Canalyzer
Canoe
© 2004, 2005 Vector CANtech, Inc
Novi, Michigan 48375 USA

The authors and publishers have used their best efforts in preparing this book. These efforts include development, research, and testing of the theories, principles, and programming sample code so as to determine their effectiveness. The authors and/or publishers make no warranty, expressed or implied, with regard to the sample code or to any other documentation contained in this book. The authors and/or publishers shall not be liable in any event for incidental or consequential damages in connection with, or arising out of, the furnishing, performance, or use of the sample code or any of the contents of this book.

All rights reserved.
No part of this book may be reproduced, in any form or by any means, without express permission in writing from Vector CANtech, Inc.
Table of Contents

CAPL Function Reference Manual ................................................................. 1
Table of Contents ....................................................................................... 3
Preface ........................................................................................................... 7
 About This Book ......................................................................................... 7
 Organization ............................................................................................... 7
 Acknowledgments ....................................................................................... 7
 Tell Us What You Think! ........................................................................... 8
 Guide to the Use of This Book ................................................................. 9
 The Main Entry Function ........................................................................... 9
 Obsolete ....................................................................................................... 9
 Syntax ......................................................................................................... 9
 Description ............................................................................................... 9
 Parameter .................................................................................................. 9
 Returns ...................................................................................................... 9
 Availability ............................................................................................... 10
 Observation ............................................................................................. 10
 Recommendation ..................................................................................... 10
 Branch Compatibility ............................................................................... 10
 Related Functions .................................................................................... 11
 Example .................................................................................................... 11
 The CAPL Functions ................................................................................ 12
 abs ............................................................................................................ 13
 atol .......................................................................................................... 14
 beep ......................................................................................................... 15
 callAllOnEnvVar .................................................................................... 16
 cancelTimer .............................................................................................. 17
 canOffline ............................................................................................... 18
 canOnline ............................................................................................... 19
 canSetChannelAcc .................................................................................. 20
 canSetChannelMode ............................................................................... 21
 canSetChannelOutput .......................................................................... 22
 cos ........................................................................................................... 23
 elCount ................................................................................................... 24
 enableControl ......................................................................................... 25
 exp .......................................................................................................... 26
 fileClose ................................................................................................. 27
 getFileBinaryBlock ............................................................................... 28
 getString ............................................................................................... 29
 fileGetStringSZ ..................................................................................... 31
 fileName ................................................................................................. 33
 filePutString .......................................................................................... 34
 fileReadArray ........................................................................................ 35
 fileReadFloat ......................................................................................... 37
 fileReadInt ............................................................................................. 38
random.........................................................................................................92
replayResume..............................................................................................93
replayStart ..............................................................................................94
replayState..............................................................................................95
replayStop..............................................................................................96
replaySuspend...........................................................................................97
resetCan .................................................................................................98
resetCanEx............................................................................................99
runError.................................................................................................100
seqFileClose ...........................................................................................101
seqFileGetBlock .....................................................................................102
seqFileGetLine .......................................................................................103
seqFileGetLineSZ ..................................................................................104
seqFileLoad.............................................................................................105
seqFileRewind .........................................................................................106
setBtr ......................................................................................................107
setBusContext .........................................................................................108
setCanCabsMode ....................................................................................109
setControlBackColor .............................................................................110
setControlForeColor .............................................................................111
setControlProperty ...............................................................................112
setDrift ....................................................................................................113
setFilePath ............................................................................................114
setJitter ...................................................................................................115
setLogFile ...............................................................................................116
setMsgTime ............................................................................................117
setOcr ......................................................................................................118
setPortBits ............................................................................................119
setPostTrigger .......................................................................................120
setPreTrigger .........................................................................................121
setStartDelay .........................................................................................122
setTimer ..................................................................................................123
setWriteDbgLevel ..................................................................................124
setWritePath ..........................................................................................125
sin ...........................................................................................................126
snprintf ..................................................................................................127
sqrt .........................................................................................................128
startLogging ..........................................................................................129
startStatisticAcquisition ......................................................................130
stop .........................................................................................................131
stopLogging ..........................................................................................132
stopStatisticAcquisition .......................................................................133
strlen......................................................................................................134
strncat ....................................................................................................135
strncmp..................................................................................................136
strncpy ....................................................................................................137
strncpyy .................................................................................................138
swapDWord ................................................................. 139
swapInt ................................................................. 140
swapLong .............................................................. 141
swapWord ............................................................... 142
sysExit ................................................................. 143
sysMinimize .......................................................... 144
timeDiff ............................................................... 145
timeNow .............................................................. 146
timeNowFloat ......................................................... 147
trigger ................................................................. 148
valOfId ................................................................. 149
write ................................................................. 150
writeClear ............................................................ 151
writeCreate .......................................................... 152
writeDbgLevel ....................................................... 153
writeDestroy ........................................................ 154
writeEx ............................................................... 155
writeLineEx .......................................................... 156
writeProFileFloat .................................................. 157
writeProFileInt ...................................................... 158
writeProFileString .................................................. 159
writeTextBkgColor ................................................ 160
writeTextColor ...................................................... 161
writeToLog ........................................................... 162
writeToLogEX ....................................................... 164
Compatibility Chart .................................................. 166
Availability Chart ................................................... 170
Preface

The *CAPL Functions Reference Manual* presents a complete description of all 150 functions of the Vector CAN Application Programming Language (CAPL), the programming language foundation of Vector CANoe and CANalyzer – two of Vector's most popular development tools. CAPL is a rich, robust tool used to extend the power of CANoe and CANalyzer beyond the tool's interfaces and to customize tool functionality to the user's requirements.

About This Book

This book assumes that the programming experience level of the user includes individuals with some experience in the C programming language, in addition to those with C coding experience, who wish to use this as a reference book to CAPL functions.

This material is suitable for college programs that focus on electrical engineering, computer engineering, computer science, distributed control systems and distributed embedded systems that use the CAN protocol. The target audience is engineering students, faculty, practicing engineers, and electronic technicians.

Organization

This book is organized into two major sections. The second section, the main section, consists of approximately one page devoted to every function in the CAPL programming language. It includes the syntax of the function, a description, any parameters, any value returned by the function, compatibility, references to related functions, and a code example of how the function is used in a CAPL program. The first section explains these sections in more detail.

Acknowledgments

The original creator of CAPL is Dr. Helmut Schelling, who also developed and authored the first compiler and first editor for the CAPL programming language.

Jurgen Kluser incorporated the data structural elements of the CAPL programming language into the Vector CANdb database tool. Additionally, those who participated in continuing the development of the CAPL programming language equally deserve credit, and these individuals include Thomas Riegraf and the CANoe/CANalyzer development teams.

On the authoring side, it is important to recognize several individuals who have made significant contributions to this book, including Jun Lin, Tom Guthrie, and Mike Alexander.
Tell Us What You Think!

We believe that you, the reader, are the most important person of all, since it is you who will benefit from reading this book. We value your input, and we would like to know what we’re doing right, what we could do better, what things you think are important that we haven't covered, and any other comments you might have.

You can fax, e-mail or write us directly to let us know what you did or didn't like about this book – as well as what we can do to make our books better.

When you write, please include the title of this book, as well as your name and phone or fax number. We will carefully review your comments and share them with the authors and editors who worked on the book.

E-mail: CAPL_Books@Vector-CANtech.com

Address: 39500 Orchard Hill Place Novi, MI USA 48375

Fax: 248-449-9704
Guide to the Use of This Book

The Main Entry Function

The functions, including single words, compound words, and abbreviations, are listed in the upper corner of each page. Functions are listed in alphabetical order, and are set in a large boldface font. In CAPL, the naming convention of functions follows three simple rules.

- All standard C functions are in lower case (e.g. `sin()`, `cos()`, `strlen()`, `strncat()`)
- Non-C one-word function names are in lower case (e.g. `trigger()`, `outport()`, `inport()`)
- For non-C function names with more than one word, capitalize the first letter of all words except the first (e.g. `swapInt()`, `timeDiff()`, `putValueToControl()`)

If the entry function has line strikethrough the name, it means it is an obsolete function. Read the Recommendation section for another function or method to use.

Obsolete

Obsolete functions can still be used in CAPL programming; however, they are not recommended for long-term use in the future especially in newer versions of the software. If the main entry function is obsolete, a replacement function is indicated. If the main entry function is not obsolete, “N/A” is displayed.

Note: No support will be given to an obsolete function as new software releases are issued.

Syntax

Functions have one or more syntax. The Syntax section describes the function return data type, the function name, and the type of parameters it has.

Description

This explains the operation of the function. If the function has more than one syntax use, it will be explained here.

Parameter

This section describes all of the parameters the function contained. Value and the parameter name are separated by an equal sign. Additional notes are within parentheses.

Returns

A function may return a value of a specific type (some do not and are so denoted as void). For some functions, the return value determines whether the function call is successful or unsuccessful. For others, the return value may be the number of characters/bytes returned (functions dealing with strings or arrays). In both cases, the return value of interest lies within one of the parameters.

For example,

    long getValue (EnvVarName, byte buffer[]);

the return value of the `getValue()` function determines the number of bytes copied. The buffer parameter holds the true value retrieve from the environment variable, `EnvVarName`. In
addition, a function with more than one syntax may return values of different types for each syntax. The `getValue()` function is on good example.

**Availability**

This section indicates the software version when the function was first introduced or last used after it became obsolete. The earliest software version this book considers is Version 2.5.

If a function has been obsolete, it generally gets replaced by another function. If that is the case, the newer function should be mentioned in the Recommendation section.

**Observation**

This section gives useful comments on using the entry function.

**Recommendation**

This section gives recommendation on using the entry function. If the entry function has a line strikethrough the name, the newer function or method is referenced here.

**Branch Compatibility**

A function may have limitation on where it can be used in the CANalyzer/CANoe software tool.

Based on the speed of currently available PCs, some CAPL functions are too slow to be used in CANalyzer’s Transmit Branch or CANoe’s Simulation Branch for some real-time activities. The `getLocalTime()` function used to get the Window’s clock is one of them. Also, it makes some sense to have it available only in the Analysis Branch for tracking data evaluations and logging.

**Figure 1 – CAPL Functions Depend on Placement**

Since the CAPL Browser cannot tell where the P Block is placed in the setup window of CANalyzer or CANoe, sometimes it will not detect whether some of the functions in the program are not allowed in that branch (e.g., `seqFile...()` functions). If such file I/O functions are used, compile the CAPL program using the compile option in CANalyzer or CANoe’s
main menu. This action provides the compiler with the location of the P Block, allowing it to recognize restricted function calls.

**Related Functions**

This section displays closely related functions to the entry function.

**Example**

This section gives an example(s) using the entry function.
The CAPL Functions

This chapter presents a detailed description of all the CAPL functions.

Every function is listed that is included in CAPL Version 2.5 or later.
The CAPL Functions

**abs**

**Syntax**

```c
int abs (int num);
long abs (long num);
double abs (double num);
```

**Description**

Returns the absolute value of a signed number. Return type matches the parameter type.

**Parameter**

`num` = number to be converted

**Returns**

Integer, long integer, or double

**Availability**

This function is supported in Version 2.5 and after.

**Branch Compatibility**

- CANalyzer's Transmit Branch = Yes
- CANalyzer's Analysis Branch = Yes
- CANoe's Simulation Branch = Yes
- CANoe's Analysis Branch = Yes

**Related Functions**

N/A

**Example**

```c
int one = -1;
long two = -2;
double three = -3.5;
write("%d %d %lf %g", abs(one), abs(two), abs(three), abs(three));
//This prints "1 2 3.500000 3.5" in the write window.
```
The CAPL Functions

### atol

#### Syntax

```
long atol (char s[]);
```

#### Description

Converts a string to a decimal number. If the string starts with "0x", base 16 is used. Leading blanks are discarded.

#### Parameter

- `s` = string to be converted

#### Returns

Long integer

#### Availability

Available in all versions.

#### Branch Compatibility

- CANalyzer's Transmit Branch = Yes
- CANalyzer's Analysis Branch = Yes
- CANoe's Simulation Branch = Yes
- CANoe's Analysis Branch = Yes

#### Related Functions

ltoa

#### Example

```
long z1;
long z2;
z1 = atol("200");
z2 = atol("0xFF");
...
//Result: z1 = 200, z2 = 255
```
The CAPL Functions

beep

Syntax
void beep (int freq, int duration);

Description
Outputs a tone to the computer's speaker.

Parameter
freq = integer for tone pitch
duration = integer for tone duration

In the Windows version, the parameters freq defines the tone output. Different sounds are defined in the section [SOUND] in the file WIN.INI:

freq = 0x0000  (SystemDefault)
freq = 0x0010  (SystemHand)
freq = 0x0020  (SystemQuestion)
freq = 0x0030  (SystemExclamation)
freq = 0x0040  (SystemAsterisk)
freq = 0xFFFF  Standard Beep

Returns
None

Availability
This function is supported prior to Version 3.0.

Observation
If no sound card is installed, Windows will generate a normal system beep. In this case, the freq parameter has no effect.

Recommendation
This function has been replaced by the msgBeep() function.

Branch Compatibility
CANalyzer's Transmit Branch = Yes
CANalyzer's Analysis Branch = Yes
CANoe's Simulation Branch = Yes
CANoe's Analysis Branch = Yes

Related Functions
msgBeep

Example
void sound()
{
   //with soundcard: 400 Hz beep
   //without soundcard: standard system beep
   beep (400, 0);
}
The CAPL Functions

**callAllOnEnvVar**

**Syntax**

```cpp
void callAllOnEnvVar ();
```

**Description**

Calls all event procedures for environment variables to execute (On EnvVar events).

**Parameter**

None

**Returns**

None

**Availability**

Available in all versions.

**Observation**

This is usually done at the start of measurement to initialize environment variables, to start timers activated in response to changes of environment variables, or to send messages on the bus with the starting values of the environment variables.

**Branch Compatibility**

- CANalyzer's Transmit Branch = No
- CANalyzer's Analysis Branch = No
- CANoe's Simulation Branch = Yes
- CANoe's Analysis Branch = Yes

**Related Functions**

getValue
putValue

**Example**

```cpp
on start
{
  callAllOnEnvVar();
}
```
The CAPL Functions

cancelTimer

Syntax
void cancelTimer (msTimer t);
void cancelTimer (timer t);

Description
Stops a running timer that has been set with setTimer(). This prevents the timer event procedure from being executed.

Parameter
timer or msTimer variable

Returns
None

Availability
Available in all versions.

Observation
If a timer is no longer running or it has been expired, this function has no effect.

Branch Compatibility
CANalyzer's Transmit Branch = Yes
CANalyzer's Analysis Branch = Yes
CANoe's Simulation Branch = Yes
CANoe's Analysis Branch = Yes

Related Functions
setTimer

Example
variables
{
    msTimer msgTimer;
    message dataMsg dMsg;
}
on timer msgTimer
{
    output(dMsg);
    setTimer(msgTimer, 200);
}
on key F1
{
    cancelTimer(msgTimer);  //cancel timer
    write("msgTimer canceled");
}
on key F2
{
    setTimer(msgTimer, 200);  //set timer to 200 ms
    write("msgTimer started");
}
**canOffline**

**Syntax**

```plaintext
void canOffline();  //Form 1 obsolete
dword canOffline(dword flags);  //Form 2
```

**Description**

Cuts the connection between a simulated network node and the bus. Form 1 only has an effect on the CAPL program. In Form 2 you can choose between the CAPL program and/or the Node Layer DLL.

**Parameter**

- `flags = 1` (deactivates the CAPL program)
- `flags = 2` (deactivates the Node Layer DLL)
- `flags = 3` (deactivates both the CAPL program and the Node Layer DLL)

**Returns**

Form 2 returns the part of the node that was online before the function call. Equal to `flags`.

**Availability**

Available in all versions.

**Observation**

If this function is called in a CAPL program, that network node will not be able to transmit messages onto the bus. However, it is still capable of receiving messages from the bus and updating the variables in that program. To activate the network node again, call the `canOnline()` function.

**Recommendation**

In some applications, the offline approach may not be appropriate. A network node can be setup to start after a delay either within the CANoe tool or using the `setStartDelay()` function.

**Branch Compatibility**

- CANalyzer’s Transmit Branch = No
- CANalyzer’s Analysis Branch = No
- CANoe’s Simulation Branch = Yes
- CANoe’s Analysis Branch = No

**Related Functions**

canOnline

**Example**

```plaintext
dword var;

  //Deactivates CAPL program and Nodelayer DLL
  var = canOffline(3);
```

---

The CAPL Functions
canOnline

Syntax

void canOnline(); //Form 1 obsolete
dword canOnline(dword flags); //Form 2

Description

Restores the connection of the node to the bus. After a call to the function canOffline() the node can be connected to the bus with the function canOnline(). Messages sent from the node are passed through to the bus. Form 1 only has an effect on the CAPL program. In Form 2 you can choose between the CAPL program and/or the Node Layer DLL.

Parameter

flags = 1 activates the CAPL program
flags = 2 activates the Node Layer DLL for Network Management
flags = 3 activates both the CAPL program and the Node Layer

Returns

Form2 returns the part of the node that was online before the function call. Equal to flags.

Availability

Available in all versions.

Branch Compatibility

CANalyzer's Transmit Branch = No
CANalyzer's Analysis Branch = No
CANoe's Simulation Branch = Yes
CANoe's Analysis Branch = No

Related Functions

canOffline

decl

dword var;

... canOnline();  //activates CAPL program, Form 1
...
var = canOnline(2);  //activates Nodelayer DLL
canSetChannelAcc

Syntax
long canSetChannelAcc (long channel, dword code, dword mask);

Description
Sets an acceptance filter for a CAN controller. The SJA1000 chip used in all Vector CAN
interfaces expect the filter partition into acceptance mask and acceptance code. For extended or
29-bit messages, the most significant bit for the mask and code are set.

Parameter
channel = CAN channel
code = acceptance code for CAN ID filtering
mask = acceptance mask for CAN ID filtering

Returns
0 = successful
!0 = unsuccessful

Availability
This function is supported in Version 5.0 and after.

Observation
This function only works with Vector drivers. The vcdndrvms.dll must be at least Version 4.2.40.

Branch Compatibility
CANalyzer’s Transmit Branch = Yes
CANalyzer’s Analysis Branch = Yes
CANoe’s Simulation Branch = Yes
CANoe’s Analysis Branch = Yes

Related Functions
canSetChannelMode
canSetChannelOutput
resetCan
resetCanEx

Example
//To block all standard (11-bit) messages on channel 2
canSetChannelAcc(2, 0x7FF, 0x7FF);

//To block all extended (29-bit) messages on channel 2
canSetChannelAcc(2, 0x8FFFFFF, 0x8FFFFFF);

//To accept on message 0x100 on channel 1
canSetChannelAcc(1, 0x100, 0x100);
**canSetChannelMode**

**Syntax**
```
long canSetChannelMode (long channel, dword setTX, dword setTXRQ);
```

**Description**
Activates/deactivates both the transmit (TX) and transmit request (TXRQ) states of the CAN controller. The settings affect all the analysis windows for tracing, displaying, and logging.

**Parameter**
- **channel** = CAN channel
- **setTX** = 0 (off) = 1 (on)
- **setTXRQ** = 0 (off) = 1 (on)

**Returns**
- 0 = successful
- !0 = unsuccessful

**Availability**
This function is supported in Version 5.0 and after.

**Observation**
This function only works with Vector drivers. The vcdndrvms.dll must be at least Version 4.2.40.

**Branch Compatibility**
- CANalyzer’s Transmit Branch = Yes
- CANalyzer’s Analysis Branch = Yes
- CANoe’s Simulation Branch = Yes
- CANoe’s Analysis Branch = Yes

**Related Functions**
- canSetChannelAcc
- canSetChannelOutput
- resetCan
- resetCanEx

**Example**
```
//To deactivate both TX and TXRQ states on channel 2
canSetChannelMode(2, 0, 0);
```
canSetChannelOutput

**Syntax**

```c
long canSetChannelOutput (long channel, long silent);
```

**Description**

Activates/deactivates the acknowledgement of incoming messages for a channel. If in silent mode, the message is received on that channel but will not be acknowledged. That illustrates the spying functionality.

**Parameter**

- `channel = CAN channel`
- `silent = 0 (no acknowledgement)`
- `1 (acknowledge all received messages)`

**Returns**

- `0 = successful`
- `!0 = unsuccessful`

**Availability**

This function is supported in Version 5.0 and after.

**Observation**

This function only works with Vector drivers. The `vcdndrvms.dll` must be at least Version 4.2.40.

**Branch Compatibility**

- CANalyzer’s Transmit Branch = Yes
- CANalyzer’s Analysis Branch = Yes
- CANoe’s Simulation Branch = Yes
- CANoe’s Analysis Branch = Yes

**Related Functions**

- `canSetChannelMode`
- `canSetChannelOutput`
- `resetCan`
- `resetCanEx`

**Example**

```c
//To not acknowledge received messages on channel 2
canSetChannelOutput(2, 0);
```
**COS**

**Syntax**

double cos (double x);

**Description**

Calculates the cosine of x.

**Parameter**

Value (in radians) whose cosine is to be calculated. To convert degrees to radians, multiply degrees by PI/180.

**Returns**

Cosine of x

**Availability**

Available in all versions.

**Observation**

The “PI” is actually a keyword used in mathematical calculations.

