The **RKS+CAN CAN-Bus Interface** offers **two methods to access it** when you write own programs using it:

- **C/C++ interface via header file and RKS-USB.DLL**
- **ASCII interface (textual interface)**

You can use only one method at a time. On the following pages, both interfaces are described.

For further documentation and examples please check
https://www.canhack.de/viewforum.php?f=26

RKS+CAN
C/C++ Interface Description
using RKS-USB.DLL

```
// RKS-USB.h : main header file for the RKS-USB DLL
//

#pragma once

#ifndef __AFXWIN_H__
   #error "include 'stdafx.h' before including this file for PCH"
#endif


// frame types
#define FRAME_TYPE_NORMAL     0x1
#define FRAME_TYPE_RTR        0x2
#define FRAME_TYPE_NORMAL_EXT 0x3
#define FRAME_TYPE_RTR_EXT    0x4
#define FRAME_TYPE_ERR        0x5

// CAN data structure
typedef struct
{
```

```
  DWORD dwID;
  BYTE byDLC;
  BYTE abyData[8]; // max 8 data bytes
} can_data_t;

// CAN error information structure
typedef struct
{
  BYTE byError;
} can_err_t;

// CAN information structure
typedef union
{
  can_data_t sData;
  can_err_t sErr;
} can_union_t;

// The final CAN frame structure used for send/receive
typedef struct
{
  BYTE byType;
  DWORD dwTimeStamp;
  can_union_t uFrm;
} can_msg_t;

// DLL exports

// Initialize USB driver, open connection to RKS+CAN hardware. Returns TRUE on
success.
//  This command must be done once before calling any other RKS... function, except
//  RKSDeviceConnected.
extern "C" __declspec(dllexport) BOOL RKSInitialize(void);

// Free USB driver and RKS+CAN hardware. Should be called to release interface/driver.

extern "C" __declspec(dllexport) void RKSFree(void);

// Check if the RKS+CAN is connected to the computer. Returns TRUE on success.
pcBufIfGUID can be NULL.
//  If pcBufIfGUID is not NULL, copy the DeviceInterfaceGUID of the RKS+CAN hardware
to it.
//  If you check simply if the RKS+CAN is connected, it is recommended to call it with
pcBufIfGUID = NULL.
extern "C" __declspec(dllexport) BOOL RKSDeviceConnected(LPSTR pcBufIfGUID, DWORD
dwBufSize);

// Set the timeouts for reading / writing data to the RKS+CAN interface. The software
waits at maximum the
//  specified values for read/write operations to complete. Returns TRUE on success.
extern "C" __declspec(dllexport) BOOL RKSSetTimeouts(DWORD dwMsTimeoutRead, DWORD
dwMsTimeoutWrite);

// Direct read from hardware USB pipes (low level). Not recommended for application
use if you do not know what you do.
extern "C" __declspec(dllexport) BOOL RKSReadPipe(PUCHAR pucBuffer, DWORD
dwBufferLength, DWORD* pdwLengthTransferred, LPOVERLAPPED pOverlapped);

// Direct write to hardware USB pipes (low level). Not recommended for application use
if you do not know what you do.
extern "C" __declspec(dllexport) BOOL RKSWritePipe(PUCHAR pucBuffer, DWORD
dwBufferLength, DWORD* pdwLengthTransferred, LPOVERLAPPED pOverlapped);


// Read from hardware using the specified timeout value as maximum time to finish the
operation. Returns TRUE on success.
extern "C" __declspec(dllexport) BOOL RKSRead(PUCHAR pucBuffer, DWORD dwBufferLength,
DWORD* pdwLengthTransferred);

// Write to hardware using the specified timeout value as maximum time to finish the
operation. Returns TRUE on success.
```

```
extern "C" __declspec(dllexport) BOOL RKSWrite(PUCHAR pucBuffer, DWORD dwBufferLength,
DWORD* pdwLengthTransferred);

// Get the version of the of the RKS+CAN hardware. Returns TRUE on success.
extern "C" __declspec(dllexport) BOOL RKSGetVersion(PUCHAR pucBuffer, DWORD
dwBufferLength);

// Get the serial number of the RKS+CAN hardware (printed on the cable case).  Returns
TRUE on success.
extern "C" __declspec(dllexport) BOOL RKSGetSerial(PUCHAR pucBuffer, DWORD
dwBufferLength);

// Return time since RKS-USB driver initialisation. bReInit can be used to reset the
time to zero.
//  Can be used to get relatively exact time information, the time is returned in
seconds.
extern "C" __declspec(dllexport) double RKSGetTimeSinceInit(BOOL bReInit = FALSE);

// Get the error status of the RKS+CAN interface in pbyStatus. Returns TRUE on
success.
extern "C" __declspec(dllexport) BOOL RKSCANGetLastStatus(BYTE* pbyStatus);

// Set RKS+CAN hardware to listen only mode. Returns TRUE on success.
extern "C" __declspec(dllexport) BOOL RKSCANSetListenOnly(BOOL bEnabled);

// Set time stamp mode of the RKS+CAN hardware. E.g. 0, 1 or 2.
//  For possible values check the ASCII interface description. Returns TRUE on
success.
extern "C" __declspec(dllexport) BOOL RKSCANSetTimeStamp(BYTE byMode);

// Get time stamp mode. Returns TRUE on success.
extern "C" __declspec(dllexport) BOOL RKSCANGetTimeStamp(BYTE*pbyMode);

// Get the supply voltage of the RKS+CAN hardware. Returns TRUE on success.
extern "C" __declspec(dllexport) BOOL RKSCANGetUb(DWORD* pdwVoltage_mV);

// Set CAN hardware filtering of the RKS+CAN hardware. Returns TRUE on success.
extern "C" __declspec(dllexport) BOOL RKSCANSetFilter(DWORD dwCode, DWORD dwMask);

// Open CAN bus with the desired bitrate. Returns TRUE on success.
extern "C" __declspec(dllexport) BOOL RKSCANOpen(DWORD dwBitrate);

// Close CAN bus. Returns TRUE on success.
extern "C" __declspec(dllexport) BOOL RKSCANClose(void);

// Get one CAN message from the receive queue, RKSCANOpen(...) must have
//  been called before. Returns TRUE on success.
extern "C" __declspec(dllexport) BOOL RKSCANRx(can_msg_t* pMsg);

// Add one CAN message to the send queue, RKSCANOpen(...) must have
//  been called before. Returns TRUE on success.
extern "C" __declspec(dllexport) BOOL RKSCANTx(can_msg_t* pMsg);
```

