most cases it is sufficient to use **MC ECU Close.vi** instead.

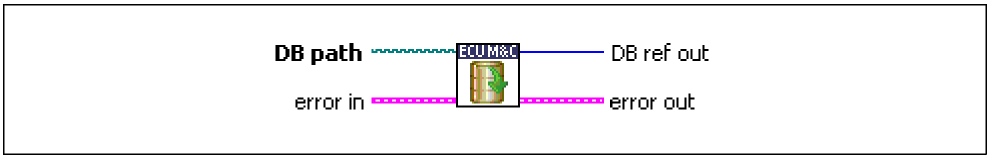
Unlike other VIs, **MC Database Close.vi** will execute when status is `TRUE` in **Error in**. Because this VI clears the task, the task reference is not wired as an output.

MC Database Open.vi

Purpose

Opens a specified A2L Database.

Format



Input

DB path is a path to a A2L database file from which to get channel names. The file must use a `.A2L` extension. You can generate A2L database files with several 3rd party tools.

Error in is a cluster which describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is `TRUE` if an error occurred. This VI will not execute when **status** is `TRUE`.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

DB reference out is the task reference which links to the opened database file.

Error out describes error conditions. If the **Error in** cluster indicated an error, the **Error out** cluster contains the same information. Otherwise, **Error out** describes the error status of this VI.

status is `TRUE` if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

MC Database Open.vi enables you to query all defined ECU names in the A2L Database using the **MC Get Names.vi** and selecting the property **ECU Names**. **MC Database Open.vi** does not start communication.

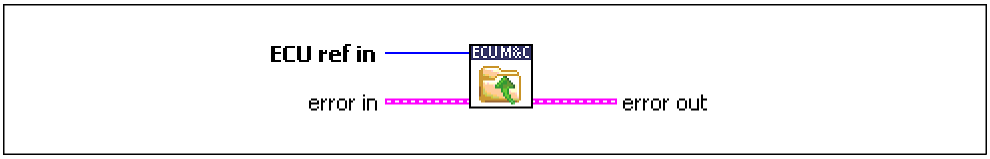
MC Database Open.vi is an advanced function for database handling. In most cases it is sufficient to use **MC ECU Open.vi** instead.

MC ECU Close.vi

Purpose

Closes the selected ECU and the associated A2L database.

Format



Input

ECU ref in is the task reference which links to the selected ECU. This reference is originally returned from **MC ECU Open.vi** or **MC ECU Select.vi**, and then wired through subsequent VIs.

Error in is a cluster which describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is `TRUE` if an error occurred. Unlike other VIs, this VI will execute when **status** is `TRUE`.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

Error out describes error conditions. If the **Error in** cluster indicated an error, the **Error out** cluster contains the same information. Otherwise, **Error out** describes the error status of this VI.

status is `TRUE` if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

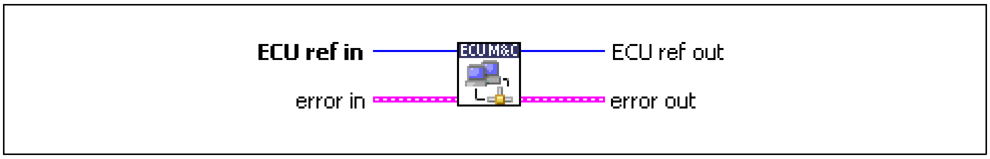
MC ECU Close.vi is the very last VI which has to be called. It deselects the ECU and closes the remaining database reference handle. **MC ECU Close.vi** must always be the final M&C VI. If you do not use **MC ECU Close.vi**, the remaining task configurations can cause problems in the execution of subsequent M&C applications. If you just want to deselect the ECU connections, call **MC ECU Deselect.vi**.

MC ECU Connect.vi

Purpose

Establishes the communication to the selected ECU through the CCP protocol. After a successful ECU Connect you can create a Measurement Task or read/write a Characteristic.

Format



Input

U32 **ECU ref in** is the task reference which links to the selected ECU. This reference is originally returned from **MC ECU Open.vi** or **MC ECU Select.vi**, and then wired through subsequent VIs.

Error in is a cluster which describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

- TF **status** is `TRUE` if an error occurred. This VI will not execute when **status** is `TRUE`.
- I32 **code** is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.
- abc **source** identifies the VI where the error occurred.

Output

U32 **ECU ref out** is the same as **ECU ref in**. Wire the task reference to subsequent VIs for this task.

Error out describes error conditions. If the **Error in** cluster indicated an error, the **Error out** cluster contains the same information. Otherwise, **Error out** describes the error status of this VI.

status is TRUE if an error occurred.

code is the error code number identifying an error. A value of 0 means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

MC ECU Connect.vi implements the CCP CONNECT command. It establishes a logical connection to an ECU, using the provided ECU Reference handle. Unless a slave device (ECU) is disconnected, it must not execute or respond to any command sent by the application. Only one CCP slave can be connected to the application at a time from a set of CCP slaves sharing identical CRO and DTO identifiers.

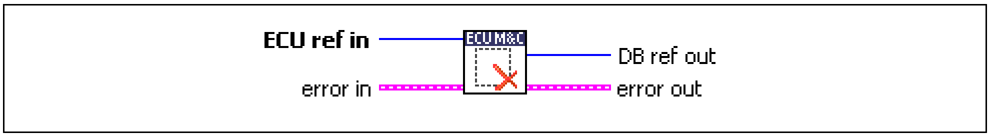
MC ECU Connect.vi is an optional function and will be automatically performed before **MC Characteristic Read.vi**, **MC Characteristic Write.vi**, **MC DAQ Initialize.vi** or any MC CCP *xxx* command is performed.

MC ECU Deselect.vi

Purpose

Deselects an ECU and invalidates the ECU reference handle.

Format



Input

ECU ref in is the task reference which links to the selected ECU. This reference is originally returned from **MC ECU Open.vi** or **MC ECU Select.vi**, and then wired through subsequent VIs.

Error in is a cluster which describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is `TRUE` if an error occurred. Unlike other VIs, this VI will execute when **status** is `TRUE`.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

DB ref out is the task reference which links to the opened database file.

Error out describes error conditions. If the **Error in** cluster indicated an error, the **Error out** cluster contains the same information. Otherwise, **Error out** describes the error status of this VI.

status is `TRUE` if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

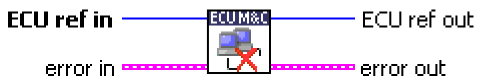
MC ECU Deselect.vi deselects the communication to the ECU. After calling this VI you can establish the communication to another ECU defined in the A2L database using **MC ECU Select.vi**.

MC ECU Disconnect.vi

Purpose

Disconnects the CCP communication to the selected ECU.

Format



Input

ECU ref in is the task reference which links to the selected ECU. This reference is originally returned from **MC ECU Open.vi** or **MC ECU Select.vi**, and then wired through subsequent VIs.

Error in is a cluster which describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is `TRUE` if an error occurred. This VI will not execute when **status** is `TRUE`.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

ECU ref out is the same as **ECU ref in**. Wire the task reference to subsequent VIs for this task.

Error out describes error conditions. If the **Error in** cluster indicated an error, the **Error out** cluster contains the same information. Otherwise, **Error out** describes the error status of this VI.

status is `TRUE` if an error occurred.

I32 **code** is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

abc **source** identifies the VI where the error occurred.

Description

MC ECU Disconnect.vi implements the CCP command DISCONNECT. **MC ECU Disconnect.vi** permanently disconnects the specified CCP slave from the communication and ends the calibration session. When the calibration session is terminated, all CCP DAQ lists of the device will be stopped and cleared and the protection masks of the device will be set to their default values.

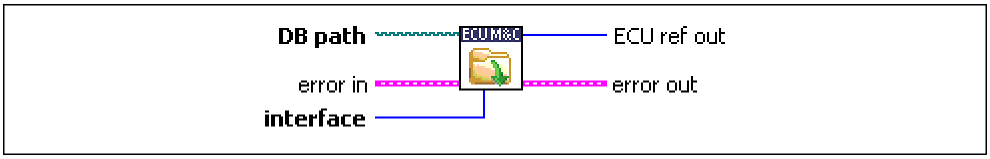
MC ECU Disconnect.vi is an optional function and will be automatically performed prior to any **MC ECU Deselect.vi** or **MC ECU Close.vi** call.

MC ECU Open.vi

Purpose

Opens a specified A2L database and selects the first ECU found in the database. If there are several ECUs stored in the A2L database use the Database Open and ECU Select VIs.

Format



Input

DB path is a path to a A2L database file from which to get channel names. The file must use a `.A2L` extension. You can generate A2L database files with several 3rd party tools.

Error in is a cluster which describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is `TRUE` if an error occurred. This VI will not execute when **status** is `TRUE`.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

CAN interface specifies the CAN interface to use for this task. The interface input uses a `ring` typedef in which value 0 selects CAN0, value 1 selects CAN1, and so on. As the ECU M&C API is based on the NI-CAN Channel API, the NI-CAN Frame API cannot used on the same CAN network interface simultaneously. If the CAN network interface is already initialized in the Frame API, this function returns an error.

Output

ECU ref out is the task reference which links to the selected ECU.

Error out describes error conditions. If the **Error in** cluster indicated an error, the **Error out** cluster contains the same information. Otherwise, **Error out** describes the error status of this VI.

status is `TRUE` if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

MC ECU Open.vi opens a specified A2L database and selects the first ECU found in the database. If there are several ECUs stored in the A2L database use **MC Database Open.vi** and **MC ECU Select.vi** to select a specific ECU.

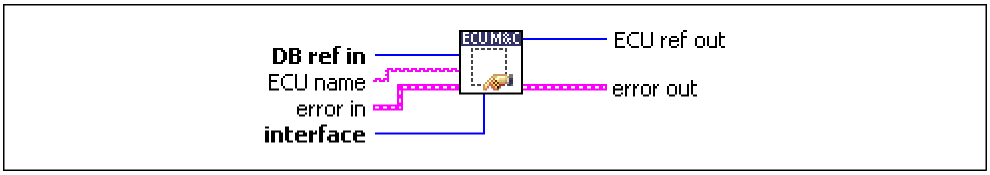
Note You can download the ASAM MCD 2MC database configuration file to a LabVIEW RT target by the File Transfer Protocol (FTP). An FTP file transfer is possible within MAX. Refer to the *LabVIEW Real-Time Graphical File Transfer Utility* section of Chapter 2, *Installation and Configuration*, for instructions on performing an FTP transfer through MAX.

MC ECU Select.vi

Purpose

Selects an ECU based upon the names stored in an A2L database.

Format



Input

U32 **DB reference in** is the task reference which links to the opened database file.

abc **ECU name** is the ECU name to select out of a A2L Database file, with which to initialize all subsequent tasks.

Error in is a cluster which describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

- TF **status** is `TRUE` if an error occurred. This VI will not execute when **status** is `TRUE`.
- I32 **code** is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.
- abc **source** identifies the VI where the error occurred.

U32 **CAN interface** specifies the CAN interface to use for this task. The interface input uses a `ring` typedef in which value 0 selects CAN0, value 1 selects CAN1, and so on. As the ECU M&C API is based on the NI-CAN Channel API, the NI-CAN Frame API cannot be used on the same CAN network interface simultaneously. If the CAN network interface is already initialized in the Frame API, this function returns an error.

Output

ECU ref out is the task reference which links to the selected ECU.

Error out describes error conditions. If the **Error in** cluster indicated an error, the **Error out** cluster contains the same information. Otherwise, **Error out** describes the error status of this VI.

status is `TRUE` if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

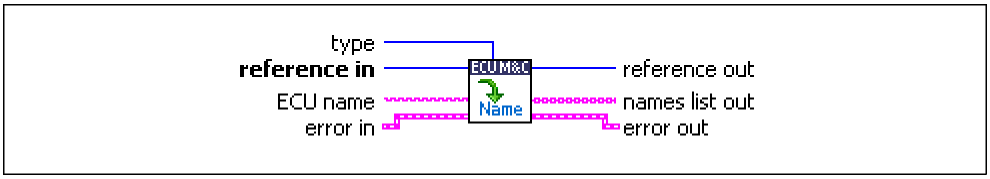
MC ECU Select.vi creates an ECU reference handle linked to the selected ECU name. **MC ECU Select.vi** does not start communication. This enables you to use **MC Set Property.vi** to change the properties of an ECU task. After you change properties, use **MC ECU Connect.vi** to start communication for the task and logically connect to the selected ECU. Prior to calling **MC ECU Select.vi**, an available ECU name can be queried by calling **MC Get Property.vi** with the parameter **ECU/Name**.

MC Get Names.vi

Purpose

Gets an array of ECU names, Measurement names, Characteristic names, or Event names from a specified A2L database file.

Format



Input

U32 **Type** (mode) is an optional input that specifies the type of names to return.

The value of Type (mode) is an enumeration:

0—**ECU Names** returns a list of channel names. You can write this list to **MC ECU Select.vi**. This is the default value.

1—**Measurement Names** returns a list of Measurement names.

2—**Characteristic Names** returns a list of Characteristic names.

3—**Event Channel Names** returns a list of Event Channel names.

U32 **Reference in** must be an ECU M&C task reference or an A2L database reference.

abc **ECU name** is the ECU name selected from an A2L database. Refer to the *Description* section for more detail.

Error in is a cluster which describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

TF **status** is `TRUE` if an error occurred. This VI will not execute when **status** is `TRUE`.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

Reference out is a copy of the reference which was passed to the **reference in** terminal.

Names list out returns the array of names, one string entry per channel. To start a Measurement task or access a Characteristic for all channels returned from **MC Get Names.vi**, wire **channel list** to **MC DAQ Initialize.vi.** To start a Measurement task or to use **MC Characteristic Write.vi** to write data to a Characteristic of an ECU. You also can wire names list to the property nodes of a front panel control such as a ring or list box. The user of the VI can then select names using this control, and the selected names can be wired to **MC DAQ Initialize.vi** or **MC Characteristic Write.vi**.

Error out describes error conditions. If the **Error in** cluster indicated an error, the **Error out** cluster contains the same information. Otherwise, **Error out** describes the error status of this VI.

status is `TRUE` if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

MC Get Names.vi is used to query the names contained within an A2L file.

The **ECU name** terminal is ignored if a valid ECU reference is connected to the **reference in** terminal. In that instance, **MC Get Names.vi** will report the names of all objects of the specified **type** inside the referenced ECU.

If a valid A2L *database reference* is passed to the **reference in** terminal, the **ECU name** terminal is used to select one of the ECUs inside the A2L database. Then, **MC Get Names.vi** will report the names of all objects of the specified **type** inside the ECU, based on the name provided. If you do not provide a name, the first ECU in the A2L file will be selected.

