

Tutorial Implementing a K+K application

Version 1.0 - 2025-05-07

These instructions describe how a PC application can be created that receives measurement data from a K+K device with the help of K+K library.

history

date	who	Library version	description
25-05-07	Loryn	21.0.0	first create

contents

history.....	2
1. Overview.....	4
1.1 K+K Library.....	4
1.2 Program languages.....	4
2. Program structure.....	5
2.1 Link K+K library.....	5
2.2 CreateMultiSource.....	6
2.3 Configurations.....	6
2.4 Open connection.....	6
2.5 Processing loop.....	7
2.6 Close connection.....	7
Appendix A Code samples.....	8
A.1 Link KK-Library.....	8
A.2 CreateMultiSource.....	8
A.3 Configurations.....	8
A.4 Open connection.....	9
A.5 Loop.....	9
A.6 Close connection.....	11
Appendix B Downloads.....	12

1. Overview

These instructions describe how to implement a PC application that communicates with a K+K device with the help of K+K library.

1.1 K+K Library

The KK library is available in different versions depending on the operating system (Windows, Linux), architecture (32-bit, 64-bit) and calling convention (stdcall, cdecl).

The K+K library is primarily used for communication with K+K devices to receive measurement data and messages from the device and to send control commands to the device.

Furthermore, the data received by the device can be distributed to other applications via the server or accessed by servers.

The generation of test data and processing of data from files is also supported.

Several parallel connections are possible with the KK library: to different devices or several connections to one device.

A detailed description of all functions can be found in *Manual_KK_Library_MultiSource*. It is highly recommended that you familiarize yourself with this guide.

The various versions of K+K library and manual are available for download on website kplusk-messtechnik.de/downloads

1.2 Program languages

The K+K library is coded in Pascal. The functional description in *Manual_KK_Library_MultiSource* is in Pascal notation with a summary of all calls in C.

A C header file *kk_library.h* is available as an aid.

There are corresponding wrapper classes for C++, Java and Python.

- *KK_Library_cpp_x.x.zip* for C++
- *brendes.jna_x.x.jar* for Java
- *Python_Modules_x.x.zip*

(x.x = version number).

2. Program structure

Regardless of whether an application with a graphical user interface or a console application is to be created, the following tasks must be carried out:

1. Link K+K ILibrary
2. Call of *CreateMultiSource* per connection.
3. Configurations
4. Open Connection(s)
5. Send commands, read in and process reports in a processing loop
6. End loop and close connection (s)

You may find code samples for each section in appendix A.

2.1 Link K+K library

The appropriate library name must be selected depending on the operating system (Windows, Linux, Linux ARM), architecture (32-bit, 64-bit) and calling convention (stdcall, cdecl)

- calling convention **stdcall**:
 - **KK_FX80E.DLL** for Windows 32-bit
 - **KK_Library_64.dll** for Windows 64-bit
 - **libkk_fx80e.so** for Linux 32-bit
 - **libkk_library_64.so** for Linux 64-bit
 - **libkk_library_arm32.so**: for Linux 32-bit with ARM-Controller
 - **libkk_library_aarch64.so**: für Linux 64-bit with ARM-Controller
- calling convention **cdecl**:
 - **libkk_library_32_cdecl.so** for Linux 32-bit
 - **libkk_library_64_cdecl.so** for Linux 64-bit
- **libkk_library_arm32_cdecl.so**
- **libkk_library_aarch64_cdecl.so**

The wrapper classes provided select the correct KK library.

2.2 CreateMultiSource

Per connection *CreateMultiSource* has to be called. This creates a new administration object within the library. *CreateMultiSource* returns a source ID which must be specified for all subsequent calls and which identifies this multi-connection. The administration objects are automatically released when the application ends.

2.3 Configurations

Per each source ID configurations should be done, before connection is opened.

The Method *Multi_GetReport* returns measurement reports (floating point values) as ASCII strings. The decimal separator to be used must be set for this: *Multi_SetDecimalSeparator*.