**Branch Compatibility**

- CANalyzer’s Transmit Branch = Yes
- CANalyzer’s Analysis Branch = Yes
- CANoe’s Simulation Branch = Yes
- CANoe’s Analysis Branch = Yes

**Related Functions**

- exp
- sin
- sqrt

**Example**

```c
double x;

x = cos(PI);
//result: -1; PI (π) is a built-in constant:

//user-defined tangent function
double tangent(double x)
{
    return sin(x) / cos(x);
}
```
elCount

Syntax
long elCount (...);

Description
Determines the number of elements in one dimension of an array. See example for usage with multi-dimensional arrays.

Parameter
Array of any type

Returns
Number of elements in the array

Availability
Available in all versions.

Branch Compatibility
CANalyzer's Transmit Branch = Yes
CANalyzer's Analysis Branch = Yes
CANoe's Simulation Branch = Yes
CANoe's Analysis Branch = Yes

Related Functions
Strlen

Example
//One-dimensional array[i]
void bsp(int ar[])
{
    int i;
    for(i = 0; i < elCount(ar); i ++)
    ...
}

//Two-dimensional array[i][j]
void bsp2(byte ar[][[]])
{
    int i, j;
    for(j = 0; j < elCount(ar); j ++ )
        for(i = 0; i <= elCount(ar[j]); i ++ )
            ...
}
enableControl

Syntax
   void EnableControl (char panel[], char control[], long enable);

Description
   Activates/deactivates a control element on a panel.

Parameter
   panel = Name of the panel (w/o it, all opened panels will be affected)
   control = Name of the element (variable type is specified: EnVar or Signal)
   Ex:
       "EnVar:EnvGearLockDsp"
       "Signal:SleepInd"
       "ElemPanelHelp"      (for Panel help)
       "ElemPanelRecorder" (for Panel recorder)
       "ElemCtrlBN"        (for Panel control button)
   enable = 0 (disable) or 1 (enable)

Returns
   None

Availability
   This function is supported in Version 4.1 and after.

Observation
   If the control element is configured as a simple display, this command will have no effect on the element.
   Since no name is assigned to the Panel Recorder, the Panel Help or the Panel Control elements, only all or none of them can be activated in a given panel.
   The turned on or turned off state of an element remains intact at the start to the end of the measurement. Because of this, a defined state should be created for the beginning of the measurement for all the elements involved (e.g. within the Start event).

Branch Compatibility
   CANalyzer’s Transmit Branch = No
   CANalyzer’s Analysis Branch = No
   CANoe’s Simulation Branch = Yes
   CANoe’s Analysis Branch = Yes

Related Functions
   setControlProperty

Example
   //Activates Panel Help in the "gateway" panel
   enableControl("gateway", "ElemPanelHelp", 1);
Syntax

```c
double exp (double x);
```

Description
Calculates the value of the exponential function with a given degree.

Parameter
- `x` = value to calculate its exponent

Returns
- Exponent to base e

Availability
Available in all versions.

Branch Compatibility
- CANalyzer’s Transmit Branch = Yes
- CANalyzer’s Analysis Branch = Yes
- CANoe’s Simulation Branch = Yes
- CANoe’s Analysis Branch = Yes

Related Functions
- cos
- sin
- sqrt

Example
```c
double x;
    x = exp(1.0);  // e^1
    // Result: 2.7182...
```
fileClose

Syntax
long fileClose (dword fileHandle);

Description
Closes a specified file referenced by a file handle.

Parameter
fileHandle = value of the file handle

Returns
0 = unsuccessful
1 = successful

Availability
This function is supported in Version 3.0 and after.

Observation
The file handle was returned by the openFileRead() or openFileWrite() function.

Branch Compatibility
CANalyzer’s Transmit Branch = Yes
CANalyzer’s Analysis Branch = Yes
CANoe’s Simulation Branch = Yes
CANoe’s Analysis Branch = Yes

Related Functions
fileGetBinaryBlock
fileGetString
fileGetStringSZ
filePutString
fileRewind
fileWriteBinaryBlock
getProFileArray
getProFileFloat
getProFileInt
getProFileString
openFileRead
openFileWrite
setFilePath
setWritePath
writeProFileFloat
writeProFileInt
writeProFileString

Example
fileClose(glbHandle); //close file with the handle name “glbHandle”
...
fileGetBinaryBlock

Syntax

long fileGetBinaryBlock (byte buffer[], long buffsize, dword fileHandle);

Description

Reads characters from a file in binary format.

Parameter

- buffer = buffer to store the characters
- buffsize = maximum number of characters to get
- fileHandle = value of the file handle

Returns

Number of characters read

Availability

This function is supported in Version 3.0 and after.

Observation

The source file must be opened in binary format by the openFileRead() function.

Branch Compatibility

- CANalyzer’s Transmit Branch = Yes
- CANalyzer’s Analysis Branch = Yes
- CANoe’s Simulation Branch = Yes
- CANoe’s Analysis Branch = Yes

Related Functions

- fileClose
- fileGetString
- fileGetStringSZ
- filePutString
- fileRewind
- fileWriteBinaryBlock
- getProFileArray
- getProFileFloat
- getProFileInt
- getProFileString
- openFileRead
- openFileWrite
- setFilePath
- setWritePath
- writeProFileFloat
- writeProFileInt
- writeProFileString

Example

```c
if(fileGetBinaryBlock(buffer, elcount(buffer), glbHandle) == 0)
{
    write("End of file. File done.");
}
else
{
    //do something with the data in the buffer
}  ```
fileGetString

Syntax
long fileGetString (char buffer[], long buffsize, dword fileHandle);

Description
Reads a string from a file. The returned string contains a new line character.

Parameter
buffer = buffer to store the string of characters
buffsize = length of the string
fileHandle = value of the file handle

Returns
0 = unsuccessful
1 = successful

Availability
This function is supported in Version 3.0 and after.

Observation
Characters continue to be read until the end of line is reached or the number of read characters is equal to buffsize - 1.

Branch Compatibility
CANalyzer’s Transmit Branch = Yes
CANalyzer’s Analysis Branch = Yes
CANoe’s Simulation Branch = Yes
CANoe’s Analysis Branch = Yes

Related Functions
fileClose
fileGetBinaryBlock
fileGetStringSZ
filePutString
fileRewind
fileWriteBinaryBlock
getProFileArray
getProFileFloat
getProFileInt
getProFileString
openFileRead
openFileWrite
setFilePath
setWritePath
writeProFileFloat
writeProFileInt
writeProFileString
Example

```c
if(fileGetString(buffer, elcount(buffer), glbHandle) == 0)
{
    write("End of file. File done.");
}
else
{
    //do something with the data in the buffer
}
```
fileGetStringSZ

**Syntax**

```c
long fileGetStringSZ (char buffer[], long buffsize, dword fileHandle);
```

**Description**

Reads a string from a file. The new line character is not included in the string.

**Parameter**

- `buffer` = buffer to store the string of characters
- `buffsize` = length of the string
- `fileHandle` = value of the file handle

**Returns**

- 0 = unsuccessful
- 1 = successful

**Availability**

This function is supported in Version 3.0 and after.

**Observation**

Characters continue to be read until the end of line is reached or the number of read characters is equal to `buffsize - 1`.

**Branch Compatibility**

- CANalyzer’s Transmit Branch = Yes
- CANalyzer’s Analysis Branch = Yes
- CANoe’s Simulation Branch = Yes
- CANoe’s Analysis Branch = Yes

**Related Functions**

- `fileClose`
- `fileGetBinaryBlock`
- `fileGetString`
- `filePutString`
- `fileRewind`
- `fileWriteBinaryBlock`
- `getProFileArray`
- `getProFileFloat`
- `getProFileInt`
- `getProFileString`
- `openFileRead`
- `openFileWrite`
- `setFilePath`
- `setWritePath`
- `writeProFileFloat`
- `writeProFileInt`
- `writeProFileString`

**Example**

```c
if(fileGetString(buffer, elcount(buffer), glbHandle) == 0)
{
    write(“End of file. File done.”);
}
else
{
    //do something with the data in the buffer
}
```
The CAPL Functions

}


The CAPL Functions

**fileName**

**Syntax**

```c
void fileName ();
```

**Description**

Outputs the name of the CAPL program to the Write window.

**Parameter**

None.

**Returns**

None.

**Availability**

Available in all versions.

**Observation**

This function is helpful in debugging to determine which program is emulating.

**Branch Compatibility**

- CANalyzer’s Transmit Branch = Yes
- CANalyzer’s Analysis Branch = Yes
- CANoe’s Simulation Branch = Yes
- CANoe’s Analysis Branch = Yes

**Related Functions**

runError

**Example**

```c
fileName();
//Result: file name of current CAPL program in the Write window
```
filePutString

Syntax
long filePutString (char buffer[], long buffsize, dword fileHandle);

Description
Writes a string to a file.

Parameter
- buffer = the string of characters
- buffsize = number of characters to write
- fileHandle = value of the file handle

Returns
- 0 = unsuccessful
- 1 = successful

Availability
This function is supported in Version 3.0 and after.

Observation
The file handle is returned by the openFileWrite() function.

Branch Compatibility
- CANalyzer’s Transmit Branch = Yes
- CANalyzer’s Analysis Branch = Yes
- CANoe’s Simulation Branch = Yes
- CANoe’s Analysis Branch = Yes

Related Functions
- fileClose
- getFileBinaryBlock
- getString
- getStringSZ
- fileRewind
- fwriteBinaryBlock
- getProFileArray
- getProFileFloat
- getProFileInt
- getProFileString
- openFileRead
- openFileWrite
- setFilePath
- setWritePath
- writeProFileFloat
- writeProFileInt
- writeProFileString

Example
on key ‘t’
{
    buffer = random(101);
    filePutString(buffer, elcount(buffer), glbHandle);
    ...
}
**fileReadArray**

**Syntax**

```c
long fileReadArray (char section[], char entry[], char buffer[], long bufferlen, char file[]);
```

**Description**

Reads an array of byte values from an INI-formatted file. The values can be decimal or hexadecimal with the "0x" prefix. Values can be separated by spaces, tabs, commas, semicolons, or slashes.

**Parameter**

- section = section within file
- entry = name of variable
- buffer = buffer for characters to be read
- bufferlen = size of buffer in bytes
- file = name of data file (backslashes should be doubled, i.e. “C:\\TEMP\\DATA.LOG”)

**Returns**

Number of characters read.

**Availability**

This function is supported prior to Version 3.0.

**Observation**

This function is equivalent to the fileReadString() function. To write an array to an INI-formatted file, use the fileWriteString() function.

**Recommendation**

This function has been replaced by the getProFileArray() function.

**Branch Compatibility**

- CANalyzer’s Transmit Branch = Yes
- CANalyzer’s Analysis Branch = Yes
- CANoe’s Simulation Branch = Yes
- CANoe’s Analysis Branch = Yes

**Related Functions**

- fileReadFloat
- fileReadInt
- fileReadString
- fileWriteFloat
- fileWriteInt
- fileWriteString
Example

//Data in TEST.INI:
...
[DATA]
FIELD = 1,2,3,0x20,100
...

//Code Example:

int len;
char buffer[20];
len = fileReadArray("DATA", "FIELD", buffer, elCount(buffer), "TEST.INI");
...

//Result: len = 5. The array buffer is filled with the values- 1,2,3,32,100.
The CAPL Functions

fileReadFloat

Syntax

float fileReadFloat (char section[], char entry[], float def, char file[]);

Description

Reads a float value from an INI-formatted file.

Parameter

section = section within file
entry = name of variable
def = default return value in case of error
file = name of data file (backslashes should be doubled, i.e. “C:\\TEMP\\DATA.LOG”)

Returns

Valid float value or the default value

Availability

This function is supported prior to Version 3.0.

Observation

The value is only returned if it is found and valid, else the default value is returned as the functional result.

Recommendation

This function has been replaced by the getProFileFloat() function.

Branch Compatibility

CANalyzer’s Transmit Branch = Yes
CANalyzer’s Analysis Branch = Yes
CANoe’s Simulation Branch = Yes
CANoe’s Analysis Branch = Yes

Related Functions

fileReadArray
fileReadInt
fileReadString
fileWriteFloat
fileWriteInt
fileWriteString

Example

//Data in TEST.INI:
...
 [DATA]
 VOLUME = 3.3
 ...

//Code Example:

float vol;
vol = fileReadFloat(“DATA”, “VOLUME”, 0, “TEST.INI”);
...
// Result: vol = 3.3
FileReadInt

**Syntax**

```c
long fileReadInt (char section[], char entry[], long def, char file[]);
```

**Description**

Reads an integer value from an INI-formatted file.

**Parameter**

- `section` = section within file
- `entry` = name of variable
- `def` = default return value in case of error
- `file` = name of data file (backslashes should be doubled, i.e. “C:\\TEMP\\DATA.LOG”)

**Returns**

Valid integer value or the default value

**Availability**

This function is supported prior to Version 3.0.

**Observation**

The value is only returned if it is found and valid, else the default value is returned as the functional result.

**Recommendation**

This function has been replaced by the `getProfileInt()` function.

**Branch Compatibility**

- CANalyzer’s Transmit Branch = Yes
- CANalyzer’s Analysis Branch = Yes
- CANoe’s Simulation Branch = Yes
- CANoe’s Analysis Branch = Yes

**Related Functions**

- `fileReadArray`
- `fileReadFloat`
- `fileReadString`
- `fileWriteFloat`
- `fileWriteInt`
- `fileWriteString`

**Example**

```c
//Data in TEST.INI:
...
[DATA]
ADDR = 200
...

//Code Example:

int myAddress;
myAddress = fileReadInt(“DATA”, “ADDR”, 0, “TEST.INI”);
...

//Result: myAddress = 200
```
FileReadString

Syntax
long fileReadString (char section[], char entry[], char def[], char buffer[], long bufferlen, char filename[]);

Description
Reads a string value from an INI-formatted file.

Parameter
section = section within file
entry = name of variable
def = default return value in case of error
buffer = buffer for characters to be read
bufferlen = size of buffer in bytes
file = name of data file (backslashes should be doubled, i.e. “C:\\TEMP\\DATA.LOG”)

Returns
Number of bytes read

Availability
This function is supported prior to Version 3.0.

Observation
The value is only returned if it is found and valid, else the default value is returned as the functional result.

Recommendation
This function has been replaced by the getProFileString() function.

Branch Compatibility
CANalyzer’s Transmit Branch = Yes
CANalyzer’s Analysis Branch = Yes
CANoe’s Simulation Branch = Yes
CANoe’s Analysis Branch = Yes

Related Functions
fileReadArray
fileReadFloat
fileReadInt
fileWriteFloat
fileWriteInt
fileWriteString
Example

//Data in TEST.INI:
...
[DATA]
NAME = Marty
...

//Code Example:

int len;
char def[6] = "error";
char buffer[20];
len = fileReadString("DATA", "NAME", def, buffer, elCount(buffer), "TEST.INI");
...
//Result: buffer = "Marty"
fileRewind

Syntax
long fileRewind (dword fileHandle);

Description
Resets the position pointer to the beginning of the file.

Parameter
fileHandle = value of the file handle

Returns
0 = unsuccessful
1 = successful

Availability
This function is supported in Version 3.0 and after.

Branch Compatibility
CANalyzer’s Transmit Branch = Yes
CANalyzer’s Analysis Branch = Yes
CANoe’s Simulation Branch = Yes
CANoe’s Analysis Branch = Yes

Related Functions
fileClose
fileGetBinaryBlock
fileGetString
fileGetStringSZ
filePutString
fileWriteBinaryBlock
getProFileArray
getProFileFloat
getProFileInt
getProFileString
openFileRead
openFileWrite
setFilePath
setWritePath
writeProFileFloat
writeProFileInt
writeProFileString

Example
if (fileRewind(glbHandle))
{
    ...  //do something after file is rewound
}
fileWriteBinaryBlock

**Syntax**

```c
long fileWriteBinaryBlock (byte buffer[], long buffsize, dword fileHandle);
```

**Description**

Writes characters to a file in binary format. The source file must be opened in binary format.

**Parameter**

- `buffer` = the block of characters to write
- `buffsize` = maximum number of characters to write
- `fileHandle` = value of the file handle

**Returns**

Number of characters written

**Availability**

This function is supported in Version 3.0 and after.

**Observation**

The file handle is returned by the `setWritePath()` function opened in binary format.

**Branch Compatibility**

- CANalyzer’s Transmit Branch = Yes
- CANalyzer’s Analysis Branch = Yes
- CANoe’s Simulation Branch = Yes
- CANoe’s Analysis Branch = Yes

**Related Functions**

- fileClose
- fileGetBinaryBlock
- fileGetString
- fileGetStringSZ
- filePutString
- fileRewind
- getProFileArray
- getProFileFloat
- getProFileInt
- getProFileString
- openFileRead
- openFileWrite
- setFilePath
- setWritePath
- writeProFileFloat
- writeProFileInt
- writeProFileString
Example on message 0x100
{
    // transmit message data into buffer to put into a file
    for(i = 0; i < elcount(this); i++)
    {
        buffer[i] = this.byte(i);
    }
    fileWriteBinaryBlock(buffer, elcount(buffer), glbHandle);
    ...
}
fileWriteFloat

Syntax
long fileWriteFloat (char section[], char entry[], float def, char file[]);

Description
Writes a float value to an INI-formatted file.

Parameter
- section = section within file
- entry = name of variable
- def = float value to write
- file = name of file (backslashes should be doubled, i.e. “C:\ TEMP \ DATA.LOG”)

Returns
- 0 = unsuccessful
- 1 = successful

Availability
This function is supported prior to Version 3.0.

Observation
Any existing value in the INI entry will be overwritten.

Recommendation
This function has been replaced by the writeProFileFloat() function.

Branch Compatibility
- CANalyzer’s Transmit Branch = Yes
- CANalyzer’s Analysis Branch = Yes
- CANoe’s Simulation Branch = Yes
- CANoe’s Analysis Branch = Yes

Related Functions
- fileReadArray
- fileReadFloat
- fileReadInt
- fileReadString
- fileWriteInt
- fileWriteString

Example
if(!fileWriteFloat(“DeviceData”, “DeviceAddr”, 2.2, “TEST.INI”))
    write(“Error writing DeviceAddr to TEST.INI”);
...  
//This call writes the following entry if successful:
[DeviceData]
DeviceAddr = 2.2
**fileWriteInt**

**Syntax**

```c
long fileWriteInt (char section[], char entry[], long def, char file[]);
```

**Description**

Writes an integer value to an INI-formatted file.

**Parameter**

- section = section within file
- entry = name of variable
- def = integer value to write
- file = name of file (backslashes should be doubled, i.e. "C: \ TEMP \ DATA.LOG")

**Returns**

- 0 = unsuccessful
- 1 = successful

**Availability**

This function is supported prior to Version 3.0.

**Observation**

Any existing value in the INI entry will be overwritten.

**Recommendation**

This function has been replaced by the writeProFileInt() function.

**Branch Compatibility**

- CANalyzer’s Transmit Branch = Yes
- CANalyzer’s Analysis Branch = Yes
- CANoe’s Simulation Branch = Yes
- CANoe’s Analysis Branch = Yes

**Related Functions**

- fileReadArray
- fileReadFloat
- fileReadInt
- fileReadString
- fileWriteFloat
- fileWriteString

**Example**

```c
if(!fileWriteInt(“DeviceData”, “DeviceAddr”, 2, “TEST.INI”))
    write(“Error writing DeviceAddr to TEST.INI”);
...  
//This call writes the following entry if successful:
[DeviceData]
DeviceAddr = 2
```
fileWriteString

Syntax
long fileWriteString (char section[], char entry[], char value[], char filename[]);

Description
Writes a string value to an INI-formatted file.

Parameter
section = section within file
entry = name of variable
def = string value to write
file = name of file (backslashes should be doubled, i.e. “C: \ TEMP \ DATA.LOG”)

Returns
0 = unsuccessful
!0 = number of characters written

Availability
This function is supported prior to Version 3.0.

Observation
Any existing value in the INI entry will be overwritten.

Recommendation
This function has been replaced by the writeProFileString() function.

Branch Compatibility
CANalyzer’s Transmit Branch = Yes
CANalyzer’s Analysis Branch = Yes
CANoe’s Simulation Branch = Yes
CANoe’s Analysis Branch = Yes

Related Functions
fileReadArray
fileReadFloat
fileReadInt
fileReadString
fileWriteFloat
fileWriteInt

Example
if(!fileWriteString(“Device”, “DeviceName”, “ABS”, “TEST.INI”))
  write(“Error writing DeviceAddr to TEST.INI”);
...

//This call writes the following entry if successful:
[Device]
DeviceName = ABS
The CAPL Functions

**getBusContext**

**Syntax**