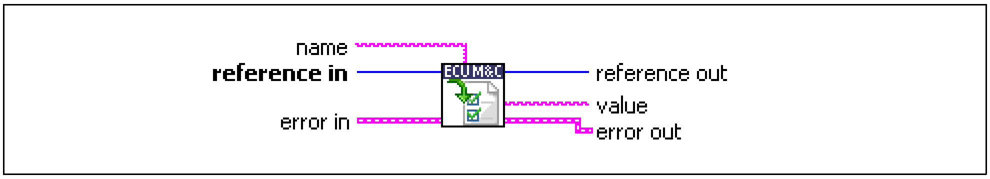
If **type** = 1, **type** = 2, or **type** = 3, the corresponding ECU name must be referenced in order to access ECU-specific properties.

MC Get Property.vi

Purpose

Gets a property for the object referenced by the **reference in** terminal. The poly VI selection determines the property to get.

Format



Input

abc **Name** specifies an individual channel within the task defined by **reference in**. The default (unwired) value of **name** is empty, which means the property applies to the entire task, not a specific channel. If a property relates to Measurement or Characteristic channels and does not apply to the entire task, but an individual channel or message within the task, you must wire the name of a Measurement or Characteristic channel from channel list into the **name** input. For other properties you must leave **name** unwired (empty).

U32 **Reference in** is the reference to any opened A2L database, a selected ECU, or an ECU which is already connected (with **MC Database Open.vi**, **MC ECU Select.vi**, **MC ECU Open.vi**, or **MC ECU Connect.vi**). The type of this reference depends on the property you want to get.

Error in is a cluster which describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

TF **status** is `TRUE` if an error occurred. This VI will not execute when **status** is `TRUE`.

I32 **code** is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

Reference out contains an ECU M&C task reference which can be wired through subsequent ECU M&C VIs.

Value is a poly output value that returns the property value. You select the property returned in value by selecting the poly VI type. The data type of value is also determined by the poly VI selection. For information about the different properties provided by **MC Get Property.vi**, refer to the *Poly VI Types* section. To select the property, right-click the VI, go to **Select Type**, and select the property by name.

Error out describes error conditions. If the **Error in** cluster indicated an error, the **Error out** cluster contains the same information. Otherwise, **Error out** describes the error status of this VI.

status is `TRUE` if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

Poly VI Types

Table 5-5. Poly Values for Value Output

Type	Hierarchy	Parameter	Description
abc		**DB File Name**	Returns the A2L Database file name with which the task has been opened. The value of this property cannot be changed using **MC Set Property.vi**.
abc	ECU	**Name**	Returns the **Name** of the selected ECU opened by **MC ECU Open.vi** or **MC ECU Select.vi**.
U32	ECU	**DTO ID**	Returns the **DTO ID** (**D**ata **T**ransmission **O**bject) which is used by the ECU to respond to CCP commands and send data and status information to the CCP master.
U32	ECU	**CRO ID**	Returns the **CRO ID** (**C**ommand **R**eceive **O**bject) which is used to send commands and data from the host to the slave device.
I32	ECU	**Interface**	Returns the interface initialized for the task, such as with **MC DAQ Initialize.vi**.
U32	ECU	**Baud Rate**	Returns the **Baud Rate** in use by the Interface. Basic baud rates such as 125000 and 500000 are specified as the numeric rate. Advanced baud rates are specified as 8000*XXYY* hex, where *YY* is the value of Bit Timing Register 0 (BTR0), and *XX* is the value of Bit Timing Register 1 (BTR1) of the CAN controller chip. For more information, refer to the **Interface Properties** dialog in MAX. The value of this property is originally set within MAX, but it can be changed using **MC Set Property.vi**.

Table 5-5. Poly Values for Value Output (Continued)

Type	Hierarchy	Parameter	Description
U32	ECU	**Station Address**	Returns the **Station Address** of the slave device. CCP is based on the idea, that several ECUs can share the same CAN Arbitration IDs for CCP communication. To avoid communication conflicts CCP defines a **Station Address** that has to be unique for all ECUs sharing the same CAN Arbitration IDs. Unless an ECU has been addressed by its Station Address, the ECU must not react to CCP commands sent by the CCP master.
U32	ECU	**Byte Order**	Returns the **Byte Order** of the CCP slave device. `0`—MSB_LAST The CCP Slave device uses the MSB_LAST (Intel) byte ordering. `1`—MSB_FIRST The CCP Slave device uses the MSB_FIRST (Motorola) byte ordering.
abc	ECU	**SeedKey Cal Name**	Returns the file name of the SeedKey DLL used for Calibration purposes.
abc	ECU	**SeedKey DAQ Name**	Returns the file name of the SeedKey DLL used for DAQ purposes.
abc	ECU	**SeedKey Prog Name**	Returns the file name of the SeedKey DLL used for programming purposes.

Table 5-5. Poly Values for Value Output (Continued)

Type	Hierarchy	Parameter	Description
abc	ECU	**Checksum DLL Name**	Returns the file name of the Checksum DLL used for verifying the checksum.
[U8]	ECU	**Master ID**	Returns CCP **Master ID** information. This ID information is optional and specific to the ECU implementation. For more information about the CCP master ID, refer to th documentation for the ECU.
[U8]	ECU	**ID**	Returns the slave device identifier. This ID information is optional and specific to the ECU implementation. For more information about the CCP slave ID, refer to the documentation for the ECU.
U32	ECU	**ID Data Byte**	Returns a data type qualifier of the slave device identifier. This ID information is optional and specific to the ECU implementation. For more information about the CCP slave ID, refer to the documentation for the ECU.
TF	ECU	**Single Byte DAQ List?**	Determines if an ECU supports single-byte or multi-byte DAQ list entries.
U32	Measurement	**Address**	Returns the address part of the address of the selected Measurement in the memory of the control unit.
U8	Measurement	**Extension**	Returns the extension part of the address. This optional parameter may contain additional address information defined in the A2L database. For instance, it can be used to distinguish different address spaces of an ECU (multi-microcontroller devices).

Table 5-5. Poly Values for Value Output (Continued)

Type	Hierarchy	Parameter	Description
U8	Measurement	**Data Type**	Returns the data type of the Measurement task.
U32	Measurement	**Byte Order**	Returns the specified byte order: 0—Intel format Bytes are in little-endian order, with most-significant bit first. 1—Motorola format Bytes are in big-endian order, with least-significant bit first.
DBL	Measurement	**Maximum**	Returns the maximum value of the Measurement.
DBL	Measurement	**Minimum**	Returns the minimum value of the Measurement.
TF	Measurement	**Read Only?**	Returns `TRUE` if the selected Measurement is read only and can only be accessed through **MC DAQ Read.vi**, or returns `FALSE` if the Measurement can be accessed through **MC Measurement Write.vi** as well.
U32	Characteristic	**Address**	Returns the address of the selected Characteristic in the memory of the ECU.
U8	Characteristic	**Extension**	Returns additional address information. For instance it can be used, to distinguish different address spaces of an ECU (multi-microcontroller devices).

Table 5-5. Poly Values for Value Output (Continued)

Type	Hierarchy	Parameter	Description
U8	Characteristic	**Data Type**	Returns the data type of the Characteristic.
U32	Characteristic	**Byte Order**	Returns the specified byte order the byte order: 0—Intel format Bytes are in little-endian order, with most-significant bit first. 1—Motorola format Bytes are in big-endian order, with least-significant bit first.
U32	Characteristic	**Dimension**	Returns the dimension of a Characteristic. 0—0 dimension The Characteristic can be accessed (read/write) through a double value. 1—1 dimension The Characteristic can be accessed (read/write) through a one-dimensional array of double value. 2—2 dimensions The Characteristic can be accessed (read/write) through a two-dimensional array of double value.
U32	Characteristic	**Sizes**	Returns the array **Sizes** for *X* and *Y* direction of the Characteristic.

Table 5-5. Poly Values for Value Output (Continued)

Type	Hierarchy	Parameter	Description
DBL	Characteristic	**X Axis**	Returns X-Axis values on which the Characteristic is defined. It is valid if the dimension of the selected Characteristic is 1 or 2.
DBL	Characteristic	**Y Axis**	Returns Y-Axis values on which the Characteristic is defined. It is valid if the dimension of the selected Characteristic is 2.
DBL	Characteristic	**Maximum**	Returns the maximum value of the Characteristic.
DBL	Characteristic	**Minimum**	Returns the minimum value of the Characteristic.
TF	Characteristic	**Read Only?**	Returns if a Characteristic is set to Read Only. In this case it is not allowed to call **MC Characteristic Write.vi** for this Characteristic.

Table 5-5. Poly Values for Value Output (Continued)

Type	Hierarchy	Parameter	Description
U32	DAQ	**Mode**	Returns the selected I/O mode for the M&C Measurement task. 0—DAQ List The data is transmitted by the ECU based on an event channel, which can be equidistant in time or sporadic. The data can be read back with the **MC DAQ Read.vi** as Single point data using sample rate = 0, or as a waveform using a sample rate > 0. Input channel data is received from the DAQ messages. Use **MC DAQ Read.vi** to obtain input samples as single-point, array, or waveform. 1—Polling In this mode the data from the Measurement task is uploaded from the ECU whenever **MC DAQ Read.vi** is called.
U32	DAQ	**DTO ID**	Returns the **DTO ID** (**D**ata **T**ransmission **O**bject) which is used by the ECU to send DAQ list data to the CCP master.
DBL	DAQ	**Sample Rate**	Returns the selected **Sample Rate** in Hz for the M&C Measurement task, which may be obtained with **MC DAQ Initialize.vi**.
U32	DAQ	**# Channels**	Returns the number of channels initialized in a DAQ channel list of a M&C Measurement task. This is the number of array entries required when using **MC DAQ Read.vi**.
abc	DAQ	**Event Channel**	Returns the selected event channel name to which the Measurement task is assigned.

Table 5-5. Poly Values for Value Output (Continued)

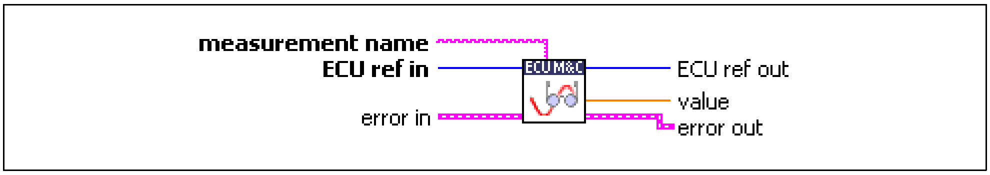
Type	Hierarchy	Parameter	Description
U16	DAQ	**Prescaler**	Returns the prescaling factor which is used to reduce the desired transmission frequency of the associated DAQ list.
U32	Version	**Major**	Returns the major version of the ECU M&C software, such as the 1 in version 1.2.5.
U32	Version	**Minor**	Returns the minor version of the ECU M&C software, such as the 2 in version 1.2.5.
U32	Version	**Update**	Returns the update version of the ECU M&C software, such as the 5 in version 1.1.5.
U32	Version	**Build**	Returns the build number of the ECU M&C software. This number applies to the Development, Alpha, and Beta phases only, and should be ignored for the Release phase.
abc	Version	**Comment**	Returns a comment string for the ECU M&C software. If you received a custom release of ECU M&C from National Instruments, this comment often describes special features of the release.

MC Measurement Read.vi

Purpose

Reads a single Measurement value from the ECU.

Format



Input

Measurement name is the name of a measurement channel stored in the A2L database file you want to read.

ECU ref in is the task reference which links to the selected ECU. This reference is originally returned from **MC ECU Open.vi** or **MC ECU Select.vi**, and then wired through subsequent VIs.

Error in is a cluster which describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is `TRUE` if an error occurred. This VI will not execute when **status** is `TRUE`.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

ECU ref out is the same as **ECU ref in**. Wire the task reference to subsequent VIs for this task.

Value returns a single sample for the Measurement channel initialized in **measurement name**.

Error out describes error conditions. If the **Error in** cluster indicated an error, the **Error out** cluster contains the same information. Otherwise, **Error out** describes the error status of this VI.

status is `TRUE` if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

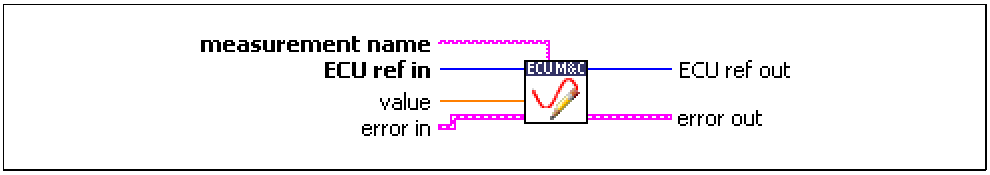
MC Measurement Read.vi performs a single point read of a single Measurement from the selected ECU without opening a Measurement task.

MC Measurement Write.vi

Purpose

Writes a single Measurement value to the ECU.

Format



Input

Measurement name is the name of a Measurement channel stored in the A2L database file to which to write a Measurement value.

ECU ref in is the task reference which links to the selected ECU. This reference is originally returned from **MC ECU Open.vi** or **MC ECU Select.vi**, and then wired through subsequent VIs.

Value writes a single sample for the Measurement channel initialized in **measurement name**.

Error in is a cluster which describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is `TRUE` if an error occurred. This VI will not execute when **status** is `TRUE`.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

ECU ref out is the same as **ECU ref in**. Wire the task reference to subsequent VIs for this task.

Error out describes error conditions. If the **Error in** cluster indicated an error, the **Error out** cluster contains the same information. Otherwise, **Error out** describes the error status of this VI.

status is `TRUE` if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

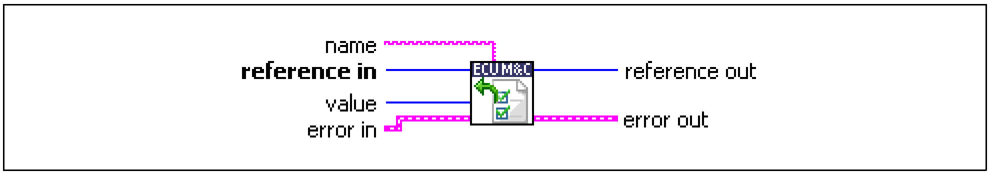
MC Measurement Write.vi performs a single point write of a Measurement into the selected ECU without opening a Measurement task. **MC Measurement Write.vi** can only be performed if the Measurement is not set to **read only**. To query if an ECU Measurement channel can be accessed by **MC Measurement Write.vi**, first call **MC Get Property.vi** with the parameter **Measurement/Read Only?**.

MC Set Property.vi

Purpose

Sets a property for the specified A2L database file, Measurement Task or Characteristic referenced by the **reference in** terminal. The poly VI selection determines the property to set.

Format



Input

Name is not used, and can be left unwired. This parameter may be used for further extensions.

Reference in specifies a valid task handle depending on the information which has to be set. If a generic property has to be set, a db ref handle is needed. If a Measurement property has to be set, a valid DAQ ref handle has to be wired into **reference in**. If an ECU property has to be set, a valid ECU ref handle has to be wired into **reference in**.

Value is a poly input that specifies the property value. You select the property to set as value by selecting the poly VI type. The data type of **value** is also determined by the poly VI selection. For information on the different properties provided by **MC Set Property.vi**, refer to the *Poly VI Types* section. To select the property, right-click the VI, go to **Select Type** and select the property by name.

