K+K

Messtechnik GmbH

Handbuch KK-Library

Version 19.3.0 vom 09.10.2022

Die KK-Library gibt es in folgenden Versionen:

KK_FX80E.DLL: für Windows 32-Bit

KK_Library_64.dll: für Windows 64-Bit

libkk_fx80e.so: für Linux 32-Bit

libkk_library_64.so: für Linux 64-Bit

Spezialversionen für Python auf Linux-Systemen mit Aufrufkonvention **cdecl**:
libkk_library_32_cdecl.so

libkk_library_64_cdecl.so.

Dieses Handbuch beschreibt ausschließlich die mit Version 18.00 neu eingeführten Multi-Source-Aufrufe mit denen mehr als eine Verbindung gleichzeitig unterhalten werden kann.

Multi-Source-Aufrufe dürfen nicht mit den bisherigen Aufrufen gemischt werden. Wir empfehlen bei Neuentwicklungen nur die Multi-Source-Aufrufe zu verwenden.

Die Beschreibung der bisherigen Aufrufe befindet sich in den Vorgänger-Versionen zu diesem Handbuch.

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

Änderungshistorie

Wann	Wer	Library-Version	Was
07.02.2019	Loryn	18.00	Neu erstellt Multi-Source-Library
20.02.2019	Loryn	18.01	Verbindungsart Simulierte Daten ergänzt
23.01.20	Loryn	18.01.03	Bis zu 4 CAN-Verbindungen pro PCAN-USB-Adapter Enum-Flag 0x04 für PCAN-USB-Adapter Multi_SetNSZ: Wert 0 für automatische Erkennung
04.03.20	Loryn	18.01.04	Multi_GetReport liefert Fehler CKK_DLL_CmdIgnored
28.04.20	Loryn	18.01.07	Daten aus Datei: Option *WAIT_INTERVAL* ergänzt
12.10.20	Loryn	18.01.10	Ergänzt: Multi_GetFirmwareVersion, Multi_SetNSZCalibrationData, Multi_HasFRAM, Multi_RemoteLogin NSZ-Kalibrierung TCP-Server erweitert um NSZLOG, NSZDIFFLOG
06.11.20	Loryn	18.02.02	Multi_SendCommand: ungültige Kommandos abweisen Multi-USB ab Firmware 63
15.03.21	Loryn	18.02.04	TCP-Server erweitert um PHASEPREDECESSORLOG, USERLOG1, USERLOG2
28.05.21	Loryn	19.00.02	Multi_GetReport liefert Fehler CKK_DLL_Reconnected Ergänzt: Multi_OpenTcpLogTime
18.11.21	Loryn	19.01.02	FHR - FXE High Resolution ergänzt Multi_IsSerialDevice ergänzt Fehler CKK_DLL_NotSupported bei serieller Verbindung
20.12.21	Loryn	19.02.00	Ergänzt: Multi_OpenTcpLogType, Multi_TcpAppData
08.04.22	Loryn	19.02.00	Korrektur C-Syntax Multi_DebugGetFilename
09.10.22	Loryn	19.03.00	Multi_TcpAppData liefert Server-Antwort

Inhaltsverzeichnis

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. Übersicht

Die KK-Library dient primär der Kommunikation mit K+K-Geräten zum Empfang von Messdaten und Nachrichten vom Gerät und zum Senden von Steuer-Kommandos an das Gerät.

Weiterhin können die vom Gerät empfangenen Daten via Server an andere Anwendungen weiterverteilt werden bzw. von Servern abgerufen werden.

Auch das Erzeugen von Testdaten und Verarbeiten von Daten aus Dateien wird unterstützt.

Ab Version 18.00 der KK-Library sind mehrere parallele Verbindungen möglich.

Alle Library-Aufrufe, die String-Typen (PAnsiChar, AnsiChar) verwenden, verarbeiten ausschliesslich ASCII-codierte, 1-Byte große Zeichen.

1.1 Kommunikation mit dem K+K-Gerät

Ein K+K-Gerät kann über vier verschiedene Schnittstellen kommunizieren

1. seriell via RS232-Verbindung
2. USB-Verbindung
3. CAN-Verbindung
4. Netzwerk-Verbindung via TCP/IP

Wenn eine Verbindung zu einem K+K-Gerät aufgebaut werden soll, muss der gewünschte Kommunikationsweg ausgewählt werden (siehe *Verbindungsarten*).

1.1.1 Empfang von Daten vom K+K-Gerät

Bei Verbindungen via USB und Netzwerk wird Library-intern ein Empfangsthread gestartet, der die vom Gerät empfangenen Bytes in einem **Empfangspuffer** ablegt. Der *Multi_GetReport*-Aufruf liest aus diesem Empfangspuffer und liefert Reports (Messdaten, Nachrichten) in lesbarer Form (ASCII-Format) zurück.

Bei serieller Verbindung oder CAN-Verbindug gibt es keinen Empfangsthread, *Multi_GetReport* liest direkt vom Gerät.

Neben den verschiedenen Schnittstellen kann ein K+K-Gerät vier Verbindungen gleichzeitig unterhalten. Jeder Verbindung wird dabei eine **UserID** (1..4) zugewiesen.

1.1.2 Senden von Steuer-Kommandos zum K+K-Gerät

Die Behandlung der via *Multi_SendCommand* an die KK-Library übergebenen Kommandos hängt von der Übertragungsart ab. Mit Ausnahme der Netzwerk-Verbindung werden Kommandos zwischengespeichert und zu einem gegebenen Zeitpunkt an das K+K-Gerät gesendet.

Bei einer Netzwerk-Verbindung wird das Kommando direkt von *Multi_SendCommand* gesendet.

Bei einer seriellen oder CAN-Verbindung wird das Kommando gesendet, sobald ein Report vollständig vom Gerät empfangen worden ist, d.h. das Kommando wird im *Multi_GetReport*-Aufruf gesendet.

Bei USB- (und auch bei Netzwerk-) Verbindungen sendet das K+K-Gerät in Blöcken, die auch mehrere Reports enthalten können. Bei USB-Verbindungen wird das Kommando gesendet, sobald ein Block vom Gerät empfangen worden ist, d.h. der Empfangsthread sendet die Kommandos.

Die Anzahl der zwischengespeicherten Kommandos ist zunächst unbegrenzt. Mit *Multi_GetPendingCmdsCount* kann diese Anzahl abgefragt und mit *Multi_SetCommandLimit* kann die Anzahl begrenzt werden.

1.2 Lokaler Server

Zu jeder Verbindung mit einem K+K-Gerät kann ein lokaler Server gestartet werden, der alle vom Gerät empfangenen Daten an die bei ihm angemeldeten Clients weiterverteilt.

Die Anwendung muss ggf. in der Windows-Firewall freigegeben werden.

1.2.1 Lokaler TCP-Server

Eine Anwendung kann einen TCP-Server starten (*Multi_StartTcpServer*), um anderen Anwendungen Messdatenstrings bzw. Logeinträge via TCP/IP zur Verfügung zu stellen.

Ist ein TCP-Server gestartet, wird er Library-intern mit Reports (Messdaten und Nachrichten) versorgt. Der TCP-Server liefert Reports in lesbarer Form (ASCII-Format) entweder auf Library-Ebene (Reports lesen via *Multi_GetReport*) oder auf LOG-Ebene (Logeinträge, die die Anwendung via *Multi_TcpReportLog* an den TCP-Server übergibt, Lesen via *Multi_GetTcpLog*).

Der TCP-Server verteilt in Echtzeit die Daten, die ihm übergeben werden an die Clients, die bei ihm angemeldet sind. Wenn ein Client nicht erreichbar ist (Verbindung zwischen Server und Client wurde geschlossen, z.B. Client wurde beendet ohne sich abzumelden), wird die Client-Anmeldung im TCP-Server gelöscht.

1.3 Verbindung zu einem Server

Neben der Kommunikation mit einem K+K-Gerät bietet die KK-Library die Möglichkeit sich mit einem Server zu verbinden, der von einer anderen Anwendung mit KK-Library gestartet worden ist. Der Server kann auf demselben Rechner (Localhost) oder auf einem anderen Rechner gestartet worden sein.In diesem Fall übernimmt die KK-Library die Rolle des Clients.

Die Anwendung muss ggf. in der Windows-Firewall freigegeben werden.

1.3.1 Verbindung zu einem TCP-Server auf Library-Ebene

Auf Library-Ebene werden die Daten vom TCP-Sever via *Multi_GetReport* gelesen. Die Verbindung muss zuvor via *Multi_OpenConnection* geöffnet worden sein. Siehe dazu *Verbindungsarten - Verbindung zu einem TCP-Server*.

1.3.2 Verbindung zu einem TCP-Server auf LOG-Ebene

Eine Anwendung kann einen TCP-Empfänger starten, der Logeinträge von einem TCP-Server empfängt (*Multi_OpenTcpLog, Multi_OpenTcpLogTime, Multi_OpenTcpLogType*).

Wird ein TCP-Empfänger gestartet öffnet dieser intern einen TCP-Server-Socket, um Reports vom TCP-Server zu empfangen. Dazu meldet er sich als Client beim TCP-Server an mit dem gewünschten Reportmodus der Logeinträge.

Der TCP-Empfänger verwaltet intern einen Empfangspuffer von 10.000 Einträgen. Sendet der TCP-Server schneller, als die Anwendung ausliest (mit *Multi_GetTcpLog*), kommt es zu einem Datenverlust (*Multi_GetTcpLog* meldet Fehler *CKK_DLL_BufferOverflow(8)*).

1.3.3 AppData an TCP-Server senden

Unabhängig davon, ob die Verbindung zu einem TCP-Server auf Library-Ebene oder LOG-Ebene besteht, kann via *Multi_TcpAppData* ein String an die Anwendung gesendet werden, die den TCP-Server gestartet hat. Die auf der Server-Seite empfangenen Strings werden als Report mit Header $7F40 beim nächsten GetReport-Aufruf an die Anwendung weitergeleitet.

1.4 Testdaten

Empfangenen Reports können als Testdaten in Dateien abgespeichert werden, um sie später erneut einzulesen und zu verarbeiten.

Die Testdaten können als binäre Reports (**SaveBinaryData*) oder als lesbare ASCII-Reports (**SaveReportData*) gespeichert werden. Die Dateien werden im Verzeichnis **OutputPath** (siehe *Multi_SetOutputPath*) angelegt.

Das Einlesen der Testdaten ist als weitere Verbindungsart implementiert. Siehe dazu *Verbindungsarten - Daten aus Datei* zum Öffnen der Verbindung, Lesen wie gewohnt per *Multi_GetReport*.

