KK_FX80E.DLL / libkk_fx80e.so

K+K Library for Windows and Linux PCs

Manual

Version 16.03, 2016-11-16

Beginning with version 15.00 there exist three variants of this Library:

KK_FX80E.dll: generated under Delphi for Win32
KK_FX80E.dll: generated under Lazarus for Win32
libkk_fx80e.so: generated under Lazarus for Linux

K+K Messtechnik GmbH, St.-Wendel-Str. 12, D-38116 Braunschweig

Tel:+49(0)531/501436, e-mail:kplusk@t-online.de

Revision History:

When	Who	Version	What
2013-10-02	Loryn	15.00	first version with this manual
2013-10-21	Loryn	15.01	added KK_GetDefaultStorageRoot to Local-RPC-Server
			Local-RPC-Server routines: FX_ renamed to KK_
2013-10-29	Loryn	15.01	Linux serial port: added example for FX_OpenPort, required membership in group dialout
2013-11-01	Loryn	15.02	LabView routines reworked, added LV_EnumerateDevices
2013-11-11	Loryn	15.03	Windows-Lazarus-DLL supporting USB Linux-DLL: added USB CDC support; deleted LV_ClosePort
2013-11-26	Loryn	15.03	Linux-DLL: added USB HID support; all three DLL variants have identical scope of operation
2014-03-27	Loryn	15.06	added FX_DebugLogLimit
2014-06-03	Loryn	15.08	added FX_OpenPortBlockingIO, LE_ReadMonPha
2014-07-02	Loryn	15.10	added routines for generation of test data
2015-03-13	Loryn	15.11	added user ID to FX_OpenPort
		15.12	added CAN connectivity to FX_OpenPort
2016-09-15	Loryn	16.00	added: TCP-Server, TCP Device in FX_OpenPort
2016-09-19	Loryn	16.00	extended description for test data and tools
2016-10-13	Loryn	16.01	added FX_OpenPort: Hint for CAN device added: KK_GetHostAndIP
2016-11-21	Loryn	16.02	do NOT use data type string as a parameter! (KK_TcpReportLog)
2016-11-21	Loryn	16.03	TCP-Server: Mode specification for DLL format deleted replaced KK_GetHostAndIP by KK_GetHostAndIP**s**

List of Contents

K+K Messtechnik GmbH, St-Wendel-Str. 12, D-38116 Braunschweig, Tel: +49(0)531/501436, kplusk@t-online.de

K+K Messtechnik GmbH, St-Wendel-Str. 12, D-38116 Braunschweig, Tel: +49(0)531/501436, kplusk@t-online.de

1. List of exported functions

All three variants (Delphi for Windows, Lazarus for Windows, Lazarus for Linux) export the same functions:

1.1 Enumerating available Interfaces

```
KK_EnumerateDevices(var Names: PChar; EnumFlags: Byte=$FF): Integer; stdcall;
KK_GetEnumerateDevicesErrorMsg: PChar; stdcall;
KK_GetHostAndIPs(HostName: PChar; IPaddr: PChar; ErrorMsg: PChar): Integer;
                                                                    stdcall;
```

1.2 Standard functions

```
FX_OpenPort(var PortBaud: PChar): Integer; stdcall;
FX_OpenPortBlockingIO(var PortBaud: PChar): Integer; stdcall;
FX_ClosePort; stdcall;
FX_GetReport(var Data: PChar): Integer; stdcall;
FX_SendCommand(var Command: PChar): Integer; stdcall;

FX_Debug(const DbgStatus: Boolean); stdcall;
FX_Log(const DbgNummer: Integer); stdcall;
FX_DebugFlags(const ReportLog: Boolean; const LowLevelLog: Boolean); stdcall;
FX_DebugLogLimit(LogType: Byte; aSize: Cardinal=0); stdcall;

KK_GetDLLVersion: PChar; stdcall;
```