```c
    dword getBusContext () ;
```

**Description**

Gets the current bus context of the network node (Gateway).

**Parameter**

None

**Returns**

Bus context of the current network node (Gateway)

**Availability**

This function is supported in Version 3.2 and after.

**Observation**

The bus context plays a role exclusively in modeling gateways. In this case, a series of CAPL functions such as `canOnline()` and `canOffline()` may have more than one meaning in terms of the bus interface (channel) to be used. A similar type of problem occurs when identical node layer modules are used simultaneously within a CAPL block. A distinction must be made between the instances of the node layer, both for calls to CAPL functions that are implemented in the node layers and for implementing callbacks.

To facilitate this distinction, a bus context is placed in the CAPL program by the runtime environment while the node layer is executing a callback. This context unambiguously identifies the node layer that is making the call. In a similar manner, the call of a CAPL function that is implemented in a node layer is forwarded on to the appropriate node layer, depending on the current bus context. This also applies to the CAPL functions `canOnline()` and `canOffline()`.

**Branch Compatibility**

- CANalyzer’s Transmit Branch = No
- CANalyzer’s Analysis Branch = No
- CANoe’s Simulation Branch = Yes
- CANoe’s Analysis Branch = No

**Related Functions**

- `getBusNameContext`
- `setBusContext`

**Example**

```c
    dword contextValue;
    contextValue = getBusContext();
```
**getBusNameContext**

**Syntax**

dword getBusNameContext (char name[]);

**Description**

Gets the bus context of the bus given by its name.

**Parameter**

name = the name of the bus

**Returns**

0 = bus not exist

unsigned value = bus context given by the name of the bus

**Availability**

This function is supported in Version 3.2 and after.

**Observation**

The bus context plays a role exclusively in modeling gateways. In this case, a series of CAPL functions such as canOnline() and canOffline() may have more than one meaning in terms of the bus interface (channel) to be used. A similar type of problem occurs when identical node layer modules are used simultaneously within a CAPL block. A distinction must be made between the instances of the node layer, both for calls to CAPL functions that are implemented in the node layers and for implementing callbacks.

To facilitate this distinction, a bus context is placed in the CAPL program by the runtime environment while the node layer is executing a callback. This context unambiguously identifies the node layer that is making the call. In a similar manner, the call of a CAPL function that is implemented in a node layer is forwarded on to the appropriate node layer, depending on the current bus context. This also applies to the CAPL functions canOnline() and canOffline().

**Branch Compatibility**

CANalyzer’s Transmit Branch = No
CANalyzer’s Analysis Branch = No
CANoe’s Simulation Branch = Yes
CANoe’s Analysis Branch = No

**Related Functions**

getBusContext
setBusContext

**Example**

dword contextValue;
contextValue = getBusNameContext("Motbus");
getCardType

Syntax
long getCardType ();

Description
Returns the type of CAN platform being used.

Parameter
None

Returns
Type of interface:
0 = DBB196 - Daimler-Benz Board with Full CAN
1 = DBB196B - Daimler-Benz Board with Basic CAN
2 = CANIB - Bosch CANIB
3 = DEMO - Demo driver
6 = CANAC2 - Softing AC2/200/ANA
7 = CANAC2X - Softing AC2/527/ANA
8 = CPC/PP - EMS wish module
9 = INDIGO - Silicon Graphics Indigo2
10 = CANCARD - PCMCIA 11 Bit
12 = CANAC2B - Softing AC2/527 11 Bit
13 = VAN462 – NSI VAN card
14 = VANDEMO – VAN Demo driver
15 = Peak CAN-Dongle
16 = Vector CAN-Dongle
17 = Vector PCMCIA CANcardX

Availability
Available in all versions.

Observation
This function is needed, for example, to program the BTR (Bit Timing Register) and OCR (Output Control Register) values.

Branch Compatibility
CANalyzer’s Transmit Branch = Yes
CANalyzer’s Analysis Branch = Yes
CANoe’s Simulation Branch = Yes
CANoe’s Analysis Branch = Yes

Related Functions
getCardTypeEx
getChipType

Example
switch(getCardType())
{
    case 6: setOcr(0,0x02); //CANAC2
        break;
    ...  
    default: write(“Unknown driver %d”, getCardType());
        break;
}
**getCardTypeEx**

**Syntax**

```c
int getCardTypeEx (int channel);
```

**Description**

Returns the type of CAN platform being used on a specific CAN channel.

**Parameter**

channel = channel number

**Returns**

Type of interface:
- 0 = DBB196 - Daimler-Benz Board with Full CAN
- 1 = DBB196B - Daimler-Benz Board with Basic CAN
- 2 = CANIB - Bosch CANIB
- 3 = DEMO - Demo driver
- 6 = CANAC2 - Softing AC2/200/ANA
- 7 = CANAC2X - Softing AC2/527/ANA
- 8 = CPC/PP - EMS wish module
- 9 = INDIGO - Silicon Graphics Indigo2
- 10 = CANCARD - PCMCIA 11 Bit
- 12 = CANAC2B - Softing AC2/527 11 Bit
- 13 = VAN462 – NSI VAN card
- 14 = VANDemo – VAN Demo driver
- 15 = Peak CAN-Dongle
- 16 = Vector CAN-Dongle
- 17 = Vector PCMCIA CANcardX
- 20 = Softing PCMCIA CANcard SJA1000
- 25 = Vector PCMCIA CANcardXL
- 27 = Vector USB CANcase
- 29 = Vector PCI CANboard
- 30 = Vector PCI CANboard for Compact PCI

**Availability**

This function is supported in Version 5.0 and after.

**Observation**

This function is needed, for example, to program the BTR (Bit Timing Register) and OCR (Output Control Register) values.

**Branch Compatibility**

- CANalyzer’s Transmit Branch = Yes
- CANalyzer’s Analysis Branch = Yes
- CANoe’s Simulation Branch = Yes
- CANoe’s Analysis Branch = Yes

**Related Functions**

- getCardType
- getChipType
Example

switch(getCardTypeEx(1))
{
    case 6: setOcr(0,0x02); //CANAC2
        break;

    default: write("Unknown driver %d", getCardTypeEx(1));
        break;
}

The CAPL Functions

getChipType

Syntax

long getChipType (long channel);

Description

Returns the type of CAN controller being used.

Parameter

channel = 0 (both channels)
   = 1 (channel 1)
   = 2 (channel 2)

Returns

Type of controller with the following values:

5   NEC 72005
200  Philips PCA82C200
462  MHS29C462 VAN Controller
526  Intel 82526
527  Intel 82527
1000, 1001 Philips SJA1000

Availability

Available in all versions.

Observation

This function may return other types of controller. Demo tool versions return the result 0 or simulate one of the existing types. If an attempt is made to access a nonexistent channel (e.g. Channel 2 for CPC/PP) or if the driver used does not support this function, the functional result is 0.

Branch Compatibility

CANalyzer’s Transmit Branch = Yes
CANalyzer’s Analysis Branch = Yes
CANoe’s Simulation Branch = Yes
CANoe’s Analysis Branch = Yes

Related Functions

getchipType
getchipTypeEx

Example

switch(getChipType())
{
   case 200: setOcr(0,0x02); //Philips PCA82C200
             break;
   ...
The CAPL Functions

getDrift

**Syntax**
```cpp
int getDrift ();
```

**Description**
Determines the constant deviation after drift is set.

**Parameter**
None

**Returns**
The drift in parts per thousand

**Availability**
This function is supported in Version 3.0 and after.

**Branch Compatibility**
- CANalyzer’s Transmit Branch = No
- CANalyzer’s Analysis Branch = No
- CANoe’s Simulation Branch = Yes
- CANoe’s Analysis Branch = No

**Related Functions**
- getJitterMax
- getJitterMin
- setDrift
- setJitter

**Example**
```cpp
int val;

// Assign the drift value to val
val = getDrift();
...
```
getFirstCANdbName

**Syntax**

dword getFirstCANdbName (char buffer[], dword size);

**Description**

Finds the name of the first assigned database.

**Parameter**

- buffer = symbolic name of database
- size = buffer size

**Returns**

- 0 = unsuccessful
- !0 = successful

**Availability**

This function is supported in Version 4.0 and after.

**Branch Compatibility**

- CANalyzer’s Transmit Branch = No
- CANalyzer’s Analysis Branch = Yes
- CANoe’s Simulation Branch = No
- CANoe’s Analysis Branch = Yes

**Related Functions**

- getMessageAttrInt
- getMessageName
- getNextCANdbName

**Example**

```c
char buffer[256];
dword pos;

pos = getFirstCANdbName(buffer, elcount(buffer));
write("Name = \%s", buffer);
```
**getJitterMax**

**Syntax**
```c
int getJitterMax ();
```

**Description**
Determines the upper deviation limit allowed when jitter is set.

**Parameter**
None

**Returns**
Upper deviation in parts per thousand

**Availability**
This function is supported in Version 3.0 and after.

**Branch Compatibility**
- CANalyzer’s Transmit Branch = No
- CANalyzer’s Analysis Branch = No
- CANoe’s Simulation Branch = Yes
- CANoe’s Analysis Branch = No

**Related Functions**
- getDrift
- getJitterMax
- getJitterMin
- setDrift
- setJitter

**Example**
```c
int val;
...
//Assign the upper value of the jitter to val
val = getJitterMax();
...
```
getJitterMin

Syntax

int getJitterMin ();

Description

Determines the lower deviation limit allowed when jitter is set.

Parameter

None

Returns

Lower deviation in parts per thousand

Availability

This function is supported in Version 3.0 and after.

Branch Compatibility

CANalyzer's Transmit Branch = No
CANalyzer's Analysis Branch = No
CANoe's Simulation Branch = Yes
CANoe's Analysis Branch = No

Related Functions

getDrift
getJitterMax
setDrift
setJitter

Example

int val;

//Assign the lower value of the jitter to val
val = getJitterMin();
...


getLocalTime

Syntax
void getLocalTime (long timeArray[]);

Description
Fills an array with details of the date and time.

Parameter
timeArray = array of type long with at least 9 entries

The entries of the array will be filled with the following information:

- timeArray[0] = Seconds (0 – 59)
- timeArray[1] = Minutes (0 – 59)
- timeArray[2] = Hours (0 – 23)
- timeArray[4] = Month of year (0 – 11)
- timeArray[5] = Year (since 1900)
- timeArray[6] = Day of week (0 – 6)
- timeArray[7] = Day of year (0 – 365)
- timeArray[8] = Daylight Savings Time (0 = not)

Returns
None

Availability
Available in all versions.

Branch Compatibility
- CANalyzer’s Transmit Branch = No
- CANalyzer’s Analysis Branch = Yes
- CANoe’s Simulation Branch = No
- CANoe’s Analysis Branch = Yes

Related Functions
getLocalTimeString
timeDiff
timeNow

Example
long timeArray[9];

getLocalTime(timeArray);

write("It is %d:%d:%d on %d/%d/%d.", timeArray[2], timeArray[1],
    timeArray[0], timeArray[4] + 1, timeArray[3], timeArray[5]);

//Result: It is 16:23:31 on 8/25/03.
getLocalTimeString

Syntax
void getLocalTimeString (char timeBuffer[]);

Description
Fills a string with details of the date and time. The format of the string is ddd mmm dd hh:mm:ss yyyy (e.g., "Fri Aug 21 15:22:24 1998").

Parameter
timeBuffer = date and time string (must be at least 26 characters long)

Returns
None

Availability
Available in all versions.

Observation
The time string is null-terminated.

Branch Compatibility
CANalyzer's Transmit Branch = No
CANalyzer's Analysis Branch = Yes
CANoe's Simulation Branch = No
CANoe's Analysis Branch = Yes

Related Functions
getLocalTime
timeDiff
timeNow

Example
char timeBuffer[32];

getLocalTimeString(timeBuffer);

//The timeBuffer will now contain, e.g., Fri Aug 21 15:22:24 2004
The CAPL Functions

**getMessageAttrInt**

**Syntax**

```c
long getMessageAttrInt (message Msg, char attributeName[]);
long getMessageAttrInt (pg parameterGroup, char attributeName[]);
```

**Description**

Returns the message attribute value from the CANdb database.

**Parameter**

- `Msg` = message variable
- `attributeName` = name of attribute

**Returns**

- `0` = attribute not found
- `value` = successful
- `default attribute value` = message attribute value not assigned

**Availability**

This function is supported in Version 3.1 and after.

**Observation**

The attribute must be of type integer. The attribute should be found directly by its selector syntax (`<Message variable>.<Attribute name>` e.g. `ABSdata.msgCycleTime`). The advantage to call this function instead of using the selector approach is any changes made to the attribute in the database while CANalyzer/CANoe’s measurement is running is updated to the new attribute value.

**Branch Compatibility**

- CANalyzer’s Transmit Branch = Yes
- CANalyzer’s Analysis Branch = Yes
- CANoe’s Simulation Branch = Yes
- CANoe’s Analysis Branch = Yes

**Related Functions**

- `getFirstCANdbName`
- `getMessageName`
- `getNextCANdbName`

**Example**

```c
on message *
{
    long cycleTimeValue;
    cycleTimeValue = getMessageAttrInt(this, “cycleTime”);
    write(“CycleTime of message id%x = %d”, this.Id, cycleTimeValue);
}
```
getMessagingName

Syntax

dword getMessageName (dword id, dword context, char buffer[], dword size);

Description

Returns the message symbolic name from the database.

Parameter

id = message identifier
context = bus type
buffer = message symbolic value
size = number of symbolic characters to get

The context can have any of these values:
  = 0x00010000 (CAN bus)
  = 0x00050000 (LIN bus)
  = 0x00060000 (MOST bus)
  = 0x00070000 (FlexRay bus)
  = 0x00080000 (BEAN bus)

Returns

0 = unsuccessful
!0 = successful

Availability

This function is supported in Version 4.0 and after.

Observation

This is a great way to check if a message is predefined in the database.

Branch Compatibility

CANalyzer’s Transmit Branch = No
CANalyzer’s Analysis Branch = Yes
CANoe’s Simulation Branch = No
CANoe’s Analysis Branch = Yes

Related Functions

getFirstCANdbName
getMessageAttrInt
getNextCANdbName

Example

on message *
{
  dword contextCAN = 0x00010000;
  char buffer[64];
  if(getMessageName(this.ID, contextCAN | this.CAN, buffer, elcount(buffer)))
  {
    write("Message ID%d = %s", this.id, buffer);
  }
  output(this);
}
The CAPL Functions

getNextCANdbName

dword getNextCANdbName (dword pos, char buffer[], dword size);

Description
Finds the name of the first assigned database.

Parameter
buffer = stores the symbolic name of database
size = buffer size

Returns
0 = unsuccessful
!0 = successful

Availability
This function is supported in Version 4.0 and after.

Branch Compatibility
CANalyzer’s Transmit Branch = No
CANalyzer’s Analysis Branch = Yes
CANoe’s Simulation Branch = No
CANoe’s Analysis Branch = Yes

Related Functions
getFirstCANdbName
getMessageAttrInt
getMessageName

Example
char buffer[256];
dword pos;

pos = getNextCANdbName(1, buffer, elcount(buffer));
write("Name = %s pos %d", buffer, pos);
getProFileArray

Syntax
    long getProFileArray (char section[], char entry[], char buffer[], long buffsize, char filename[]);

Description
    Reads an array of byte values from an INI-formatted file.

Parameter
    section = section within file
    entry = name of variable
    buffer = buffer for bytes to be read
    buffsize = size of buffer in bytes
    filename = name of data file

Returns
    Number of bytes read

Availability
    This function is supported in Version 3.0 and after.

Observation
    The values can be decimal or hexadecimal with the "0x" prefix. Values can be separated by spaces, tabs, commas, semicolons, or slashes. The file path is set by either the setWritePath() or setFilePath() function. If neither function is used, the data file must be located either in the same directory as the databases file(s) or configuration file(s) of CANalyzer/CANoe.

Branch Compatibility
    CANalyzer's Transmit Branch = Yes
    CANalyzer's Analysis Branch = Yes
    CANoe's Simulation Branch = Yes
    CANoe's Analysis Branch = Yes

Related Functions
    fileClose
    fileGetBinaryBlock
    fileGetString
    fileGetStringSZ
    filePutString
    fileRewind
    fileWriteBinaryBlock
    getProFileFloat
    getProFileInt
    getProFileString
    openFileRead
    openFileWrite
    setFilePath
    setWritePath
    writeProFileFloat
    writeProFileInt
    writeProFileString
Example

//Data in TEST.INI:
...
[data]
FIELD = 1,2,3,0x20,100
...

//Code Example:

int len;
char buffer[20];
len = getProfileArray("DATA", "FIELD", buffer, elCount(buffer),
"TEST.INI");
...

//Result: len = 5. The array buffer is filled with the values-1,2,3,32,100.
getProFileFloat

Syntax
long getProFileFloat (char section[], char entry[], long def, char filename[]);

Description
Reads a float value from an INI-formatted file.

Parameter
- section = section within file
- entry = name of variable
- def = default return value in case of error
- filename = name of data file

Returns
Valid float value or the default value

Availability
This function is supported in Version 3.0 and after.

Observation
The value is only returned if it is found and valid, else the default value is returned as the functional result. The file path is set by either the setWritePath() or setFilePath() function. If neither function is used, the data file must be located either in the same directory as the databases file(s) or configuration file(s) of CANalyzer/CANoe.

Branch Compatibility
- CANalyzer’s Transmit Branch = Yes
- CANalyzer’s Analysis Branch = Yes
- CANoe’s Simulation Branch = Yes
- CANoe’s Analysis Branch = Yes

Related Functions
- fileClose
- fileGetBinaryBlock
- fileGetString
- fileGetStringSZ
- filePutString
- fileRewind
- fileWriteBinaryBlock
- getProFileArray
- getProFileInt
- getProFileString
- openFileRead
- openFileWrite
- setFilePath
- setWritePath
- writeProFileFloat
- writeProFileInt
- writeProFileString
Example

// Data in TEST.INI:
...
[DATA]
VOLUME = 3.3
...

// Code Example:

float vol;
vol = getProfileFloat("DATA", "VOLUME", 0, "TEST.INI");

// Result: vol = 3.3
getProFileInt

Syntax
long getProFileInt (char section[], char entry[], long def, char filename[]);

Description
Reads an integer value from an INI-formatted file.

Parameter
section = section within file
entry = name of variable
def = default return value in case of error
filename = name of data file

Returns
Valid integer value or the default value

Availability
This function is supported in Version 3.0 and after.

Observation
The value is only returned if it is found and valid. Else the default value is returned as the functional result. The file path is set by either the setWritePath() or setFilePath() function. If neither function is used, the data file must be located either in the same directory as the databases file(s) or configuration file(s) of CANalyzer/CANoe.

Branch Compatibility
CANalyzer’s Transmit Branch = Yes
CANalyzer’s Analysis Branch = Yes
CANoe’s Simulation Branch = Yes
CANoe’s Analysis Branch = Yes

Related Functions
fileClose
fileGetBinaryBlock
fileGetString
fileGetStringSZ
filePutString
fileRewind
fileWriteBinaryBlock
getProFileArray
getProFileFloat
getProFileString
openFileRead
openFileWrite
setFilePath
setWritePath
writeProFileFloat
writeProFileInt
writeProFileString
Example

// Data in TEST.INI:
...
[DATA]
ADDR = 200
...

// Code Example:

int myAddress;
myAddress = getProFileInt("DATA","ADDR",0, "TEST.INI");
...

// Result: myAddress = 200
The CAPL Functions

**getProFileString**

**Syntax**