1.5 Verbindungsarten

Eine neu zu öffnende Verbindung wird durch den **OpenString** beschrieben, der an *Multi_OpenConnection* übergeben wird.

Eine Ausnahme bildet die Verbindung zu einem TCP-Server auf LOG-Ebene. Dort ist *Multi_OpenTcpLog* zu verwenden.

Neben der Verbindungsart kann auch eine UserID vorgegeben werden.

Der OpenString besteht aus dem ConnectionString und einer ggf. angehängten UserID. Der Aufbau des ConnectionStrings wird nachfolgend beschrieben.

1.5.1 Lokal an den PC per Kabel angeschlossene K+K-Geräte

1.5.1.1 Serielle Schnittstelle

ConnectionString = <*SerialName*> mit

- <*SerialName*>: Name der seriellen Schnittstelle
  - Windows: COMx, z.B.: 'COM1'
  - Linux: /dev/tty*, z.B.: '/dev/ttyS0'

Hinweis:

Damit unter Linux auf eine serielle Schnittstelle zugegriffen werden kann, ist es notwendig, dass der Benutzer zur Gruppe **dialout** gehört.

1.5.1.2 USB

ConnectionString = <*DeviceName*> mit

- <*DeviceName*> muss mit dem von *Multi_EnumerateDevices* gelieferten Namen übereinstimmen

Hinweis:

Damit unter Linux auf eine USB-Schnittstelle zugegriffen werden kann, ist es notwendig, dass der Benutzer zur Gruppe **root** gehört.

(ab 18.2)

Ab Firmware 63 kann eine Anwendung bis zu 4 Verbindungen über dieselbe USB-Schnittstelle betreiben.

1.5.1.3 CAN

ConnectionString = CAN:MENLO<Node> mit

(ab 18.1.3)

ConnectionString = CAN:MENLO<Node> [CHANNEL<Channel>] mit

- <*Node*> Device-Knotennummer in HEX (0..F), Bestandteil der CAN-ID; CAN-ID ist eine Hardcodierte Einstellung für Menlo-CAN via PCAN-Basic API
- <Channel> Verbindungsnummer 0..3, zur Unterscheidung von bis zu 4 verschiedenen Verbindungen. Voreingestellt ist Verbindungsnummer 0

Hinweis:

Nur unter Windows mit Peak-Systems PCAN-USB-Adapter möglich.

1.5.2 Verbindung via Netzwerk zu einem K+K-Gerät

ConnectionString = <IP-Adresse>[:<Port>] mit

- <*IP-Adresse*> IP-Adresse des K+K-Gerätes im Format einer IPv4-Adresse, z.B. 192.168.178.98
- *[:<Port>]* Portnummer des lokalen PCs auf der Messdaten empfangen werden, kann weggelassen werden, dann wird die Portnummer vom Betriebssystem vergeben; Portnummer muss ein positiver 16-Bit-Wert sein

Hinweis:

Das K+K-Gerät identifiziert die Verbindung mit IP-Adresse und Portnummer. Damit das K+K-Gerät bei einem erneuten Verbindungsaufbau denselben Übertragungspuffer zuweist, muss entweder die User-ID gesetzt sein oder dieselbe Portnummer verwendet werden.

Die Anwendung muss ggf. in der Windows-Firewall freigegeben werden.

1.5.3 Verbindung zu einem TCP-Server

ConnectionString = TCP:<IP-Adresse>:<Port> mit

- *<IP-Adresse>* IP-Adresse des TCP-Servers im Format einer IPv4-Adresse, z.B. 192.168.178.98 oder 127.0.0.1 für Localhost
- *<Port>* Portnummer des TCP-Servers, muss ein positiver 16-Bit-Wert sein

Hinweis:

Empfängt Reports vom TCP-Server auf Library-Ebene - Messdaten und Nachrichten. Der TCP-Server muss von einer Anwendung mit KK-Library via *Multi_StartTcpServer* gestartet worden sein.

Die Anwendung muss ggf. in der Windows-Firewall freigegeben werden.

1.5.4 Daten aus Datei

ConnectionString = <Filename>[LOOP_FOR_EVER][WAIT_INTERVAL] mit

- *<Filename>* Dateiname einer Messdatendatei
- *[LOOP_FOR_EVER]* optional: Daten werden in einer Endlosschleife immer wieder von vorn aus der Datei gelesen
- *[WAIT_INTERVAL]* optional: Wartezeit zwischen den Reports passend zu Report-Intervall

Die Messdatendatei wird von *Multi_StartSaveBinaryData* für binäre Daten vom K+K-Gerät, bzw. von *Multi_StartSaveReportData* für lesbare Reports der KK-Library erzeugt. Die Unterscheidung binäre oder Report-Daten wird über den Dateinamen getroffen. Endet der Dateiname mit KK_ReportData.txt, werden Reportdaten vorausgesetzt.

1.5.5 Simulierte Daten

ConnectionString = DemoData

Library-intern werden Messdaten und FDI-Daten von einem Simulator erzeugt.

1.5.6 User-ID

Ein K+K-Gerät kann bis zu vier Verbindungen gleichzeitig betreiben. Jeder Verbindung wird eine User-ID und damit ein Übertragungspuffer zugeteilt. Die gewünschte User-ID kann angegeben werden mit:

USERID=<nr> mit

- *<nr>* bezeichnet User-ID = 1..4

Die Angabe der User-ID ist nur wirksam bei Verbindung zu einem K+K-Gerät, allerdings **nicht bei serieller** Verbindung. Damit bei einem erneuten Verbindungsaufbau derselbe Übertragungspuffer verwendet wird, sollte dieselbe User-ID verwendet werden.

1.6 Mehrfach-Verbindungen

Ab KK-Library-Version 18.00 wurde die Möglichkeit geschaffen mehr als eine Verbindung gleichzeitig zu unterhalten. Dazu wurden die Multi_*-Routinen ergänzt.

Eine Anwendung, die mehr als eine Verbindung gleichzeitig unterhalten will, muss diese Multi_*-Aufrufe verwenden.

Pro Verbindung muss *CreateMultiSource* aufgerufen werden. Damit wird Library-intern ein neues Verwaltungs-Objekt angelegt. *CreateMultiSource* liefert eine Source-ID zurück, die bei allen folgenden Aufrufen angegeben werden muss und diese Multi-Verbindung identifiziert. Die Verwaltungs-Objekte werden bei Anwendungsende automatisch freigegeben.

Pro Source ist eine Verbindung zugelassen, die jedoch gewechselt werden kann.

Die Multi_*-Aufrufe sind an die bisherigen Library-Funktionen angelehnt. Allerdings muss der Aufrufer den Puffer für PAnsiChar-Parameter bereitstellen. Es wird vorausgesetzt, dass 1024 Byte Puffer zur Verfügung stehen.

Alle Multi_*-Aufrufe sind Thread-sicher.

Hinweis:

Multi_*-Aufrufe dürfen nicht mit den bisherigen Aufrufen gemischt werden. Wir empfehlen bei Neuentwicklungen nur die in diesem Handbuch beschriebenen Multi_*-Aufrufe zu verwenden.

1.7 NSZ-Daten und deren Kalibrierung

Sind in einem K+K-Gerät (D)NSZ-Messkarten verbaut, werden NSZ-Daten in Reports mit Header $7900 gesendet. Pro Kanal werden die Messwerte in Nanosekunden gemeldet (Gleitkommastring mit zwei Nachkommastellen).

NSZ-Kalibrierwerte werden im Flash-Speicher oder F-RAM des K+K-Gerätes abgespeichert.

Ab Firmware-Version 62 können diese Kalibrierwerte mit Kommando $02 angefordert werden. Das K+K-Gerät antwortet mit Report $7901, der pro Kanal einen Kalibrierwert in Nanosekunden enthält (Gleitkommastring mit drei Nachkommastellen).

Mittels der Funktion *Multi_SetNSZCalibrationData* können NSZ-Kalibrierwerte an das K+K-Gerät übertragen werden. Voraussetzungen sind

- Firmware-Version ab 62
- K+K-Library-Version ab 18.01.10

Werden Kalibrierwerte geändert, antwortet das K+K-Gerät mit Report $7901 an alle User.

Der TCP-Server auf Library-Ebene, speichert den aktuellen $7901-Report und sendet ihn an alle Clients, die sich neu auf Library-Ebene anmelden.

1.8 FHR – FXE High Resolution

FHR ist ein mehrkanaliges 'Dual-Mixer'-Frontend für den FXE-Phasenzähler, das jeden seiner HF-Eingänge mit einem gemeinsamen lokalen Oszillator (LO) mischt, um Zwischenfrequenzen (IF) von ca. 500kHz zu erreichen. Für jeden Kanal ist die 20. Subharmonische eines rauscharmen 10-MHz-Oszillators phasenstarr mit seiner ZF, während die Phase dieser 10MHz von FXE gemessen wird, wodurch die Phasenauflösung um einen Faktor von RF / (RF-LO) verbessert wird.

Zur Verwendung von FHR-Karten in einem K+K-Gerät werden die FHR-Einstellungen pro Kanal im Flash-Speicher oder F-RAM des K+K-Gerätes abgespeichert.

FHR-Einstellungen pro Kanal: <Nominalfrequenz>, <LO-Frequenz>, <zugelassen>

Nominalfrequenz und LO-Frequenz in Hz, zugelassen: 0 = ohne FHR, 1 = mit FHR

Daraus kann zu jedem Kanal mit zugelassen=1 der FHR-Faktor berechnet werden:

FHR-Faktor = Nominalfrequenz / (Nominalfrequenz – LO-Frequenz)

Ab Firmware-Version 67 können die FHR-Einstellungen gelesen und geschrieben werden:

- *Multi_ReadFHRData* fordert FHR-Einstellungen an; das K+K-Gerät antwortet mit Report $7902
- *Multi_SetFHRData* setzt die FHR-Einstellungen im K+K-Gerät.

Werden FHR-Einstellungen geändert, antwortet das K+K-Gerät mit Report $7902 an alle User.

Der TCP-Server auf Library-Ebene, speichert den aktuellen $7902-Report und sendet ihn an alle Clients, die sich neu auf Library-Ebene anmelden.