1.3 Special functions for LabView programs

```
LV_EnumerateDevices(Names: PChar; EnumFlags: Byte): Integer; stdcall;
LV_OpenPort(PortBaud: PChar): Integer; stdcall;
LV_ClosePort; stdcall;
LV_GetReport(Data: PChar): Integer; stdcall;
LV_SendCommand(Command: PChar): Integer; stdcall;
LV_OpenTcpLog(IpPort: PChar; Mode: PChar): Integer; stdcall;
LV_GetTcpLog(Data: PChar): Integer; stdcall;
```

1.4 Custom functions

```
FX_GetFXM(var Data: PChar): Integer; stdcall;
LE_ReadMonitor(var Data: PChar): Integer; stdcall;
LE_ReadMonPha(var Data: PChar): Integer; stdcall;
```

1.5 Local RPC server

KK_StartRpcServer(var aPort: Word; var rootPath: PChar): Boolean; stdcall;
KK_StopRpcServer: Boolean; stdcall;
KK_TakeWarning(Warning: PChar); stdcall;
KK_EnableStoraging; stdcall;
KK_GetDefaultStorageRoot: PChar; stdcall;

1.6 Remote RPC server

KK_GetArchiveData(var aRange: TDateTimeRange; var Data: PChar;
var len: LongWord): Integer; stdcall;

1.7 Generating test data

StartSaveBinaryData(Flag: Boolean): Boolean; stdcall;
StopSaveBinaryData: Boolean; stdcall;
StartSaveReportData; stdcall;
StopSaveReportData: Boolean; stdcall;

1.8 Tools

KK_OpenDevice(var DeviceName: PChar; aWatchList: TCFWatchList = nil)
Integer; stdcall;
KK_ReadDevice(len: Byte; var Bytes: PChar; var ErrorMsg: PChar): Integer; stdcall;
KK_WriteDevice(Cmd: PChar; len: Byte; var ErrorMsg: PChar): Integer; stdcall;

1.9 Local TCP-Server

KK_StartTcpServer(var aPort: Word): Boolean; stdcall;
KK_StopTcpServer: Boolean; stdcall;
KK_GetTcpServerError: PChar; stdcall;
KK_TcpReportLog(Data: PChar; logType: Integer); stdcall;

1.10 TCP client

KK_OpenTcpLog(var IpPort: PChar; Mode: PChar): Integer; stdcall;
KK_CloseTcpLog; stdcall;
KK_GetTcpLog(var Data: PChar): Integer; stdcall;

2. Function Description

All function names, calling parameters and return values are the same for all DLL variants (Delphi for Windows, Lazarus for Windows, Lazarus for Linux).

2.1 Enumerating available Interfaces

2.1.1 ***function KK_EnumerateDevices(var Names: PChar; EnumFlags: Byte = $FF): Integer; stdcall;***

Returns in *Names* a comma separated list of interfaces / K+K devices according to the selection given by *EnumFlags*.
Names=nil if no interfaces / devices were found.

Result = 0 : no error
<>0 : Error(s) analog *EnumFlags*: an error occurred when listing the respective group of interfaces / devices.

In case of an error, the corresponding error message(s) can be obtained by calling *KK_GetEnumerateDevicesErrorMsg* .

2.1.2 ***function KK_GetEnumerateDevicesErrorMsg: PChar; stdcall;***

The result is the error message caused by calling *KK_EnumerateDevices* .

2.1.3 ***EnumFlags***

Name	Value	Description
EnumFlag_ComPorts	*$01*	Lists all serial ports locally available at the PC (this includes virtual COM ports via USB adapters)
EnumFlag_USB	*$02*	Lists all K+K USB devices connected to the PC (HID and CDC modes)
EnumFlag_LocalDevices	*$03*	Serial ports and K+K USB devices
EnumFlag_All	*$FF*	Lists all

2.1.4 ***function KK_GetHostAndIPs(HostName: PChar; IPaddr: PChar;*** (since 16.03) ***ErrorMsg: PChar): Integer; stdcall;***

Returns in *HostName* the PC's name and in *IPaddr* a comma separated list of all IP addresses of the PC.

If no network adapter is active, *127.0.0.1* for local host is returned.

All PChars must point to buffers of at least 80 byte allocated by the calling application!