Set output path for files created by K+K library: *Multi_SetOutputPath*. This applies to the debug log and test data. Test data are useful if the PC application should be tested always with the same input data in order to find errors.

2.4 Open connection

First of all, you have to decide for each source ID which connection type is to be selected:

- local via cable connected K+K device
- via network to K+K device
- via network to K+K server
- Data from file
- Simulated data

For details please see chapter *Connection types* in *Manual_KK_Library_MultiSource*.

K+K server must have been started by another application (e.g. FXX_App) as local TCP server.

Open connection via *Multi_OpenConnection* or *Multi_OpenTcpLog* for connections to K+K server on LOG level.

The transfer of received reports to a local TCP server takes place automatically within the library. All you have to do is start a local TCP server.

The application is responsible for forwarding reports at LOG level.

2.5 Processing loop

For applications with a graphical user interface or with several parallel connections, the processing loop should run in a separate thread.

In general, commands are sent and reports are received and processed in the loop for each source ID. An overview of commands and report strings can be found in *FXE_commands_and_reports.pdf*.

At the beginning the version command should be sent with deletion of the transmission buffer (\$81) and the response report (\$7001) should be waited for.

2.6 Close connection

Depending on how the connection was opened, it is closed via *Multi_CloseConnection* or *Multi_CloseTcpLog*.

Appendix A Code samples

The code samples in Python uses wrapper class *NativeLib* (see *kklib.py*) and are excerpts from, *tutorial_1.py*.

tutorial_2.py processes binary reports.

Code samples in C++ are in *tutorialMain_1.cpp* and *tutorialMain_2.cpp*.

A.1 Link KK-Library

```
# load library
try:
    _kknative = NativeLib()
except NativeLibError as exc:
    # stop execution
    sys.exit(str(exc))
print("K+K library version "+_kknative.get_version())
```

A.2 CreateMultiSource

```
# get source ID
_source_id = _kknative.get_source_id()
```

A.3 Configurations

```
# output path: subdir "Debug"
_path = "."+os.sep+"Debug"
kkres = _kknative.set_output_path(_source_id, _path)
if kkres.result_code != ErrorCode.KK_NO_ERR:
    print("set_output_path: unexpected Error")
    print("set_output_path: "+kkres.data)
else:
    print("set_output_path: ok")

# decimal separator, needed to convert floating point to string
# decimal separator english
_dec_sep = "."
# decimal separator german
_dec_sep = ","
kkres = _kknative.set_decimal_separator(_source_id, _dec_sep)
if kkres.data != None:
    print("set decimal separator failed: "+kkres.data)
```

A.4 Open connection

```
# default IP
_connection = "192.168.178.98" # firewall settings needed!

# Blocking I/O -> get_report returns with data or after timeout
_blocking = True
```



```
print("open connection "+_connection)
kkres = _kknative.open_connection(_source_id, _connection, _blocking)
if kkres.result_code != ErrorCode.KK_NO_ERR:
    # stop execution
    sys.exit("open_connection failed: "+kkres.data)
```