Format $7902-Report:

7902<Kanal1>/<Kanal2>/ … /<Kanal24>

<pro Kanal>:

;<Nominalfrequenz>;<LO-Frequenz>;<zugelassen>

<Nominal-Frequenz>, <LO-Frequenz>:

Gleikommastring in Hz

<zugelassen>:

0 = ohne FHR, 1 = mit FHR

2. Exportierte Funktionen

Alle Versionen der KK-Library (*KK_FX80E.DLL* für Windows 32-Bit, KK_Library_64.dll für Windows 64-Bit, *libkk_fx80e.so* für Linux 32-Bit, *libkk_library_64.so* für Linux 64-Bit) exportieren die gleichen Funktionen.

Alle Funktionen haben die Aufrufkonvention **stdcall**:

- Parameterübergabe von rechts nach links
- Rückgabewerte in Register
- Aufgerufene Funktion bereinigt den Stack. Daher sind keine variablen Argumentlisten möglich.

Alle exportierten Routinen werden hier in Pascal-Notation beschrieben. Kapitel 4 beschreibt die Funktionsdeklarationen in C.

Hinweis:

Für Python auf Linux-Systemen stehen KK-Library-Versionen mit Aufrufkonvention **cdecl** zur Verfügung: libkk_library_32_cdecl.so und libkk_library_64_cdecl.so.

2.1 Mehrfach-Verbindung anlegen

- function **CreateMultiSource**: Integer; stdcall;

2.2 Verfügbare Schnittstellen auflisten

- function **Multi_EnumerateDevices**(Names: PAnsiChar; EnumFlags: Byte): Integer; stdcall;
- function **Multi_GetEnumerateDevicesErrorMsg**: PAnsiChar; stdcall;
- function **Multi_GetHostAndIPs**(HostName: PAnsiChar; IPaddr: PAnsiChar; ErrorMsg: PAnsiChar): Integer; stdcall;

2.3 Pfadangaben

- function **Multi_GetOutputPath**(ID: Integer): PAnsiChar; stdcall;
- function **Multi_SetOutputPath**(ID: Integer; Path: PAnsiChar): PAnsiChar; stdcall;

2.4 Debug-Protokoll

- function **Multi_Debug**(ID: Integer; DbgOn: Boolean; DbgID: PAnsiChar): PAnsiChar; stdcall;
- function **Multi_DebugFlags**(ID: Integer; ReportLog: Boolean; LowLevelLog: Boolean): Integer; stdcall;
- function **Multi_DebugLogLimit**(ID: Integer; LogType: Byte; aSize: Cardinal): Integer; stdcall;
- function **Multi_DebugGetFilename**(ID: Integer): PAnsiChar; stdcall;

2.5 Info-Abfragen

- function **Multi_GetDLLVersion**: PAnsiChar; stdcall;
- function **Multi_GetBufferAmount**(ID: Integer): Integer; stdcall;
- function **Multi_GetTransmitBufferAmount**(ID: Integer): Integer; stdcall;
- function **Multi_GetUserID**(ID: Integer): Byte; stdcall;
- function **Multi_IsFileDevice**(ID: Integer): Boolean; stdcall;
- function **Multi_IsSerialDevice**(ID: Integer): Boolean; stdcall;
- function **Multi_GetFirmwareVersion**(ID: Integer): Integer; stdcall;
- function **Multi_HasFRAM**(ID: Integer): Boolean; stdcall;

2.6 Verbindung öffnen, schliessen

- function **Multi_OpenConnection**(ID: Integer; Connection: PAnsiChar; BlockingIO: Boolean): Integer; stdcall;
- procedure **Multi_CloseConnection**(ID: Integer); stdcall;

2.7 Reports einlesen

- function **Multi_SetDecimalSeparator**(ID: Integer; Separator: AnsiChar): Integer; stdcall;
- function **Multi_SetNSZ**(ID: Integer; aNSZ: Integer): Integer; stdcall;
- function **Multi_GetReport**(ID: Integer; Data: PAnsiChar): Integer; stdcall;

2.8 Kommandos senden

- function **Multi_GetPendingCmdsCount**(ID: Integer): Cardinal; stdcall;
- function **Multi_SetCommandLimit**(ID: Integer; Limit: Cardinal): Integer; stdcall;
- function **Multi_SendCommand**(ID: Integer; Command: PAnsiChar; Len: Integer): Integer; stdcall;
- function **Multi_RemoteLogin**(ID: Integer; Password: LongWord; err: PAnsiChar): Integer; stdcall;

2.9 Lokaler TCP-Server

- function **Multi_StartTcpServer**(ID: Integer; **var** aPort: Word): Integer; stdcall;
- function **Multi_StopTcpServer**(ID: Integer): Integer; stdcall;
- function **Multi_GetTcpServerError**(ID: Integer): PAnsiChar; stdcall;
- procedure **Multi_TcpReportLog**(ID: Integer; Data: PAnsiChar; logType: Integer); stdcall;

2.10 Verbindung zu einem TCP-Server auf LOG-Ebene

- function **Multi_OpenTcpLog**(ID: Integer; IpPort: PAnsiChar; Mode: PAnsiChar): Integer; stdcall;
- function **Multi_OpenTcpLogTime**(ID: Integer; IpPort: PAnsiChar; Mode: PAnsiChar; Format: PAnsiChar): Integer; stdcall;
- function **Multi_OpenTcpLogType**(ID: Integer; IpPort: PAnsiChar; LogType: Integer; Format: PAnsiChar): Integer; stdcall;
- procedure **Multi_CloseTcpLog**(ID: Integer); stdcall;
- function **Multi_GetTcpLog**(ID: Integer; Data: PAnsiChar): Integer; stdcall;

2.11 AppData an TCP-Server senden

- function **Multi_TcpAppData**(ID: Integer; Data: PAnsiChar): Integer; stdcall;

2.12 Testdaten erzeugen

- function **Multi_StartSaveBinaryData**(ID: Integer; DbgID: PAnsiChar): Integer; stdcall;
- function **Multi_StopSaveBinaryData**(ID: Integer): Integer; stdcall;
- function **Multi_StartSaveReportData**(ID: Integer; DbgID: PAnsiChar): Integer; stdcall;
- function **Multi_StopSaveReportData**(ID: Integer): Integer; stdcall;

2.13 NSZ-Kalibrierung

- function **Multi_SetNSZCalibrationData**(ID: Integer; Data: PAnsiChar): Integer; stdcall;

2.14 FHR-Einstellungen

- function **Multi_ReadFHRData**(ID: Integer): Integer;
- function **Multi_SetFHRData**(ID: Integer; Data: PAnsiChar): Integer;

3. Funktionsbeschreibung

Funktionsnamen, Aufrufparameter und Rückgabewerte sind in allen Versionen der KK-Library identisch.

3.1 Mehrfach-Verbindung anlegen

3.1.1 CreateMultiSource

function CreateMultiSource: Integer; stdcall;

Erzeugt ein neues Verwaltungs-Objekt und liefert eine Source-ID zurück. Diese Source-ID muss bei nachfolgenden Multi_*-Aufrufen angegeben werden.

Liefert:

- Source-ID (>= 0)

Hinweis:

Alle Verwaltungs-Objekte werden bei Programmende automatisch freigegeben.

3.2 Verfügbare Schnittstellen auflisten

3.2.1 Multi_EnumerateDevices

function Multi_EnumerateDevices(Names: PAnsiChar; EnumFlags: Byte): Integer; stdcall;

Listet die über *EnumFlags* ausgewählten Schnittstellen/K+K-Geräte auf und gibt sie in *Names* zurück.

Parameter:

- *Names*: enthält gefundene Einträge durch Komma getrennt oder 0, wenn keine Schnittstelle/Gerät gefunden wurde. Muss auf einen Puffer mit 1024 Byte zeigen, Ausgabe in Names wird ggf. auf 1024 Byte gekürzt
- *EnumFlags*: definiert zu suchende Elemente, siehe Tabelle unten

Liefert:

- 0: kein Fehler
- < 0: Fehlerflag analog zu *EnumFlags*: Beim Auflisten der entsprechenden Schnittstellen/Geräte ist ein Fehler aufgetreten.

Im Fehlerfall kann die Fehlermeldung mit *Multi_GetEnumerateDevicesErrorMsg* abgefragt werden.

EnumFlags

Name	Wert	Beschreibung
EnumFlag_ComPorts	$01	Alle lokal am Rechner vorhandenen seriellen Ports aufzählen (schließt virtuelle serielle Ports via USB-Adapter ein)
EnumFlag_USB	$02	Alle lokal am Rechner angeschlossenen K+K-USB-Geräte aufzählen (HID + CDC)
EnumFlag_LocalDevices	$03	Serielle Ports und K+K-USB-Geräte
EnumFlag_PCAN	$04	PCAN-USB-Adapter
EnumFlag_All	$FF	alles aufzählen

3.2.2 Multi_GetEnumerateDevicesErrorMsg

function Multi_GetEnumerateDevicesErrorMsg: PAnsiChar; stdcall;

Liefert die bei *Multi_EnumerateDevices* erzeugte Fehlermeldung oder Nullzeiger zurück.

Liefert:

- *nil:* keine Fehlermeldung
- <> *nil:* Fehlermeldung

3.2.3 Multi_GetHostAndIPs

function Multi_GetHostAndIPs(HostName: PAnsiChar; IPaddr: PAnsiChar; ErrorMsg: PAnsiChar): Integer; stdcall;

Bestimmt eigenen Hostnamen und alle IP-Adressen (durch Komma getrennt).

Abweichend zu den anderen Multi_*-Aufrufen genügen hier jeweils 80 Byte Puffergröße

Parameter:

- *HostName:* muss auf einen Puffer mit 80 Byte zeigen, enthält nach Aufruf den eigenen Hostnamen oder 0 im Fehlerfall
- *IPaddr:* muss auf einen Puffer mit 80 Byte zeigen, enthält nach Aufruf die eigenen IP-Adressen durch Komma getrennt oder 0 im Fehlerfall
- *ErrorMsg:* muss auf einen Puffer mit 80 Byte zeigen, enthält 0 oder im Fehlerfall eine Fehlermeldung

Liefert:

- *CKK_DLL_NoError*(1): ok
- *CKK_DLL_BufferTooSmall*(6): ein oder mehrere Rückgabestrings wurde auf 80 Zeichen gekürzt
- *CKK_DLL_Error*(0): Fehler, *ErrorMsg* enthält eine Fehlermeldung

3.3 Pfadangaben

3.3.1 Multi_GetOutputPath

function Multi_GetOutputPath(ID: Integer): PAnsiChar; stdcall;

Liefert aktuellen Dateien-Ausgabepfad (Debug-Protokoll, Testdaten) für Source ID.