Error in is a cluster which describes error conditions occurring before the VI executes. If an error has already occurred, the VI returns the value of the **error in** cluster to **error out**.

status is `TRUE` if an error occurred. This VI will not execute when **status** is `TRUE`.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Output

Reference out is a copy of the **reference in** terminal which can be wired through subsequent ECU M&C VIs.

Error out describes error conditions. If the **Error in** cluster indicated an error, the **Error out** cluster contains the same information. Otherwise, **Error out** describes the error status of this VI.

status is `TRUE` if an error occurred.

code is the error code number identifying an error. A value of `0` means success. A negative value means error: VI did not execute the intended operation. A positive value means warning: VI executed intended operation, but an informational warning is returned. For a description of the **code**, wire the error cluster to a LabVIEW error-handling VI, such as the **Simple Error Handler**.

source identifies the VI where the error occurred.

Description

There are two types of properties which can be modified in the poly input **value**, ECU-specific properties and DAQ-specific properties.

ECU-Specific Properties

ECU-specific properties relate to the setting of the ECU. If you need to change a property of the ECU you need a valid ECU reference, but the ECU should not be connected. First, call **MC ECU Open.vi**, followed by **MC Set Property.vi** and then **MC ECU Connect.vi**. If you have already connected to the ECU, you can change an ECU property by calling **MC ECU Disconnect.vi**, followed by **MC Set Property.vi**, and then **MC ECU Connect.vi** again. Refer to Table 5-6 for a list of ECU-specific properties that can be used to define the poly input **value**.

DAQ-Specific Properties

You cannot set a property while the task is running. If you need to change a property prior to starting the task, call **MC DAQ Initialize.vi**, followed by **MC Set Property.vi** and then **MC DAQ Start.vi**. After you start the task, you also can change a property by calling **MC DAQ Stop.vi**, followed by **MC Set Property.vi**, and then restart the task with **MC DAQ Start.vi**. Refer to Table 5-7 for a list of DAQ-specific properties that can be used to define the poly input **value**.

Poly VI Types

Table 5-6. ECU-Specific Property Value Types for the POLY Input Value

Type	Hierarchy	Parameter	Description
U32	ECU	**DTO ID**	Sets the **DTO ID**, which is the CAN identifier for the **D**ata **T**ransmission **O**bject (DTO). The DTO is used by the CCP slave devices to return data and status information to the application.
U32	ECU	**CRO ID**	Sets the **CRO ID** (**C**ommand **R**eceive **O**bject) which is used to send commands and data from the host to the slave device.
U32	ECU	**Baud Rate**	Sets the **Baud Rate** in use by the interface. This property applies to all tasks initialized with the Interface. You can specify the following basic baud rates as the numeric rate: 33333, 83333, 100000, 125000, 200000, 250000, 400000, 500000, 800000, and 1000000. You also can specify advanced baud rates in the form 8000*XXYY* hex, where *YY* is the value of Bit Timing Register 0 (BTR0), and *XX* is the value of Bit Timing Register 1 (BTR1). For more information, refer to the **Interface Properties** dialog in MAX. The value of this property is originally set within MAX, but it can be changed using **MC Set Property.vi**.

Table 5-6. ECU-Specific Property Value Types for the POLY Input Value (Continued)

Type	Hierarchy	Parameter	Description
U32	ECU	**Station Address**	Sets the **Station Address** of the slave device. CCP is based on the idea that several ECUs can share the same CAN Arbitration IDs for CCP communication. To avoid communication conflicts, CCP defines a **Station Address** that has to be unique for all ECUs sharing the same CAN Arbitration IDs. Unless an ECU has been addressed by its **Station Address**, the ECU must not react to CCP commands sent by the CCP master.
U32	ECU	**Byte Order**	Sets the byte order of the CCP slave device. 0—MSB_LAST The CCP slave device uses the MSB_LAST (Intel) byte ordering. 1—MSB_FIRST The CCP slave device uses the MSB_FIRST (Motorola) byte ordering.
abc	ECU	**SeedKey Cal Name**	Sets the file name of the SeedKey DLL used for Calibration purposes.
abc	ECU	**SeedKey DAQ Name**	Sets the file name of the SeedKey DLL used for DAQ purposes.
abc	ECU	**SeedKey Prog Name**	Sets the file name of the SeedKey DLL used for programming purposes.
abc	ECU	**Checksum DLL Name**	Sets the file name of the Checksum DLL used for verifying the checksum.

Table 5-6. ECU-Specific Property Value Types for the POLY Input Value (Continued)

Type	Hierarchy	Parameter	Description
U8	ECU	**Master ID**	Sets the CAN identifier of the CCP master that is used by the CCP command EXCHANGE_ID as a parameter.
TF	ECU	**Single Byte DAQ List?**	Sets the ECU to support single-byte or multi-byte DAQ list entries.

Table 5-7. DAQ-Specific Property Value Types for the POLY Input Value

Type	Hierarchy	Parameter	Description
I32	DAQ	**Mode**	Sets the selected I/O mode for the M&C Measurement task. 0—DAQ List The data is transmitted from the ECU in equidistant time intervals as defined in the A2L database. The data can be read back with the **MC DAQ Read.vi** as Single point data using sample rate = 0, or as a waveform using a sample rate > 0. Input channel data is received from the DAQ messages. Use **MC DAQ Read.vi** to obtain input samples as single-point, array, or waveform. 1—Polling In this mode the data from the Measurement task is uploaded from the ECU whenever **MC DAQ Read.vi** is called.
U32	DAQ	**DTO ID**	Sets the **DTO ID** (**D**ata **T**ransmission **O**bject) which is used by the ECU to send DAQ list data to the CCP master.

Table 5-7. DAQ-Specific Property Value Types for the POLY Input Value (Continued)

Type	Hierarchy	Parameter	Description
abc	DAQ	**Event Channel**	Sets the event channel name to which the Measurement task is assigned.
U16	DAQ	**Prescaler**	Sets the prescaling factor, which reduces the desired transmission frequency of the associated DAQ list.

6 ECU M&C API for C

This chapter lists the ECU M&C functions and describes the format, purpose, and parameters. Unless otherwise stated, each ECU M&C function suspends execution of the calling thread until it completes. The functions in this chapter are listed alphabetically.

Section Headings

The following are section headings found in the ECU M&C API for C functions.

Purpose

Each function description includes a brief statement of the purpose of the function.

Format

The format section describes the format of each function for the C programming language.

Input and Output

The input and output parameters for each function are listed.

Description

The description section gives details about the purpose and effect of each function.

List of Data Types

The following data types are used with functions of the ECU M&C API for C.

Table 6-1. Data Types for the ECU M&C API for C

Data Type	Purpose
i8	8-bit signed integer
i16	16-bit signed integer
i32	32-bit signed integer

Table 6-1. Data Types for the ECU M&C API for C (Continued)

Data Type	Purpose
u8	8-bit unsigned integer
u16	16-bit unsigned integer
u32	32-bit unsigned integer
f32	32-bit floating-point number
f64	64-bit floating-point number
str	ASCII string represented as an array of characters terminated by null character ('\0'). This type is used with output strings. **str** is typically used in the ECU M&C API as a pointer to a string, as `char*`.
cstr	ASCII string represented as an array of characters terminated by null character ('\0'). This type is used with input strings. **cstr** is typically used in the ECU M&C API as a pointer to a string, as `const char*`.
mcTypeTaskRef	Reference to an initialized database task, ECU task, or Measurement task.
mcAddress	C struct which represents the target address for a specific CCP operation in the ECU.

List of Functions

The following table contains an alphabetical list of the ECU M&C API for C functions.

Table 6-2. Functions for the ECU M&C API for C

Function	Purpose
`mcCCPActionService`	Calls an implementation-specific action service on the ECU.
`mcCCPBuildChecksum`	Calculates a checksum over a defined memory range.
`mcCCPCalculateChecksum`	Calculates the checksum of a data block.
`mcCCPClearMemory`	Clears the contents of the specified ECU memory.
`mcCCPDiagService`	Calls an implementation-specific diagnostic service on the ECU.
`mcCCPDownload`	Downloads data to an ECU.
`mcCCPGeneric`	Sends a generic CCP command.
`mcCCPGetActiveCalPage`	Retrieves the ECU Memory Transfer Address pointer to the calibration data page.

Table 6-2. Functions for the ECU M&C API for C (Continued)

Function	Purpose
`mcCCPGetResult`	Uploads data from the ECU when the Memory Transfer Address pointer 0 (MTA0) has been set.
`mcCCPGetSessionStatus`	Retrieves the current status of the Calibration Session.
`mcCCPGetVersion`	Retrieves CCP version implemented in the ECU.
`mcCCPMoveMemory`	Moves a memory block on the ECU.
`mcCCPProgram`	Programs a memory block on the ECU.
`mcCCPSelectCalPage`	Sets the specified address to be the start address of the calibration data page.
`mcCCPSetSessionStatus`	Updates the ECU with the current state of the calibration session.
`mcCCPUpload`	Uploads data from an ECU.
`mcCharacteristicRead`	Reads all data from a named Characteristic on the ECU which is identified by the ECU Reference handle.
`mcCharacteristicReadSingleValue`	Reads a single value from a named Characteristic on the ECU which is identified by the ECU Reference handle.
`mcCharacteristicWrite`	Downloads data to a Characteristic for a selected ECU.
`mcCharacteristicWriteSingleValue`	Writes a single value to a named Characteristic on the ECU.
`mcDAQClear`	Stops communication for the Measurement task and clears the task.
`mcDAQInitialize`	Initializes a Measurement task for the specified Measurement channel list.
`mcDAQRead`	Reads samples from a Measurement task. Samples are obtained from received CAN messages.
`mcDAQStart`	Starts the transmission of the DAQ lists assigned to the Measurement task on the ECU.
`mcDAQStop`	Stops transmission of the DAQ lists for the specified Measurement task.
`mcDatabaseClose`	Closes a specified A2L Database reference.
`mcDatabaseOpen`	Opens a specified A2L Database.

Table 6-2. Functions for the ECU M&C API for C (Continued)

Function	Purpose
`mcECUConnect`	Establishes communication to the selected ECU through CCP. After a successful ECU Connect you can create a Measurement Task or read/write a Characteristic.
`mcECUDeselect`	Deselects an ECU and invalidates the ECU reference handle.
`mcECUDisconnect`	Disconnects CCP communication to the selected ECU.
`mcECUSelect`	Selects an ECU from the names stored in an A2L database.
`mcGetNames`	Retrieves a comma-separated list of ECU, Measurement, Characteristic, or Event names from a specified A2L database.
`mcGetNamesLength`	Retrieves the amount of memory required to store the names returned by mcGetNames.
`mcGetProperty`	Retrieves a property of the driver, the database, the ECU, a Characteristic, a Measurement, or a Measurement task.
`mcMeasurementRead`	Reads a single Measurement value from the ECU.
`mcMeasurementWrite`	Writes a single Measurement value to the ECU.
`mcSetProperty`	Sets a property of the driver, the database, the ECU, a Characteristic, a Measurement, or a Measurement task.
`mcStatusToString`	Converts a status code into a descriptive string.

mcCCPActionService

Purpose

Calls an implementation-specific action service on the ECU.

Format

```
mcTypeStatus        mcCCPActionService(
                        mcTypeTaskRef ECURefNum,
                        u16 ServiceNo,
                        u8 Params[4],
                        u8 *ResultLength,
                        u8 *DataType);
```

Input

`ECURefNum`	`ECURefNum` is the task reference which links to the selected ECU. This reference is originally returned from `mcECUSelect`.
`ServiceNo`	`ServiceNo` determines the service that will be executed inside the ECU. For more information about the services that are implemented in the ECU refer to the documentation for the ECU.
`Params`	`Params` passes the parameters of the service function as an array of bytes to the ECU. Since the parameters and their data types are specific to the ECU implementation, you are responsible of providing the required parameters in the correct byte ordering.

Output

`*ResultLength`	`ResultLength` indicates the amount of data that can be uploaded from the ECU as a result of the execution of the service. The result of this service can be accessed by calling the function `mcCCPGetResult` right after `mcCCPActionService`.
`*DataType`	`DataType` is a data type qualifier that determines the data format of the result.

Return Value

The return value indicates the status of the function call as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `mcStatusToString` function of the ECU M&C API to obtain a descriptive string for the return value.

Description

`mcCCPActionService` implements the CCP command ACTION_SERVICE. The ECU carries out the requested service and automatically sets the Memory Transfer Address MTA0 to the location from which the CCP master may upload the requested action service return information (if applicable).

The result of this service can be accessed by calling the function `mcCCPGetResult` right after `mcCCPActionService`.

mcCCPBuildChecksum

Purpose

Calculates a checksum over a defined memory range.

Format

```
mcTypeStatus        mcBuildChecksum(
                       mcTypeTaskRef ECURefNum,
                       mcAddress Address,
                       u32 BlockSize,
                       u8 *SizeOfChecksum
                       u8 *Checksum);
```

Input

`ECURefNum`	`ECURefNum` is the task reference which links to the selected ECU. This reference is originally returned from `mcECUSelect`.
`Address`	Configures the target address for the checksum operation in the ECU. `mcAddress` is a C struct consisting of: **Address** Specifies the address part of the target address. **Extension** Extension contains the extension part of the target address.
`BlockSize`	`BlockSize` determines the size of the block on which the checksum has to be calculated.

Output

`SizeofChecksum`	`SizeofChecksum` returns the size in bytes of the calculated checksum.
`Checksum`	`Checksum` is the calculated checksum.

Return Value

The return value indicates the status of the function call as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `mcStatusToString` function of the ECU M&C API to obtain a descriptive string for the return value.

Description

`mcCCPBuildChecksum` implements the CCP BUILD_CHKSUM command defined by the CCP specification.

The function `mcCCPBuildChecksum` is used to calculate the checksum of the specified memory block inside the ECU starting at the selected `Address`. The checksum algorithm is not specified by CCP and the checksum algorithm may be different on different devices.

mcCCPCalculateChecksum

Purpose

Calculates the checksum of a data block.

Format

```
mcTypeStatus        mcCCPCalculateChecksum(
                        mcTypeTaskRef ECURefNum,
                        u32 BlockSize,
                        u8 *Data,
                        u8 *SizeOfChecksum,
                        u8 *Checksum);
```

Input

`ECURefNum`	`ECURefNum` is the task reference which links to the selected ECU. This reference is originally returned from `mcECUSelect`.
`BlockSize`	`BlockSize` determines the size of the block on which the checksum has to be calculated.

Output

`Data`	`Data` is a byte array over which the checksum calculation will be performed.
`SizeofChecksum`	`SizeofChecksum` returns the size in bytes of the calculated checksum.
`Checksum`	`Checksum` is the calculated checksum.

Return Value

The return value indicates the status of the function call as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `mcStatusToString` function of the ECU M&C API to obtain a descriptive string for the return value.