Results:
CKK_DLL_NoError(1) = ok
CKK_DLL_Error(0) = Error; *ErrorMsg* contains an error message

2.2 Standard functions

2.2.1 ***function FX_OpenPort(var PortBaud: PChar): Integer; stdcall;***

(see also 2.2.2 function FX_OpenPortBlockingIO)

Closes any open connection and then opens the interface/device/remote connection given by the *PortBaud* string.

Result:
CKK_DLL_NoError(1) = ok
CKK_DLL_Error(0) = Error; *PortBaud* contains the error message.

Interface/device/remote connection may be one of the following:

A) Local devices:

A.1: Serial port

Windows: *COMx:*<Baudrate> e.g.: *COM1:115200*
Linux: */dev/tty*:*<Baudrate> e.g.: */dev/ttyS0:115200*

Hint: Under Linux the user must be member of the group *dialout* to have access to a serial port.

A.2: USB devices:

<DeviceName>[*:*[<Baudrate>]] , where

<DeviceName> must be one of the elements returned in *Names* by *KK_EnumerateDevices*;
[*:*[<Baudrate>]] may be given, but will be ignored.

Hint: Under Linux the user must be member of the group *root* to have access to a USB port.

A.3) CAN device: (since 15.12)

CAN:MENLO<Node> , where
<Node> is the device node number in Hex (*0..F*), part of the CAN ID .
(uses hard-coded settings for Menlo-CAN via PCAN-Basic API)

Hint: Works only under Windows with a Peak Systems PCAN-USB-Adapter.

B) Remote devices:

<IP-Address>[*:*[<Port>]] , where

<IP-Address> is the IPv4 address of the remote device;

[*:*[<Port>]] is the optional port number of the own local TCP client for reception of the data, may be omitted, will then be taken as Port=0 for automatic assignment of a port number by the system;<Port> must be in the range *0...65535* ;

C) Remote RPC Server:

SERVER:<IP-Address>*:*<Port> , where
<IP-Address> is the IPv4 address of the remote RPC server;
<Port> is the port number of the remote RPC server (started by an application via *KK_StartRpcServer*); must be in the range *0..65535*

D) File device: (since 15.10)

<Filename> , where
<Filename> is the complete path and name of a file of measurement data.

Such a file may be created by *StartSaveBinaryData* for binary data from a K+K device, or by *StartSaveReportData* for text reports from the DLL.

The distinction between binary and text format is made by file name: Names ending with '*...KK_ReportData.txt*' will be considered text files.

E) TCP device: (since 16.00, since 16.03 without the mode specifier)

TCP:<IP-Address>*:*<Port> , where
<IP-Address> is the IPv4 address of the TCP server, e.g. *192.168.2.98*;
<Port> is the port number of the TCP server, must be *0..65535* .

Receives DLL reports (measurement results and all messages, warnings...). The server is started by an application calling *KK_StartTcpServer* (see 2.9)

Since 15.11, a specific UserID may be requested for all connections by adding, separated by a blank,

·*USERID*=<nr> , where
<nr> identifies the chosen UserID = 1..4 ; e.g. *192.168.2.98·USERID=2*

To be observed under Linux:

The user must be member of certain groups to have the necessary access rights:
For access to a serial port: group *dialout*;
for USB access: group *root* .

Linux command (requires *root* rights!): *usermod -a -G* <group> <user>

2.2.2 ***function FX_OpenPortBlockingIO(var PortBaud: PChar): Integer;*** (since 15.08) ***stdcall;***

Similar to *FX_OpenPort*, but with blocking I/O operation, i.e. function calls wait for the characters to be received until eventually a timeout occurs (appr. 200ms).

NOT available for remote server and file device connections.

2.2.3 ***procedure FX_ClosePort; stdcall;***

Closes any currently open connection.

2.2.4 ***FX_GetReport(var Data: PChar): Integer; stdcall;***

Expects in *Data* a single character to be used as the decimal separator in formatting the reports.

Returns in *Data* the next available measurement report or message.
Data=nil if no report or message are available.

