Interface Programming between CANape and MATLAB
Version 7.5
2018-06-19
Application Note AN-IMC-1-004

Table of Contents
1 Overview .........................................................................................................................4
  1.1 General Usage of the API .........................................................................................4
2 General Preparation ........................................................................................................4
  2.1 Installation ..............................................................................................................4
  2.2 CANape....................................................................................................................5
  2.3 MATLAB ..................................................................................................................5
3 Using the CANape API in MATLAB ..............................................................................5
  3.1 Measurement Via CANape API ...............................................................................5
4 CANape API Function Overview ....................................................................................6
  4.1 General ....................................................................................................................6
  4.2 ml_CANapeInit .......................................................................................................6
  4.3 ml_CANapeInitEx .....................................................................................................8
  4.4 ml_CANapeCreateModule .......................................................................................8
  4.5 ml_CANapeSetTCPOptions .....................................................................................9
  4.6 ml_CANapeCreateModuleOnOffline .......................................................................10
  4.7 ml_CANapeCreateModuleOnOfflineCache .............................................................10
  4.8 ml_CANapeOpenAsap2File ....................................................................................10
  4.9 ml_CANapeGetEcuTasks ......................................................................................11
  4.10 ml_CANapeGetEcuTasksInfo ..............................................................................11
  4.11 ml_CANapeGetEcuTasksInfoName .....................................................................12
  4.12 ml_CANapeGetEcuTasksAllInfos .......................................................................12
  4.13 ml_CANapeAddMeasurementModule ..................................................................13
  4.14 ml_CANapeStartMeasurement ..........................................................................13
  4.15 ml_CANapeGetFifoDataOfTask ..........................................................................13
  4.16 ml_CANapeGetFifoBlockDataOfTask .................................................................14
  4.17 ml_CANapeCheckFifoBlock ..............................................................................14
  4.18 ml_CANapeGetCurrentValueMatrix ....................................................................15
  4.19 ml_CANapeGetCurrentValues ...........................................................................15
  4.20 ml_CANapeStopMeasurement ...........................................................................16
  4.21 ml_CANapeResetDataAcquisitionChnls ..............................................................16
  4.22 ml_CANapeSetDataAcquisitionChnls ................................................................16
  4.23 ml_CANapeReadCalibrationObject ....................................................................17
  4.24 ml_CANapeWriteCalibrationObject.....................................................................19
  4.25 ml_CANapeReadObjectByAddress ..................................................................21
  4.26 ml_CANapeWriteObjectByAddress ..................................................................22
  4.27 ml_CANapeExit .................................................................................................22
  4.28 ml_CANapeQuitNonModalMode .......................................................................22
  4.29 ml_CANapeMatlabConversion (deprecated) .....................................................23
### 4 Interface Programming between CANape and MATLAB

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ml_CANapeMDFConvert</td>
<td>23</td>
</tr>
<tr>
<td>ml_CANapeMDFConverterCount</td>
<td>24</td>
</tr>
<tr>
<td>ml_CANapeMDFConverterInfo</td>
<td>24</td>
</tr>
<tr>
<td>ml_CANapeSetDebugInfo</td>
<td>24</td>
</tr>
<tr>
<td>ml_CANapeGetLastError</td>
<td>25</td>
</tr>
<tr>
<td>ml_CANapeGetLastErrorText</td>
<td>25</td>
</tr>
<tr>
<td>ml_CANapeGetApplicationVersion</td>
<td>25</td>
</tr>
<tr>
<td>ml_CANapeSwitchECUOnOffLine</td>
<td>26</td>
</tr>
<tr>
<td>ml_CANapeExecuteScript</td>
<td>26</td>
</tr>
<tr>
<td>ml_CANapeExecuteScriptEx</td>
<td>27</td>
</tr>
<tr>
<td>ml_CANapeGetLastError</td>
<td>27</td>
</tr>
<tr>
<td>ml_CANapeStopScript</td>
<td>28</td>
</tr>
<tr>
<td>ml_CANapeGetScriptState</td>
<td>28</td>
</tr>
<tr>
<td>ml_CANapeGetScriptResultValue</td>
<td>29</td>
</tr>
<tr>
<td>ml_CANapeGetScriptResultString</td>
<td>29</td>
</tr>
<tr>
<td>ml_CANapeReleaseScript</td>
<td>30</td>
</tr>
<tr>
<td>ml_CANapeCheckOverrun</td>
<td>30</td>
</tr>
<tr>
<td>ml_CANapeIsECUOnline</td>
<td>31</td>
</tr>
<tr>
<td>ml_CANapeUseNAN</td>
<td>31</td>
</tr>
<tr>
<td>ml_CANapeIsNANUsed</td>
<td>31</td>
</tr>
<tr>
<td>ml_CANapeGetDBObjectListNames</td>
<td>32</td>
</tr>
<tr>
<td>ml_CANapeGetMeasureListNames</td>
<td>32</td>
</tr>
<tr>
<td>ml_CANapeGetDevices</td>
<td>33</td>
</tr>
<tr>
<td>ml_CANapeGetCalibrationListNames</td>
<td>33</td>
</tr>
<tr>
<td>ml_CANapeDefineRecorder</td>
<td>33</td>
</tr>
<tr>
<td>ml_CANapeEnableRecorder</td>
<td>34</td>
</tr>
<tr>
<td>ml_CANapeGetRecorderByName</td>
<td>34</td>
</tr>
<tr>
<td>ml_CANapeGetRecorderCount</td>
<td>34</td>
</tr>
<tr>
<td>ml_CANapeGetRecorderDataReduction</td>
<td>35</td>
</tr>
<tr>
<td>ml_CANapeGetRecorderMdfFileName</td>
<td>35</td>
</tr>
<tr>
<td>ml_CANapeGetRecorderName</td>
<td>36</td>
</tr>
<tr>
<td>ml_CANapeGetRecorderState</td>
<td>36</td>
</tr>
<tr>
<td>ml_CANapeGetSelectedRecorder</td>
<td>36</td>
</tr>
<tr>
<td>ml_CANapeGetRecorderEnabled</td>
<td>37</td>
</tr>
<tr>
<td>ml_CANapePauseRecorder</td>
<td>37</td>
</tr>
<tr>
<td>ml_CANapeRemoveRecorder</td>
<td>37</td>
</tr>
<tr>
<td>ml_CANapeSelectRecorder</td>
<td>38</td>
</tr>
<tr>
<td>ml_CANapeSetRecorderDataReduction</td>
<td>38</td>
</tr>
<tr>
<td>ml_CANapeSetRecorderMdfFileName</td>
<td>38</td>
</tr>
<tr>
<td>ml_CANapeSetRecorderMdfFileName</td>
<td>38</td>
</tr>
</tbody>
</table>

### 5 Diagnostic

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ml_CANapeDiagCreateSymbolicRequest</td>
<td>39</td>
</tr>
<tr>
<td>ml_CANapeDiagExecute</td>
<td>39</td>
</tr>
<tr>
<td>ml_CANapeDiagGetServiceState</td>
<td>40</td>
</tr>
<tr>
<td>ml_CANapeDiagReleaseService</td>
<td>40</td>
</tr>
<tr>
<td>ml_CANapeDiagGetResponseCount</td>
<td>40</td>
</tr>
<tr>
<td>ml_CANapeDiagIsPositiveResponse</td>
<td>41</td>
</tr>
<tr>
<td>ml_CANapeDiagGetResponseCode</td>
<td>41</td>
</tr>
<tr>
<td>ml_CANapeDiagGetStringParameter</td>
<td>42</td>
</tr>
<tr>
<td>ml_CANapeDiagGetStringResponseParameter</td>
<td>42</td>
</tr>
<tr>
<td>ml_CANapeDiagSetRawParameter</td>
<td>42</td>
</tr>
<tr>
<td>ml_CANapeDiagGetRawResponseParameter</td>
<td>43</td>
</tr>
<tr>
<td>ml_CANapeDiagIsComplexResponseParameter</td>
<td>43</td>
</tr>
<tr>
<td>ml_CANapeDiagIsComplexResponseParameter</td>
<td>43</td>
</tr>
</tbody>
</table>
5.13 ml_CANapeDiagGetComplexIterationCount .................................................. 44
5.14 ml_CANapeDiagGetComplexRawResponseParameter ................................. 44
5.15 ml_CANapeDiagGetComplexStringResponseParameter ............................. 45
5.16 ml_CANapeDiagExecuteJob ................................................................. 45

6 Measurement example .................................................................................. 46
  6.1 CCPSim ................................................................................................. 46
  6.2 CANape ............................................................................................... 46
  6.3 MATLAB .............................................................................................. 46

7 Calibration example ...................................................................................... 48
  7.1 CCPSim ................................................................................................. 48
  7.2 CANape ............................................................................................... 48

8 MATLAB ........................................................................................................ 48
  8.1 Read Calibration .................................................................................... 48
  8.2 Write Calibration ................................................................................... 51

9 Database Example .......................................................................................... 52
  9.1 Read all Objects .................................................................................... 52

10 ECU Example ................................................................................................ 54

11 Contacts ........................................................................................................ 56
1 Overview
The purpose of this application note is to show how MATLAB/Simulink can be used for accessing an ECU for measurement and calibration. This application note explains how to use the interface between MATLAB and CANape and how to program with the CANape API.
This document should be used as a framework which enables the reader to write his own application.

