AnaGate API

Programmer's Manual

Analytica GmbH

A. Schmidt, Analytica GmbH
S. Welisch, Analytica GmbH

AnaGate API: Programmer's Manual

Analytica GmbH
by A. Schmidt and S. Welisch

This document was generated with DocBook at 2012-03-13 10:04:16.

Hilfe-Datei (dtsch.): *AnaGate-API.chm*

Hilfe-Datei (engl.): *AnaGate-API-EN.chm*

PDF-Datei (dtsch.): *AnaGate-API-1.10.pdf*

PDF-Datei (engl.): *AnaGate-API-1.10-EN.pdf*

Publication date 09. September 2010
Copyright © 2007-2010 Analytica GmbH

Abstract

The AnaGate Programmer's Manual includes the exact description of the programming interfaces to all models of *AnaGate* hardware series.

This manual bases on the actual *AnaGate* Application Programming Interface (API) in Version 1.10 and the AnaGate communication protocol V1.3 (see [TCP-2010]).

All rights reserved. All the information in this manual was compiled with the greatest of care. However, no warranty can be given for it.

No parts of this manual or the program are to be reproduced in any way (printing, photocopying, microfilm or any other process) without written authorisation. Any processing, duplication or distribution by means of any electronic system is also strictly prohibited.

You are also advised that all the names and brand names of the respective companies mentioned in this documentation are generally protected by brand, trademark or patent laws.

Analytica GmbH
Vorholzstraße 36
76137 Karlsruhe
Germany
Fon +49 (0) 721-43035-0
Fax +49 (0) 721-43035-20
<support@analytica-gmbh.de>

www.analytica-gmbh.de [http://www.analytica-gmbh.de]
www.anagate.de [http://www.anagate.de]

Revision History			
Revision 1.4	01.10.2010	ASce	Complete revision of all chapters
Revision 1.3	12.07.2010	SWe	CAN UPD fucntions added (LUA only)
Revision 1.2	04.06.2010	SWe	I2C RAW functions added temporarily
Revision 1.1	01.04.2010	ASc	english version
Revision 1.0	08.06.2009	ASc	Manual changed to **DocBook** format

Table of Contents

Introduction ix
I. AnaGate API 1
1. The Programming interface of AnaGate product line 3
2. Notes concerning the communication protocol TCP 5
2.1. Important properties of the network protocol 5
3. Common function reference 7
DLLInfo 8
4. CAN API reference 9
CANOpenDevice, CANOpenDeviceEx 10
CANCloseDevice 12
CANSetGlobals 13
CANGetGlobals 15
CANSetFilter 17
CANGetFilter 19
CANSetTime 20
CANWrite, CANWriteEx 21
CANSetCallback, CANSetCallbackEx 23
CANReadDigital 25
CANWriteDigital 27
CANRestart 28
CANDeviceConnectState 29
CANStartAlive 30
CANErrorMessage 31
5. SPI API reference 32
SPIOpenDevice 33
SPICloseDevice 35
SPISetGlobals 36
SPIGetGlobals 38
SPIDataReq 40
SPIReadDigital 42
SPIWriteDigital 44
SPIErrorMessage 45
6. I2C API reference 46
I2COpenDevice 47
I2CCloseDevice 49
I2CReset 50
I2CRead 51
I2CWrite 52
I2CSequence 53
I2CReadDigital 55
I2CWriteDigital 57
I2CErrorMessage 58
I2CReadEEPROM 59
I2CWriteEEPROM 61
7. Programming examples 64
7.1. Programming language C/C++ 64
7.2. Programming language Visual Basic 6 65
7.3. Programming language VB.NET 69
II. Scripting language LUA 72
8. The LUA scripting interface of the *AnaGate* product line 75
8.1. Creating scripts 76

 © 2007-2010 Analytica GmbH

8.2. Running scripts on personal computer 76
8.3. Running scripts on *AnaGate* hardware 77
9. Common function reference 80
LS_DeviceInfo 81
LS_GetTime 82
LS_Sleep 83
10. CAN Reference 84
LS_CANOpenDevice 85
LS_CANCloseDevice 87
LS_CANRestartDevice 88
LS_CANSetGlobals 89
LS_CANGetGlobals 91
LS_CANWrite 93
LS_CANWriteEx 95
LS_CANSetCallback 97
LS_CANGetMessage 99
LS_CANSetFilter 101
LS_CANGetFilter 102
LS_CANSetTime 103
LS_CANErrorMessage 104
LS_CANReadDigital 105
LS_CANWriteDigital 106
11. SPI Reference 107
LS_SPIOpenDevice 108
LS_SPICloseDevice 109
LS_SPISetGlobals 110
LS_SPIGetGlobals 112
LS_SPIDataReq 114
LS_SPIErrorMessage 116
LS_SPIReadDigital 117
LS_SPIWriteDigital 118
12. I2C Reference 119
LS_I2COpenDevice 120
LS_I2CCloseDevice 122
LS_I2CReset 123
LS_I2CRead 124
LS_I2CWrite 125
LS_I2CReadDigital 126
LS_I2CWriteDigital 127
LS_I2CErrorMessage 128
LS_I2CReadEEPROM 129
LS_I2CWriteEEPROM 131
13. CANOpen functions 133
LS_CANopenSetConfig 134
LS_CANopenGetConfig 135
LS_CANopenSetSYNCMode 136
LS_CANopenSetCallbacks 137
LS_CANopenGetPDO 138
LS_CANopenGetSYNC 139
LS_CANopenGetEMCY 140
LS_CANopenGetGUARD 141
LS_CANopenGetUndefined 142
LS_CANopenSendNMT 143
LS_CANopenSendSYNC 144

© 2007-2010 Analytica GmbH

LS_CANopenSendTIME 145
LS_CANopenSendPDO 146
LS_CANopenSendSDORead 147
LS_CANopenSendSDOWrite 148
LS_CANopenSendSDOReadBlock 149
LS_CANopenSendSDOWriteBlock 150
Programmer example 151
14. LUA programming examples 153
14.1. Examples for devices with CAN interface 153
14.2. Examples for devices with SPI interface 154
14.3. Examples for devices with I2C interface 155
A. API return codes 158
B. I2C slave address formats 160
C. Programming I2C EEPROM 162
D. FAQ - Frequent asked questions 164
E. FAQ - Programming API 168
F. Technical support 169
Bibliography 170

© 2007-2010 Analytica GmbH

List of Figures

7.1. Input form of SPI example (VB6) 66
8.1. Edit LUA script in a text editor 76
8.2. HTTP interface, LUA settings 78
B.1. Definition of a I2C slave address in 7-bit format 160
B.2. Definition of a I2C slave address in 10-bit format 160

 © 2007-2010 Analytica GmbH

List of Tables

1.1. Library files for Windows 3
1.2. Library files for Linux 3
2.1. AnaGate devices and related port numbers 5
4.1. mask filter examples for CAN identifier 17
A.1. Common return values for all devices of AnaGate series 158
A.2. Return values for AnaGate I2C 158
A.3. Return values for AnaGate CAN 159
A.4. Return values for AnaGate Renesas 159
A.5. Return values for LUA scripting 159
B.1. I2C EEPROM addressing examples 160
C.1. Usage of the CHIP-Enable Bits of I2C EEPROMs 162
D.1. Using AnaGate hardware with firewall 165

© 2007-2010 Analytica GmbH

List of Examples

13.1. CANOpen - LUA script example 152
14.1. 153
14.2. 154
14.3. 155
14.4. 156
14.5. 157

© 2007-2010 Analytica GmbH

Introduction

The AnaGate Programmer's Manual includes the exact description of the programming interfaces to all models of AnaGate hardware series.

The existing interfaces will be decribed below:

- Application Programming Interface (Part I, “AnaGate API”)
- LUA Scripting Interface (Part II, “Scripting language LUA”)

© 2007-2010 Analytica GmbH

Part I. AnaGate API

Table of Contents

1. The Programming interface of AnaGate product line 3
2. Notes concerning the communication protocol TCP 5
    2.1. Important properties of the network protocol 5
3. Common function reference 7
    DLLInfo 8
4. CAN API reference 9
    CANOpenDevice, CANOpenDeviceEx 10
    CANCloseDevice 12
    CANSetGlobals 13
    CANGetGlobals 15
    CANSetFilter 17
    CANGetFilter 19
    CANSetTime 20
    CANWrite, CANWriteEx 21
    CANSetCallback, CANSetCallbackEx 23
    CANReadDigital 25
    CANWriteDigital 27
    CANRestart 28
    CANDeviceConnectState 29
    CANStartAlive 30
    CANErrorMessage 31
5. SPI API reference 32
    SPIOpenDevice 33
    SPICloseDevice 35
    SPISetGlobals 36
    SPIGetGlobals 38
    SPIDataReq 40
    SPIReadDigital 42
    SPIWriteDigital 44
    SPIErrorMessage 45
6. I2C API reference 46
    I2COpenDevice 47
    I2CCloseDevice 49
    I2CReset 50
    I2CRead 51
    I2CWrite 52
    I2CSequence 53
    I2CReadDigital 55
    I2CWriteDigital 57
    I2CErrorMessage 58
    I2CReadEEPROM 59
    I2CWriteEEPROM 61
7. Programming examples 64
    7.1. Programming language C/C++ 64
    7.2. Programming language Visual Basic 6 65
    7.3. Programming language VB.NET 69

© 2007-2010 Analytica GmbH

Chapter 1. The Programming interface of AnaGate product line

The *AnaGate* product line consist of several hardware devices, which offers access to different bus systems (I2C, SPI, CAN) or processors (Renesas) via standard network protocol.

The communication to the individual devices always is done through a documented and disclosed proprietary network protocol. Thus, all products which incorporates a socket interface (like personal computers, PLC, ...) are allowed to access the devices of the *AnaGate* product line.

Analytica provides a programming interface for users of Windows and Linux operating systems (X86) which implements the proprietary communication protocol and make it available through simple function calls. The software API (Application Programming Interface) is available free of charge for Windows and Linux operarting systems.

Table 1.1. Library files for Windows

Device	Windows library
AnaGate CAN	AnaGateCAN.dll
AnaGate CAN uno / duo / quattro	AnaGateCAN.dll
AnaGate CAN USB	AnaGateCAN.dll
AnaGate SPI	AnaGateSPI.dll
AnaGate I2C / I2C X7	AnaGateI2C.dll
AnaGate Universal Programmer	AnaGateSPI.dll, AnaGateI2C.dll

Note

To provide a widespread support of differrent programming languages like C++, Visual Basic, Delphi and the programming languages of the .NET family, the cdecl calling convention is used in all function calls. Using this calling convention means that all function parameters are pushed on the stack in reverse order (from right to left) and that the caller is responsible for the stack handling. Most programming languages support this calling convention.

Table 1.2. Library files for Linux

Device	Linux library (X86)	ARM9
AnaGate CAN	libCANdll.a, libAnaCommon.a	-
AnaGate CAN uno / duo / quattro	libCANdll.a, libAnaCommon.a	available
AnaGate CAN USB	libCANdll.a, libAnaCommon.a	available
AnaGate SPI	libSPIdll.a, libAnaCommon.a	-
AnaGate I2C / I2C X7	libI2Cdll.a, libAnaCommon.a	-
AnaGate Universal Programmer	libSPIdll.a, libI2Cdll.a, libAnaCommon.a	available

© 2007-2010 Analytica GmbH

The different libraries include common and specific functions which are neccessary for accessing and controlling the devices of the *AnaGate* product line. In the following, all library functions of the software API are documented in detail.

Tip

It is possible to extend individually the newer device models with embedded Linux (kernel 2.6) and ARM9 processor. The complete software API is available in a cross-compiled version and can be used on the devices itself to create individual device extensions. To do so is very easy because the programming interface on the personal computer and the device is completely identical.

A preconfigured virtual machine (Virtual-Box-Image) with Ubuntu-Linux “READY-to-USE” with installed development environment (**Kdevelop**, **Eclipse**) and all neccessary program libraries (**GCC**, cross compiler, libraries, **LUA**, ...) is available optional .

 © 2007-2010 Analytica GmbH

Chapter 2. Notes concerning the communication protocol TCP

Access to the different models of the *AnaGate* product line is always done via the most frequently used network protocol TCP (Transmission Control Protocol).

TCP is connection-oriented packet-switched transport protocol which is located in layer 4 of the of the OSI reference model. In principle TCP is an end-to-end connection which allows exchange of data in both directions at the same time. An end-point is a pair formed of an IP address and a port number and. Such a pair builds a bidirectional software interface and is called socket.

The *AnaGate* device offers its functionality as so-called TCP server. It creates a socket with its IP address and a device-specific port number. On the models with CAN interface(s) a seperate socket with different port number is created for each existing CAN interface, on every socket up to 5 concurrent client connections are accepted. The SPI, I2C and Renesas interfaces accept only one concurrent connection at the same time.

Table 2.1. AnaGate devices and related port numbers

Device	Port number
AnaGate I2C, AnaGate Universal Programmer	5000
AnaGate CAN, AnaGate CAN uno	5001
AnaGate CAN duo	5001, 5101
AnaGate CAN quattro	5001, 5101, 5201, 5301
AnaGate SPI, AnaGate Universal Programmer	5002
AnaGate Renesas, AnaGate Universal Programmer	5008

Important

Please ensure that all used ports are set active on the personal computer to grant access to the *AnaGate* device. Any existing firewalls are to be configured accordingly.

2.1. Important properties of the network protocol

In most cases TCP is based on the internet protocol (IP). IP is package-oriented, whereby it is possible that data packets are lost or the packets can be received in wrong order or perhaps more than once.

TCP eliminates this behaviour and ensures that the the data packets are received in correct order at the recipient. Is a sent data packet not confirmed by the recipient within a timeout limit, the packet is sent again. Double packets are recognized at the recipient and are deleted. During connection the data transmission may be impaired,

© 2007-2010 Analytica GmbH

delayed or completely interrupted. A successful connection do not guarantee a permanently stable data transmission.

Detection and evaluation of network and line malfunctions can be difficult, if there is only sporadic communication on the line. How is possible to distinguish between a malfunction on the line or simply no data from the connected endpoint?

To amend this problematic nature TCP provides an internal keep alive mechanism. Keep-alives are special data packets which are sent in regular intervalls between the two endpoints of an opened communication channel. The recipient of a keep-alive packet has to confirm the receipt to the sender within a certain period of time. Are there no keep-alives or confirmations of keep-alives receivedm the communication partner assumes that the channel is interrupted or the corresponding socket is malfunctioning.

The keep-alive mechanism of TCP is not active per default and has to be activated by the `setsockopt` function for each connection. The API functions which establish a connection to an *AnaGate* - like the `CANOpenDevice()` function - strictly activate the keep-alive mechisam of TCP.

Note

On Windows operating systems some settings concerning keep-alives can be set individually. These settings are valid for all network connection on this computer and can not be set individually for dedicated connections.

To do so the Windows registry keys **KeepAliveTime** and **KeepAliveInterval** of node `\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Tcpip\Parameters` has to be adjusted (administrator rights).

Especially the CAN-Ethernet gateways can be affected by the above described problems, for example if customer-specific needs ask for faster detection of connection aborts as possible via the standard mechanism. So, in the AnaGate models with CAN interface and linux OS an application-specific keep-alive algorithm is integrated in the device firmware to enhance connection control. On base of a predefined time period additional data packets are exchanged between the AnaGate hardware and the controlling unit/personal computer, which have to be confirmed by the corresponding endpoint (ALIVE_REQ, see [TCP-2010]). This integrated alive machanism can be activated individually on each connection with a different timeout interval.

Note

Users of the AnaGate-API do not have to implement the application-specific alive mechanism to use it. With a simple call to the API function `CANStartAlive` a concurrent thread is started which automatically monitors the communication channel time-controlled.

 © 2007-2010 Analytica GmbH

Chapter 3. Common function reference

© 2007-2010 Analytica GmbH

DLLInfo

DLLInfo — Determines the current version information of the AnaGate DLL.

Syntax

#include <AnaGateDLL.h>

```
int DLLVersion(char * pcMessage, int nMessageLen);
```

Parameter

pcMessage Data buffer that is to accept the version reference number of the AnaGate DLL.

nMessageLen Size in bytes of the transferred data buffer.

Return value

Actual size of the returned version reference number.

Remarks

If the version reference number is too large for the transferred data buffer, it is abbreviated to the given number of characters (*nMessageLen*).

 © 2007-2010 Analytica GmbH

Chapter 4. CAN API reference

The CAN API can be used with all CAN gateway models of the AnaGate series. The programming interface is identical for all devices and uses the network protocol TCP or UDP in general.

Following devices can be addresse via the CAN API interface:

- AnaGate CAN
- AnaGate CAN uno
- AnaGate CAN duo
- AnaGate CAN quattro
- AnaGate CAN USB

© 2007-2010 Analytica GmbH

CANOpenDevice, CANOpenDeviceEx

CANOpenDevice, CANOpenDeviceEx — Opens an network connection (TCP or UDP) to an AnaGate CAN device.

Syntax

#include <AnaGateDllCan.h>

```
int CANOpenDevice(int *pHandle, BOOL bSendDataConfirm, BOOL bSendDataInd, int nCANPort, const char * pcIPAddress, int nTimeout );
```

```
int CANOpenDeviceEx(int *pHandle, BOOL bSendDataConfirm, BOOL bSendDataInd, int nCANPort, const char * pcIPAddress, int nTimeout , int nSocketType );
```

Parameter

pHandle
: Pointer to a variable, in which the access handle is saved in the event of a successful connection to the AnaGate device.

bSendDataConfirm
: It set to TRUE, all incoming and outgoing Data requests are confirmed by the internal message protocol. Without confirmations a better transmittion performance is reached.

bSendDataInd
: If set to FALSE, all incoming telegrams are discarded.

nCANPort
: CAN port number. Allowed values are:

  `0` for port A (Modells AnaGate CAN uno, AnaGate CAN duo, AnaGate CAN quattro, AnaGate CAN USB and AnaGate CAN)

  `1` for port B (AnaGate CAN duo, AnaGate CAN quattro)

  `2` for port C (AnaGate CAN quattro)

  `3` for port D (AnaGate CAN quattro)

pcIPAddress
: Network address of the AnaGate partner.

nTimeout
: Default timeout for accessing the AnaGate in milliseconds.

  A timeout is reported if the AnaGate partner does not respond within the defined timeout period. This global timeout value is valid on the current network connection for all commands and functions which do not offer a specific timeout value.

nSocketType
: Specifies the socket type (ethernet layer 4) which is to be used for the new connection. Only two different types are supported: TCP and UDP. The funttion `CANOpenDevice` alyways uses TCP sockets. Use the following constants fpr the parameter:

  `1` (SOCK_STREAM) TCP (Transmission Control Protocol)

2 (SOCK_DGRAM)	UDP (User Datagram Protocol)

Return value

Returns `Null` if successful, or an error value otherwise (Appendix A, *API return codes*).

Description

Opens a TCP/IP connection to an CAN interface of a AnaGate CAN device. With `CANOpenDeviceEx` it is possible to set the ethernet layer4 protocol (tcp or udp). If the connection is established, CAN telegrams can be sent and received.

The connection should be closed with the function `CANCloseDevice` if not longer needed.

Important

It is recommended to close the connection, because all internally allocated system resources are freed again and the connected AnaGate device is signalled that the active connection is not longer in use and can be used again for new connect requests.

See the following example for the initial programming steps.

```
#include <AnaGateCANDll.h>
int main()
{
   int hHandle;
   int nRC = CANOpenDevice(&hHandle, TRUE, TRUE, 0, "192.168.0.254", 5000);
   if ( nRC == 0 )
   {
       // ... now do something
       CANCloseDevice(hHandle);
   }
   return 0;
}
```

Remarks

The *`CANOpenDeviceEx`* function is supported for library versions 1.5-1.10 or higher and firmware version 1.3.7 or higher.

Device models of type AnaGate CAN (hardware version 1.1.A) do not listen for UPD connection requests. If trying to connect such a device via UPD, the `CANOpenDeviceEx` returns with a timeout error.

See also

CANCloseDevice

CANRestart

 © 2007-2010 Analytica GmbH

CANCloseDevice

CANCloseDevice — Closes an open network connection to an AnaGate CAN device.

Syntax

```
#include <AnaGateDllCan.h>

int CANCloseDevice(int hHandle);
```

Parameter

hHandle Valid access handle.

Return value

Returns `Null` if successful, or an error value otherwise (Appendix A, *API return codes*).

Description

Closes an open network connection to an AnaGate CAN device. The *hHandle* parameter is a return value of a succesfull call to the function `CANOpenDevice`.

Important

It is recommended to close the connection, because all internally allocated system resources are freed again and the connected AnaGate device is signalled that the active connection is not longer in use and can be used again for new connect requests.

See also

CANOpenDevice, CANOpenDeviceEx

 © 2007-2010 Analytica GmbH

CANSetGlobals

CANSetGlobals — Sets the global settings, which are to be used on the CAN bus

Syntax

```
#include <AnaGateDllCAN.h>
```

```
int CANSetGlobals(int hHandle, int nBaudrate, unsigned char
nOperatingMode, BOOL bTermination, BOOL bHighSpeedMode, BOOL
bTimeStampOn);
```

Parameter

hHandle Valid access handle.

nBaudrate The baud rate to be used. Following values are allowed:

- 10.000 für 10kBit
- 20.000 für 20kBit
- 50.000 für 50kBit
- 62.500 für 62,5kBit
- 100.000 für 100kBit
- 125.000 für 125kBit
- 250.000 für 250kBit
- 500.000 für 500kBit
- 800.000 für 800kBit (not AnaGate CAN)
- 1.000.000 für 1MBit

nOperatingMode The operating mode to be used. Following values are allowed.

- 0 = default mode.
- 1 = loop back mode: All telegrams sent by a connected partner is routed back to all active connected partners.
- 2 = listen mode: Device operates as passive bus partner, this means no telegrams are sent to the CAN bus (no ACKs for incoming telegrams too).

bTermination Use integrated CAN bus termination (TRUE= yes, FALSE = no). This setting is not supported by all AnaGate CAN models.

bHighSpeedMode Use high speed mode (TRUE= yes, FALSE= no). This setting is not supported by all AnaGate CAN models.

 © 2007-2010 Analytica GmbH

The high speed mode was created for large baud rates with continuously high bus load. In this mode telegrams are not confirmed on procol layer and the software filters defined via `CANSetFilter` are ignored.

bTimeStampOn Use time stamp mode (TRUE= yes, FALSE= no). This setting is not supported by all AnaGate CAN models.

In activated time stamp mode an additional timestamp is sent with the CAN telegram. This timestamp indicates when the incomming message is received by the CAN controller or when the outgoing message is confirmed by the CAN controller.

Return value

Returns `Null` if successful, or an error value otherwise (Appendix A, *API return codes*).

Description

Sets the global settings of the used CAN interface. These settings are effective for all concurrent connections to the CAN interface. The settings are not saved permantently on the device and are reset every device restart.

Remarks

The settings of the integrated CAN bus termination, the high speed mode and the time stamp are not supported by the AnaGate CAN (hardware version 1.1.A). These settings are ignored by the device.

See also

CANGetGlobals

© 2007-2010 Analytica GmbH

CANGetGlobals

CANGetGlobals — Returns the currently used global settings on the CAN bus.

Syntax

```
#include <AnaGateDllCAN.h>

int CANGetGlobals(int hHandle, int * pnBaudrate, unsigned char * pnOperatingMode, BOOL * pbTermination, BOOL * pbHighSpeedMode, BOOL * pbTimeStampOn);
```

Parameter

hHandle Valid access handle.

pnBaudrate The baud rate currently used on the CAN bus.

pnOperatingMode The operating mode to be used. Following values are returned.

- 0 = default mode.
- 1 = loop back mode: All telegrams sent by a connected partner is routed back to all active connected partners.
- 2 = listen mode: Device operates as passive bus partner, this means no telegrams are sent to the CAN bus (no ACKs for incoming telegrams too).

pbTermination Is the integrated CAN bus termination used? (TRUE= yes, FALSE= no). This setting is not supported by all AnaGate CAn modells.

pbHighSpeedMode Is the high speed mode switched on? (TRUE= yes, = no). This setting is not supported by all AnaGate CAN modells.

The high speed mode was created for large baud rates with continuously high bus load. In this mode telegrams are not confirmed on procol layer and the software filters defined via `CANSetFilter` are ignored.

pbTimeStampOn Is a timestamp mode activated on the current network connection? (TRUE= yes, FALSE= no). This setting is not supported by all AnaGate CAN modells.

In activated time stamp mode an additional timestamp is sent with the CAN telegram. This timestamp indicates when the incomming message is received by the CAN controller or when the outgoing message is confirmed by the CAN controller.

Return value

Returns `Null` if successful, or an error value otherwise (Appendix A, *API return codes*).

 © 2007-2010 Analytica GmbH

Description

Returns the global settings of the used CAN interface. These settings are effective for all concurrent connections to the CAN interface.

Remarks

The settings of the integrated CAN bus termination, the high speed mode and the time stamp are not supported by the AnaGate CAN (hardware version 1.1.A). These settings are ignored by the device.

See also

CANGetGlobals

 © 2007-2010 Analytica GmbH

CANSetFilter

CANSetFilter — Sets the current filter settings for the connection.

Syntax

#include <AnaGateDllCAN.h>

```
int CANSetFilter(int hHandle, const int * pnFilter);
```

Parameter

hHandle Valid access handle.

pnFilter Pointer to an array of 8 filter entries (4 mask and 4 range filter entries). A filter entry contains of two 32-bit values. Unused mask filter entries must be initialized with 0 values. Unused range filter entries must be initialized with a 0 for the start value and 0x1FFFFFFF for the end value.