Results:
CKK_DLL_NoError(1) = ok
CKK_DLL_Error(0) = Error; *Data* contains an error message
CKK_DLL_UnknownError(2) = should never occur
CKK_DLL_WriteError(3) = Write Error; *Data* contains an error message
CKK_DLL_ServerDownError(4) = no connection to remote device or server error; *Data* contains an error message

On errors 3 and 4 the application should try to re-open the connection, i.e. call again *FX_OpenPort*.

2.2.5 ***function FX_SendCommand(var Command: PChar): Integer; stdcall;***

Expects in *Command* a command string to be transmitted to the device through the DLL's command FiFo buffer.

Results:
CKK_DLL_NoError(1) = ok
CKK_DLL_Error(0) = Error; *Command* contains an error message

2.2.6 ***procedure FX_Debug(const DbgStatus: Boolean); stdcall;***

Opens / closes the DLL internal debug log in the mode set by *FX_DebugFlags*. The log will be written to a file *FX_DLL_LOG.txt* with the same path as the application, by default without file size limit, overwriting an already existing log.

See *FX_Log*, F*X_DebugFlags* and *FX_DebugLogLimit* for options.

2.2.7 ***procedure FX_Log(const DbgNummer: Integer); stdcall;***

Opens / closes a DLL-internal debug log with indexed file name:

DbgNummer = -1: closes the debug log file
DbgNummer >= 0: closes a current log file and opens the file with name *FX_DLL_LOG_*<DbgNummer>*.txt*

2.2.8 ***procedure FX_DebugFlags(const ReportLog: Boolean; const LowLevelLog: Boolean); stdcall;***

Controls the amount of information stored in the debug log file. Must be called prior to opening the debug log!

Default: *ReportLog=True*, *LowLevelLog=False*

ReportLog =*True*: Reports delivered by *FX_GetReport* will be logged
LowLevelLog=*True*: all data exchanged between the device and the DLL will be logged.

2.2.9 ***procedure FX_DebugLogLimit(LogType: Byte; aSize: Cardinal=0); stdcall;***

(since 15.06)

Sets limits for the log type and file size (in byte). Procedure must be called prior to opening the debug log!

Default: *LogType*=0, *aSize*=0 -> 20Mbyte.

LogType	Value	Description
logUnlimited	$00	A single file *FX_DLL_LOG.txt* is written with no size limit (Default). *aSize* is irrelevant.
logOverwrite	$01	The log file is limited in size (Default *aSize*=0: 20MB). When full, the file will be overwritten.
logCreateNew	$02	File names are indexed with date and time: *FX_DLL_LOG_*<yyyymmdd>_<hhmmss>*.txt* The log file is limited in size (Default *aSize*=0: 20MB). When full, a new file is created.

2.2.10 ***function KK_GetDLLVersion: PChar; stdcall;***
(since 14.08)

Results in a string containing the DLL version date and number.

2.3 Special functions for LabView programs

LabView cannot handle pointers to pointers like '*var Name: PChar*'.

Therefore, a set of functions *LV_** is available analog to the Standard functions *FX_** described above, where the calling application has to allocate at least 1024 byte of string space for each parameter.

All *LV_** functions may result in an additional error code:
CKK_DLL_BufferTooSmall(6) = The return string was truncated to 1024 bytes

2.3.1 ***function LV_EnumerateDevices(Names: PChar; EnumFlags: Byte): Integer; stdcall;***

Since 15.02, analog *KK_EnumerateDevices*

2.3.2 ***function LV_OpenPort(PortBaud: PChar): Integer; stdcall;***

analog *FX_OpenPort*

2.3.3 ***procedure LV_ClosePort; stdcall;***

no longer available since 15.03.

2.3.4 ***function LV_GetReport(Data: PChar): Integer; stdcall;***

analog *FX_GetReport*

2.3.5 ***function LV_SendCommand(Command: PChar): Integer; stdcall;***

analog zu *FX_SendCommand*

2.3.6 ***function LV_OpenTcpLog(IpPort: PChar; Mode: PChar): Integer; stdcall;***

Since 16.00, analog *KK_OpenTcpLog*

2.3.7 ***function LV_GetTcpLog(Data: PChar): Integer; stdcall;***

Since 16.00, analog *KK_GetTcpLog*

2.4 Custom functions

These functions are used by K+K in applications specifically designed to fit their own or individual customer's needs. Contact K+K if interested.

