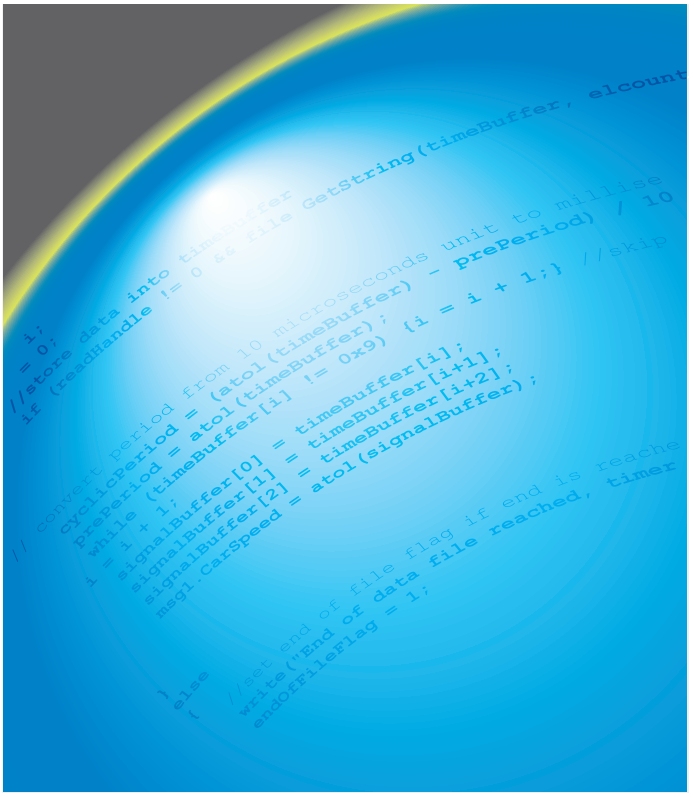
CAPL Function Reference Manual



CANalyzer
CANoe

vector
the art of engineering

CAPL Function Reference Manual

November 23, 2004

Second printing

Vector CANtech, Inc.
Suite 550
39500 Orchard Hill Place
Novi, MI 48375
USA
http://www.vector-cantech.com

© 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

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 ina 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 strikethough 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 strikethough 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.

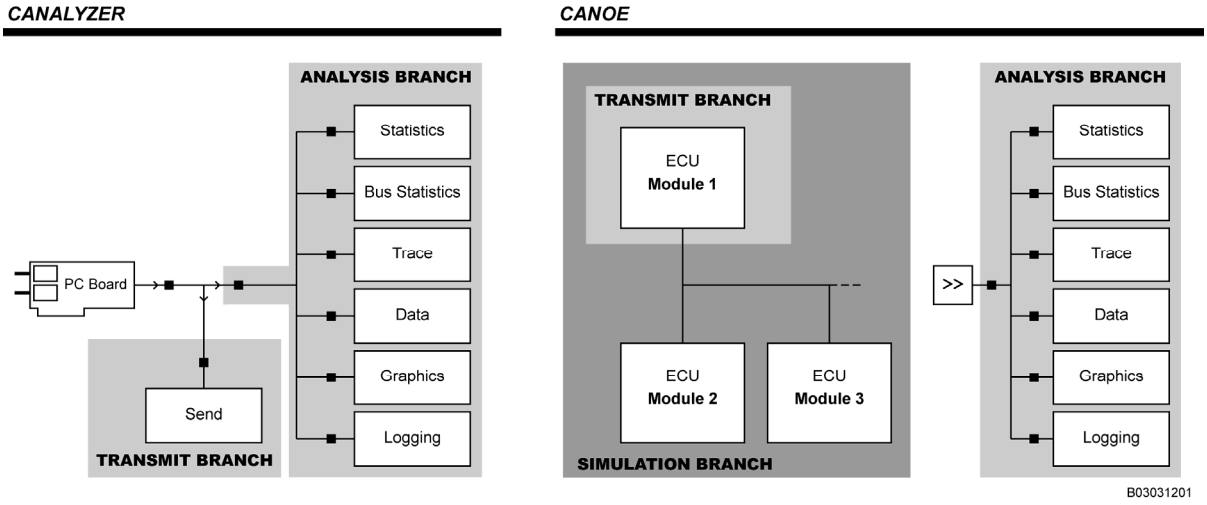


Figure 1 – CAPL Functions Depend on Placement

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.

abs

Syntax

```
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

```
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.
```

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
```

~~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);
}
```

callAllOnEnvVar

Syntax

```
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

```
on start
{
   callAllOnEnvVar();
}
```