```c
long getProFileString (char section[], char entry[], char def[], char buffer[], long buffsize, char filename[]);
```

**Description**

Reads a string value from an INI-formatted file.

**Parameter**

- `section` = section within file
- `entry` = name of variable
- `def` = default return value in case of error
- `buffer` = buffer for characters to be read
- `buffsize` = size of buffer in bytes
- `filename` = name of data file

**Returns**

Number of bytes read

**Availability**

This function is supported in Version 3.0 and after.

**Observation**

The value is only returned if it is found and valid. Else the default value is returned as the functional result. The file path is set by either the `setWritePath()` or `setFilePath()` function. If neither function is used, the data file must be located either in the same directory as the databases file(s) or configuration file(s) of CANalyzer/CANoe.

**Branch Compatibility**

- CANalyzer’s Transmit Branch = Yes
- CANalyzer’s Analysis Branch = Yes
- CANoe’s Simulation Branch = Yes
- CANoe’s Analysis Branch = Yes

**Related Functions**

- fileClose
- fileGetBinaryBlock
- fileGetString
- fileGetStringSZ
- filePutString
- fileRewind
- fileWriteBinaryBlock
- getProFileArray
- getProFileFloat
- getProFileInt
- openFileRead
- openFileWrite
- setFilePath
- setWritePath
- writeProFileFloat
- writeProFileInt
- writeProFileString
Example

// Data in TEST.INI:
...
[DATA]
NAME = Marty
...

// Code Example:

int len;
char def[6] = "error";
char buffer[20];
len = getProfileString("DATA", "NAME", def, buffer, elCount(buffer), "TEST.INI");
...
// Result: buffer = "Marty"
getStartdelay

Syntax
   int getStartdelay ();

Description
   Determines the delay time value configured for a network node in the Simulation Setup window.

Parameter
   None

Returns
   0 = delay not set
   !0 = delay time value

Availability
   This function is supported in Version 3.0 and after.

Branch Compatibility
   CANalyzer's Transmit Branch = No
   CANalyzer's Analysis Branch = No
   CANoe's Simulation Branch = Yes
   CANoe's Analysis Branch = No

Related Functions
   setStartdelay

Example
   int val;

   //Assign the value of the start delay to val
   val = getStartdelay();
getValue

Syntax
int getValue (EnvVarName);  //Form 1
float getValue (EnvVarName);  //Form 2
long getValue (EnvVarName, char buffer[]);  //Form 3
long getValue (EnvVarName, byte buffer[]);  //Form 4
long getValue (EnvVarName, byte buffer[], long offset);  //Form 5

Description
Returns the value of an environment variable. Return value type is based on the type of environment variable. For character array or string environment variables (Form 3) the active value is saved to a buffer.

Parameter
EnvVarName = environment variable name
buffer = environment variable value
offset = starting position (byte)

Returns
Environment variable value for Forms 1 and 2
Number of bytes copied for Form 3, 4, and 5

Availability
Available in all versions.

Branch Compatibility
CANalyzer’s Transmit Branch = No
CANalyzer’s Analysis Branch = No
CANoe’s Simulation Branch = Yes
CANoe’s Analysis Branch = Yes

Related Functions
callAllOnEnvVar
gGetValueSize
putValue
putValueToControl

Example
int val;
float fval;
char buff[25];

//Assign to val the value of the environment variable “Switch”
val = getValue(Switch);

//Assign to fval the value of the environment variable “Temperature”
fval = getValue(Temperature);

//Read the value of environment variable “NodeName”
val = getValue(NodeName, buff);
getValueSize

**Syntax**
```c
int getValueSize (EnvVarName);
```

**Description**
Returns the size of an environment variable in bytes.

**Parameter**
EnvVarName = environment variable name

**Returns**
Number of bytes

**Availability**
Available in all versions.

**Observation**
For environment variables of type string, the string length plus the terminating null character will be returned.

**Branch Compatibility**
- CANalyzer's Transmit Branch = No
- CANalyzer's Analysis Branch = No
- CANoe's Simulation Branch = Yes
- CANoe's Analysis Branch = Yes

**Related Functions**
- getValue
- putValue
- putValueToControl

**Example**
```c
// Size of an environment variable of data type integer:
int varSize;
...
varSize = getValueSize(switch);
```
The CAPL Functions

halt

Syntax
void halt ();

Description
Halts the execution of the simulation. The simulation is resume with the <F9> key. The halt instruction is ignored in Real mode.

Parameter
None

Returns
None

Availability
This function is supported in Version 4.1 and after.

Observation
This function is only effective if CANoe is in the Simulated mode instead of the default Real mode. In addition, the halt instruction causes an update to the variables displayed on the Inspect pane of the Write window.

Branch Compatibility
CANalyzer’s Transmit Branch = No
CANalyzer’s Analysis Branch = No
CANoe’s Simulation Branch = Yes
CANoe’s Analysis Branch = No

Related Functions
inspect
stop

Example

halt();  //Halts the simulation after this statement
...

The CAPL Functions

inport

Syntax
byte inport (word addr);

Description
Reads a byte from the parallel port.

Parameter
addr = port address
The built-in constants LPT1, LPT2, and LPT3 can be used as a port address:
LPT1 = 0x378
LPT2 = 0x278
LPT3 = 0x3BC

Returns
Port value

Availability
Available in all versions.

Observation
For Windows NT and 2000 users, a generic I/O driver must be installed to use this function. Follow the Readme.txt file in the Exec\GploDrv directory.

Branch Compatibility
CANalyzer’s Transmit Branch = Yes
CANalyzer’s Analysis Branch = Yes
CANoe’s Simulation Branch = Yes
CANoe’s Analysis Branch = Yes

Related Functions
inportLPT
outport
outportLPT

Example
val = inport(0x3f8); //Reads port 0x3f8
...
inportLPT

Syntax
byte inportLPT (word addr);

Description
Reads a byte from the parallel port.

Parameter
addr = port address
The built-in constants LPT1, LPT2, and LPT3 can be used as a port address:
LPT1 = 0x378
LPT2 = 0x278
LPT3 = 0x3BC

Returns
Port value

Availability
This function is supported in Version 3.1 and after.

Observation
This function changes the transmission mode of the parallel port automatically to input. If you want
to read from a parallel port, the port has to be in a bi-directional mode (PS/2 or “Byte” Modus).
Please check this in the CMOS setup (BIOS). Also for Windows NT and 2000 users, a generic I/O
driver must be installed to use this function. Follow the Readme.txt file in the Exec\GpIoDrv
directory.

Branch Compatibility
CANalyzer’s Transmit Branch = Yes
CANalyzer’s Analysis Branch = Yes
CANoe’s Simulation Branch = Yes
CANoe’s Analysis Branch = Yes

Related Functions
inport
outport
outportLPT

Example
byte val;
val = inportLPT(LPT1);
**Syntax**

```c
void inspect ();
```

**Description**

Updates the variables in the Inspect pane of the Write window.

**Parameter**

None

**Returns**

None

**Availability**

This function is supported in Version 4.1 and after.

**Branch Compatibility**

- CANalyzer's Transmit Branch = No
- CANalyzer's Analysis Branch = No
- CANoe's Simulation Branch = Yes
- CANoe's Analysis Branch = No

**Related Functions**

Halt

**Example**

```c
//Timer used to update the Inspect pane of the Write window
on timer inspectTimer
{
   inspect();
   settimer(inspectTimer, 100)  //update every 100ms
}
```
isExtId

Syntax
long isExtId (dword id);
long isExtId (message msg);

Description
Checks parameter for extended identifier (29 bit).

Parameter
msg = message variable
id = message identifier

Returns
0 = false
1 = true

Availability
Available in all versions.

Branch Compatibility
CANalyzer's Transmit Branch = Yes
CANalyzer's Analysis Branch = Yes
CANoe's Simulation Branch = Yes
CANoe's Analysis Branch = Yes

Related Functions
isStdId
mkExtId
valOfId

Example
//Passing the message ID as a parameter:
on message *
{
    if(isExtId(this.ID))
        write("29-bit identifier message received.");
}

//Passing the message variable as a parameter:
on message *
{
    if(isExtId(this))
        write("29-bit identifier message received.");
}
isStatisticAcquisitionRunning

Syntax
    int isStatisticAcquisitionRunning ();

Description
    Tests whether an acquisition range has already been activated in the Statistics window.

Parameter
    None

Returns
    0 = not running
    1 = running

Availability
    This function is supported in Version 3.0 and after.

Observation
    The CAPL program block this function appears must be located directly before the Statistics block
    in the Analysis Branch of CANalyzer and CANoe.

Branch Compatibility
    CANalyzer's Transmit Branch = No
    CANalyzer's Analysis Branch = Yes
    CANoe's Simulation Branch = No
    CANoe's Analysis Branch = Yes

Related Functions
    startStatisticAcquisition
    stopStatisticAcquisition

Example
    //Tests for activated acquisition range and stops it.
    //If no statistical data acquisition is
    //active a new one is started.
    if(isStatisticAcquisitionRunning)
    {
        //Stops the running acquisition range
        stopStatisticAcquisition();
    }
    else
    {
        //Starts a new acquisition range
        startStatisticAcquisition();
    }
The CAPL Functions

isStdId

Syntax
long isStdId (dword id);
long isStdId (message msg);

Description
Checks parameter for standard identifier (11 bit).

Parameter
msg = message variable
id = message identifier

Returns
0 = not standard
1 = standard

Availability
Available in all versions.

Branch Compatibility
CANalyzer's Transmit Branch = Yes
CANalyzer's Analysis Branch = Yes
CANoe's Simulation Branch = Yes
CANoe's Analysis Branch = Yes

Related Functions
isExtId
mkExtId
valOfId

Example
//Passing the message ID as a parameter:
on message *
{  if(isStdId(this.ID))
       write("11-bit identifier message received.");
}

//Passing the message variable as a parameter:
on message *
{  if(isStdId(this))
       write("11-bit identifier message received.");
}
**keypressed**

**Syntax**

defkey keypressed();

**Description**

Returns the key code of a pressed key. If no key is being pressed it returns 0.

**Parameter**

None

**Returns**

Key code of the pressed key
If the 8 lower bits do not equal 0, keypressed() returns the ASCII code of the next key in the keyboard buffer. If the 8 lower bits do not equal 0, the 8 upper bits represent the extended key code (see IBM PC Technical Reference Manual).

**Availability**

Available in all versions.

**Observation**

Only one key can be pressed at a time.

**Branch Compatibility**

CANalyzer’s Transmit Branch = Yes
CANalyzer’s Analysis Branch = Yes
CANoe’s Simulation Branch = Yes
CANoe’s Analysis Branch = Yes

**Related Functions**

N/A

**Example**

variables
{
    msTimer mytimer; //timer
    message 100 msg; //CAN message
}
on key F1
{
    setTimer(mytimer,100); //start 100 ms timer
    write("F1 pressed"); //output to write window
}
on timer mytimer
{
    if(keypressed()) //true if any key is pressed
    {
        setTimer(mytimer,100); //restart timer
        output(msg); //send while key pressed
    }
    else
    {
        write("F1 let go");
    }
}
**Syntax**

```c
void ltoa (long val, char s[], long base);
```

**Description**

Converts a number of a specific base into a string. The string must be large enough to accept the converted number!

**Parameter**

- `val` = number to be converted
- `s` = string which will contain the converted number
- `base` = numeric base

**Returns**

None

**Availability**

Available in all versions.

**Branch Compatibility**

- CANalyzer's Transmit Branch = Yes
- CANalyzer's Analysis Branch = Yes
- CANoe's Simulation Branch = Yes
- CANoe's Analysis Branch = Yes

**Related Functions**

- atol

**Example**

```c
long z = 255;
char s1[9];
char s2[9];
ltoa(z, s1, 2);  //binary
ltoa(z, s1, 10);  //decimal
...
// Result: s1 = 11111111, s2 = 255
```
The CAPL Functions

makeRGB

Syntax
   long makeRGB (long Red, long Green, long Blue);

Description
   Calculates the color value from the three primary color components.

Parameters
   Red = Red color component (0 – 255)
   Green = Green color component (0 – 255)
   Blue = Blue color component (0 – 255)

Returns
   Color value

Availability
   This function is supported in Version 4.1 and after.

Observation
   This is a very useful function if any color properties in the panels require changes.

Branch Compatibility
   CANalyzer’s Transmit Branch = No
   CANalyzer’s Analysis Branch = No
   CANoe’s Simulation Branch = Yes
   CANoe’s Analysis Branch = Yes

Related Functions
   enableControl
   putValueToControl
   setControlForeColor
   setControlBackColor
   setControlProperty

Example
   //set the back color of an indicator to green
   setControlProperty("Measurements", "StatusIndicator", "BackColor", makeRGB(0, 255, 0));
The CAPL Functions

mkExtId

dword mkExtId (dword id);

Description
Generates an extended (29-bit) message identifier from a standard (11-bit) message identifier.

Parameter
id = message identifier

Returns
Extended message identifier

Availability
Available in all versions.

Branch Compatibility
CANalyzer’s Transmit Branch = Yes
CANalyzer’s Analysis Branch = Yes
CANoe’s Simulation Branch = Yes
CANoe’s Analysis Branch = Yes

Related Functions
isExtId
isStdId
valOfId

Example
on message *
{
    ...
    msg.id = mkExtId(this.id);
    ...
}
The CAPL Functions

msgBeep

Syntax
msgBeep (long soundType);

Description
Plays back a sound predefined by the Windows system.

Parameter
soundType = Integer for the predefined sound. Specifically these are:
0 = MB_ICONASTERISK SystemAsterisk
1 = MB_ICONEXCLAMATION SystemExclamation
2 = MB_ICONHAND SystemHand
3 = MB_ICONQUESTION SystemQuestion
4 = MB_OK SystemDefault
5 = Standard beep using the PC speaker (default)

Returns
None

Availability
This function is supported in Version 3.0 and after.

Observation
If the sound type cannot be played, the standard beep is used. Also make sure the sound is activated within the Windows Control Panel.

Branch Compatibility
CANalyzer’s Transmit Branch = Yes
CANalyzer’s Analysis Branch = Yes
CANoe’s Simulation Branch = Yes
CANoe’s Analysis Branch = Yes

Related Functions
beep

Example
...//Standard signal question
msgBeep(3);
...
openFileRead

Syntax

dword openFileRead (char filename[], dword mode);

Description

Opens a file for read access.

Parameter

filename = name of file
mode = type of file
 0 = ASCII mode
 1 = binary mode

Returns

File handle used for read operations
 0 = unsuccessful

Availability

This function is supported in Version 3.0 and after.

Branch Compatibility

CANalyzer's Transmit Branch = Yes
CANalyzer's Analysis Branch = Yes
CANoe's Simulation Branch = Yes
CANoe's Analysis Branch = Yes

Related Functions

fileClose
fileGetBinaryBlock
fileGetString
fileGetStringSZ
filePutString
fileRewind
fileWriteBinaryBlock
getProFileArray
getProFileFloat
getProFileInt
getProFileString
openFileWrite
setFilePath
setWritePath
writeProFileFloat
writeProFileInt
writeProFileString

Example

dword glbHandle = 0;
glebHandle = openFileRead("datafile.txt", 0);
openFileWrite

Syntax
   dword openFileWrite (char filename[], dword mode);

Description
   Opens a file for write access. An already existing file will be overwritten.

Parameter
   filename = name of file
   mode = type of file
   0 = ASCII mode
   1 = binary mode
   2 = append data to end of file in ASCII mode
   3 = append data to end of file in binary mode

Returns
   File handle used for write operations.
   0 = unsuccessful

Availability
   This function is supported in Version 3.0 and after.

Observation
   Use the setWritePath() function to write to another directory; the default directory is the same as
   the active saved configuration.

Branch Compatibility
   CANalyzer's Transmit Branch = Yes
   CANalyzer's Analysis Branch = Yes
   CANoe's Simulation Branch = Yes
   CANoe's Analysis Branch = Yes

Related Functions
   fileClose
   fileGetBinaryBlock
   fileGetString
   fileGetStringSZ
   filePutString
   fileRewind
   fileWriteBinaryBlock
   getProFileArray
   getProFileFloat
   getProFileInt
   getProFileString
   openFileRead
   setFilePath
   setWritePath
   writeProFileFloat
   writeProFileInt
   writeProFileString

Example
   dword glbHandle = 0;
   glbHandle = openFileWrite("destination.bmp", 1);
The CAPL Functions

outport

Syntax
void outport (word addr, byte value);

Description
Outputs a byte to a parallel port.

Parameter
addr = port address
value = byte to send

Returns
None

Availability
Available in all versions.

Observation
For Windows NT and 2000 users, a generic I/O driver must be installed to use this function. Follow the Readme.txt file in the Exec\GpIoDrv directory.

Branch Compatibility
CANalyzer’s Transmit Branch = Yes
CANalyzer’s Analysis Branch = Yes
CANoe’s Simulation Branch = Yes
CANoe’s Analysis Branch = Yes

Related Functions
inport
inportLPT
outportLPT

Example
outport(0x3f8, 12);    //sends 12 to port 0x3f8
outport(LPT2, ‘x’);   //sends ‘x’ to LPT2
The CAPL Functions

outportLPT

Syntax
byte outportLPT (word addr, byte value);

Description
Outputs a byte to a parallel port.

Parameter
value = byte to send
addr = port address or predefined LPTx constant
The built-in constants LPT1, LPT2, and LPT3 can be used as a port address:
  LPT1 = 0x378
  LPT2 = 0x278
  LPT3 = 0x3BC

Returns
None

Availability
This function is supported in Version 3.1 and after.

Observation
This function changes the transmission mode of the parallel port automatically to output. If you want to write to a parallel port, the port has to be in a bi-directional mode (PS/2 or “Byte” Modus). Please check this in the CMOS setup (BIOS). Also for Windows NT and 2000 users, a generic I/O driver must be installed to use this function. Follow the Readme.txt file in the Exec\GpioDrv directory.

Branch Compatibility
  CANalyzer’s Transmit Branch = Yes
  CANalyzer’s Analysis Branch = Yes
  CANoe’s Simulation Branch = Yes
  CANoe’s Analysis Branch = Yes

Related Functions
  inport
  inportLPT
  outport

Example
  //output hex value to LPT1
  outportLPT(LPT1, 0x55);
The CAPL Functions

**output**

**Syntax**

```c
void output (message msg);
```

**Description**

Sends different types of messages from the program block onto the CAN bus.

**Parameter**

- `msg`: message of a specific type

**Returns**

None

**Availability**

Available in all versions.

**Observation**

This function supports other types of message from different buses or protocols. See example below.

**Branch Compatibility**

- CANalyzer's Transmit Branch = Yes
- CANalyzer's Analysis Branch = Yes
- CANoe's Simulation Branch = Yes
- CANoe's Analysis Branch = Yes

**Related Functions**

N/A

**Example**

```c
...  
message 0x100 msg;  
pg 0xFE01x pgmsg;  
LINmessage 0x10 LINmsg;  
GMLANmessage 0x1234x GMLANmsg;  
...  
output(msg);  
output(pgmsg);  
output(LINmsg);  
output(GMLANmsg);  
...  
```
putValue

Syntax
void putValue (EnvVarName, int val); //Form 1
void putValue (EnvVarName, float val); //Form 2
void putValue (EnvVarName, char val[]); //Form 3
void putValue (EnvVarName, byte val[]); //Form 4
void putValue (EnvVarName, byte val[], long offset); //Form 5

Description
Sets an environment variable. For character array or string environment variables (Form 3, 4, and 5) the active value is saved to a buffer.

Parameter
EnvVarName = environment variable name
val = environment variable value
offset = starting position (byte)

Returns
None

Availability
Available in all versions.

Branch Compatibility
CANalyzer's Transmit Branch = No
CANalyzer's Analysis Branch = No
CANoe's Simulation Branch = Yes
CANoe's Analysis Branch = Yes

Related Functions
callAllOnEnvVar
getValue
getValueSize
putValueToControl

Example
//Assign the value 0 to the environment variable “Switch”
putValue(Switch, 0);

//Assign the value 22.5 to the environment variable “Temperature”
putValue(Temperature, 22.5);

//Assign the value Master to environment variable “NodeName”
putValue(NodeName, “Master”);
putValueToControl

Syntax

void putValueToControl (char panel[], char control[], float val);
void putValueToControl (char panel[], char control[], long val);
void putValueToControl (char panel[], char control[], char val[]);
void putValueToControl (char panel[], char control[], BEANmessage val);
void putValueToControl (char panel[], char control[], message val);
void putValueToControl (char panel[], char control[], pg val);
void putValueToControl (char panel[], char control[], LINmessage val);
void putValueToControl (char panel[], char control[], VANmessage val);

Description

Displays a value to the Multi-Display element on a panel. The value can be numeric, string, or data bytes from a specific message type.

Parameter

  panel = panel title
  control = name of the Multi-Display element
  val = value of various format from numeric to text to message data bytes

Returns

  None

Availability

  This function is supported in Version 4.0 and after.

Observation

  Environment variables are not used when using this function.

Branch Compatibility

  CANalyzer's Transmit Branch = No
  CANalyzer's Analysis Branch = No
  CANoe's Simulation Branch = Yes
  CANoe's Analysis Branch = Yes

Related Functions

  getValue
  getValueSize
  putValue

Example

  //display a message's data bytes
  on message *
  {
    putValueToControl("Gateway", "NameOfControl", this);
  }
Syntax

dword random (dword x);

Description

Calculates a random value n such that 0 <= n < x.

Parameter

x = upper limit for the random value.

Returns

Random value

Availability

Available in all versions.

Branch Compatibility

CANalyzer's Transmit Branch = Yes
CANalyzer's Analysis Branch = Yes
CANoe's Simulation Branch = Yes
CANoe's Analysis Branch = Yes

Related Functions

N/A

Example

dword randVal;
randVal = random(101);  //returns a value from 0 to 100
The CAPL Functions

replayResume

dword replayResume (char pName[]);

Description
Resumes a Replay block after it was suspended by the replaySuspend() function.

Parameter
pName = name of the Replay block

Returns
1 = successful
0 = cannot be resumed or the Replay block does not exist

Availability
This function is supported in Version 4.0 and after.

Branch Compatibility
CANalyzer’s Transmit Branch = Yes
CANalyzer’s Analysis Branch = Yes
CANoe’s Simulation Branch = Yes
CANoe’s Analysis Branch = Yes

Related Functions
replayStart
replayState
replayStop
replaySuspend

Example
on key ‘r’
{
    replayResume(“nameofReplayblock”);
}
The CAPL Functions

replayStart

dword replayStart (char pName[]);

Description
Starts a Replay block to replay the associated log file. The data at the beginning of the file always
starts replaying first.

Parameter
pName = name of the Replay block

Returns
1 = successful
0 = cannot be started or the Replay block does not exist

Availability
This function is supported in Version 4.0 and after.

Recommendation
To replay a file that has been suspended or paused, use the replayResume() function.

Branch Compatibility
CANalyzer’s Transmit Branch = Yes
CANalyzer’s Analysis Branch = Yes
CANoe’s Simulation Branch = Yes
CANoe’s Analysis Branch = Yes

Related Functions
replayResume
replayState
replayStop
replaySuspend

Example
on key ‘s’
{
    replayStart(“nameofReplayblock”);
}
The CAPL Functions

**replayState**

**Syntax**

dword replayState (char pName);

**Description**

Returns the state of a Replay block.

**Parameter**

pName = name of the Replay block

**Returns**

-1 = Replay block does not exist
0 = Replay block is stopped
1 = Replay block is running
2 = Replay block is suspended

**Availability**

This function is supported in Version 4.0 and after.

**Branch Compatibility**

CANalyzer’s Transmit Branch = Yes
CANalyzer’s Analysis Branch = Yes
CANoe’s Simulation Branch = Yes
CANoe’s Analysis Branch = Yes

**Related Functions**

replayResume
replayStart
replayStop
replaySuspend

**Example**