Return value

Returns `Null` if successful, or an error value otherwise (Appendix A, *API return codes*).

Description

This function sets the current filter settings for the current connection. Filter can be used to suppress messages with specific CAN message ids.

A mask filter contains of a mask value, which defines the bits of the CAN identifier to examine, and the appropriate filter value. If the CAN identifier matches in the indicated filter mask with the filter value, the incoming CAN telegram is sent to the PC, otherwise not.

A range filter defines an address range with a appropriate start and end address. If the CAN identifier do not lie in the indicated filter range, the incoming CAN telegram is not sent to the PC.

Filter are only active, if the parameter *bSendDataInd* is set via the `CANOpenDevice` function. If the pararmeter *bHighSpeedMode* of the `CANSetGlobals` is set, all filters are deactivated to increase the data pass through.

Table 4.1. mask filter examples for CAN identifier

CAN id	mask value	filter value	result
0x0F	0x0E	0x0C	suppressed
0x0C	0x0E	0x0C	ok
0x5D	0x0E	0x0C	ok

See the following example for setting some filters.

```
#include <AnaGateCANDll.h>
```

 © 2007-2010 Analytica GmbH

```
int main()
{
{  int anFilter[16] = {
       0xFF, 0x0F,    // mask  filter 1: mask = 0xFF, value = 0x0F: route only 0x*0F values
       0, 0,          // mask  filter 2: unused
       0, 0,          // mask  filter 3: unused
       0, 0,          // mask  filter 4: unused
       0, 0x00000FFF, // range filter 1: all ids greater than 0xFFF are discarded
       0, 0x1FFFFFFF, // range filter 2: unused
       0, 0x1FFFFFFF, // range filter 3: unused
       0, 0x1FFFFFFF, // range filter 4: unused
   };
   int hHandle;
   int nRC = CANOpenDevice(&hHandle, TRUE, TRUE, 0, "192.168.0.254", 5000);
   if ( nRC == 0 )
   {
       nRC = CANSetFilter( hHandle, &anFilter );
       // ... now do something
       CANCloseDevice(hHandle);
   }
   return 0;
}
```

See also

CANGetFilter

 © 2007-2010 Analytica GmbH

CANGetFilter

CANGetFilter — Returns the current filter settings for the connection.

Syntax

#include <AnaGateDllCAN.h>

```
int CANGetFilter(int hHandle, int * pnFilter);
```

Parameter

hHandle Valid access handle.

pnFilter Pointer to an array of 8 filter entries (4 mask and 4 range filter entries). A filter entry contains of two 32-bit values. Unused mask filter entries are initialized with 0 values. Unused range filter entries are initialized with (0,0x1FFFFFFF) value pairs.

Return value

Returns `Null` if successful, or an error value otherwise (Appendix A, *API return codes*).

Description

This function retrieves the current filter settings for the current connection. Filter can be used to suppress messages with specific CAN message ids.

See also

CANSetFilter

 © 2007-2010 Analytica GmbH

CANSetTime

CANSetTime — Sets the current system time on the AnaGate device.

Syntax

```
#include <AnaGateDllCAN.h>

int CANSetTime(int hHandle, long nSeconds, long nMicroseconds);
```

Parameter

hHandle Valid access handle.

nSeconds Time in seconds from 01.01.1970.

nMicroseconds Micro seconds.

Return value

Returns `Null` if successful, or an error value otherwise (Appendix A, *API return codes*).

Description

The `CANSetTime` function sets the system time on the AnaGate hardware.

If the time stamp mode is switched on by the `CANSetGlobals` function, the AnaGate hardware adds a time stamp to each incoming CAN telegram and a time stamp to the confirmation of a telegram sent via the API (only if confirmations are switched on for data requests).

Remarks

The `CANSetTime` function is supported by library version 1.4-1.8 or higher.

The setting of the base time for the time stamp mode is not supported by the AnaGate CAN (hardware version 1.1.A). This setting is ignored by the device.

© 2007-2010 Analytica GmbH

CANWrite, CANWriteEx

CANWrite, CANWriteEx — Send a CAN telegram to the CAN bus via the AnaGate device.

Syntax

#include <AnaGateDllCan.h>

```
int CANWrite(int hHandle, int nIdentifier, const char * pcBuffer, int nBufferLen, int nFlags);
```

```
int CANWriteEx(int hHandle, int nIdentifier, const char * pcBuffer, int nBufferLen, int nFlags, long * pnSeconds, long * pnMicroSeconds);
```

Parameter

hHandle — Valid access handle.

nIdentifier — CAN identifier of the sender. Parameter *nFlags* defines, if the address is in extended format (29-bit) or standard format (11-bit).

pcBuffer — Pointer to the data buffer.

nBufferLen — Length of data buffer (max. 8 bytes).

nFlags — The format flags are defined as follows.

- Bit 0: If set, the CAN identifier is in extended format (29 bit), otherwise not (11 bit).
- Bit 1: If set, the telegram is marked as remote frame.
- Bit 2: If set, the telegram has a valid timestamp. This bit is only set for incoming data telegrams and must not be set for the `CANWrite` and `CANWriteEx` functions.

pnSeconds — Timestamp of the confirmation of the CAN controller (seconds from 01.01.1970).

pnMicroSeconds — Micro seconds portion of the timestamp.

Return value

Returns `Null` if successful, or an error value otherwise (Appendix A, *API return codes*).

Description

Both functions sends a CAN telegram to the CAN bus via the AnaGate device.

The `CANWriteEx` additionallly returns a timestamp of the time at which the telegram is sent.

© 2007-2010 Analytica GmbH

Note

With remote frames (RTR = remote transmission request) a destination node can request data from a source node. The data length is to be set to the number of requested bytes - on the CAN bus no data is sent only the data size information.

When using the `CANWrite` bzw. `CANWriteEx` functions to send remote frames the data buffer and the buffer size equal to the number of requested bytes have to be set correctly.

See the following example for sending a data telegram to the connected CAN bus.

```
#include <AnaGateCANDll.h>
int main()
{
   char cMsg[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
   int hHandle = 0;
   int nRC = 0;
   int nFlags = 0x0; //  11bit address + standard (not remote frame)
   int nIdentifier = 0x25; // send with CAN ID 0x25;

   int nRC = CANOpenDevice(&hHandle, TRUE, TRUE, 0, "192.168.0.254", 5000);
   if ( nRC == 0 )
   {
       // send 8 bytes with CAN id 37
       nRC = CANWrite( hHandle, nIdentifier, cMsg, 8, nFlags );

       // send a remote frame to CAN id 37 (request 4 data bytes)
       nRC = CANWrite( hHandle, nIdentifier, cMsg, 4, 0x02 );

       CANCloseDevice(hHandle);
   }
   return 0;
}
```

Remarks

The `CANWriteEx` function is supported by library version 1.4-1.8 or higher.

For devices of type AnaGate CAN (hardware version 1.1.A) the function `CANWriteEx` is equal to `CANWrite`, the return values *pnSeconds* and *pnMicroSeconds* will remain unchanged.

 © 2007-2010 Analytica GmbH

CANSetCallback, CANSetCallbackEx

CANSetCallback, CANSetCallbackEx — Defines an asynchronous callback function, which is called for each incoming CAN telegram.

Syntax

```
#include <AnaGateDllCan.h>

typedef void (WINAPI * CAN_PF_CALLBACK)(int nIdentifier, const char *
pcBuffer, int nBufferLen, int nFlags, int hHandle);

int CANSetCallback(int hHandle, CAN_PF_CALLBACK pCallbackFunction);

typedef void (WINAPI * CAN_PF_CALLBACK_EX)(int nIdentifier, const char
* pcBuffer, int nBufferLen, int nFlags, int hHandle, long nSeconds,
long nMicroseconds);

int CANSetCallbackEx(int hHandle, CAN_PF_CALLBACK_Ex
pCallbackFunctionEx);
```

Parameter

hHandle	Valid access handle.
pCallbackFunction	Function pointer to the private callback function. Set this parameter to `NULL` to deactivate the callback function. The parameters of the callback function are described in the documentation of the `CANWrite` funciton.
pCallbackFunctionEx	Function pointer to the private callback function. Set this parameter to `NULL` to deactivate the callback function. The parameters of the callback function are described in the documentation of the `CANWriteEx` funciton.

Return value

Returns `Null` if successful, or an error value otherwise (Appendix A, *API return codes*).

Description

Incoming CAN telegrams can bei received via a callback function, which can be set by a simple API call. If a callback function us set, is will be called by the API asynchronous.

Caution

The callback fucntion is called from a thread which is started by the CAN API and which is reading data from the socket. Because of this behaviour the callback code is executed by the thread context of the API and therefore it uses it the heap memeory of the API DLL and not

 © 2007-2010 Analytica GmbH

the application program. So programming code should not use functions like `new`, `delete`, `alloc` or `free` which allocate, free or reallocate heap memory inside the callback.

See the following example for using a callback.

```
#include <AnaGateCANDll.h>

// Defintion of a callback, which writes incoming CAN data with timestamp to console
void WINAPI MyCallbackEx(int nIdentifier, const char * pcBuffer, int nBufferLen, int nFlags,
                         int hHandle, long nSeconds, long nMicroseconds)
{
   std::cout << "CAN-ID=" << nIdentifier << ", Data=";
   for ( int i = 0; i < nBufferLen; i++ )
   {
      std::cout << " 0x" << std::hex << int((unsigned char)(pcBuffer[i]));
   }
   time_t tTime = nSeconds;
   struct tm * psLocalTime = localtime(&tTime );
   std::cout << " " << std::setw(19) << asctime( psLocalTime ) << " ms(" << std::dec
             << std::setw(3) << nMicroseconds/1000 << "." << nMicroseconds%1000 << ")" << std::endl;
}

int main()
{
   int hHandle = 0;
   int nRC = 0;

   int nRC = CANOpenDevice(&hHandle, TRUE, TRUE, 0, "192.168.0.254", 5000);
   if ( nRC == 0 )
   {
       // deactivate callback
       nRC = CANSetCallbackEx( hHandle, MyCallbackEx );

       getch();  // wait for keyboard input

       // deactivate callback
       nRC = CANSetCallbackEx( hHandle, 0 );

       CANCloseDevice(hHandle);
   }
   return 0;
}
```

Remarks

The two different callback functions have to be used depending on the active setting of the global timestamp option (`CANSetGlobals`). Only one of the callbacks can be activated at the same time.

See also

CANWrite, CANWriteEx

 © 2007-2010 Analytica GmbH

CANReadDigital

CANReadDigital — Reads the current values of digital input and output registers of the AnaGate device.

Syntax

#include <AnaGateDllCan.h>

```
int CANReadDigital(int hHandle, unsigned long * pnInputBits, unsigned long * pnOutputBits);
```

Parameter

hHandle	Valid access handle.
pnInputBits	Pointer to the current value of the digital input register. Currently only bits 0 to 3 are used, other bits are reserved for future use and are set to 0.
pnOutputBits	Pointer to the current value of the digital output register. Currently only bits 0 to 3 are used, other bits are reserved for future use and are set to 0.

Return value

Returns `Null` if successful, or an error value otherwise (Appendix A, *API return codes*).

Description

All models of the AnaGate series (except the model AnaGate CAN uno in DIN rail case) have connectors for 4 digtial inputs and 4 digital outputs at the rear panel.

The current values of the digital inputs and outputs can be retrieved with the `CANReadDigital` function.

See the following example for setting an reading the digital IO.

```
#include <AnaGateCANDll.h>
int main()
{
   int hHandle = 0;
   int nRC = 0;
   unsigned long nInputs;
   unsigned long nOutputs = 0x03;

   int nRC = CANOpenDevice(&hHandle, TRUE, TRUE, 0, "192.168.0.254", 5000);
   if ( nRC == 0 )
   {
       // set the digital output register (PIN 0 and PIN 1 to HIGH value)
       nRC = CANWriteDigital( hHandle, nOutputs );

       // read all input and output registers
```


© 2007-2010 Analytica GmbH

```
        nRC = CANReadDigital( hHandle, &nInputs, &nOutputs );

        CANCloseDevice(hHandle);
    }
    return 0;
}
```

See also

CANWriteDigital

© 2007-2010 Analytica GmbH

CANWriteDigital

CANWriteDigital — Writes a new value to the digital output register of the AnaGate device.

Syntax

```
#include <AnaGateDllCAN.h>

int CANWriteDigital(int hHandle, unsigned long nOutputBits);
```

Parameter

hHandle — Valid access handle.

nOutputBits — New register value. Currently only bits 0 to 3 are used, other bits are reserved for future use.

Return value

Returns `Null` if successful, or an error value otherwise (Appendix A, *API return codes*).

Description

All models of the AnaGate series (except the model AnaGate CAN uno in DIN rail case) have connectors for 4 digtial inputs and 4 digital outputs ant the rear panel.

The digital outputs can be written with the `CANWriteDigital` function.

A simple example for reading/writing of the IOs can be found at the description of `CANReadDigital`.

See also

CANReadDigital

 © 2007-2010 Analytica GmbH

CANRestart

CANRestart — Restarts a AnaGate CAN device.

Syntax

#include <AnaGateDllCan.h>

```
int CANRestart(const char * pcIPAddress, int nTimeout );
```

Parameter

pcIPAddress — Network address of the AnaGate partner.

nTimeout — Default timeout for accessing the AnaGate in milliseconds. A timeout is reported if the AnaGate partner does not respond within the defined timeout period.

Return value

Returns `Null` if successful, or an error value otherwise (Appendix A, *API return codes*).

Description

Restarts the AnaGate CAN device at the specified network address. It disconnects implicitly all open network connections to all existing CAN interfaces. The Restart command is even possible if the maximum number of allowed connections is reached.

Important

It is recommended to use this command only in emergency cases, if there is a need to connect even if the maximum number of concurrent connections is reached.

See also

CANOpenDevice

© 2007-2010 Analytica GmbH

CANDeviceConnectState

CANDeviceConnectState — Retrieves the current network connection state of the current AnaGate connection.

Syntax

#include <AnaGateDllCAN.h>

```
int CANDeviceConnectState(int hHandle);
```

Parameter

hHandle   Valid connection handle of a successful call to CANOpenDevice.

Return value

Returns the current network connection state. Following values are possible:

- `1 = DISCONNECTED`: The connection to the AnaGate is disconnected.
- `2 = CONNECTING`: The connection is connecting.
- `3 = CONNECTED` : The connection is established.
- `4 = DISCONNECTING`: The connection is disonnecting.
- `5 = NOT_INITIALIZED`: The network protocol is not successfully initialized.

Description

This function can be used to check if an already connected device is disconnected.

The detection period of a state change depends on the use of internal AnaGate-ALIVE mechanism. This ALIVE mechanism has to be switched on explicitly via `CANStartAlive` function. Once activated the connection state is periodically checked by the ALIVE mechanism.

Remarks

The `CANDeviceConnectState` function is supported by library version 1.4-1.10 or higher.

See also

CANStartAlive

© 2007-2010 Analytica GmbH

CANStartAlive

CANStartAlive — Starts the ALIVE mechanism, which checks periodically the state of the network connection to the AnaGate hardware.

Syntax

```
#include <AnaGateDllCan.h>

int CANStartAlive(int hHandle, int nAliveTime );
```

Parameter

hHandle Valid access handle.

nAliveTime Default time out in seconds for the ALIVE mechanism.

Return value

Returns `Null` if successful, or an error value otherwise (Appendix A, *API return codes*).

Description

The AnaGate communication protocol (see [TCP-2010]) supports an application specific connection control which allows faster detection of broken connection lines.

The `CANStartAlive` function starts a concurrent process in the DLL in order to send defined alive telegrams (ALIVE_REQ) peridically (approx. every half of the given time out) to the Anagate device via the current network connection. Is the alive telegram not confirmed within the alive time the connection is marked as `disconnected` and the socket is closed if not already closed.

Use the `CANDeviceConnectState` function to check the current network connection state.

Remarks

The `CANStartAlive` function is supported by library version 1.4-1.10 or higher.

It requires firmware version 1.3.8 or higher installed on the hardware, devices of type AnaGate CAN (hardware version 1.1.A) does not support the application specific alive mechanism.

See also

CANDeviceConnectState

Section 2.1, “ Important properties of the network protocol”

 © 2007-2010 Analytica GmbH

CANErrorMessage

CANErrorMessage — Returns a description of the given error code as a text string.

Syntax

#include <AnaGateDllCAN.h>

```
int CANErrorMessage(int nRetCode, char * pcMessage, int nMessageLen);
```

Parameter

nRetCode   Error code for which the error description is to be determined.

pcMessage   Data buffer that is to accept the error description.

nMessageLen   Size in bytes of the transferred data buffer.

Return value

Actual size of the returned description.

Description

Returns a textual description of the parsed error code (see Appendix A, *API return codes*). If the destination buffer is not big enough to store the text, the text is shortened to the specified buffer size.

See the following example in C/C++ language.

```
int nRC;
char cBuffer[200];
int nRC;
//... call a API function here
CANErrorMessage(nRC, cBuffer, 200);
std::cout << "Fehler: " << cBuffer << std::endl;
```

 © 2007-2010 Analytica GmbH

Chapter 5. SPI API reference

The Serial Peripheral Interface (SPI) is a synchroneous data link standard named by Motorola which operates in full duplex mode. The SPI gateway models of the AnaGate series provides access to a SPI bus via a standard networking.

With the SPI API these SPI gateways can be easily controlled. The programming interface is identical for all devices and used the network protocol TCP in general.

Following devices can be addresse via the SPI API interface:

- AnaGate SPI
- AnaGate Universal Programmer

© 2007-2010 Analytica GmbH

SPIOpenDevice

SPIOpenDevice — Opens a network connection to an AnaGate SPI device.

Syntax

#include <AnaGateDllSPI.h>

`int SPIOpenDevice(int * pHandle, const char * pcIPAddress, int nTimeout);`

Parameter

pHandle	Pointer to a variable, in which the access handle is saved in the event of a successful connection to the AnaGate device.
pcIPAddress	Network address of the AnaGate partner.
nTimeout	Default timeout for accessing the AnaGate in milliseconds. A timeout is reported if the AnaGate partner does not respond within the defined timeout period. This global timeout value is valid on the current network connection for all commands and functions which do not offer a specific timeout value.

Return value

Returns `Null` if successful, or an error value otherwise (Appendix A, *API return codes*).

Description

Opens a TCP/IP connection to an AnaGate SPI (resp. AnaGate Universal Programmer). After the connection is established, access to the SPI bus is possible.

Note

The AnaGate SPI (resp. the SPI interface of an AnaGate Universal Programmer) does not allow more than one concurrent network connection. During an established network connection all new connections are refused.

See the following example for the initial programming steps.

```
#include <AnaGateDllSPI.h>
int main()
{
   int hHandle;
   int nRC = SPIOpenDevice(&hHandle, "192.168.0.254", 5000);
   if ( nRC == 0 )
   {
       // ... now do something
       SPICloseDevice(hHandle);
   }
```

 © 2007-2010 Analytica GmbH

```
    return 0;
}
```

See also

SPICloseDevice

© 2007-2010 Analytica GmbH

SPICloseDevice

SPICloseDevice — Closes an open network connection to an AnaGate SPI device.

Syntax

#include <AnaGateDllSPI.h>

```
int SPICloseDevice(int hHandle);
```

Parameter

hHandle Valid access handle.

Return value

Returns `Null` if successful, or an error value otherwise (Appendix A, *API return codes*).

Description

Closes an open network connection to an AnaGate SPI device. The *`hHandle`* parameter is a return value of a succesfull call to the function `SPIOpenDevice`.

Important

It is recommended to close the connection, because all internally allocated system resources are freed again and the connected AnaGate device is signalled that the active connection is not longer in use and can be used again for new connect requests.

See also

SPIOpenDevice

© 2007-2010 Analytica GmbH

SPISetGlobals

SPISetGlobals — Sets the global settings, which are to be used on the AnaGate SPI.

Syntax

#include <AnaGateDllSPI.h>

```
int SPISetGlobals(int hHandle, int nBaudrate, unsigned char nSigLevel,
unsigned char nAuxVoltage, unsigned char nClockMode);
```

Parameter

hHandle Valid access handle.

nBaudrate The baud rate to be used. The values can be set individually, like

- 500.000 for 500kBit
- 1.000.000 for 1MBit
- 5.000.000 for 5MBit

Note

The required baud rate can be different from the value actually used because of internal hardware restrictions (frequency of the oscillator). If it is not possible to adjust the baud rate exactly to the parsed value, the nearest smaller possible value is used instead.

nSigLevel The voltage level for SPI signals to be used. Following values are allowed:

- 0 = Outputs in High Impedance Modus (Standard mode).
- 1 = +5.0V for the signals.
- 2 = +3.3V for the signals.
- 3 = +2.5V for the signals.

nAuxVoltage The voltage level of the support voltage to be used. Following values are allowed:

- 0 = support voltage is +3.3V.
- 1 = support voltage is 2.5V.

nClockMode The phase and polarity of the clock signal. Following values are allowed:

- 0 = CPHA=0 and CPOL=0.
- 1 = CPHA=0 and CPOL=1.

 © 2007-2010 Analytica GmbH

- 2 = CPHA=1 and CPOL=0.
- 3 = CPHA=1 and CPOL=1.

Return value

Returns `Null` if successful, or an error value otherwise (Appendix A, *API return codes*).

Description

Sets the global settings of SPI interface of the AnaGate SPI or the AnaGate Universal Programmer. These settings are not saved permantently on the device and are reset every device restart.

See also

SPIGetGlobals

 © 2007-2010 Analytica GmbH

SPIGetGlobals

SPIGetGlobals — Returns the currently used global settings of the AnaGate SPI.

Syntax

#include <AnaGateDllSPI.h>

```
int SPIGetGlobals(int hHandle, int * pnBaudrate, unsigned char * pnSigLevel, unsigned char * pnAuxVoltage, unsigned char * pnClockMode);
```

Parameter

hHandle Valid access handle.

pnBaudrate The baud rate currently used on the SPI bus in kBit.

pnSigLevel The voltage level currently used by the AnaGate SPI. Following values are possible:

- 0 = Outputs in High Impedance Modus (Standard mode).
- 1 = +5.0V for the signals.
- 2 = +3.3V for the signals.
- 3 = +2.5V for the signals.

pnAuxVoltage The voltage level of the support voltage currently used by the AnaGate SPI. Following values are possible:

- 0 = support voltage is +3.3V.
- 1 = support voltage is 2.5V.

pnClockMode The phase and polarity of the colck signal currently used by the AnaGate SPI. Following values are possible:

- 0 = CPHA=0 and CPOL=0.
- 1 = CPHA=0 and CPOL=1.
- 2 = CPHA=1 and CPOL=0.
- 3 = CPHA=1 and CPOL=1.

Return value

Returns `Null` if successful, or an error value otherwise (Appendix A, *API return codes*).

Description

Returns the currently used global settings of SPI interface of the AnaGate SPI or the AnaGate Universal Programmer.

 © 2007-2010 Analytica GmbH

See also

SPISetGlobals

© 2007-2010 Analytica GmbH

SPIDataReq

SPIDataReq — Writes and reads data to/from SPI bus.

Syntax

#include <AnaGateDllSPI.h>

`int SPIDataReq(int hHandle, const char * pcBufWrite, int nBufWriteLen, char * pcBufRead, int nBufReadLen);`

Parameter

hHandle	Valid access handle.
pcBufWrite	Buffer with the data that is to be sent to the SPI partner.
nBufWriteLen	Length of the data buffer *pcBufWrite* (byte count).
pcBufRead	Byte buffer which holds the data received from the SPI partner.
nBufReadLen	Number of bytes to read.

Description

Sends data to the SPI bus and receives data from the SPI bus.

On the SPI bus Data is transferred on two seperates data lines full duplex (SDO and SDI). The `SPIDatReq` has to split a single data transfer in two steps because of the spacial separation to the SPI bus. First the write data buffer is put into a TCP data telegram and sent to the AnaGate SPI. The AnaGate SPI makes the real data transfer on the SPI bus and send back a confirmation including the data received from the bus.

Important

It is impossible to detect that no device is present at the SPI bus. So, if no device is attached, the requested number of bytes are returned anyway - in this case the read buffer is filled with `0`.

See the following example for sending a command to the connected SPI bus.

```
#include <AnaGateDllSPI.h>
int main()
{
   char cBufWrite[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
   char cBufReceive[100];
   int hHandle = 0;
   int nRC = 0;

   int nRC = SPIOpenDevice(&hHandle, "192.168.1.254", 5000);
   if ( nRC == 0 )
   {
       // send 1 byte and receive 1 byte
```

 © 2007-2010 Analytica GmbH

```
        nRC = SPIDataReq( hHandle, cBufWrite, 1, cBufReceive, 1 );
        // send 1 byte and receive 5 byte
        nRC = SPIDataReq( hHandle, cBufWrite, 1, cBufReceive, 5 );
        // send 2 byte and receive 1 byte
        nRC = SPIDataReq( hHandle, cBufW2ite, 2, cBufReceive, 1 );

        SPICloseDevice(hHandle);
    }
    return 0;
}
```

Return value

Returns `Null` if successful, or an error value otherwise (Appendix A, *API return codes*).

 © 2007-2010 Analytica GmbH

SPIReadDigital

SPIReadDigital — Reads the current values of digital input and output registers of the AnaGate device.