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

```
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

Form2 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 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

```
dword var;

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

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

Example

```
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, 0x8FFFFFFF, 0x8FFFFFFF);

//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

```
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 acknowledge. 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

```
//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
```
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);
```

exp

Syntax

```
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

```
double x;
x = exp(1.0);  // e¹
               // 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

```
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

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

fileGetStringSZ

Syntax

```
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

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

```
}
```

fileName

Syntax

```
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

```
...
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
fileGetBinaryBlock
fileGetString
fileGetStringSZ
fileRewind
fileWriteBinaryBlock
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

```
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.
```

~~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
```
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
```
//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

```
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
```
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
```
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
```

getBusContext

Syntax

```
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

```
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

```
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;
}
```

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

getCardType
getCardTypeEx

Example

```
switch(getChipType())
{
      case 200: setOcr(0,0x02);  //Philips PCA82C200
         break;
      ...
      default: write("Unknown CAN chip type: %d", getChipType());
         break;
}
```

getDrift

Syntax

```
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

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

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

getJitterMax

Syntax

```
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

```
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[3] = Day of the month (1 – 31)
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
```

getMessageAttrInt

Syntax

```
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

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

getMessageName

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);
}
```

getNextCANdbName

Syntax

```
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
```

getProFileString

Syntax

```
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
getValueSize
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

```
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

```
//Size of an environment variable of data type integer:
int varSize;
...
varSize = getValueSize(switch);
```

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
...
```

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\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

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);
```

inspect

Syntax

```
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

```
//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();
}
```

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

```
dword 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");
}
```

ltoa

Syntax

```
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

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

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));
```

mkExtId

Syntax

```
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);
   ...
}
```

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;
glbHandle = 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 = unsucessful

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);
```

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
```

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);
```

output

Syntax
```
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
```
...
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);
}
```

random

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
```

replayResume

Syntax
```
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");
}
```

replayStart

Syntax

```
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");
}
```

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

```
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

```
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

```
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

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

runError

Syntax

```
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

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

~~seqFileClose~~

Syntax
```
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
```
long fileHandle;
long errorCode;

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

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

~~seqFileGetBlock~~

Syntax

```
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

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

fileHandle = seqFileLoad("capl.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.");
}
```

~~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
...
```

setBusContext

Syntax

```
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 (SIGNLE-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");
}
```

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));
...
```

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));
...
```

setDrift

Syntax

```
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

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

setFilePath

Syntax

```
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

```
...
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);
...
```

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

```
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

```
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

```
...
setOcr(0, 0x02);
resetCan();
...
```

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)

Transceiver: 252, 1041, 1053, 1054 (Bit 4-7 must be zeros)				
Channels	**CAN 1**		**CAN 2**	
Bit Location	**Bit 0**	**Bit 1**	**Bit 2**	**Bit 3**
Normal Mode	1	0	1	0
Sleep Mode	0	1	0	1
No Change	0	0	0	0
No Change	1	1	1	1

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

Transceiver: 5790 (Bit 6-7 must be zeros)				
Channels	**CAN 1**		**CAN 2**	
Bit Location	**Bit 0**	**Bit 1**	**Bit 2**	**Bit 3**
HighVoltage Mode	1	0	1	0
HighSpeed Mode	0	1	0	1
Sleep Mode	0	0	0	0
Normal Mode	1	1	1	1

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

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

```
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

```
//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

```
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

```
//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

getStartDelay

Example

```
on preStart
{
   //Sets delay time to 10 seconds
   setStartdelay(10000);
}
```

setTimer

Syntax

```
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

```
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

```
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

```
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");
...
```

sin

Syntax

```
double sin (double x);
```

Description

Calculates the sine of x.

Parameter

x = 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);   //PI is a built-in constant

//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:

"%ld" or "%d" = decimal display
"%lx" or "%x" = hexadecimal display
"%lX" 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

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"
```

sqrt

Syntax

```
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
```

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);
```

startStatisticAcquisition

Syntax
```
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
```
...
//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();
}
...
```

stop

Syntax
```
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
```
on errorframe
{
   stop();  //End measurement when error frame is received
}
```

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);
```

stopStatisticAcquisition

Syntax
```
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
```
...
//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
```
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
elCount
strncat
strncmp
strncpy

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

strncat

Syntax
```
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
```
...
char s1[7] = "Vector";
char s2[10] = "CANalyzer";
strncat(s1,s2,17);
...
//Result: s1 = "VectorCANalyzer"
```

strncmp

Syntax

```
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

```
...
if(strncmp(s1, s2, strlen(s1))
   write("not equal");
else
   write("equal");
...
```

strncpy

Syntax
```
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
```
...
char s1[7] = "Vector";
char s2[10] = "CANalyzer";
strncpy(s1, s2, strlen(s2) + 1);
...
//Result: s1 = "CANalyzer"
```

swapDWord

Syntax

```
dword swapDWord (dword 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

swapInt
swapLong
swapWord

Example

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

//Result: 0x78563412
```