Description

`mcCCPCalculateChecksum` implements a checksum calculation over a given data block. The checksum algorithm will be performed over a dedicated checksum function provided by a specific DLL. The name of the Checksum DLL is defined in the A2L data base and can be changed by the application by the `mcSetProperty` function using the `mcPropECU_Checksum` property.

mcCCPClearMemory

Purpose

Clears the contents of the specified ECU memory.

Format

```
mcTypeStatus        mcClearMemory(
                       mcTypeTaskRef ECURefNum,
                       mcAddress Address,
                       u32 BlockSize);
```

Input

`ECURefNum`	`ECURefNum` is the task reference which links to the selected ECU. This reference is originally returned from `mcECUSelect`.
`Address`	Configures the target address to be cleared in the ECU. `mcAddress` is a C struct consisting of: **Address** Specifies the address part of the target address. **Extension** Extension contains the extension part of the target address.
`BlockSize`	`BlockSize` determines the size of the block on which the checksum has to be calculated.

Output

Return Value

The return value indicates the status of the function call as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `mcStatusToString` function of the ECU M&C API to obtain a descriptive string for the return value.

Description

`mcCCPClearMemory` can be used to clear the contents of the flash memory prior to reprogramming it. The Memory Transfer Address 0 (MTA 0) will be set to the start of the memory block automatically by this function. The size parameter is the size of the block to be erased.

mcCCPDiagService

Purpose

Calls an implementation-specific diagnostic service on the ECU.

Format

```
mcTypeStatus        mcCCPDiagService(
                        mcTypeTaskRef ECURefNum,
                        u16 ServiceNo,
                        u8 Params[4],
                        u8 *ResultLength,
                        u8 *DataType);
```

Input

`ECURefNum`	`ECURefNum` is the task reference which links to the selected ECU. This reference is originally returned from `mcECUSelect`.
`ServiceNo`	`ServiceNo` determines the diagnostic service that will be executed inside the ECU. For more information about the services that are implemented in the ECU refer to the documentation for the ECU.
`Params`	`Params` passes an array of bytes to the ECU that might be needed by the ECU to run the diagnostic service. Since the definition of the parameters is specific to the implementation of the ECU, the parameters can only be passed as an array of bytes. It is your responsibility to pass the correct number of parameters in the correct byte ordering to this function.

Output

`*ResultLength`	`ResultLength` returns the number of bytes that can be uploaded from the ECU as a result of the execution of the service. The result of this service can be accessed by calling the function `mcCCPGetResult` right after `mcCCPDiagService`.
`*DataType`	`DataType` is a data type qualifier which provides information about the data type of the result of the diagnostic service.

Return Value

The return value indicates the status of the function call as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `mcStatusToString` function of the ECU M&C API to obtain a descriptive string for the return value.

Description

`mcCCPDiagService` implements the CCP command DIAG_SERVICE which calls a diagnostic service on the ECU and waits until it is finished. The selected `ServiceNo` specifies the diagnostic service that has to be executed inside the ECU. For more information about the available services that are implemented in the ECU refer to the documentation for the ECU.

The result of this service can be accessed by calling the function `mcCCPGetResult` right after `mcCCPDiagService`.

mcCCPDownload

Purpose

Downloads data to an ECU.

Format

```
mcTypeStatus        mcCCPDownload(
                        mcTypeTaskRef ECURefNum,
                        mcAddress Address,
                        u32 BlockSize
                        u8 *Data);
```

Input

`ECURefNum`	`ECURefNum` is the task reference which links to the selected ECU. This reference is originally returned from `mcECUSelect`.
`Address`	Configures the target address for the download operation in the ECU. `mcAddress` is a C struct consisting of: **Address** Specifies the address part of the target address. **Extension** Extension contains the extension part of the target address.
`BlockSize`	`BlockSize` determines the size of the data block to be downloaded.

Output

`Data`	Data pointer to the information to be downloaded.

Return Value

The return value indicates the status of the function call as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `mcStatusToString` function of the ECU M&C API to obtain a descriptive string for the return value.

Description

`mcCCPDownload` is used to download data to an ECU. The data will be stored starting at the selected **Address** and **Extension** in the ECU memory. The function is able to download more than 5 data bytes to the ECU. If the selected `BlockSize` is higher than 5 bytes `mcCCPDownload` performs several CCP DNLOAD commands, until all data bytes are downloaded to the ECU.

`mcCCPDownload` implements the CCP DNLOAD command defined by the CCP specification.

mcCCPGeneric

Purpose

Sends a generic CCP command.

Format

```
mcTypeStatus        mcCCPGeneric(
                        mcTypeTaskRef ECURefNum,
                        u8 Command,
                        u8 Data[6],
                        u32 Timeout,
                        u8 *ErrorCode,
                        u8 ReturnValue[5]);
```

Input

`ECURefNum`	`ECURefNum` is the task reference which links to the selected ECU. This reference is originally returned from `mcECUSelect`.
`Command`	`Command` is the CCP command code to send to the ECU.
`Data`	`Data` contains the parameters of the command as an array of bytes. For more information about the parameters of the (user defined) commands implemented in the ECU, refer to the documentation for the ECU.
`Timeout`	`Timeout` specifies the maximum number of milliseconds to wait for a response from the ECU. If the `Timeout` expires before an ECU response occurs, the error `mcErrorTimeout` is returned.

Output

`ErrorCode`	`ErrorCode` describes the error returned from the ECU during the communication.
`ReturnValue`	`ReturnValue` may contain an array of bytes returned from the ECU as a response to the command sent to the ECU.

Return Value

The return value indicates the status of the function call as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `mcStatusToString` function of the ECU M&C API to obtain a descriptive string for the return value.

Description

`mcCCPGeneric` can be used to send commands to the ECU that are not defined by the CCP specification. The command code and the parameters of this command will be sent to the ECU and the data returned by the ECU will be returned in the parameter `ReturnValue`. Since the ECU M&C driver has no knowledge of the parameters of the command and their data types, all parameters and return values will be passed as an array of bytes. Therefore you are responsible for the correct type casting of all parameters and return values of this command. Make sure that all parameters are passed in the correct byte ordering for this function. For more information about the (user defined) commands and their parameters refer to the documentation for the ECU.

mcCCPGetActiveCalPage

Purpose

Retrieves the ECU Memory Transfer Address pointer to the calibration data page.

Format

```
mcTypeStatus        mcCCPGetActiveCalPage(
                       mcTypeTaskRef ECURefNum,
                       mcAddress *Address);
```

Input

`ECURefNum`	`ECURefNum` is the task reference which links to the selected ECU. This reference is originally returned from `mcECUSelect`.

Output

`Address`	Returns the address for the active calibration page in the ECU. `mcAddress` is a C struct consisting of: **Address** Specifies the address part of the address. **Extension** Extension contains the extension part of the address.

Return Value

The return value indicates the status of the function call as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `mcStatusToString` function of the ECU M&C API to obtain a descriptive string for the return value.

Description

`mcCCPGetActiveCalPage` retrieves the start address of the active calibration data page in the ECU memory.

mcCCPGetResult

Purpose

Uploads data from the ECU when the Memory Transfer Address pointer 0 (MTA0) has been set.

Format

```
mcTypeStatus        mcCCPGetResult(
                        mcTypeTaskRef ECURefNum,
                        u32 BlockSize,
                        u8 *Data);
```

Input

`ECURefNum`	`ECURefNum` is the task reference which links to the selected ECU. This reference is originally returned from `mcECUSelect`.
`BlockSize`	`BlockSize` determines the size of the data block to be uploaded from the ECU.

Output

`Data`	`Data` contains the data uploaded from the ECU memory.

Return Value

The return value indicates the status of the function call as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `mcStatusToString` function of the ECU M&C API to obtain a descriptive string for the return value.

Description

This function uploads data from the ECU. It is assumed that the Memory Transfer Address 0 (MTA0) has already been set to the start address of the data to be uploaded. Functions like `mcCCPActionService` or `mcCCPDiagService` implicitly set the Memory Transfer Address 0 (MTA0) to the beginning of their result. To upload data from a specified address, use `mcCCPUpload` instead.

mcCCPGetSessionStatus

Purpose

Retrieves the current status of the Calibration Session.

Format

```
mcTypeStatus        mcCCPGetSessionStatus(
                        mcTypeTaskRef ECURefNum,
                        u8 *SessionStatus,
                        u8 *Qualifier0,
                        u8 *Qualifier1);
```

Input

`ECURefNum`	`ECURefNum` is the task reference which links to the selected ECU. This reference is originally returned from `mcECUSelect`.

Output

`SessionStatus`	The current `SessionStatus` which is returned from the ECU.
`Qualifier0`	`Qualifier0` describes an additional status qualifier.
`Qualifier1`	`Qualifier1` describes an additional status qualifier.

Return Value

The return value indicates the status of the function call as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `mcStatusToString` function of the ECU M&C API to obtain a descriptive string for the return value.

Description

`mcCCPGetSessionStatus` retrieves the current calibration session status of the ECU. The return value `SessionStatus` is a bit mask that represents several session states inside the ECU. `Qualifier0` and `Qualifier1` contain additional status information. The content of these parameters is ECU-specific and not defined by CCP. For more information about the parameter `SessionStatus`, refer to the description of `mcCCPSetSessionStatus`.

mcCCPGetVersion

Purpose

Retrieves CCP version implemented in the ECU.

Format

```
mcTypeStatus        mcCCPGetVersion(
                       mcTypeTaskRef ECURefNum,
                       u8 *MajorVersion,
                       u8 *MinorVersion);
```

Input

`ECURefNum`	`ECURefNum` is the task reference which links to the selected ECU. This reference is originally returned from `mcECUSelect`.

Output

`MajorVersion`	`MajorVersion` returns the major version number of the CCP implementation.
`MinorVersion`	`MinorVersion` returns the minor version number of the CCP implementation.

Return Value

The return value indicates the status of the function call as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `mcStatusToString` function of the ECU M&C API to obtain a descriptive string for the return value.

Description

`mcCCPGetVersion` can be used to query the CCP version implemented in the ECU. This command performs a mutual identification of the protocol version in the slave device to agree on a common protocol version.

`mcCCPGetVersion` implements the CCP command GET_CCP_VERSION defined by the CCP specification.

mcCCPMoveMemory

Purpose

Moves a memory block on the ECU.

Format

```
mcTypeStatus        mcCCPMoveMemory(
                        mcTypeTaskRef ECURefNum,
                        mcAddress Source,
                        mcAddress Destination,
                        u32 BlockSize);
```

Input

`ECURefNum`	`ECURefNum` is the task reference which links to the selected ECU. This reference is originally returned from `mcECUSelect`.
`Source`	Configures the source address for the memory move operation in the ECU. `mcAddress` is a C struct consisting of: **Address** Specifies the address part of the source address. **Extension** Extension contains the extension part of the source address.
`Destination`	Configures the destination address for the memory move operation in the ECU. `mcAddress` is a C struct consisting of: **Address** Specifies the address part of the destination address. **Extension** Extension contains the extension part of the destination address.
`BlockSize`	`BlockSize` determines the size of the data block on which the checksum has to be calculated.

Output

Return Value

The return value indicates the status of the function call as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `mcStatusToString` function of the ECU M&C API to obtain a descriptive string for the return value.

Description

`mcCCPMoveMemory` is used to move the memory contents of an ECU from one memory location to another. Upfront this function sets the Memory Transfer Address pointers MTA0 as defined in the source struct and MTA1 as defined in the destination struct to appropriate values.

`mcCCPMoveMemory` implements the CCP command MOVE defined by the CCP specification.

mcCCPProgram

Purpose

Programs a memory block on the ECU.

Format

```
mcTypeStatus        mcCCPProgram(
                        mcTypeTaskRef ECURefNum,
                        mcAddress Address,
                        u32 BlockSize,
                        u8 *Data);
```

Input

`ECURefNum`	`ECURefNum` is the task reference which links to the selected ECU. This reference is originally returned from `mcECUSelect`.
`Address`	Configures the target address for the programming operation in the ECU. `mcAddress` is a C struct consisting of: **Address** Specifies the address part of the programming address. **Extension** Extension contains the extension part of the address.
`BlockSize`	`BlockSize` determines the size of the data block which will be transferred to the ECU and used for programming from the MTA0 target.
`Data`	`data` contains the byte array that will be transmitted to the ECU.

Output

Return Value

The return value indicates the status of the function call as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `mcStatusToString` function of the ECU M&C API to obtain a descriptive string for the return value.

Description

`mcCCPProgram` implements the CCP command PROGRAM. The command is used to program the specified data into nonvolatile ECU memory (Flash, EEPROM, etc). Programming starts at the selected MTA0 address and extension defined in the `Address` struct. The `mcCCPProgram` function auto-increments the ECU MTA0 address.

mcCCPSelectCalPage

Purpose

Sets the specified address to be the start address of the calibration data page.

Format

```
mcTypeStatus        mcCCPSelectCalPage(
                       mcTypeTaskRef ECURefNum,
                       mcAddress Address);
```

Input

`ECURefNum`	`ECURefNum` is the task reference which links to the selected ECU. This reference is originally returned from `mcECUSelect`.
`Address`	Configures the target address for the programming operation in the ECU. `mcAddress` is a C struct consisting of: **Address** Specifies the address part of the target address. **Extension** Extension contains the extension part of the address.

Output

Return Value

The return value indicates the status of the function call as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `mcStatusToString` function of the ECU M&C API to obtain a descriptive string for the return value.

Description

`mcCCPSelectCalPage` implements the CCP command SELECT_CAL_PAGE. This command sets the beginning of the calibration data page to the specified address within the ECU.

mcCCPSetSessionStatus

Purpose

Updates the ECU with the current state of the calibration session.

Format

```
mcTypeStatus        mcCCPSetSessionStatus(
                       mcTypeTaskRef ECURefNum,
                       u8 SessionStatus);
```

Input

`ECURefNum`	`ECURefNum` is the task reference which links to the selected ECU. This reference is originally returned from `mcECUSelect`.
`SessionStatus`	`SessionStatus` contains the new status to be set in the ECU.

Output

Return Value

The return value indicates the status of the function call as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `mcStatusToString` function of the ECU M&C API to obtain a descriptive string for the return value.

Description

`mcCCPSetSessionStatus` implements the CCP command SET_S_STATUS and is used to keep the ECU informed about the current state of the calibration session. The session status bits of an ECU can be read and written. Possible conditions are: reset on power-up, session log-off, and in applicable error conditions. The calibration session status is organized as a bit mask with the following assignment.

Table 6-3. Bit Mask Assignments for Calibration Session Status

Bit	Name	Description
0	CAL	Calibration data initialized.
1	DAQ	DAQ list(s) initialized.

Table 6-3. Bit Mask Assignments for Calibration Session Status (Continued)

Bit	Name	Description
2	RESUME	Request to save DAQ set-up during shutdown in CCP slave. CCP slave automatically restarts DAQ after start-up.
3	Reserved	—
4	Reserved	—
5	Reserved	—
6	STORE	Request to save calibration data during shut-down in CCP slave.
7	RUN	Session in progress.

mcCCPUpload

Purpose

Uploads data from an ECU.