1.1 General Usage of the API
CANape contains an interface, called CANape API, which provides services to other programs for accessing an ECU at runtime for measurement and calibration. Its main purpose is to use the functionality of CANape from another program like MATLAB. The CANape API enables the user to access ECUs through CANape. Since version 2.0, it supports the simultaneous usage of more than one ECU.

![Diagram of MATLAB/Simulink, CANape, and ECU connected through CANape API](image)

**Figure 1:** Normal operation between MATLAB/Simulink, CANape and the ECU

2 General Preparation
This chapter describes the installation/setup process.

2.1 Installation
The CANape/MATLAB Interface requires at least the following product versions:

> CANape 13 or later
> MATLAB 7 Release 14.0 SP3
> WIN 2000/WINXP

After execution of the SETUP, the CANape/MATLAB Interface files are located in the following directory:

```
$(MATLABROOT)\API-DLL-Interface
```
Note
After the installation you must add the following path to your MATLAB environment:
$(MATLABROOT)\API-DLL-Interface

The sample files are located in the following directory:
$(MATLABROOT)\API-DLL-Interface

2.2 CANape
If you want to use CANape in the "non-modal" mode, you have to start it with the option --u. This mode allows the simultaneous usage of CANape as the server and the client application (MATLAB).

2.3 MATLAB
After the start of MATLAB, the drop-down menu Current Directory must be set to the installation path (e.g. C:\MATLAB6p1\API-DLL-Interface).

Figure 2: MATLAB/Simulink workspace

3 Using the CANape API in MATLAB

3.1 Measurement Via CANape API
The first step is to open a session and to initialize CANape. The second step is to load an A2L file and to configure the measurement. Finally, the measurement must be started. The configured measurement data is transmitted to the application. It is the job of the application to process this data. The measurement can be stopped by the CANape API:
4 CANape API Function Overview

4.1 General

The generated MATLAB/CANape Interface DLL contains all necessary functions of the CANape API. These functions are used to control the behavior of CANape. The interface between CANape and MATLAB is referenced in the MEX functions. These MEX functions are responsible for the function calls to the CANape API interface.

4.2 ml_CANapeInit

Function:

double ml_CANapeInit(char *Projectpath, double Fifosize, double Samplesize, double TimeOut, double Modalmode);

Parameters:

- char *Projectpath
  Sets the path of the working folder
- double Fifosize
  Sets the FIFO buffer size for the measurement
- double Samplesize
  Sets the sample buffer size for the measurement
> double TimeOut
    Sets the timeout in ms
> double Modalmode
    Sets the start mode of CANape
    Value:
    Modal mode = 1  =>  non-modal (MATLAB Client and CANape)
    Modal mode = 0  =>  modal (only MATLAB Client)

**Description:**
The `ml_CANapeInit` function is used for initializing and starting the CANape runtime environment. The first parameter sets the project path where the A2L file and the other configuration files are located. The FIFO and sample size buffer are used for the measurement buffer. The whole memory is calculated by the following formula.

\[
\text{Size} = \text{FIFO} \times \text{sample}
\]

**Note**
For more information, read the documentation of the CANape API.

**Example:**
```
ml_CANapeInit('D:\Programme\CANape\CCFsim', 4000, 200, 1);
```

**Return:**
double  OK = 1, NOK = 0, fatal error = -1

If it is necessary to use a specific driver, the user must use the function `ml_CANapeCreateModule`. 
4.3 ml_CANapeInitEx

Function:

```c
double ml_CANapeInitEx(char *Projectpath, double Fifosize, double Samplesize, double TimeOut,
                        double ClearDeviceList, double Modalmode);
```

Parameters:

- `char *Projectpath`
  Sets the path of the working folder
- `double Fifosize`
  Sets the FIFO buffer size for the measurement
- `double Samplesize`
  Sets the sample buffer size for the measurement
- `double TimeOut`
  Sets the timeout in ms
- `double ClearDeviceList`
  - `ClearDeviceList = 1` all devices are cleared
  - `ClearDeviceList = 0` all devices are added, which are described in the CANape.ini
- `double Modalmode`
  Sets the start mode of CANape
  - `Modal mode = 1` → non-modal (MATLAB Client and CANape)
  - `Modal mode = 0` → modal (only MATLAB Client)

Description:

The ml_CANapeInitEx function is an enlargement of the ml_CANapeInit function. This function has an additional parameter. This parameter is responsible to clear the device list when CANape started.

4.4 ml_CANapeCreateModule

Function:

```c
double ml_CANapeCreateModule(char *moduleName, char *databaseFilename, double Driver, double canChnl);
```

Parameters:

- `char *moduleName`
  Name of the module to create
- `char *databaseFilename`
  Sets the path and the name of the a2l or db file
  Note: The parameter must include the path name and the file name.
- `double Driver`
  Set driver type
  - `ASAP3_DRIVER_CCP = 1`,
  - `ASAP3_DRIVER_XCP = 2`,
  - `ASAP3_DRIVER_CAN = 20`,
  - `ASAP3_DRIVER_HEXEDIT = 40`,
  - `ASAP3_DRIVER_ANALOG = 50`,
  - `ASAP3_DRIVER_CANOPEN = 60`,
  - `ASAP3_DRIVER_CANDELA = 70`,
  - `ASAP3_DRIVER_ENVIRONMENT = 80`,
  - `ASAP3_DRIVER_LIN = 90`,
Interface Programming between CANape and MATLAB

\[
\begin{align*}
\text{ASAP3\_DRIVER\_FLX} & = 100, \\
\text{ASAP3\_DRIVER\_ETH} & = 150,
\end{align*}
\]

> double canChnl
Set the logical communication channel to be used from CANape.
Value:
- channel = 1 - 4 \(\rightarrow\) can channel 1-4
- channel = 5 \(\rightarrow\) TCP/IP
- channel = 261 \(\rightarrow\) FlexRay

**Note**
If the XCP driver is used, the channel for TCP/IP or UDP has to be set to the value TCP = 255 or UDP = 256 and for FlexRay = 261.

**Description:**
The `ml\_CANapeCreateModule` function is used for creating a new module/device and for loading an ASAP2 file or a DB file. The function configures the logical communication channel which will be used (like CCP: 1-4 = CAN1-CAN4) and the driver type like KWPOnCAN, or CCP. The return value is a two-dimensional array which contains the module handle of the device and if the function is successful.

**Example:**
\[
[x] = \text{ml\_CANapeCreateModule('CCPSim', 'D:\Programme\CANape\CCPsim\CCPSIM.a2l', 1,1));
\]

Return Value = \(x(:,1)\) // OK = 1, NOK = 0 and fatal error = -1;
Module Handle = \(x(:,2)\) // Module handle for this device

**Return:**
- double[0] OK = 1, NOK = 0, fatal error = -1
- double[1] Module handle of the device

**4.5 ml\_CANapeSetTCPOptions**

**Function:**
double ml\_CANapeSetTCPOptions(char ‘ipNumber, short port);

**Description:**

**Note**
This function must be called before any of the `ml\_CANapeInit` calls. It will prepare the ASAP3 module for the connection of the ASAP3 CANape Server.

**Parameters:**
- char ‘ipNumber
  Set the IP number
- short port
  Value for the used port

**Example:**
\[
[x] = \text{ml\_CANapeSetTCPOptions('172.0.0.1',333);
\]
Return:
double[0] OK = 1, NOK = 0

4.6 ml_CANapeCreateModuleOnOffline

Function:
double ml_CANapeCreateModuleOnOffline(char *moduleName, char *databaseFilename, double Driver, double canChn, double OnOfflinemode)

Description:
This function is used in the same way as ml_CANapeCreateModule. The difference is only an additional parameter which is responsible for creating a device in the offline or online mode.

Example:
[x] = ml_CANapeCreateModuleOnOffline('KWPsim',
    'D:\CANape\KWPsim\Example-2.0.5.cdd',70,1,1);

Return:
x(:,1) // OK = 1, NOK = 0 and fatal error = -1;
x(:,2) // Module handle for this device

4.7 ml_CANapeCreateModuleOnOfflineCache

Function:
double ml_CANapeCreateModuleOnOfflineCache(char *moduleName, char *databaseFilename, double Driver, double canChn, double OnOfflinemode, double EnableCache)

Description:
This function is used in the same way as ml_CANapeCreateModuleOnOffline. The difference is only the parameter “EnableCache” which enables or disables the cache.

Example:
[x] = ml_CANapeCreateModuleOnOfflineCache('KWPsim',
    'D:\CANape\KWPsim\Example-2.0.5.cdd',70,1,1,1);

Return:
x(:,1) // OK = 1, NOK = 0 and fatal error = -1;
x(:,2) // Module handle for this device

4.8 ml_CANapeOpenAsap2File

Function:
double ml_CANapeOpenAsap2File(char *A2LFilename, short CANchannel);

Note
This function can only be used with a CCP Driver!
Parameters:

> char *A2LFilename
Sets the filename of ASAP 2 File

Note
If the ASAP2 file is not located in the project path, the parameter must include the path name and the file name.