2.4.1 ***function FX_GetFXM(var Data: PChar): Integer; stdcall;***

analog *FX_GetReport* for K+K FXM phase meter cards.

2.4.2 ***function LE_ReadMonitor(var Data: PChar): Integer; stdcall;***

analog *FX_GetReport* für K+K DNSZ twin time interval counters.

2.4.3 ***function LE_ReadMonPha(var Data: PChar): Integer; stdcall;***

combination of *FX_GetReport* and *LE_ReadMonitor* for FXE and DNSZ based time and frequency monitors.

2.5 Local RPC Server

These functions were developed for internal use only. Contact K+K if interested.

2.5.1 ***function KK_StartRpcServer(var aPort: Word; var rootPath: PChar): Boolean; stdcall;***

(renamed: was *FX_StartRpcServer* prior to 15.01)

2.5.2 ***function KK_StopRpcServer: Boolean; stdcall;***

(renamed: was *FX_StopRpcServer* prior to 15.01)

2.5.3 ***procedure KK_TakeWarning(Warning: PChar); stdcall;***

(renamed: was FX_TakeWarning prior to 15.01)

2.5.4 ***procedure KK_EnableStoraging; stdcall;***

2.5.5 ***function KK_GetDefaultStorageRoot: PChar; stdcall;***

2.6 Remote RPC Server

2.6.1 ***function KK_GetArchiveData(var aRange: TDateTimeRange; var Data: PChar; var len: LongWord): Integer; stdcall;***

2.7 Generating Test Data

2.7.1 ***function StartSaveBinaryData(RawData: Boolean): Boolean;*** (since 15.10) ***stdcall;***

Writes reports received from a K+K device into a binary file named <Date>_<Time>_*KK_BinaryData.bin* (same path as the application)

RawData should be set to *False*.

Results: *True*: binary file opened
False: writing binary data is not supported by the K+K device

2.7.2 ***function StopSaveBinaryData: Boolean; stdcall;*** (since 15.10)

Stops writing binary data and closes the file.

Results: *True*: binary file closed
False: writing binary data is not supported by the K+K device

2.7.3 ***procedure StartSaveReportData; stdcall;*** (since 15.10)

Writes text reports delivered by the DLL to the application into a text file named <Date>_<Time>_*KK_ReportData.txt* (same path as the application)

2.7.4 ***function StopSaveReportData: Boolean; stdcall;*** (since 15.10)

Stops writing text reports and closes the text file.

Results: *True*: text file closed
False: text file was not open

2.8 Tools

These routines were included to help in developing e.g. the Upload tool.
They were developed for internal use only. Contact K+K if interested.

2.8.1 ***function KK_OpenDevice(var DeviceName: PChar; aWatchList: TCFWatchList = nil): Integer; stdcall;***

Opens a connection to a K+K device in the tool mode, where no measurement data are being read.

Results:
CKK_DLL_NoError(1) = ok
CKK_DLL_Error(0) = Error, *DeviceName* contains an error message

2.8.2 ***function KK_ReadDevice(len: Byte; var Bytes: PChar; var ErrorMsg: PChar): Integer; stdcall;***

Reads *len* bytes of data from the K+K device into *Bytes*.

Results:
CKK_DLL_NoError(1) = ok
CKK_DLL_Error(0) = Error, *ErrorMsg* contains an error message

2.8.3 ***function KK_WriteDevice(Cmd: PChar; len: Byte; var ErrorMsg: PChar): Integer; stdcall;***

Sends a command *Cmd* of length *len* bytes to the K+K device.

Results:
CKK_DLL_NoError(1) = ok
CKK_DLL_Error(0) = Error, *ErrorMsg* contains an error message

2.9 Local TCP Server

In order to provide to other applications via TCP/IP the binary data received from the K+K device and/or any text messages compiled by the application, an application may start a local TCP server.