Format

```
mcTypeStatus        mcCCPUpload(
                        mcTypeTaskRef ECURefNum,
                        mcAddress Address,
                        u32 BlockSize,
                        u8 *Data);
```

Input

`ECURefNum`	`ECURefNum` is the task reference which links to the selected ECU. This reference is originally returned from `mcECUSelect`.
`Address`	Configures the source address for the upload operation in the ECU. `mcAddress` is a C struct consisting of: **Address** Specifies the address part of the source address. **Extension** Extension contains the extension part of the address.
`BlockSize`	`BlockSize` is the size of the data block in bytes to be uploaded.

Output

`Data`	`Data` is a byte array which receives the uploaded data information from the ECU.

Return Value

The return value indicates the status of the function call as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `mcStatusToString` function of the ECU M&C API to obtain a descriptive string for the return value.

Description

`mcCCPUpload` implements the CCP command UPLOAD. A data block of the specified length starting at the specified address will be uploaded from the ECU. This function will set the Memory Transfer Address pointer MTA0 to the appropriate value as defined in the `Address` struct.

mcCharacteristicRead

Purpose

Reads all data from a named Characteristic on the ECU which is identified by the ECU Reference handle.

Format

```
mcTypeStatus        mcCharacteristicRead(
                       mcTypeTaskRef ECURefNum,
                       char *CharacteristicName,
                       f64 *Values,
                       u32 NumberOfValues);
```

Input

`ECURefNum`	`ECURefNum` is the task reference which links to the selected ECU. This reference is originally returned from `mcECUSelect`.
`CharacteristicName`	`CharacteristicName` is the name of the Characteristic defined in the A2L database file.
`NumberOfValues`	Specifies the number of values to read. To determine the dimension of the Characteristic use the `mcGetProperty` function upfront using the parameter `mcPropChar_Dimension`. To determine the size of each dimension use the `mcGetProperty` function with the parameter `mcPropChar_Sizes`.

Output

`Values`	Returns a single value, a 1-dimensional array, or a 2-dimensional array of values for the selected Characteristic.

Return Value

The return value indicates the status of the function call as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `mcStatusToString` function of the ECU M&C API to obtain a descriptive string for the return value.

Description

`mcCharacteristicRead` reads values from a named Characteristic on the ECU which is identified by the ECU Reference handle. The function returns a double, 1D, or 2D array.

mcCharacteristicReadSingleValue

Purpose

Reads a single value from a named Characteristic on the ECU which is identified by the ECU Reference handle.

Format

```
mcTypeStatus        mcCharacteristicReadSingleValue(
                        mcTypeTaskRef ECURefNum,
                        char *CharacteristicName,
                        f64 Value,
                        u32 X,
                        u32 Y);
```

Input

`ECURefNum`	`ECURefNum` is the task reference which links to the selected ECU. This reference is originally returned from `mcECUSelect`.
`CharacteristicName`	`CharacteristicName` is the name of the Characteristic defined in the A2L database file.
`X`	`X` is the horizontal index if the Characteristic consists of 1 or 2 dimensions.
`Y`	`Y` is the vertical index if the Characteristic consists of 2 dimensions.

Output

`Value`	Returns a single value from the selected Characteristic.

Return Value

The return value indicates the status of the function call as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `mcStatusToString` function of the ECU M&C API to obtain a descriptive string for the return value.

Description

`mcCharacteristicReadSingleValue` reads a value from a named Characteristic on the ECU which is identified by the ECU Reference handle. The value to be read is identified by the `X` and `Y` indices.

If the Characteristic array is 0-dimensional, `X` and `Y` can be set to 0.

If the Characteristic array is 1-dimensional, `Y` can be set to 0.

mcCharacteristicWrite

Purpose

Downloads data to a Characteristic for a selected ECU.

Format

```
mcTypeStatus        mcCharacteristicWrite(
                        mcTypeTaskRef ECURefNum,
                        char *CharacteristicName,
                        f64 Values,
                        u32 NumberOfValues);
```

Input

`ECURefNum`	`ECURefNum` is the task reference which links to the selected ECU. This reference is originally returned from `mcECUSelect`.
`CharacteristicName`	`CharacteristicName` is the name of the Characteristic defined in the A2L database file.
`Values`	`Values` contains a pointer to a double, a double 1D, or 2D array which will be sent to the ECU.
`NumberOfValues`	Specifies the number of values to write for the task.

Output

Return Value

The return value indicates the status of the function call as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `mcStatusToString` function of the ECU M&C API to obtain a descriptive string for the return value.

Description

`mcCharacteristicWrite` writes the value(s) of a named Characteristic to an ECU identified by the ECU reference handle `ECURefNum`.

mcCharacteristicWriteSingleValue

Purpose

Writes a single value to a named Characteristic on the ECU.

Format

```
mcTypeStatus        mcCharacteristicWriteSingleValue(
                        mcTypeTaskRef ECURefNum,
                        char *CharacteristicName,
                        f64 Value,
                        u32 X,
                        u32 Y);
```

Input

`ECURefNum`	`ECURefNum` is the task reference which links to the selected ECU. This reference is originally returned from `mcECUSelect`.
`CharacteristicName`	`CharacteristicName` is the name of the Characteristic defined in the A2L database file, to which the values will be written.
`Value`	`Value` contains the value which will be sent to the ECU.
`X`	`X` refers to the array offset of the Characteristic defined in the A2L database file as 1- or 2-dimensional. If the Characteristic is defined as 0-dimensional you can set `X` to 0.
`Y`	`Y` refers to the array offset of the Characteristic defined in the A2L database file as 2-dimensional. If the Characteristic is defined as 0- or 1-dimensional you can set `Y` to 0.

Output

Return Value

The return value indicates the status of the function call as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `mcStatusToString` function of the ECU M&C API to obtain a descriptive string for the return value.

Description

`mcCharacteristicWriteSingleValue` writes a value to a named Characteristic on the ECU which is identified by the ECU Reference handle `ECURefNum`. The location to which the value will be written is identified by the `X` and `Y` indices. If the Characteristic array is 0- or 1-dimensional, `Y` and/or `X` can be set to 0.

mcDAQClear

Purpose

Stops communication for the Measurement task and clears the task.

Format

```
mcTypeStatus          mcDAQClear(
                         mcTypeTaskRef *DAQRefNum);
```

Input

`DAQRefNum`	`DAQRefNum` is the task reference which links to the selected Measurement task. This reference is originally returned from `mcDAQInitialize`.

Output

Return Value

The return value indicates the status of the function call as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `mcStatusToString` function of the ECU M&C API to obtain a descriptive string for the return value.

Description

`mcDAQClear` must always be the final function called for a Measurement task. If you do not use `mcDAQClear`, the remaining Measurement task configuration can cause problems in the execution of subsequent applications. Because this function clears the Measurement task, the Measurement task reference is not given as an output but will be transferred into an ECU reference task handle. To change properties of a running Measurement task, use `mcDAQStop` to stop the task, `mcSetProperty` to change the desired DAQ property, then `mcDAQStart` to restart the Measurement task.

mcDAQInitialize

Purpose

Initializes a Measurement task for the specified Measurement channel list.

Format

```
mcTypeStatus        mcDAQInitialize(
                        cstr MeasurementNames,
                        mcTypeTaskRef ECURefNum,
                        i32 DAQMode,
                        u32 DTO_ID,
                        f64 SampleRate,
                        mcTypeTaskRef *DAQRefNum);
```

Input

`MeasurementNames`	Comma-separated list of Measurement names to initialize as a task. You can type in the channel list as a string constant or you can obtain the list from an A2L database file by using the `mcGetNames` function.
`ECURefNum`	`ECURefNum` is the task reference which links to the selected ECU. This reference is originally returned from `mcECUSelect`.
`DAQMode`	`DAQMode` specifies the I/O mode for the task. For an overview of the I/O modes, including figures, refer to the *Basic Programming Model* section of Chapter 4, *Using the ECU M&C API*.

mcDAQModeDAQList

Data is transmitted automatically by the ECU using DAQ lists. The data can be read back with the `mcDAQRead` as Single point data using sample rate = 0 or as waveform using a sample rate > 0. Input channel data is received from the DAQ messages.

mcDAQModePolling

In this mode the data from the Measurement task are uploaded from the ECU whenever `mcDAQRead` is called.

`DTO_ID`	`DTO_ID` is the CAN identifier for the **D**ata **T**ransmission **O**bject (**DTO**) used to transmit the data from the DAQ lists to the host. The default value is -1 which means that the `DTO ID` used to transmit the DAQ list data is the same that is used for the rest of the CCP communication.

`SampleRate`	`SampleRate` specifies the timing to use for samples of the (NI-CAN) task. The sample rate is specified in Hertz (samples per second). A sample rate of zero means to sample immediately. For a `DAQMode` of **mcDAQModeDAQList**, `SampleRate` of zero means that `mcDAQRead` returns a single sample from the most recent messages received, and greater than zero means that `mcDAQRead` returns samples timed at the specified rate. For `DAQMode` of **mcDAQModePolling**, `SampleRate` is ignored.

Output

`DAQRefNum`	`DAQRefNum` is the reference handle for the Measurement task. Use this Measurement task reference in subsequent M&C DAQ functions for this task.

Return Value

The return value indicates the status of the function call as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `mcStatusToString` function of the ECU M&C API to obtain a descriptive string for the return value.

Description

`mcDAQInitialize` does not start the transmission of the DAQ lists on the ECU. This enables you to use `mcSetProperty` to change the properties of a Measurement task. After you change properties, use `mcDAQStart` to start the transmission of the Measurement task.

mcDAQRead

Purpose

Reads samples from a Measurement task. Samples are obtained from received CAN messages.

Format

```
mcTypeStatus        mcDAQRead(
                        mcTypeTaskRef DAQRefNum,
                        u32 NumberOfSamplesToRead,
                        nctTypeTimestamp *StartTime,
                        nctTypeTimestamp *DeltaTime,
                        f64 *SampleArray,
                        u32 *NumberOfSamplesReturned);
```

Input

`DAQRefNum`	`DAQRefNum` is the task reference from the previous Measurement task function. The task reference is originally returned from `mcDAQInitialize`, and then reused by subsequent Measurement task functions.
`NumberOfSamplesToRead`	Specifies the number of samples to read for the task. For single-sample input, pass 1 to this parameter. If the initialized sample rate is zero, you must pass `NumberOfSamplesToRead` no greater than 1. `SampleRate` of zero means `mcDAQRead` immediately returns a single sample from the most recent message(s) received.

Output

`StartTime`	Returns the time of the first CAN sample in `SampleArray`. This parameter is optional. If you pass NULL for the `StartTime` parameter, the mcDAQRead function proceeds normally. If the initialized sample rate is greater than zero, the `StartTime` is determined by the sample timing. If the initialized `SampleRate` is zero, the `StartTime` is zero, because the most recent sample is returned regardless of timing. `StartTime` uses the mcTypeTimestamp data type. The `mcTypeTimestamp` data type is a 64-bit unsigned integer compatible with the Microsoft Win32 `FILETIME` type. This absolute time is kept in a Coordinated Universal Time (UTC)

	format. UTC time is loosely defined as the current date and time of day in Greenwich, England. Microsoft defines its UTC time (`FILETIME`) as a 64-bit counter of 100 ns intervals that have elapsed since 12:00 a.m., January 1, 1601. Because `mcTypeTimestamp` is compatible with Win32 `FILETIME`, you can pass it into the Win32 `FileTimeToLocalFileTime` function to convert it to the local time zone, and then pass the resulting local time to the Win32 `FileTimeToSystemTime` function to convert to the Win32 `SYSTEMTIME` type. `SYSTEMTIME` is a struct with fields for year, month, day, and so on. For more information on Win32 time types and functions, refer to the Microsoft Win32 documentation.
`DeltaTime`	Returns the time between each sample in `SampleArray`. This parameter is optional. If you pass NULL for the `DeltaTime` parameter, the `mcDAQRead` function proceeds normally. If the initialized sample rate is greater than zero, the `DeltaTime` is determined by the sample timing. If the initialized sample rate is zero, the `DeltaTime` is zero, because the most recent sample is returned regardless of timing. `DeltaTime` uses the `mcTypeTimestamp` data type. The delta time is a relative 64-bit counter of 100 ns intervals, not an absolute UTC time. Nevertheless, you can use functions like the Win32 `FileTimeToSystemTime` function to convert to the Win32 `SYSTEMTIME` type. In addition, you can use the 32-bit `LowPart` of `DeltaTime` to obtain a simple 100 ns count, because values for `SampleRate` as slow as 0.4 Hz are still limited to a 32-bit 100 ns count.
`SampleArray`	Returns a 2D array, one array for each channel initialized in the task. The array of each channel must have `NumberOfSamplesToRead` entries allocated. The order of channel entries in `SampleArray` is the same as the order in the original `ChannelList`. If you need to determine the number of channels in the task after initialization, get the `mcPropDAQ_NumChannels` property for the task reference. If no message has been received since you started the task, 0 is returned as default value for of the channel in all entries of `SampleArray`.
`NumberOfSamplesReturned`	`NumberOfSamplesReturned` indicates the number of samples returned for each channel in `SampleArray`. The remaining entries are left unchanged (zero).

Return Value

The return value indicates the status of the function call as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `mcStatusToString` function of the ECU M&C API to obtain a descriptive string for the return value.

Description

If the initialized `SampleRate` is greater than zero, this function returns an array of samples, each of which indicates the value of the CAN channel at a specific point in time. The `mcDAQRead` function waits for these samples to arrive in time before returning. In other words, the `SampleRate` specifies a virtual clock that copies the most recent value from CAN messages for each sample time. The changes in sample values from message to message enable you to view the channel over time, such as for comparison with other CAN or DAQ input channels. To avoid internal waiting, you can use `mcGetProperty` to obtain `nctPropSamplesPending` property, and pass that as the `NumberOfSamplesToRead` parameter to `mcDAQRead`.

If the initialized `SampleRate` is zero, `mcDAQRead` immediately returns a single sample from the most recent message(s) received. For this single-point read, you must pass the `NumberOfSamplesToRead` parameter as 1. You can use the return value of `mcDAQRead` to determine whether a new message has been received since the previous call to `mcDAQRead` (or `mcDAQStart`). If no message has been received, the warning code CanWarnOldData is returned. If a new message has been received, the success code 0 is returned. If no message has been received since you started the task, the default value of the channel (`nctPropChanDefaultValue`) is returned in all entries of `SampleArray`.

mcDAQStart

Purpose

Starts the transmission of the DAQ lists assigned to the Measurement task on the ECU.

Format

```
mcTypeStatus        mcDAQStart(
                       mcTypeTaskRef DAQRefNum);
```

Input

`DAQRefNum`	`DAQRefNum` is the task reference from the previous Measurement task function. The task reference is originally returned from `mcDAQInitialize`, and then reused by subsequent Measurement task functions.