> short CANchannel
Sets the channel of the CAN bus to be used

Value:
CANchannel = 1 → CAN bus channel 1
CANchannel = 2 → CAN bus channel 2

Description:
The ml_CANapeOpenAsap2File function is used for loading the ASAP2 file and for creating a new module/device. Additionally the function sets the logical communication channel which will be used by the CCP Driver (like 1-4 = CAN1-CAN4).

Example:
[x] = ml_CANapeOpenAsap2File('D:\Programme\CANape\CCPsim\CCPSIM.a2l',1);

ReturnValue = x(:,1) // OK = 1, NOK = 0 and fatal error = -1;
ModuleHandle = x(:,2) // Module handle for this device

Return:
First return value:
double OK = 1, NOK = 0, fatal error = -1

Second return value:
double module handle of the device

4.9 ml_CANapeGetEcuTasks

Function:
double ml_CANapeGetEcuTasks(int ModuleHandle);

Parameters:
> int ModuleHandle
Identification handle of the device

Description:
The ml_CANapeGetEcuTasks determines the number of available data acquisition tasks and their accessory properties, like task id, task cycle and the description.

Example:
CountOfTasks = ml_CANapeGetEcuTasks(ModuleHandle);

Return:
The double return value is the count of tasks or 0.

4.10 ml_CANapeGetEcuTasksInfo

3 dim. array of *double ml_CANapeGetEcuTasksInfo(int ModuleHandle, double TaskNumber)
Parameters:
> int ModuleHandle
   Identification handle of the device
> double TasksNumber
   Set the number of the data acquisition tasks

Description:
The ml_CANapeGetEcuTasksInfo(...) determines the properties of available data acquisition tasks like task id and task cycle.

Example:
[t] = ml_CANapeGetEcuTasksInfo(ModuleHandle, TaskNumber);

Return:
The first return value t(1) is OK = 1, NOK = 0 or fatal error = -1.
The second return value t(2) is the task id.
The third return value t(3) describes the task cycle time.

4.11 ml_CANapeGetEcuTasksInfoName

Function:
char *ml_CANapeGetEcuTasksInfoName(int ModuleHandle, double TasksNumber);

Parameters:
> int ModuleHandle
   Identification handle of the device
> double TasksNumber
   Sets the number of the data acquisition tasks

Description:
The ml_CANapeGetEcuTasksInfoName determines the name of the available data acquisition task.

Example:
[t] = ml_CANapeGetEcuTasksInfoName(ModuleHandle, TaskNumber);

Return:
The return value is a string which contains the name of the task.

4.12 ml_CANapeGetEcuTasksAllInfos

Function:
char array ml_CANapeGetEcuTasksAllInfos(int ModuleHandle)

Parameters:
> int ModuleHandle
   Identification handle of the device

Description:
This function returns the names and settings of all ECU tasks as a list.

Example:
[ECU_Settings] = ml_CANapeGetEcuTasksAllInfos(ModuleHandle);

Return:
The return value contains a list of all ECU tasks and theirs properties (e.g. Chapter 9).
4.13 ml_CANapeAddMeasurementModule

Function:
double ml_CANapeAddMeasurementModule(int ModuleHandle, char *Channelname, int Taskindex, bool Saveoption)

Parameters:
- int ModuleHandle
  Identification handle of the device
- char *Channelname
  Name of the measurement object
- int Taskindex
  Index of the task to query. Defines the sampling rate of the ECU and the rate at which channel data samples are put into the FIFO buffer.
- bool Saveoption
  If the value is 1, the measurement of this object will be saved to disk (in the project path).

Description:
The ml_CANapeAddMeasurementModule function is used to add a measurement object to the data acquisition channel list.

Example:
double ml_CANapeAddMeasurementModule(ModuleHandle, 'channel2', 3, 0);

Return:
double OK = 1, NOK = 0

4.14 ml_CANapeStartMeasurement

Function:
double ml_CANapeStartMeasurement(void)

Parameters:
- none

Description:
The ml_CANapeStartMeasurement function initializes and starts the measurement.

Example:
ml_CANapeStartMeasurement();

Return:
double OK = 1, NOK = 0, fatal error = -1

4.15 ml_CANapeGetFifoDataOfTask

Function:
double ml_CANapeGetFifoDataOfTask(ModuleNumber, TaskIndex)

Parameters:
- int ModuleHandle
  Identification handle of the device
- short Taskindex
  Index of the task to query.
Description:
The `ml_CANapeGetFifoDataOfTask` function is used for getting the timestamp and measurement values from the according FIFO buffer.

Example:

```matlab
[Array] = ml_CANapeGetFifoDataOfTask(0, 2);
```

Return:

Array of double
double fatal error = -1

4.16 `ml_CANapeGetFifoBlockDataOfTask`

Function:

```matlab
double ml_CANapeGetFifoBlockDataOfTask(int moduleNumber, short TaskIndex)
```

Parameters:

- `int ModuleHandle`
  Identification handle of the device
- `short TaskIndex`
  Index of the task to query

Example:

```matlab
data[] = ml_CANapeGetFifoBlockDataOfTask(0, 2);
```

Return:

double: 2 dim. array of double (data[]).

Description:

This is an example with four measurement objects in the same event list.

<table>
<thead>
<tr>
<th>Channel 1</th>
<th>Channel 1</th>
<th>Channel 1</th>
<th>Channel 1</th>
<th>Channel 1</th>
<th>Channel 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Channel 2</td>
<td>Channel 2</td>
<td>Channel 2</td>
<td>Channel 2</td>
<td>Channel 2</td>
<td>Channel 2</td>
</tr>
<tr>
<td>Channel 3</td>
<td>Channel 3</td>
<td>Channel 3</td>
<td>Channel 3</td>
<td>Channel 3</td>
<td>Channel 3</td>
</tr>
<tr>
<td>Channel 4</td>
<td>Channel 4</td>
<td>Channel 4</td>
<td>Channel 4</td>
<td>Channel 4</td>
<td>Channel 4</td>
</tr>
</tbody>
</table>

Table 1: Example

double \( NOK = 0 \), fatal error = -1

4.17 `ml_CANapeCheckFifoBlock`

Function:

```matlab
double ml_CANapeCheckFifoBlock(int ModuleNumber, short TaskIndex)
```

Parameters:

- `int ModuleHandle`
  Identification handle of the device
- `short TaskId`
  Index of the task to query.

Description:

Check if data have been lost due to FIFO overrun.
Example:
[State] = ml_CANapeCheckFifoBlock(0, 2);

Return:
State [0] = Buffer Overrun or fatal error = -1
State [1] = FIFO Error

4.18 ml_CANapeGetCurrentValueMatrix

Function:
double[] ml_CANapeGetCurrentValueMatrix(int ModuleHandle, short Taskid)

Parameters:
> int ModuleHandle
   Identification handle of the device
> short Taskid
   Index of the task to query

Description:
The ml_CANapeGetCurrentValueMatrix function is used for getting current values and returns the last received values, does not use or affects the FIFO.

Example:
[Array] = ml_CANapeGetCurrentValueMatrix (ModuleHandle, Taskid);

Return:
Array of double or fatal error = -1

4.19 ml_CANapeGetCurrentValues

Function:
double[] ml_CANapeGetCurrentValues(int ModuleHandle, short Taskid)

Parameters:
> int ModuleHandle
   Identification handle of the device
> short Taskid
   Index of the task to query

Description:
The ml_CANapeGetCurrentValues function is used for getting current values and returns the last received values, does not use or affects the FIFO. Is the same like ml_CANapeGetCurrentValueMatrix.

Example:
[Array] = ml_CANapeGetCurrentValueMatrix (ModuleHandle, Taskid);

Return:
Array of double or fatal error = -1
Example for three channels in DAQ mode:
Channel_1 = Array[1]
Channel_2 = Array[2]
Channel_3 = Array[3]
TimeStamp = Array[4]

4.20 ml_CANapeStopMeasurement

Function:
double ml_CANapeStopMeasurement(void)

Parameters:
> none

Description:
The ml_CANapeStopMeasurement function stops the measurement.

Example:
ml_CANapeStopMeasurement();

Return:
double OK = 1, NOK = 0

4.21 ml_CANapeResetDataAcquisitionChnls

Function:
double ml_CANapeResetDataAcquisitionChnls(void)

Parameters:
> none

Description:
This function is responsible for clearing the complete data acquisition list.

Note
This function should be used every time before ml_CANapeAddMeasurementModule and
ml_CANapeGetEcuTasks, and after ml_CANapeInit.

Return:
double OK = 1, NOK = 0, fatal error = -1

4.22 ml_CANapeSetDataAcquisitionChnls

Function:
double ml_CANapeSetDataAcquisitionChnls(int ModuleHandle, char *ObjectName, double Taskid,
    double PollingRate, double Save2File)

Parameters:
> Module handle: Identification handle of the device
> ObjectName: Name of object to measure
> Taskid: Id of the task to query
PollingRate: If acquisition mode is polling, it specifies the polling rate
Save2File: Save this channel to measurement file (currently selected recorder will be used)

Description:
This function is responsible to add a measurement object to the data acquisition channel list.