Syntax

```
#include <AnaGateDllSPI.h>
```

```
int SPIReadDigital(int hHandle, unsigned long * pnInputBits, unsigned long * pnOutputBits);
```

Parameter

hHandle	Valid access handle.
pnInputBits	Pointer to the current value of the digital input register. Currently only bits 0 to 3 are used, other bits are reserved for future use and are set to 0.
pnOutputBits	Pointer to the current value of the digital output register. Currently only bits 0 to 3 are used, other bits are reserved for future use and are set to 0.

Return value

Returns `Null` if successful, or an error value otherwise (Appendix A, *API return codes*).

Description

All models of the AnaGate series (except the model AnaGate CAN uno in DIN rail case) have connectors for 4 digtial inputs and 4 digital outputs at the rear panel.

The current values of the digital inputs and outputs can be retrieved with the `SPIReadDigital` function.

See the following example for setting an reading the digital IO.

```
#include <AnaGateDllSPI.h>
int main()
{
   int hHandle = 0;
   int nRC = 0;
   unsigned long nInputs;
   unsigned long nOutputs = 0x03;

   int nRC = SPIOpenDevice(&hHandle, "192.168.0.254", 5000);
   if ( nRC == 0 )
   {
       // set the digital output register (PIN 0 and PIN 1 to HIGH value)
       nRC = SPIWriteDigital( hHandle, nOutputs );

       // read all input and output registers
```


© 2007-2010 Analytica GmbH

```
        nRC = SPIReadDigital( hHandle, &nInputs, &nOutputs );

        SPICloseDevice(hHandle);
    }
    return 0;
}
```

See also

SPIWriteDigital

 © 2007-2010 Analytica GmbH

SPIWriteDigital

SPIWriteDigital — Write a new value to the digital output register of the AnaGate device.

Syntax

#include <AnaGateDllSPI.h>

```
int SPIWriteDigital(int hHandle, unsigned long nOutputBits);
```

Parameter

hHandle	Valid access handle.
nOutputBits	New register value. Currently only bits 0 to 3 are used, other bits are reserved for future use.

Return value

Returns `Null` if successful, or an error value otherwise (Appendix A, *API return codes*).

Description

All models of the AnaGate series (except the model AnaGate CAN uno in DIN rail case) have connectors for 4 digtial inputs and 4 digital outputs at the rear panel.

The digital outputs can be written with the `SPIWriteDigital` function.

A simple example for reading/writing of the IOs can be found at the description of `SPIReadDigital`.

See also

SPIReadDigital

© 2007-2010 Analytica GmbH

SPIErrorMessage

SPIErrorMessage — Returns a description of the given error code as a text string.

Syntax

#include <AnaGateDllCAN.h>

```
int SPIErrorMessage(int nRetCode, char * pcMessage, int nMessageLen);
```

Parameter

nRetCode   Error code for which the error description is to be determined.

pcMessage   Data buffer that is to accept the error description.

nMessageLen   Size in bytes of the transferred data buffer.

Return value

Actual size of the returned description.

Description

Returns a textual description of the parsed error code (see Appendix A, *API return codes*). If the destination buffer is not big enough to store the text, the text is shortened to the specified buffer size.

See the following example in C/C++ language.

```
int nRC;
char cBuffer[200];
int nRC;
//... call a API function here
SPIErrorMessage(nRC, cBuffer, 200);
std::cout << "Fehler: " << cBuffer << std::endl;
```

 © 2007-2010 Analytica GmbH

Chapter 6. I2C API reference

Philips Semiconductors (now NXP Semiconductors) has developed a simple bidirectional 2-wire bus for efficient inter-IC control. This bus is called the Inter-IC or I2C-bus. Only two bus lines are required: a serial data line (SDA) and a serial clock line (SCL). Serial, 8-bit oriented, bidirectional data transfers can be made at up to 100 kbit/s in the Standard-mode, up to 400 kbit/s in the Fast-mode, up to 1 Mbit/s in the Fast-mode Plus (Fm+), or up to 3.4 Mbit/s in the High-speed mode. [NXP-I2C].

The I2C gateway models of the AnaGate series provides access to a I2C bus via a standard networking. With the I2C API these I2C gateways can be easily controlled. The programming interface is identical for all devices and used the network protocol TCP in general.

Following devices can be addresse via the I2C API interface:

- AnaGate I2C
- AnaGate Universal Programmer

© 2007-2010 Analytica GmbH

I2COpenDevice

I2COpenDevice — Opens a network connection to an AnaGate I2C or an AnaGate Universal Programmer).

Syntax

#include <AnaGateDllI2C.h>

```
int I2COpenDevice(int * pHandle, unsigned int nBaudrate, const char * pcIPAddress, int nTimeout);
```

Parameter

pHandle
: Pointer to a variable, in which the access handle is saved in the event of a successful connection to the AnaGate device.

nBaudrate
: Baud rate to be used for the I2C bus. Teh value can be set individually, like

- 100000 for 100kBit (standard mode)
- 400000 for 400kBit (fast mode)

Note

Values above 400kBit are ignored by the AnaGate SPI.

pcIPAddress
: Network address of the AnaGate partner.

nTimeout
: Default timeout for accessing the AnaGate in milliseconds.

A timeout is reported if the AnaGate partner does not respond within the defined timeout period. This global timeout value is valid on the current network connection for all commands and functions which do not offer a specific timeout value.

Return value

Returns `Null` if successful, or an error value otherwise (Appendix A, *API return codes*).

Description

Opens a TCP/IP connection to an AnaGate I2C (resp. AnaGate Universal Programmer). After the connection is established, access to the I2C bus is possible.

Note

The AnaGate I2C (resp. the I2C interface of an AnaGate Universal Programmer) does not allow more than one concurrent network connection. During an established network connection all new connections are refused.

 © 2007-2010 Analytica GmbH

See the following example for the initial programming steps.

```
#include <AnaGateDllI2C.h>
int main()
{
   int hHandle;
   int nRC = I2COpenDevice(&hHandle, 100000, "192.168.0.254", 5000);
   if ( nRC == 0 )
   {
       // ... now do something
       I2CCloseDevice(hHandle);
   }
   return 0;
}
```

See also

I2CCloseDevice

© 2007-2010 Analytica GmbH

I2CCloseDevice

I2CCloseDevice — Closes an open network connection to an AnaGate I2C device.

Syntax

#include <AnaGateDllI2C.h>

```
int I2CCloseDevice(int hHandle);
```

Parameter

hHandle Valid access handle.

Return value

Returns `Null` if successful, or an error value otherwise (Appendix A, *API return codes*).

Description

Closes an open network connection to an AnaGate I2C device. The *`hHandle`* parameter is a return value of a succesfull call to the function `I2COpenDevice`.

Important

It is recommended to close the connection, because all internally allocated system resources are freed again and the connected AnaGate device is signalled that the active connection is not longer in use and can be used again for new connect requests.

See also

I2COpenDevice

© 2007-2010 Analytica GmbH

I2CReset

I2CReset — Resets the I2C Controller in an AnaGate I2C device.

Syntax

```
#include <AnaGateDllI2C.h>

int I2CReset(int hHandle);
```

Parameter

hHandle Valid access handle.

Return value

Returns `Null` if successful, or an error value otherwise (Appendix A, *API return codes*).

Description

Resets the I2C Controller in an AnaGate I2C device.

 © 2007-2010 Analytica GmbH

I2CRead

I2CRead — Reads data from an I2C partner.

Syntax

#include <AnaGateDllI2C.h>

```
int I2CRead(int hHandle, unsigned short nSlaveAddress, const char *
pcBuffer, int nBufferLen);
```

Parameter

hHandle	Valid access handle.
nSlaveAddress	Slave address of the I2C partner. The slave address can represent a so-called 7-bit or 10-bit address. (siehe Appendix B, *I2C slave address formats*).
pcBuffer	Byte buffer in which the data received from the I2C partner is to be stored.
nBufferLen	Number of bytes to read.

Return value

Returns `Null` if successful, or an error value otherwise (Appendix A, *API return codes*).

Description

Reads data from an I2C partner. The user must ensure that the setup of the data buffer and the address of the I2C partner are correct.

The R/W bit of the slave address does not have to be explicitly set by the user.

See also

I2CWrite

 © 2007-2010 Analytica GmbH

I2CWrite

I2CWrite — Writes data to an I2C partner.

Syntax

#include <AnaGateDllI2C.h>

```
int I2CWrite(int hHandle, unsigned short nSlaveAddress, const char *
pcBuffer, int nBufferLen);
```

Parameter

hHandle	Valid access handle.
nSlaveAddress	Slave address of the I2C partner. The slave address can represent a so-called 7-bit or 10-bit address. (siehe Appendix B, *I2C slave address formats*).
pcBuffer	Byte buffer with the data that is to be sent to the I2C partner.
nBufferLen	Size of bytes to be read.

Return value

Returns `Null` if successful, or an error value otherwise (Appendix A, *API return codes*).

Description

Writes data to an I2C partner. The user must ensure that the setup of the data buffer and the address of the I2C partner are correct.

The R/W bit of the slave address does not have to be explicitly set by the user.

See also

I2CRead

 © 2007-2010 Analytica GmbH

I2CSequence

I2CSequence — This command is used to write a sequence of write and read commands to an I2C device.

Syntax

#include <AnaGateDllI2C.h>

```
int I2CSequence(int hHandle, const char * pcWriteBuffer, int
nNumberOfBytesToWrite, char * pcReadBuffer, int nNumberOfBytesToRead,
int * pnNumberOfBytesRead, int * pnByteNumberLastError);
```

Parameter

hHandle

Valid access handle.

pcWriteBuffer

byte buffer, containing the commands which are to be sent to the AnaGate I2C. The single commands are stored sequential in this byte buffer.

A read command is defined as follows:

Structure of read command for I2CSequence

Read command	Description
2 bytes (LSB,MSB)	slave address in 7- or 10-bit format, the R/W bit must be set to explicitely to `1`.
2 bytes (LSB,MSB)	Bit 0-14: number of data bytes to be read from the I2C device. The successfully read data bytes are stored in the *pcReadBuffer* buffer. Bit 15: If this bit is set then the stop bit at the end of the read command is omitted.

A write command is defined as follows:

Structure write command for I2CSequence

Write command	Description
2 bytes (LSB,MSB)	slave address in 7- or 10-bit format, the R/W bit must be set to explicitely to `1`.
2 bytes (LSB,MSB)	Bit 0-14: number of data bytes to be written to the I2C device. Bit 15: If this bit is set then the stop bit at the end of the write command is omitted.

 © 2007-2010 Analytica GmbH

Write command	Description
N bytes	data bytes.

nNumberOfBytesToWrite — byte size of the data to write

pcReadBuffer — byte buffer, in which the received data is to be stored. The received data from different commands are stored in the buffer sequential (first the data of command 1, then the data of command 2, ...).

nNumberOfBytesRead — byte size of the read buffer (must be big enough for all included read requests)

pnNumberOfBytesRead — byte count, which is read from I2C.

pnByteNumberLastError — Number of byte in the *pcWriteBuffer* buffer, which raises an error.

Return value

Returns `Null` if successful, or an error value otherwise (Appendix A, *API return codes*).

Description

The user must ensure that the setup of the data buffer and the address of the I2C partner are correct.

© 2007-2010 Analytica GmbH

I2CReadDigital

I2CReadDigital — Reads the current values of digital input and output registers of the AnaGate device.

Syntax

#include <AnaGateDllI2C.h>

`int **I2CReadDigital**(int *hHandle*, unsigned long * *pnInputBits*, unsigned long * *pnOutputBits*);`

Parameter

hHandle — Valid access handle.

pnInputBits — Pointer to the current value of the digital input register. Currently only bits 0 to 3 are used, other bits are reserved for future use and are set to 0.

pnOutputBits — Pointer to the current value of the digital output register. Currently only bits 0 to 3 are used, other bits are reserved for future use and are set to 0.

Return value

Returns `Null` if successful, or an error value otherwise (Appendix A, *API return codes*).

Description

All models of the AnaGate series (except the model AnaGate CAN uno in DIN rail case) have connectors for 4 digtial inputs and 4 digital outputs at the rear panel.

The current values of the digital inputs and outputs can be retrieved with the `I2CReadDigital` function.

See the following example for setting an reading the digital IO.

```
#include <AnaGateDllI2C.h>
int main()
{
   int hHandle = 0;
   int nRC = 0;
   unsigned long nInputs;
   unsigned long nOutputs = 0x03;

   int nRC = I2COpenDevice(&hHandle, 400000, "192.168.0.254", 5000);
   if ( nRC == 0 )
   {
       // set the digital output register (PIN 0 and PIN 1 to HIGH value)
       nRC = I2CWriteDigital( hHandle, nOutputs );

       // read all input and output registers
```

 © 2007-2010 Analytica GmbH

```
        nRC = I2CReadDigital( hHandle, &nInputs, &nOutputs );

        CANCloseDevice(hHandle);
    }
    return 0;
}
```

See also

I2CWriteDigital

© 2007-2010 Analytica GmbH

I2CWriteDigital

I2CWriteDigital — Writes a new value to the digital output register of the AnaGate device.

Syntax

```
int I2CWriteDigital(int hHandle, unsigned long nOutputBits);
```

Parameter

hHandle	Valid access handle.
nOutputBits	New register value. Currently only bits 0 to 3 are used, other bits are reserved for future use.

Return value

Returns `Null` if successful, or an error value otherwise (Appendix A, *API return codes*).

Description

All models of the AnaGate series (except the model AnaGate CAN uno in DIN rail case) have connectors for 4 digtial inputs and 4 digital outputs at the rear panel.

The digital outputs can be written with the `I2CWriteDigital` function.

A simple example for reading/writing of the IOs can be found at the description of `I2CReadDigital`.

See also

I2CReadDigital

© 2007-2010 Analytica GmbH

I2CErrorMessage

I2CErrorMessage — Returns a description of the given error code as a text string.

Syntax

#include <AnaGateDllCAN.h>

`int I2CErrorMessage(int nRetCode, char * pcMessage, int nMessageLen);`

Parameter

nRetCode   Error code for which the error description is to be determined.

pcMessage   Data buffer that is to accept the error description.

nMessageLen   Size in bytes of the transferred data buffer.

Return value

Actual size of the returned description.

Description

Returns a textual description of the parsed error code (see Appendix A, *API return codes*). If the destination buffer is not big enough to store the text, the text is shortened to the specified buffer size.

See the following example in C/C++ language.

```
int nRC;
char cBuffer[200];
int nRC;
//... call a API function here
I2CErrorMessage(nRC, cBuffer, 200);
std::cout << "Fehler: " << cBuffer << std::endl;
```


© 2007-2010 Analytica GmbH

I2CReadEEPROM

I2CReadEEPROM — Reads data from an EEPROM on the I2C bus.

Syntax

```
#include <AnaGateDllI2C.h>

int I2CReadEEPROM(int hHandle, unsigned short nSubAddress, unsigned int nOffset, const char * pcBuffer, int nBufferLen, unsigned int nOffsetFormat);
```

Parameter

hHandle
: Valid access handle.

nSubAddress
: Subaddress of the EEPROM to communicate with. The valid values for *nSubAddress* are governed by the setting used in the parameter *nOffsetFormat* (bits 8-10). If the EEPROM type needs to use bits of the *Chip Enable Address* to address the internal memory, only the remaining bits can be used to select the device itself.

- No bit is used for addressing: 0 to 7
- 1 bit is used for addressing: 0 to 3
- 2 bits are used for addressing: 0 to 1
- 3 bits are used for addressing: 0

nOffset
: Data offset on the EEPROM from which the transferred data is to be read.

pcBuffer
: Character string buffer in which the received data is to be stored.

nBufferLen
: Length of the data buffer.

nOffsetFormat
: Defines how a memory address of EEPROM has to be specified when accessing the device.

Bits 0-7 indicate the number of bits which are used in the address byte (word) of the I2C command for addressing the device memory.

Bits 8-10 indicate how much and which bits of the *Chip Enable Bits* are used for addressing the device memory (see Table C.1, “ Usage of the CHIP-Enable Bits of I2C EEPROMs” for allowed values).

Note

The maximum addressable size of an EEPROM is derived from the sum of all the bits. For example a M24C08 uses 8 bits of the address byte and an extra bit in the slave address. The total 9 bits can address up to 512 bytes.

© 2007-2010 Analytica GmbH

Return value

Returns `Null` if successful, or an error value otherwise (Appendix A, *API return codes*).

Description

The `I2CReadEEPROM` function reads data from an I2C EEPROM.

Of course all access to the memory of an EEPROM is done by standard I2C read or write commands. So, when reading from the memory only the matching slave address, the memory offset address and the data has to be sent to the I2C bus.

`I2CReadEEPROM` translates the given memory address on the chip by means of the sub address and the addressing mode of the present EEPROM type. The slave address of the EEPROM is automatically determined and not mandatory for the fucntion call.

A programming example which clears a **ST24C1024** can be found at the description of I2WriteEEPROM.

See also

I2CWriteEEPROM

Appendix C, *Programming I2C EEPROM*

© 2007-2010 Analytica GmbH

I2CWriteEEPROM

I2CWriteEEPROM — Writes data to an I2C EEPROM.

Syntax

#include <AnaGateDllI2C.h>

```
int I2CWriteEEPROM(int hHandle, unsigned short nSubAddress, unsigned
int nOffset, const char * pcBuffer, int nBufferLen, unsigned int
nOffsetFormat);
```

Parameter

hHandle
: Valid access handle.

nSubAddress
: Subaddress of the EEPROM to communicate with. The valid values for `nSubAddress` are governed by the setting used in the parameter `nOffsetFormat` (bits 8-10). If the EEPROM type needs to use bits of the *Chip Enable Address* to address the internal memory, only the remaining bits can be used to select the device itself.

  - No bit is used for addressing: 0 to 7
  - 1 bit is used for addressing: 0 to 3
  - 2 bits are used for addressing: 0 to 1
  - 3 bits are used for addressing: 0

nOffset
: Data offset on the EEPROM to which the transferred data is to be written.

pcBuffer
: Character string buffer with the data that is to be written.

nBufferLen
: Length of the data buffer.

nOffsetFormat
: Defines how a memory address of EEPROM has to be specified when accessing the device.

  Bits 0-7 indicate the number of bits which are used in the address byte (word) of the I2C command for addressing the device memory.

  Bits 8-10 indicate how much and which bits of the *Chip Enable Bits* are used for addressing the device memory (see Table C.1, “ Usage of the CHIP-Enable Bits of I2C EEPROMs” for allowed values).

  Note

  The maximum addressable size of an EEPROM is derived from the sum of all the bits. For example a M24C08 uses 8 bits of the address byte and an extra bit in the slave address. The total 9 bits can address up to 512 bytes.

 © 2007-2010 Analytica GmbH

Return value

Returns `Null` if successful, or an error value otherwise (Appendix A, *API return codes*).

Description

The `I2CWriteEEPROM` function writes data to an I2C EEPROM.

Of course all access to the memory of an EEPROM is done by standard I2C read or write commands. So, when writing to the memory only the matching slave address, the memory offset address and the data has to be sent to the I2C bus.

`I2CWriteEEPROM` translates the given memory address on the chip by means of the sub address and the addressing mode of the present EEPROM type. The slave address of the EEPROM is automatically determined and not mandatory for the fucntion call.

Tip

It is important to note that an EEPROM is divided into memory pages, and that a single write command can only program data within a page. Users of `I2CWriteEEPROM` must ensure to do not write across page limits. The page size depends on the EEEPOM type.

See the following example for writing data to a **ST24C1024**.

```
#include <AnaGateDllSPI.h>
int main()
{
   char           cBufferPage[256];
   int            hHandle = 0;
   int            nRC = 0;
   unsigned short nSubAddress = 0; 1
   unsigned int   nOffsetFormat = 0x10|0x0F; 2

   int nRC = I2COpenDevice(&hHandle, 400000, "192.168.0.254", 5000);
   if ( nRC == 0 )
   {
       memset(cBufferPage,0,256);  // clear page buffer
       for (int i=0; i<512;i++)
       {
          I2CWriteEEPROM( hHandle, nSubAddress, i*256, cBufferPage, 256, nOffsetFormat ); 3
       }
       I2CCloseDevice(hHandle);
   }
   return 0;
}
```

1. It is possible to address 4 individual ST24C1024 on a single I2C bus. By selection of subaddress `0` the control pins E2 and E1 have to be LOW.
2. 17 address bits are used to address the 128KB of a ST24C1024. 16 bits are set via the address bytes of the write command: `16=0x0F`. The address bit A16 is set via the E0 bit of the *Chip Enable Address*, therefore addressing mode `1` (E2-E1-A0) must be set: `0x10`.
3. The page size of a ST24C1024 is 256 byte, every page is programmed full within the for-loop.

See also

I2CReadEEPROM

© 2007-2010 Analytica GmbH

Appendix C, *Programming I2C EEPROM*

 © 2007-2010 Analytica GmbH

Chapter 7. Programming examples

7.1. Programming language C/C++

The AnaGate programming API can be used on Windows systems as well as on linux systems (X86). All available API functions are coded operating system independant, so that source code once created can be used on both operating systems. Only the way the libraries are linked on the different operating systems or different compilers have to be customized.

Windows operating systems

There are basically two ways to access the API functions for the C/C++ programmer:

- When directly accessing the library all functions has to made known by preceding calls to the Win32 methods `LoadLibrary` and `GetProcAddress`.
- The functions can be alternatively accessed easily via an import library, which automatically loads all DLL functions and make them available implicitely.

In the following examples it is assumed that the second option is used and the corresponding import library is linked.

7.1.1. CAN Console application C/C++ (MSVC)

This programming example for C++ demonstrates how to connect to an AnaGate CAN and how to process received CAN data via a callback function.

Note

The source code of the example can be found in directory `Samples/CAN-VC6` resp. `Samples/CAN-VC7` on the CD.

```
#include "AnaGateDLLCAN.h"

WINAPI void MyCallback(int nIdentifier, const char * pcBuffer, int nBufLen, int nFlags, int nHandle)
{
    std::cout << "CAN-ID=" << nIdentifier << ", Data=";
    for ( int i = 0; i < nBufLen; i++ )
    {
       std::cout " 0x" << std::hex << pcBuffer[i];
    }
    std::cout << std::endl;
}

int main( )
{
   int hHandle = NULL;

   // opens AnaGate CAN duo device on port A, timeout after 1000 millseconds
   int nRC = CANOpenDevice(&hHandle,FALSE,TRUE,0,"192.168.2.1",1000);

   if ( nRC == 0 )
   {
      nRC = CANSetCallback(hHandle,MyCallback);
      getch();  // wait for keyborad input
   }
```

```
    if ( nRC == 0 )
    {
       nRC = CANCloseDevice(hHandle);// close device
    }
}
```

7.2. Programming language Visual Basic 6

As already described in the previous chapters, the libraries of the *AnaGate*-API use the cdecl calling convention to parse function parameters to the program stack. Unfortunately this is generally not supported by the programming language *Visual Basic 6*.

To work around this limitiation, the libraries for the AnaGate devices are available in a specfic version for programming VB6 applications. In this versions the stdcall calling convention is used, which is the only supported by VB6. Except for the way the parameters are pushed on the stack these specific VB6 versions of the API libraries are exactly identical to the standard versions.

Note

Use library *`AnaGateCAN.dll`* instead of library *`AnaGateCANVB6.dll`*.

Use library *`AnaGateSPI.dll`* instead of library *`AnaGateSPIVB6.dll`*.

7.2.1. SPI Example with user interface for VB6

This programming example for Visual Basisc 6 demonstrates how to connect to an AnaGate SPI and how to execute a command on the SPI bus:

- Getting the global device settings like baud rate for example
- Sending a single command to the SPI bus

Note

The source code of the example can be found in directory `Samples/SPI-VB6` on the CD.

 © 2007-2010 Analytica GmbH

7.2.1.1. User interface

Figure 7.1. Input form of SPI example (VB6)

Dialog fields

Network address
: Network address of the AnaGate SPI.

Check address
: Establishes a connection to the AnaGate SPI with the specified network address and reads back some device information and global device settings.

Baud rate
: The baud rate to be used. The value can be set individually.

Signal Level
: The voltage level for SPI signals to be used.

Aux. Voltage
: The voltage level of the support voltage to be used.

Clock mode
: The phase and polarity of the clock signal.

SPI command
: SPI command to be sent to the connected SPI device. The command has to be entered as blank-separated hexadecimal byte groups ("`05 1F 3A`" for example).

Execute command
: Executes a SPI command and displays the result in the *`Result`* dialog field. Please keep in mind, that the SPI bus is used as full duplex line, this means that data is written and received parallel. Make sure that you write the same number of bytes to the bus as you want to receive (in this cace add padding bytes to the SPI command).

: For example is the **Read Status Register** command of a **M25P80** defined as `0x05`. The result of the command is a single byte (8 bit) representing the current value of the Status Register.