```cpp
on key ‘s’
{
    state = replayState("nameofReplayblock");
    switch(state)
    {
        case 0:
            write("Replay block is stopped");
            break;
        case 1:
            write("Replay block is running");
            break;
        case 2:
            write("Replay block is suspended");
            break;
        default:
            write("Error: Replay block has an unknown state!");
            break;
    }
}
```
replayStop

Syntax
   dword replayStop (char pName);

Description
   Stops a Replay block from replaying.

Parameter
   pName = name of the Replay block

Returns
   1 = successful
   0 = cannot be stopped or the Replay block does not exist

Availability
   This function is supported in Version 4.0 and after.

Branch Compatibility
   CANalyzer’s Transmit Branch = Yes
   CANalyzer’s Analysis Branch = Yes
   CANoe’s Simulation Branch = Yes
   CANoe’s Analysis Branch = Yes

Related Functions
   replayResume
   replayStart
   replayState
   replaySuspend

Example
   on key ‘s’
   {
      replayStop(“nameofReplayblock”);
   }
replaySuspend

**Syntax**
dword replaySuspend (char pName);

**Description**
Suspends a Replay block from replaying. The Replay Block can be resumed by the replayResume() function.

**Parameter**
pName = name of the Replay block

**Returns**
1 = successful
0 = cannot be suspended or the Replay block does not exist

**Availability**
This function is supported in Version 4.0 and after.

**Recommendation**
To resume back at the beginning of the file, use the replayStart() function.

**Branch Compatibility**
CANalyzer’s Transmit Branch = Yes
CANalyzer’s Analysis Branch = Yes
CANoe’s Simulation Branch = Yes
CANoe’s Analysis Branch = Yes

**Related Functions**
replayResume
replayStart
replayState
replayStop

**Example**
```cpp
on key ‘s’
{
    replaySuspend(“nameofReplayblock”);
}
```
resetCan

Syntax
   void resetCan ();

Description
   Resets all the CAN controller.

Parameter
   None

Returns
   None

Availability
   Available in all versions.

Observation
   Typical condition when this function is invoked is when the CAN controller went “busoff”. Since execution of the function takes some time and the CAN controller is briefly disconnected from the bus, messages can be lost during a reset.

Recommendation
   To only reset a specific CAN controller by channel number, use the resetCanEx() function.

Branch Compatibility
   CANalyzer’s Transmit Branch = Yes
   CANalyzer’s Analysis Branch = No
   CANoe’s Simulation Branch = Yes
   CANoe’s Analysis Branch = No

Related Functions
   resetCanEx
   setBtr
   setOcr

Example
   on busoff
   {
      resetCan();
   }
resetCanEx

**Syntax**

```c
void resetCanEx (long channel);
```

**Description**

Resets the CAN controller for a specific CAN channel.

**Parameters**

- CAN channel

**Returns**

None

**Availability**

This function is supported in Version 4.1 and after.

**Observation**

Typical condition when this function is invoked is when the CAN controller went “busoff”. Since execution of the function takes some time and the CAN controller is briefly disconnected from the bus, messages can be lost during a reset.

**Recommendation**

To reset all the CAN controller at once, use the resetCan() function.

**Branch Compatibility**

- CANalyzer’s Transmit Branch = Yes
- CANalyzer’s Analysis Branch = Yes
- CANoe’s Simulation Branch = Yes
- CANoe’s Analysis Branch = Yes

**Related Functions**

- resetCan
- setBtr
- setOcr

**Example**

```c
on key ‘r’
{
    //channel 1 is reset when ‘r’ key is pressed
    resetCanEx(1);
}
```
**runError**

**Syntax**

```c
void runError (long err, long x);
```

**Description**

Triggers a run-time error. Outputs the error message to the Write window indicating the error number, the passed number, and then terminates the measurement.

**Parameter**

- `err` = numbers that are represented in CANalyzer/CANoe as references for the user (values under 1000 are reserved for internal purposes)
- `x` = reserved for future expansion (can be any number)

**Returns**

None

**Availability**

Available in all versions.

**Branch Compatibility**

- CANalyzer’s Transmit Branch = Yes
- CANalyzer’s Analysis Branch = Yes
- CANoe’s Simulation Branch = Yes
- CANoe’s Analysis Branch = Yes

**Related Functions**

N/A

**Example**

```c
if(rpm < 0) runError(1001,1);
...
**seqFileClose**

**Syntax**

```c
long seqFileClose (long fileHandle);
```

**Description**

Closes a specific file through its handle assigned by the seqFileLoad() function.

**Parameter**

- `fileHandle` = value of the file handle

**Returns**

- `0` = successful
- `!0` = unsuccessful

**Availability**

This function is supported prior to Version 3.0.

**Recommendation**

This function has been replaced by the fileClose() function.

**Branch Compatibility**

- CANalyzer’s Transmit Branch = Yes
- CANalyzer’s Analysis Branch = No
- CANoe’s Simulation Branch = Yes
- CANoe’s Analysis Branch = No

**Related Functions**

- seqFileGetBlock
- seqFileGetLine
- seqFileGetLineSZ
- seqFileLoad
- seqFileRewind

**Example**

```c
long fileHandle;
long errorCode;

fileHandle = seqFileLoad("cap1.dat");
...
errorCode = seqFileClose(fileHandle);

if(errorCode == 0)
{
    write("File closed.");
}
else
{
    write("Error closing file.");
}
```
**seqFileGetBlock**

**Syntax**

```c
long seqFileGetBlock (char buffer[], dword bufferSize, long fileHandle);
```

**Description**

Reads a block of characters from a file. Newline characters are also read into the buffer. The file position indicator is advanced by the number of characters successfully read.

**Parameter**

- `buffer` = block of characters
- `bufferSize` = size of the buffer
- `fileHandle` = value of the file handle

**Returns**

- 0 = unsuccessful
- !0 = number of characters successfully read, which may be less than bufferSize if the end-of-file character is encountered

**Availability**

This function is supported prior to Version 3.0.

**Recommendation**

This function has been replaced by the `fileGetBinaryBlock()` function.

**Branch Compatibility**

- CANalyzer’s Transmit Branch = Yes
- CANalyzer’s Analysis Branch = No
- CANoe’s Simulation Branch = Yes
- CANoe’s Analysis Branch = No

**Related Functions**

- `seqFileClose`
- `seqFileGetLine`
- `seqFileGetLineSZ`
- `seqFileLoad`
- `seqFileRewind`

**Example**

```c
char buffer[10];
long fileHandle;
long charsRead;

fileHandle = seqFileLoad("cap1.dat");
charsRead = seqFileGetBlock(buffer, 10, fileHandle);

if(charsRead > 0)
{
    write("Characters read: %d", charsRead);
}
else
{
    write("Error reading file.");
}
```
seqFileGetLine

Syntax
long seqFileGetLine (char buffer[], dword bufferSize, long fileHandle);

Description
Reads a line from a file until a newline character or it reaches the buffer size limit. The function
retains the newline character, but the line is not null-terminated. The null character must be placed
into the character array after the data if the buffer is to be used as a string.

Parameter
buffer = line characters
bufferSize = size of buffer
fileHandle = value of the file handle

Returns
Number of characters successfully read
<0 = unsuccessful

Availability
This function is supported prior to Version 3.0.

Recommendation
This function has been replaced by the fileGetString() function.

Branch Compatibility
CANalyzer's Transmit Branch = Yes
CANalyzer's Analysis Branch = No
CANoe's Simulation Branch = Yes
CANoe's Analysis Branch = No

Related Functions
seqFileClose
seqFileGetBlock
seqFileGetLineSZ
seqFileLoad
seqFileRewind

Example
char buffer[100];
long fileHandle;
long charsRead;

fileHandle = seqFileLoad("capl.dat");
charsRead = seqFileGetLine(buffer, 100, fileHandle);

if(charsRead >= 0)
{
    write("Characters read: %d", charsRead);
    //Add a null to end before printing
    buffer[charsRead] = 0;
    write("The string read: %s", buffer);
}
else
{
    write("Error reading file.");
}
seqFileGetLineSZ

Syntax
long seqFileGetLineSZ (char buffer[], dword bufferSize, long fileHandle, unsigned long nullTerm);

Description
Reads a line from a file until a newline character or it reaches the buffer size limit. The function retains the newline character, and it is null-terminated.

Parameter
- buffer = line characters
- bufferSize = size of buffer
- fileHandle = value of the file handle
- nullTerm = 0 (not null-terminated)
  1 (null-terminated)

Returns
Number of characters successfully read
<0 = unsuccessful

Availability
This function is supported prior to Version 3.0.

Recommendation
This function has been replaced by the fileGetStringSZ() function.

Branch Compatibility
- CANalyzer's Transmit Branch = Yes
- CANalyzer's Analysis Branch = No
- CANoe's Simulation Branch = Yes
- CANoe's Analysis Branch = No

Related Functions
- seqFileClose
- seqFileGetBlock
- seqFileGetLine
- seqFileLoad
- seqFileRewind

Example
char buffer[100];
long fileHandle;
long charsRead;

fileHandle = seqFileLoad("capl.dat");
charsRead = seqFileGetLineSZ(buffer, 100, fileHandle, 1);

if(charsRead >= 0)
{
    write("Characters read: %d", charsRead);
    write("The line read: %s", buffer);
}
else
{
    write("Error reading file.");
}
The CAPL Functions

seqFileLoad

Syntax
long seqFileLoad (char fileName[]);

Description
Opens the file for read-only. The path of the file is given by the seqFilePath entry of the [CAPL]
section within the CAN.ini file located in the Exec32 directory. Any drive and path information
provided in the parameter is ignored.

Parameter
fileName = name of the file

Returns
<=0 = unsuccessful
>0 = file handle value

Availability
This function is supported prior to Version 3.0.

Observation
The CAN.ini file must be properly set up before using this function.

Recommendation
This function has been replaced by the setWritePath() or setFilePath() function.

Branch Compatibility
CANalyzer’s Transmit Branch = Yes
CANalyzer’s Analysis Branch = No
CANoe’s Simulation Branch = Yes
CANoe’s Analysis Branch = No

Related Functions
seqFileClose
seqFileGetBlock
seqFileGetLine
seqFileGetLineSZ
seqFileRewind

Example
long fileHandle;

fileHandle = seqFileLoad(“setup.txt”);

if(fileHandle <= 0)
{
    write(“Error opening setup.txt.”);
}
else
{
    write(“setup.txt opened with file handle %d”, fileHandle);
}
seqFileRewind

Syntax
long seqFileRewind (long fileHandle);

Description
Sets the file position indicator back to the beginning of the file.

Parameter
fileHandle = value of the file handle

Returns
0 = successful
!0 = unsuccessful

Availability
This function is supported prior to Version 3.0.

Recommendation
This function has been replaced by the fileRewind() function.

Branch Compatibility
CANalyzer’s Transmit Branch = Yes
CANalyzer’s Analysis Branch = No
CANoe’s Simulation Branch = Yes
CANoe’s Analysis Branch = No

Related Functions
seqFileClose
seqFileGetBlock
seqFileGetLine
seqFileGetLineSZ
seqFileRewind

Example
long fileHandle;
long errCode;

fileHandle = seqFileLoad("setup.txt");

... errCode = seqFileRewind(fileHandle);
if(errCode == 0)
{ write("setup.txt rewind successful."); }
else
{ write("Rewind failed."); }
setBtr

Syntax
void setBtr (long channel, byte btr0, byte btr1);

Description
Sets the baud rate based on the Bit Timing Register of a CAN controller. The values become effective until the next call of the function resetCan() or resetCanEx().

Parameter
channel = 0 (both CAN controllers)
1 (channel 1)
2 (channel 2)
btr0 = value of Bit Timing Register 0
btr1 = value of Bit Timing Register 1

Returns
None

Availability
Available in all versions.

Branch Compatibility
CANalyzer’s Transmit Branch = Yes
CANalyzer’s Analysis Branch = No
CANoe’s Simulation Branch = Yes
CANoe’s Analysis Branch = No

Related Functions
resetCan
resetCanEx
setOcr

Example
... setBtr(0, 0x00, 0x3a); //500 kbaud for 82c200
resetCan();  //activate
...
The CAPL Functions

setBusContext

dword setBusContext (dword context);

Description
Sets the bus context of the network node (Gateway). The bus context plays a role exclusively in modeling gateways. In this case, a series of CAPL functions such as canOnline() and canOffline() may have more than one meaning in terms of the bus interface (channel) to be used. A similar type of problem occurs when identical node layer modules are used simultaneously within a CAPL block. A distinction must be made between the instances of the node layer, both for calls to CAPL functions that are implemented in the node layers and for implementing callbacks.
To facilitate this distinction, a bus context is placed in the CAPL program by the runtime environment while a callback is being executed by the node layer. This context unambiguously identifies the node layer that is making the call. In a similar manner, the call of a CAPL function that is implemented in a node layer is forwarded on to the appropriate node layer, depending on the current bus context. This also applies to the CAPL functions mentioned above, canOnline() and canOffline().

Parameters
context = the new context to be set

Returns
Bus context that was valid before the call was made

Availability
This function is supported in Version 3.2 and after.

Branch Compatibility
CANalyzer’s Transmit Branch = No
CANalyzer’s Analysis Branch = No
CANoe’s Simulation Branch = Yes
CANoe’s Analysis Branch = No

Related Functions
getBusContext
getBusNameContext

Example
dword oldValue, newContextValue;

///previous context value is stored in oldValue:
oldValue = setBusContext(newContextValue);
setCanCabsMode

Syntax
long setCanCabsMode (long ntype, long nchannel, long nmode, long nflags);

Description
Sets the mode of a CANcab. The modes do not apply to all CANcabs.

Parameters
ntype = unused; must be set to 0
nchannel = CAN channel
nmode = 0 (NORMAL)
  1 (SLEEP)
  2 (HI-VOLTAGE)
  3 (HI-SPEED)
  4 (DUAL_WIRE)
  5 (SINGLE_WIRE_LOW)
  6 (SINGLE_WIRE_HIGH)
  7 (RESERVED)
  8 (EVA_1)
  9 (EVA_2)
 10 (EVA_3)
nflags = 0 (AUTOWAKEUP; only together with SLEEP mode)
  1 (HIGHPRIO; only together with CANcab 5790c, to clear tx-buffers

Returns
0 = successful
!0 = unsuccessful

Availability
This function is supported in Version 4.1 and after.

Branch Compatibility
  CANalyzer's Transmit Branch = Yes
  CANalyzer's Analysis Branch = Yes
  CANoe's Simulation Branch = Yes
  CANoe's Analysis Branch = Yes

Related Functions
setPortBits

Example
on key ‘n’
{  ntype = 0;
   nmode = 0;
   nchannel = 1;
   nflags = 0;
   setCanCabsMode(ntype, nchannel, nmode, nflags);
   write("normal mode");
}

The CAPL Functions

setControlBackColor

Syntax
void setControlBackColor (char panel[], char control[], long color);

Description
Sets the background color of panel elements.

Parameters
panel = panel name ("" – references all opened panels)
control = name of the panel element ("" – references all elements on the panel)
color = color value (e.g. calculated by makeRGB() function)

Returns
None

Availability
This function is supported in Version 4.1 and after.

Branch Compatibility
CANalyzer’s Transmit Branch = No
CANalyzer’s Analysis Branch = No
CANoe’s Simulation Branch = Yes
CANoe’s Analysis Branch = Yes

Related Functions
enableControl
putValueToControl
setControlForeColor
setControlProperty

Example
...
setControlBackColor("motor", "PedalPos", makeRGB(255,0,0));
...

110
setControlForeColor

Syntax
void setControlForeColor (char panel[], char control[], long color);

Description
Sets the foreground color of panel elements.

Parameters
panel = panel name ("" – references all opened panels)
control = name of the panel element ("" – references all elements on the panel)
color = color value (e.g. calculated by makeRGB() function)

Returns
None

Availability
This function is supported in Version 4.1 and after.

Branch Compatibility
CANalyzer’s Transmit Branch = No
CANalyzer’s Analysis Branch = No
CANoe’s Simulation Branch = Yes
CANoe’s Analysis Branch = Yes

Related Functions
enableControl
putValueToControl
setControlBackColor
setControlProperty

Example
...
setControlForeColor("motor", "PedalPos", makeRGB(255,0,0));
...

setControlProperty

Syntax
void setControlProperty (char panel[], char control[], char property[], long value);
void setControlProperty (char panel[], char control[], char property[], float value);
void setControlProperty (char panel[], char control[], char property[], char value[]);

Description
Sets a property of an ActiveX control.

Parameters
- panel = panel name ("" – references all opened panels)
- control = name of the panel element ("" – references all elements on the panel)
- property = name of the property
- value = value to be set (long, float or string value)

Returns
None

Availability
This function is supported in Version 4.1 and after.

Branch Compatibility
- CANalyzer’s Transmit Branch = No
- CANalyzer’s Analysis Branch = No
- CANoe’s Simulation Branch = Yes
- CANoe’s Analysis Branch = Yes

Related Functions
- enableControl
- putValueToControl
- setControlBackColor
- setControlForeColor

Example
... setControlProperty("Measurements", "StatusIndicator", "Caption", "running");
setControlProperty("Measurements", "StatusIndicator", "BackColor", makeRGB(0,145,255));
...
The CAPL Functions

**setDrift**

**Syntax**

```c
void setDrift (int drift);
```

**Description**

Sets the constant deviation for timers of a network node. Inputs for the two values may lie between –10000 and 10000 (corresponds to –100.00% and 100.00%). If the value does not lie within this range, a message is output in the Write window.

**Parameter**

- `drift` = integer for the constant deviation

**Returns**

None

**Availability**

This function is supported in Version 3.0 and after.

**Observation**

Setting a drift causes any existing jitter to be reset.

**Branch Compatibility**

- CANalyzer’s Transmit Branch = No
- CANalyzer’s Analysis Branch = No
- CANoe’s Simulation Branch = Yes
- CANoe’s Analysis Branch = No

**Related Functions**

- getDrift
- getJitterMax
- getJitterMin
- setJitter

**Example**

```c

//Sets the drift to 35.5 percent
setDrift(3550);
...
```
setFilePath

**Syntax**

```c
void setFilePath (char path[], unsigned int mode);
```

**Description**

Sets the read and write path to the directory. The path can be given as absolute or relative to the currently active configuration.

**Parameter**

- **path** = the path to the directory
- **mode** = 0 (read only)
  - 1 (write only)
  - 2 (both read/write)

**Returns**

None

**Availability**

This function is supported in Version 4.1 and after.