Example:
```c
double ml_CANapeSetDataAcquisitionChnls(ModuleHandle, 'channel2', 3, 5, 0);
```

Return:
double OK = 1, NOK = 0, fatal error = -1

### 4.23 ml_CANapeReadCalibrationObject

Function:
double ml_CANapeReadCalibrationObject(int ModuleHandle, char *ObjectName, double format)

Parameters:
- int ModuleHandle: Identification handle of the device
- char *ObjectName: Name of the calibration object
- double format: Format of ECU calibration data
  - ECUINTERNAL = 0 \(\supset\) HEX: ECU-internal data format
  - PHYSICALREPRESENTATION = 1 \(\supset\) PHYS: physical value including unit

Description:
The `ml_CANapeReadCalibrationObject` function reads the data from a calibration object like scalar, string, map and curve.

Example:
```
[data] = ml_CANapeReadCalibrationObject(ModuleHandle, 'KF2', 1)
```

Return:
Returns the values of the objects or fatal error = -1

Detailed Description:
The return value contains the object of one of the following three types:

#### Scalar:
data[] = Value

Example:
```
[data] = ml_CANapeReadCalibrationObject(ModuleHandle, 'ampl', 1)
data[] = 1
```

#### Curve:
data[] = X - Axes, Values
Example:

\[\text{[data]} = \text{ml_CANapeReadCalibrationObject(ModuleHandle, 'KL3', 1)}\]

data[] =

<table>
<thead>
<tr>
<th>X-Axes</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>41</td>
<td>51</td>
</tr>
<tr>
<td>42</td>
<td>52</td>
</tr>
<tr>
<td>43</td>
<td>53</td>
</tr>
<tr>
<td>44</td>
<td>54</td>
</tr>
<tr>
<td>45</td>
<td>55</td>
</tr>
<tr>
<td>46</td>
<td>56</td>
</tr>
<tr>
<td>48</td>
<td>58</td>
</tr>
<tr>
<td>52</td>
<td>62</td>
</tr>
</tbody>
</table>

Table 2: Matrix for curves

Map:

data[] = X - Axes, Y - Axes, Values

Example:

\[\text{[data]} = \text{ml_CANapeReadCalibrationObject(ModuleHandle, 'KF2', 1)}\]
data[] =

<table>
<thead>
<tr>
<th>Axes</th>
<th>Values</th>
<th>Values</th>
<th>Values</th>
<th>Values</th>
<th>Values</th>
<th>Values</th>
<th>Values</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y/X – Axes</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>18</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>22</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>

Table 3: Matrix for maps

Figure 5: 3-D model

4.24 ml_CANapeWriteCalibrationObject

Function:
```c
double ml_CANapeWriteCalibrationObject (int ModuleHandle, char *ObjectName, double format, array data)
```

Parameters:
- int ModuleHandle
  Identification handle of the device
- char *ObjectName
  Name of the calibration object
- double format
  Format of ECU calibration data

Description:
The ml_CANapeWriteCalibrationObject function changes the values of a calibration object like scalar, map and curve.
Example:
For data:

\[ \text{ml\_CANapeWriteCalibrationObject(ModuleHandle, 'KF2', 1, data)} \]

For string:

Note
The modified string cannot be longer than the original string.

\[ \text{ml\_CANapeWriteCalibrationObject(ModuleHandle,'testString',1,'ModifyStri')} \]

Return:
Dualt OK = 1, NOK = 0, fatal error = -1

Detail Description:
The record layout contains 4 different types.

Scalar:
\[ \text{data[]} = \text{Value} \]

Example:
\[ \text{ml\_CANapeWriteCalibrationObject(ModuleHandle, 'ampl', 1, data)} \]
Format:
\[ \text{data[]} = 1 \]

Curve:
\[ \text{data[]} = \text{X - Axes, Values} \]

Example:
\[ \text{ml\_CANapeWriteCalibrationObject(ModuleHandle, 'KL3', 1, data)} \]
Format:
\[ \begin{array}{ccc}
\text{X-Axes} & \text{Values} & \text{Nothing} \\
41 & 51 & 0 \\
42 & 52 & 0 \\
43 & 53 & 0 \\
44 & 54 & 0 \\
45 & 55 & 0 \\
46 & 56 & 0 \\
48 & 58 & 0 \\
52 & 62 & 0 \\
\end{array} \]

Table 4: 2-D matrix model

Map:
\[ \text{data[]} = \text{X - Axes, Y - Axes, Values} \]

Example:
\[ \text{ml\_CANapeWriteCalibrationObject(ModuleHandle, 'KF2', 1, data)} \]
Format:
data[] =

<table>
<thead>
<tr>
<th>X-Axes</th>
<th>Y-Axes</th>
<th>Values</th>
<th>Values</th>
<th>Values</th>
<th>Values</th>
<th>Values</th>
<th>Values</th>
<th>Values</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>12</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>13</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>14</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>15</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td>16</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>8</td>
<td>18</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>12</td>
<td>22</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>

Table 5: 3-D Example for a symmetric matrix model

<table>
<thead>
<tr>
<th>X-Axes</th>
<th>Y-Axes</th>
<th>Values</th>
<th>Values</th>
<th>Values</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

Table 6: 3-D Example for an asymmetric matrix model

**Description of the data format:**
Data[] = X(1), X(2), X(3), X(4), X(5), X(6), X(7), Y(1), Y(2), Y(3), Y(4), VALUES(1), VALUES(2), VALUES(3)...

**4.25 ml_CANapeReadObjectByAddress**

**Function:**
char array ml_CANapeReadObjectByAddress(int ModuleHandle, unsigned long Address, unsigned char addrExt, unsigned long size)

**Parameters:**
> int ModuleHandle
  Identification handle of the device
> unsigned long Address
  Memory address
> unsigned char addrExt
  Address extension
> unsigned long size
  Number of bytes of data

**Example:**
[data] = ml_CANapeReadObjectByAddress(ModuleHandle, 4293188, 0, 10)

**Description:**
The function ml_CANapeReadObjectByAddress reads a defined number of bytes from the given memory address.

**Return:**
Return a byte stream.
4.26 ml_CANapeWriteObjectByAddress

Function:
char array ml_CANapeWriteObjectByAddress(int ModuleHandle, unsigned long Address,
unsigned char addrExt, double *Data)

Parameters:
> int ModuleHandle
  Identification handle of the device
> unsigned long Address
  Memory address
> unsigned char addrExt
  Address extension
> double *Data
  Array of data

Example:
[data] = ml_CANapeWriteObjectByAddress(ModuleHandle, 4293188, 0, data)

Description:
The function ml_CANapeWriteObjectByAddress writes a defined number of bytes to the given memory address.

Return:
double OK = 1, NOK = 0

4.27 ml_CanapeExit

Function:
double ml_CanapeExit(void)

Parameters:
> none

Description:
The ml_CanapeExit function terminates the connection to CANape and closes CANape.

Example:
ml_CanapeExit();

Return:
double OK = 1, NOK = 0

4.28 ml_CANapeQuitNonModalMode

Function:
double ml_CANapeQuitNonModalMode(int closeMode)

Parameters:
> int closeMode
  This parameter describes the mode of the shutdown of CANape. If the closeMode = 1, CANape is closed otherwise only the connection to CANape is terminated.

Description:
The ml_CANapeQuitNonModalMode function terminates the connection to CANape.
Example:
ml_CANapeQuitNonModalMode(1);

Return:
double OK = 1, NOK = 0

4.29 ml_CANapeMatlabConversion (deprecated)

Use ml_CANapeMDFConvert instead of that.

Function:
double ml_CANapeMatlabConversion(char *mdfFilename[], char *matlabFilename[])

Parameters:
> char *mdfFilename
  This parameter describes the name and the path of the MDF file which has to be converted.
> char *matlabFilename
  This parameter describes the name and the path of the MAT file.

Description:
The ml_CANapeMatlabConversion function is used for converting an MDF file into MATLAB format.

Example:
ml_CANapeMatlabConversion('D:\CCPsim\CANape.mdf',
                          'D:\CANape\CCPsim\CANape.mat');

Return:
double OK = 1, NOK = 0

4.30 ml_CANapeMDFConvert

Function:
double ml_CANapeMDFConvert(char *ID[], char *mdfFilename[], char *matlabFilename[], double overwrite)

Parameters:
> char *ID
  This parameter identifies the converter that should be used.
> char *mdfFilename
  This parameter describes the name and the path of the MDF file that has to convert.
> char *matlabFilename
  This parameter describes the name and the path of the result MAT file.
> double overwrite
  This parameter indicates the allowance to overwrite an existing result file. 1: overwrite allowed, 0: no overwrite

Description:
The ml_CANapeMDFConvert function is used for converting a recorder file into another format.

Return:
double OK = 1, NOK = 0

Example:
ml_CANapeMDFConvert('MDF2MAT','D:\CCPsim\CANape.mdf',
                       'D:\CANape\CCPsim\CANape.mat',1);
4.31 ml_CANapeMDFConverterCount

Function:
int[] ml_CANapeMDFConverterCount(void)

Parameter:
> none

Description:
The ml_CANapeMDFConverterCount function returns the number of available format converter.

Example:
[Res] = ml_CANapeMDFConverterCount();

Return:
int array with 2 elements
Res(:,1): 1 = OK, otherwise NOK
Res(:,2): amount value

4.32 ml_CANapeMDFConverterInfo

Function:
ml_CANapeMDFConverterInfo(int32 index)

Parameter:
> int32 index

Description:
The ml_CANapeMDFConverterInfo function delivers further information related to the converter with the given index.