 © 2007-2010 Analytica GmbH

7.2.1.2. Getting global device settings

All SPI related functions of the AnaGate API are declared for Visual Basic users in `AnaGateSPI.bas` and are read-to-use. The following code snipplet includes some exemplary declarations of the API functions used below.

```
Public Declare Function SPIOpenDevice Lib "AnaGateSPIVB6" _
                Alias "_SPIOpenDevice@12" (ByRef Handle As Long, _
                                           ByVal TCPAddress As String, _
                                           ByVal Timeout As Long) As Long

Public Declare Function SPICloseDevice Lib "AnaGateSPIVB6" _
                Alias "_SPICloseDevice@4" (ByVal Handle As Long) As Long

Public Declare Function SPIGetGlobals Lib "AnaGateSPIVB6" _
               Alias "_SPIGetGlobals@20" (ByVal hHandle As Long, _
                                          ByRef nBaudrate As Long, _
                                          ByRef SigLevel As Byte, _
                                          ByRef nAuxVoltage As Byte, _
                                          ByRef nClockMode As Byte) As Long
```

The event procedure `btnCheckAddress_Click` is called on click of the ***Check Address*** button.

```
Private Sub btnCheckAddress_Click()
    Dim nRC As Long, Data As String, sText As String, I As Long

    nRC = SPIOpenDevice(hHandle, Me.IPAddresse.Text, 2000) 1
    If nRC <> 0 Then
        Me.lblErrorMsg.Caption = "Fehler bei SPIOpenDevice: " & GetErrorMsg(nRC)
    Else
        Me.lblErrorMsg.Caption = GetAnagateInfo(hHandle)
    End If
    nRC = SPICloseDevice(hHandle) 2
End Sub
```

1. A call to `SPIOpenDevice` establishes a network connection to the device. If the function fails, a textual error description is returned via Funktion `GetErrorMsg`.
2. The connection to the device is closed with the `SPICloseDevice` function.

Reading the device settings and creation of the textual presentation of the data is done by the `GetAnagateInfo` function.

```
Private Function GetAnagateInfo(hHandle As Long) As String
   Dim nRC As Long, sText As String
   Dim nBaudrate As Long, nSigLevel As Byte, nAuxVoltage As Byte, nClockMode As Byte
   Dim nDigitalOutput As Long, nDigitalInput As Long

   nRC = SPIGetGlobals(hHandle, nBaudrate, nSigLevel, nAuxVoltage, nClockMode)
   If (nRC = 0) Then
        sText = sText & "Baudrate=" & CStr(nBaudrate) & ", Siglevel="
        Select Case nSigLevel
            Case 1: sText = sText & "+5.0V"
            Case 2: sText = sText & "+3.3V"
            Case 3: sText = sText & "+2.5V"
            Case Else: sText = sText & "High impedance"
        End Select
        sText = sText & vbCrLf & "AuxVoltage="
        Select Case nAuxVoltage
            Case 1: sText = sText & "+2.5V"
            Case Else: sText = sText & "+3.3V"
        End Select
        sText = sText & ", ClockMode="
        Select Case nClockMode
            Case 1: sText = sText & "CPHA=0 und CPOL=1"
            Case 2: sText = sText & "CPHA=1 und CPOL=0"
            Case 3: sText = sText & "CPHA=1 und CPOL=1"
            Case Else: sText = sText & "CPHA=0 und CPOL=0"
        End Select
```

 © 2007-2010 Analytica GmbH

```
   Else
      sText = sText & "Fehler bei SPIGetGlobals: " & GetErrorMsg(nRC) & vbCrLf
   End If
   GetAnagateInfo = sText
End Function
```

7.2.1.3. Executing a command on the SPI bus

The AnaGate SPI can send arbitrary commands to the connected SPI bus. To write and read data by the PC only the `SPIDataReq` function is required.

```
Public Declare Function SPIDataReq Lib "AnaGateSPIVB6" Alias "_SPIDataReq@20" (ByVal hHandle As Long, _
                                       ByVal lpBufferWrite As Any, _
                                       ByVal nBufferWriteLen As Long, _
                                       ByVal lpBufferRead As Any, _
                                       ByVal nBufferReadlLen As Long) As Long
```

The event procedure `btnStart_Click` is called on click of the ***Execute command*** button.

```
Private Sub btnStart_Click()
    Dim nRC As Long, sText As String, I As Integer, sByteText As String
    Dim nBaudrate As Long, nSigLevel As Byte, nAuxVoltage As Byte, nClockMode As Byte
    Dim nBufferWriteLen As Long, nBufferReadLen As Long
    Dim arrWrite(1 To 255) As Byte, arrRead(1 To 255) As Byte

    nRC = SPIOpenDevice(hHandle, Me.IPAddresse.Text, 2000)
    If nRC <> 0 Then
        sText = "Fehler bei SPIOpenDevice: " & GetErrorMsg(nRC)
    Else
       nBaudrate = CLng(Me.txtBaudrate)
       nSigLevel = CLng(Me.cmbSigLevel.ListIndex)
       nAuxVoltage = CLng(Me.cmbAuxVoltage.ListIndex)
       nClockMode = CLng(Me.cmbClockMode.ListIndex)
       nRC = SPISetGlobals(hHandle, nBaudrate, nSigLevel, nAuxVoltage, nClockMode) 1

       If nRC <> 0 Then
             sText = sText & "Fehler bei SPISetGlobals: " & GetErrorMsg(nRC) & vbCrLf
       End If
       Me.lblDeviceInfo.Caption = GetAnagateInfo(hHandle)

       nBufferWriteLen = GetCommand(arrWrite)  2
       nBufferReadLen = nBufferWriteLen

       nRC = SPIDataReq(hHandle, VarPtr(arrWrite(1)), nBufferWriteLen, _
                                 VarPtr(arrRead(1)), nBufferReadLen)      3
       If nRC = 0 Then
           For I = 1 To nBufferReadLen
              sByteText = sByteText & "0x" & ToHex(arrRead(I)) & " "
           Next I
           Me.txtBufferRead = sByteText
           sText = sText & "SPIDataReq OK: " & vbCrLf
       Else
           sText = sText & "Fehler bei SPIDataReq: " & GetErrorMsg(nRC) & vbCrLf
       End If

       nRC = SPICloseDevice(hHandle)
    End If

End Sub
```

1. A call to `SPISetGlobals` sets the global settings on the device, the parameter values of the input fields in the dialog form are used.
2. The `GetCommand` function converts the texual SPI command entered in the input field on the form to a byte array structure.
3. To process the data in the read and receive buffers, a byte array is used as VB6 data type for both buffers. For this to work, the real memory address of the array data has to be parsed to the DLL function. This will be done by using the `VarPtr` function on the first byte array element.

 © 2007-2010 Analytica GmbH

.

7.3. Programming language VB.NET

Of course, it is also possible to use the functions of the AnaGate API with the .NET programming languages. For these languages the functions have only to be declared correctly in one of the .NET languages. Loading and unloading of the declared API functions is done automatically by the .NET framework.

7.3.1. CAN Console application VB.NET

This programming example for VB.NET demonstrates how to connect to an AnaGate CAN and how to process received CAN data via a callback function.

```
Declare Function CANOpenDevice Lib "AnaGateCAN" (ByRef Handle As Int32, _
                                                 ByVal ConfirmData As Int32, _
                                                 ByVal MonitorOn As Int32, _
                                                 ByVal PortNumber As Int32, _
                                                 ByVal TCPAddress As String, _
                                                 ByVal Timeout As Int32) As Int32
Declare Function CANCloseDevice Lib "AnaGateCAN" (ByVal Handle As Int32) As Int32
Public Delegate Sub CAN_CALLBACK(ByVal ID As Int32, ByVal Buffer As IntPtr, _
                                 ByVal Bufferlen As Int32, ByVal Flags as Int32, _
                                 ByVal Handle as Int32)
Declare Function CANSetCallback Lib "AnaGateCAN" (ByVal Handle As Int32, _
                                                  ByVal MyCB As CAN_CALLBACK) As Int32

Sub CANCallback( ByVal ID As Int32, ByVal Buffer As IntPtr, ByVal Bufferlen As Int32, _
                 ByVal Flags as Int32, ByVal Handle as Int32)
  Dim Bytes as Byte(8)

  System.Runtime.InteropServices.Marshal.Copy(Buffer, Bytes, 0, Bufferlen )
  Console.Out.Write( "CAN-ID=" )
  Console.Out.Write( ID )
  Console.Out.Write( ",Data=" )
  For I As Integer = 0 To BufferLen - 1
    Console.Out.Write( Bytes(I) )
  Next

End Sub

Function Main(ByVal CmdArgs() As String) As Integer

  'Opens the single CAN port of a AnaGate CAN
  dim RC as Int32 = CANOpenDevice(Handle, 0, 1, 400, 0, "192.168.2.1", 1000)
  If RC = 0 Then
    CANSetCallback( Handle, AddressOf CANCallback )
  end If
  If RC = 0 Then
    CANCloseDevice( Handle )
  end if
 end Function
```

7.3.2. SPI Console application VB.NET

This programming example for VB.NET demonstrates how to connect to an AnaGate SPI and how to execute SPI commands.

Note

The source code of the example can be found in directory `Samples/SPI-VB.NET` on the CD.

 © 2007-2010 Analytica GmbH

```
Sub Main()

      Dim hHandle As Int32, nIndex As Integer
      Dim BufferWrite(100) As Byte, BufferRead(100) As Byte
      Dim nBaudrate As Int32 = 5000000 ' 500kBit
      Dim nSigLevel As Byte = 2   ' +3.3V for the signals.
      Dim nAuxVoltage As Byte = 0 ' support voltage is +3.3V.
      Dim nClockMode As Byte = 3  ' CPHA=1 and CPOL=1.

      Dim nRC = SPIOpenDevice(hHandle, "192.168.1.254", 5000) 1
      If nRC <> 0 Then
         Console.WriteLine("Error SPIOpenDevice: " & GetErrorMsg(nRC) & vbCrLf)
      Else
         nRC = SPISetGlobals(hHandle, nBaudrate, nSigLevel, nAuxVoltage, nClockMode) 2
         nRC = SPIGetGlobals(hHandle, nBaudrate, nSigLevel, nAuxVoltage, nClockMode)

         For nIndex = 0 To 100  ' init buffers with some data
            BufferWrite(nIndex) = 69
            BufferRead(nIndex) = 96
         Next nIndex

         BufferWrite(0) = 5 * 16 ' 0x50 = READ STATUS (M25P80)
         BufferWrite(1) = 5 * 16 ' 0x50 = READ STATUS (M25P80)

         nRC = SPIDataReq(hHandle, BufferWrite, 2, BufferRead, 2) 3
         If nRC <> 0 Then
            Console.WriteLine("Error SPIDatReg: " & GetErrorMsg(nRC) & vbCrLf)
         Else
            Console.Write("Result: DATAREQ")
            For nIndex = 0 To 1  ' init buffers with some data
               Console.Write(BufferRead(nIndex) & " ")
            Next
            Console.WriteLine()
         End If

         SPICloseDevice(hHandle) 4
      End If
   End Sub
```

1. A call to `SPIOpenDevice` establishes a network connection to the device. If the function fails, a textual error description is returned via Funktion `GetErrorMsg`.
2. `SPISetGlobals` sets the global parameters of the device (baud rate, signal level, voltage level of the support voltage, clock mode).
3. Via the `SPIDataReq` function data is written to the SPI bus. If the command is successful, the data read from the SPI partner is returned in the receive buffer.
4. The connection to the device is closed with the `SPICloseDevice` function.

The functions of the programming API are defined in a wrapper module. In the following you can see a part of the wrapper module, which includes the declarations of all API functions.

```
Imports System.Runtime.InteropServices

Namespace Analytica.AnaGate
    Public Module AnaGateAPI

       Declare Function SPIOpenDevice Lib "AnaGateSPI" (ByRef Handle As Int32,  _
                                                      ByVal TCPAddress As String, _
                                                      ByVal Timeout As Int32) As Int32

       Declare Function SPICloseDevice Lib "AnaGateSPI" (ByVal Handle As Int32) As Int32

       Declare Function SPISetGlobals Lib "AnaGateSPI" (ByVal Handle As Int32, _
                                                       ByVal Baudrate As Int32, _
                                                       ByVal SigLevel As Byte, _
                                                       ByVal AuxVoltage As byte, _
                                                       ByVal ClockMode As byte) As Int32

       Declare Function SPIGetGlobals Lib "AnaGateSPI" (ByVal Handle As Int32, _
                                                       ByRef Baudrate As Int32, _
                                                       ByRef SigLevel As Byte, _
```

 © 2007-2010 Analytica GmbH

```
                                                        ByRef AuxVoltage As Byte, _
                                                        ByRef ClockMode As Byte) As Int32

       Declare Function SPIDataReq Lib "AnaGateSPI" (ByVal Handle As Int32, _
                                                    <MarshalAs(UnmanagedType.LPArray)> ByVal BufferWrite() As Byte, _
                                                    ByVal BufferWriteLen As Int32, _
                                                    <MarshalAs(UnmanagedType.LPArray)> ByVal BufferRead() As Byte, _
                                                    ByVal BufferReadLen As Int32) As Int32

       Declare Function SPIErrorMessage Lib "AnaGateSPI" (ByVal RC As Int32, _
                                                          ByVal Buffer As IntPtr, _
                                                          ByVal Bufferlen As Int32) As Int32
    End Module
End Namespace
```


© 2007-2010 Analytica GmbH

Part II. Scripting language LUA

Table of Contents

8. The LUA scripting interface of the *AnaGate* product line ... 75
  8.1. Creating scripts ... 76
  8.2. Running scripts on personal computer ... 76
  8.3. Running scripts on *AnaGate* hardware ... 77
9. Common function reference ... 80
  LS_DeviceInfo ... 81
  LS_GetTime ... 82
  LS_Sleep ... 83
10. CAN Reference ... 84
  LS_CANOpenDevice ... 85
  LS_CANCloseDevice ... 87
  LS_CANRestartDevice ... 88
  LS_CANSetGlobals ... 89
  LS_CANGetGlobals ... 91
  LS_CANWrite ... 93
  LS_CANWriteEx ... 95
  LS_CANSetCallback ... 97
  LS_CANGetMessage ... 99
  LS_CANSetFilter ... 101
  LS_CANGetFilter ... 102
  LS_CANSetTime ... 103
  LS_CANErrorMessage ... 104
  LS_CANReadDigital ... 105
  LS_CANWriteDigital ... 106
11. SPI Reference ... 107
  LS_SPIOpenDevice ... 108
  LS_SPICloseDevice ... 109
  LS_SPISetGlobals ... 110
  LS_SPIGetGlobals ... 112
  LS_SPIDataReq ... 114
  LS_SPIErrorMessage ... 116
  LS_SPIReadDigital ... 117
  LS_SPIWriteDigital ... 118
12. I2C Reference ... 119
  LS_I2COpenDevice ... 120
  LS_I2CCloseDevice ... 122
  LS_I2CReset ... 123
  LS_I2CRead ... 124
  LS_I2CWrite ... 125
  LS_I2CReadDigital ... 126
  LS_I2CWriteDigital ... 127
  LS_I2CErrorMessage ... 128
  LS_I2CReadEEPROM ... 129
  LS_I2CWriteEEPROM ... 131
13. CANOpen functions ... 133
  LS_CANopenSetConfig ... 134
  LS_CANopenGetConfig ... 135
  LS_CANopenSetSYNCMode ... 136
  LS_CANopenSetCallbacks ... 137
  LS_CANopenGetPDO ... 138
  LS_CANopenGetSYNC ... 139

 © 2007-2010 Analytica GmbH

LS_CANopenGetEMCY 140
LS_CANopenGetGUARD 141
LS_CANopenGetUndefined 142
LS_CANopenSendNMT 143
LS_CANopenSendSYNC 144
LS_CANopenSendTIME 145
LS_CANopenSendPDO 146
LS_CANopenSendSDORead 147
LS_CANopenSendSDOWrite 148
LS_CANopenSendSDOReadBlock 149
LS_CANopenSendSDOWriteBlock 150
Programmer example 151
14. LUA programming examples 153
14.1. Examples for devices with CAN interface 153
14.2. Examples for devices with SPI interface 154
14.3. Examples for devices with I2C interface 155

 © 2007-2010 Analytica GmbH

Chapter 8. The LUA scripting interface of the *AnaGate* product line

> **LUA** [http://www.lua.org] is a lightweight multi-paradigm programming language designed as a scripting language with extensible semantics as a primary goal. The name comes from the Portuguese word lua meaning "moon". *LUA* was created in 1993 by Roberto Ierusalimschy, Luiz Henrique de Figueiredo, and Waldemar Celes, members of the *Computer Graphics Technology Group (Tecgraf)* at the Pontifical Catholic University of Rio de Janeiro, in Brazil. ...
>
> ... In general, *LUA* strives to provide flexible meta-features that can be extended as needed, rather than supply a feature-set specific to one programming paradigm. As a result, the base language is light - in fact, the full reference interpreter is only about 150 kB compiled - and easily adaptable to a broad range of applications.
>
> —Wikipedia, *LUA*

In order to be able to solve simple programming problems concerning the *AnaGate* devices with the scripting language *LUA*, the *LUA* interpreter is extended by several functions to operate the different *AnaGate* devices. These additional functions are described in detail in the following chapters and are closely related to the functions of the *AnaGate API* libraries.

Source files for *LUA* (called scripts) are created and edited on a personal computer (Windows or Linux) in a standard text editor. Then, the script is simply executed in the command shell via a free *LUA* interpreter. The full standard functionality of the *LUA* language can be used as well as additional functional extensions for access of the *AnaGate* hardware.

The scripting language *LUA* is very well-suited for use on *embedded systems* because of its good performance and small runtime size. For this reason, the *LUA* interpreter is integrated in the firmware of the *AnaGate* hardware [1]. So, it is possible to execute scripts not only on the personal computer, but also on the *AnaGate* device itself.

Note

Please refer to the printed paperbacks *LUA Reference Manual* ([LuaRef2006-EN]) and *Programming in Lua* ([LuaProg2006-EN]) for detailed information about LUA. The reference manual is also available online at Lua.org [http://www.lua.org].

[1]only AnaGate CAN uno, AnaGate CAN duo, AnaGate CAN quattro, AnaGate CAN USB and AnaGate Universal Programmer

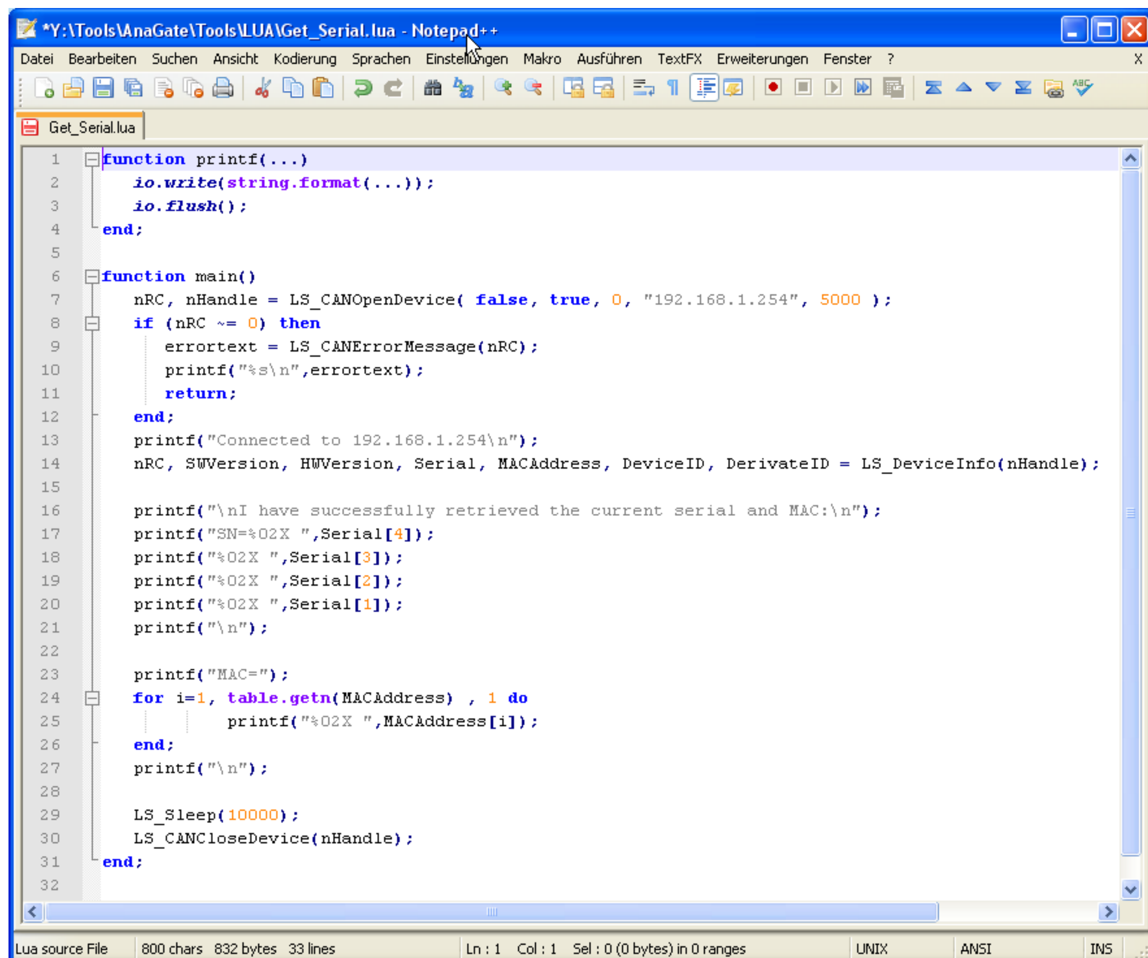
 © 2007-2010 Analytica GmbH

8.1. Creating scripts

Creating and editing of script files for the scripting language *LUA* is exceptionally easy, because a standard text editor is sufficient to do that. On Windows operating systems Notepad or Wordpad can be used for example, on linux systems vi or other text tools.

In the meantime some text editors, partly free of charge, support syntax-highlighting for *LUA*, which makes it really easier for a programmer to develop.

Figure 8.1. Edit LUA script in a text editor



When coding of a script is finished, it can be executed and tested on a personal computer as decribed below.

8.2. Running scripts on personal computer

To execute *LUA* script files on a personal computer, an actual program version of the *LUA* interpreter must be available.

On the CD-ROM, which is included in the scope of delivery, a modified *LUA* interpreter can be found in the directory `LUA`. This interpreter consists of a single executable

© 2007-2010 Analytica GmbH

named `LUA.exe`, which includes all functional extensions to operate the *AnaGate* hardware.

Tip

The lastest version of `LUA.exe` can be downloaded free of charge via the support pages of the product homepage [http://www.anagate.de/support/download.htm].

Except of the program executable `LUA.exe` no other program files are needed, so that there is only one single file to copy to the computer harddisk (or file server, SUB stick, ..).

A script file is executed easily via the command line shell, only the name of the scriptfile has to specified to start it.

Following example shows how a script file named `Get_Serial.lua` is executed in the windows command shell.

```
T:\Tools\LUA>LUA.exe Get_Serial.lua 1
Connected to 192.168.1.254  2
I have successfully retrieved the current serial and MAC:
SN=01 02 02 1D
MAC=00 50 C2 3C B2 1D
T:\Tools\LUA>
```

1. The filename of the script to execute has to be supplied as parameter on start of the interpreter.
2. The serial number and MAC address of a device at IP address `192.168.1.254` is retrieved and written to the standard output.

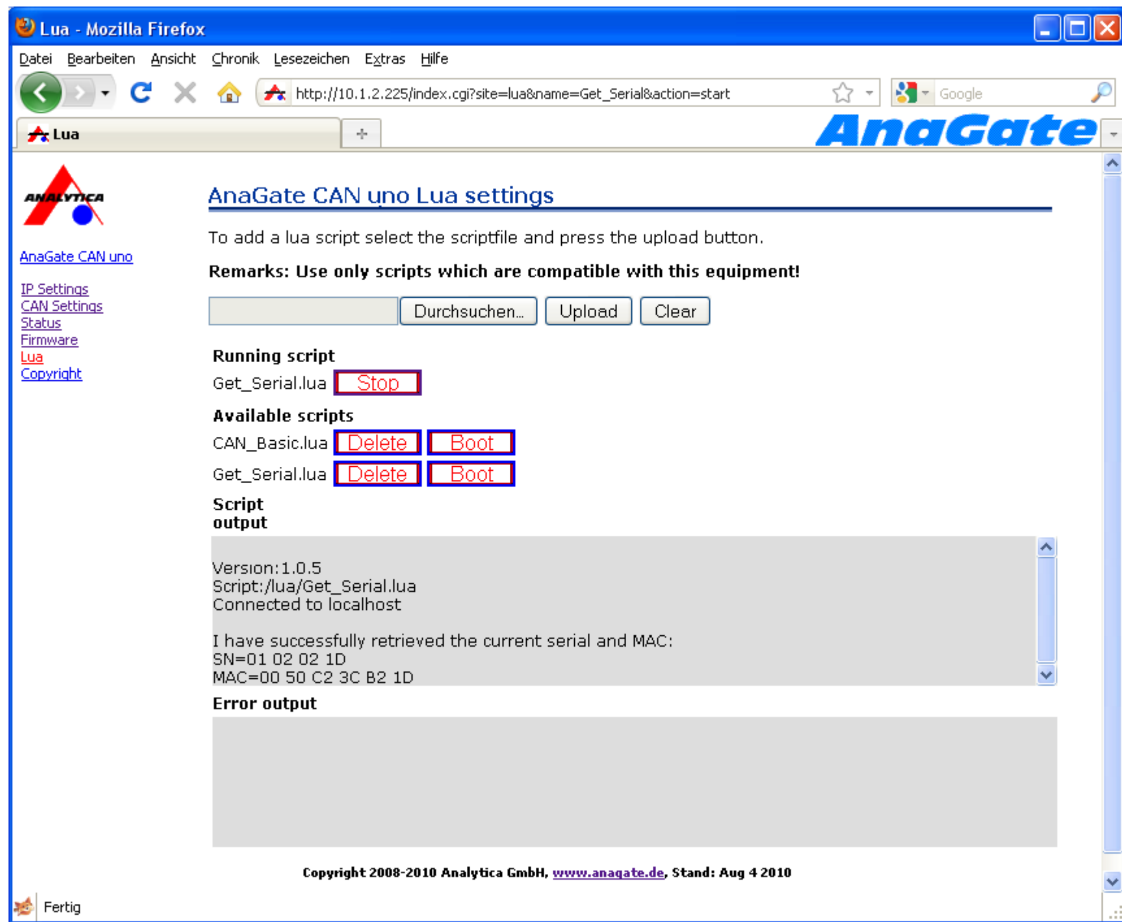
8.3. Running scripts on *AnaGate* hardware

Like already mentioned before, it is possible to execute self-created application scripts with an installed LUA script interpreter directly on the *AnaGate* hardware.