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

```
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

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

//Result: 0x3412
```

sysExit

Syntax

```
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

```
...
sysExit();
...
```

sysMinimize

Syntax

```
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

```
...
sysMinimize();
...
```

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.

Recommendaton

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.

Observaton

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

Recommendaton

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
```

timeNowFloat

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.

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

Recommendaton
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
}
```

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
"%lx" or "%x" = hexadecimal display
"%lX" 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

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

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

writeClear

Syntax
```
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
```
...
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
```
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:

"%ld" or "%d" = decimal display
"%lx" or "%x" = hexadecimal display
"%lX" 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

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
writeProFileString

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 messges)
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 messges)
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

```
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:

“%ld” or “%d” = decimal display
“%lx” or “%x” = hexadecimal display
“%lX” 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

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

```
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:

“%ld” or “%d” = decimal display
“%lx” or “%x” = hexadecimal display
“%lX” 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

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

```
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

Functions	Real-Time Branches		Analysis Branches	
	CANalyzer (Windows) Transmission Branch	CANoe Simulation Setup Window	CANalyzer (Windows) Analysis Branch	CANoe Measurement Setup Window
abs	Yes	Yes	Yes	Yes
atol	Yes	Yes	Yes	Yes
beep	Yes	Yes	Yes	Yes
callAllOnEnvVar	Yes	Yes	Yes	Yes
cancelTimer	Yes	Yes	Yes	Yes
canOffline		Yes		
canOnline		Yes		
canSetChannelAcc	Yes	Yes	Yes	Yes
canSetChannelMode	Yes	Yes	Yes	Yes
canSetChannelOutput	Yes	Yes	Yes	Yes
cos	Yes	Yes	Yes	Yes
elCount	Yes	Yes	Yes	Yes
enableControl		Yes		Yes
exp	Yes	Yes	Yes	Yes
fileClose	Yes	Yes	Yes	Yes
fileGetBinaryBlock	Yes	Yes	Yes	Yes
fileGetString	Yes	Yes	Yes	Yes
fileGetStringSZ	Yes	Yes	Yes	Yes
fileName	Yes	Yes	Yes	Yes
filePutString	Yes	Yes	Yes	Yes
fileReadArray	Yes	Yes	Yes	Yes
fileReadFloat	Yes	Yes	Yes	Yes
fileReadInt	Yes	Yes	Yes	Yes
fileReadString	Yes	Yes	Yes	Yes
fileRewind	Yes	Yes	Yes	Yes
fileWriteBinaryBlock	Yes	Yes	Yes	Yes
fileWriteFloat	Yes	Yes	Yes	Yes
fileWriteInt	Yes	Yes	Yes	Yes
fileWriteString	Yes	Yes	Yes	Yes
getBusContext		Yes		
getBusNameContext		Yes		
getCardType	Yes	Yes	Yes	Yes
getCardTypeEx	Yes	Yes	Yes	Yes
getChipType	Yes	Yes	Yes	Yes

getDrift		Yes		
getFirstCANdbName			Yes	Yes
getJitterMax		Yes		
getJitterMin		Yes		
getLocalTime			Yes	Yes
getLocalTimeString			Yes	Yes
getMessageAttrInt	Yes	Yes	Yes	Yes
getMessageName			Yes	Yes
getNextCANdbName			Yes	Yes
getProFileArray	Yes	Yes	Yes	Yes
getProFileInt	Yes	Yes	Yes	Yes
getProFileFloat	Yes	Yes	Yes	Yes
getProFileString	Yes	Yes	Yes	Yes
getStartdelay		Yes		
getValue		Yes		Yes
getValueSize		Yes		Yes
halt		Yes		
inport	Yes	Yes	Yes	Yes
inportLPT	Yes	Yes	Yes	Yes
inspect		Yes		
isExtId	Yes	Yes	Yes	Yes
isStatisticAcquisitionRunning			Yes	Yes
isStdId	Yes	Yes	Yes	Yes
keypressed	Yes	Yes	Yes	Yes
ltoa	Yes	Yes	Yes	Yes
makeRGB		Yes		Yes
mkExtId	Yes	Yes	Yes	Yes
msgBeep	Yes	Yes	Yes	Yes
openFileRead	Yes	Yes	Yes	Yes
openFileWrite	Yes	Yes	Yes	Yes
outport	Yes	Yes	Yes	Yes
outportLPT	Yes	Yes	Yes	Yes
output	Yes	Yes	Yes	Yes
putValue		Yes		Yes
putValueToControl		Yes		Yes
random	Yes	Yes	Yes	Yes
replayResume	Yes	Yes	Yes	Yes
replayStart	Yes	Yes	Yes	Yes
replayState	Yes	Yes	Yes	Yes
replayStop	Yes	Yes	Yes	Yes
replaySuspend	Yes	Yes	Yes	Yes
resetCan	Yes	Yes		
resetCanEx	Yes	Yes	Yes	Yes
runError	Yes	Yes	Yes	Yes