Example:
[ConvInfo] = ml_CANapeMDFConverterInfo(int32(3))
ConvInfo =
    Comment: ''
    ConverterName: 'MATLAB Export-Konverter'
    ID: 'MDF2MAT'

Return:
Structure with 3 named elements; (Comment, ConverterName and ID). The ID could be used directly in the function ml_CANapeMDFConvert.

4.33 ml_CANapeSetDebugInfo

Function:
double ml_CANapeSetDebugInfo (int Debugverbosity)

Parameters:
> int Debugverbosity
    This parameter is only for debugging.

Description:
The ml_CANapeSetDebugInfo function is used for setting the debug verbosity. If it is set to 1, some message boxes appear with information. If it set to 2, all messages are displayed in the program of
DebugView from Sysinternals. This program can be downloaded from [www.sysinternals.com](http://www.sysinternals.com). The debug mode is turned off with 0.

Example:
ml_CANapeSetDebugInfo(1);

Return:
Returns the actual state of the debug info.

### 4.34 ml_CANapeGetLastError

**Function:**
double ml_CANapeGetLastError(void)

**Parameters:**
> none

**Description:**
This function is used for retrieving error information (error codes) about the previously executed function call.

Example:
Errorcode = ml_CANapeGetLastError();

**Note**
The description of the error values are defined in the CANape API help document (CANapAPI.chm).

**Return:**
double ErrorCode

### 4.35 ml_CANapeGetLastErrorText

**Function:**
double ml_CANapeGetLastErrorText(int ErrorNumber)

**Parameters:**
> int ErrorNumber

**Description:**
This function is used for retrieving error information (Text) about the previously executed function call.

Example:
Errorcode = ml_CANapeGetLastError();
Text = ml_CANapeGetLastErrorText(Errorcode);

**Return:**
char ErrorText

### 4.36 ml_CANapeGetApplicationVersion

**Function:**
double ml_CANapeGetApplicationVersion(void)
Parameters:
> none

Description:
This function is used for retrieving information about CANape. Following information is obtainable:
Main version, sub version and service pack number

Example:
```c
double ml_CANapeGetApplicationVersion();
```

Return:
```c
double array Version[0]. OK = 1, NOK = 0, fatal error = -1
    Version[1]. = major version
    Version[2]. = minor version
    Version[3]. = service pack version
```

4.37 ml_CANapeSwitchECUOnOffLine

Function:
double ml_CANapeSwitchECUOnOffLine(int ModuleHandle, int OnOff, int UpDownLoad)

Parameters:
> int ModuleHandle  
  Identification handle of the device  
> int OnOff  
  Switches the device from online to offline and the other way round  
> int UpDownLoad  
  If this parameter is set to true, CANape will execute a download when the ECU state is switched to online.

Description:
This function is used for connecting/disconnecting CANape and the ECU.

Example:
```c
Errorcode = ml_CANapeSwitchECUOnOffLine(ModuleHandle, 1, 1);
```

Return:
```c
double ErrorCode. Ok = 1, NOK = 0, fatal error = -1
```

4.38 ml_CANapeExecuteScript

Function:
double ml_CANapeExecuteScript(int ModuleHandle, char *ScriptText, bool ScriptType)

Parameters:
> int ModuleHandle  
  Identification handle of the device  
> scriptText  
  A script file name or a single script command  
> scriptType  
  Declares interpretation of parameter scriptText. If 'scriptType' is true, parameter 'scriptText' is interpreted as the file name of the script file to be executed. If 'scriptType' is false, 'scriptText' is interpreted as a single script command.
**Interface Programming between CANape and MATLAB**

**Note**
This function should be only used in the non-modal mode.
(ml_CANapeInit('c:\4000,200,100,1')).

**Description:**
This function executes a script file or a single script command.

**Example:**
```
ReturnCode = ml_CANapeExecuteScript(ModuleHandle,"Trigger()","",0);
```

**Return:**
double ErrorCode. Ok = 1, NOK = 0, fatal error = -1

### 4.39 ml_CANapeExecuteScriptEx

**Function:**
```
int[] ml_CANapeExecuteScriptEx(int ModuleHandle, char *ScriptText, bool ScriptType, int ScriptHandle)
```

**Parameters:**
- **int** ModuleHandle: Identification handle of the device
- **scriptText**: A script file name or a single script command
- **ScriptType**: Declares interpretation of parameter scriptText. If 'scriptType' is true, parameter 'scriptText' is interpreted as the file name of the script file to be executed. If 'scriptType' is false, 'scriptText' is interpreted as a single script command.

**Description:**
This function prepares the script treatment. In comparison to ml_CANapeExecuteScript, the script treatment does not start immediately.

**Example:**
```
[ScriptHandle] = ml_CANapeExecuteScriptEx(ModuleHandle,"ScriptBeep.scr",0);
```

**Return:**
int array with 2 elements
- Res(:,1): Ok = 1, NOK = 0, fatal error = -1
- Res(:,2): ScriptHandle - Identification handle of the script

(used in the script related functions e.g. ml_CANapeStartScript, ml_CANapeGetScriptState, ml_CANapeReleaseScript … )

The return array contains the result and the script identifier.

### 4.40 ml_CANapeStartScript

**Function:**
```
double ml_CANapeStartScript(int ScriptHandle, <char *command line, int module>)
```
4.41 ml_CANapeStopScript

Function:
double ml_CANapeStopScript(int ScriptHandle)

Parameters:
> int ScriptHandle
  Identification handle of the script

Description:
This function stops the script treatment.

Example:
ret = ml_CANapeStopScript(ScriptHandle);

Return:
ret: OK = 1, NOK = 0, fatal error = -1

4.42 ml_CANapeGetScriptState

Function:
int[] ml_CANapeGetScriptState(int ScriptHandle)

Parameters:
> int ScriptHandle
  Identification handle of the script

Description:
This function returns the current state of script treatment.

Example:
ret = ml_CANapeGetScriptState(ScriptHandle);

Return:
int array with 2 elements
  ret(:,1): OK = 1, NOK = 0, fatal error = -1
  ret(:,2): state of the script treatment
1: Ready // Initial status after creation of the task
2: Starting // Waiting in the list for execution
3: Running // Status if task was not finished in one Eval step
4: Sleeping // Function contained Sleep function
5: Suspended // Suspended by user
6: Terminated // Terminated by user
7: FinishedReturn // Successful finish, Return value available
8: FinishedCancel // Successful finish, No Return value
9: Failure // Failure
10: Timeout // Terminated due to timeout

4.43 ml_CANapeGetScriptResultValue

Function:
double[] ml_CANapeGetScriptResultValue (int ScriptHandle)

Parameters:
> int ScriptHandle
  Identification handle of the script

Description:
This function returns the result value of script treatment.

Example:
ret = ml_CANapeGetScriptResultValue (ScriptHandle);

Return:
double array with 2 elements
  ret(:,1): OK = 1, NOK = 0, fatal error = -1
  ret(:,2): result value

Note
The script has to use the script function “SetScriptResult” to provide a result value that could be retrieved.

4.44 ml_CANapeGetScriptResultString

Function:
cell[] ml_CANapeGetScriptResultString (int ScriptHandle)

Parameters:
> int ScriptHandle
  Identification handle of the script

Description:
This function returns the result string of script treatment.
Example:

\[
\text{[ret]} = \text{ml\_CANapeGetScriptResultString} \left( \text{ScriptHandle} \right);
\]

Return:

cell array with 2 elements

\[
\text{ret}\{;1\}: \text{OK} = 1, \text{NOK} = 0, \text{fatal error} = -1
\]

\[
\text{ret}\{;2\}: \text{result string}
\]

Note

The script has to use the script function “SetScriptResult” to provide a result string that could be retrieved.

4.45 ml\_CANapeReleaseScript

Function:

double ml\_CANapeReleaseScript(int ScriptHandle)

Parameters:

> int ScriptHandle

Identification handle of the script

Description:

This function removes the script from the task list.

Example:

\[
\text{ret} = \text{ml\_CANapeReleaseScript} \left( \text{ScriptHandle} \right);
\]

Return:

ret: OK = 1, NOK = 0, fatal error = -1

4.46 ml\_CANapeCheckOverrun

Function:

double ml\_CANapeCheckOverrun (int ModuleHandle, int TaskId, bool ResetOverrun)

Parameters:

> int ScriptHandle

Identification handle of the script

Description:

This function checks if data have been lost due to a FIFO overrun.
Example:
ret = ml_CANapeCheckOverrun (ModuleHandle, TaskId, True);

Return:
ret: OK = 1, NOK = 0, fatal error = -1

4.47 ml_CANapelsECUOnline

Function:
double ml_CANapelsECUOnline (int ModuleHandle)

Parameters:
> int ModuleHandle
  Identification handle of the device

Description:
This function returns the information whether the ECU is online or offline.

Example:
ReturnCode = ml_CANapelsECUOnline (ModuleHandle);

Return:
double ReturnCode
Is the value ReturnCode = 1, the device is online.
Is the value ReturnCode = 0, the device is offline.
Fatal error = -1.