**Branch Compatibility**

- CANalyzer's Transmit Branch = Yes
- CANalyzer's Analysis Branch = Yes
- CANoe's Simulation Branch = Yes
- CANoe's Analysis Branch = Yes

**Related Functions**

- fileClose
- fileGetBinaryBlock
- fileGetString
- fileGetStringSZ
- filePutString
- fileRewind
- fileWriteBinaryBlock
- getProFileArray
- getProFileFloat
- getProFileInt
- getProFileString
- openFileRead
- openFileWrite
- setWritePath
- writeProFileFloat
- writeProFileInt
- writeProFileString

**Example**

```c
...  
  setFilePath("c:\\Desktop\\Project", 2);
  ...
...```
setJitter

Syntax
void setJitter (int min, int max);

Description
Sets the jitter interval for the timers of a network node. The two values may lie between –10000 and 10000 (corresponds to –100.00% and 100.00%). If one of the two values does not lie within this range, a message is output in the Write window.

Parameter
min = integer for the lower interval limit
max = integer for the upper interval limit

Returns
None

Availability
This function is supported in Version 3.0 and after.

Observation
Setting a jitter causes any existing drift to be reset. To utilize both jitter and drift simultaneously, look at the example below.

Branch Compatibility
CANalyzer’s Transmit Branch = No
CANalyzer’s Analysis Branch = No
CANoe’s Simulation Branch = Yes
CANoe’s Analysis Branch = No

Related Functions
getDrift
getJitterMax
getJitterMin
setDrift

Example

//Set a jitter with +–4 percent
setJitter(-400, 400);
...

... //Set a jitter with +–4 percent and a drift of 17 percent
setJitter(1300, 2100);
...
The CAPL Functions

setLogFileName

Syntax
void setLogFileName (char fileName[]);

Description
Sets the name of the log file.

Parameter
fileName = new name of the log file.

Returns
None

Availability
Available in all versions.

Observation
The file name must not contain a file extension. The name may be an absolute path or just a file
name. If a path is supplied, the path must exist prior to the start of the simulation. If the path does
not exist, the call to setLogFileName() will be ignored. If a single file name is supplied, the log file
will be placed in the directory of the current configuration. The directories of the path must be
separated by double backslash ("\").

Branch Compatibility
CANalyzer's Transmit Branch = No
CANalyzer's Analysis Branch = Yes
CANoe's Simulation Branch = No
CANoe's Analysis Branch = Yes

Related Functions
setPostTrigger
setPreTrigger
startLogging
stopLogging
trigger
writeToLog
writeToLogEx

Example
//Set the name of the logging file to “newlog” in the
//directory of the current configuration.

... setLogFileName("newlog");
...

//Set the absolute path of the logging file.
//The path // c:\canw\demo\automot\newlog must
//be created before the simulation begins.

... setLogFileName("c:\canw\demo\automot\newlog");
...
**setMsgTime**

**Syntax**
```c
void setMsgTime (message m1, NOW);
void setMsgTime (message m1, message m2));
```

**Description**
Assigns a time source to a message.

**Parameter**
- `m1` = message to be assigned
- `NOW` = current simulation/measurement time
- `m2` = message where the time is extracted

**Returns**
None

**Availability**
This function is supported prior to Version 2.5.

**Recommendation**
This function is no longer use. It has been replaced by the TIME message selector. The TIME selector represents the time stamp of a message.

**Branch Compatibility**
- CANalyzer’s Transmit Branch = Yes
- CANalyzer’s Analysis Branch = Yes
- CANoe’s Simulation Branch = Yes
- CANoe’s Analysis Branch = Yes

**Related Functions**
N/A

**Example**
None
setOcr

**Syntax**

```c
void setOcr (long channel, byte ocr);
```

**Description**

Sets the CAN controller's Output Control Register. The value become effective until the next call of the function resetCan() or resetCanEx(). It should be noted that this value depends on the CAN platform used or the CAN hardware used (CANcardX or XL does not require this function call).

**Parameter**

- `channel` = 0 (both CAN controllers)
  - 1 (channel 1)
  - 2 (channel 2)
- `ocr` = value of the Output Control Registers

**Returns**

None

**Availability**

Available in all versions.

**Branch Compatibility**

- CANalyzer’s Transmit Branch = Yes
- CANalyzer’s Analysis Branch = No
- CANoe’s Simulation Branch = Yes
- CANoe’s Analysis Branch = No

**Related Functions**

- resetCan
- resetCanEx
- setBtr

**Example**

```c
... setOcr(0, 0x02);
resetCan();
...
```
The CAPL Functions

setPortBits

Syntax
void setPortBits (byte mode);

Description
This function is replaced by an simplified function, setCanCabsMode(). Both functions are used to set the mode of a CANcab or CANpiggy (CAN transceivers). Be extremely careful on using this function. First, this function applies to both CAN channels. Second, not all CANcabs or CANpiggies can have different mode settings. Highspeed 82C251 (251 in short) transceiver does not use this function because it can only operate in normal mode.

Parameters
mode = 8 bits parameter used to set both CAN transceivers on a controller (e.g. CANcardX, CANcardXL)

<table>
<thead>
<tr>
<th>Channels</th>
<th>CAN 1</th>
<th>CAN 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Transceiver: 252, 1041, 1053, 1054 (Bit 4-7 must be zeros)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bit Location</td>
<td>Bit 0</td>
<td>Bit 1</td>
</tr>
<tr>
<td>Normal Mode</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Sleep Mode</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>No Change</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>No Change</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Please note that bit 7 is most significant and bit 0 is least significant bit.

<table>
<thead>
<tr>
<th>Channels</th>
<th>CAN 1</th>
<th>CAN 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Transceiver: 5790 (Bit 6-7 must be zeros)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bit Location</td>
<td>Bit 0</td>
<td>Bit 1</td>
</tr>
<tr>
<td>HighVoltage Mode</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>HighSpeed Mode</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Sleep Mode</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Normal Mode</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

For the single-wired CAN transceiver 5790, bit 4 for high priority on Channel 1, bit 5 for high priority on Channel 2. These high priority flags are used to clear all transmit buffers.

Returns
None

Availability
This function is supported in Version 4.1 and after.

Recommendation
This function has been replaced by the setCanCabsMode() function.

Branch Compatibility
CANalyzer’s Transmit Branch = Yes
CANalyzer’s Analysis Branch = Yes
CANoe’s Simulation Branch = Yes
CANoe’s Analysis Branch = Yes
The CAPL Functions

Related Functions
setCanCabsMode

Example
//For 1054 transceivers: set channel 1 to normal
setPortBit(0x01);

//For 1054 transceivers: set channel 1 to sleep
setPortBit(0x02);

//For 1054 transceivers: set channel 2 to sleep
setPortBit(0x08);

//For 1054 transceivers: set channel 1 to sleep
//and channel 2 to normal mode
setPortBit(0x06);

//For 5790 transceivers: send a high voltage message
//on channel 1 and set channel 2 to normal.
setPortBit(0x0D);
output(msg);
//after the wakeup message is sent, the channel will
//set to normal mode automatically
**SetPostTrigger**

**Syntax**

```c
void setPostTrigger (long postTriggerTime);
```

**Description**

Sets the posttrigger time for logging. The posttrigger time set with this function is valid until the end of the measurement or until the next call of this function.

**Parameter**

`postTriggerTime` = new posttrigger time in milliseconds (-1 will set it until measurement stops)

**Returns**

0 = unsuccessful  
1 = successful

**Availability**

Available in all versions.

**Observation**

The post-trigger can also be set with the `stopLogging()` function.

**Branch Compatibility**

- CANalyzer’s Transmit Branch = No  
- CANalyzer’s Analysis Branch = Yes  
- CANoe’s Simulation Branch = No  
- CANoe’s Analysis Branch = Yes

**Related Functions**

- `setLogFileName`
- `setPreTrigger`
- `startLogging`
- `stopLogging`
- `trigger`
- `writeToLog`
- `writeToLogEx`

**Example**

```c
//Set the posttrigger time of logging to 2.5 seconds
...
setPostTrigger(2500);
...

//Set the posttrigger time for logging to when measurement stops
...
setPostTrigger(-1);
...```
### setPreTrigger

**Syntax**

```c
void setPreTrigger (long preTriggerTime);
```

**Description**

Sets the pretrigger time for logging. The pretrigger time set with this function is valid until the end of the measurement or until the next call of this function.

**Parameter**

- `preTriggerTime` = new pretrigger time in milliseconds

**Returns**

- `0` = unsuccessful
- `1` = successful

**Availability**

Available in all versions.

**Observation**

The pre-trigger can also be set with the `startLogging()` function.

**Branch Compatibility**

- CANalyzer’s Transmit Branch = No
- CANalyzer’s Analysis Branch = Yes
- CANoe’s Simulation Branch = No
- CANoe’s Analysis Branch = Yes

**Related Functions**

- `setLogFileName`
- `setPostTrigger`
- `startLogging`
- `stopLogging`
- `trigger`
- `writeToLog`
- `writeToLogEx`

**Example**

```c
// Set the pretrigger time of logging to 25 milliseconds
...
setPreTrigger(25);
...
```
setStartDelay

Syntax
void setStartDelay (int delay);

Description
Sets up a delay time for a network node to start. This function can only be called in the preStart event procedure. After it is called, the delay time can no longer be changed.

Parameter
delay = time to delay in ms (0 to 99999)

Returns
None

Availability
This function is supported in Version 3.0 and after.

Observation
It is possible in CANoe to set up a network node to start with a delay by right-clicking on the network node and select Configuration.

Recommendation
If a network node simulation require to pause its message transmission, the canOffline() and canOnline() functions are used.

Branch Compatibility
CANalyzer’s Transmit Branch = No
CANalyzer’s Analysis Branch = No
CANoe’s Simulation Branch = Yes
CANoe’s Analysis Branch = No

Related Functions
gGetStartDelay

Example
on preStart
{
    //Sets delay time to 10 seconds
    setStartDelay(10000);
}

**setTimer**

**Syntax**

```c
void setTimer (msTimer t, long duration);
void setTimer (timer t, long duration);
```

**Description**

Sets a timer in milliseconds or seconds depending on the data type.

**Parameter**

- `t` = timer variable of either milliseconds or seconds resolution
- `duration` = timer duration in either milliseconds or seconds

**Returns**

None

**Availability**

Available in all versions.

**Branch Compatibility**

- CANalyzer's Transmit Branch = Yes
- CANalyzer's Analysis Branch = Yes
- CANoe's Simulation Branch = Yes
- CANoe's Analysis Branch = Yes

**Related Functions**

cancelTimer

**Example**

```c
variables
{
    timer t1;
}

on start
{
    //Initialize a 5-second cyclic timer:
    setTimer(t1, 5);
}

on timer t1
{
    //Reset timer for another 5 seconds:
    setTimer(t1, 5);
}
```
**setWriteDbgLevel**

**Syntax**

```c
void setWriteDbgLevel (unsigned int priority);
```

**Description**

Sets the priority level for the writeDbgLevel() CAPL function. The output priority can be set for every network node.

**Parameter**

- `priority` = priority of current CAPL node for outputs to the Write window (0 to 15)
  - 0 = only write outputs with a priority of 0 are shown in the Write window
  - 5 = write outputs with a priority ranging from 0 to 5 are shown
  - 15 = all outputs are shown

**Returns**

None

**Availability**

This function is supported in Version 3.1 and after.

**Observation**

After applying this function, use the writeDbgLevel() function to output text into the Write window if the priority is greater than or equal to the set priority.

**Branch Compatibility**

- CANalyzer's Transmit Branch = No
- CANalyzer's Analysis Branch = No
- CANoe's Simulation Branch = Yes
- CANoe's Analysis Branch = No

**Related Functions**

- `write`
- `writeDbgLevel`

**Example**

```c
int i = 10;
int j = 12;

setWriteDbgLevel(7); //set priority for this node

writeDbgLevel(4, "This is shown: h = %lxh", j);
//Result in Write window: This is shown: h = 0ch

writeDbgLevel(9, "This is not shown: d = %ld", i);
//No output
```
setWritePath

Syntax
void setWritePath (char relativeOrAbsolutePath[]);

Description
Sets the write path for the function openFileWrite(). The path can be given as absolute or relative to the current configuration.

Parameter
The file path as a string. Use double back slashes.

Returns
None

Availability
This function is supported in Version 3.0 and after.

Branch Compatibility
CANalyzer’s Transmit Branch = Yes
CANalyzer’s Analysis Branch = Yes
CANoe’s Simulation Branch = Yes
CANoe’s Analysis Branch = Yes

Related Functions
fileClose
fileGetBinaryBlock
fileGetString
fileGetStringSZ
filePutString
fileRewind
fileWriteBinaryBlock
getProFileArray
getProFileFloat
getProFileInt
getProFileString
openFileRead
openFileWrite
setFilePath
writeProFileFloat
writeProFileInt
writeProFileString

Example
...
setWritePath(“C:\\temp”);
...

126
The CAPL Functions

**sin**

double sin (double x);

**Description**
Calculates the sine of x.

**Parameter**

\( x = \text{value in radians whose sine is to be calculated} \)

**Returns**

Sine of x

**Availability**
Available in all versions.

**Branch Compatibility**

- CANalyzer’s Transmit Branch = Yes
- CANalyzer’s Analysis Branch = Yes
- CANoe’s Simulation Branch = Yes
- CANoe’s Analysis Branch = Yes

**Related Functions**

- cos
- exp
- sqrt

**Example**

double x;
\( x = \sin(\pi); \quad //\pi \text{ is a built-in constant} \)

\( //\text{tangent function} \)

double tangent(double x)
{
    return sin(x) / cos(x);
}

snprintf

Syntax
long snprintf (char buffer[], long len, char format[], ...);

Description
This function corresponds to the C function sprintf(), but also with an parameter to indicate the
maximum length of the buffer. The overall length of the buffer may not exceed 100.

Parameter
Similar to the write() function, this function can take a variable number of arguments and it saves
the formatted string into a buffer. There should be a string formatting expressions for every format
parameter in the format string. The string formatting expressions are the same as the write() function and are listed here:

“%Id” or “%d” = decimal display
“%Ix” or “%x” = hexadecimal display
“%IX” or “%X” = hexadecimal display (with upper case letters)
“%Iu” or “%u” = unsigned display
“%Io” or “%o” = octal display
“%g” or “%f” = floating point display
“%s” = displays a string
“%c” = displays a character
“%%%” = displays ‘%’ character

Returns
Length of buffer

Availability
Available in all versions.

Branch Compatibility
CANalyzer’s Transmit Branch = No
CANalyzer’s Analysis Branch = No
CANoe’s Simulation Branch = Yes
CANoe’s Analysis Branch = Yes

Related Functions
write

Example
char infoStr[100];
int vol = 55;
byte bal = 3;
long res;

res = snprintf(infoStr, 100, “volume = %d, Balance = %u”, vol, bal);
//Result: res = 24; infoStr = “Volume = 55, Balance = 3”
The CAPL Functions

sqrt

double sqrt (double x);

Description
Calculates the square root of the parameter.

Parameter
x = value whose square root is to be calculated

Returns
Square root of x

Availability
Available in all versions.

Branch Compatibility
CANalyzer’s Transmit Branch = Yes
CANalyzer’s Analysis Branch = Yes
CANoe’s Simulation Branch = Yes
CANoe’s Analysis Branch = Yes

Related Functions
cos
exp
sin

Example
double x;
x = sqrt(4.0);
//Result: x = 2.0
The CAPL Functions

startLogging

Syntax
void startLogging(); //Form 1
void startLogging (char blockName[]); //Form 2
void startLogging (char blockName[], long preTriggerTime); //Form 3

Description
Form 1 – starts all Logging blocks immediately, bypassing all logging trigger settings
Form 2 – starts a specific Logging block
Form 3 – starts a specific Logging block with a pre-trigger logging time

Parameter
blockName = name of Logging block
preTriggerTime = pre-trigger time in milliseconds

Returns
None

Availability
This function is supported in Version 4.1 and after.

Branch Compatibility
CANalyzer’s Transmit Branch = Yes
CANalyzer’s Analysis Branch = Yes
CANoe’s Simulation Branch = Yes
CANoe’s Analysis Branch = Yes

Related Functions
setLogFileName
setPostTrigger
setPreTrigger
stopLogging
trigger
writeToLog
writeToLogEx

Example
//starts “blockname” with a pre-trigger time of 2 seconds
startLogging(“blockname”, 2000);
The CAPL Functions

startStatisticAcquisition

**Syntax**

```c
void startStatisticAcquisition();
```

**Description**

Activates a new acquisition range in the Statistics window. If an acquisition range has already been activated, the function has no effect since it cannot influence the currently active range.

**Parameter**

None

**Returns**

None

**Availability**

This function is supported in Version 3.0 and after.

**Observation**

The CAPL program block this function appears must be located directly before the Statistics block in the Analysis Branch of CANalyzer and CANoe.

**Branch Compatibility**

- CANalyzer’s Transmit Branch = No
- CANalyzer’s Analysis Branch = Yes
- CANoe’s Simulation Branch = No
- CANoe’s Analysis Branch = Yes

**Related Functions**

- isStatisticAcquisitionRunning
- stopStatisticAcquisition

**Example**

```c
//Tests for acquisition range and stops it.
//If no statistical data acquisition is active
//a new one is started.
if(isStatisticAcquisitionRunning)
{
    //Stops the running acquisition range
    stopStatisticAcquisition();
}
else
{
    //Starts a new acquisition range
    startStatisticAcquisition();
}
...```
Syntax

```c
void stop();
```

Description
Stops the ongoing measurement immediately.

Parameter
None

Returns
None

Availability
Available in all versions.

Recommendation
Under the Bus Off condition of a CAN controller, the CANalyzer or CANoe measurement doesn’t have to be stopped in order to reinitialize the controller to communicate again. A reset can be perform while the measurement is running with either the resetCAN() or resetCANEx() function.

Branch Compatibility
- CANalyzer’s Transmit Branch = Yes
- CANalyzer’s Analysis Branch = Yes
- CANoe’s Simulation Branch = Yes
- CANoe’s Analysis Branch = Yes

Related Functions
halt

Example
```c
on errorframe
{
    stop();  //End measurement when error frame is received
}
```
The CAPL Functions

stopLogging

Syntax
void stopLogging(); //Form 1
void stopLogging (char blockName[]); //Form 2
void stopLogging (char blockName[], long postTriggerTime); //Form 3

Description
Form 1 – stops all Logging blocks immediately, bypassing all logging trigger settings
Form 2 – stops a specific Logging block
Form 3 – stops a specific Logging block with a post-trigger logging time

Parameter
blockName = name of Logging block
postTriggerTime = post-trigger time in milliseconds

Returns
None

Availability
This function is supported in Version 4.1 and after.

Branch Compatibility
CANalyzer’s Transmit Branch = Yes
CANalyzer’s Analysis Branch = Yes
CANoe’s Simulation Branch = Yes
CANoe’s Analysis Branch = Yes

Related Functions
setLogFileName
setPostTrigger
setPreTrigger
startLogging
trigger
writeToLog
writeToLogEx

Example
//stops "blockname" with a post-trigger time of 2 seconds
stopLogging("blockname", 2000);
The CAPL Functions

**stopStatisticAcquisition**

**Syntax**
```c
void stopStatisticAcquisition();
```

**Description**
Stops an already started acquisition range in the Statistics window. If no acquisition range has been started yet, this function has no effect.

**Parameter**
None

**Returns**
None

**Availability**
This function is supported in Version 3.0 and after.

**Observation**
The CAPL program block this function appears must be located directly before the Statistics block in the Analysis Branch of CANalyzer and CANoe.

**Branch Compatibility**
- CANalyzer's Transmit Branch = No
- CANalyzer's Analysis Branch = Yes
- CANoe's Simulation Branch = No
- CANoe’s Analysis Branch = Yes

**Related Functions**
isStatisticAcquisitionRunning
startStatisticAcquisition

**Example**
```c
//Tests for a running acquisition range and stops it.
//If no statistical data acquisition is active a new one is started.
if(isStatisticAcquisitionRunning)
{
    //stops the running acquisition range
    stopStatisticAcquisition();
}
else
{
    //Starts a new acquisition range
    startStatisticAcquisition();
}
...
**strlen**

**Syntax**
```c
long strlen (char s[]);
```

**Description**
Determines the length of string s.

**Parameter**
s = string whose length we wish to find

**Returns**
Length of string

**Availability**
Available in all versions.

**Branch Compatibility**
- CANalyzer's Transmit Branch = Yes
- CANalyzer's Analysis Branch = Yes
- CANoe's Simulation Branch = Yes
- CANoe's Analysis Branch = Yes

**Related Functions**
eCount
strncat
strncmp
strncpy

**Example**
```c
... length = strlen("CANalyzer"); ...
//Result: length = 9
```
**strncat**

**Syntax**
```c
void strncat (char dest[], char src[], long len);
```

**Description**
Concatenates two strings into one.

**Parameter**
- `dest` = original string to be concatenated
- `src` = string to append
- `len` = the maximum length of the resulting string

**Returns**
None

**Availability**
Available in all versions.

**Observation**
The function ensures that there is a terminating '\0' in the destination string. Thus, a maximum number of characters minus 1 are copied.

**Branch Compatibility**
- CANalyzer’s Transmit Branch = Yes
- CANalyzer’s Analysis Branch = Yes
- CANoe’s Simulation Branch = Yes
- CANoe’s Analysis Branch = Yes

**Related Functions**
- `strlen`
- `strncmp`
- `strncpy`

**Example**
```c
#define __2内部控制__  
char s1[7] = "Vector";  
char s2[10] = "CANalyzer";  
strncat(s1, s2, 17);  
//Result: s1 = "VectorCANalyzer"
```
**strncmp**

**Syntax**

```c
void strncmp (char s1[], char s2[], long len);
```

**Description**

Compares two strings together up to a specific number of characters

**Parameter**

- `s1`, `s2` = strings to compare
- `len` = number of characters to compare

**Returns**

- `-1` = if `s1 < s2`
- `0` = if `s1 = s2`
- `1` = if `s2 > s1`

**Availability**

Available in all versions.

**Branch Compatibility**

- CANalyzer’s Transmit Branch = Yes
- CANalyzer’s Analysis Branch = Yes
- CANoe’s Simulation Branch = Yes
- CANoe’s Analysis Branch = Yes

**Related Functions**

- strlen
- strncat
- strncpy

**Example**

```c
if(strncmp(s1, s2, strlen(s1))
    write(“not equal”);
else
    write(“equal”);
...```
**Syntax**
```c
void strncpy (char dest[], char src[], long len);
```

**Description**
Copies one string to replace another up to a specific number of characters.

**Parameter**
- `dest` = original string to be replaced
- `src` = new string to copy
- `len` = number of characters to copy + 1

**Returns**
None

**Availability**
Available in all versions.

**Observation**
The function ensures that there is a terminating ‘\0’ in the destination string. Thus, a maximum number of characters minus 1 are copied.

**Branch Compatibility**
- CANalyzer's Transmit Branch = Yes
- CANalyzer's Analysis Branch = Yes
- CANoe's Simulation Branch = Yes
- CANoe's Analysis Branch = Yes

**Related Functions**
- `strlen`
- `strncat`
- `strncmp`

**Example**
```c
... char s1[7] = “Vector”;
char s2[10] = “CANalyzer”;
strncpy(s1, s2, strlen(s2) + 1);

//Result: s1 = “CANalyzer”
...```
The CAPL Functions


dword swapDWord (dword x);

Swaps four bytes of data.

\( x \) = value whose bytes are to be swapped

Value with bytes swapped

Available in all versions.

CAPL arithmetic follows the little-endian format (Intel). The function swaps bytes to transits to and from the big-endian format (Motorola).

CANalyzer’s Transmit Branch = Yes
CANalyzer’s Analysis Branch = Yes
CANoe’s Simulation Branch = Yes
CANoe’s Analysis Branch = Yes

swapInt
swapLong
swapWord

Example

dword value = 0x12345678;
write("%x", swapDWord(value));

//Result: 0x78563412
The CAPL Functions

swapInt

Syntax
int swapInt (int x);

Description
Swaps two bytes of data.