Via the HTTP interface of each device LUA script files can be downloaded to the device and executed locally. In the following you can see a screenshot of the *LUA* configuration page of a *AnaGate CAN uno*.

 © 2007-2010 Analytica GmbH

Figure 8.2. HTTP interface, LUA settings



Browse... Opens a file upload dialog to select a LUA script file.

Upload Uploads the selected script file to the device.

Clear Clears the current script file selection.

Boot script — Script file executed on system startup. Via the button **Delete** the boot script can be deactivated. Only one boot script is allowed.

Running script — Displays the currently executing script file. Via the button **Stop** the execution can be cancelled.

Available scripts — Displays all scripts which are currently available on the device.

To start the execution of a script click on the button **Start**. Via button **Delete** a script can be deleted on the device and via **Boot** a script can be defined as boot script.

script output area — In this text area the standard output (stdout) of the currently executing script is displayed. Via the button **Clear** this text area can be cleared.

 © 2007-2010 Analytica GmbH

error output area	In this text area the standard error output (stderr) of the currently executing script is displayed. Via the button **Clear** this text area can be cleared.

Tip

The text areas for script and error output are not refreshed automatically. A manual page reload of the current page refreshes both text areas.

© 2007-2010 Analytica GmbH

Chapter 9. Common function reference

© 2007-2010 Analytica GmbH

LS_DeviceInfo

LS_DeviceInfo — Retrieves some global information from the AnaGate hardware.

Syntax

```
int RC, int nSWVersion, int nHWVersion, table(4) tabSerial, table(6)
tabMACAddress, int nDeviceID, int nSWDerivateID = LS_DeviceInfo(int
hHandle);
```

Parameter

hHandle Valid access handle returned by call to `LS_CANOpenDevice`, `LS_I2COpenDevice` oder `LS_SPIOpenDevice`.

Return values

RC
: Returns `0` if successful, or an error value otherwise (Appendix A, *API return codes*).

nSWVersion
: Firmware version. The version number consists of 3 numbers (major.minor.revision), which are stored in a 4-byte integer value.

nHWVersion
: Hardware version. The version number consists of 3 numbers (major.minor.revision), which are stored in a 4-byte integer value.

tabSerial
: Serial number of the AnaGate hardware (4 byte).

tabMACAddress
: MAC address of the AnaGate hardware (6 byte).

nDeviceID
: Device specific identifier. Specifies the device type of the hardware.

- 1 = AnaGate I2C
- 2 = AnaGate CAN
- 3 = AnaGate SPI
- 8 = AnaGate Universal Programmer
- 9 = AnaGate Renesas

nSWDerivateID
: Indicates a customer-specific firmware version, if not `0x00`.

Description

Returns specific information about a device of the AnaGate product line.

See also

LS_CANOpenDevice, LS_SPIOpenDevice, LS_I2COpenDevice

© 2007-2010 Analytica GmbH

LS_GetTime

LS_GetTime — Returns the current system time.

Syntax

```
int RC, table(2) tabTime = LS_GetTime(void);
```

Parameter

This function does not have any parameters.

Return values

RC	Returns `0` if successful, or an error value otherwise (Table A.5, "Return values for LUA scripting").
tabTime(1), tabTime(2)	*tabTime(1)* specifies the number of seconds elapsed since 01.01.1970, in *tabTime(2)* the fractions of a second is returned in milliseconds..

Description

Returns the system time as the number of elapsed seconds and milliseconds since midnight of January 1, 1970.

© 2007-2010 Analytica GmbH

LS_Sleep

LS_Sleep — Suspends the execution until the time-out interval elapses.

Syntax

```
int RC = LS_Sleep(unsigned int nMilliseconds);
```

Parameter

nMilliseconds — The time interval for which execution is to be suspended, in milliseconds.

Return value

RC — Returns `0` if successful, or an error value otherwise (Table A.5, “Return values for LUA scripting”).

Description

Suspends the execution until the time-out interval in milliseconds elapses.

© 2007-2010 Analytica GmbH

Chapter 10. CAN Reference

The CAN API can be used with all CAN gateway models of the AnaGate series. The programming interface is identical for all devices and uses the network protocol TCP or UDP in general.

Following devices can be addresse via the CAN API interface:

- AnaGate CAN
- AnaGate CAN uno
- AnaGate CAN duo
- AnaGate CAN quattro
- AnaGate CAN USB

Note

All CAN specific functionality of the AnaGate C-API is also available für LUA users, the LUA function extensions are documented in the following.

© 2007-2010 Analytica GmbH

LS_CANOpenDevice

LS_CANOpenDevice — Opens an network connection (TCP) to an AnaGate CAN device.

Syntax

```
int RC, int Handle = LS_CANOpenDevice(boolbSendDataConfirm, bool bSendDataInd, int nCANPort, string sIPAddress, int nTimeout );
```

Parameter

bSendDataConfirm — It set to TRUE, all incoming and outgoing Data requests are confirmed by the internal message protocol. Without confirmations a better transmittion performance is reached.

bSendDataInd — If set to FALSE, all incoming telegrams are discarded.

nCANPort — CAN port number. Allowed values are:

- `0` for port A (Modells AnaGate CAN uno, AnaGate CAN duo, AnaGate CAN quattro, AnaGate CAN USB and AnaGate CAN)
- `1` for port B (AnaGate CAN duo, AnaGate CAN quattro)
- `2` for port C (AnaGate CAN quattro)
- `3` for port D (AnaGate CAN quattro)

sIPAddress — Network address of the AnaGate partner.

nTimeout — Default timeout for accessing the AnaGate in milliseconds.

A timeout is reported if the AnaGate partner does not respond within the defined timeout period. This global timeout value is valid on the current network connection for all commands and functions which do not offer a specific timeout value.

Return value

RC — Returns `0` if successful, or an error value otherwise (Appendix A, *API return codes*).

Handle — Access handle if successfully connected to the AnaGate device.

Description

Opens a TCP/IP connection to an CAN interface of a AnaGate CAN device. If the connection is established, CAN telegrams can be sent and received.

The connection should be closed with the function `CANCloseDevice` if not longer needed.

© 2007-2010 Analytica GmbH

Important

It is recommended to close the connection, because all internally allocated system resources are freed again and the connected AnaGate device is signalled that the active connection is not longer in use and can be used again for new connect requests.

See the following example for the initial programming steps.

```
-- open: use no confirmations and receive incomping CAN data
nRC, hHandle = LS_CANOpenDevice( false, true, 0, "192.168.0.254", 5000);
if ( nRC == 0 ) then
   -- now do something
   LS_CANCloseDevice(hHandle);
end;
```

See also

LS_CANCloseDevice

LS_CANRestartDevice

 © 2007-2010 Analytica GmbH

LS_CANCloseDevice

LS_CANCloseDevice — Closes an open network connection to an AnaGate CAN device.

Syntax

```
int RC = LS_CANCloseDevice(int hHandle);
```

Parameter

hHandle   Valid access handle.

Return value

RC   Returns `0` if successful, or an error value otherwise (Appendix A, *API return codes*).

Description

Closes an open network connection to an AnaGate CAN device. The *`hHandle`* parameter is a return value of a succesfull call to the function `LS_CANOpenDevice`.

Important

It is recommended to close the connection, because all internally allocated system resources are freed again and the connected AnaGate device is signalled that the active connection is not longer in use and can be used again for new connect requests.

See also

LS_CANOpenDevice

© 2007-2010 Analytica GmbH

LS_CANRestartDevice

LS_CANRestartDevice — Restarts a AnaGate CAN device.

Syntax

```
int RC = LS_CANRestartDevice(string sIPAddress, int nTimeout );
```

Parameter

sIPAddress Network address of the AnaGate partner.

nTimeout Default timeout for accessing the AnaGate in milliseconds. A timeout is reported if the AnaGate partner does not respond within the defined timeout period.

Return value

RC Returns `0` if successful, or an error value otherwise (Appendix A, *API return codes*).

Description

Restarts the AnaGate CAN device at the specified network address. It disconnects implicitly all open network connections to all existing CAN interfaces. The Restart command is even possible if the maximum number of allowed connections is reached.

Important

It is recommended to use this command only in emergency cases, if there is a need to connect even if the maximum number of concurrent connections is reached.

See also

LS_CANOpenDevice

 © 2007-2010 Analytica GmbH

LS_CANSetGlobals

LS_CANSetGlobals — Sets the global settings, which are to be used on the CAN bus

Syntax

```
int RC = LS_CANSetGlobals(int hHandle, int nBaudrate, int nOperatingMode, bool bTermination, bool bHighSpeedMode, bool bTimeStampOn);
```

Parameter

hHandle
: Valid access handle.

nBaudrate
: The baud rate to be used. Following values are allowed:

- 10.000 für 10kBit
- 20.000 für 20kBit
- 50.000 für 50kBit
- 62.500 für 62,5kBit
- 100.000 für 100kBit
- 125.000 für 125kBit
- 250.000 für 250kBit
- 500.000 für 500kBit
- 800.000 für 800kBit (not AnaGate CAN)
- 1.000.000 für 1MBit

nOperatingMode
: The operating mode to be used. Following values are allowed.

- 0 = default mode.
- 1 = loop back mode: All telegrams sent by a connected partner is routed back to all active connected partners.
- 2 = listen mode: Device operates as passive bus partner, this means no telegrams are sent to the CAN bus (no ACKs for incoming telegrams too).

bTermination
: Use integrated CAN bus termination (`true`= yes, `false` = no). This setting is not supported by all AnaGate CAN models.

bHighSpeedMode
: Use high speed mode (`true`= yes, `false`= no). This setting is not supported by all AnaGate CAN models.

The high speed mode was created for large baud rates with continuously high bus load. In this mode telegrams are not

© 2007-2010 Analytica GmbH

confirmed on procol layer and the software filters defined via `LS_CANSetFilter` are ignored.

bTimeStampOn — Use time stamp mode (`true`= yes, `false`= no). This setting is not supported by all AnaGate CAN models.

In activated time stamp mode an additional timestamp is sent with the CAN telegram. This timestamp indicates when the incomming message is received by the CAN controller or when the outgoing message is confirmed by the CAN controller.

Return value

RC Returns `0` if successful, or an error value otherwise (Appendix A, *API return codes*).

Description

Sets the global settings of the used CAN interface. These settings are effective for all concurrent connections to the CAN interface. The settings are not saved permantently on the device and are reset every device restart.

Remarks

The settings of the integrated CAN bus termination, the high speed mode and the time stamp are not supported by the AnaGate CAN (hardware version 1.1.A). These settings are ignored by the device.

See also

LS_CANGetGlobals

© 2007-2010 Analytica GmbH

LS_CANGetGlobals

LS_CANGetGlobals — Returns the currently used global settings on the CAN bus.

Syntax

```
int RC, int nBaudrate, int nOperatingMode, bool bTermination, bool
bHighSpeedMode, bool bTimeStampOn = LS_CANGetGlobals(int hHandle);
```

Parameter

hHandle Valid access handle.

Return values

RC	Returns `0` if successful, or an error value otherwise (Appendix A, *API return codes*).
nBaudrate	The baud rate currently used on the CAN bus.
nOperatingMode	The operating mode to be used. Following values are returned. • 0 = default mode. • 1 = loop back mode: All telegrams sent by a connected partner is routed back to all active connected partners. • 2 = listen mode: Device operates as passive bus partner, this means no telegrams are sent to the CAN bus (no ACKs for incoming telegrams too).
bTermination	Is the integrated CAN bus termination used? `true`= yes, `false`= no). This setting is not supported by all AnaGate CAn modells.
bHighSpeedMode	Is the high speed mode switched on? (`true`= yes, `false`= no). This setting is not supported by all AnaGate CAN modells. The high speed mode was created for large baud rates with continuously high bus load. In this mode telegrams are not confirmed on procol layer and the software filters defined via `LS_CANSetFilter` are ignored.

Description

Returns the global settings of the used CAN interface. These settings are effective for all concurrent connections to the CAN interface.

Remarks

The settings of the integrated CAN bus termination, the high speed mode and the time stamp are not supported by the AnaGate CAN (hardware version 1.1.A). These settings are ignored by the device.

© 2007-2010 Analytica GmbH

See also

LS_CANSetGlobals

© 2007-2010 Analytica GmbH

LS_CANWrite

CANWriteEx — Send a CAN telegram to the CAN bus via the AnaGate device.

Syntax

```
RC = CANWrite(int hHandle, int nCANId, int nDataLen, table (nDataLen)
tabData, int nFlags);
```

Parameter

hHandle — Valid access handle.

nCANId — CAN identifier of the sender. Parameter *nFlags* defines, if the address is in extended format (29-bit) or standard format (11-bit).

nDataLen — Length of data buffer (max. 8 bytes).

tabData — Data buffer with telegram data.

nFlags — The format flags are defined as follows.

- Bit 0: If set, the CAN identifier is in extended format (29 bit), otherwise not (11 bit).
- Bit 1: If set, the telegram is marked as remote frame.

Return value

RC Returns `0` if successful, or an error value otherwise (Appendix A, *API return codes*).

Description

Both functions sends a CAN telegram to the CAN bus via the AnaGate device like the `LS_CANWriteEx` function.

The `LS_CANWriteEx` additionally returns a timestamp of the time at which the telegram is sent.

Note

With remote frames (RTR = remote transmission request) a destination node can request data from a source node. The data length is to be set to the number of requested bytes - on the CAN bus no data is sent only the data size information.

When using the `LS_CANWrite` bzw. `LS_CANWriteEx` functions to send remote frames the data buffer and the buffer size equal to the number of requested bytes have to be set correctly.

See the following example for sending a data telegram to the connected CAN bus.

```
tabData = {};
```

 © 2007-2010 Analytica GmbH

```
for i=1, 8 , 1 do
   table.insert(tabData, i );
end;

nFlags = 0x0; //  11bit address + standard (not remote frame)
nCANId = 0x25; // send with CAN ID 0x25;

nRC, hHandle = LS_CANOpenDevice(, true, true, 0, "192.168.0.254", 5000);
if ( nRC == 0 ) then
   // send 8 bytes with CAN id 37
   nRC = LS_CANWrite( hHandle, nCANId, 8, tabData, nFlags );

   // send a remote frame to CAN id 37 (request 4 data bytes)
   nRC = LS_CANWrite( hHandle, nCANId, 4, tabData, 0x02 );

   LS_CANCloseDevice(hHandle);
end;
```

Remarks

For devices of type AnaGate CAN (hardware version 1.1.A) the function `CANWriteEx` is equal to `CANWrite`, the return values *nSeconds* and *pnMicroseconds* will remain unchanged.

See also

LS_CANWriteEx

© 2007-2010 Analytica GmbH

LS_CANWriteEx

CANWriteEx — Send a CAN telegram to the CAN bus via the AnaGate device.

Syntax

```
RC, int nSeconds, int nMicroseconds = CANWriteEx(int hHandle, int nCANId,
int nDataLen, table (nDataLen) tabData, int nFlags);
```

Parameter

hHandle — Valid access handle.

nCANId — CAN identifier of the sender. Parameter *nFlags* defines, if the address is in extended format (29-bit) or standard format (11-bit).

nDataLen — Length of data buffer (max. 8 bytes).

tabData — Data buffer with telegram data.

nFlags — The format flags are defined as follows.

- Bit 0: If set, the CAN identifier is in extended format (29 bit), otherwise not (11 bit).
- Bit 1: If set, the telegram is marked as remote frame.

Return value

RC — Returns `0` if successful, or an error value otherwise (Appendix A, *API return codes*).

nSeconds — Timestamp of the confirmation of the CAN controller (seconds from 01.01.1970).

nMicroseconds — Micro seconds portion of the timestamp.

Description

Both functions sends a CAN telegram to the CAN bus via the AnaGate device like the `LS_CANWrite` function.

The `LS_CANWriteEx` additionally returns a timestamp of the time at which the telegram is sent.

Note

With remote frames (RTR = remote transmission request) a destination node can request data from a source node. The data length is to be set to the number of requested bytes - on the CAN bus no data is sent only the data size information.

When using the `LS_CANWrite` bzw. `LS_CANWriteEx` functions to send remote frames the data buffer and the buffer size equal to the number of requested bytes have to be set correctly.

 © 2007-2010 Analytica GmbH

See the following example for sending a data telegram to the connected CAN bus.

```
tabData = {};
for i=1, 8 , 1 do
   table.insert(tabData, i );
end;

nFlags = 0x0; //  11bit address + standard (not remote frame)
nCANId = 0x25; // send with CAN ID 0x25;

nRC, hHandle = LS_CANOpenDevice(, true, true, 0, "192.168.0.254", 5000);
if ( nRC == 0 ) then
   // send 8 bytes with CAN id 37
   nRC, nSeconds, nMicroSeconds = LS_CANWriteEx( hHandle, nCANId, 8, tabData, nFlags );

   // send a remote frame to CAN id 37 (request 4 data bytes)
   nRC, nSeconds, nMicroSeconds = LS_CANWriteEx( hHandle, nCANId, 4, tabData, 0x02 );

   LS_CANCloseDevice(hHandle);
end;
```

Remarks

For devices of type AnaGate CAN (hardware version 1.1.A) the function `CANWriteEx` is equal to `CANWrite`, the return values *`nSeconds`* and *`pnMicroseconds`* will remain unchanged.

See also

LS_CANWrite

 © 2007-2010 Analytica GmbH

LS_CANSetCallback

LS_CANSetCallback — Defines an asynchronous callback function, which is called for each incoming CAN telegram.

Syntax

```
int RC = LS_CANSetCallback(int hHandle, string sCallbackFunction);
```

```
function MY_LS_CALLBACK(int nCANId, int nDataLen, table(nDataLen) tabData, int nFlags, int hHandle, int nSeconds, int nMicroseconds);
```

Parameter

hHandle — Valid access handle.

sCallbackFunction — Name of the private callback function. Set this parameter to `""` to deactivate the callback function. The parameters of the callback function are described in section *Callback-Parameter*.

Callback-Parameter

nCANId — CAN identifier of the sender. Parameter *nFlags* defines, if the address is in extended format (29-bit) or standard format (11-bit).

nDataLen — Length of data buffer (max. 8 bytes).

tabData — Data buffer with telegram data.

nFlags — The format flags are defined as follows.

- Bit 0: If set, the CAN identifier is in extended format (29 bit), otherwise not (11 bit).
- Bit 1: If set, the telegram is marked as remote frame.
- Bit 2: If set, the telegram has a valid timestamp.

hHandle — Valid access handle.

nSeconds — Timestamp of the confirmation of the CAN controller (seconds from 01.01.1970).

nMicroseconds — Micro seconds portion of the timestamp.

Return value

RC — Returns `0` if successful, or an error value otherwise (Appendix A, *API return codes*).

Description

Incoming CAN telegrams can be received via a callback function, which can be set by a simple API call. If a callback function is used, is will be called by the API

© 2007-2010 Analytica GmbH

asynchronous. In alternative to the callback function, incoming data telegrams can be retrieved with the `LS_CANGetMessage` function.

See the following example for using a callback.

```
function MyCallback(nID, nLen, tabData, nFlags, nHandle, nSecond, nMSecond)
   io.write(string.format("%.3X::%.3X,%d", nHandle, nID, nLen));
   for i=1, nLen, 1 do
      io.write(string.format(%.2X ", tabData[i]));
   end;
   io.write("\n");
   io.flush();
end;

function main()
    nRC, nHandle = LS_CANOpenDevice( false, true, 0, "127.0.0.1", 5000 );
    if (nRC ~= 0) then
        errortext = LS_CANErrorMessage(nRC);
        printf("%s\n",errortext);
        return;
    end;
    -- set globals: 100kBit, standard mode, termination off, no highspeed, no timestamp
    nRC = LS_CANSetGlobals( nHandle, 100000, 0, true, false, false );

    nRC = LS_CANSetCallback( nHandle, "MyCallback");

    while true do   -- forever
        LS_Sleep(500);
    end;

    nRC = LS_CANSetCallback( hHandle, "" );

    LS_CANCloseDevice(nHandle);
end;
```

See also

LS_CANWrite, LS_CANWriteEx, LS_CANGetMessage

 © 2007-2010 Analytica GmbH

LS_CANGetMessage

LS_CANGetMessage — .

Syntax

```
int nAvail, int nCANId, int nDataLen, table() tabData, int nFlags,
int nSeconds, int nMicroseconds = LS_CANGetMessage(int hHandle, int
nTimeout);
```

Parameter

hHandle — Valid access handle.

nTimeout — Maximum period of time in milliseconds to wait for the a new data telegram.

Return values

nAvail — Number of message which are left in the internal message puffer. If there is currently no message available `-10` (ERR_NO_DATA) is returned.

nCANId — CAN identifier of the sender. Parameter *nFlags* defines, if the address is in extended format (29-bit) or standard format (11-bit).

nDataLen — Length of data buffer.

tabData — Data buffer with telegram data.

nFlags — The format flags are defined as follows.

- Bit 0: If set, the CAN identifier is in extended format (29 bit), otherwise not (11 bit).
- Bit 1: If set, the telegram is marked as remote frame.
- Bit 2: If set, the telegram has a valid timestamp.

nSeconds — Timestamp of the confirmation of the CAN controller (seconds from 01.01.1970).

nMicroseconds — Micro seconds portion of the timestamp.

Description

This function reads a single CAN data telegram out of an internal message buffer. The message buffer is automatically filled with all incoming CAN data telegrams in a seperate thread.

The parameter *nTimeout* defines a maximum period of time, which the function should wait for a new data telegram, if there is currently no telgram in the internal buffer. If no new message is received within the time out, the function returns in *nAvail* the recturn code `-10` (ERR_NO_DATA).

 © 2007-2010 Analytica GmbH

Warning

If using a individual callback function (see `LS_CANSetCallback`), which is called if an incoming CAN data telegram is received, the internal message puffer is not filled. In this case is not possible to retrieve message via the `LS_CANGetMessage` function.

See the following example which handles incoming CAN data telegrams.

```
nRC, hHandle = LS_CANOpenDevice(true, true, 0, "192.168.0.254", 5000);
if ( nRC == 0 ) then
    -- set globals: 500Kbit, standard mode, termination on, no high speed, no timestamp
    nRC = LS_CANSetGlobals( hHandle, 500000, 0, true, false, false);
    nCurMsg = 0

    repeat
        nAvail, ID, Len, Data, Sec, Microsec = LS_CANGetMessage(hHandle, 100);
        if  nAvail>0 then
            nCurMsg = nCurMsg + 1;

            -- now do something with the incomming message data
            io.write( string.format(ID) );  -- for example, write out CAN id
        else
            LS_Sleep(25);  -- wait 25 ms if no message available
        end;
    until nCurMsg >= 100;  -- read only 100 messages, then stop

   LS_CANCloseDevice(hHandle);
end;
```

Remarks

For devices of type AnaGate CAN (hardware version 1.1.A) the return values *nSeconds* and *pnMicroseconds* are always set to zero.

See also

LS_CANSetCallback

 © 2007-2010 Analytica GmbH

LS_CANSetFilter

LS_CANSetFilter — Sets the current filter settings for the connection.

Syntax

```
int RC = LS_CANSetFilter(int hHandle, table(16)tabFilter);
```

Parameter

hHandle   Valid access handle.

tabFilter   Pointer to an array of 8 filter entries (4 mask and 4 range filter entries). A filter entry contains of two 32-bit values. Unused mask filter entries must be initialized with 0 values. Unused range filter entries must be initialized with a 0 for the start value and 0x1FFFFFFF for the end value.

Return value

RC   Returns `0` if successful, or an error value otherwise (Appendix A, *API return codes*).

Description

This function sets the current filter settings for the current connection. Filter can be used to suppress messages with specific CAN message ids.

A mask filter contains of a mask value, which defines the bits of the CAN identifier to examine, and the appropriate filter value. If the CAN identifier matches in the indicated filter mask with the filter value, the incoming CAN telegram is sent to the PC, otherwise not.

A range filter defines an address range with a appropriate start and end address. If the CAN identifier do not lie in the indicated filter range, the incoming CAN telegram is not sent to the PC.

Filter are only active, if the parameter *`bSendDataInd`* is set via the `LS_CANOpenDevice` function.

See also

LS_CANGetFilter

 © 2007-2010 Analytica GmbH

LS_CANGetFilter

LS_CANGetFilter — Returns the current filter settings for the connection.

Syntax

```
int RC, table(16) tabFilter = LS_CANGetFilter(int hHandle);
```

Parameter

hHandle Valid access handle.

Return value

RC	Returns 0 if successful, or an error value otherwise (Appendix A, *API return codes*).
tabFilter	Pointer to an array of 8 filter entries (4 mask and 4 range filter entries). A filter entry contains of two 32-bit values. Unused mask filter entries are initialized with 0 values. Unused range filter entries are initialized with (0,0x1FFFFFFF) value pairs.

Description

This function retrieves the current filter settings for the current connection. Filter can be used to suppress messages with specific CAN message ids.

See also

LS_CANSetFilter

© 2007-2010 Analytica GmbH

LS_CANSetTime

LS_CANSetTime — Sets the current system time on the AnaGate device.