4.48 ml_CANapeUseNAN

Function:
double ml_CANapeUseNAN (bool useNAN)

Parameters:
> bool useNAN
  new logical value for NAN handling

Description:
This function switches the NAN functionality on (1) or off (0).

Example:
ReturnCode = ml_CANapeUseNAN (false);

Return:
double ReturnCode
OK = 1, NOK = 0, Fatal error = -1.

4.49 ml_CANapelsNANUsed

Function:
struct ml_CANapelsNANUsed()
Parameters:
> none

Description:
This function returns the value of the NAN identification.

Example:
[Result] = ml_CANapeisNANUsed();

Return:
Result[0]: double, OK = 1, NOK = 0, fatal error = -1
Result[1]: logical, NAN value

4.50 ml_CANapeGetDBObjectListNames

Function:
char array ml_CANapeGetDBObjectListNames(int ModuleHandle)

Parameters:
> int ModuleHandle
Identification handle of the device

Description:
This function returns the names of all database objects as a list.

Example:
[DB_Names] = ml_CANapeGetDBObjectListNames(ModuleHandle);

Return:
The return value contains a list of all database names.

4.51 ml_CANapeGetMeasureListNames

Function:
char array ml_CANapeGetMeasureListNames(int ModuleHandle)

Parameters:
> int ModuleHandle
Identification handle of the device

Description:
This function returns the names of all measurement database objects as a list.

Example:
[DB_Measure_Names] = ml_CANapeGetMeasureListNames(ModuleHandle);

Return:
The return value contains a list of all measurement database names.
4.52 ml_CANapeGetDevices

Function:
char array ml_CANapeGetDevices(void)

Parameters:
> none

Description:
This function returns all devices and A2L file names from the CANape project.

Example:
[Devices] = ml_CANapeGetDevices();

Return:
The results are the device and A2L file name as a list like that.
'CCPsim'
'CCPsim.a2l'

4.53 ml_CANapeGetCalibrationListNames

Function:
char array ml_CANapeGetCalibrationListNames(int ModuleHandle)

Parameters:
> int ModuleHandle
Identification handle of the device

Description:
This function returns the names of all calibration database objects as a list.
Example:
[DB_Calibration_Names] = ml_CANapeGetCalibrationListNames(ModuleHandle);

Return:
The return value contains a list of all calibration database names.

4.54 ml_CANapeDefineRecorder

Function:
int[] ml_CANapeDefineRecorder(char* RecorderName)

Parameters:
> char* recorderName
name of the new recorder to allocate

Description:
This function defines a new MDF-recorder with the name ‘RecorderName’.

Example:
[recorderId] = ml_CANapeDefineRecorder(“RecorderNew”);
Return:
int array with 2 elements
   Res(:,1):  OK = 1, NOK = 0, fatal error = -1
   Res(:,2): recorderId
The return array contains the result and the recorder identifier of the allocated recorder.

4.55 ml_CANapeEnableRecorder

Function:
double ml_CANapeEnableRecorder(uint recorderId, bool enable)

Parameters:
   > uint recorderId
      Identification of the recorder.
   > bool enable
      Enables a recorder (enable=true) or deactivates it (enable=false).

Description:
This function enables or disables a recorder.

Example:
ret = ml_CANapeEnableRecorder(recorderId, true);

Return:
ret:  OK = 1, NOK = 0, fatal error = -1

4.56 ml_CANapeGetRecorderByName

Function:
struct ml_CANapeGetRecorderByName(char * recorderName)

Parameters:
   > char* recorderName
      Name of the recorder, see ml_CANapeDefineRecorder.

Description:
This function returns the recorderId of a specific recorder with a certain name.

Example:
[recByName] = ml_CANapeGetRecorderByName("RecorderNew");

Return:
recByName[0]:  OK = 1, NOK otherwise
recByName[1]:  recorderId

4.57 ml_CANapeGetRecorderCount

Function:
int[] ml_CANapeGetRecorderCount(void)

Parameters:
   > none

Description:
This function returns the number of available recorders.
Example:
[count] = ml_CANapeGetRecorderCount();

Return:
int array with 2 elements
count[0]: OK = 1, NOK otherwise
count[1]: amount value

4.58 ml_CANapeGetRecorderDataReduction

Function:
int[] ml_CANapeGetRecorderDataReduction(int recorderId)

Parameters:
> int recorderId
Identification of the recorder.

Description:
This function returns the data reduction parameter for a recorder.

Example:
[res] = ml_CANapeGetRecorderDataReduction(recId);

Return:
res[0]: OK = 1, NOK otherwise
res[1]: data reduction parameter

4.59 ml_CANapeGetRecorderMdfFileName

Function:
struct ml_CANapeGetRecorderMdfFileName (int recorderId)

Parameters:
> int recorderId
Identification of the recorder.

Description:
This function returns the name of the recorder with the given recorderId.

Example:
[res] = ml_CANapeGetRecorderMdfFileName(recId);
if(res.Result > 0)
    recFileName = res.FileName;
end

Return:
Structure with 2 named elements: Result, FileName.
res.Result: OK = 1, NOK otherwise
res.FileName: filename, if successful
4.60 ml_CANapeGetRecorderName

Function:
`char* ml_CANapeGetRecorderName (int recorderId)`

Parameters:
> int recorderId
   Identification of the recorder.

Description:
This function returns the name of the recorder with the given recorderId.

Example:
```
[res] = ml_CANapeGetRecorderName (recorderId);
```

Return:
Structure with 2 named elements: Result, Recordername.
res.Result : OK = 1, NOK otherwise
res.Recordername: recorder name, if successful

4.61 ml_CANapeGetRecorderState

Function:
`int[] ml_CANapeGetRecorderState (int recorderId)`

Parameters:
> int recorderId
   Identification of the recorder.

Description:
This function returns the operational state of a recorder.

Example:
```
[res] = ml_CANapeGetRecorderState (recorderId);
```

Return:
res[0] : OK = 1, NOK otherwise
res[1] recorder state
  0: recorder configured
  1: recorder is active and ready to run
  2: recorder is running
  3: recorder paused, but measurement is still running
  4: recorder has stopped

4.62 ml_CANapeGetSelectedRecorder

Function:
`int[] ml_CANapeGetSelectedRecorder (void)`

Parameters:
> None

Description:
This function returns the selected recorder.
Example:
[res] = ml_CANapeGetSelectedRecorder();

Return:
int array with 2 elements
res[0]: OK = 1, NOK otherwise
res[1]: recorderId

4.63 ml_CANapeRecorderEnabled

Function:
struct ml_CANapeRecorderEnabled (int recorderId)

Parameters:
> int  recorderId
   Identification of the recorder.

Description:
This function returns the enable/disable state of the recorder.

Example:
[res] = ml_CANapeGetCalibrationListNames(recorderId);

Return:
Structure with 2 named elements: Result, Enabled.
res.Result:  OK = 1, NOK otherwise
res.Enabled: enabled state, if successful (true = enabled)

4.64 ml_CANapePauseRecorder

Function:
double ml_CANapePauseRecorder (int recorderId, bool pause)

Parameters:
> int  recorderId
   Identification of the recorder.
> bool  pause
   pause = true, recorder pauses recording.
pause = false, recorder continues recording.

Description:
This function pauses or continues the recording in the mdf-file.

Example:
res = ml_CANapePauseRecorder(recorderId, true);

Return:
res: OK = 1, NOK = 0, fatal error = -1

4.65 ml_CANapeRemoveRecorder

Function:
double ml_CANapeRemoveRecorder (int recorderId)
Parameters:
> int recorderId
   Identification of the recorder

Description:
This function deletes the recorder with the recorder id.

Example:
[res] = ml_CANapeRemoveRecorder(recorderId);

Return:
res: OK = 1, NOK = 0, fatal error = -1

4.66 ml_CANapeSelectRecorder

Function:
double ml_CANapeSelectRecorder (int recorderId)

Parameters:
> int recorderId
   Identification of the recorder

Description:
This function selects the recorder with the recorder id.

Example:
res = ml_CANapeSelectRecorder(recorderId);

Return:
res: OK = 1, NOK = 0, fatal error = -1

4.67 ml_CANapeSetRecorderDataReduction

Function:
double ml_CANapeSetRecorderDataReduction (int recorderId, int reduction rate)

Parameters:
> int recorderId
   Identification of the recorder
> int reduction rate

Description:
This function sets the data reduction parameter.

Example:
res = ml_CANapeSetRecorderDataReduction(recorderId, int32(4));

Return:
res: OK = 1, NOK = 0, fatal error = -1

4.68 ml_CANapeSetRecorderMdfFileName

Function:
double ml_CANapeSetRecorderMdfFileName (int recorderId, char* fileName)
Parameters:

> int recorderId
  Identification of the recorder
> char* fileName
  name of the recorder file

Description:
This function sets the file name of the MDF- recorder file.

Example:
res = ml_CANapeSetRecorderMdfFileName(recorderId, 'CAN_Test1.mdf');

Return:
res: OK = 1, NOK = 0, fatal error = -1

5  Diagnostic

5.1  ml_CANapeDiagCreateSymbolicRequest

Function:
double array ml_CANapeDiagCreateSymbolicRequest(int ModuleHandle, char* ServiceName, int TimeOut)

Parameters:
> int ModuleHandle
  Identification handle of the device
> char * ServiceName
  This parameter describes the name of the service.
> int TimeOut
  This parameter is responsible for activating a callback function, if calling a CANapeDiagExecute (... ) request.