Parameter:

- *ID:* Sorce-ID, auf die sich dieser Aufruf bezieht, von *CreateMultiSource* gelieferter Wert

Liefert:

- aktuell gesetzter Ausgabepfad für Source *ID*
- Fehlermeldung "Source-ID <*ID*> not found"

3.3.2 Multi_SetOutputPath

function Multi_SetOutputPath(ID: Integer; Path: PAnsiChar): PAnsiChar; stdcall;

Setzt Dateien-Ausgabepfad (Debug-Protokoll, Testdaten) für Source ID und testet Zugriffsrechte. Voreinstellung ist Pfad der EXE-Datei.

Der Ausgabepfad wirkt sich NUR auf zukünftig zu erstellende Dateien aus, bereits geöffnete Dateien behalten ihren Dateipfad.

Parameter:

- *ID:* Sorce-ID, auf die sich dieser Aufruf bezieht, von *CreateMultiSource* gelieferter Wert
- *Path:* Ausgabepfad, Pfadtrennzeichen wird ggf. angehängt

Liefert:

- *nil:* ok
- <> nil: Fehlermeldung, Ausgabepfad wurde nicht gesetzt

Hinweis:

Um eindeutige Dateiname zu erzeugen, müssen entweder verschiedene Ausgabepfade pro Source-ID gesetzt werden oder verschiedene Source-Kennungen beim Erzeugen der Dateien angegeben werden (siehe *Multi_Debug, Multi_StartSave*Data*).

3.4 Debug-Protokoll

Es kann ein Library-internes Debug-Protokoll erstellt werden, das mit den nachfolgend genannten Funktionen geöffnet, geschlossen und in seinem Umfang konfiguriert werden kann. Das Debug-Protokoll existiert pro Source, d.h. verschiedene Verwaltungs-Objekte schreiben in getrennte Dateien.

3.4.1 Multi_Debug

function Multi_Debug(ID: Integer; DbgOn: Boolean; DbgID: PAnsiChar): PAnsiChar; stdcall;

Öffnet/schließt Library-internes Debug-Protokoll für Source *ID*. Das Debug-Protokoll wird in der Voreinstellung in eine Datei *KK_DLL_LOG_<DbgID>.txt* geschrieben (gleiches Verzeichnis wie die Anwendung, bzw. in das von *Multi_SetOuputPath* vorgegebene Verzeichnis). Eine evt. vorhandene Datei wird überschrieben. Siehe *Multi_DebugLogLimit* für erweiterte Einstellungen des Debug-Protokolls. Wenn die Debug-Datei nicht erzeugt werden konnte (z.B. fehlende Zugriffsrechte), wird eine Fehlermeldung zurückgegeben.

Parameter:

- *ID:* Sorce-ID, auf die sich dieser Aufruf bezieht, von *CreateMultiSource* gelieferter Wert
- *DbgOn: False* schliesst das Protokoll, *True* öffnet es
- *DbgID:* Source-Kennung des Debug-Protokolls, Bestandteil des Dateinamens

Liefert:

- *nil:* ok
- <> *nil:* Fehlermeldung

Hinweis:

DbgID dient der Unterscheidung der Debug-Protokolle zu verschiedenen Sourcen um eindeutige Dateinamen zu generieren. Dies ist insbesondere dann notwendig, wenn die Ausgabeverzeichnisse gleich sind. Wenn nicht mit mehreren Sourcen gearbeitet wird, kann *DbgID* auch nil sein.

3.4.2 Multi_DebugFlags

function Multi_DebugFlags(ID: Integer; ReportLog: Boolean; LowLevelLog: Boolean): Integer; stdcall;

Modifiziert Umfang des Debug-Protokolls für Source *ID*, Voreinstellung: *ReportLog*=True, *LowLevelLog*=False. Die neuen Einstellungen werden sofort wirksam und können im laufenden Betrieb beliebig geändert werden.

Parameter:

- *ID:* Sorce-ID, auf die sich dieser Aufruf bezieht, von *CreateMultiSource* gelieferter Wert
- *ReportLog=True*: von *Multi_GetReport* gelieferte Reports werden ins Debug-Protokoll geschrieben
- *LowLevelLog=True:* Datenstrom, der über die Verbindung gesendet und empfangen wird, wird ins Debug-Protokoll geschrieben.

Liefert:

- *CKK_DLL_NoError*(1): ok
- *CKK_DLL_SourceNotFound*(10): *ID* ist keine gültige Source-ID

3.4.3 Multi_DebugLogLimit

function Multi_DebugLogLimit(ID: Integer; LogType: Byte; aSize: Cardinal): Integer; stdcall;

Setzt Grenzen für das Debug-Protokoll von Source *ID*, Voreinstellung: *LogType*=0, *aSize*=0. Damit diese Einstellungen wirksam werden, müssen sie gesetzt werden **bevor** das Debug-Protokoll mit *Multi_Debug* geöffnet wird.

Parameter:

- *ID:* Sorce-ID, auf die sich dieser Aufruf bezieht, von *CreateMultiSource* gelieferter Wert
- *LogType:* siehe Tabelle unten
- *aSize:* max. Größe des Protokolls in Anzahl Bytes, *aSize*=0: Größe wird auf 20 MB gesetzt

Liefert:

- *CKK_DLL_NoError*(1): ok
- *CKK_DLL_SourceNotFound*(10): *ID* ist keine gültige Source-ID

Werte für LogType

Name	Wert	Beschreibung
logUnlimited	0	Es wird in ein Debug-Protokoll geschrieben. Die Datei ist in ihrer Größe unbegrenzt (Voreinstellung). Der Wert für aSize bleibt unberücksichtigt. Dateiname: KK_DLL_LOG_<*DbgID*>.txt
logOverwrite	1	Das Debug-Protokoll wird in seiner Größe begrenzt (aSize=0: Voreinstellung 20MB). Ist die Größe erreicht, wird das Protokoll von Anfang an überschrieben. Dateiname: KK_DLL_LOG_<*DbgID*>.txt
logCreateNew	2	Das Debug-Protokoll wird in seiner Größe begrenzt (aSize=0: Voreinstellung 20MB). Ist die Größe erreicht, wird eine neue Datei angelegt. Dateiname enthält Datum und Zeitpunkt der Erzeugung (KK_DLL_LOG_<*DbgID*>_<yyyymmdd>_<hhmmss>.txt).

3.4.4 Multi_DebugGetFilename

function Multi_DebugGetFilename(ID: Integer): PAnsiChar; stdcall;

Liefert den absoluten Dateinamen der aktuellen Debug-Protokoll-Datei für Source *ID*.

Parameter:

- *ID:* Sorce-ID, auf die sich dieser Aufruf bezieht, von *CreateMultiSource* gelieferter Wert

Liefert:

- aktuellen Dateinamen; nil, wenn kein Debug-Protokoll geschrieben wird
- Fehlermeldung "Source-ID <*ID*> not found"

3.5 Info-Abfragen

3.5.1 Multi_GetDLLVersion

function Multi_GetDLLVersion: PAnsiChar; stdcall;

Liefert:

- Datum und Version der KK-Library.

3.5.2 Multi_GetBufferAmount

function Multi_GetBufferAmount(ID: Integer): Integer; stdcall;

Gibt Füllstand des Empfangspuffers für Source *ID* in Anzahl Bytes zurück

Parameter:

- *ID:* Sorce-ID, auf die sich dieser Aufruf bezieht, von *CreateMultiSource* gelieferter Wert

Liefert:

- *0:* Puffer ist leer oder Fehler (*ID* ungültig, keine aktive Verbindung)
- *>0:* Anzahl noch nicht gelesener Bytes im Empfangspuffer für Source *ID*

3.5.3 Multi_GetTransmitBufferAmount

function Multi_GetTransmitBufferAmount(ID: Integer): Integer; stdcall;

Gibt Füllstand des Übertragungspuffers im K+K-Gerät zurück, das mit Source *ID* verbunden ist.

Parameter:

- *ID:* Sorce-ID, auf die sich dieser Aufruf bezieht, von *CreateMultiSource* gelieferter Wert

Liefert:

- *0:* Puffer ist leer oder Fehler (*ID* ungültig, keine Verbindung zu einem K+K-Gerät)
- *>0:* Anzahl noch nicht gesendeter Bytes im Übertragungsspuffer des K+K-Gerätes, das mit Source *ID* verbunden ist

Hinweis:

Bei einer seriellen Verbindung kann der Füllstand des Übertragungspuffers nicht ermittelt werden; es wird immer 0 geliefert.

3.5.4 Multi_GetUserID

function Multi_GetUserID(ID: Integer): Byte; stdcall;

Gibt die vom K+K-Gerät zugeteilte User-ID zurück. Siehe dazu *Verbindungsarten*.

Parameter:

- *ID:* Sorce-ID, auf die sich dieser Aufruf bezieht, von *CreateMultiSource* gelieferter Wert

Liefert:

- *1..4:* vom K+K-Gerät zugeteilte User-ID
- *255:* Fehler (*ID* ungültig, keine Verbindung zu einem K+K-Gerät)

3.5.5 Multi_IsFileDevice

function Multi_IsFileDevice(ID: Integer): Boolean; stdcall;

Bei Verarbeitung von Daten aus einer Datei (siehe *Verbindungsarten - Daten aus Datei*) wird hier True zurückgeliefert. Wenn *Multi_GetReport* den Fehler *CKK_DLL_DeviceNotConnected(7)* meldet sollte der Leseprozess beendet werden, da keine Daten mehr geliefert werden.

Parameter:

- *ID:* Sorce-ID, auf die sich dieser Aufruf bezieht, von *CreateMultiSource* gelieferter Wert

Liefert:

- Es werden Messdaten aus einer Datei gelesen

3.5.6 Multi_IsSerialDevice

function Multi_IsSerialDevice(ID: Integer): Boolean; stdcall;

(ab 19.1.2)

Bei einer seriellen Verbindung wird hier True zurückgeliefert.

Hiermit kann geprüft werden, ob *CKK_DLL_NotSupported* bei *Multi_SetNSZCalibrationData , Multi_SetFHRData* eine falsche Firmware-Version bezeichnet oder eine serielle Verbindung.

Parameter:

- *ID:* Sorce-ID, auf die sich dieser Aufruf bezieht, von *CreateMultiSource* gelieferter Wert

Liefert:

- serielle Verbindung

3.5.7 Multi_GetFirmwareVersion

function Multi_GetFirmwareVersion(ID: Integer): Integer; stdcall;

(ab 18.01.10)

Liefert Firmware-Versionsnummer aus einem vorhergehend empfangenen Versions-Report. Ein Versions-Report (Header $7001) wird durch Kommando $01 bzw. $81 angefordert.