Output

Return Value

The return value indicates the status of the function call as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `mcStatusToString` function of the ECU M&C API to obtain a descriptive string for the return value.

Description

`mcDAQStart` is an optional command to start communication for a M&C Measurement task. If not performing `mcDAQStart` before using `mcDAQRead` the Measurement task will be started by the first call of `mcDAQRead`. After you start the transmission of the DAQ lists, you can no longer change the configuration of the Measurement task with `mcSetProperty`.

mcDAQStop

Purpose

Stops transmission of the DAQ lists for the specified Measurement task.

Format

```
mcTypeStatus        mcDAQStop(
                       mcTypeTaskRef DAQRefNum);
```

Input

`DAQRefNum`	`DAQRefNum` is the task reference from the previous Measurement task function. The task reference is originally returned from `mcDAQInitialize`, and then reused by subsequent Measurement task functions.

Output

Return Value

The return value indicates the status of the function call as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `mcStatusToString` function of the ECU M&C API to obtain a descriptive string for the return value.

Description

`mcDAQStop` stops the transmission of the DAQ lists on the ECU so that you can change the configuration of the task, such as by using `mcSetProperty`. After you change the configuration, use `mcDAQStart` to start the task again. This function does not clear the configuration for the Measurement task.

`mcDAQClear` must always be the last ECU M&C function for each Measurement task.

mcDatabaseClose

Purpose

Closes a specified A2L Database.

Format

```
mcTypeStatus        mcDatabaseClose(
                       mcTypeTaskRef *DBRefNum);
```

Input

`DBRefNum`	`DBRefNum` is the task reference from the initial database task function. The database task reference is originally returned from `mcDatabaseOpen`.

Output

Return Value

The return value indicates the status of the function call as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `mcStatusToString` function of the ECU M&C API to obtain a descriptive string for the return value.

Description

`mcDatabaseClose` must always be the final ECU M&C function called for each database task. If you do not use the `mcDatabaseClose` function, the remaining task configurations can cause problems in the execution of subsequent Measurement and Calibration applications.

mcDatabaseOpen

Purpose

Opens a specified A2L Database.

Format

```
mcTypeStatus        mcDatabaseOpen(
                        cstr Database,
                        mcTypeTaskRef *DBRefNum);
```

Input

`Database`	`Database` is a path to an A2L database file from which to get Measurement or calibration channel names. The file must use a `.A2L` extension. You can generate A2L database files with several 3rd party tools.

Output

`DBRefNum`	`DBRefNum` is the task reference from the initial database task function. The database task reference is originally returned from `mcDatabaseOpen`.

Return Value

The return value indicates the status of the function call as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `mcStatusToString` function of the ECU M&C API to obtain a descriptive string for the return value.

Description

The `mcDatabaseOpen` function does not start communication. This enables you to query all defined ECU names in the A2L Database using the `mcGetNames` function and selecting the property value ECU Names.

To use the ECU M&C Toolkit on a LabVIEW RT system, you must download your ASAM MCD 2MC database (`*.A2L`) file to the RT target.

mcECUConnect

Purpose

Establishes communication to the selected ECU through CCP. After a successful ECU Connect you can create a Measurement task or read/write a Characteristic.

Format

```
mcTypeStatus        mcECUConnect(

                       mcTypeTaskRef ECURefNum);
```

Input

`ECURefNum`	`ECURefNum` is the task reference which links to the selected ECU. This reference is originally returned from `mcECUSelect`.

Output

Return Value

The return value indicates the status of the function call as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `mcStatusToString` function of the ECU M&C API to obtain a descriptive string for the return value.

Description

`mcECUConnect` implements the CCP CONNECT command. It establishes a logical connection to an ECU, using the provided ECU Reference handle `ECURefNum`. Unless a slave device (ECU) is unconnected, it must not execute or respond to any command sent by the application. The only exception to this rule is the Test command, to which the CCP slave with the specific address may return an acknowledgement. Only a single CCP slave can be connected to the application at a time.

mcECUDeselect

Purpose

Deselects an ECU and invalidates the ECU reference handle.

Format

```
mcTypeStatus        mcECUDeselect(
                        mcTypeTaskRef *ECURefNum);
```

Input

`ECURefNum`	`ECURefNum` is the task reference which links to the selected ECU. This reference is originally returned from `mcECUSelect`.

Output

Return Value

The return value indicates the status of the function call as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `mcStatusToString` function of the ECU M&C API to obtain a descriptive string for the return value.

Description

`mcECUDeselect` deselects the an ECU and clears all internal driver data stored for this ECU. After calling this function it is no longer possible to communicate with the specified ECU. The task reference `ECURefNum` will be transferred into a database handle `DBRefNum`.

mcECUDisconnect

Purpose

Disconnects CCP communication to the selected ECU.

Format

```
mcTypeStatus        mcECUDisconnect(

                       mcTypeTaskRef ECURefNum);
```

Input

`ECURefNum`	`ECURefNum` is the task reference which links to the selected ECU. This reference is originally returned from `mcECUSelect`.

Output

Return Value

The return value indicates the status of the function call as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `mcStatusToString` function of the ECU M&C API to obtain a descriptive string for the return value.

Description

`mcECUDisconnect` implements the CCP command DISCONNECT. `mcECUDisconnect` disconnects the specified CCP slave from the actual communication and ends the calibration session. When the calibration session is terminated, all CCP DAQ lists of the device will be stopped and cleared and the protection masks of the device will be set to their default values.

`mcECUDisconnect` is an optional command as disconnecting from the ECU is performed by the function `mcECUDeselect`.

mcECUSelect

Purpose

Selects an ECU from the names stored in an A2L database.

Format

```
mcTypeStatus        mcECUSelect(
                       mcTypeTaskRef DBRefNum,
                       cstr ECUName,
                       i32 Interface,
                       mcTypeTaskRef *ECURefNum);
```

Input

`DBRefNum`	`DBRefNum` is the task reference from the initial database task function. The database task reference is originally returned from `mcDatabaseOpen`.
`ECUName`	`ECUName` is the selected ECU name out of an A2L Database file with which to initialize all subsequent tasks.
`Interface`	Specifies the CAN interface to use for this task. The interface input uses an enumeration in which value 0 selects CAN0, value 1 selects CAN1, and so on. As the ECU M&C API is based on the NI-CAN Channel API, the NI-CAN Frame API cannot be used on the same CAN network interface simultaneously. If the CAN network interface is already initialized in the Frame API, this function returns an error.

Output

`ECURefNum`	`ECURefNum` is the task reference which links to the selected ECU.

Return Value

The return value indicates the status of the function call as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `mcStatusToString` function of the ECU M&C API to obtain a descriptive string for the return value.

Description

`mcECUSelect` creates an ECU reference handle to the selected ECU name. The `mcECUSelect` function does not start communication. This enables you to use `mcSetProperty` to change the properties of an ECU task. After you change properties use `mcECUConnect` to start communication for the task and logically connect to the selected ECU.

mcGetNames

Purpose

Retrieves a comma-separated list of ECU, Measurement, or Characteristic names from a specified A2L database.

Format

```
mcTypeStatus        mcGetNames(
                        mcTypeTaskRef RefNum,
                        u32 Type,
                        cstr ECUName,
                        u32 SizeOfNamesList,
                        str NameList);
```

Input

`RefNum`	`RefNum` is any ECU M&C task reference which consists of a valid link to the opened A2L database (`DBRefNum`), a selected ECU (`ECURefNum`) or a Measurement task (`DAQRefNum`). `RefNum` has to be valid for the related `Type`.
`Type`	Specifies the `Type` of names to return. 0—**mcTypeECUNames** Return list of ECU names. You can pass one of the returned names to `mcECUSelect`. 1—**mcTypeMeasurementNames** Return list of Measurement names. You can pass the returned `NamesList` to `mcDAQInitialize`. 2—**mcTypeCharacteristicNames** Return list of Characteristic Names. You can pass a single name out of the `NamesList` to `mcCharacteristicWrite` or `mcCharacteristicRead`. 3—**mcTypeEventChannelNames** Return list of Event Channel names.
`ECUName`	If the `Type` = **mcTypeMeasurementNames** or `Type` = **mcTypeCharacteristicNames** and `RefNum` contains a **DBRefNum**, the corresponding ECU name has to be referenced in order to access ECU specific properties. If `RefNum` contains an **ECURefNum** or **DAQRefNum** the parameter `ECUName` will be ignored and can be set to NULL.

`SizeOfNamesList`	Size of the buffer provided to take the names list. After calling `mcGetNamesLength`, you can allocate an array of size `SizeofNamesList`, and then pass that array to `mcGetNames` using the same input parameters. This ensures that `mcGetNames` will return all names without error.

Output

`NameList`	Returns the comma-separated list of names specified by `Type`.

Return Value

The return value indicates the status of the function call as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `mcStatusToString` function of the ECU M&C API to obtain a descriptive string for the return value.

Description

Get a comma-separated list of ECU, Measurement, Characteristic, or Event Channel names from a specified A2L database file.

mcGetNamesLength

Purpose

Retrieves the amount of memory required to store the names returned by `mcGetNames`.

Format

```
mcTypeStatus        mcGetNamesLength(
                        mcTypeTaskRef RefNum,
                        u32 Type,
                        cstr ECUName,
                        u32 *SizeOfNamesList);
```

Input

`RefNum`	`RefNum` is any ECU M&C task reference which consists of a valid link to the opened A2L database (`DBRefNum`), a selected ECU (`ECURefNum`) or a Measurement task (`DAQRefNum`). `RefNum` has to be valid for the related `Type`.
`Selector`	Specifies the `Type` of names to return. 0—**mcTypeECUNames** Return list of ECU names. 1—**mcTypeMeasurementNames** Return list of Measurement names. 2—**mcTypeCharacteristicNames** Return list of Characteristic Names. 3—**mcTypeEventChannelNames** Return list of Event Channel names.
`ECUName`	If the `Type` = **mcTypeMeasurementNames** or `Type` = **mcTypeCharacteristicNames** and **RefNum** contains a **DBRefNum**, the corresponding ECU name has to be referenced in order to access ECU specific properties. If **RefNum** contains an **ECURefNum** or **DAQRefNum** the parameter `ECUName` will be ignored and can be set to NULL.

Output

`SizeOfNamesList`	Number of bytes required for `mcGetNames` to return all names for the specified ECUName and `Type`. After calling `mcGetNamesLength`, you can allocate an array of size `SizeofNamesList`, then pass that array to `mcGetNames` using the same input parameters. This ensures that `mcGetNames` will return all names without error.

Return Value

The return value indicates the status of the function call as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `mcStatusToString` function of the ECU M&C API to obtain a descriptive string for the return value.

Description

After calling `mcGetNamesLength`, you can allocate an array of size `SizeofNamesList`, then pass that array to `mcGetNames` using the same input parameters. This ensures that `mcGetNames` will return all names without error.

mcGetProperty

Purpose

Retrieves a property of the driver, the database, the ECU, a Characteristic, a Measurement, or a Measurement task.

Format

```
mcTypeStatus        mcGetProperty(
                       mcTypeTaskRef RefNum,
                       cstr Name,
                       u32 PropertyID,
                       u32 SizeOfValue,
                       void *Value);
```

Input

`RefNum`	`RefNum` is any ECU M&C task reference which consists of a valid link to the opened A2L database (`DBRefNum`), a selected ECU (`ECURefNum`) or a Measurement task (`DAQRefNum`). `RefNum` has to be valid for the related `PropertyID` type.
`Name`	Specifies an individual name (ECU name, Measurement channel name, or Characteristic name) within the task.
`PropertyID`	Selects the property to get. For a description of each property, including its data type and `PropertyId`, refer to the *Properties* section.
`SizeOfValue`	Number of bytes allocated for the `Value` output. This size normally depends on the data type listed in the description of the property.

Output

`Value`	Returns the property value. `PropertyId` determines the data type of the returned value.

Return Value

The return value indicates the status of the function call as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `mcStatusToString` function of the ECU M&C API to obtain a descriptive string for the return value.

Properties

Table 6-4. Values for PropertyID

Data Type	Name	Description
str	`mcPropDB_Filename`	Returns the A2L Database file name with which the task has been opened. The value of this property cannot be changed using `mcSetProperty`.
u32	`mcPropDB_Filename_Size`	Returns the number of bytes to be allocated if you call `mcGetProperty` with the parameter `mcPropDB_Filename`.
u32	`mcPropECU_DTO_ID`	Returns the **DTO ID** (**D**ata **T**ransmission **O**bject) which is used by the ECU to respond to CCP commands and send data and status information to the CCP master.
u32	`mcPropECU_CRO_ID`	Returns the **CRO ID** (**C**ommand **R**eceive **O**bject) which is used to send commands and data from the host to the slave device.
u32	`mcPropECU_Interface`	Returns the interface initialized for the task, such as with `mcDAQInitialize`.
u32	`mcPropECU_BaudRate`	Returns the baud rate in use.
u32	`mcPropECU_Station Address`	Returns the station address of the slave device. CCP is based on the idea that several ECUs can share the same CAN Arbitration IDs for CCP communication. To avoid communication conflicts, CCP defines a Station Address that has to be unique for all ECUs sharing the same CAN Arbitration IDs. Unless an ECU has been addressed by its Station Address, the ECU must not react to CCP commands sent by the CCP master.

Table 6-4. Values for PropertyID (Continued)

Data Type	Name	Description
u32	`mcPropECU_ByteOrder`	Returns the byte order of the CCP slave device. 0—**MSB_LAST** The CCP Slave device uses the MSB_LAST (Intel) byte ordering. 1—**MSB_FIRST** The CCP Slave device uses the MSB_FIRST (Motorola) byte ordering.
str	`mcPropECU_SeedKey_Cal`	Returns the file name of the SeedKey DLL used for Calibration purposes.
str	`mcPropECU_SeedKey_DAQ`	Returns the file name of the SeedKey DLL used for DAQ purposes.
str	`mcPropECU_SeedKey_Prog`	Returns the file name of the SeedKey DLL used for programming purposes.
str	`mcPropECU_Checksum`	Returns the file name of the Checksum DLL used for verifying the checksum.
str	`mcPropECU_Name`	Returns the name of the selected ECU opened by `mcECUSelect`.
[u8]	`mcPropECU_MasterID`	Returns CCP master ID information. This ID information is optional and specific to the ECU implementation. For more information about the CCP master ID information refer to the documentation for the ECU.
u8	`mcPropECU_ID_DataType`	Returns a data type qualifier of the slave device ID information. This ID information is optional and specific to the ECU implementation. For more information about the CCP slave ID information refer to the documentation for the ECU.
u8	`mcPropECU_ID_Length`	Returns the length of the slave device identifier in bytes.