Description:
This function returns the handle of the diagnostic session.

Example:
[t] = ml_CANapeDiagCreateSymbolicRequest(ModuleHandle, 'IDENTIFICATION/Serial_Number/Write', 3000)

Return:
The return value contains 2 values.
t[0] = Return code OK=1, NOK = 0, Timeout = 2
t[1] = Diagnostic session handle

5.2  ml_CANapeDiagExecute

Function:
double ml_CANapeDiagExecute(unsigned long DiagHandle, int SupressPositiveResponse);

Parameters:
> unsigned long DiagHandle
  Identification handle of the diagnostic session
> int SupressPositiveResponse
    This parameter suppresses the positive response from the ECU

**Description:**
The function executes a diagnostic request.

**Example:**
ReturnCode = ml_CANapeDiagExecute(DiagHandle0, 1);

**Return:**
double ErrorCode. Ok = 1, NOK = 0

### 5.3 ml_CANapeDiagGetServiceState

**Function:**
double array ml_CANapeDiagGetServiceState(unsigned long DiagHandle);

**Parameters:**
> unsigned long DiagHandle
    Identification handle of the diagnostic session

**Description:**
The function queries the state of a Request.

**Example:**
[t] = ml_CANapeDiagGetServiceState(DiagHandle);

**Return:**
The return value contains 2 values.

\[ t[0] = \text{Return code OK=1, NOK = 0} \]
\[ t[1] = \text{state of a request} \]

### 5.4 ml_CANapeDiagReleaseService

**Function:**
double array ml_CANapeDiagReleaseService(unsigned long DiagHandle);

**Parameters:**
> unsigned long DiagHandle
    Identification handle of the diagnostic session

**Description:**
The function removes a service.

**Example:**
ErrorCode = ml_CANapeDiagReleaseService(DiagHandle);

**Return:**
double ErrorCode. Ok = 1, NOK = 0

### 5.5 ml_CANapeDiagGetResponseCount

**Function:**
double array ml_CANapeDiagGetResponseCount(unsigned long DiagHandle);
Parameters:
  > unsigned long DiagHandle
  Identification handle of the diagnostic session

Description:
The function returns the count of incoming responses.

Example:
\[ \texttt{[t]} = \texttt{ml\_CANapeDiagGetResponseCount(DiagHandle)}; \]

Return:
The return value contains 2 values.
\[ \texttt{t[0]} = \text{Return code OK}=1, \text{NOK} = 0 \]
\[ \texttt{t[1]} = \text{Count of responses} \]

5.6 \texttt{ml\_CANapeDiagIsPositiveResponse}

Function:
double array \texttt{ml\_CANapeDiagIsPositiveResponse(unsigned long DiagHandle, long CurrentID)};

Parameters:
  > unsigned long DiagHandle
  Identification handle of the diagnostic session
  > long CurrentID
  This is the ID of the response

Description:
The Ask whether the response is positive or negative.

Example:
\[ \texttt{[t]} = \texttt{ml\_CANapeDiagIsPositiveResponse(DiagHandle,CurrentID)}; \]

Return:
Return code yes=1, no = 0

5.7 \texttt{ml\_CANapeDiagGetResponseCode}

Function:
char array \texttt{ml\_CANapeDiagGetResponseCode(unsigned long DiagHandle, long ResponseID)};

Parameters:
  > unsigned long DiagHandle
  Identification handle of the diagnostic session
  > long ResponseID
  This is the ID of the response

Description:
Return the raw response byte stream.

Example:
\[ \texttt{[Data]} = \texttt{ml\_CANapeDiagGetResponseCode(DiagHandle, ResponseID)}; \]

Return:
Data stream
5.8 ml_CANapeDiagSetStringParameter

Function:
double ml_CANapeDiagSetStringParameter(unsigned long DiagHandle, char *ParameterName, char *Value);

Parameters:
> unsigned long DiagHandle
  Identification handle of the diagnostic session
> char *ParameterName
  This is the identification name of the parameter
> char *Value
  This sets the value of the identification name

Description:
The function set a symbolic parameter value.

Example:
ReturnCode = ml_CANapeDiagSetStringParameter(DiagHandle, 'Serial_Number', '4');

Return:
double ErrorCode. Ok = 1, NOK = 0

5.9 ml_CANapeDiagGetStringResponseParameter

Function:
char array ml_CANapeDiagGetStringResponseParameter (unsigned long DiagHandle, char *Name, long ResponseID);

Parameters:
> unsigned long DiagHandle
  Identification handle of the diagnostic session
> char *ParameterName
  This is the identification name of the parameter
> long ResponseID
  This is the ID of the response

Description:
The function returns a symbolic response parameter.

Example:
[ReturnString] = ml_CANapeDiagGetStringResponseParameter(DiagHandle, 'Serial_Number', CurrentID);

Return:
The function returns the value as string.

5.10 ml_CANapeDiagSetRawParameter

Function:
double ml_CANapeDiagSetRawParameter(unsigned long DiagHandle, char *Name, unsigned char *Data);

Parameters:
> unsigned long DiagHandle
  Identification handle of the diagnostic session
5.11 ml_CANapeDiagGetRawResponseParameter

Function:
char array ml_CANapeDiagGetRawResponseParameter(unsigned long DiagHandle, char *Name, long ResponseID);

Parameters:
- unsigned long DiagHandle
  Identification handle of the diagnostic session
- char *ParameterName
  This is the identification name of the parameter
- long ResponseID
  This is the ID of the response

Description:
The function returns a raw response parameter.

Example:
[Raw_Data] = ml_CANapeDiagGetRawResponseParameter(DiagHandle,'Serial_Number', CurrentID);

Return:
The function returns raw values.

5.12 ml_CANapeDiagIsComplexResponseParameter

Function:
double ml_CANapeDiagIsComplexResponseParameter(unsigned long DiagHandle char *Name, long ResponseID);

Parameters:
- unsigned long DiagHandle
  Identification handle of the diagnostic session
- char *ParameterName
  This is the identification name of the parameter
- long ResponseID
  This is the ID of the response

Description:
The function checks whether the response parameter is complex or not.
Example:
```c
returnValue = ml_CANapeDiagIsComplexResponseParameter(DiagHandle, 'LIST_OF_DTC_AND_STATUS', 0);
```

Return:
double returnValue. Ok = 1, NOK = 0

5.13 ml_CANapeDiagGetComplexIterationCount

Function:
double ml_CANapeDiagGetComplexIterationCount(unsigned long DiagHandle char *Name, long ResponseID);

Parameters:
- unsigned long DiagHandle
  Identification handle of the diagnostic session
- char *ParameterName
  This is the identification name of the parameter
- long ResponseID
  This is the ID of the response

Description:
The function returns the count of iterations concerning a given parameter.

Example:
```c
Count = ml_CANapeDiagGetComplexIterationCount(DiagHandle, 'LIST_OF_DTC_AND_STATUS', CurrentID);
```

Return:
double value.

5.14 ml_CANapeDiagGetComplexRawResponseParameter

Function:
char array ml_CANapeDiagGetComplexRawResponseParameter(unsigned long DiagHandle char *Name, long ResponseID, char *SubParameter, unsigned long IterationIndex);

Parameters:
- unsigned long DiagHandle
  Identification handle of the diagnostic session
- char *Name
  This is the identification name of the parameter
- long ResponseID
  This is the ID of the response
- unsigned long IterationIndex
  This is the counter of the iteration index

Description:
Returns the raw response by stream.

Example:
```c
[t] = ml_CANapeDiagGetComplexRawResponseParameter(DiagHandle, 'LIST_OF_DTC_AND_STATUS', 0, 'DTCStatusByte.TestFailed', index);
```
Return:
The return value contains the whole byte stream.

5.15 ml_CANapeDiagGetComplexStringResponseParameter

Function:
char array ml_CANapeDiagGetComplexStringResponseParameter(unsigned long DiagHandle char *Name, long ResponseID, char *SubParameter, unsigned long IterationIndex);

Parameters:
> unsigned long DiagHandle
  Identification handle of the diagnostic session
> char *Name
  This is the identification name of the parameter
> long ResponseID
  This is the ID of the response
> unsigned long IterationIndex
  This is the counter of the iteration index

Description:
Returns a symbolic response parameter.

Example:
[t] = ml_CANapeDiagGetComplexStringResponseParameter
  (DiagHandle, 'LIST_OF_DTC_AND_STATUS', 0,
   'DTCStatusByte.TestFailed', index);

Return:
The return value contains the whole string.

5.16 ml_CANapeDiagExecuteJob

Function:
void ml_CANapeDiagExecuteJob(unsigned short ModuleHandle, char *Job, char *Comment);

Parameters:
> unsigned short ModuleHandle
  Identification handle of the module
> char *Job
  This is the identification name of the job, which should be executed
> char *Comment
  This is the comment or parameter which can be used

Description:
Returns a value.