In your development environment, keep sure that RKS-USB.DLL can be found by the linker.

You can find example projects at:
https://www.canhack.de/viewtopic.php?t=137

RKS+CAN
ASCII-Interface Description

The RKS+CAN interface allows to be used as a virtual serial port with a simple character (ASCII) based interface. A command ends with [CR], the reply of the interface is … [CR] (0x0C) on success or [BELL] (0x07) on failure.

Supported commands:

1. Setup CAN bitrate,short form:
   Sn[CR] where n ∈ { 0, 1, 2, 3, 4, 5, 6, 7, 8 }
   0 = 10kBit/s
   1 = 20kBit/s
   2 = 50kBit/s
   3 = 100kBit/s
   4 = 125kBit/s
   5 = 250kBit/s
   6 = 500kBit/s
   7 = 800kBit/s
   8 = 1000kBit/s

   The command can only be sent when the CAN channel is closed.
   Example: S6[CR] to setup the CAN for 500kBit/s

2. Setup CAN bitrate, SJA 1000 form:
   snnmm[CR] where nn = BTR0 and mm = BTR1 hexadecimal values for a 16MHz clock.
   The command can only be sent when the CAN channel is closed.

   The RKS+CAN hardware does not use a SJA1000 hardware but provides this interface for compatibility purposes. Example: s0714[CR] for 250kBit/s. For details how to calculate the values, see SJA 1000 datasheet or Google/MSN.
3. Setup CAN bitrate, easy form:
   synnnnn[CR] where nnnnn = hexadecimal value of the bitrate.

   The command can only be sent when the CAN channel is closed.
   Example: sy14585[CR] for 83,333kBit/s.

4. Open CAN channel:
   O[CR], if no bitrate has been set before, it defaults to 500kBit/s.

5. Close CAN channel:
   C[CR], close previously opened CAN channel.

6. Get interface serial, compatibility form:
   N[CR], returns the compatibility form of the interface serial number e.g. N1234[CR].