seqFileClose	*	*		
seqFileGetBlock	*	*		
seqFileGetLine	*	*		
seqFileGetLineSZ	*	*		
seqFileLoad	*	*		
seqFileRewind	*	*		
setBtr	Yes	Yes		
setBusContext		Yes		
setCanCabsMode	Yes	Yes	Yes	Yes
setControlBackColor		Yes		Yes
setControlForeColor		Yes		Yes
setControlProperty		Yes		Yes
setDrift		Yes		
setFilePath	Yes	Yes	Yes	Yes
setJitter		Yes		
setLogFileName			Yes	Yes
setMsgTime	Yes	Yes	Yes	Yes
setOcr	Yes	Yes		
setPortBit	Yes	Yes	Yes	Yes
setPostTrigger			Yes	Yes
setPreTrigger			Yes	Yes
setStartdelay		Yes		
setTimer	Yes	Yes	Yes	Yes
setWriteDbgLevel		Yes		
setWritePath	Yes	Yes	Yes	Yes
sin	Yes	Yes	Yes	Yes
snPrintf		Yes		Yes
sqrt	Yes	Yes	Yes	Yes
startLogging	Yes	Yes	Yes	Yes
startStatisticAcquisition			Yes	Yes
stop	Yes	Yes	Yes	Yes
stopLogging	Yes	Yes	Yes	Yes
stopStatisticAcquisition			Yes	Yes
strlen	Yes	Yes	Yes	Yes
strncat	Yes	Yes	Yes	Yes
strncmp	Yes	Yes	Yes	Yes
strncpy	Yes	Yes	Yes	Yes
swapDWord	Yes	Yes	Yes	Yes
swapInt	Yes	Yes	Yes	Yes
swapLong	Yes	Yes	Yes	Yes
swapWord	Yes	Yes	Yes	Yes
sysExit			Yes	Yes
sysMinimize			Yes	Yes
timeDiff	Yes	Yes	Yes	Yes

timeNow	Yes	Yes	Yes	Yes
timeNowFloat	Yes	Yes	Yes	Yes
trigger	Yes	Yes	Yes	Yes
valOfId	Yes	Yes	Yes	Yes
write	Yes	Yes	Yes	Yes
writeClear	Yes	Yes	Yes	Yes
writeCreate	Yes	Yes	Yes	Yes
writeDbgLevel		Yes		
writeDestroy	Yes	Yes	Yes	Yes
writeEx	Yes	Yes	Yes	Yes
writeLineEx	Yes	Yes	Yes	Yes
writeProFileFloat	Yes	Yes	Yes	Yes
writeProFileInt	Yes	Yes	Yes	Yes
writeProFileString	Yes	Yes	Yes	Yes
writeTextBkgColor	Yes	Yes	Yes	Yes
writeTextColor	Yes	Yes	Yes	Yes
writeToLog	Yes	Yes	Yes	Yes
writeToLogEx	Yes	Yes	Yes	Yes

* = function can be used if CAN.INI is configured correctly.