Parameter
x = value whose bytes are to be swapped

Returns
Value with bytes swapped

Availability
Available in all versions.

Observation
CAPL arithmetic follows the little-endian format (Intel). The function swaps bytes to transits to and from the big-endian format (Motorola).

Branch Compatibility
- CANalyzer’s Transmit Branch = Yes
- CANalyzer’s Analysis Branch = Yes
- CANoe’s Simulation Branch = Yes
- CANoe’s Analysis Branch = Yes

Related Functions
- swapDWord
- swapLong
- swapWord

Example
int value = 0x1234;
write("%x", swapInt(value));

//Result: 0x3412
swapLong

Syntax
long swapLong (long x);

Description
Swaps four bytes of data.

Parameter
x = value whose bytes are to be swapped

Returns
Value with bytes swapped

Availability
Available in all versions.

Observation
CAPL arithmetic follows the little-endian format (Intel). The function swaps bytes to transits to and from the big-endian format (Motorola).

Branch Compatibility
CANalyzer’s Transmit Branch = Yes
CANalyzer’s Analysis Branch = Yes
CANoe’s Simulation Branch = Yes
CANoe’s Analysis Branch = Yes

Related Functions
swapDWord
swapInt
swapWord

Example
long value = 0x12345678;
write("%x", swapLong(value));

//Result: 0x78563412
**swapWord**

**Syntax**

```c
word swapWord (word x);
```

**Description**

Swaps two bytes of data. CAPL arithmetic follows the little-endian format (Intel). The function swaps bytes to transits to and from the big-endian format (Motorola).

**Parameter**

- `x` = value whose bytes are to be swapped

**Returns**

Value with bytes swapped

**Availability**

Available in all versions.

**Observation**

CAPL arithmetic follows the little-endian format (Intel). The function swaps bytes to transits to and from the big-endian format (Motorola).

**Branch Compatibility**

- CANalyzer’s Transmit Branch = Yes
- CANalyzer’s Analysis Branch = Yes
- CANoe’s Simulation Branch = Yes
- CANoe’s Analysis Branch = Yes

**Related Functions**

- `swapDWord`
- `swapInt`
- `swapLong`

**Example**

```c
word value = 0x1234;
write("%x", swapWord(value));

//Result: 0x3412
```
The CAPL Functions

**sysExit**

**Syntax**

```c
void sysExit();
```

**Description**

Exits the system (CANalyzer or CANoe) from within a CAPL program.

**Parameter**

None

**Returns**

None

**Availability**

Available in all versions.

**Observation**

All captured data will be lost with an exception to the data already logged into a file.

**Branch Compatibility**

- CANalyzer’s Transmit Branch = No
- CANalyzer’s Analysis Branch = Yes
- CANoe’s Simulation Branch = No
- CANoe’s Analysis Branch = Yes

**Related Functions**

sysMinimize

**Example**

```c
...
sysExit();
...
```
sysMinimize

**Syntax**

```c
void sysMinimize();
```

**Description**

Minimizes or restores the application window of CANalyzer or CANoe.

**Parameter**

None

**Returns**

None

**Availability**

Available in all versions.

**Branch Compatibility**

- CANalyzer’s Transmit Branch = No
- CANalyzer’s Analysis Branch = Yes
- CANoe’s Simulation Branch = No
- CANoe’s Analysis Branch = Yes

**Related Functions**

- sysExit

**Example**

```c
...
    sysMinimize();
    ...
```
The CAPL Functions

timeDiff

Syntax
long timeDiff (message msg1, NOW);
long timeDiff (message msg1, message msg2);

Description
Calculates the time difference between messages or between a message and the current measurement time in ms.

Parameter
NOW = a keyword that represents current measurement time
msg1,msg2 = messages to get the time difference

Returns
Time difference in ms

Availability
Available in all versions.

Recommendation
The most precise can be access by the TIME message selector. The resolution return by this selector is in 10 microseconds (assigned by the CAN controller). The syntax is a message variable follow by a period and then the word “TIME”.

Branch Compatibility
CANalyzer's Transmit Branch = Yes
CANalyzer's Analysis Branch = Yes
CANoe's Simulation Branch = Yes
CANoe's Analysis Branch = Yes

Related Functions
getLocalTime
getLocalTimeString
timeNow

Example
long diff;
diff = timeDiff(m100, now); //old method
diff = timeNow() - m100.time; //new method
diff = m200.time - m100.time; //new method
timeNow

Syntax
   dword timeNow ();

Description
   Returns the current system time.

Parameter
   None

Returns
   Time since the start of the current measurement in units of 10 µsec.

Availability
   Available in all versions.

Observation
   This time is established with the help of the PC timer with a resolution of 1 msec.

Recommendation
   To get a precise time stamp of a message, use the TIME message selector. The resolution return
   by this selector is in 10 microseconds (assigned by the CAN controller). The syntax is a message
   variable follow by a period and then the word "TIME".

Branch Compatibility
   CANalyzer’s Transmit Branch = Yes
   CANalyzer’s Analysis Branch = Yes
   CANoe’s Simulation Branch = Yes
   CANoe’s Analysis Branch = Yes

Related Functions
   getLocalTime
   getLocalTimeString
   timeDiff
   timeNowFloat

Example
   float x;
   x = timeNow() / 100000.0; // current time in seconds
Syntax

dword timeNowFloat ();

Description
Returns the current system time in float.

Parameter
None

Returns
Time since the start of the current measurement in units of 10 µsec.

Availability
Available in all versions.

Observation
This time is established with the help of the PC timer with a resolution of 1 msec.

Recommendation
To get a precise time stamp of a message, use the TIME message selector. The resolution return by this selector is in 10 microseconds (assigned by the CAN controller). The syntax is a message variable follow by a period and then the word "TIME".

Branch Compatibility
CANalyzer’s Transmit Branch = Yes
CANalyzer’s Analysis Branch = Yes
CANoe’s Simulation Branch = Yes
CANoe’s Analysis Branch = Yes

Related Functions
getLocalTime
getLocalTimeString
timeDiff
timeNow

Example
float x;
x = timeNowFloat() / 100000.0;  // current time in seconds
trigger

Syntax
void trigger();

Description
Activates logging.

Parameter
None

Returns
None

Availability
Available in all versions.

Recommendation
The newer startLogging() and stopLogging() functions can handle logging more extensively.

Branch Compatibility
- CANalyzer’s Transmit Branch = Yes
- CANalyzer’s Analysis Branch = Yes
- CANoe’s Simulation Branch = Yes
- CANoe’s Analysis Branch = Yes

Related Functions
- setLogFileName
- setPostTrigger
- setPreTrigger
- startLogging
- stop
- stopLogging
- writeToLog
- writeToLogEx

Example
on message 100
{
    write("logging start");
    trigger(); //start logging
}
The CAPL Functions

valOfId

Syntax
long valOfId (dword id);
long valOfId (message m);

Description
Returns the value of a message identifier regardless its type. Useful function on extended
protocols.

Parameter
id = message identifier
m = message variable

Returns
Identifier as long value

Availability
Available in all versions.

Recommendation
It may be helpful sometimes just to use the ID message selector to access the message identifier.
The syntax is the name of the message follow by a period and then the word "ID".

Branch Compatibility
CANalyzer’s Transmit Branch = Yes
CANalyzer’s Analysis Branch = Yes
CANoe’s Simulation Branch = Yes
CANoe’s Analysis Branch = Yes

Related Functions
isExtId
isStdId
mkExtId

Example
on message *
{
  long id;
  id = valOfId(this);  //works with extended ID as well
  ...
}
**write**

**Syntax**
void write (char format[], ...);

**Description**
Outputs a text message to the Write window.

**Parameter**
The `write()` function allows a varying number of parameters. The format for the parameters is a format string containing string formatting expressions followed by zero or more arguments, each of which corresponds to one of the string formatting expressions shown below:

- `"%ld"` or `"%d"` = decimal display
- `"%bx"` or `"%x"` = hexadecimal display
- `"%bx"` or `"%X"` = hexadecimal display (with upper case letters)
- `"%lu"` or `"%u"` = unsigned display
- `"%lo"` or `"%o"` = octal display
- `"%g"` or `"%lf"` = floating point display
- `"%s"` = displays a string
- `"%c"` = displays a character
- `"%%"` = displays ‘%’ character

**Returns**
None

**Availability**
Available in all versions.

**Observation**
This function is identical to the `printf()` function used in the C language.

**Branch Compatibility**
- CANalyzer’s Transmit Branch = Yes
- CANalyzer’s Analysis Branch = Yes
- CANoe’s Simulation Branch = Yes
- CANoe’s Analysis Branch = Yes

**Related Functions**
- `snprintf`
- `writeClear`
- `writeCreate`
- `writeDestroy`
- `writeEx`
- `writeLineEx`
- `writeToLog`
- `writeToLogEx`

**Example**
```c
void display()
{
    int i = 10;
    int j = 25;

    write("d = %ld, h = 0x%lx",i,j);
}
//Result: “d = 10, h = 0x19”
```
writeClear

**Syntax**

```c
void writeClear (dword identifier);
```

**Description**

Clears the texts of a pane in the Write window except the All pane.

**Parameter**

- `identifier = 0` (System pane)
- `= 1` (CAPL pane)
- `= x` (pane identifier returned by function `writeCreate()`)

**Returns**

None

**Availability**

This function is supported in Version 3.2 and after.

**Branch Compatibility**

- CANalyzer’s Transmit Branch = Yes
- CANalyzer’s Analysis Branch = Yes
- CANoe’s Simulation Branch = Yes
- CANoe’s Analysis Branch = Yes

**Related Functions**

- `write`
- `writeCreate`
- `writeDestroy`
- `writeEx`
- `writeLineEx`

**Example**

```c
... writeClear(1); //clears the CAPL pane in the Write window
...```
writeCreate

**Syntax**

dword writeCreate (char name[]);

**Description**

Creates a new pane in the Write window.

**Parameter**

name = the name of the new pane

**Returns**

Identifier to this new pane

**Availability**

This function is supported in Version 3.2 and after.

**Branch Compatibility**

CANalyzer’s Transmit Branch = Yes  
CANalyzer’s Analysis Branch = Yes  
CANoe’s Simulation Branch = Yes  
CANoe’s Analysis Branch = Yes

**Related Functions**

write  
writeClear  
writeDestroy  
writeEx  
writeLineEx

**Example**

```c
int x;
x = writeCreate(“CAPL2”);  //creates the CAPL2 pane  
//output its identifier to this new window  
writeLineEx(x, 1, “CAPL2 identifier = %d”, x);
```
writeDbgLevel

Syntax
void writeDbgLevel (unsigned int priority, char format1[], char format2[], ...);

Description
Outputs a message to the write window after a priority check with the node. The priority level can
be set for every network node using the setWriteDbgLevel() function.

Parameter

priority = output priority from 0 to 15
format = format string, variables or expressions

Legal format expressions:
“%Id” or “%d” = decimal display
“%Ix” or “%x” = hexadecimal display
“%Ix” or “%X” = hexadecimal display (with upper case letters)
“%Iu” or “%u” = unsigned display
“%Io” or “%o” = octal display
“%g” or “%If” = floating point display
“%s” = displays a string
“%c” = displays a character
“%" = displays ‘%’ character

Returns
None

Availability
This function is supported in Version 3.0 and after.

Observation
This function can be used for debugging to vary the output to the write window.

Branch Compatibility
CANalyzer’s Transmit Branch = No
CANalyzer’s Analysis Branch = No
CANoe’s Simulation Branch = Yes
CANoe’s Analysis Branch = No

Related Functions
setWriteDbgLevel
write

Example
int i = 10;
int j = 12;

setWriteDbgLevel(7); //sets priority for this node

writeDbgLevel(4, "This is shown: h= %lxh", j);
//Result: This is shown: h= 0ch

writeDbgLevel(9, "This is not shown: d= %ld", i);
//No output
writeDestroy

Syntax
void writeDestroy (dword identifier);

Description
Removes a user-defined pane from the Write Window.

Parameter
identifier = identifier to the pane previously returned by the writeCreate() function

Returns
None

Availability
This function is supported in Version 3.2 and after.

Observation
That pane must be created by the writeCreate() function.

Branch Compatibility
  CANalyzer’s Transmit Branch = Yes
  CANalyzer’s Analysis Branch = Yes
  CANoe’s Simulation Branch = Yes
  CANoe’s Analysis Branch = Yes

Related Functions
write
writeClear
writeCreate
writeEx
writeLineEx

Example
int x;
x = writeCreate(“CAPL2”); //creates the CAPL2 pane
//removes the CAPL2 pane
writeDestroy(x);
writeEx

Syntax
void writeEx (dword identifier, dword severity, char format[], ...);

Description
Writes to a Write window without first executing a line feed.

Parameter
identifier = pane identifier of the Write window (can be user-defined pane)
-3 = all Trace windows
-2 = write to log file
-1 = CAPL pane
0 = System pane
x = pane identifier returned by writeCreate()

severity = type of message (no effect when writing to a Trace window)
0 = success
1 = information
2 = warning
3 = error

Returns
None

Availability
This function is supported in Version 3.2 and after.

Observation
For writing to a log file, severity = 0 means write with comments and severity = 1 means write without comments.

Branch Compatibility
CANalyzer’s Transmit Branch = Yes
CANalyzer’s Analysis Branch = Yes
CANoe’s Simulation Branch = Yes
CANoe’s Analysis Branch = Yes

Related Functions
write
writeClear
writeCreate
writeDestroy
writeLineEx

Example
int x;
x = writeCreate("CAPL2"); //creates the CAPL2 pane
//writes to the CAPL2 pane without line feed
writeEx(x, 1, "Window ID = %d", x);
writeLineEx

Syntax
void writeLineEx (dword identifier, dword severity, char format[], ...);

Description
Writes to a Write window by first executing a line feed.

Parameter
identifier = pane identifier of the Write window (can be user-defined pane)
-3 = all Trace windows
-2 = write to log file
-1 = CAPL pane
0 = System pane
x = pane identifier returned by writeCreate()

severity = type of message (no effect when writing to a Trace window)
0 = success
1 = information
2 = warning
3 = error

Returns
None

Availability
This function is supported in Version 3.2 and after.

Observation
For writing to a log file, severity = 0 means write with comments and severity = 1 means write without comments.

Branch Compatibility
CANalyzer’s Transmit Branch = Yes
CANalyzer’s Analysis Branch = Yes
CANoe’s Simulation Branch = Yes
CANoe’s Analysis Branch = Yes

Related Functions
write
writeClear
writeCreate
writeDestroy
writeEx

Example
int x;
x = writeCreate(“CAPL2”); //creates the CAPL2 pane

//write to the CAPL2 pane with line feed:
writeLineEx(x, 1, “Window ID = %d”, x);
writeProFileFloat

Syntax
long writeProFileFloat (char section[], char entry[], float value, char filename[]);

Description
Writes a float value to an INI-formatted file. Any existing value will be overwritten.

Parameter
  section = section within file
  entry = name of variable
  value = float value to write
  filename = name of data file

Returns
  0 = unsuccessful
  1 = successful

Availability
This function is supported in Version 3.0 and after.

Observation
The file path is set by either the setWritePath() or setFilePath() function. If neither function is used, the data file must be located either in the same directory as the databases file(s) or configuration file(s) of CANalyzer/CANoe.

Branch Compatibility
  CANalyzer’s Transmit Branch = Yes
  CANalyzer’s Analysis Branch = Yes
  CANoe’s Simulation Branch = Yes
  CANoe’s Analysis Branch = Yes

Related Functions
  fileClose
  fileGetBinaryBlock
  fileGetString
  fileGetStringSZ
  filePutString
  fileRewind
  fileWriteBinaryBlock
  getProFileArray
  getProFileFloat
  getProFileInt
  getProFileString
  openFileRead
  openFileWrite
  setFilePath
  setWritePath
  writeProFileInt
  writeProFileString

Example
long val = 149.5;

  //assigns 149.5 to the Weight entry
  writeProFileFloat("Input", "Weight", val, "Test.txt");
writeProFileInt

Syntax
long writeProFileInt (char section[], char entry[], long value, char filename[]);

Description
Writes an integer value to an INI-formatted file. Any existing value will be overwritten.

Parameter
section = section within file
entry = name of variable
value = integer value to write
filename = name of data file

Returns
0 = unsuccessful
1 = successful

Availability
This function is supported in Version 3.0 and after.

Observation
The file path is set by either the setWritePath() or setFilePath() function. If neither function is used, the data file must be located either in the same directory as the databases file(s) or configuration file(s) of CANalyzer/CANoe.

Branch Compatibility
CANalyzer's Transmit Branch = Yes
CANalyzer's Analysis Branch = Yes
CANoe's Simulation Branch = Yes
CANoe's Analysis Branch = Yes

Related Functions
fileClose
fileGetBinaryBlock
fileGetString
fileGetStringSZ
filePutString
fileRewind
fileWriteBinaryBlock
getProFileArray
getProFileFloat
getProFileInt
getProFileString
openFileRead
openFileWrite
setFilePath
setWritePath
writeProFileFloat
writeProFileName

Example
long val = 20;

//assign 20 to the Age entry
writeProFileInt("Input", "Age", val, "Test.txt");
writeProFileString

Syntax
long writeProFileString (char section[], char entry[], char value[],
char filename[]);

Description
Writes a string value to an INI-formatted file. Any existing value will be overwritten.

Parameter
section = section within file
entry = name of variable
value = string value to write
filename = name of data file

Returns
0 = unsuccessful
1 = successful

Availability
This function is supported in Version 3.0 and after.

Observation
The file path is set by either the setWritePath() or setFilePath() function. If neither function is used,
the data file must be located either in the same directory as the databases file(s) or configuration
file(s) of CANalyzer/CANoe.

Branch Compatibility
CANalyzer's Transmit Branch = Yes
CANalyzer's Analysis Branch = Yes
CANoe's Simulation Branch = Yes
CANoe's Analysis Branch = Yes

Related Functions
fileClose
fileGetBinaryBlock
fileGetString
fileGetStringSZ
filePutString
fileRewind
fileWriteBinaryBlock
getProFileArray
getProFileFloat
getProFileInt
getProFileString
openFileRead
openFileWrite
setFilePath
setWritePath
writeProFileFloat
writeProFileInt

Example
char cname[7] = "MyName";

//assign "MyName to the Name entry
writeProFileString("Input", "Name", cname, "Test.txt");
writeTextBkgColor

Syntax
void writeTextBkgColor (dword paneID, dword red, dword green, dword blue);

Description
Sets the background color of a specific pane in the Write window. That pane may be created by the writeCreate() function.

Parameter
paneID = identifier to the pane previously returned by the writeCreate() function
= 0 (System pane messages)
= 1 (CAPL pane messages)
red = intensity of the red color (0 to 255)
green = intensity of the green color (0 to 255)
blue = intensity of the blue color (0 to 255)

Returns
None

Availability
This function is supported in Version 5.0 and after.

Observation
Background color can be changed in a new pane created by the writeCreate() function.

Branch Compatibility
CANalyzer's Transmit Branch = Yes
CANalyzer's Analysis Branch = Yes
CANoe's Simulation Branch = Yes
CANoe's Analysis Branch = Yes

Related Functions
write
writeClear
writeCreate
writeDestroy
writeEx
writeLineEx
writeTextColor

Example
//Change CAPL messages background to red
writeTextBkgColor(1, 255, 0, 0);
writeTextColor

Syntax

void writeTextColor (dword paneID, dword red, dword green, dword blue);

Description

Sets the text color of a specific pane in the Write window. That pane may be created by the writeCreate() function.

Parameter

- paneID = identifier to the pane previously returned by the writeCreate() function
  - = 0 (System pane messages)
  - = 1 (CAPL pane messages)
- red = intensity of the red color (0 to 255)
- green = intensity of the green color (0 to 255)
- blue = intensity of the blue color (0 to 255)

Returns

None

Availability

This function is supported in Version 5.0 and after.

Observation

Text color can be changed in a new pane created by the writeCreate() function.

Branch Compatibility

- CANalyzer’s Transmit Branch = Yes
- CANalyzer’s Analysis Branch = Yes
- CANoe’s Simulation Branch = Yes
- CANoe’s Analysis Branch = Yes

Related Functions

- write
- writeClear
- writeCreate
- writeDestroy
- writeEx
- writeLineEx
- writeTextBkgColor

Example

//Change CAPL messages to red
writeTextColor(1, 255, 0, 0);
writeToLog

**Syntax**