7. Get interface version numbering, compatibility form:
   V[CR], returns the compatibility form of the interface software and hardware versions, e.g. V1002[CR].

8. Get RKS-CAN interface serial:
   x[CR], returns full interface serial, e.g. x7641123456789[CR].

9. Read error flags:
   F[CR], returns Fxx where xx is a 8-bit hexadecimal number representing the error flags.

Bit	Represents
1	CAN receive FIFO queue full
2	CAN transmit FIFO queue full
3	Error warning (EI), see SJA 1000 datasheet
4	Data Overrun (DOI), see SJA 1000 datasheet
5	unused
6	Error Passive (EPI), see SJA 1000 datasheet
7	Arbitration Lost (ALI), see SJA 1000 datasheet
8	Bus Error (BEI), see SJA 1000 datasheet

   Example: F01[CR] if CAN receive FIFO is full. This function is only available, when the CAN is open.

10. Send 11 bit CAN frame:
    tiiildddddddddddddddd[CR], sends a CAN frame with hexadecimal ID iii, length l and data bytes dd (according to length). Example: t280411223344[CR] sends 4 bytes, CAN ID 0x280, bytes 0x11, 0x22, 0x33, 0x44. This function is only available, when the CAN is open.

11. Send 29 bit CAN frame:
    Tiiiiiiiildddddddddddddddd[CR], sends a CAN frame with hexadecimal ID iiiiiiii, length l and data bytes dd (according to length). Example: T0000E3883223344[CR] sends 3 bytes, CAN ID 0xE388, bytes 0x22, 0x33, 0x44. This function is only available, when the CAN is open.

12. Send 11 bit CAN RTR frame:
    riiil[CR], sends a CAN frame with hexadecimal ID iii and length l (normally zero). Example: r2880[CR] sends RTR (Remote Transmission Request) message for CAN ID 0x288. This function is only available, when the CAN is open.

13. Send 29 bit CAN RTR frame:
    Riiiiiiiil[CR], sends a CAN frame with hexadecimal ID iiiiiiii, length l. Example: R000E3880 sends RTR message for CAN ID 0xE388. This function is only available, when the CAN is open.

14. Receive 11 or 29 bit CAN frames:
    When the CAN channel is open and there is traffic on the bus, the received CAN messages are reported in the same form as they are sent. See commands 10 and 11. If hardware timestamps are activated, they are added on the end of the message.

15. Hardware timestamps command:
    Zn[CR] where n = 0 means hardware timestamps are off, n = 1 means timestamps on, measured in milliseconds going from 0 to 60000 ms. If n = 2, hardware timestamps in 10 us units are sent, going from 0 to 600 ms. The timestamps are cat on the end of received messages as two hexadecimal bytes in ASCII form.

16. Set acceptance code register, SJA 1000 compatibility:
    Mxxxxxxxx[CR] allows to set up CAN message filtering according to SJA 1000 datasheet. This function is only available, when the CAN is closed.

17. Set acceptance mask register, SJA 1000 compatibility:
    mxxxxxxxx[CR] allows to set up CAN message filtering according to SJA 1000 datasheet. This function is only available, when the CAN is closed.

18. Set listen only mode:
    Lx[CR] enables or disables listen only mode. In listen only mode, the CAN controller does not acknowledge received messages. Because of this, listen only mode does only make sense if there are at least two CAN nodes connected to a bus.
    Listening to only one CAN controller results in error frames as the messages are never acknowledged!

    The command can only be sent when the CAN channel is closed.
    Example: L1[CR] enables listen only mode, L0[CR] disables it.

19. Get Ub voltage:
    U[CR] measures the voltage on the OBD2 port, it returns Uxxx[CR] where xxx is the hexadecimal value of the voltage in 10mV units. E.g. U561[CR] is a measurement of 13770mV.

You find further info for using the system at the following internet addresses:

http://www.canhack.de

http://www.canhack.de/viewtopic.php?t=137

http://shop.dieselschrauber.org/can-interface-kit-p-313.php

Kaufmann Automotive GmbH
Süsswinkelstrasse 9
CH-9453 Eichberg
http://www.kaufmann-automotive.ch