Parameter:

- *ID:* Sorce-ID, auf die sich dieser Aufruf bezieht, von *CreateMultiSource* gelieferter Wert

Liefert:

- -1: Fehler: Source-ID ist ungültig oder es wurde noch kein Versions-Report empfangen
- > 1: Firmware-Versionsnummer

3.5.8 Multi_HasFRAM

function Multi_IsFileDevice(ID: Integer): Boolean; stdcall;

(ab 18.01.10)

Liefert K+K-Gerät ist mit einem F-RAM ausgestattet (siehe *NSZ-Kalibrierung*). Dazu ist ein vorhergehend empfangener Versions-Report notwendig. Ein Versions-Report (Header $7001) wird durch Kommando $01 bzw. $81 angefordert.

Parameter:

- *ID:* Sorce-ID, auf die sich dieser Aufruf bezieht, von *CreateMultiSource* gelieferter Wert

Liefert:

- *False:* K+K-Gerät hat kein F-RAM bzw. es liegt kein Versions-Report vor
- *True:* K+K-Gerät hat F-RAM (z.B. für NSZ-Kalibrierungsdaten)

3.6 Verbindung öffnen, schliessen

3.6.1 Multi_OpenConnection

function Multi_OpenConnection(ID: Integer; Connection: PAnsiChar; BlockingIO: Boolean): Integer; stdcall;

Öffnet für Source *ID* die in *Connection* beschriebene Verbindung. Eine Vorgänger-Verbindung für Source *ID* wird geschlossen.

Parameter:

- *ID:* Sorce-ID, auf die sich dieser Aufruf bezieht, von *CreateMultiSource* gelieferter Wert
- *Connection*: siehe *Verbindungsarten*, muss auf einen Puffer mit 1024 Bytes zeigen, enthält im Fehlerfall eine Fehlermeldung
- *BlockingIO=True:* Verbindung mit blockierendem Lesen/Schreiben öffnen

Liefert:

- *CKK_DLL_NoError*(1): ok

  sonst Fehlernummer und *Connection* enthält eine Fehlermeldung:
- *CKK_DLL_Error*(0): Fehler
- *CKK_DLL_SourceNotFound*(10): *ID* ist keine gültige Source-ID

Hinweis *BlockingIO*:

BlockingIO führt zu blockierenden Ein-/Ausgabeoperationen. Das heißt alle Lese-/Schreibaufrufe kehren nach einer Wartezeit (Timeout: ca. 200 ms) zurück, wenn nichts empfangen wurde. **BlockingIO wird NICHT unterstützt bei Daten aus Datei**.

Es sollte mit einem eigenen Lese-Thread gearbeitet werden, um die Anwendung nicht zu blockieren.

Hinweis *(ab 18.2)*:

Ab Firmware 63 kann eine Anwendung bis zu 4 Verbindungen über dieselbe USB-Schnittstelle betreiben.

Unter Linux zu beachten:

Damit unter Linux auf ein lokal angeschlossenes Gerät zugegriffen werden kann, sind bestimmte Zugriffsrechte nötig:

- USB: der Benutzer muss zur Gruppe ***root*** gehören
- Seriell: der Benutzer muss zur Gruppe **dialout** gehören
- Linux-Befehl (dazu sind root-Rechte nötig!) um einen Benutzer einer Gruppe hinzuzufügen: *usermod -a -G <gruppe> <benutzer>*

3.6.2 Multi_CloseConnection

procedure Multi_CloseConnection(ID: Integer); stdcall;

Schliesst aktuelle Verbindung für Source *ID*. Eventuell noch nicht gesendete Kommandos werden gelöscht.

Parameter:

- *ID:* Sorce-ID, auf die sich dieser Aufruf bezieht, von *CreateMultiSource* gelieferter Wert

3.7 Reports einlesen

3.7.1 Multi_SetDecimalSeparator

function Multi_SetDecimalSeparator(ID: Integer; Separator: AnsiChar): Integer; stdcall;

Die KK-Library liefert vom K+K-Gerät empfangene Messwerte (Gleitkommazahlen) als AnsiString zurück. Hier kann das bei der Darstellung einer Gleitkommazahl zu verwendende Dezimal-Trennzeichen für Source *ID* vorgegeben werden. Dieses Dezimal-Trennzeichen wird auch zur Konvertierung von Strings in Gleitkommawerte verwendet.

Parameter:

- *ID:* Sorce-ID, auf die sich dieser Aufruf bezieht, von *CreateMultiSource* gelieferter Wert
- *Separator:* Dezimal-Trennzeichen, andere Werte als '.' oder ',' werden ignoriert

Liefert:

- *CKK_DLL_NoError*(1): ok
- *CKK_DLL_SourceNotFound*(10): *ID* ist keine gültige Source-ID

3.7.2 Multi_SetNSZ

function Multi_SetNSZ(ID: Integer; aNSZ: Integer): Integer; stdcall;

Abhängig von der verwendeten Messkarte werden NSZ-Reports als Single-NSZ oder Double-NSZ übertragen. Hier wird für Source *ID* vorgegeben, wie NSZ-Reports übertragen werden, als Single- oder Double-NSZ.

(ab 18.1.3)

Wert 0 für *aNSZ*: automtische Erkennung des NSZ-Formats

Parameter:

- *ID:* Sorce-ID, auf die sich dieser Aufruf bezieht, von *CreateMultiSource* gelieferter Wert
- *aNSZ:* 0=automatische Erkennung, 1=Single-NSZ, 2=Double-NSZ, andere Werte werden ignoriert

Liefert:

- *CKK_DLL_NoError*(1): ok
- *CKK_DLL_SourceNotFound*(10): *ID* ist keine gültige Source-ID

Hinweis:

Bisherige Aufrufe aus Vorgänger-Handbuch

- *FX_GetReport* erwartet Single-NSZ
- *LE_ReadMonPha* erwartet Double-NSZ

3.7.3 Multi_GetReport

function Multi_GetReport(ID: Integer; Data: PAnsiChar): Integer; stdcall;

Gibt für Source *ID* den nächsten empfangenen Report in *Data* zurück.

Parameter:

- *ID:* Sorce-ID, auf die sich dieser Aufruf bezieht, von *CreateMultiSource* gelieferter Wert
- *Data*: muss auf einen Puffer mit 1024 Bytes zeigen, enthält nach Aufruf den nächsten Report (Messdaten, Nachricht) oder 0, wenn kein Report vorhanden ist oder im Fehlerfall eine Fehlermeldung

Liefert:

- *CKK_DLL_NoError*(1): ok
- *CKK_DLL_BufferTooSmall*(6): Report in *Data* wurde auf 1024 Byte gekürzt

  sonst Fehlernummer und *Data* enthält eine Fehlermeldung:
- *CKK_DLL_Error*(0): Fehler
- *CKK_DLL_WriteError*(3): Fehler bei Kommando-Senden, Verbindung wurde geschlossen. siehe *Multi_SendCommand*
- *CKK_DLL_ServerDownError*(4): Ziel-Server nicht erreichbar, Verbindung wurde geschlossen
- *CKK_DLL_DeviceNotConnected*(7): keine Verbindung vorhanden
- *CKK_DLL_HardwareFault*(9): Peer sendet keine Messdaten, Verbindung wurde geschlossen
- *CKK_DLL_SourceNotFound*(10): *ID* ist keine gültige Source-ID
- *CKK_DLL_CmdIgnored(11)*: Kommando wurde nicht gesendet (Simulator weist Kommando ab)
- *CKK_DLL_Reconnected*(13): Wiederherstellung einer gestörten Verbindung mit Datenverlust

Hinweis:

Bei Fehlern 3 und 4 sollte die Anwendung die Verbindung neu öffnen (*Multi_OpenConnection* erneut aufrufen).

3.8 Kommandos senden

3.8.1 Multi_GetPendingCmdsCount

function Multi_GetPendingCmdsCount(ID: Integer): Cardinal; stdcall;

Gibt Anzahl zwischengespeicherter Kommandos für Source *ID* zurück

Parameter:

- *ID:* SorceID, auf die sich dieser Aufruf bezieht, von *CreateMultiSource* gelieferter Wert

Liefert:

- 0: keine wartenden Kommandos oder Fehler (*ID* ungültig, keine aktive Verbindung)
- > 0: Anzahl zwischengespeicherter Kommandos

3.8.2 Multi_SetCommandLimit

function Multi_SetCommandLimit(ID: Integer; Limit: Cardinal): Integer; stdcall;

Setzt Länge der Kommando-Warteschlange für Source *ID* auf *Limit*. Mit *Limit*=0 ist die Kommando-Anzahl unbegrenzt, Voreinstellung = 0

Parameter:

- *ID:* Sorce-ID, auf die sich dieser Aufruf bezieht, von *CreateMultiSource* gelieferter Wert
- *Limit:* Kommando-Anzahl in der Warteschlange (0=unbegrenzt)

Liefert:

- *CKK_DLL_NoError* (1): ok
- *CKK_DLL_SourceNotFound*(10): *ID* ist keine gültige Source-ID

3.8.3 Multi_SendCommand

function Multi_SendCommand(ID: Integer; Command: PAnsiChar; Len: Integer): Integer; stdcall;

Trägt *Command* in die Kommando-Warteschlange für Source *ID* ein. Wenn keine aktive Verbindung vorhanden ist oder die Kommandoliste voll ist (*Multi_SetCommandLimit*), wird das Kommando abgewiesen (*CKK_CmdIgnored*).

In der Regel wird das Kommando beim nächsten *Multi_GetReport*-Aufruf gesendet und im Fehlerfall dort die entsprechende Fehlermeldung zurückgegeben.

Bei einer Netzwerk-Verbindug wird das Kommando sofort gesendet. Im Fehlerfall wird hier bereits die entsprechende Fehlermeldung zurückgegeben.

(ab 18.2):

Ungültige Kommandos werden mit Fehler *CKK_CmdIgnored(11)* abgewiesen.

Parameter:

- *ID:* Sorce-ID, auf die sich dieser Aufruf bezieht, von *CreateMultiSource* gelieferter Wert
- *Command:* zu sendendes Kommando, im Fehlerfall enthält *Command* eine Fehlermeldung, muss auf einen Puffer mit 1024 Byte zeigen
- *Len:* Kommando-Länge in Bytes

Liefert:

- *CKK_DLL_NoError (1):* ok

  sonst Fehlernummer und *Command* enthält eine Fehlermeldung:
- *CKK_DLL_Error(0):* Fehler
- *CKK_DLL_SourceNotFound(10): ID* ist keine gültige Source-ID
- *CKK_CmdIgnored(11):* Kommando wurde abgewiesen

Hinweis:

Kommandos werden **nicht gesendet** bei Daten aus Datei und bei Verbindung zu einem TCP-Server.