Table 6-4. Values for PropertyID (Continued)

Data Type	Name	Description
[u8]	`mcPropECU_ID`	Returns the slave device identifier. This ID information is optional and specific to the ECU implementation. For more information about the CCP slave ID information refer to the documentation for the ECU.
u32	`mcPropECU_SeedKey_Cal_Size`	Returns the number of bytes to be allocated if you call `mcGetProperty` with the parameter `mcPropECU_SeedKey_Cal`.
u32	`mcPropECU_SeedKey_DAQ_Size`	Returns the number of bytes to be allocated if you call `mcGetProperty` with the parameter `mcPropECU_SeedKey_DAQ`.
u32	`mcPropECU_SeedKey_Prog_Size`	Returns the number of bytes to be allocated if you call `mcGetProperty` with the parameter `mcPropECU_SeedKey_Prog`.
u32	`mcPropECU_Checksum_Size`	Returns the number of bytes to be allocated if you call `mcGetProperty` with the parameter `mcPropECU_Checksum`.
u32	`mcPropECU_Name_Size`	Returns the number of bytes to be allocated if you call `mcGetProperty` with the parameter `mcPropECU_Name`.
nctType Taskref	`mcPropECU_DTO_Task`	NI-CAN Task reference to the CAN Task assigned to the DTO ID.
nctType Taskref	`mcPropECU_CRO_Task`	NI-CAN Task reference to the CAN Task assigned to the CRO ID.
u8	`mcPropECU_Single_Byte_DAQ_Lists`	Determines if an ECU supports single-byte or multi-byte DAQ list entries.
u16	`mcPropMeas_Address`	Returns the address of the selected Measurement in the memory of the control unit.
u8	`mcPropMeas_Extension`	Returns the address extension of the ECU address. This optional parameter may contain additional address information defined in the A2L database. For instance it can be used, to distinguish different address spaces of an ECU (multi-microcontroller devices).

Table 6-4. Values for PropertyID (Continued)

Data Type	Name	Description
u8	`mcPropMeas_Datatype`	Returns the data type of the Measurement task.
u32	`mcPropMeas_ByteOrder`	Returns the specified byte order: 0—Intel format Bytes are in little-endian order, with most-significant first. 1—Motorola format Bytes are in big-endian order, with least-significant first.
u32	`mcPropMeas_IsVirtual`	Virtual measurements are not transmitted by the ECU, but are calculated in the application.
f64	`mcPropMeas_Maximum`	Returns the maximum value of the Measurement.
f64	`mcPropMeas_Minimum`	Returns the minimum value of the Measurement.
u32	`mcPropMeas_ReadOnly`	Returns `TRUE` if the selected Measurement is read only and can only be accessed through `mcMeasurementRead`, or returns `FALSE` if the Measurement can be accessed through `mcMeasurementWrite` as well.
u16	`mcPropChar_Address`	Returns the address of the selected Characteristic in the memory of the ECU.
u8	`mcPropChar_Extension`	Returns additional address information. For instance it can be used to distinguish different address spaces of an ECU (multi-microcontroller devices).
u8	`mcPropChar_Datatype`	Returns the data type of the Characteristic.

Table 6-4. Values for PropertyID (Continued)

Data Type	Name	Description
u32	`mcPropChar_ByteOrder`	Returns the specified byte order: 0—Intel format Bytes are in little-endian order, with most-significant first. 1—Motorola format Bytes are in big-endian order, with least-significant first.
u32	`mcPropChar_Dimension`	Returns the dimension of the Characteristic: 0—0-dimensional: The Characteristic can be accessed (read/write) through a double value. 1—1-dimensional: The Characteristic can be accessed (read/write) through a one-dimensional array of double value. 2—2-dimensional: The Characteristic can be accessed (read/write) through a two-dimensional array of double value.
u32	`mcPropChar_Sizes`	Returns the Array Sizes for the X and Y directions of the Characteristic.
f64	`mcPropChar_X_Axis`	Returns X-axis values on which the Characteristic is defined. Valid if the selected Characteristic is 1- or 2-dimensional.
f64	`mcPropChar_Y_Axis`	Returns Y-axis values on which the Characteristic is defined, Valid if the selected Characteristic is 2-dimensional.
f64	`mcPropChar_Maximum`	Returns the Maximum value of the Characteristic.
f64	`mcPropChar_Minimum`	Returns the Minimum value of the Characteristic.
u32	`mcPropChar_ReadOnly`	Returns if a Characteristic is set to read only. In this case it is not allowed to call `mcCharacteristicWrite` for this Characteristic.

Table 6-4. Values for PropertyID (Continued)

Data Type	Name	Description
u32	`mcPropDAQ_Mode`	Returns the selected mode of an M&C Measurement task. 0—**DAQ List** The data is transmitted from the ECU in equidistant time intervals as defined in the A2L database. The data can be read back with `mcDAQRead` as Single point data using sample rate = 0, or as waveform using a sample rate > 0. Input channel data is received from the DAQ messages. Use `mcDAQRead` to obtain input samples as single-point, array, or waveform. 1—**Polling** In this mode the data from the Measurement task is uploaded from the ECU whenever `mcDAQRead` is called.
u32	`mcPropDAQ_DTO_ID`	Returns the DTO ID (**D**ata **T**ransmission **O**bject) which is used by the ECU to respond to send data from the DAQ lists to the CCP master.
f64	`mcPropDAQ_SampleRate`	Returns the selected Sample Rate in Hz for the M&C Measurement task.
u32	`mcPropDAQ_NumChannels`	Returns the number of channels initialized in a DAQ channel list of a M&C Measurement task. This is the number of array entries required when using `mcDAQRead`.
str	`mcPropDAQ_EventChannelName`	Returns the selected event channel name to which the Measurement task is assigned.
nctType Taskref	`mcPropDAQ_DTO_Task`	NI-CAN task reference to the CAN Task assigned to the DTO ID of the Measurement task.
u16	`mcPropDAQ_Prescaler`	Prescaler for the Measurement task on the ECU.
u32	`mcPropDAQ_EventChannelName_Size`	Returns the number of bytes to be allocated if you call `mcGetProperty` with the parameter `mcPropDAQ_EventChannelName`.

Table 6-4. Values for PropertyID (Continued)

Data Type	Name	Description
u32	`mcPropGen_Version_Major`	Returns the major version of the ECU M&C software, such as the 1 in version 1.2.5.
u32	`mcPropGen_Version_Minor`	Returns the minor version of the ECU M&C software, such as the 2 in version 1.2.5.
u32	`mcPropGen_Version_Update`	Returns the update version of the ECU M&C software, such as the 5 in version 1.2.5.
u32	`mcPropGen_Version_Build`	Returns the build number of the ECU M&C software. This number applies to Development, Alpha, and Beta phase only, and should be ignored for Release phase.
str	`mcPropGen_Version_Comment`	Returns a comment string for the ECU M&C software. If you received a custom release of ECU M&C from National Instruments, this comment often describes special features of the release.
u32	`mcPropGen_Version_Comment_Size`	Returns the number of bytes to be allocated if you call `mcGetProperty` with the parameter `mcPropGen_Version_Comment`.

mcMeasurementRead

Purpose

Reads a single Measurement value from the ECU.

Format

```
mcTypeStatus        mcMeasurementRead(
                       mcTypeTaskRef ECURefNum,
                       char *MeasurementName,
                       f64 *Value);
```

Input

`ECURefNum`	`ECURefNum` is the task reference which links to the selected ECU. This reference is originally returned from `mcECUSelect`.
`MeasurementName`	`MeasurementName` is the name of a Measurement channel stored in the A2L database file from which a Measurement value is to be read.

Output

`Value`	Returns a single sample for the Measurement channel initialized in `MeasurementName`.

Return Value

The return value indicates the status of the function call as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `mcStatusToString` function of the ECU M&C API to obtain a descriptive string for the return value.

Description

`mcMeasurementRead` performs a single point read (upload) of a single Measurement from the selected ECU without opening a Measurement task.

mcMeasurementWrite

Purpose

Writes a single Measurement value to the ECU.

Format

```
mcTypeStatus        mcMeasurementWrite(
                       mcTypeTaskRef ECURefNum,
                       char *MeasurementName,
                       f64 Values);
```

Input

`ECURefNum`	`ECURefNum` is the task reference which links to the selected ECU. This reference is originally returned from `mcECUSelect`.
`MeasurementName`	`MeasurementName` is the name of a Measurement channel stored in the A2L database file to which a Measurement value is to be written.
`Values`	Writes a single sample for the Measurement channel initialized in `MeasurementName`.

Output

Return Value

The return value indicates the status of the function call as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `mcStatusToString` function of the ECU M&C API to obtain a descriptive string for the return value.

Description

`mcMeasurementWrite` performs a single point write (download) of a Measurement into the selected ECU without opening a Measurement task. `mcMeasurementWrite` can only be performed if the Measurement channel is not set to read only. To query if an ECU Measurement channel can be accessed by `mcMeasurementWrite`, call `mcGetProperty` with the parameter `mcPropMeas_ReadOnly`.

mcSetProperty

Purpose

Sets a property of the driver, the database, the ECU, a Characteristic, a Measurement, or a Measurement task.

Format

```
mcTypeStatus        mcSetProperty(
                       mcTypeTaskRef RefNum,
                       cstr Name,
                       u32 PropertyID,
                       u32 SizeOfValue,
                       void *Value);
```

Input

`RefNum`	`RefNum` is any ECU M&C task reference which consists of a valid link to the opened A2L database (`DBRefNum`), a selected ECU (`ECURefNum`) or a Measurement task (`DAQRefNum`). `RefNum` has to be valid for the related `PropertyID` type.
`Name`	Name is not used and can be set to NULL. This parameter maybe used for further extensions.
`PropertyID`	Selects the property to set. For a description of each property, including its data type and `PropertyId`, refer to the *Properties* section.
`SizeOfValue`	Number of bytes allocated for the `Value` output. This size normally depends on the data type listed in the description of the property.
`Value`	Provides the property value. `PropertyId` determines the data type of the value.

Output

Return Value

The return value indicates the status of the function call as a signed 32-bit integer. Zero means the function executed successfully. A negative value specifies an error, which means the function did not perform the expected behavior. A positive value specifies a warning, which means the function performed as expected, but a condition arose that may require attention.

Use the `mcStatusToString` function of the ECU M&C API to obtain a descriptive string for the return value.

Description

ECU-Specific Properties

You cannot set an ECU property while the application is connected to the ECU. If you need to change a ECU property prior to connecting, call `mcECUSelect`, followed by `mcSetProperty`, and then `mcECUConnect`. After you connect to the ECU, you also can change a property by calling `mcECUDisconnect`, followed by `mcSetProperty`, and then `mcECUConnect` to restart the task. Table 6-5 contains a listing of ECU-specific values for `PropertyID`.

DAQ-Specific Properties

You cannot set a DAQ property while a Measurement task is running. If you need to change a property prior to starting a Measurement task call `mcDAQInitialize`, followed by `mcSetProperty`, and then `mcDAQStart`. After you start the Measurement task, you also can change a property by calling `mcDAQStop`, followed by `mcSetProperty`, and then `mcDAQStart` to restart the task. Table 6-6 contains a listing of ECU-specific values for `PropertyID`.

Properties

Table 6-5. ECU-Specific Value Types for the PropertyID Input Value

Data Type	Name	Description
u32	`mcPropECU_DTO_ID`	Sets the DTO ID (**D**ata **T**ransmission **O**bject) which is used by the ECU to respond to CCP commands and send data and status information to the CCP master.
u32	`mcPropECU_CRO_ID`	Sets the CAN identifier for the CRO ID (**C**ommand **R**eceive **O**bject), which is used to send commands and data from the host to the slave device.

Table 6-5. ECU-Specific Value Types for the PropertyID Input Value (Continued)

Data Type	Name	Description
u32	`mcPropECU_BaudRate`	Sets the Baud rate in use by the selected interface. This property applies to all tasks initialized with the NI-CAN interface. You can specify the following basic baud rates as the numeric rate: 33333, 83333, 100000, 125000, 200000, 250000, 400000, 500000, 800000, and 1000000. You can specify advanced baud rates as 8000*XXYY* hex, where *YY* is the value of Bit Timing Register 0 (BTR0), and *XX* is the value of Bit Timing Register 1 (BTR1). For more information, refer to the **Interface Properties** dialog in MAX. The value of this property is originally set within MAX, but it can be changed using `mcSetProperty`.
u32	`mcPropECU_Station Address`	Sets the station address of the slave device. CCP is based on the idea that several ECUs can share the same CAN Arbitration IDs for CCP communication. To avoid communication conflicts, CCP defines a Station Address that has to be unique for all ECUs sharing the same CAN Arbitration IDs. Unless an ECU has been addressed by its Station Address, the ECU must not react to CCP commands sent by the CCP master.
u32	`mcPropECU_ByteOrder`	Sets the Byte Order of the CCP slave device. 0—**MSB_LAST** The CCP Slave device uses the MSB_LAST (Intel) byte ordering. 1—**MSB_FIRST** The CCP Slave device uses the MSB_FIRST (Motorola) byte ordering.
str	`mcPropECU_SeedKey_Cal`	Sets the file name of the SeedKey DLL used for Calibration purposes.
str	`mcPropECU_SeedKey_DAQ`	Sets the file name of the SeedKey DLL used for DAQ purposes.

Table 6-5. ECU-Specific Value Types for the PropertyID Input Value (Continued)

Data Type	Name	Description
str	`mcPropECU_SeedKey_Prog`	Sets the file name of the SeedKey DLL used for programming purposes.
str	`mcPropECU_Checksum`	Sets the file name of the Checksum DLL used for verifying the checksum.
u32	`mcPropECU_MasterID`	Sets CCP master ID information. This ID information is optional and specific to the ECU implementation. For more information about the CCP master ID information refer to the documentation for the ECU.
u8	`mcPropECU_Single_Byte_DAQ_Lists`	Sets the ECU to support single-byte or multi-byte DAQ list entries.

Table 6-6. DAQ-Specific Value Types for the PropertyID Input Value

Data Type	Name	Description
i32	`mcPropDAQ_Mode`	Sets the mode of an M&C Measurement task. 0—**DAQ List** The data is transmitted from the ECU in equidistant time intervals as defined in the A2L database. The data can be read back with `mcDAQRead` as Single point data using sample rate = 0, or as waveform using a sample rate > 0. Input channel data is received from the DAQ messages. Use `mcDAQRead` to obtain input samples as single-point, array, or waveform. 1—**Polling** In this mode the data from the Measurement task is uploaded from the ECU whenever `mcDAQRead` is called.
u32	`mcPropDAQ_DTO_ID`	Sets the DTO ID (**D**ata **T**ransmission **O**bject) which is used by the ECU to respond to send data from the DAQ lists to the CCP master.

Table 6-6. DAQ-Specific Value Types for the PropertyID Input Value (Continued)

Data Type	Name	Description
abc	`mcPropDAQ_EventChannelName`	Sets the selected event channel name to which the Measurement task is assigned.
u16	`mcPropDAQ_Prescaler`	Sets the Prescaler, which reduces the desired transmission frequency of the associated DAQ list.

mcStatusToString

Purpose

Converts a status code into a descriptive string.

Format

```
mcTypeStatus        mcStatusToString(
                       mcTypeTaskRef Status,
                       u32 SizeofString,
                       str ErrorString);
```

Input

`Status`	Nonzero status code returned from an ECU M&C function.
`SizeofString`	`SizeofString` buffer (in bytes).