```c
void writeToLog (char format[], ...);
```

**Description**

Writes an output string to an ASCII logging file. Since the compiler cannot check the format string, illegal format entries will lead to undefined results.

**Parameter**

The `writeToLog()` function allows various parameters. Since this function is based on the C function "printf", the format for the parameters is a format string containing string-formatted expressions followed by zero or more arguments, each of which corresponds to one of the string formatting expressions. The string formatting expressions are shown below:

- "%Id" or "%d" = decimal display
- "%Ix" or "%x" = hexadecimal display
- "%IX" or "%X" = hexadecimal display (with upper case letters)
- "%Iu" or "%u" = unsigned display
- "%Io" or "%o" = octal display
- "%g" or "%If" = floating point display
- "%%" = displays '%' character
- "%s" = displays a string
- "%c" = displays a character
- "%%" = displays '%' character

**Returns**

None

**Availability**

Available in all versions.

**Observation**

Data is only written to a log file when logging is enabled in CANalyzer or CANoe. A call to this function is ignored when logging is disabled.

**Recommendation**

Use the `writeToLogEx()` function to output without a timestamp and the comment characters "//".

**Branch Compatibility**

- CANalyzer’s Transmit Branch = Yes
- CANalyzer’s Analysis Branch = Yes
- CANoe’s Simulation Branch = Yes
- CANoe’s Analysis Branch = Yes

**Related Functions**

- setLogFileName
- setPostTrigger
- setPreTrigger
- snprintf
- startLogging
- stopLogging
- trigger
- write
- writeToLogEx
Example
void MarkLogFile(int marker)
{
    //marks line of ASCII logging file with an integer:
    writeToLog("Marker Number = %d", marker);
}

//Result of calling MarkLogFile(3) once as shown in an ASCII log file:
// 1.2632  Marker Number = 3
**writeToLogEX**

**Syntax**

```c
void writeToLogEX (char format[], ...);
```

**Description**

Writes an output string to an ASCII logging file. Since the compiler cannot check the format string, illegal format entries will lead to undefined results.

**Parameter**

The `writeToLogEX()` function allows various parameters. Since this function is based on the C function "printf", the format for the parameters is a format string containing string-formatted expressions followed by zero or more arguments, each of which corresponds to one of the string formatting expressions. The string formatting expressions are shown below:

- "%Id" or "%d" = decimal display
- "%Ix" or "%x" = hexadecimal display
- "%IX" or "%X" = hexadecimal display (with upper case letters)
- "%Iu" or "%u" = unsigned display
- "%Io" or "%o" = octal display
- "%g" or "%If" = floating point display
- "%s" = displays a string
- "%c" = displays a character
- "%%" = displays ‘%’ character

**Returns**

None

**Availability**

This function is supported in Version 3.0 and after.

**Observation**

Data is only written to a log file when logging is enabled in CANalyzer or CANoe. A call to this function is ignored when logging is disabled.

**Recommendation**

Use the `writeToLog()` function to output with a timestamp and the comment characters "//".

**Branch Compatibility**

- CANalyzer’s Transmit Branch = Yes
- CANalyzer’s Analysis Branch = Yes
- CANoe’s Simulation Branch = Yes
- CANoe’s Analysis Branch = Yes

**Related Functions**

- setLogFileName
- setPostTrigger
- setPreTrigger
- snprintf
- startLogging
- stopLogging
- trigger
- write
- writeToLog
Example

```c
void MarkLogFileEX(int marker)
{
    //marks line of ASCII logging file with an integer:
    writeToLogEX("Marker Number = %d", marker);
}

//Result of calling MarkLogFileEx(3) once as shown in an ASCII log file:
Marker Number = 3
```
## Compatibility Chart

<table>
<thead>
<tr>
<th>Functions</th>
<th>Real-Time Branches</th>
<th>Analysis Branches</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CANalyzer (Windows)</td>
<td>CANoe Simulation Window</td>
</tr>
<tr>
<td></td>
<td>CANoe Analysis Branch</td>
<td>CANoe Measurement Setup Window</td>
</tr>
<tr>
<td>abs</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>atol</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>beep</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>callAllOnEnvVar</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>cancelTimer</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>canOffline</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>canOnline</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>canSetChannelAcc</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>canSetChannelMode</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>canSetChannelOutput</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>cos</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>elCount</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>enableControl</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>exp</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>fileClose</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>fileGetBinaryBlock</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>fileGetString</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>fileGetStringSZ</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>fileName</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>filePutString</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>fileReadArray</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>fileReadFloat</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>fileRewind</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>fileWriteBinaryBlock</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>fileWriteFloat</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>fileWriteInt</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>fileWriteString</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>getBusContext</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>getBusNameContext</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>getCardType</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>getCardTypeEx</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>getChipType</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Function</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>-----------------------------------------</td>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td>getDrift</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>getFirstCANdbName</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>getJitterMax</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>getJitterMin</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>getLocalTime</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>getLocalTimeString</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>getMessageAttrInt</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>getMessageName</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>getNextCANdbName</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>getProfileArray</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>getProfileInt</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>getProfileFloat</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>getProfileString</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>getStartdelay</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>getValue</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>getValueSize</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>halt</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>inpport</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>inpportLPT</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>inspect</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>isExtld</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>isStatisticAcquisitionRunning</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>isStdld</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>keypressed</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>itoa</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>makeRGB</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>mkExtld</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>msgBeep</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>openFileRead</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>openFileWrite</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>output</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>outputLPT</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>putValue</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>putValueToControl</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>random</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>replayResume</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>replayStart</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>replayState</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>replayStop</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>replaySuspend</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>resetCan</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>resetCanEx</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>runError</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
## Compatibility Chart

<table>
<thead>
<tr>
<th>Function</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>seqFileClose</td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>seqFileGetBlock</td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>seqFileGetLine</td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>seqFileGetLineSZ</td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>seqFileLoad</td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>seqFileRewind</td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>setBtr</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>setBusContext</td>
<td></td>
<td></td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>setCanCabsMode</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>setControlBackColor</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>setControlForeColor</td>
<td></td>
<td></td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>setCanCabsMode</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>setControlBackColor</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>setControlForeColor</td>
<td></td>
<td></td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>setDrift</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>setFilePath</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>setJitter</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>setLogFile</td>
<td></td>
<td></td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>setMsgTime</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>setOcr</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>setPortBit</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>setPostTrigger</td>
<td>Yes</td>
<td></td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>setPreTrigger</td>
<td>Yes</td>
<td></td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>setStartDelay</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>setTimer</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>setWriteDbgLevel</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>setWritePath</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>sin</td>
<td></td>
<td></td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>snprintf</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>sqrt</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>startLogging</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>startStatisticAcquisition</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>stop</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>stopLogging</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>stopStatisticAcquisition</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>strlen</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>strncat</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>strncmp</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>strncpy</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>swapDWord</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>swapInt</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>swapLong</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>swapWord</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>sysExit</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>sysMinimize</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>timeDiff</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Function</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>timeNow</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>timeNowFloat</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>trigger</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>valOfld</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>write</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>writeClear</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>writeCreate</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>writeDbgLevel</td>
<td></td>
<td></td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>writeDestroy</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>writeEx</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>writeLineEx</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>writeProFileFloat</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>writeProFileInt</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>writeProFileString</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>writeTextBkgColor</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>writeTextColor</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>writeToLog</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>writeToLogEx</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

* = function can be used if CAN.INI is configured correctly.

Table x – CAPL Function Compatibilities
## Availability Chart

<table>
<thead>
<tr>
<th>Functions</th>
<th>Status</th>
<th>Versions Supported</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>S = Supported</td>
<td>O = Obsolete</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>abs</td>
<td>S</td>
<td>2.5 and after</td>
</tr>
<tr>
<td>atol</td>
<td>S</td>
<td>All</td>
</tr>
<tr>
<td>beep</td>
<td>O</td>
<td>Prior to 3.0</td>
</tr>
<tr>
<td>callAllOnEnvVar</td>
<td>S</td>
<td>All</td>
</tr>
<tr>
<td>cancelTimer</td>
<td>S</td>
<td>All</td>
</tr>
<tr>
<td>canOffline</td>
<td>S</td>
<td>All</td>
</tr>
<tr>
<td>canOnline</td>
<td>S</td>
<td>All</td>
</tr>
<tr>
<td>canSetChannelAcc</td>
<td>S</td>
<td>5.0 and after</td>
</tr>
<tr>
<td>canSetChannelMode</td>
<td>S</td>
<td>5.0 and after</td>
</tr>
<tr>
<td>canSetChannelOutput</td>
<td>S</td>
<td>5.0 and after</td>
</tr>
<tr>
<td>cos</td>
<td>S</td>
<td>All</td>
</tr>
<tr>
<td>elCount</td>
<td>S</td>
<td>All</td>
</tr>
<tr>
<td>enableControl</td>
<td>S</td>
<td>4.1 and after</td>
</tr>
<tr>
<td>exp</td>
<td>S</td>
<td>All</td>
</tr>
<tr>
<td>fileClose</td>
<td>S</td>
<td>3.0 and after</td>
</tr>
<tr>
<td>fileGetBinaryBlock</td>
<td>S</td>
<td>3.0 and after</td>
</tr>
<tr>
<td>fileGetString</td>
<td>S</td>
<td>3.0 and after</td>
</tr>
<tr>
<td>fileGetStringSZ</td>
<td>S</td>
<td>3.0 and after</td>
</tr>
<tr>
<td>fileName</td>
<td>S</td>
<td>All</td>
</tr>
<tr>
<td>filePutString</td>
<td>S</td>
<td>3.0 and after</td>
</tr>
<tr>
<td>fileReadArray</td>
<td>O</td>
<td>Prior to 3.0</td>
</tr>
<tr>
<td>fileReadFloat</td>
<td>O</td>
<td>Prior to 3.0</td>
</tr>
<tr>
<td>fileReadInt</td>
<td>O</td>
<td>Prior to 3.0</td>
</tr>
<tr>
<td>fileReadString</td>
<td>O</td>
<td>Prior to 3.0</td>
</tr>
<tr>
<td>fileRewind</td>
<td>S</td>
<td>Prior to 3.0</td>
</tr>
<tr>
<td>fileWriteBinaryBlock</td>
<td>S</td>
<td>3.0 and after</td>
</tr>
<tr>
<td>fileWriteFloat</td>
<td>O</td>
<td>Prior to 3.0</td>
</tr>
<tr>
<td>fileWriteInt</td>
<td>O</td>
<td>Prior to 3.0</td>
</tr>
<tr>
<td>fileWriteString</td>
<td>O</td>
<td>Prior to 3.0</td>
</tr>
<tr>
<td>getBusContext</td>
<td>S</td>
<td>3.2 and after</td>
</tr>
<tr>
<td>getBusNameContext</td>
<td>S</td>
<td>3.2 and after</td>
</tr>
<tr>
<td>getCardType</td>
<td>S</td>
<td>All</td>
</tr>
<tr>
<td>getCardTypeEx</td>
<td>S</td>
<td>5.0 and after</td>
</tr>
<tr>
<td>getChipType</td>
<td>S</td>
<td>All</td>
</tr>
<tr>
<td>getDrift</td>
<td>S</td>
<td>3.0 and after</td>
</tr>
<tr>
<td>getFirstCANdbName</td>
<td>S</td>
<td>4.0 and after</td>
</tr>
<tr>
<td>Function</td>
<td>Availability</td>
<td>Notes</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>--------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>getJitterMax</td>
<td>S</td>
<td>3.0 and after</td>
</tr>
<tr>
<td>getJitterMin</td>
<td>S</td>
<td>3.0 and after</td>
</tr>
<tr>
<td>getLocalTime</td>
<td>S</td>
<td>All</td>
</tr>
<tr>
<td>getLocalTimeString</td>
<td>S</td>
<td>All</td>
</tr>
<tr>
<td>getMessageAttrInt</td>
<td>S</td>
<td>3.1 and after</td>
</tr>
<tr>
<td>getMessageName</td>
<td>S</td>
<td>4.0 and after</td>
</tr>
<tr>
<td>getNextCANdbName</td>
<td>S</td>
<td>4.0 and after</td>
</tr>
<tr>
<td>getProFileArray</td>
<td>S</td>
<td>3.0 and after</td>
</tr>
<tr>
<td>getProFileInt</td>
<td>S</td>
<td>3.0 and after</td>
</tr>
<tr>
<td>getProFileFloat</td>
<td>S</td>
<td>3.0 and after</td>
</tr>
<tr>
<td>getProFileString</td>
<td>S</td>
<td>3.0 and after</td>
</tr>
<tr>
<td>getStartdelay</td>
<td>S</td>
<td>3.0 and after</td>
</tr>
<tr>
<td>getValue</td>
<td>S</td>
<td>All</td>
</tr>
<tr>
<td>getValueSize</td>
<td>S</td>
<td>All</td>
</tr>
<tr>
<td>halt</td>
<td>S</td>
<td>4.1 and after</td>
</tr>
<tr>
<td>inport</td>
<td>S</td>
<td>All</td>
</tr>
<tr>
<td>inportLPT</td>
<td>S</td>
<td>3.1 and after</td>
</tr>
<tr>
<td>inspect</td>
<td>S</td>
<td>4.1 and after</td>
</tr>
<tr>
<td>isExtId</td>
<td>S</td>
<td>All</td>
</tr>
<tr>
<td>isStatisticAcquisitionRunning</td>
<td>S</td>
<td>3.0 and after</td>
</tr>
<tr>
<td>isStdId</td>
<td>S</td>
<td>All</td>
</tr>
<tr>
<td>keypressed</td>
<td>S</td>
<td>All</td>
</tr>
<tr>
<td>ltoa</td>
<td>S</td>
<td>All</td>
</tr>
<tr>
<td>makeRGB</td>
<td>S</td>
<td>4.1 and after</td>
</tr>
<tr>
<td>mkExtId</td>
<td>S</td>
<td>All</td>
</tr>
<tr>
<td>msgBeep</td>
<td>S</td>
<td>3.0 and after</td>
</tr>
<tr>
<td>openFileRead</td>
<td>S</td>
<td>3.0 and after</td>
</tr>
<tr>
<td>openFileWrite</td>
<td>S</td>
<td>3.0 and after</td>
</tr>
<tr>
<td>output</td>
<td>S</td>
<td>All</td>
</tr>
<tr>
<td>outportLPT</td>
<td>S</td>
<td>3.1 and after</td>
</tr>
<tr>
<td>output</td>
<td>S</td>
<td>All</td>
</tr>
<tr>
<td>putValue</td>
<td>S</td>
<td>All</td>
</tr>
<tr>
<td>putValueToControl</td>
<td>S</td>
<td>4.0 and after</td>
</tr>
<tr>
<td>random</td>
<td>S</td>
<td>All</td>
</tr>
<tr>
<td>replayResume</td>
<td>S</td>
<td>4.0 and after</td>
</tr>
<tr>
<td>replayStart</td>
<td>S</td>
<td>4.0 and after</td>
</tr>
<tr>
<td>replayState</td>
<td>S</td>
<td>4.0 and after</td>
</tr>
<tr>
<td>replayStop</td>
<td>S</td>
<td>4.0 and after</td>
</tr>
<tr>
<td>replaySuspend</td>
<td>S</td>
<td>4.0 and after</td>
</tr>
<tr>
<td>resetCan</td>
<td>S</td>
<td>All</td>
</tr>
<tr>
<td>resetCanEx</td>
<td>S</td>
<td>4.1 and after</td>
</tr>
<tr>
<td>runError</td>
<td>S</td>
<td>All</td>
</tr>
<tr>
<td>seqFileClose</td>
<td>O</td>
<td>Prior to 3.0</td>
</tr>
<tr>
<td>seqFileGetBlock</td>
<td>O</td>
<td>Prior to 3.0</td>
</tr>
<tr>
<td>Function</td>
<td>Availability</td>
<td></td>
</tr>
<tr>
<td>----------------------------------</td>
<td>--------------------</td>
<td></td>
</tr>
<tr>
<td>seqFileGetLine</td>
<td>O</td>
<td>Prior to 3.0</td>
</tr>
<tr>
<td>seqFileGetLineSZ</td>
<td>O</td>
<td>Prior to 3.0</td>
</tr>
<tr>
<td>seqFileLoad</td>
<td>O</td>
<td>Prior to 3.0</td>
</tr>
<tr>
<td>seqFileRewind</td>
<td>O</td>
<td>Prior to 3.0</td>
</tr>
<tr>
<td>setBtr</td>
<td>S</td>
<td>All</td>
</tr>
<tr>
<td>setBusContext</td>
<td>S</td>
<td>3.2 and after</td>
</tr>
<tr>
<td>setCanCabsMode</td>
<td>S</td>
<td>4.1 and after</td>
</tr>
<tr>
<td>setControlBackColor</td>
<td>S</td>
<td>4.1 and after</td>
</tr>
<tr>
<td>setControlForeColor</td>
<td>S</td>
<td>4.1 and after</td>
</tr>
<tr>
<td>setControlProperty</td>
<td>S</td>
<td>4.1 and after</td>
</tr>
<tr>
<td>setDrift</td>
<td>S</td>
<td>3.0 and after</td>
</tr>
<tr>
<td>setFilePath</td>
<td>S</td>
<td>4.1 and after</td>
</tr>
<tr>
<td>setJitter</td>
<td>S</td>
<td>3.0 and after</td>
</tr>
<tr>
<td>setLogFileName</td>
<td>S</td>
<td>All</td>
</tr>
<tr>
<td>setMsgTime</td>
<td>O</td>
<td>Prior to 2.5</td>
</tr>
<tr>
<td>setOcr</td>
<td>S</td>
<td>All</td>
</tr>
<tr>
<td>setPortBit</td>
<td>S</td>
<td>4.1 and after</td>
</tr>
<tr>
<td>setPostTrigger</td>
<td>S</td>
<td>All</td>
</tr>
<tr>
<td>setPreTrigger</td>
<td>S</td>
<td>All</td>
</tr>
<tr>
<td>setStartdelay</td>
<td>S</td>
<td>3.0 and after</td>
</tr>
<tr>
<td>setTimer</td>
<td>S</td>
<td>All</td>
</tr>
<tr>
<td>setWriteDbgLevel</td>
<td>S</td>
<td>3.1 and after</td>
</tr>
<tr>
<td>setWritePath</td>
<td>S</td>
<td>3.0 and after</td>
</tr>
<tr>
<td>sin</td>
<td>S</td>
<td>All</td>
</tr>
<tr>
<td>snprintf</td>
<td>S</td>
<td>All</td>
</tr>
<tr>
<td>sqrt</td>
<td>S</td>
<td>All</td>
</tr>
<tr>
<td>startLogging</td>
<td>S</td>
<td>4.1 and after</td>
</tr>
<tr>
<td>startStatisticAcquisition</td>
<td>S</td>
<td>3.0 and after</td>
</tr>
<tr>
<td>stop</td>
<td>S</td>
<td>All</td>
</tr>
<tr>
<td>stopLogging</td>
<td>S</td>
<td>4.1 and after</td>
</tr>
<tr>
<td>stopStatisticAcquisition</td>
<td>S</td>
<td>3.0 and after</td>
</tr>
<tr>
<td>strlen</td>
<td>S</td>
<td>All</td>
</tr>
<tr>
<td>strncat</td>
<td>S</td>
<td>All</td>
</tr>
<tr>
<td>strncmp</td>
<td>S</td>
<td>All</td>
</tr>
<tr>
<td>strncpy</td>
<td>S</td>
<td>All</td>
</tr>
<tr>
<td>swapDWord</td>
<td>S</td>
<td>All</td>
</tr>
<tr>
<td>swapInt</td>
<td>S</td>
<td>All</td>
</tr>
<tr>
<td>swapLong</td>
<td>S</td>
<td>All</td>
</tr>
<tr>
<td>swapWord</td>
<td>S</td>
<td>All</td>
</tr>
<tr>
<td>sysExit</td>
<td>S</td>
<td>All</td>
</tr>
<tr>
<td>sysMinimize</td>
<td>S</td>
<td>All</td>
</tr>
<tr>
<td>timeDiff</td>
<td>S</td>
<td>All</td>
</tr>
<tr>
<td>timeNow</td>
<td>S</td>
<td>All</td>
</tr>
<tr>
<td>timeNowFloat</td>
<td>S</td>
<td>All</td>
</tr>
<tr>
<td>trigger</td>
<td>S</td>
<td>All</td>
</tr>
<tr>
<td>--------------</td>
<td>--------</td>
<td>----------------</td>
</tr>
<tr>
<td>valOfId</td>
<td>S</td>
<td>All</td>
</tr>
<tr>
<td>write</td>
<td>S</td>
<td>All</td>
</tr>
<tr>
<td>writeClear</td>
<td>S</td>
<td>3.2 and after</td>
</tr>
<tr>
<td>writeCreate</td>
<td>S</td>
<td>3.2 and after</td>
</tr>
<tr>
<td>writeDbgLevel</td>
<td>S</td>
<td>3.0 and after</td>
</tr>
<tr>
<td>writeDestroy</td>
<td>S</td>
<td>3.2 and after</td>
</tr>
<tr>
<td>writeEx</td>
<td>S</td>
<td>3.2 and after</td>
</tr>
<tr>
<td>writeLineEx</td>
<td>S</td>
<td>3.2 and after</td>
</tr>
<tr>
<td>writeProFileFloat</td>
<td>S</td>
<td>3.0 and after</td>
</tr>
<tr>
<td>writeProFileInt</td>
<td>S</td>
<td>3.0 and after</td>
</tr>
<tr>
<td>writeProFileString</td>
<td>S</td>
<td>3.0 and after</td>
</tr>
<tr>
<td>writeTextBkgColor</td>
<td>S</td>
<td>5.0 and after</td>
</tr>
<tr>
<td>writeTextColor</td>
<td>S</td>
<td>5.0 and after</td>
</tr>
<tr>
<td>writeToLog</td>
<td>S</td>
<td>All</td>
</tr>
<tr>
<td>writeToLogEx</td>
<td>S</td>
<td>3.0 and after</td>
</tr>
</tbody>
</table>