Syntax

```
int RC = LS_CANSetTime(int hHandle, long nSeconds, long nMicroseconds);
```

Parameter

hHandle Valid access handle.

nSeconds Time in seconds from 01.01.1970.

nMicroseconds Micro seconds.

Return value

RC Returns `0` if successful, or an error value otherwise (Appendix A, *API return codes*).

Description

The `LS_CANSetTime` function sets the system time on the AnaGate hardware.

If the time stamp mode is switched on by the `LS_CANSetGlobals` function, the AnaGate hardware adds a time stamp to each incoming CAN telegram and a time stamp to the confirmation of a telegram sent via the API (only if confirmations are switched on for data requests).

Remarks

The setting of the base time for the time stamp mode is not supported by the AnaGate CAN (hardware version 1.1.A). This setting is ignored by the device.

 © 2007-2010 Analytica GmbH

LS_CANErrorMessage

LS_CANErrorMessage — Returns a description of the given error code as a text string.

Syntax

```
string sErrorMsg LS_CANErrorMessage(int nRetCode);
```

Parameter

nRetCode   Error code for which the error description is to be determined.

Return value

sErrorMsg   Textual description of the error code.

Description

Returns a textual description of the parsed error code (see Appendix A, *API return codes*).

See the following example in LUA scripting language.

```
nRC=0;
sErrorText = 'No Error';

//... call a API function here

sErrorText = LS_CANErrorMessage(nRC);
print(sErrorText);
```


© 2007-2010 Analytica GmbH

LS_CANReadDigital

LS_CANReadDigital — Reads the current values of digital input and output registers of the AnaGate device.

Syntax

`int RC, int nInputBits, int nOutputBits = **LS_CANReadDigital**(int *hHandle*);`

Parameter

hHandle Valid access handle.

Return value

RC Returns `0` if successful, or an error value otherwise (Appendix A, *API return codes*).

nInputBits Pointer to the current value of the digital input register. Currently only bits 0 to 3 are used, other bits are reserved for future use and are set to 0.

nOutputBits Pointer to the current value of the digital output register. Currently only bits 0 to 3 are used, other bits are reserved for future use and are set to 0.

Description

All models of the AnaGate series (except the model AnaGate CAN uno in DIN rail case) have connectors for 4 digtial inputs and 4 digital outputs at the rear panel.

The current values of the digital inputs and outputs can be retrieved with the `LS_CANReadDigital` function.

See the following example for setting an reading the digital IO.

```
nOutputs = 0x03;

nRC, hHandle = LS_CANOpenDevice( 400000, "192.168.0.254", 5000 );
if ( nRC == 0 ) then
   // set the digital output register (PIN 0 and PIN 1 to HIGH value)
   nRC = LS_CANWriteDigital( hHandle, nOutputs );

   // read all input and output registers
   nRC, nInputs, nOutputs = LS_CANReadDigital( hHandle );

   LS_CANCloseDevice(hHandle);
end;
```

See also

LS_CANWriteDigital

© 2007-2010 Analytica GmbH

LS_CANWriteDigital

LS_CANWriteDigital — Writes a new value to the digital output register of the AnaGate device.

Syntax

```
int RC = LS_CANWriteDigital(int hHandle, int nOutputBits);
```

Parameter

hHandle — Valid access handle.

nOutputBits — New register value. Currently only bits 0 to 3 are used, other bits are reserved for future use.

Return value

RC — Returns `0` if successful, or an error value otherwise (Appendix A, *API return codes*).

Description

All models of the AnaGate series (except the model AnaGate CAN uno in DIN rail case) have connectors for 4 digtial inputs and 4 digital outputs at the rear panel.

The digital outputs can be written with the `LS_CANWriteDigital` function.

A simple example for reading/writing of the IOs can be found at the description of `LS_SPIReadDigital`.

See also

LS_SPIReadDigital

© 2007-2010 Analytica GmbH

Chapter 11. SPI Reference

The Serial Peripheral Interface (SPI) is a synchroneous data link standard named by Motorola which operates in full duplex mode. The SPI gateway models of the AnaGate series provides access to a SPI bus via a standard networking.

With the SPI API these SPI gateways can be easily controlled. The programming interface is identical for all devices and used the network protocol TCP in general.

Following devices can be addresse via the SPI API interface:

- AnaGate SPI
- AnaGate Universal Programmer

Note

All SPI specific functionality of the AnaGate C-API is also available für LUA users, the LUA function extensions are documented in the following.

 © 2007-2010 Analytica GmbH

LS_SPIOpenDevice

LS_SPIOpenDevice — Opens a network connection to an AnaGate SPI device.

Syntax

```
int RC, int Handle = LS_SPIOpenDevice(string sIPAddress, int nTimeout);
```

Parameter

sIPAddress Network address of the AnaGate partner.

nTimeout Default timeout for accessing the AnaGate in milliseconds.

A timeout is reported if the AnaGate partner does not respond within the defined timeout period. This global timeout value is valid on the current network connection for all commands and functions which do not offer a specific timeout value.

Return values

RC Returns 0 if successful, or an error value otherwise (Appendix A, *API return codes*).

Handle Access handle if successfully connected to the AnaGate device.

Description

Opens a TCP/IP connection to an AnaGate SPI (resp. AnaGate Universal Programmer). After the connection is established, access to the SPI bus is possible.

Note

The AnaGate SPI (resp. the SPI interface of an AnaGate Universal Programmer) does not allow more than one concurrent network connection. During an established network connection all new connections are refused.

See the following example for the initial programming steps.

```
nRC, nHandle  = LS_SPIOpenDevice( "192.168.0.254", 5000 );
if (nRC ~= 0) then
   print(LS_SPIErrorMessage(nRC));
   exit();
end;

-- now do something

LS_SPICloseDevice(nHandle);
```

See also

LS_SPICloseDevice

 © 2007-2010 Analytica GmbH

LS_SPICloseDevice

LS_SPICloseDevice — Closes an open network connection to an AnaGate SPI device.

Syntax

```
int RC = LS_SPICloseDevice(int hHandle);
```

Parameter

hHandle Valid access handle.

Return value

RC Returns `0` if successful, or an error value otherwise (Appendix A, *API return codes*).

Description

Closes an open network connection to an AnaGate SPI device. The *`hHandle`* parameter is a return value of a succesfull call to the function `LS_SPIOpenDevice`.

Important

It is recommended to close the connection, because all internally allocated system resources are freed again and the connected AnaGate device is signalled that the active connection is not longer in use and can be used again for new connect requests.

See also

LS_SPIOpenDevice

 © 2007-2010 Analytica GmbH

LS_SPISetGlobals

LS_SPISetGlobals — Sets the global settings, which are to be used on the AnaGate SPI.

Syntax

```
int LS_SPISetGlobals(int hHandle, int nBaudrate, unsigned char
nSigLevel, unsigned char nAuxVoltage, unsigned char nClockMode);
```

Parameter

hHandle
: Valid access handle.

nBaudrate
: The baud rate to be used. The values can be set individually, like

- 500.000 for 500kBit
- 1.000.000 for 1MBit
- 5.000.000 for 5MBit

Note

The required baud rate can be different from the value actually used because of internal hardware restrictions (frequency of the oscillator). If it is not possible to adjust the baud rate exactly to the parsed value, the nearest smaller possible value is used instead.

nSigLevel
: The voltage level for SPI signals to be used. Following values are allowed:

- 0 = Outputs in High Impedance Modus (Standard mode).
- 1 = +5.0V for the signals.
- 2 = +3.3V for the signals.
- 3 = +2.5V for the signals.

nAuxVoltage
: The voltage level of the support voltage to be used. Following values are allowed:

- 0 = support voltage is +3.3V.
- 1 = support voltage is 2.5V.

nClockMode
: The phase and polarity of the clock signal. Following values are allowed:

- 0 = CPHA=0 and CPOL=0.
- 1 = CPHA=0 and CPOL=1.

© 2007-2010 Analytica GmbH

- 2 = CPHA=1 and CPOL=0.
- 3 = CPHA=1 and CPOL=1.

Return value

RC Returns `0` if successful, or an error value otherwise (Appendix A, *API return codes*).

Description

Sets the global settings of SPI interface of the AnaGate SPI or the AnaGate Universal Programmer. These settings are not saved permantently on the device and are reset every device restart.

See also

LS_SPIGetGlobals

 © 2007-2010 Analytica GmbH

LS_SPIGetGlobals

LS_SPIGetGlobals — Returns the currently used global settings of the AnaGate SPI.

Syntax

```
int RC, int nBaudrate, int nSigLevel, int nAuxVoltage, int nClockMode
= LS_SPIGetGlobals(int hHandle);
```

Parameter

hHandle Valid access handle.

Return values

RC
: Returns `0` if successful, or an error value otherwise (Appendix A, *API return codes*).

nBaudrate
: The baud rate currently used on the SPI bus in kBit.

nSigLevel
: The voltage level currently used by the AnaGate SPI. Following values are possible:
  - 0 = Outputs in High Impedance Modus (Standard mode).
  - 1 = +5.0V for the signals.
  - 2 = +3.3V for the signals.
  - 3 = +2.5V for the signals.

nAuxVoltage
: The voltage level of the support voltage currently used by the AnaGate SPI. Following values are possible:
  - 0 = support voltage is +3.3V.
  - 1 = support voltage is 2.5V.

nClockMode
: The phase and polarity of the colck signal currently used by the AnaGate SPI. Following values are possible:
  - 0 = CPHA=0 and CPOL=0.
  - 1 = CPHA=0 and CPOL=1.
  - 2 = CPHA=1 and CPOL=0.
  - 3 = CPHA=1 and CPOL=1.

Description

Returns the currently used global settings of SPI interface of the AnaGate SPI or the AnaGate Universal Programmer.

© 2007-2010 Analytica GmbH

See also

LS_SPISetGlobals

 © 2007-2010 Analytica GmbH

LS_SPIDataReq

LS_SPIDataReq — Writes and reads data to/from SPI bus.

Syntax

`int RC, table(nReadLen) tabRead =` **`LS_SPIDataReq`**`(int` *`hHandle`*`, int` *`nWriteLen`*`, int` *`nReadLen`*`, table(nWriteLen)` *`tabWrite`*`);`

Parameter

hHandle	Valid access handle.
nWriteLen	Length of the data buffer *`pcBufWrite`* (byte count).
nReadLen	Number of bytes to read.
tabWrite	Buffer with the data that is to be sent to the SPI partner.

Return value

RC	Returns `0` if successful, or an error value otherwise (Appendix A, *API return codes*).
tabRead	Table with data received from the SPI partner.

Description

Sends data to the SPI bus and receives data from the SPI bus.

On the SPI bus Data is transferred on two seperates data lines full duplex (SDO and SDI). The `SPIDatReq` has to split a single data transfer in two steps because of the spacial separation to the SPI bus. First the write data buffer is put into a TCP data telegram and sent to the AnaGate SPI. The AnaGate SPI makes the real data transfer on the SPI bus and send back a confirmation including the data received from the bus.

Important

It is impossible to detect that no device is present at the SPI bus. So, if no device is attached, the requested number of bytes are returned anyway - in this case the read buffer is filled with `0`.

See the following example for sending a command to the connected SPI bus.

```
tabWrite = {};
for i=1, 10 , 1 do
   table.insert(tabWrite, i );
end;

nRC, hHandle = SPIOpenDevice("192.168.1.254", 5000);
if ( nRC == 0 ) then
   // send 1 byte and receive 1 byte
```

 © 2007-2010 Analytica GmbH

```
   nRC, tabRead = LS_SPIDataReq( hHandle, 1, 1, tabWrite );
   // send 1 byte and receive 5 byte
   nRC, tabRead = LS_SPIDataReq( hHandle, 1, 5, tabWrite );
   // send 2 byte and receive 1 byte
   nRC, tabRead = LS_SPIDataReq( hHandle, 2, 1, tabWrite );

   LS_SPICloseDevice(hHandle);
end;
```

 © 2007-2010 Analytica GmbH

LS_SPIErrorMessage

LS_SPIErrorMessage — Returns a description of the given error code as a text string.

Syntax

```
string sErrorMsg LS_SPIErrorMessage(int nRetCode);
```

Parameter

nRetCode   Error code for which the error description is to be determined.

Return value

sErrorMsg   Textual description of the error code.

Description

Returns a textual description of the parsed error code (see Appendix A, *API return codes*).

See the following example in LUA scripting language.

```
nRC=0;
sErrorText = 'No Error';

//... call a API function here

sErrorText = LS_SPIErrorMessage(nRC);
print(sErrorText);
```


© 2007-2010 Analytica GmbH

LS_SPIReadDigital

LS_SPIReadDigital — Reads the current values of digital input and output registers of the AnaGate device.

Syntax

```
int RC, int nInputBits, int nOutputBits = LS_SPIReadDigital(int hHandle);
```

Parameter

hHandle Valid access handle.

Return value

RC	Returns `0` if successful, or an error value otherwise (Appendix A, *API return codes*).
nInputBits	Pointer to the current value of the digital input register. Currently only bits 0 to 3 are used, other bits are reserved for future use and are set to 0.
nOutputBits	Pointer to the current value of the digital output register. Currently only bits 0 to 3 are used, other bits are reserved for future use and are set to 0.

Description

All models of the AnaGate series (except the model AnaGate CAN uno in DIN rail case) have connectors for 4 digtial inputs and 4 digital outputs at the rear panel.

The current values of the digital inputs and outputs can be retrieved with the `LS_SPIReadDigital` function.

See the following example for setting an reading the digital IO.

```
nOutputs = 0x03;

nRC, hHandle = LS_SPIOpenDevice( 400000, "192.168.0.254", 5000 );
if ( nRC == 0 ) then
   // set the digital output register (PIN 0 and PIN 1 to HIGH value)
   nRC = LS_SPIWriteDigital( hHandle, nOutputs );

   // read all input and output registers
   nRC, nInputs, nOutputs = LS_SPIReadDigital( hHandle );

   LS_SPICloseDevice(hHandle);
end;
```

See also

LS_SPIWriteDigital

 © 2007-2010 Analytica GmbH

LS_SPIWriteDigital

LS_SPIWriteDigital — Writes a new value to the digital output register of the AnaGate device.

Syntax

```
int RC = LS_SPIWriteDigital(int hHandle, int nOutputBits);
```

Parameter

hHandle	Valid access handle.
nOutputBits	New register value. Currently only bits 0 to 3 are used, other bits are reserved for future use.

Return value

RC Returns `0` if successful, or an error value otherwise (Appendix A, *API return codes*).

Description

All models of the AnaGate series (except the model AnaGate CAN uno in DIN rail case) have connectors for 4 digtial inputs and 4 digital outputs at the rear panel.

The digital outputs can be written with the `LS_SPIWriteDigital` function.

A simple example for reading/writing of the IOs can be found at the description of `LS_SPIReadDigital`.

See also

LS_SPIReadDigital

© 2007-2010 Analytica GmbH

Chapter 12. I2C Reference

Philips Semiconductors (now NXP Semiconductors) has developed a simple bidirectional 2-wire bus for efficient inter-IC control. This bus is called the Inter-IC or I2C-bus. Only two bus lines are required: a serial data line (SDA) and a serial clock line (SCL). Serial, 8-bit oriented, bidirectional data transfers can be made at up to 100 kbit/s in the Standard-mode, up to 400 kbit/s in the Fast-mode, up to 1 Mbit/s in the Fast-mode Plus (Fm+), or up to 3.4 Mbit/s in the High-speed mode. [NXP-I2C].

The I2C gateway models of the AnaGate series provides access to a I2C bus via a standard networking. With the I2C API these I2C gateways can be easily controlled. The programming interface is identical for all devices and used the network protocol TCP in general.

Following devices can be addresse via the I2C API interface:

- AnaGate I2C
- AnaGate Universal Programmer

© 2007-2010 Analytica GmbH

LS_I2COpenDevice

LS_I2COpenDevice — Opens a network connection to an AnaGate I2C or an AnaGate Universal Programmer).

Syntax

```
int RC, int Handle = LS_I2COpenDevice(unsigned int nBaudrate, string
sIPAddress, int nTimeout);
```

Parameter

nBaudrate
: Baud rate to be used for the I2C bus. Teh value can be set individually, like

- 100000 for 100kBit (standard mode)
- 400000 for 400kBit (fast mode)

Note

Values above 400kBit are ignored by the AnaGate SPI.

sIPAddress
: Network address of the AnaGate partner.

nTimeout
: Default timeout for accessing the AnaGate in milliseconds.

A timeout is reported if the AnaGate partner does not respond within the defined timeout period. This global timeout value is valid on the current network connection for all commands and functions which do not offer a specific timeout value.

Return values

RC
: Returns `0` if successful, or an error value otherwise (Appendix A, *API return codes*).

Handle
: Access handle if successfully connected to the AnaGate device.

Description

Opens a TCP/IP connection to an AnaGate I2C (resp. AnaGate Universal Programmer). After the connection is established, access to the I2C bus is possible.

Note

The AnaGate I2C (resp. the I2C interface of an AnaGate Universal Programmer) does not allow more than one concurrent network connection. During an established network connection all new connections are refused.

See the following example for the initial programming steps.

 © 2007-2010 Analytica GmbH

```
nRC, nHandle  = LS_I2COpenDevice( 1000000, "192.168.0.254", 5000 );
if (nRC ~= 0) then
   print(LS_I2CErrorMessage(nRC));
   exit();
end;

-- now do something

LS_I2CCloseDevice(nHandle);
```

See also

LS_I2CCloseDevice

© 2007-2010 Analytica GmbH

LS_I2CCloseDevice

LS_I2CCloseDevice — Closes an open network connection to an AnaGate I2C device.

Syntax

```
int RC = LS_I2CCloseDevice(int hHandle);
```

Parameter

hHandle Valid access handle.

Return value

RC Returns 0 if successful, or an error value otherwise (Appendix A, *API return codes*).

Description

Closes an open network connection to an AnaGate I2C device. The *hHandle* parameter is a return value of a succesfull call to the function LS_I2COpenDevice.

Important

It is recommended to close the connection, because all internally allocated system resources are freed again and the connected AnaGate device is signalled that the active connection is not longer in use and can be used again for new connect requests.

See also

LS_I2COpenDevice

 © 2007-2010 Analytica GmbH

LS_I2CReset

LS_I2CReset — Resets the I2C Controller in an AnaGate I2C device.

Syntax

```
int RC = LS_I2CReset(int hHandle);
```

Parameter

hHandle Valid access handle.

Return value

RC Returns `0` if successful, or an error value otherwise (Appendix A, *API return codes*).

Description

Resets the I2C Controller in an AnaGate I2C device.

© 2007-2010 Analytica GmbH

LS_I2CRead

LS_I2CRead — Reads data from an I2C partner.

Syntax

```
int RC, table(nBufferLen) tabBuffer = LS_I2CRead(int hHandle, unsigned
short nSlaveAddress, int nBufferLen);
```

Parameter

hHandle	Valid access handle.
nSlaveAddress	Slave address of the I2C partner. The slave address can represent a so-called 7-bit or 10-bit address. (siehe Appendix B, *I2C slave address formats*).
nBufferLen	Number of bytes to read.

Return values

RC	Returns `0` if successful, or an error value otherwise (Appendix A, *API return codes*).
tabBuffer	Byte buffer in which the data received from the I2C partner is stored.

Description

Reads data from an I2C partner. The user must ensure that the setup of the address of the I2C partner are correct.

The R/W bit of the slave address does not have to be explicitly set by the user.

See also

LS_I2CWrite

© 2007-2010 Analytica GmbH

LS_I2CWrite

LS_I2CWrite — Writes data to an I2C partner.

Syntax

```
int RC, int ErrorByte = LS_I2CWrite(int hHandle, unsigned short
nSlaveAddress, table(nBufferLen) tabBuffer, int nBufferLen);
```

Parameter

hHandle	Valid access handle.
nSlaveAddress	Slave address of the I2C partner. The slave address can represent a so-called 7-bit or 10-bit address. (siehe Appendix B, *I2C slave address formats*).
nBufferLen	Size of bytes to be read.
tabBuffer	Byte buffer with the data that is to be sent to the I2C partner.

Return value

RC	Returns `0` if successful, or an error value otherwise (Appendix A, *API return codes*).
ErrorByte	Number of byte in data buffer which raises the error if the function failed.

Description

Writes data to an I2C partner. The user must ensure that the setup of the data buffer and the address of the I2C partner are correct.

The R/W bit of the slave address does not have to be explicitly set by the user.

See also

LS_I2CRead

© 2007-2010 Analytica GmbH

LS_I2CReadDigital

LS_I2CReadDigital — Reads the current values of digital input and output registers of the AnaGate device.

Syntax

`int RC, int nInputBits, int nOutputBits = **LS_I2CReadDigital**(int *hHandle*);`

Parameter

hHandle Valid access handle.

Return value

RC	Returns `0` if successful, or an error value otherwise (Appendix A, *API return codes*).
nInputBits	Pointer to the current value of the digital input register. Currently only bits 0 to 3 are used, other bits are reserved for future use and are set to 0.
nOutputBits	Pointer to the current value of the digital output register. Currently only bits 0 to 3 are used, other bits are reserved for future use and are set to 0.

Description

All models of the AnaGate series (except the model AnaGate CAN uno in DIN rail case) have connectors for 4 digtial inputs and 4 digital outputs at the rear panel.

The current values of the digital inputs and outputs can be retrieved with the `LS_I2CReadDigital` function.

See the following example for setting an reading the digital IO.

```
nOutputs = 0x03;

nRC, hHandle = LS_I2COpenDevice( 400000, "192.168.0.254", 5000 );
if ( nRC == 0 ) then
   // set the digital output register (PIN 0 and PIN 1 to HIGH value)
   nRC = LS_I2CWriteDigital( hHandle, nOutputs );

   // read all input and output registers
   nRC, nInputs, nOutputs = LS_I2CReadDigital( hHandle );

   LS_I2CCloseDevice(hHandle);
end;
```

See also

LS_I2CWriteDigital

 © 2007-2010 Analytica GmbH

LS_I2CWriteDigital

LS_I2CWriteDigital — Writes a new value to the digital output register of the AnaGate device.

Syntax

```
int RC = LS_I2CWriteDigital(int hHandle, int nOutputBits);
```

Parameter

hHandle	Valid access handle.
nOutputBits	New register value. Currently only bits 0 to 3 are used, other bits are reserved for future use.

Return value

RC Returns `0` if successful, or an error value otherwise (Appendix A, *API return codes*).

Description

All models of the AnaGate series (except the model AnaGate CAN uno in DIN rail case) have connectors for 4 digtial inputs and 4 digital outputs at the rear panel.

The digital outputs can be written with the `LS_I2CWriteDigital` function.

A simple example for reading/writing of the IOs can be found at the description of `LS_I2CReadDigital`.

See also

LS_I2CReadDigital

© 2007-2010 Analytica GmbH

LS_I2CErrorMessage

LS_I2CErrorMessage — Returns a description of the given error code as a text string.

Syntax

```
string sErrorMsg LS_I2CErrorMessage(int nRetCode);
```

Parameter

nRetCode Error code for which the error description is to be determined.

Return value

sErrorMsg Textual description of the error code.

Description

Returns a textual description of the parsed error code (see Appendix A, *API return codes*).

See the following example in LUA scripting language.

```
nRC=0;
sErrorText = 'No Error';

//... call a API function here

sErrorText = LS_I2CErrorMessage(nRC);
print(sErrorText);
```


© 2007-2010 Analytica GmbH

LS_I2CReadEEPROM

LS_I2CReadEEPROM — Reads data from an EEPROM on the I2C bus.

Syntax

```
int RC, tabData(nDatLen) = LS_I2CReadEEPROM(int hHandle, unsigned short nSubAddress, unsigned int nOffset, unsigned int nOffsetFormat, int nDataLen);
```

Parameter

hHandle — Valid access handle.

nSubAddress — Subaddress of the EEPROM to communicate with. The valid values for *nSubAddress* are governed by the setting used in the parameter *nOffsetFormat* (bits 8-10). If the EEPROM type needs to use bits of the *Chip Enable Address* to address the internal memory, only the remaining bits can be used to select the device itself.