Output

`ErrorString`	ASCII string that describes `Status`.

Description

When the status code returned from an ECU M&C function is nonzero, an error or warning is indicated. This function is used to obtain a description of the error/warning for debugging purposes.

The return code is passed into the `Status` parameter. The `SizeofString` parameter indicates the number of bytes available in the string for the description. The description will be truncated to size `SizeofString` if needed, but a size of 300 characters is large enough to hold any description. The text returned in `ErrorString` is null-terminated, so it can be used with ANSI C functions such as `printf`. For applications written in C or C++, each ECU M&C function returns a status code as a signed 32-bit integer. The following table summarizes the ECU M&C use of this status.

Table 6-7. Description of Return Codes

Status Code	Definition
Negative	Error—Function did not perform expected behavior.
Positive	Warning—Function performed as expected, but a condition arose that may require attention.
Zero	Success—Function completed successfully.

The application code should check the status returned from every ECU M&C function. If an error is detected, you should close all ECU M&C handles and exit the application. If a warning is detected, you can display a message for debugging purposes or simply ignore the warning.

The following piece of code shows an example of handling ECU M&C status during application debugging.

```
status= ncDatabaseOpen ("TestDataBase.A2L", &MyDbHandle);

PrintStat (status, "mcOpenDatabase");
```

where the function `PrintStat` has been defined at the top of the program as:

```
void PrintStat(mcTypeStatus status, char *source)

{

   char statusString[300];

   if(status !=0)

   {

      mcStatusToString(status, sizeof(statusString), statusString);

      printf("\n%s\nSource = %s\n", statusString, source);

      if (status < 0)

      {

            mcDatabaseClose(MyDbHandle);

            exit(1);

      }

   }

}
```

In some situations, you may want to check for specific errors in the code. For example, when `mcCharacteristicRead` times out, you may want to continue communication, rather than exit the application. To check for specific errors, use the constants defined in `niemc.h`. These constants have the same names as described in this manual. For example, to check for a function timeout, use:

```
if (status == mcErrorTimeout)

   ...
```

A

Summary of the CCP Standard

Controller Area Network (CAN)

Bosch developed the Controller Area Network (CAN) in the mid-1980s. Using CAN, devices (controllers, sensors, and actuators) are connected on a common serial bus. This network of devices can be thought of as a scaled-down, real-time, low-cost version of the networks used to connect personal computers. Any device on a CAN network can communicate with any other device using a common pair of wires.

As CAN implementations increased in the automotive industry, CAN was standardized internationally as ISO 11898. CAN chips were created by major semiconductor manufacturers such as Intel, Motorola, and Philips. With these developments, manufacturers of industrial automation equipment began to consider CAN for use in industrial applications. Comparison of the requirements for automotive and industrial device networks showed numerous similarities, including the transition away from dedicated signal lines, low cost, resistance to harsh environments, and high real-time capabilities.

CAN Calibration Protocol (CCP)

The amount of electronics introduced into the automobile has increased significantly. This trend is expected to continue as automobile manufacturers initiate further advances in safety, reliability and comfort. The introduction of advanced control systems—combining multiple sensors, actuators and electronic control units—has begun to place extensive demands on the existing Controller Area Network (CAN) communication bus. To enable the new generation of automotive electronics, new and highly sophisticated software, calibration, measurement, and diagnostic equipment must be used. At this time almost no standards exist in the area of software interfaces for such devices. Each company has its proprietary systems and interfaces to support the development of these high-end configurations.

The CAN Calibration Protocol was originally developed and introduced by Ingenieurbüro Helmut Kleinknecht, a manufacturer of calibration systems, and is used in various application areas in the automotive industry. Afterwards CCP was taken over by the ASAP working group and enhanced with optional functions and is now maintained by the ASAM organization.

Scope of CCP

The CAN Calibration Protocol is a CAN-based master-slave protocol for calibration and data acquisition using the CAN 2.0B standard (11-bit and 29-bit identifiers), which includes 2.0A (11-bit identifier). A single master device (host) can be connected to one or more slave devices. Before a slave device may accept commands from the host, the host must establish a logical point-to-point connection to the slave device. After this connection has been established, the slave device has to acknowledge each command received from the host within a specific time.

CCP offers continuous or event driven data acquisition from the controllers, as well as memory transfers to and control functions in the controllers for calibration purposes.

With these functions, CCP may be used in:

- The development of electronic control units (ECU)
- Systems for functional and environmental tests of an ECU
- Test systems and test stands for controlled devices (combustion engines, gearboxes, suspension systems, climate-control systems, body systems, anti-locking systems)
- On-board test and measurement systems of pre-series vehicles
- Any non-automotive application of CAN-based distributed electronic control systems

CCP defines two function sets—one for control/memory transfer, and one for data acquisitions that are independent of each other and may run asynchronously. The control commands are used to carry out functions in the slave device, and may use the slave to perform tasks on other devices. The data acquisition commands are used for continuous data acquisition from a slave device. The devices continuously transmit internal data according to a list that has been configured by the host. Data acquisition is initiated by the host, then executed by the slave device, and may be based on a fixed sampling rate or be event-driven.

CCP Protocol Definition

Two communication objects are defined by CCP to handle the communication between host and slave devices—The **C**ommand**R**eceive**O**bject (**CRO**), which is used to send commands and data from the host to the slave device; and the **D**ata**T**ransmission**O**bject (**DTO**), which is used to transmit handshake messages, data and status information from the slave device to the host. Each of these message objects is assigned a unique CAN ID. Messages that are returned from the slave as a message to a command are called **C**ommand**R**eturn**M**essages (**CRM**).

A Command Receive Object is a CAN message consisting of eight bytes. The first byte of a CRO is the **command** code, followed by the **command counter** byte. The command counter is generated for reference by the host to make sure that the CRM returned by a slave device corresponds to the correct host command. The rest of the message builds the parameter and data fields. The structure is as follows:

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
CMD	CTR	Parameter and Data Field					

A DataTransmissionObject has a **P**acket**ID** (**PID**) as the first byte. This PID determines how the rest of the message is interpreted. CCP differentiates between three types of DTOs:

PID	**Type**
0x00—0xFD	Data Acquisition Message
0xFE	Event Message
0xFF	Command Return Message

Command Return Messages and Event Messages have the following structure:

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
PID	ERR	Parameter and Data Field					

In the case of an Event Message, the **Counter** field does not contain valid data and has to be ignored by the host. For Command Return Messages the

Counter field must have the same value as the counter field of the corresponding CRO. The error field contains information about the error state. The **parameter** and **data** fields contain the data returned from the slave device to the host. Command Return Messages and Event Messages consist of eight bytes.

Data **A**cquisition **M**essages (DAQ Messages or **DAM**s) have a PID in the first byte, and the rest of the message contains data. DAMs may be shorter than eight bytes:

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
PID	Parameter and Data Field						

Since the PIDs 0x00—0xFD are reserved for Data Acquisition Messages, a CCP slave device can send up to 253 different DAMs. Each DAQ message can transfer up to seven bytes of data. The number of DAQ Messages supported by a slave device depends on the device itself.

Data acquisition is performed through a CCP slave device by reading data from a device's memory and copying it into the data field of a DAQ message. So the CCP slave device keeps a list of entries for each DAM. These lists are called **O**bject**D**efinition**T**ables (**ODT**s). Each ODT entry holds information about the memory address where data is stored inside the device and the size of the data to be sent. The data of the first ODT entry will be placed in the first byte of the data field of the DAQ message. The data of the next entry will be placed at the first free byte of the DAQ message, and so on.

B Technical Support and Professional Services

Visit the following sections of the National Instruments Web site at `ni.com` for technical support and professional services:

- **Support**—Online technical support resources at `ni.com/support` include the following:
  - **Self-Help Resources**—For answers and solutions, visit the award-winning National Instruments Web site for software drivers and updates, a searchable KnowledgeBase, product manuals, step-by-step troubleshooting wizards, thousands of example programs, tutorials, application notes, instrument drivers, and so on.
  - **Free Technical Support**—All registered users receive free Basic Service, which includes access to hundreds of Application Engineers worldwide in the NI Developer Exchange at `ni.com/exchange`. National Instruments Application Engineers make sure every question receives an answer.

    For information about other technical support options in your area, visit `ni.com/services` or contact your local office at `ni.com/contact`.
- **Training and Certification**—Visit `ni.com/training` for self-paced training, eLearning virtual classrooms, interactive CDs, and Certification program information. You also can register for instructor-led, hands-on courses at locations around the world.
- **System Integration**—If you have time constraints, limited in-house technical resources, or other project challenges, National Instruments Alliance Partner members can help. To learn more, call your local NI office or visit `ni.com/alliance`.

If you searched `ni.com` and could not find the answers you need, contact your local office or NI corporate headquarters. Phone numbers for our worldwide offices are listed at the front of this manual. You also can visit the Worldwide Offices section of `ni.com/niglobal` to access the branch office Web sites, which provide up-to-date contact information, support phone numbers, email addresses, and current events.

Glossary

Symbol	Prefix	Value
m	milli	10^{-3}
k	kilo	10^{3}
M	mega	10^{6}

Numbers/Symbols

2MC (`*.A2L`) database file	*see* ASAM MCD 2MC.

A

A2L file	ECU device database file in ASAM MCD 2MC format.
address extension	An additional parameter to the address that may be used to switch between data of several memory banks.
API	Application Program Interface—A set of routines, protocols, and tools for building software applications.
arbitration ID	An 11- or 29-bit ID transmitted as the first field of a CAN frame. The arbitration ID determines the priority of the frame, and is normally used to identify the data transmitted in the frame.
ASAM	Association of Standardization of Automation and Measurement Systems.
ASAM MCD 2MC	ASAM MCD 2MC is a file interface standardized by ASAM which describes the internal ECU data, interfaces, and communication protocols. It contains all information about relevant data objects in the ECU like Characteristic variables (parameters, characteristic curves, and maps), real/virtual measurement variables, and variant dependencies. For each of these objects information is needed, such as storage address, record layout, data type, and conversion rules to convert the data into their physical units.

B

baudrate	A user-defined property which provides the baud rate at which communication will occur. For more information, refer to the **Interface Properties** dialog in MAX, or the *NI-CAN Hardware and Software Manual*. The baud rate is originally set within MAX.
byte order	The *byte order* refers to which bytes are most significant in multi-byte data types. The term describes the order in which a sequence of bytes is stored in computer memory.

C

calibration data page	A portion of the ECU memory containing data that controls the behavior of the ECU.
CAN	Controller Area Network. The Controller Area Network (CAN) is a joint development of Robert Bosch GmbH and Intel Corporation. CAN is used in many high-end automotive control systems, like engine management, as well as in industrial control systems. Controller chips for CAN are available from various semiconductor manufacturers.
CCP	CAN Calibration Protocol.
CCP master	The CCP master device (host) is a calibration/monitoring tool for initiating data transfers on the CAN by sending commands to slave devices.
CCP slave	Typically an ECU which communicates through CCP with the CCP master.
Characteristic	A Characteristic is a memory area within the ECU which defines the behavior of a control subsystem. Calibration is a process to optimize the Characteristic. A Characteristic can be represented by a single value (parameter), a one-dimensional array of values (curve), or a two-dimensional array of values (map).
Checksum DLL	A Dynamic Link Library which implements a function to calculate a checksum over a given data block.
Command Receive Object (CRO)	A Command Receive Object (CRO) is sent from the CCP master device to one of the slave devices. The slave device answers with a Data Transmission Object (DTO) containing a Command Return Message (CRM).

Controller Area Network	*see* CAN.
CRM	Command Return Message—A CCP communication object used to send commands and data from a host device to a slave device. The CRO is 8 bytes wide, consisting of a Command byte, a Command Counter byte, and a 6-byte parameter/data field.
CRO	CommandReceiveObject—A CCP communication object used to send commands and data from a host device to a slave device. The CRO is 8 bytes wide, consisting of a Command byte, a Command Counter byte, and a 6-byte parameter/data field.
CRO ID	CAN identifier of the Command Receive Object (CRO)

D

DAQ	Data Acquisition.
DAQ channel	A single DAQ Measurement entry in a DAQ list.
DAQ list	A list of DAQ channels that is transmitted by the ECU.
DAQ mode	Data acquisition mode.
database task	A task reference handle to the selected ASAM MCD 2MC database file.
Data Transfer Object	A message sent from the slave device to the master device (Command Return Message, Event Message, or Data Acquisition Message).
DLL	Dynamic Link Library.
DTO	*see* Data Transfer Object.
DTO ID	CAN identifier of the DTO.

E

ECU	Electronic Control Unit—An electronic device with a central processing unit performing programmed functions with its peripheral circuitry.

ECU M&C Channel API	The API of the ECU M&C Toolkit that you use to read and write channels. A Characteristic or Measurement channel consists of one more floating-point values in physical units (such as Volts, rpm, km/h, °C, and so on) that is converted to/from a raw value in measurement hardware. The ECU M&C API Read and Write functions provide access to Characteristic or Measurement channels. When a CAN message is received, ECU M&C Toolkit converts raw fields in the message into physical units, which you then obtain using the ECU M&C API Read function. When you call a ECU M&C API Write function, you provide floating-point values in physical units, which ECU M&C Toolkit converts into raw fields and transmits as a CAN message based on the CCP protocol.
ECU reference	Reference handle to a selected ECU.
ECU task	*see* ECU reference.
Event Channel	Specifies the generic signal source that effectively determines the data transmission timing.
Extended arbitration ID	A 29-bit arbitration ID. Frames that use extended IDs are often referred to as CAN 2.0 Part B (the specification which defines them).

M

Master ID	A 6-byte string identifying the CCP master device.
Measurement	*see* DAQ.
Measurement task	A collection of DAQ channels that you can read or write.
Memory Transfer Address	Address pointer in the ECU that holds the source/target address for data sent or received via CCP. The address extension depends on the slave controller's organization and may identify a switchable memory bank or a memory segment.
MTA	*see* Memory Transfer Address.

O

ODT	Object Descriptor Table—A list of elements (variables) used for organization of data acquisition (DAQ).

P

PID	**P**acket**ID**—The first byte of a DTO corresponding to the ODT to which the DTO is assigned. The values for DAQ list PIDs range from 0x00–0xFD. The PIDs 0xFE and 0xFF are reserved for Event Messages and Command Return Messages.
Prescaler	A factor defined to allow reduction of the desired transmission rate. The prescaler is applied to the Event Channel. The prescaler value factor must be greater than or equal to 1.

S

SeedKey DLL	A Dynamic Link Library that implements a function to calculate a key to a given seed to unlock access to ECU resources.
slave device identifier	An ECU-specific array of bytes used by the master device to identify the ECU.
Station Address	A property which specifies an address to generate a logical point-to-point connection with a selected slave station for the master-slave command protocol. One ECU may support several station addresses.

T

task reference	An identifier returned as an output parameter of Database, ECU or Measurement initialization functions.

Index