Example:
[t] = ml_CANapeDiagExecuteJob(ModuleHandle, 'Genric_Jobs_New_Job_1_2', '');
Return:
The return value contains a string or a double value

- Ok = 1
- OK = ‘String’
- Failed = 0.

6 Measurement example

The current chapter describes an example of a measurement with the ECU simulation CCPSim from Vector.

6.1 CCPSim

The first step is to execute CCPSim. The program is located in the folder ($CANape\ccpsim).

The second step is to execute CCPSim again with other parameters (e.g., CCPSim.exe –dto1 –cto2), for more information type “h”.

Note
Please check the configuration settings of the CAN hardware if there is no connection between CANape and CCPSim.

6.2 CANape

After the execution of 2 * CCPSim, CANape can be started with the option –u for the non-modal mode. CANape is automatically executed by using the function CANapeInit (see chapter 4.2).

6.3 MATLAB

Start MATLAB and change the working directory to the path which was set during the installation (see chapter 2.3) and open the file CCPSIMDaqMehrkanaeleMatrix.m.
The following window appears:

![Script example (measurement DAQ)](image)

Figure 6: Script example (measurement DAQ)

Start the program by clicking the button (for more information, see the MATLAB help). The program starts CANape, initializes and sets the project folder. The next instructions load the A2L files and configure the CAN channel. The other two statements are used for the configuration of the measurement (for more details see chapter 4.13). If the configuration of the data acquisition has been successful, the measurement is started with the instruction `ml_CANapeStartMeasurement`.

The data acquisition starts and with the function `ml_CANapeGetCurrentValueMatrix` the records are returned to the specified parameters (data, time). The plot command from MATLAB displays the whole measurement data.
With the function `ml_CANapeStopMeasurement` the measurement is stopped. The function `ml_CANapeExit` terminates CANape and closes all windows of CANape.

7 Calibration example

This example describes a configuration for calibration with the ECU simulation CCPSim from Vector.

7.1 CCPSim

The first step is to execute `CCPSim`. The program is located in the folder ($CANape\ccpsim).

7.2 CANape

After the execution of CCPSim, CANape can be started with the option –u for the non-modal mode. CANape is automatically executed by using the function `CANapeInit` (see chapter 4.2).

8 MATLAB

8.1 Read Calibration

Start MATLAB and change the working directory to the path which has been set during the installation (see chapter 2.3) and open the file `CCPSIMCalibrationMap.m`. The following window appears:
Figure 8: Script example calibration (read)

Start the program by clicking the button (for more information see MATLAB help). The program starts CANape, initializes and sets the project folder. The next instruction loads the A2L file and configures the CAN channel. The calibration starts with the instruction ml_CANapeReadCalibrationObject. This example loads the calibration data from the map KF2 and the other three commands from MATLAB display the following window:
Figure 9: 3-D model
8.2 Write Calibration

Open the M-File `CCPSIMWrite CalibrationMap.m` with the MATLAB editor.

```matlab
%% REVISION HISTORY
%% Date       Version   Author    Description
% 2003-18-09  1.00.00   DS        - Created
% 2005-25-01  1.00.02   DS        - Change of ml_CANapeInit
% - Implementation of a return value by
%  ml_CANapeExit

*** Init and load ASAP-File
Return = ml_CANapeInit('D:\Programme\CANape 6.0\CCPSim\', 9000, 300, 20000, 0);
[y] = ml_CANapeOpenAsap2File('D:\Programme\CANape 6.0\CCPSim\CCPSim.asl', 1);
ModuleHandle_1 = 2(:,1)
data = [ ];
for p=1:2
    for q=1:8
        data[q,p]=q*0.1
    end
end

% Vector Kennfeld
g=0;
    for p=3:10
        s=s+1;
        for q=1:8
            data[q,p]=p^1.1
        end
end
[y] = ml_CANapeWriteCalibrationObject(ModuleHandle_1,'map4_30_uw',1,data)
Vert = ml_CANapeExit(10);
```

Figure 10: Script example calibration (write)

Start the script by clicking the run button. The script starts CANape, loads the ASAP file, and connects to the CCPSim ECU simulator.
The next instruction creates a data matrix like this:

<table>
<thead>
<tr>
<th>X-Axes</th>
<th>Y-Axes</th>
<th>Values</th>
<th>Values</th>
<th>Values</th>
<th>Values</th>
<th>Values</th>
<th>Values</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>0.1</td>
<td>3.3</td>
<td>4.4</td>
<td>5.5</td>
<td>6.6</td>
<td>7.7</td>
<td>8.8</td>
<td>9.9</td>
</tr>
<tr>
<td>0.2</td>
<td>0.2</td>
<td>3.3</td>
<td>4.4</td>
<td>5.5</td>
<td>6.6</td>
<td>7.7</td>
<td>8.8</td>
<td>9.9</td>
</tr>
<tr>
<td>0.3</td>
<td>0.3</td>
<td>3.3</td>
<td>4.4</td>
<td>5.5</td>
<td>6.6</td>
<td>7.7</td>
<td>8.8</td>
<td>9.9</td>
</tr>
<tr>
<td>0.4</td>
<td>0.4</td>
<td>3.3</td>
<td>4.4</td>
<td>5.5</td>
<td>6.6</td>
<td>7.7</td>
<td>8.8</td>
<td>9.9</td>
</tr>
<tr>
<td>0.5</td>
<td>0.5</td>
<td>3.3</td>
<td>4.4</td>
<td>5.5</td>
<td>6.6</td>
<td>7.7</td>
<td>8.8</td>
<td>9.9</td>
</tr>
<tr>
<td>0.6</td>
<td>0.6</td>
<td>3.3</td>
<td>4.4</td>
<td>5.5</td>
<td>6.6</td>
<td>7.7</td>
<td>8.8</td>
<td>9.9</td>
</tr>
<tr>
<td>0.7</td>
<td>0.7</td>
<td>3.3</td>
<td>4.4</td>
<td>5.5</td>
<td>6.6</td>
<td>7.7</td>
<td>8.8</td>
<td>9.9</td>
</tr>
<tr>
<td>0.8</td>
<td>0.8</td>
<td>3.3</td>
<td>4.4</td>
<td>5.5</td>
<td>6.6</td>
<td>7.7</td>
<td>8.8</td>
<td>9.9</td>
</tr>
</tbody>
</table>

Table 7: 3-D matrix

After the creation of the matrix, the function `ml_CanapeWriteCalibrationObject` writes all of the values to the map "map4_80_uc".

9 Database Example

The current chapter describes an example to determine the names of database objects with the ECU simulation CCPSim from Vector.

9.1 Read all Objects

Start MATLAB and change the working directory to the path which has been set during the installation (see chapter 2.3) and open the file `CCPSIMDBListAllNames.m`. 
The following window appears:

```
27 - k = [];  
28 - s = [];  
29 - % Set the debug info verbosity  
30 - ml_CANapeSetDebugInfo(0);  
31 - % Get the current version  
32 - [v] = ml_CANapeGetInterfaceVersion();  
33 - k = x;  
34 - Major = k(:,1)  
35 - Minor = k(:,2)  
36 - % Start CANape  
37 - Return = ml_CANapeInit('D:\Programme\CANape Versions\CANape 6.0');  
38 - % Load the A2L file  
39 - [x] = ml_CANapeOpenA2LFile('D:\Programme\CANape Versions\CANape 6.0\CCPSim.a2l');  
40 - s = x;  
41 - ModuleHandle_1 = s(:,2)  
42 - ChannelNames = [];  
43 - % Determine all database objects  
44 - [t] = ml_CANapeGetObjectListNames(ModuleHandle_1);  
45 - ChannelNames = [ChannelNames, t];  
46 - % Show the whole list of all database names  
47 - disp(ChannelNames);  
48 - % Close CANape  
49 - return = ml_CANapeExit();
```

Figure 11: Script example to read database objects

Start the program by clicking the button (for more information see MATLAB help). The program starts CANape, initializes and sets the project folder. The next instruction loads the A2L file and configures the CAN channel. With the instruction `ml_CCFISIMDBListAllNames` it is possible to determine all of the database objects. This example loads the database names from the `CCPSim.a2l` file. The other command from MATLAB displays the whole list in the command window.
10 ECU Example

Start MATLAB and change the working directory to the path which has been set during the installation (see chapter 2.3) and open the file CCPSIMECUTasksListAll.m.
Start the program by clicking the button (for more information see MATLAB help). The program starts CANape, initializes and sets the project folder. The next instruction loads the A2L file and configures the CAN channel. With the instruction `ml_GetEcuTasksAllInfos` it is possible to determine all tasks of the ECU. This example loads the database names from the CCPSim.a2l file. The other command from MATLAB displays the whole list in the command window.
The interpretation of the list:

<table>
<thead>
<tr>
<th>Index</th>
<th>Task ID</th>
<th>Rate</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0 = Event</td>
<td>polling</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>0</td>
<td>Cyclic</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>10 = Time in ms</td>
<td>10 ms</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>100</td>
<td>100 ms</td>
</tr>
<tr>
<td>4</td>
<td>12</td>
<td>0</td>
<td>Key_T</td>
</tr>
</tbody>
</table>

Table 8: Task ID matrix

11 Contacts

For a full list with all Vector locations and addresses worldwide, please visit [http://vector.com/contact/](http://vector.com/contact/).