3.8.4 Multi_RemoteLogin

function Multi_RemoteLogin(ID: Integer; Password: LongWord; err: PAnsiChar): Integer; stdcall;

Sofern für Source *ID* eine Netzwerk-Verbindung zum K+K-Gerät besteht, wird *Password* verschlüsselt in einem Remote-Login-Kommando versendet (siehe *Multi_SendCommand*), andernfalls wird die Funktion ohne Fehler beendet.

Parameter:

- *ID:* Sorce-ID, auf die sich dieser Aufruf bezieht, von *CreateMultiSource* gelieferter Wert
- Password: unverschlüsseltes Password, Integer im Bereich 0..4294967295
- err: enthält 0 oder eine Fehlermeldung, muss auf einen Puffer mit 1024 Byte zeigen

Liefert:

- *CKK_DLL_NoError (1):* ok

  sonst Fehlernummer und *err* enthält eine Fehlermeldung:
- *CKK_DLL_Error(0):* Login wurde abgewiesen (falsches Passwort)
- *CKK_DLL_SourceNotFound(10): ID* ist keine gültige Source-ID
- *CKK_CmdIgnored(11):* Kommando wurde abgewiesen

Hinweis:

Remote-Login ist notwendig, sofern über eine Netzwerk-Verbindung der Flash-Speicher bzw. das F-RAM des K+K-Gerätes verändert werden soll.

Beispiel: *Multi_SetNSZCalibrationData*

3.9 Lokaler TCP-Server

3.9.1 Multi_StartTcpServer

function Multi_StartTcpServer(ID: Integer; var aPort: Word): Integer; stdcall;

Startet lokalen TCP-Server für Source *ID* auf TCP-Port *aPort*. Im Fehlerfall kann via *Multi_GetTcpServerError* eine Fehlermeldung abgerufen werden.

Parameter:

- *ID:* Sorce-ID, auf die sich dieser Aufruf bezieht, von *CreateMultiSource* gelieferter Wert
- *aPort:* TCP-Portnummer auf der der Server erreichbar ist, 0: System vergibt Portnummer, diese wird zurückgegeben

Liefert:

- *CKK_DLL_NoError(1):* ok
- *CKK_DLL_SourceNotFound(10): ID* ist keine gültige Source-ID
- *CKK_DLL_Error(0):* Fehler, Fehlermeldung via *Multi_GetTcpServerError* abrufen

Hinweis:

Der TCP-Server ist nur über seine Serverportnummer erreichbar. Sie muss bekanntgemacht werden, damit Clients mit dem TCP-Server Verbindung aufnehmen können.

3.9.2 Multi_StopTcpServer

function Multi_StopTcpServer(ID: Integer): Integer; stdcall;

Stoppt lokalen TCP-Server für Source *ID*. Alle bestehenden Client-Verbindungen werden beendet. Im Fehlerfall kann via *Multi_GetTcpServerError* eine Fehlermeldung abgerufen werden.

Parameter:

- *ID:* Sorce-ID, auf die sich dieser Aufruf bezieht, von *CreateMultiSource* gelieferter Wert

Liefert:

- *CKK_DLL_NoError(1):* ok
- *CKK_DLL_SourceNotFound(10): ID* ist keine gültige Source-ID
- *CKK_DLL_Error(0):* Fehler, Fehlermeldung via *Multi_GetTcpServerError* abrufen

3.9.3 Multi_GetTcpServerError

function Multi_GetTcpServerError(ID: Integer): PAnsiChar; stdcall;

Liefert im TCP-Server gespeicherte Fehlermeldung zurück.

Parameter:

- *ID:* Sorce-ID, auf die sich dieser Aufruf bezieht, von *CreateMultiSource* gelieferter Wert

Liefert:

- *nil:* Es liegt keine Fehlermeldung vor
- *<> nil:* Fehlermeldung

3.9.4 Multi_TcpReportLog

procedure Multi_TcpReportLog(ID: Integer; Data: PAnsiChar; logType: Integer); stdcall;

Logeintrag zur Weiterverteilung an lokalen TCP-Server übergeben. Wenn *ID* keine gültige Source-ID ist oder zu dieser *ID* kein TCP-Server existiert oder *Data* = nil ist, wird der Aufruf ignoriert.

Parameter:

- *ID:* Sorce-ID, auf die sich dieser Aufruf bezieht, von *CreateMultiSource* gelieferter Wert
- *Data:* Logeintrag, der weiterverteilt werden soll
- *logType:* klassifiziert Logeintrag zur Weitergabe an TCP-Clients. Die Clients geben bei der Anmeldung beim TCP-Server einen Reportmodus an, der vorgibt welche Logeinträge an den Client gesendet werden. *logType* und Reportmodus hängen wie folgt zusammen:

logType	Reportmodus
0	PHASELOG
1	FREQLOG
2	PHASEDIFFLOG
3	NSZLOG
4	NSZDIFFLOG
5	PHASEPREDECESSORLOG
6	USERLOG1
7	USERLOG2

3.10 Verbindung zu einem TCP-Server auf LOG-Ebene

3.10.1 Multi_OpenTcpLog

function Multi_OpenTcpLog(ID: Integer; IpPort: PAnsiChar; Mode: PAnsiChar): Integer; stdcall;

Erzeugt einen TCP-Empfänger für den Empfang von Logeinträgen im Reportmodus *Mode* von einem TCP-Server, der mit *IpPort* addressiert wird. Es erfolgt eine Client-Anmeldung beim TCP-Server.

Parameter:

- *ID:* Sorce-ID, auf die sich dieser Aufruf bezieht, von *CreateMultiSource* gelieferter Wert
- *IpPort:* IP-Adresse und Portnummer des TCP-Servers im Format <Ipv4>:<Port>, im Fehlerfall enthält *IpPort* eine Fehlermeldung, muss auf einen Puffer mit 1024 Byte zeigen
- *Mode:* anzumeldender Reportmodus, einer der Strings
    - PHASELOG
    - FREQLOG
    - PHASEDIFFLOG
    - NSZLOG
    - NSZDIFFLOG
    - PHASEPREDECESSORLOG
    - USERLOG1
    - USERLOG2

Liefert:

- *CKK_DLL_NoError(1):* ok

  sonst Fehlernummer und *IpPort* enthält eine Fehlermeldung
- *CKK_DLL_SourceNotFound(10): ID* ist keine gültige Source-ID
- *CKK_DLL_Error(0):* Fehler (z.B. Reportmodus nicht verfügbar)

Hinweis:

Wenn der TCP-Server den Client nicht mehr erreichen kann (Verbindung zwischen Client und Server wurde geschlossen), wird die Client-Anmeldung im TCP-Server gelöscht.

3.10.2 Multi_CloseTcpLog

procedure Multi_CloseTcpLog(ID: Integer); stdcall;

Beendet TCP-Empfänger für Source *ID*. TCP-Empfänger meldet sich beim TCP-Server ab.

Nach dem Schließen können noch evt. vorhandene Daten abgerufen werden (*Multi_GetTcpLog*).

Wenn *ID* keine gültige Source-ID ist oder es zu dieser *ID* keinen TCP-Receiver gibt, wird der Aufruf ignoriert.

Parameter:

- *ID:* Sorce-ID, auf die sich dieser Aufruf bezieht, von *CreateMultiSource* gelieferter Wert

3.10.3 Multi_GetTcpLog

function Multi_GetTcpLog(ID: Integer; Data: PAnsiChar): Integer; stdcall;

Übergibt den nächsten vom TCP-Empfänger empfangenen Logeintrag in *Data*.

Der TCP-Empfänger enthält intern einen Empfangspuffer von 10.000 Einträgen. Sendet der TCP-Server schneller, als die Anwendung ausliest (mit *Multi_GetTcpLog*), kommt es zu einem Datenverlust. Der TCP-Empfänger verwirft empfangene Daten, wenn der Empfangspuffer voll ist.

Parameter:

- *ID:* Sorce-ID, auf die sich dieser Aufruf bezieht, von *CreateMultiSource* gelieferter Wert
- *Data:* muss auf einen Puffer mit 1024 Byte zeigen, enthält nach Aufruf:
  - 0 kein Report vorhanden
  - Report
  - Fehlermeldung im Fehlerfall

Liefert:

- *CKK_DLL_NoError(1):* ok
- *CKK_DLL_BufferTooSmall (6):* Logeintrag in *Data* wurde auf 1024 Bytes gekürzt

  sonst Fehlernummer und *Data* enthält eine Fehlermeldung:
- *CKK_DLL_SourceNotFound(10): ID* ist keine gültige Source-ID
- *CKK_DLL_ServerDownError(4):* TCP-Server hat Verbindung geschlossen
- *CKK_DLL_BufferOverflow(8):* Es hat einen Datenverlust gegeben

3.10.4 Multi_OpenTcpLogTime

function Multi_OpenTcpLogTime(ID: Integer; lpPort: PAnsiChar; Mode: PAnsiChar; Format: PAnsiChar): Integer; stdcall;

(ab 19.00.02)

Wie *Multi_OpenTcpLog* mit dem Unterschied, dass die empfangenen Logeinträge einen Zeitstempel in UTC vorweggestellt haben, der mit dem im Parameter *Format* angegebenen Formatstring formatiert ist.

Parameter:

- *ID:* Sorce-ID, auf die sich dieser Aufruf bezieht, von *CreateMultiSource* gelieferter Wert
- *lpPort:* IP-Adresse und Portnummer des TCP-Servers im Format <Ipv4>:<Port>, im Fehlerfall enthält *lpPort* eine Fehlermeldung, muss auf einen Puffer mit 1024 Byte zeigen
- *Mode:* anzumeldender Reportmodus, siehe *Multi_OpenTcpLog*
- *Format:* Formatierungsangabe für den Zeitstempel

Liefert:

- *CKK_DLL_NoError(1):* ok

  sonst Fehlernummer und *lpPort* enthält eine Fehlermeldung
- *CKK_DLL_SourceNotFound(10): ID* ist keine gültige Source-ID
- *CKK_DLL_Error(0):* Fehler (z.B. Reportmodus nicht verfügbar)

Hinweis:

Wenn der TCP-Server den Client nicht mehr erreichen kann (Verbindung zwischen Client und Server wurde geschlossen), wird die Client-Anmeldung im TCP-Server gelöscht.