Once started, the server will be provided DLL-internally with all binary data received from the K+K device. In addition, the application can upload to the server (see 2.9.4 *KK_TcpReportLog*) text messages in three modes: Phase, Phase Difference, and Frequency.

The server provides these data to its clients in real time, enabling them to either use the DLL-internal binary data (2.2.1 *FX_OpenPort* with E) TCP device) for a 'daughter application', or use the application compiled text messages (2.10.3 *KK_GetTcpLog*) for some customer written evaluation software.

2.9.1 ***function KK_StartTcpServer(var aPort: Word): Boolean; stdcall;***
(since 16.00)

Starts a TCP server with port *aPort*. With *aPort*=0, the system will assign a port number, which is returned in *aPort*.

Note: The TCP server can be accessed only with this port number.
The port number must therefore be published to the clients!

Results: *True*: TCP server is running
False: Error; for the error message see *KK_GetTcpServerError*

2.9.2 ***function KK_StopTcpServer: Boolean; stdcall;***
(since 16.00)

Stops the TCP Server; any existing client connections will be closed.

Results: *True*: TCP server is running
False: Error; for the error message see *KK_GetTcpServerError*

2.9.3 ***function KK_GetTcpServerError: PChar; stdcall;***
(since 16.00)

Results in the error message stored in the TCP server, or a null pointer.

2.9.4 ***procedure KK_TcpReportLog(Data: PChar; logType: Integer);*** (since 16.00) ***stdcall;***

Called by an application to upload to the TCP server text messages in three different modes, for dissemination to clients in real time.

For example, the standard FXE sample application uploads to the TCP server the very same lines which optionally may be stored in log files.

logType specifies the mode assigned to the uploaded message.

Correspondingly, a client connecting to a TCP server needs to specify the message *Mode* it wants to receive (see 2.10.1 *KK_OpenTcpLog*):

logType	TCP client *Mode*
0	*PHASELOG*
1	*FREQLOG*
2	*PHASEDIFFLOG*

2.10 TCP-Client

An application starts a TCP client to read text messages with a specific *Mode* from a TCP server. That server may either run on the same PC (via local host *127.0.0.1*) or on another PC connected through a network.

Note: Use *FX_OpenPort* with F) TCP device to receive DLL-style binary data for a daughter display.

2.10.1 ***function KK_OpenTcpLog(var IpPort: PChar; Mode: PChar): Integer; stdcall;***
(since 16.00)

Creates a TCP client to receive messages with a specific *Mode* from a TCP server with IP address and port number *IpPort*.

IpPort: <Ipv4-Address>*:*<Port> of the TCP server; e.g. *192.168.178.98:5000*
may be *127.0.0.1* for local host
<Port> must be in the range *0..65535*

Mode: Message *Mode* to be received: *PHASELOG*, *FREQLOG* or *PHASEDIFFLOG* (see also 2.9.4 *procedure KK_TcpReportLog*)

Results:
CKK_DLL_NoError(1) = ok
CKK_DLL_Error(0) = Error, *IpPort* contains an error message
(e.g. report mode not available)

The TCP client maintains a receive buffer for 10k messages. If the TCP server delivers the messages faster than the application reads them (*KK_GetTcpLog*), messages will be lost, as received messages are discarded if the buffer is full.

2.10.2 ***procedure KK_CloseTcpLog; stdcall;***
(since 16.00)

Disconnects the TCP client from the TCP server and stops the client.

Remaining messages may still be retrieved by calling KK_GetTcpLog.

2.10.3 ***function KK_GetTcpLog(var Data: PChar): Integer; stdcall;*** (since 16.00)

Returns in *Data* the next text message from the TCP client's receive buffer. If no messages are available, *Data*=nil.

The TCP client maintains a receive buffer for 10k messages. If the TCP server delivers the messages faster than the application reads them (*KK_GetTcpLog*), messages will be lost, as received messages are discarded if the buffer is full.

Results:
CKK_DLL_NoError(1) = ok
CKK_DLL_BufferOverflow (8) = some messages have been lost