A.5 Loop

```
# define some commands
l = [0x81]
cmdVersion = bytes(l) # firmware version, clear buffer
l = [0x41]
cmdPhaseAvg = bytes(l) # report mode phase average
l = [0x43]
cmdFreqAvg = bytes(l) # report mode frequency average
l = [0x29]
cmdRate1s = bytes(l) # report interval 1s
l = [0x26]
cmdRate100ms = bytes(l) # report interval 100ms
l = [0x34]
cmdChannel4 = bytes(l) # limit reports to 4 channels

# loop
class LoopStateE(Enum):
    lsSendVersionCmd = 0,
    lsWaitVersion = 1,
    lsWait7020 = 2,
    lsSendModeCmd = 3,
    lsSendIntervalCmd = 4,
    lsGetReports = 5

loop_state = LoopStateE.lsSendVersionCmd

# Limit report counts: end while-get_report-loop
_empty_string_limit = 50
_report_limit = 200

# receive reports and check command results
print("get reports...")
end = False
emptyStringCounter = 0
reportCounter = 0
```

```
while (not end):
    if loop_state == LoopStateE.lsSendVersionCmd:
        # send command get version and clear buffer
        kkres = _kknative.send_command(_source_id, cmdVersion)
        if kkres.result_code != ErrorCode.KK_NO_ERR:
            print("send version command failed: "+kkres.data)
            end = True
        else:
            loop_state = LoopStateE.lsWaitVersion
            print("version command sent, wait for version string")
    elif loop_state == LoopStateE.lsSendModeCmd:
        # send mode command
        kkres = _kknative.send_command(_source_id, cmdPhaseAvg)
        if kkres.result_code != ErrorCode.KK_NO_ERR:
            print("send mode command failed: "+kkres.data)
            end = True
        else:
            print("mode command sent")
            loop_state = LoopStateE.lsWait7020
    elif loop_state == LoopStateE.lsSendIntervalCmd:
        # send interval command
        kkres = _kknative.send_command(_source_id, cmdRate1s)
        if kkres.result_code != ErrorCode.KK_NO_ERR:
            print("send interval command failed: "+kkres.data)
            end = True
        else:
            print("interval command sent")
            loop_state = LoopStateE.lsWait7020
    else:
        # read report
        kkres = _kknative.get_report(_source_id)
        if kkres.result_code != ErrorCode.KK_NO_ERR:
            print("get_report failed: "+kkres.data)
            end = True
        elif kkres.data is None:
            # no data
            emptyStringCounter += 1
            if emptyStringCounter > _empty_string_limit:
                end = True
                print("no more data -> stop")
            else:
                print("no data emptyCount=", emptyStringCounter)
                time.sleep(0.1) # 100ms
        else:
            print(kkres.data)

            # reset counter
            emptyStringCounter = 0

            #report header
            subS = kkres.data[:4]
            aHeader = int(subS, 16) # HEX!
            # ignore empty message (7000)
            if aHeader == 0x7000:
                # should not occur, as 7016 is set
                print("ignore empty report")
```

```
        continue

    # count report
    reportCounter += 1

    if loop_state == LoopStateE.lsWaitVersion:
        if aHeader == 0x7001:
            print("version received")
            loop_state = LoopStateE.lsSendModeCmd
            fw = _kknative.get_firmware_version(_source_id)
            print("Firmware version "+str(fw))
    elif loop_state == LoopStateE.lsWait7020:
        if aHeader == 0x7020:
            subS = kkres.data[7:]
            print("new header="+subS)
            # phase average expected
            if subS[0] != '1':
                print("switch mode to phase average")
                loop_state = LoopStateE.lsSendModeCmd
            # 1s expected
            elif subS[1] != '9':
                print("switch interval to 1s")
                loop_state = LoopStateE.lsSendIntervalCmd
            # expected header
            else:
                print("expected header received")
        # else LoopStateE.lsGetReports

    if reportCounter > _report_limit:
        end = True
        print('more than', _report_limit, 'reports received, stop now')
```

A.6 Close connection

```
print("close connection")
_kknative.close_connection(_source_id)
```

Appendix B Downloads

On website kplusk-messtechnik.de/downloads actual software versions, samples code as well as PDF documents in either german or english are available for download.

Software:

- KK Library versions for Windows / Linux / Linux ARM in 32 and 64 bit
- KK FXX Graphic user interface for Windows and Linux. Displays data from various connections in text and graphics in different windows. Extensive logging, generation of test data, includes demo mode. Displays firmware configuration and offers configuration modification.
- KK Uploader tool for firmware updates and firmware configuration.
- Wrapper classes for C++, Python and Java with sample code.

- Firmware files for FXE and FXE2

Documents:

- Manuals KK Multi Source Library in german and english
- FXX manual
- List of FXE commands and reports
- Getting started
- Special features for operation under Linux
- Tutorials
- Hardware manuals FXE, FDI, ...