Beispiel-Format:

YYYYMMDD HH:NN:SS.ZZZ mit

- YYYY: Jahr 4-stellig
- MM: Monat
- DD: Tag
- HH: Stunde (24-Stunden-Angabe)
- NN: Minuten
- SS: Sekunden
- ZZZ: Millisekunden

3.10.5 Multi_OpenTcpLogType

function Multi_OpenTcpLogType(ID: Integer; IpPort: PAnsiChar; LogType: Integer; Format: PAnsiChar): Integer; stdcall;

(ab 19.02.00)

Analog zu *Multi_OpenTcpLog, Multi_OpenTcpLogTime* mit dem Unterschied, dass der gewünschte Modus als Integer (0..7) angegeben wird.

Sofern Format ungleich nil ist, haben die empfangenen Logeinträge einen Zeitstempel in UTC vorweggestellt.

Parameter:

- *ID:* Sorce-ID, auf die sich dieser Aufruf bezieht, von *CreateMultiSource* gelieferter Wert
- *IpPort:* IP-Adresse und Portnummer des TCP-Servers im Format <Ipv4>:<Port>, im Fehlerfall enthält *IpPort* eine Fehlermeldung, muss auf einen Puffer mit 1024 Byte zeigen
- *LogType:* anzumeldender Reportmodus (0..7)
- *Format:* Formatierungsangabe für den Zeitstempel (nil: ohne Zeitstempel)

Liefert:

- *CKK_DLL_NoError(1):* ok

  sonst Fehlernummer und *IpPort* enthält eine Fehlermeldung
- *CKK_DLL_SourceNotFound(10): ID* ist keine gültige Source-ID
- *CKK_DLL_Error(0):* Fehler (z.B. Reportmodus nicht verfügbar)

Hinweis:

Wenn der TCP-Server den Client nicht mehr erreichen kann (Verbindung zwischen Client und Server wurde geschlossen), wird die Client-Anmeldung im TCP-Server gelöscht.

3.11 AppData an TCP-Server senden

function Multi_TcpAppData(ID: Integer; Data: PAnsiChar): Integer; stdcall;

(ab 19.02.00)

(ab 19.03.00 mit Antwort)

Sendet den in *Data* enthaltenen String an den für Source *ID* verbundenen TCP-Server auf Library-Ebene oder Log-Ebene. Der Empfänger-TCP-Server gibt *Data* beim nächsten GetReport-Aufruf als Report mit Header $7F40 an die Anwendung weiter.

Ab Version 19.3.0 enthält Data nach dem Aufruf die Server-Antwort.

Parameter:

- *ID:* Sorce-ID, auf die sich dieser Aufruf bezieht, von *CreateMultiSource* gelieferter Wert; muss Verbindung zu TCP-Server sein
- *Data:* enthält den zu übertragenden String, muss auf einen Puffer mit 1024 Byte zeigen. Enthält nach Aufruf Server-Antwort oder im Fehlerfall eine Fehlermeldung.

Liefert:

- *CKK_DLL_NoError(1):* ok, *Data* enthält Server-Antwort

  sonst Fehlernummer und *Data* enthält eine Fehlermeldung
- *CKK_DLL_SourceNotFound(10): ID* ist keine gültige Source-ID
- *CKK_DLL_ServerDownError* (4): Es besteht keine Verbindung zu einem TCP-Server
- *CKK_DLL_Error(0):* Vom TCP-Server gemeldeter Fehler

3.12 Testdaten erzeugen

Empfangenen Reports können als Testdaten in Dateien abgespeichert werden, um sie später erneut einzulesen und zu verarbeiten.

Die Testdaten können als binäre Reports (**SaveBinaryData*) oder als lesbare ASCII-Reports (**SaveReportData*) gespeichert werden. Die Dateien werden im Verzeichnis OutputPath (siehe *Multi_SetOutputPath*) angelegt.

Das Einlesen der Testdaten ist als weitere Verbindungsart implementiert. Siehe dazu *Verbindungsarten - Daten aus Datei* zum Öffnen der Verbindung, Lesen wie gewohnt per *Multi_GetReport*.

3.12.1 Multi_StartSaveBinaryData

function Multi_StartSaveBinaryData(ID: Integer; DbgID: PAnsiChar): Integer; stdcall;

Startet das Speichen von empfangenen Reports für Source *ID* in Binärdatei *<Date>_<Time>_<DbgID>_KK_BinaryData.bin* im Verzeichnis OutputPath (siehe *Multi_SetOutputPath*). Im Fehlerfall wird nicht gespeichert.

Parameter:

- *ID:* Sorce-ID, auf die sich dieser Aufruf bezieht, von *CreateMultiSource* gelieferter Wert
- *DbgID:* Source-Kennung der Binärdaten, Bestandteil des Dateinamens

Liefert:

- *CKK_DLL_NoError(1):* ok
- *CKK_DLL_SourceNotFound(10): ID* ist keine gültige Source-ID
- *CKK_DLL_DeviceNotConnected(7):* keine aktive Verbindung

Hinweis:

DbgID dient der Unterscheidung der Binärdaten zu verschiedenen Sourcen um eindeutige Dateinamen zu generieren. Dies ist insbesondere dann notwendig, wenn die Ausgabeverzeichnisse gleich sind. Wenn nicht mit mehreren Sourcen gearbeitet wird, kann *DbgID* auch nil sein.

3.12.2 Multi_StopSaveBinaryData

function Multi_StopSaveBinaryData(ID: Integer): Integer; stdcall;

Beendet das Speichern von binären Reports für Source *ID* und schliesst die Binärdatei.

Parameter:

- *ID:* Sorce-ID, auf die sich dieser Aufruf bezieht, von *CreateMultiSource* gelieferter Wert

Liefert:

- *CKK_DLL_NoError(1):* ok
- *CKK_DLL_SourceNotFound(10): ID* ist keine gültige Source-ID
- *CKK_DLL_DeviceNotConnected(7):* keine aktive Verbindung

3.12.3 Multi_StartSaveReportData

function Multi_StartSaveReportData(ID: Integer; DbgID: PAnsiChar): Integer; stdcall;

Startet das Speichen von gemeldeten Reports für Source *ID* in Textdatei *<Date>_<Time>_<ID>_KK_ReportData.txt* in Verzeichnis OutputPath (siehe *Multi_SetOutputPath*).

Parameter:

- *ID:* Sorce-ID, auf die sich dieser Aufruf bezieht, von *CreateMultiSource* gelieferter Wert
- *DbgID:* Source-Kennung der Reportdaten, Bestandteil des Dateinamens

Liefert:

- *CKK_DLL_NoError(1):* ok
- *CKK_DLL_SourceNotFound(10): ID* ist keine gültige Source-ID

Hinweis:

DbgID dient der Unterscheidung der Reportdaten zu verschiedenen Sourcen um eindeutige Dateinamen zu generieren. Dies ist insbesondere dann notwendig, wenn die Ausgabeverzeichnisse gleich sind. Wenn nicht mit mehreren Sourcen gearbeitet wird, kann *DbgID* auch nil sein.

3.12.4 Multi_StopSaveReportData

function Multi_StopSaveReportData(ID: Integer): Integer; stdcall;

Beendet das Speichern der Reports für Source *ID* und schließt die Textdatei.

Parameter:

- *ID:* Sorce-ID, auf die sich dieser Aufruf bezieht, von *CreateMultiSource* gelieferter Wert

Liefert:

- *CKK_DLL_NoError(1):* ok
- *CKK_DLL_Error(0):* Datei war nicht offen
- *CKK_DLL_SourceNotFound(10): ID* ist keine gültige Source-ID

3.13 NSZ-Kalibrierung

Ab Firmware-Version 62 können Kalibrierdaten im K+K-Gerät gespeichert, abgefragt und geändert werden.

Eine Änderung der Kalibrierdaten via Netzwerk-Verbindung ist nur nach vorherigem Remote-Login möglich. Bei einer seriellen Verbindung wird die Änderung abgewiesen.

Je nach Hardware-Ausstattung, werden die Kalibrierdaten in den Flash-Speicher oder in ein zusätzlich vorhandenes F-RAM geschrieben. Das Vorhandensein eines F-RAMs wird im Versions-Report durch die Kennung „F“ angezeigt, siehe auch *Multi_HasFRAM*.

Änderungen des Flash-Speichers führen immer zu einer Störung der FXE-Messwerte und je nach Timing auch zu einer Störung der NSZ-Messwerte.

Es wird empfohlen, vor einer Änderung der Kalibrierdaten, die Version und Ausstattung des K+K-Gerätes zu überprüfen: Kommando $01 bzw. $81 fordert den Versions-Report $7001 an.

3.13.1 Multi_SetNSZCalibrationData

function Multi_SetNSZCalibrationData(ID: Integer; Data: PAnsiChar): Integer; stdcall;

(ab 18.01.10)

Überträgt NSZ-Kalibrierdaten über Source-Verbindung *ID* (ausgenommen serielle Verbindung) an das K+K-Gerät, wo sie im Flash bzw. F-RAM abgespeichert werden. Das K+K-Gerät informiert alle User mittels $7901-Report über geänderte Kalibrierdaten.

Data enhält die Kalibrierwerte pro Kanal (maximal 24 Kanäle) durch Semikolon getrennt. Die Kalibrierwerte müssen in Nanosekunden angegeben werden (Gleitkommastring mit maximal 3 Nachkommastellen).

Parameter:

- *ID:* Sorce-ID, auf die sich dieser Aufruf bezieht, von *CreateMultiSource* gelieferter Wert
- *Data*: Gleitkommastring mit maximal 3 Nachkommastellen in Nanosekunden pro Kanal, durch Semikolon getrennt

Liefert:

- *CKK_DLL_NoError(1):* ok
- *CKK_DLL_Error(0):* Formatfehler Data: mehr als 24 Kanäle oder Konvertierung in Float schlägt fehl, siehe *Multi_SetDecimalSeparator*
- *CKK_DLL_SourceNotFound(10): ID* ist keine gültige Source-ID
- *CKK_DLL_NotSupported(12):* Firmware-Version < 62, serielle Verbindung

Hinweis:

Repräsentiert *ID* eine Netzwerk-Verbindung, so muss sich der Aufrufer zuvor durch ein Remote-Login authentifiziert haben (siehe *Multi_RemoteLogin*). Andernfalls wird die Änderung nicht übernommen und es gibt keinen $7901-Report als Antwort.