Table x – CAPL Function Compatibilities

Availability Chart

Functions	Status S = Supported O = Obsolete	Versions Supported
abs	S	2.5 and after
atol	S	All
beep	O	Prior to 3.0
callAllOnEnvVar	S	All
cancelTimer	S	All
canOffline	S	All
canOnline	S	All
canSetChannelAcc	S	5.0 and after
canSetChannelMode	S	5.0 and after
canSetChannelOutput	S	5.0 and after
cos	S	All
elCount	S	All
enableControl	S	4.1 and after
exp	S	All
fileClose	S	3.0 and after
fileGetBinaryBlock	S	3.0 and after
fileGetString	S	3.0 and after
fileGetStringSZ	S	3.0 and after
fileName	S	All
filePutString	S	3.0 and after
fileReadArray	O	Prior to 3.0
fileReadFloat	O	Prior to 3.0
fileReadInt	O	Prior to 3.0
fileReadString	O	Prior to 3.0
fileRewind	S	Prior to 3.0
fileWriteBinaryBlock	S	3.0 and after
fileWriteFloat	O	Prior to 3.0
fileWriteInt	O	Prior to 3.0
fileWriteString	O	Prior to 3.0
getBusContext	S	3.2 and after
getBusNameContext	S	3.2 and after
getCardType	S	All
getCardTypeEx	S	5.0 and after
getChipType	S	All
getDrift	S	3.0 and after
getFirstCANdbName	S	4.0 and after

getJitterMax	S	3.0 and after
getJitterMin	S	3.0 and after
getLocalTime	S	All
getLocalTimeString	S	All
getMessageAttrInt	S	3.1 and after
getMessageName	S	4.0 and after
getNextCANdbName	S	4.0 and after
getProFileArray	S	3.0 and after
getProFileInt	S	3.0 and after
getProFileFloat	S	3.0 and after
getProFileString	S	3.0 and after
getStartdelay	S	3.0 and after
getValue	S	All
getValueSize	S	All
halt	S	4.1 and after
inport	S	All
inportLPT	S	3.1 and after
inspect	S	4.1 and after
isExtId	S	All
isStatisticAcquisitionRunning	S	3.0 and after
isStdId	S	All
keypressed	S	All
ltoa	S	All
makeRGB	S	4.1 and after
mkExtId	S	All
msgBeep	S	3.0 and after
openFileRead	S	3.0 and after
openFileWrite	S	3.0 and after
outport	S	All
outportLPT	S	3.1 and after
output	S	All
putValue	S	All
putValueToControl	S	4.0 and after
random	S	All
replayResume	S	4.0 and after
replayStart	S	4.0 and after
replayState	S	4.0 and after
replayStop	S	4.0 and after
replaySuspend	S	4.0 and after
resetCan	S	All
resetCanEx	S	4.1 and after
runError	S	All
seqFileClose	O	Prior to 3.0
seqFileGetBlock	O	Prior to 3.0

seqFileGetLine	O	Prior to 3.0
seqFileGetLineSZ	O	Prior to 3.0
seqFileLoad	O	Prior to 3.0
seqFileRewind	O	Prior to 3.0
setBtr	S	All
setBusContext	S	3.2 and after
setCanCabsMode	S	4.1 and after
setControlBackColor	S	4.1 and after
setControlForeColor	S	4.1 and after
setControlProperty	S	4.1 and after
setDrift	S	3.0 and after
setFilePath	S	4.1 and after
setJitter	S	3.0 and after
setLogFileName	S	All
setMsgTime	O	Prior to 2.5
setOcr	S	All
setPortBit	S	4.1 and after
setPostTrigger	S	All
setPreTrigger	S	All
setStartdelay	S	3.0 and after
setTimer	S	All
setWriteDbgLevel	S	3.1 and after
setWritePath	S	3.0 and after
sin	S	All
snPrintf	S	All
sqrt	S	All
startLogging	S	4.1 and after
startStatisticAcquisition	S	3.0 and after
stop	S	All
stopLogging	S	4.1 and after
stopStatisticAcquisition	S	3.0 and after
strlen	S	All
strncat	S	All
strncmp	S	All
strncpy	S	All
swapDWord	S	All
swapInt	S	All
swapLong	S	All
swapWord	S	All
sysExit	S	All
sysMinimize	S	All
timeDiff	S	All
timeNow	S	All
timeNowFloat	S	all

trigger	S	All
valOfId	S	All
write	S	All
writeClear	S	3.2 and after
writeCreate	S	3.2 and after
writeDbgLevel	S	3.0 and after
writeDestroy	S	3.2 and after
writeEx	S	3.2 and after
writeLineEx	S	3.2 and after
writeProFileFloat	S	3.0 and after
writeProFileInt	S	3.0 and after
writeProFileString	S	3.0 and after
writeTextBkgColor	S	5.0 and after
writeTextColor	S	5.0 and after
writeToLog	S	All
writeToLogEx	S	3.0 and after

Visit our Website for more information on
>News
>Products
>Demo Software
>Support
>Training Classes
>Global Locations

www.vector-cantech.com

vector
the art of engineering