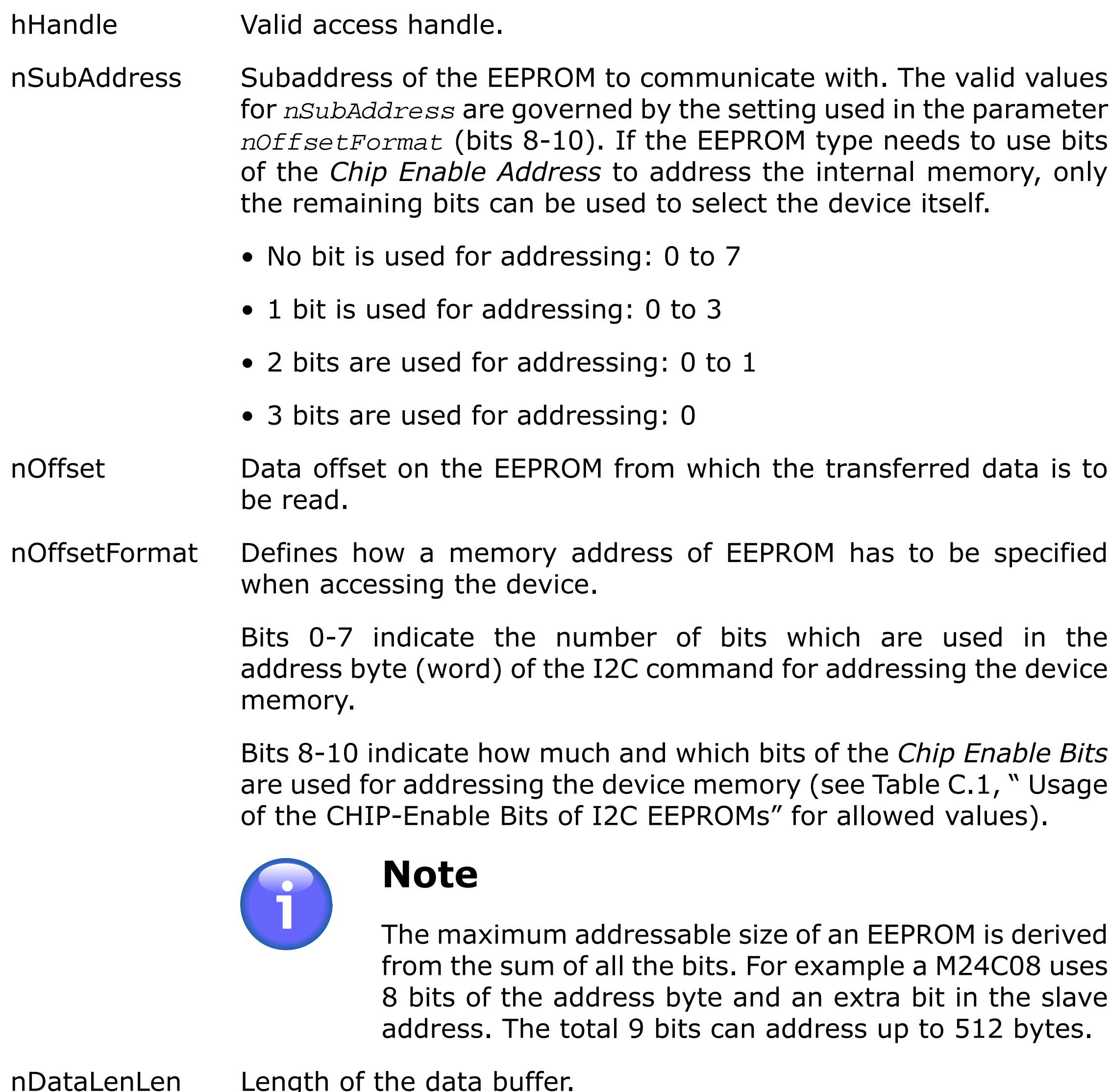
- No bit is used for addressing: 0 to 7
- 1 bit is used for addressing: 0 to 3
- 2 bits are used for addressing: 0 to 1
- 3 bits are used for addressing: 0

nOffset — Data offset on the EEPROM from which the transferred data is to be read.

nOffsetFormat — Defines how a memory address of EEPROM has to be specified when accessing the device.

Bits 0-7 indicate the number of bits which are used in the address byte (word) of the I2C command for addressing the device memory.

Bits 8-10 indicate how much and which bits of the *Chip Enable Bits* are used for addressing the device memory (see Table C.1, " Usage of the CHIP-Enable Bits of I2C EEPROMs" for allowed values).



Note

The maximum addressable size of an EEPROM is derived from the sum of all the bits. For example a M24C08 uses 8 bits of the address byte and an extra bit in the slave address. The total 9 bits can address up to 512 bytes.

nDataLenLen — Length of the data buffer.

Return values

RC — Returns `0` if successful, or an error value otherwise (Appendix A, *API return codes*).

© 2007-2010 Analytica GmbH

tabData Table with data read from EEPROM.

Description

The `LS_I2CReadEEPROM` function reads data from an I2C EEPROM.

Of course all access to the memory of an EEPROM is done by standard I2C read or write commands. So, when reading from the memory only the matching slave address, the memory offset address and the data has to be sent to the I2C bus.

`LS_I2CReadEEPROM` translates the given memory address on the chip by means of the sub address and the addressing mode of the present EEPROM type. The slave address of the EEPROM is automatically determined and not mandatory for the fucntion call.

A programming example which clears a **ST24C1024** can be found at the description of LS_I2WriteEEPROM.

See also

LS_I2CWriteEEPROM

Appendix C, *Programming I2C EEPROM*

© 2007-2010 Analytica GmbH

LS_I2CWriteEEPROM

LS_I2CWriteEEPROM — Writes data to an I2C EEPROM.

Syntax

```
int RC = LS_I2CWriteEEPROM(int hHandle, unsigned short nSubAddress,
unsigned int nOffset, unsigned int nOffsetFormat, int nDataLen,
table(nDataLen) tabData);
```

Parameter

hHandle — Valid access handle.

nSubAddress — Subaddress of the EEPROM to communicate with. The valid values for *nSubAddress* are governed by the setting used in the parameter *nOffsetFormat* (bits 8-10). If the EEPROM type needs to use bits of the *Chip Enable Address* to address the internal memory, only the remaining bits can be used to select the device itself.

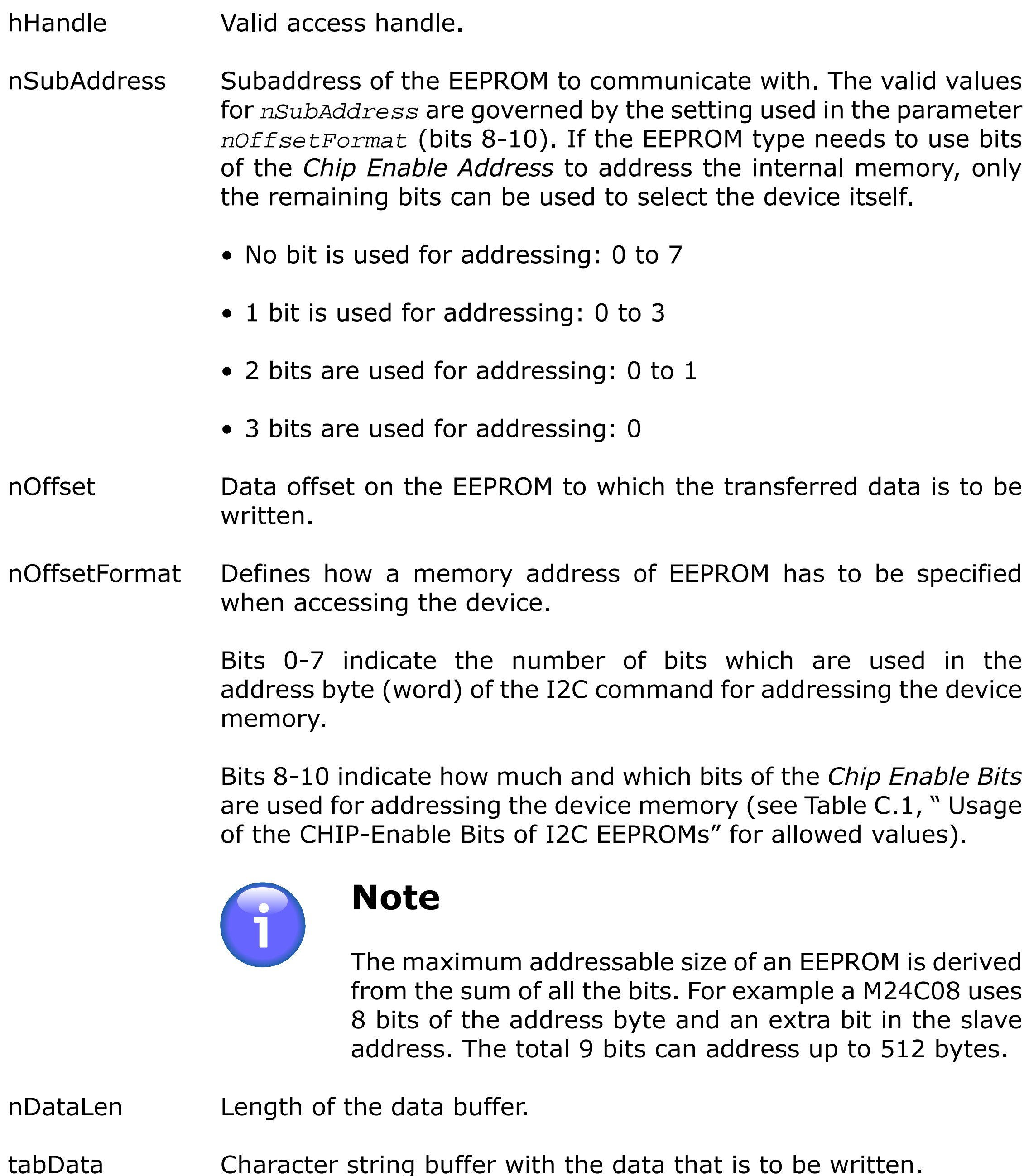
- No bit is used for addressing: 0 to 7
- 1 bit is used for addressing: 0 to 3
- 2 bits are used for addressing: 0 to 1
- 3 bits are used for addressing: 0

nOffset — Data offset on the EEPROM to which the transferred data is to be written.

nOffsetFormat — Defines how a memory address of EEPROM has to be specified when accessing the device.

Bits 0-7 indicate the number of bits which are used in the address byte (word) of the I2C command for addressing the device memory.

Bits 8-10 indicate how much and which bits of the *Chip Enable Bits* are used for addressing the device memory (see Table C.1, “ Usage of the CHIP-Enable Bits of I2C EEPROMs” for allowed values).



Note

The maximum addressable size of an EEPROM is derived from the sum of all the bits. For example a M24C08 uses 8 bits of the address byte and an extra bit in the slave address. The total 9 bits can address up to 512 bytes.

nDataLen — Length of the data buffer.

tabData — Character string buffer with the data that is to be written.

 © 2007-2010 Analytica GmbH

Return value

RC Returns `0` if successful, or an error value otherwise (Appendix A, *API return codes*).

Description

The `LS_I2CWriteEEPROM` function writes data to an I2C EEPROM.

Of course all access to the memory of an EEPROM is done by standard I2C read or write commands. So, when writing to the memory only the matching slave address, the memory offset address and the data has to be sent to the I2C bus.

`LS_I2CWriteEEPROM` translates the given memory address on the chip by means of the sub address and the addressing mode of the present EEPROM type. The slave address of the EEPROM is automatically determined and not mandatory for the fucntion call.

Tip

It is important to note that an EEPROM is divided into memory pages, and that a single write command can only program data within a page. Users of `LS_I2CWriteEEPROM` must ensure to do not write across page limits. The page size depends on the EEEPOM type.

See the following example for writing data to a **ST24C1024**.

```
tabData = {};
for i=1, 256 , 1 do
   table.insert(tabData, 0x0 );
end;

nSubAddress = 0; 1
nOffsetFormat = 0x10+0x0F; 2

RC, hHandle = LS_I2COpenDevice(400000, "192.168.0.254", 5000);
if ( RC == 0 ) then
   for page=0, 512-1, 1 do
      RC = LS_I2CWriteEEPROM( hHandle, nSubAddress, i*256, nOffSetFormat, 256, tabData ); 3
   end;
   LS_I2CCloseDevice(hHandle);
end;
```

1. It is possible to address 4 individual ST24C1024 on a single I2C bus. By selection of subaddress `0` the control pins E2 and E1 have to be LOW.
2. 17 address bits are used to address the 128KB of a ST24C1024. 16 bits are set via the address bytes of the write command: `16=0x0F`. The address bit A16 is set via the E0 bit of the *Chip Enable Address*, therefore addressing mode `1` (E2-E1-A0) must be set: `0x10`.
3. The page size of a ST24C1024 is 256 byte, every page is programmed full within the for-loop.

See also

LS_I2CReadEEPROM

Appendix C, *Programming I2C EEPROM*

© 2007-2010 Analytica GmbH

Chapter 13. CANOpen functions

CANopen® is a communication protocol and device profile specification for embedded systems used in automation. Standardised in Europe as `EN 50325-4` (see [CiA-DS301]), it is mangaged by the user organisation CAN in Automation (**CiA**).

Users of the *AnaGate API* are allowed to execute the in following decribed CANopen services as CANopen master.

DRAFT

© 2007-2010 Analytica GmbH

LS_CANopenSetConfig

LS_CANopenSetConfig — Configure the connection specific device settings for CANOpen operation.

Synopsis

```
int RC = LS_CANopenSetConfig( int Handle , int CANOpenConfig );
```

Description

Parameter

int Handle

int CANOpenConfig	0:	Die CANOpen Funktionalität wird nicht unterstützt (Standard). Ein Aufruf einer CANOpen Funktion wird negativ (FFh) quittiert. Empfangene CAN-Daten werden immer als Standard DataIndication (OP_ANAGATE_CAN_DATA_IND) gesendet.
	1:	Die CANOpen Funktionalität ist eingeschaltet und es können die nachfolgend beschriebenen Funktionen verwendet werden.

Return values

If no error occurs, the return will be null(0). Otherwise the returncode will be different null(0).

 © 2007-2010 Analytica GmbH

LS_CANopenGetConfig

LS_CANopenGetConfig

Synopsis

```
int RC = LS_CANopenGetConfig( int Handle );
```

Description

Parameter

int Handle

Return values

int RC	If no error occurs, the return will be null(0). Otherwise the returncode will be different null(0).
int CANOpenConfig	0: Die CANOpen Funktionalität wird nicht unterstützt (Standard). Ein Aufruf einer CANOpen Funktion wird negativ (FFh) quittiert. Empfangene CAN-Daten werden immer als Standard DataIndication (OP_ANAGATE_CAN_DATA_IND) gesendet. 1: Die CANOpen Funktionalität ist eingeschaltet und es können die nachfolgend beschriebenen Funktionen verwendet werden.

DRAFT

 © 2007-2010 Analytica GmbH

LS_CANopenSetSYNCMode

LS_CANopenSetSYNCMode

Synopsis

```
int RC = LS_CANopenSetSYNCMode( int Handle , int PeriodTime );
```

Description

Parameter

int Handle

int PeriodTime

Return values

int RC If no error occurs, the return will be null(0). Otherwise the returncode will be different null(0).

DRAFT

© 2007-2010 Analytica GmbH

LS_CANopenSetCallbacks

LS_CANopenSetCallbacks

Synopsis

int RC = **LS_CANopenSetCallbacks**(int *Handle* , string *CallbackFunctionPDO* , string *CallbackFunctionSYNC* , string *CallbackFunctionEMCY* , string *CallbackFunctionGUARD* , string *CallbackFunctionUndefined*);

Description

Parameter

int Handle

string CallbackFunctionPDO

string
CallbackFunctionSYNC

string
CallbackFunctionEMCY

string
CallbackFunctionGUARD

string
CallbackFunctionUndefined

Return values

int RC If no error occurs, the return will be null(0). Otherwise the returncode will be different null(0).

 © 2007-2010 Analytica GmbH

LS_CANopenGetPDO

LS_CANopenGetPDO

Synopsis

```
int AvailMessages, int NodeID, int PDOTyp, table(1-8) Data, int Seconds,
int Microseconds = LS_CANopenGetPDO( int Handle , int Timeout );
```

Description

Parameter

int Handle

int Timeout

Return values

= -2:

= -1:

>= 0:

int NodeID

int PDOTyp

table(1-8) Data

int Seconds

int Microseconds

DRAFT

© 2007-2010 Analytica GmbH

LS_CANopenGetSYNC

LS_CANopenGetSYNC

Synopsis

```
int AvailMessages, int ReturnCode, int Seconds, int Microseconds =
LS_CANopenGetSYNC( int Handle , int Timeout );
```

Description

Parameter

int Handle

int Timeout

Return values

= -2:

= -1:

>=
0:

int ReturnCode

int Seconds

int Microseconds

DRAFT

 © 2007-2010 Analytica GmbH

LS_CANopenGetEMCY

LS_CANopenGetEMCY

Synopsis

```
int AvailMessages, int NodeID, int ErrorCode, int ErrorRegister, table(5) ErrorDescription, int Seconds, int Microseconds = LS_CANopenGetEMCY( int Handle , int Timeout );
```

Description

Parameter

int Handle

int Timeout

Return values

= -2:

= -1:

>= 0:

int NodeID

int ErrorCode

int ErrorRegister

table(5) ErrorDescritpion

int Seconds

int Microseconds

© 2007-2010 Analytica GmbH

LS_CANopenGetGUARD

LS_CANopenGetGUARD

Synopsis

```
int AvailMessages, int NodeID, int Status, int Seconds, int Microseconds
= LS_CANopenGetGUARD( int Handle , int Timeout );
```

Description

Parameter

int Handle

int Timeout

Return values

= -2:

= -1:

>= 0:

int NodeID

int Status

int Seconds

int Microseconds

DRAFT

 © 2007-2010 Analytica GmbH

LS_CANopenGetUndefined

LS_CANopenGetUndefined

Synopsis

```
int AvailMessages, int CANID, int NodeID, int FunctionCode, table(1-8)
Data, int Seconds, int Microseconds = LS_CANopenGetUndefined( int Handle
, int Timeout );
```

Description

Parameter

int Handle

int Timeout

Return values

= -2:

= -1:

>= 0:

int CANID

int NodeID

int FunctionCode

table(1-8) Data

int Seconds

int Microseconds

Als undefinierte Nachrichten werden die eingehenden Nachrichten bezeichnet die anhand des Funktionscodes keiner CANOpen Funktion zugeordnet werden können. Sollte nur in Netzen vorkommen in denen gleichzeitig zu CANOpen noch Standard CAN oder andere Protokolle betrieben werden.

 © 2007-2010 Analytica GmbH

LS_CANopenSendNMT

LS_CANopenSendNMT

Synopsis

```
int RC = LS_CANopenSendNMT( int Handle , int NodeID , int NMTTyp );
```

Description

Parameter

int Handle

int NodeID

int NMTTyp

Return values

int RC   If no error occurs, the return will be null(0). Otherwise the returncode will be different null(0).

DRAFT

 © 2007-2010 Analytica GmbH

LS_CANopenSendSYNC

LS_CANopenSendSYNC

Synopsis

```
int RC = LS_CANopenSendSYNC( int Handle );
```

Description

Parameter

int Handle

Return values

int RC If no error occurs, the return will be null(0). Otherwise the returncode will be different null(0).

 © 2007-2010 Analytica GmbH

LS_CANopenSendTIME

LS_CANopenSendTIME

Synopsis

```
int RC = LS_CANopenSendTIME( int Handle , int Day , int Milliseconds );
```

Description

Parameter

int Handle

int Day

int Milliseconds

Rückgabewerte

int RC If no error occurs, the return will be null(0). Otherwise the returncode will be different null(0).

 © 2007-2010 Analytica GmbH

DRAFT

LS_CANopenSendPDO

LS_CANopenSendPDO

Synopsis

```
int RC = LS_CANopenSendPDO( int Handle , int NodeID , int PDOTyp , int
DataLength , table(1-8) SendData );
```

Description

Parameter

int Handle

int NodeID

int PDOTyp

int DataLength

table(1-8) SendData

Return values

int RC If no error occurs, the return will be null(0). Otherwise the returncode will be different null(0).

DRAFT

 © 2007-2010 Analytica GmbH

LS_CANopenSendSDORead

LS_CANopenSendSDORead

Synopsis

```
int RC, int SDOReadType, int SDOReadData = LS_CANopenSendSDORead( int
Handle , int NodeID , int Index , int Subindex , int Timeout );
```

Description

Parameter

int Handle

int NodeID

int Index

int Subindex

int TimeOut

Return values

int RC	If no error occurs, the return will be null(0). Otherwise the returncode will be different null(0).
int SDOReadType	
int SDOReadData	

DRAFT

 © 2007-2010 Analytica GmbH

LS_CANopenSendSDOWrite

LS_CANopenSendSDOWrite

Synopsis

`int RC, int SDOReadType, int SDOReadData =` **`LS_CANopenSendSDOWrite`**`( int` *`Handle`* `, int` *`NodeID`* `, int` *`SDOWriteTyp`* `, int` *`Index`* `, int` *`Subindex`* `, int` *`Timeout`* `, int` *`SDOWriteData`* `);`

Description

Parameter

int Handle

int NodeID

int SDOWriteTyp

int Index

int Subindex

int Timeout

int SDOWriteData

Return values

int RC — If no error occurs, the return will be null(0). Otherwise the returncode will be different null(0).

int SDOReadTyp

int SDOReadData

DRAFT

 © 2007-2010 Analytica GmbH

LS_CANopenSendSDOReadBlock

LS_CANopenSendSDOReadBlock

Synopsis

int RC, int SDOReadType, int ReadLen, table ReadData = **LS_CANopenSendSDOReadBlock**(int *Handle* , int *NodeID* , int *Index* , int *Subindex* , int *Timeout* , int *ReadLen*);

Description

Parameter

int Handle

int NodeID

int Index

int Subindex

int Timeout

int ReadLen

Return values

int RC	If no error occurs, the return will be null(0). Otherwise the returncode will be different null(0).
int SDOReadType	
int ReadLen	
table ReadData	

DRAFT

 © 2007-2010 Analytica GmbH

LS_CANopenSendSDOWriteBlock

LS_CANopenSendSDOWriteBlock

Synopsis

```
int RC, int SDOReadTyp, int SDOReadData = LS_CANopenSendSDOWriteBlock( int Handle , int SDOReadTyp , int SDOReadData );
```

Description

Parameter

int Handle

int NodeID

int Index

int Subindex

int Timeout

int WriteLen

Return values

int RC If no error occurs, the return will be null(0). Otherwise the returncode will be different null(0).

int SDOReadType

int SDOReadData

 © 2007-2010 Analytica GmbH

Programmer example

Programmer example

 © 2007-2010 Analytica GmbH

Example 13.1. CANOpen - LUA script example

```
-- Filter: alle CAN-Identifier akzeptieren
aFilter = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
            0x00, 0x1FFFFFFF, 0x00, 0x1FFFFFFF,
            0x00, 0x1FFFFFFF, 0x00, 0x1FFFFFFF};

aSendData = { 0xF1, 0xF2, 0xF3, 0xF4, 0xF5, 0xF6, 0xF7, 0xF8 };
--************************************************
function printf(...)
   io.write(string.format(...))
   io.flush();
end
--************************************************
function main()
   -- Open
   nRC, nHandle = LS_CANOpenDevice( false, true, 5001, "10.1.2.160", 5000 );
   if (nRC ~= 0) then
      print(LS_CANErrorMessage(nRC));
      exit();
   end;

   nRC, nHandle2 = LS_CANOpenDevice( false, true, 5001, "10.1.2.161", 5000 );
   if (nRC ~= 0) then
      print(LS_CANErrorMessage(nRC));
      exit();
   end;

   -- Filter setzen
   nRC = LS_CANSetFilter(nHandle,  aFilter);
   nRC = LS_CANSetFilter(nHandle2, aFilter);

   -- aktuelle Zeit auf den AnaGate Device setzen
   nRC, oTime = LS_GetTime();
   nRC = LS_CANSetTime(nHandle,  oTime[1], oTime[2]);
   nRC = LS_CANSetTime(nHandle2, oTime[1], oTime[2]);

   -- Globals setzen
   nRC = LS_CANSetGlobals( nHandle,  500000, 0, true, false, false );
   nRC = LS_CANSetGlobals( nHandle2, 500000, 0, true, false, false );

   -- Endlosschleife
   repeat
      -- 1 Datenpackete auf dem 1. AnaGate CAN Device versenden
      nRC = LS_CANWrite(nHandle, 1, 8, aSendData );

      LS_Sleep(20); -- 20Millisekunden warten

      -- Datenpacket auf 2. AnaGate CAN Device empfangen
      nAvail, ID, Len, Data, Sec, Microsec = LS_CANGetMessage(nHandle2, 10);
      while nAvail>=0 do
         nAvail, ID, Len, Data, Sec, Microsec = LS_CANGetMessage(nHandle2, 10);
      end;
   until (false);

   -- Verbindungen beenden
   LS_CANCloseDevice(nHandle);
   LS_CANCloseDevice(nHandle2);
end;
```

 © 2007-2010 Analytica GmbH

Chapter 14. LUA programming exa

14.1

```
-- Filter: alle CAN-Identifier akzeptieren
aFilter = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
            0x00, 0x1FFFFFFF, 0x00, 0x1FFFFFFF,
            0x00, 0x1FFFFFFF, 0x00, 0x1FFFFFFF};

aSendData = { 0xF1, 0xF2, 0xF3, 0xF4, 0xF5, 0xF6, 0xF7, 0xF8 };
--**************************************************
function printf(...)
   io.write(string.format(...))
   io.flush();
end
--**************************************************
function main()
   -- Open
   nRC, nHandle = LS_CANOpenDevice( false, true, 5001, "10.1.2.160", 5000 );
   if (nRC ~= 0) then
      print(LS_CANErrorMessage(nRC));
      exit();
   end;

   nRC, nHandle2 = LS_CANOpenDevice( false, true, 5001, "10.1.2.161", 5000 );
   if (nRC ~= 0) then
      print(LS_CANErrorMessage(nRC));
      exit();
   end;

   -- Filter setzen
   nRC = LS_CANSetFilter(nHandle,  aFilter);
   nRC = LS_CANSetFilter(nHandle2, aFilter);

   -- aktuelle Zeit auf den AnaGate Device setzen
   nRC, oTime = LS_GetTime();
   nRC = LS_CANSetTime(nHandle,  oTime[1], oTime[2]);
   nRC = LS_CANSetTime(nHandle2, oTime[1], oTime[2]);

   -- Globals setzen
   nRC = LS_CANSetGlobals( nHandle,  500000, 0, true, false, false );
   nRC = LS_CANSetGlobals( nHandle2, 500000, 0, true, false, false );

   -- Endlosschleife
   repeat
      -- 1 Datenpackete auf dem 1. AnaGate CAN Device versenden
      nRC = LS_CANWrite(nHandle, 1, 8, aSendData );

      LS_Sleep(20); -- 20Millisekunden warten

      -- Datenpacket auf 2. AnaGate CAN Device empfangen
      nAvail, ID, Len, Data, Sec, Microsec = LS_CANGetMessage(nHandle2, 10);
      while nAvail>=0 do
         nAvail, ID, Len, Data, Sec, Microsec = LS_CANGetMessage(nHandle2, 10);
      end;
   until (false);

   -- Verbindungen beenden
   LS_CANCloseDevice(nHandle);
   LS_CANCloseDevice(nHandle2);
end;
```


© 2007-2010 Analytica GmbH

14.2. Examples for devices with SPI interface

Example 14.2.

```
--*************************************************
function printf(...)
   io.write(string.format(...))
   io.flush();
end;
--*************************************************
function main()
   -- Verbindung zu AnaGate SPI-Device herstellen
   nRC, nHandle    = LS_SPIOpenDevice( "10.1.2.162", 5000 );
   if (nRC ~= 0) then
      errortext = LS_SPIErrorMessage(nRC);
      print(errortext);
      exit();
   end

   -- Setzen der globalen Einstellungen
   nRC = LS_SPISetGlobals( nHandle, 100, 2, 0, 0 );

   -- OP-Codes des SPI-Partners mit Daten
   OPWriteEnab = {0x06};
   OPStatusReg = {0x05, 0x00};
   OPRead      = {0x03, 0x00, 0x00, 0x00};

   -- WriteEnable-Flag des SPI-Partners setzen
   nRC, Value = LS_SPIDataReq(nHandle, 1, 1, OPWriteEnab);

   -- Statusregister des SPI-Partners abfragen
   nRC, Value = LS_SPIDataReq(nHandle, 2, 2, OPStatusReg);
   for i=1, table.getn(Value), 1 do
      printf("Data Status: %02X\n", Value[i]);
   end;

   -- Lesen von 20Bytes ab Adresse 0x00
   nRC, Value = LS_SPIDataReq(nHandle, 4, 20, OPRead);
   for i=1, table.getn(Value), 1 do
      printf("Data: %02X\n", Value[i]);
   end;

   -- Alle digitalen Ausgaenge zuruecksetzen
   LS_SPIWriteDigital(nHandle, 0);

   -- Verbindung zu AnaGate SPI-Device beenden
   LS_SPICloseDevice(nHandle);
end;
```

 © 2007-2010 Analytica GmbH

14.3. Examples for devices with I2C interface

Example 14.3.