Achtung:

Änderungen des Flash-Speichers führen immer zu einer Störung der FXE-Messwerte und je nach Timing auch zu einer Störung der NSZ-Messwerte.

3.14 FHR-Einstellungen

Ab Firmware-Version 67 können FHR-Einstellungen im K+K-Gerät gespeichert, abgefragt und geändert werden.

Eine Änderung der FHR-Einstellungen via Netzwerk-Verbindung ist nur nach vorherigem Remote-Login möglich. Bei einer seriellen Verbindung können die FHR-Einstellungen zwar gelesen, aber nicht geändert werden.

Je nach Hardware-Ausstattung, werden die FHR-Einstellungen in den Flash-Speicher oder in ein zusätzlich vorhandenes F-RAM geschrieben. Das Vorhandensein eines F-RAMs wird im Versions-Report durch die Kennung „F“ angezeigt, siehe auch *Multi_HasFRAM*.

Änderungen des Flash-Speichers führen immer zu einer Störung der FXE-Messwerte und je nach Timing auch zu einer Störung der NSZ-Messwerte.

Es wird empfohlen, vor einer Änderung der FHR-Einstellungen, die Version und Ausstattung des K+K-Gerätes zu überprüfen: Kommando $01 bzw. $81 fordert den Versions-Report $7001 an.

3.14.1 Multi_ReadFHRData

function Multi_ReadFHRData(ID: Integer): Integer; stdcall;

(ab 19.1.2)

Fordert FHR-Einstellungen für Source-Verbindung *ID* vom Gerät an.

Die Daten werden im Erfolgsfall in einem $7902-Report gemeldet.

Parameter:

- *ID:* Sorce-ID, auf die sich dieser Aufruf bezieht, von *CreateMultiSource* gelieferter Wert

Liefert:

- *CKK_DLL_NoError(1):* ok
- *CKK_DLL_Error(0):* Kommando fehlgeschlagen
- *CKK_DLL_SourceNotFound(10): ID* ist keine gültige Source-ID
- *CKK_DLL_NotSupported(12):* Firmware-Version < 67

3.14.2 Multi_SetFHRData

function Multi_SetFHRData(ID: Integer; Data: PAnsiChar): Integer; stdcall;

(ab 19.1.2)

Überträgt FHR-Einstellungen über Source-Verbindung *ID* (ausgenommen serielle Verbindung) an das K+K-Gerät, wo sie im Flash bzw. F-RAM abgespeichert werden. Das K+K-Gerät informiert alle User mittels $7902-Report über geänderte FHR-Einstellungen.

Data enhält die FHR-Einstellungen pro Kanal (maximal 24 Kanäle) durch Schrägstrich getrennt. Pro Kanal Nominal-Frequenz, LO-Frequenz und zugelassen durch Semikolon getrennt. Die Frequenzwerte müssen in Hz angegeben werden (Gleitkommastring). Zugelassen ist entweder 0 oder 1.

Parameter:

- *ID:* Sorce-ID, auf die sich dieser Aufruf bezieht, von *CreateMultiSource* gelieferter Wert
- *Data*: Gleitkommastring mit FHR-Einstellungen pro Kanal (maximal 24 Kanäle)

Liefert:

- *CKK_DLL_NoError(1):* ok
- *CKK_DLL_Error(0):* Format-Fehler
- *CKK_DLL_SourceNotFound(10): ID* ist keine gültige Source-ID
- *CKK_DLL_NotSupported(12):* Firmware-Version < 67, serieller Verbindung

Hinweis:

Repräsentiert *ID* eine Netzwerk-Verbindung, so muss sich der Aufrufer zuvor durch ein Remote-Login authentifiziert haben (siehe *Multi_RemoteLogin*). Andernfalls wird die Änderung nicht übernommen und es gibt keinen $7902-Report als Antwort.

Achtung:

Änderungen des Flash-Speichers führen immer zu einer Störung der FXE-Messwerte und je nach Timing auch zu einer Störung der NSZ-Messwerte.

4. Funktionsdeklarationen in C

Alle Versionen der KK-Library exportieren die gleichen Funktionen.

Alle Funktionen haben die Aufrufkonvention **stdcall**:

- Parameterübergabe von rechts nach links
- Rückgabewerte in Register
- Aufgerufene Funktion bereinigt den Stack. Daher sind keine variablen Argumentlisten möglich.

Es werden folgende Integer-Größen vorausgesetzt:

- int: 4 Byte
- short: 2 Byte
- char: 1 Byte
- bool: 1 Byte

Bei allen Parametern vom Typ *char* * muss der Aufrufer 1024 Byte Puffer zur Verfügung stellen.

Hinweis:

Für Python auf Linux-Systemen stehen KK-Library-Versionen mit Aufrufkonvention **cdecl** zur Verfügung: libkk_library_32_cdecl.so und libkk_library_64_cdecl.so.

4.1 Mehrfach-Verbindung anlegen

- int **CreateMultiSource**(void);

4.2 Verfügbare Schnittstellen auflisten

- int **Multi_EnumerateDevices**(char *Names, unsigned char EnumFlags);
- char * **Multi_GetEnumerateDevicesErrorMsg**(void);
- int **Multi_GetHostAndIPs**(char *HostName, char *IPaddr, char * ErrorMsg);

Hinweis:

Char-Parameter bei **Multi_GetHostAndIPs**: es genügen 80 Byte Puffer

4.3 Pfadangaben

- char * **Multi_GetOutputPath**(int ID);
- char * **Multi_SetOutputPath**(int ID, char *path);

4.4 Debug-Protokoll

- char * **Multi_Debug**(int ID, bool DbgOn, char *DbgID);
- int **Multi_DebugFlags**(int ID, bool ReportLog, bool LowLevelLog);
- int **Multi_DebugLogLimit**(int ID, unsigned char LogType, unsigned int aSize);
- char* **Multi_DebugGetFilename**(int ID);

4.5 Info-Abfragen

- char * **Multi_GetDLLVersion**(void);
- int **Multi_GetBufferAmount**(int ID);
- int **Multi_GetTransmitBufferAmount**(int ID);
- unsigned char **Multi_GetUserID**(int ID);
- bool **Multi_IsFileDevice**(int ID);
- bool **Multi_IsFileDevice**(int ID);
- int **Multi_GetFirmwareVersion**(int ID);
- bool **Multi_HasFRAM**(int ID);

4.6 Verbindungen öffnen, schliessen

- int **Multi_OpenConnection**(int ID, char *Connection, bool BlockingIO);
- void **Multi_CloseConnection**(int ID);

4.7 Reports einlesen

- int **Multi_SetDecimalSeparator**(int ID, char Separator);
- int **Multi_SetNSZ**(int ID, int aNSZ);
- int **Multi_GetReport**(int ID, char *Data);

4.8 Kommandos senden

- unsigned int **Multi_GetPendingCmdsCount**(int ID);
- int **Multi_SetCommandLimit**(int ID, unsigned int Limit);
- int **Multi_SendCommand**(int ID, char *Command, int Len);
- int **Multi_RemoteLogin**(int ID, unsigned int Password, char *err);

4.9 Lokaler TCP-Server

- int **Multi_StartTcpServer**(int ID, unsigned short *aPort);
- int **Multi_StopTcpServer**(int ID);
- char * **Multi_GetTcpServerError**(int ID);
- void **Multi_TcpReportLog**(int ID, char *Data, int logType);

4.10 Verbindung zu einem TCP-Server auf LOG-Ebene

- int **Multi_OpenTcpLog**(int ID, char *IpPort, char *Mode);
- int **Multi_OpenTcpLogTime**(int ID, char *IpPort, char *Mode, char *Format);
- int **Multi_OpenTcpLogType**(int ID, char *IpPort, int LogType, char *Format);
- void **Multi_CloseTcpLog**(int ID);
- int **Multi_GetTcpLog**(int ID, char *Data);

4.11 AppData an TCP-Server senden

- int **Multi_TcpAppData**(int ID: Integer; char *Data);

4.12 Testdaten erzeugen

- int **Multi_StartSaveBinaryData**(int ID, char *DbgID);
- int **Multi_StopSaveBinaryData**(int ID);
- int **Multi_StartSaveReportData**(int ID, char *DbgID);
- int **Multi_StopSaveReportData**(int ID);

4.13 NSZ-Kalibrierung

- int **Multi_SetNSZCalibrationData**(int ID, char *Data);

4.14 FHR-Einstellungen

- int **Multi_ReadFHRData**(int ID);
- int **Multi_SetFHRData**(int ID, char *Data);

5. Funktionsdeklarationen in Java

Für die Einbindung der KK-Library in Java steht die Wrapper-Klasse *KK_Library* im Package **brendes.jna-1.7.jar** zur Verfügung. Weiterhin sind die Packages **jna-4.2.2.jar** und **jna-platform-4.2.2.jar** notwendig.

Das Package **brendes.jna-1.7.jar** enthält Javadoc und Source-Klassen und wird hier nicht näher beschrieben.

Bei Interesse wenden Sie sich bitte an K+K GmbH.

Abhängigkeiten der Java-Packages zur Version der KK-Library:

Package	Mindest-Version KK-Library
brendes.jna-1.0.jar	KK_FX80E Version 16.04
brendes.jna-1.1.jar	KK_FX80E Version 16.05
brendes.jna-1.2.jar	KK_FX80E Version 16.10
brendes.jna-1.3.jar	KK_FX80E Version 18.00
brendes.jna-1.4.jar	KK_FX80E, KK_Library_64 Version 18.01.10
brendes.jna-1.5.jar	KK_FX80E, KK_Library_64 Version 19.00.02
brendes.jna-1.6.jar	19.01.02
brendes.jna-1.7.jar	19.02.00
brendes.jna-1.8.jar	19.03.00

6. Funktionsdeklarationen in Python

Für die Einbindung der KK-Library in Python steht die Wrapper-Klasse *NativeLib* im Package **kklib** zur Verfügung.

Weiterhin gibt es Beispielprogramme **LogReceiver** und **ReportReceiver**, die die Verwendung von *NativeLib* demonstrieren.

Alle Python-Module werden im Quelltext zur Verfügung gestellt, eine weitere Beschreibung entfällt an dieser Stelle.

Bei Interesse wenden Sie sich bitte an K+K GmbH.

Abhängigkeiten des Python Package kklib zur Version der KK-Library:

Version Package kklib	Mindest-Version KK-Library
1.0	18.0
1.0.1	18.01.05
1.0.2	18.01.06
1.0.3	18.01.10
1.0.4	18.02.04
1.1.0	19.00.02
1.2	19.01.02
1.3	19.02.00
1.4	19.03.00