-
-

```
--**************************************************
function printf(...)
   io.write(string.format(...))
   io.flush();
end;
--**************************************************
function getn(t)
   if type(t.n) == "number" then return t.n end;
   local max = 0
   for i, _ in t do
      if type(i) == "number" and i>max then max=i end;
   end;
   return max;
end;
--**************************************************
function main()
   aSendData = {};
   for i=1, 128 , 1 do
      table.insert(aSendData, i-1 );
   end;

   nRC, nHandle    = LS_I2COpenDevice( 1000000, "10.1.2.162", 5000 );
   if (nRC ~= 0) then
      print(LS_I2CErrorMessage(nRC));
      exit();
   end;

   --Read EEPROM
   CountBytes = 1024;
   for Address = 0, CountBytes*64, CountBytes do
      nRC, Value = LS_I2CReadEEProm(nHandle, 1, Address, CountBytes, 16);
      for j=1, table.getn(Value), 1 do
         printf("%02X ", Value[j]);
         if (j%16 == 0) then
            printf("\n");
         end;
      end;
   end;

   --Write EEPROM
   CountBytes = table.getn(aSendData);
   for Address = 0, CountBytes * 10, CountBytes do
      nRC = LS_I2CWriteEEProm(nHandle, 1, Address, 16, CountBytes, aSendData);
   end;

   LS_I2CWriteDigital(nHandle, 0);
   LS_I2CCloseDevice(nHandle);
end;
```

 © 2007-2010 Analytica GmbH

Example 14.4.

-
-

```
--*************************************************
function printf(...)
   io.write(string.format(...))
   io.flush();
end;
--*************************************************
function getn(t)
   if type(t.n) == "number" then return t.n end;
   local max = 0
   for i, _ in t do
      if type(i) == "number" and i>max then max=i end;
   end;
   return max;
end;
--*************************************************
function main()
   aSendData = {};
   for i=1, 128 , 1 do
      table.insert(aSendData, i-1 );
   end;

   nRC, nHandle    = LS_I2COpenDevice( 1000000, "10.1.2.162", 5000 );
   if (nRC ~= 0) then
      print(LS_I2CErrorMessage(nRC));
      exit();
   end;

   --Write
   aData = {0x00, 0x05}; -- ab Adresse 5 lesen
   nRC, Value = LS_I2CWrite(nHandle, 0xa2, 2, aData);

   --Read
   nRC, Value = LS_I2CRead(nHandle, 0xa2, 1024);
   for i=1 , table.getn(Value), 1 do
      printf("%02X ", Value[i]);
   end;
   printf("\n");

   LS_I2CWriteDigital(nHandle, 0);
   LS_I2CCloseDevice(nHandle);
end;
```

 © 2007-2010 Analytica GmbH

Example 14.5.

•

```
--**************************************************
function printf(...)
   io.write(string.format(...))
   io.flush();
end;
--**************************************************
function getn(t)
   if type(t.n) == "number" then return t.n end;
   local max = 0
   for i, _ in t do
      if type(i) == "number" and i>max then max=i end;
   end;
   return max;
end;
--**************************************************
function main()
   aSendData = {};
   for i=1, 128 , 1 do
      table.insert(aSendData, i-1 );
   end;

   nRC, nHandle    = LS_I2COpenDevice( 1000000, "10.1.2.162", 5000 );
   if (nRC ~= 0) then
      print(LS_I2CErrorMessage(nRC));
      exit();
   end;

   --Sequence
   aData = {0xa2, 0x00,  --SLA
            0x02, 0x00,  --Laenge Schreibkommando
            0x00, 0x00,  --Daten Schreibkommando
            0xa3, 0x00,  --SLA 2. Lesekommando
            0x30, 0x00,  --Laenge 1. Lesekommando
            0xa3, 0x00,  --SLA 2. Lesekommando
            0x20, 0x00}; --Laenge 2. Lesekommando

   nRC,CountRead,LastError,Value = LS_I2CSequence(nHandle, 0x0E, 0x0050, aData);
   printf("CountRead:%02X  LastError:%02X\n", CountRead, LastError);
   for i=1, CountRead, 1 do
      printf("%02X ", Value[i]);
   end;
   printf("\n");

   LS_I2CWriteDigital(nHandle, 0);
   LS_I2CCloseDevice(nHandle);
end;
```


© 2007-2010 Analytica GmbH

Appendix A. API return codes

Followed a list of the return values of the API functions. This values are defined in the header file `AnaGateErrors.h`.

Table A.1. Common return values for all devices of AnaGate series

Value	Name	Description
0	ERR_NONE	No errors.
0x000001	ERR_OPEN_MAX_CONN	Open failed, maximal count of connections reached.
0x0000FF	ERR_OP_CMD_FAILED	Command failed with unknown failure.
0x020000	ERR_TCPIP_SOCKET	Socket error in TCP/IP layer occured.
0x030000	ERR_TCPIP_NOTCONNECTED	Connection to TCP/IP partner can't established or is disconnected.
0x040000	ERR_TCPIP_TIMEOUT	No answer received from TCP/IP partner within the defined timeout.
0x050000	ERR_TCPIP_CALLNOTALLOWED	Command is not allowed at this time.
0x060000	ERR_TCPIP_NOT_INITIALIZED	TCP/IP-Stack can't be initialized.
0x0A0000	ERR_INVALID_CRC	AnaGate TCP/IP telegram has incorrect checksum (CRC).
0x0B0000	ERR_INVALID_CONF	AnaGate TCP/IP telegram wasn't receipted from partner.
0x0C0000	ERR_INVALID_CONF_DATA	AnaGate TCP/IP telegram wasn't receipted correct from partner.
0x900000	ERR_INVALID_DEVICE_HANDLE	Invalid device handle.
0x910000	ERR_INVALID_DEVICE_TYPE	Function can't be executed on this device handle, as she is assigned to another device type of AnaGate series.

Table A.2. Return values for AnaGate I2C

Value	Name	Description
0x000120	ERR_I2C_NACK	I2C-NACK
0x000121	ERR_I2C_TIMEOUT	I2C Timeout

A textual description of the return value can be retrieved with the function `I2CErrorMessage()`.

© 2007-2010 Analytica GmbH

Table A.3. Return values for AnaGate CAN

Value	Name	Description
0x000220	ERR_CAN_NACK	CAN-NACK
0x000221	ERR_CAN_TX_ERROR	CAN Transmit Error
0x000222	ERR_CAN_TX_BUF_OVERLOW	CAN buffer overflow
0x000223	ERR_CAN_TX_MLOA	CAN Lost Arbitration
0x000224	ERR_CAN_NO_VALID_BAUDRATE	CAN Setting no valid Baudrate

A textual description of the return value can be retrieved with the function `CANErrorMessage()`.

Table A.4. Return values for AnaGate Renesas

Value	Name	Description
0x000920	ERR_RENESAS_TIMEOUT	Renesas timeout
0x000921	ERR_RENESAS_INVALID_ID	Renesas Invalid ID
0x000922	ERR_RENESAS_FLASH_ERASE_FAILED	Renesas failed erase the flash
0x000923	ERR_RENESAS_PAGE_PROG_FAILED	Renesas failed prog the page

A textual description of the return value can be retrieved with the function `RenesasErrorMessage()`.

Table A.5. Return values for LUA scripting

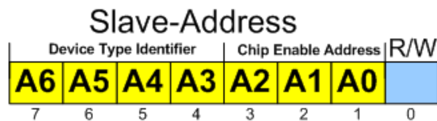
Value	Name	Description
-1	ERR_SYNTAX	Syntax error
-2	ERR_RANGE	Value out of valid range.
-3	ERR_NOT_A_NUMBER	Parameter is not of type *number*.
-4	ERR_NOT_A_STRING	Parameter is not of type *string*.
-5	ERR_NOT_A_BOOL	Parameter is not of type *boolean*.
-6	ERR_NOT_A_TABLE	Parameter is not of type *table*.
-10	ERR_NO_DATA	No data available.

© 2007-2010 Analytica GmbH

Appendix B. I2C slave address formats

A standard I2C address is the first byte sent be the I2C master, whereas only the first seven bit form the adress, the last bit (R/W-bit) defines the direction in which the data is sent. I2C has a 7-bit address space and can address 112 slaves on a single bus (16 of the 128 addresses are reserved fo special purposes).

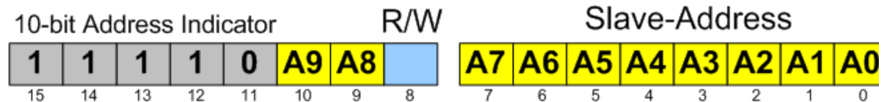
Figure B.1. Definition of a I2C slave address in 7-bit format



Each I2C-able IC has a determined bus address. The 4 upper bits of the bus address are called *Device Type Identifier* and define the chip type. The lowest three bits called sub-address or *Chip Enable Address* are usually defined by the corresponding wired control pins. So, in total up to 8 similar IC's can be used on a single I2C bus.

Because of a lack of address space a 10-bit addressing mode was introduced later. This new mode is downwards compatible to the 7-bit standard mode through usage of 4 of the 16 reserved addresses. Both addressing modes can be used simultaneously, which implies that 1136 slaves can be used on a single bus.

Figure B.2. Definition of a I2C slave address in 10-bit format



Note

Devices of type *AnaGate SPI* and *AnaGate Universal Programmer* do support both addressing modes in general. The API functions `I2CRead` and `I2CWrite` address the slaves via a two byte parameter.

Addressing of serial EEPROM

The device type identifier of a serial EEPROM is defined as `0xA`. This results to the following schematic structure of an address (the chip enable bits are often named E0, E1 and E2 in literature):

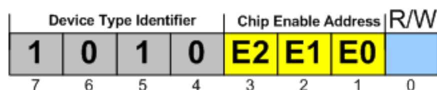


Table B.1. I2C EEPROM addressing examples

	Device Type Identifier				Chip Enable [1] [2]			R/W	EEPROM-Memory
	b7	b6	b5	b4	b3	b2	b1	b0	
M24C01	1	0	1	0	E2	E1	E0	R/W	128 byte

© 2007-2010 Analytica GmbH

	Device Type Identifier				**Chip Enable** [1 2]			**R/W**	**EEPROM-Memory**
	b7	**b6**	**b5**	**b4**	**b3**	**b2**	**b1**	**b0**	
M24C02	1	0	1	0	E2	E1	E0	R/W	256 byte
M24C04	1	0	1	0	E2	E1	A8	R/W	512 byte
M24C08	1	0	1	0	E2	A9	A8	R/W	1024 byte
M24C16	1	0	1	0	A10	A9	A8	R/W	2048 byte
M24C64	1	0	1	0	E2	E1	E0	R/W	8192 byte

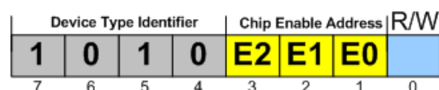
[1]E0,E1 and E2 are compared against the respective external pins on the memory device.
[2]A10, A9 and A8 represent most significant bits of the address.

 © 2007-2010 Analytica GmbH

Appendix C. Programming I2C EEPROM

The *AnaGate I2C* and the *AnaGate Universal Programmer* is very well suited for programming serial I2C EEPROM. To support this special requirement two different API functions are made available: `I2CReadEEPROM` and `I2CWriteEEPROM`.

Like all other I2C-capable devices EEPROM's are addressable on the I2C bus via a unique slave address (see also Appendix B, *I2C slave address formats*). The so-called *Device Type Identifier* for these types of devcies is `0xA`. In principle 8 similar devices can be connected and addressed via the *Chip Enable Bits* E0, E1 und E0.



A data transmission is started with a **Start** signal by the master, followed by the slave address. The slave address is confirmed by the slave with a **ACK**. Depending on the R/W bit data is written (data to slave) or read (data from slave). The last byte of a read access has to be confirmed with a **NAK** by the master to signal the slave end of read transmission. The data transmission is terminated always by a **Stop** signal from the master.

When using EEPROM's the memory address is transmitted after transmission of the slave address, to advice the slave which memory address is to be written or read. Depending on the used EEPROM type the memory address is sent as single byte (8 bit) or as two bytes (16 bit, MSB First).

To expand the address space from 8 bit (or 16 bit), some EEPROM types use the *Chip Enable Bits* E0, E1, E2 as additional address bits. Which bits are used in individual cases is defined by the chip producer. In the following, all possible combinations of the bits usage are listed:

Table C.1. Usage of the CHIP-Enable Bits of I2C EEPROMs

Mode [1]	**Usage**	**Description**
0x0	E2-E1-E0	Bits are only used to select the chip.
0x1	E2-E1-A0	Bit E0 is used to expand the addresse space. It is used for address bit A8 (resp. A16).
0x2	E2-A1-A0	E0 and E1 are used to expand the addresse space. E0 is used for address bit A8 (resp. A16) and E1 is used for A9 (resp. A17).
0x3	A2-A1-A0	E0, E1 and E2 are used to expand the addresse space. E0 is used for address bit A8 (resp. A16), E1 for A9 (resp. A17) and E2 for A10 (resp. A18).
0x5	A0-E1-E0	Bit E2 is used to expand the addresse space. It is used for address bit A8 (resp. A16). Das E2-Bit wird für die Addressierung verwendet. Es entspricht dabei dem Adressbit A8 bzw. A16.

© 2007-2010 Analytica GmbH

Mode [1]	**Usage**	**Description**
0x6	A1-A0-E0	E2 and E1 are used to expand the addresse space. E1 is used for address bit A8 (resp. A16) and E2 is used for A9 (resp. A17).

[1]Set this mode flag in bit 8-10 of parameter *nOffsetFormat* in the API funtions `I2CReadEEPROM` and `I2CWriteEEPROM`.

© 2007-2010 Analytica GmbH

Appendix D. FAQ - Frequent asked questions

Here is a list of frequently asked questions concerning installation and usage of the *AnaGate* product.

D.1. Common questions

Q: No network connection (1)

A: Please check first the physical connection to the device. Basically the *AnaGate* have to be connected directly to a personal computer or to an active network component (hub, switch). If the AnaGate device is connected to a personal computer a cross-wired network cable must be used to connect the device, otherwise the included network cable is to be used.

The physical interconnection is ok, if the yellow link LED changes to light on, if LAN cable is plug in. The yellow light keeps beeing on until the connection break down. On some hardware models the link LED flickers synchronous to the green activity LED if there is traffic on the network line.

If the link LED is always off, then please check the wiring between the *AnaGate* and the hub, switch or the personal computer.

Q: No network connection (2)

A: If the link LED indicates a proper ethernet connection (see previous FAQ), but you still can't connect to the *AnaGate* then please try the following:

1. Check if the *AnaGate* can be reached via ping. To do so in Windows, open a command prompt and enter the command **ping a.b.c.d**", where `a.b.c.d` is the device IP address.
2. In case the *AnaGate* is unreachable via ping, reset the device to factory settings. Set the IP adress of your PC to `192.168.1.253` and the subnet mask to `255.255.255.0`. Check if the *AnaGate* can be reached via **ping 192.168.1.254**.
3. If the device can be reached via ping then the next step is to try if you can open a TCP connection to port 5001. Open a Windows command prompt and enter **telnet a.b.c.d 5001**, where `a.b.c.d` is the device IP address. If this command fails, check if a firewall runs on your PC or if there is a packet filter in the network between your PC and the *AnaGate*.

© 2007-2010 Analytica GmbH

Q: No network connection after changing the network address

A: After changing the network address of the AnaGate device via web interface, the device is not longer reachable. The used internet browsers displays only an emtpy web side, additional error messages are not available.

Please check if your anti-virus software has blocked the new network address. After changing the network address, you are redirected to the new network address in the browser. Such activity is suspicious for some anti-virus software, so they block the new webside, sometimes even without notification of the user.

Q: Connection problems using multiple devices

A: If multiple devices with identical IP addresses are used in a local area network at the same time, the connections to the devices are not stable. Because of this behaviour it is recommend to use different IP addresses.

This problem can also occur, if devices with identical IP addresses are used not concurrently, but within short intervals. For example this can arise, if some new devices, which have the default IP address `192.168.1.254`, are configured from a single PC.

The **Address Resolution Protocol (ARP)** is used by IP4 networks to determine the MAC address of a given IP address. The neccessary information is cached in the *ARP table*. If there is a wrong entry in the ARP table or even an entry, which is not up-to-date, it is not possible to communicate with the corresponding host.

An entry in the ARP table is deleted if it is not used any more after a short period time. The time intervall used depends on the operating system. On a current linux distribution an unused entry is discarded after about 5 minutes. The ARP cache can be displayed and manipulated with the **arp** on windows and linux.

```
C:\>arp -a

Schnittstelle: 10.1.2.50 --- 0x2
  Internetadresse       Physikal. Adresse     Typ
  192.168.1.254         00-50-c2-3c-b0-df     dynamisch
```

The command **arp -d** can be used to delete the *ARP Cache*.

Note

Maybe the *ARP cache* of the PC has to be deleted, if the IP address of a device is changed.

Q: Using a firewall

A: When working with a firewall, the a TCP port has to be opened for communication with the AnaGate device:

Table D.1. Using AnaGate hardware with firewall

Device	Port number
AnaGate I2C	5000

 © 2007-2010 Analytica GmbH

Device	Port number
AnaGate I2C X7	5100, 5200, 5300, 5400, 5500, 5600, 5700
AnaGate CAN	5001
AnaGate CAN USB	5001
AnaGate CAN uno	5001
AnaGate CAN duo	5001, 5101
AnaGate CAN quattro	5001, 5101, 5201, 5301
AnaGate SPI	5002
AnaGate Renesas	5008
AnaGate Universal Programmer	5000, 5002, 5008

D.2. Questions concerning AnaGate CAN

Q: What is the value of the termination resistor when the termination option of the device is activated?

A: The termination resistor of the *AnaGate* is driven by an FET transistor. The resistor itself has 110 Ohm while the internal resistance of the FET is 10 Ohm if the FET is activated. So the resulting resitance is 120 Ohm, as required by the CAN bus.

Q: Does Analytica offer a CAN gateway which does not have an galvanically isolated CAN interface?

A: Any device that is actively connected to a CAN bus should be galvanically isolated. Especially when using USB-operated devices (like the *AnaGate USB*), it is essential to have an galvanically isolated device, because the device is power supplied by the PC.

Q: How to direct interconnect two CAN ports!

A: If you want to interconnect two *AnaGate CAN* just via a direkt link CAN cable, you have to switch on the internal termination on both *AnaGate CAN* devices. A CAN bus network must have a termination on each side.

Note

Maybe it is working with lower baurates without termination, but it is recommend to use a termination.

Q: Receiving a NAK when sending a CAN telegram.

A: If no CAN partner is connected to the *AnaGate CAN* (aka the CAN network), it is not possible to send CAN telegrams, the *AnaGate CAN* gets a NAK from the CAN controller. These NAK errors are send to the AnaGate client via a data confirmation telegram.

Warning

If data confirmations are switched off, no erros are sent to the client. The option *confirmations for data requeste* can be set via the

 © 2007-2010 Analytica GmbH

`CANSetGlobals` function. In Highspeed-Mode the data confirmations are always switched off.

D.3. Questions concerning AnaGate I2C

Q: What is the correct order to connect the GND / SCL and SDA when using an external power supply?

A: To avoid potential damage to the *AnaGate I2C*, the GND pin MUST be connected to the application board first. Only then can the SCL/SDA pins be allowed to make contact with the application board.

© 2007-2010 Analytica GmbH

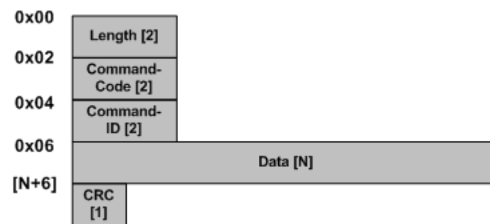
Appendix E. FAQ - Programming API

Here is a list of frequently asked questions concerning the programming API and the communication protocol.

E.1. Questions concerning the communication protocol

Q: The calculation of the check sum (CRC) do not work!

A: The following figure illustrates the princible layout of an *AnaGate* telegram.



The checksum is defined as a byte calculated by XOR from all the existing bytes in an AnaGate telegram, excluding the length bytes and the CRC byte.

The following C code function computes a valid CRC of an already created command telegram.

```
unsigned char CalcCRC( char * pBuffer, int nBufferLength )
{
   int i;
   unsigned char nCRC = pBuffer[2]; // skip the length bytes

   // XOR over all bytes in the message except the length information and the last byte
   for( i = 3; i < nBufferLength -1; i++ )
   {
      nCRC ^= pBuffer[i];
   }
   return nCRC;
}
```

When using the function `CalcCRC` the parameter *`pBuffer`* must point to the data buffer, which contains the already created complete data telegram. The length parameter *`nBufferLength`* depends on the created command type and can be computed as shown below:

```
buffer length = sizeof( command length ) + sizeof( command code )
                 + sizeof( command id ) + sizeof( CRC ) + sizeof(data)
              = 7 + sizeof(data)
```

 © 2007-2010 Analytica GmbH

Appendix F. Technical support

The AnaGate hardware series, software tools and all existing programming interfaces are developed and supported by Analytica GmbH. Technical support can be requested as follows:

Internet

The AnaGate web site [http://www.anagate.de/en/index.html] of Analytica GmbH contains information and software downloads for AnaGate Library users:

- Product updates featuring bug fixes or new features are available here free of charge.

EMail

If you require technical assistance over the Internet, please send an e-mail to

<support@anagate.de>

To help us provide you with the best possible support, please keep the following information and details at hand when you contact our Support Team.

- Version number of the used programming tool or AnaGate library
- AnaGate hardware series model and firmware version
- Name and version of the operating system you are using

© 2007-2010 Analytica GmbH

Bibliography

Books

[LuaRef2006-EN] Roberto Ierusalimschy, Luiz Henrique Figueiredo, and Waldemar Celes. Copyright © 2006 R. Ierusalimschy, L. H. de Figueiredo, W. Celes. Isbn 85-903798-3-3. Lua.org. *Lua 5.1 Reference Manual*.

[LuaProg2006-EN] Roberto Ierusalimschy. Copyright © 2006 Roberto Ierusalimschy, Rio de Janeiro. Isbn 85-903798-2-5. Lua.org. *Programming in LUA (second edition)*.

Other publications

[NXP-I2C] NXP Semiconductors. Copyright © 2007 NXP Semiconductors. *UM10204*. I2C-bus specification and user manual. Rev. 03. 19.06.2007.

[TCP-2010] Analytica GmbH. Copyright © 2010 Analytica GmbH. *Manual TCP-IP communication*. Version 1.2.6. 15.05.2008.

[Prog-2010] Analytica GmbH. Copyright © 2010 Analytica GmbH. *AnaGate API*. Programmer's Manual. Version 1.4. 01.10.2010.

[CiA-DS301] Copyright © 2002 CAN in Automation (CiA) e. V.. CAN in Automation (CiA) e.V.. 13.02.2002. *Cia 301, CANopen Application Layer and Communication Profile*.

 © 2007-2010 Analytica GmbH