

Katana Native Interface Documentation [KNI]

Table Of Contents

Introduction.....	2
Structure Overview.....	2
How To Start.....	3
Notes.....	3
Error handling.....	3
Documentation.....	4
Configuration File.....	4
Dependencies.....	4
Bug reporting.....	5
Support.....	5
Compiling KNI.....	5
Linking applications against KNI.....	5
Python Module.....	5
License.....	5

Introduction

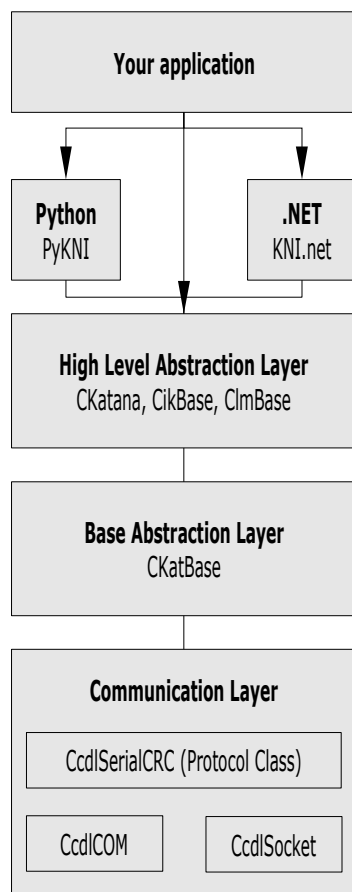
The *Katana Native Interface* (KNI) is a software library which enables you to write controlling software for Katana robots produced by Neuronics AG.

The KNI is written in the C++ programming language. It makes use of the following features: Classes, Inheritance, Overloading, Templates and Exceptions.

The library gives you an easy method to develop your own software for Katana without need to fiddle with the underlying protocols.

Structure Overview

The library uses three layers. The following chart will give you an overview about the dependencies between them:



The “Communication Layer” shields the direct communication to the robot. It consists of a serial communication abstraction (CcdICOM) and a protocol class (CcdSerialCRC) which handles the CRC calculation. (There is also an old protocol implementation available: The SerialZero protocol.)

If you want to send command packages (as described in the Katana Manual) directly to the firmware, this is the point to start.

The “Base Abstraction Layer” consists of the following classes: CKatBase, CMotBase, CSctBase and a large variety of structures to represent and handle internal status of the Katana, the Motor- and Sensor-Controllers. All available firmware commands are implemented here.

The “High Level Abstraction Layer” gives you methods like “MoveRobotTo(...cartesian coordinates here...)” and “freezeRobot()”. This is what most users want. Corresponding classes are: CKatana, CInvKin, CLMBase.

In addition to the C++ library, there are several ways to connect your application to the library:

- Via SWIG (Simplified Wrapper and Interface Generator, www.swig.org) you can use KNI from several scripting and programming languages on different operating systems. Currently Python is the only supported language (tested on Windows and Linux).
- On the Windows platform, there is an ActiveX control, which can be used from different programming languages like C++, VB and Delphi.

How To Start

The best way to start programming using the KNI is to study the demo applications. The „kinematics“ demo is probably the most advanced one and makes use of nearly every high-level method.

In general, you can start like this:

```
Sample Code

#include „kniBase.h“

int main(int argc, char* argv[]) {

    int port = 1; // use „port = 0“ for first port on *nix/Linux/OSX
    // Parameters for serial port device, do not change
    TCdlCOMDesc ccd = {port, 57600, 8, 'N', 1, 300, 0};
    CCdlCOM *device = new CcdlCOM(ccd);

    CCplSerialCRC *protocol = new CcplSerialCRC; // instance of protocol class
    protocol->init(device);

    // You can also instantiate CKatana or CikBase if you don't need
    // linear movements or don't want to move the robot by coordinates
    // Configuration files are provided with the package,
    // make sure you choose the right one for your robot model.
    CLMBase* katana = new CLMBase;
    katana->create(„path to your configuration file“, protocol);

    // If you didn't get any exceptions up to this point, you're ready to go.
    // Next step will probably be the calibration (only needed after reset):
    katana.calibrate();

    // ...

    // ... and a movement
    katana.moveRobotTo(x, y, z, phi, theta, psi);

    /* cleanup */

}
```

Notes

Error handling

Exceptions are used throughout the library. This means that methods which return more than one value or return nothing at all are declared as 'void'. All methods can throw exceptions if not documented as “non-fail guarantee provided”.

Example 1

```
cout << katana.getNumberOfMotors() << endl;    // will not fail since this is a
                                                local state which involves no
                                                communication with the robot
```

Example 2

```
freezeRobot();                                // can throw exceptions in the
                                                Communication Layer since it
```

```
involves communication with the
robot
```

To get an error-string from the exceptions, just catch them as `std::exception` and use the `what()`-method. To get error-codes, catch the exceptions as `KNI::Exception` and use the method `error_number()`. If you want a full traceback, read the documentation to the `libebt` library on libebt.sourceforge.net on how to do it.

The `error_number()` method can be used to integrate the KNI in an application without exceptions.

Documentation

To get more information – detailed function lists for example – read the generated HTML documentation included in the distribution. This is the most up-to-date information about the KNI, since it is being generated automatically (using `doxygen`) before distribution.

Configuration File

Configuration files are provided with the package. They are named like “`katanaXMyyyZ.cfg`”, where **X** is the number of motors (5 or 6), **yyy** the configuration of the last segment ('90' or '180') and **Z** the configuration of the last motor ('G' for Gripper and 'T' for Turner). Make sure you choose the right one. Make sure you also read the Katana Manual if unsure which one to use.

You can change the following settings in the configuration file (not conclusive):

- Order in which the motors are calibrated
- z-Rotation of the coordinate system
- Gripper settings
- Position for a motor after calibration
- Segment lengths
- Angle offset and range

Settings you should not change, are (not conclusive):

- The dynamic and static motor limits
- Encoder-related values
- The model name
- Rotation directions and calibration rotation directions
- Slave ID's, address, motor count

Dependencies

The KNI uses the following libraries:

- `boost` from www.boost.org
- `STL` (Standard Type Library)

Besides that, you need the following applications to start hacking the KNI:

- a decent C++ compiler, KNI has been tested with the following compilers:
 - GCC 3.4 and 4.0 (on Linux and Windows)
 - VS.NET 7.1 and 8.0 (VS.NET2003 and 2005)
 - Note: other compilers are not supported
- `doxygen` from www.stack.nl/~dimitri/doxygen to generate the detailed documentation
- the `Graphviz` package from www.graphviz.org/ to generate the class dependency graphs

Bug reporting

If you discover a problem, we need some information to be able to support you. Please make sure you include the following points in your mail:

- Steps you did before you encountered the error
- The version of KNI you are using
- The model-name of your Katana and the configuration (and/or serialnumber)
- Description of the application and/or environment
- The name and version of compiler, operating system and architecture (x86, amd64, ppc)
- The complete error message. This can be:
 - Compiler output, warning messages
 - Debugger output
 - Screenshots
- If you changed anything that came with the standard distribution, please attach a patch, the file itself or a description
- If possible, also include some lines of your code which triggered the error

Support

Compiling KNI

KNI has been tested using the compilers mentioned above. There is no guarantee that it compiles with other compilers. Specifically VC++ 6.0 is not supported (because of the lacking support for standard C++ features), although you can still use the pre-compiled libraries.

Linking applications against KNI

Due to the nature of C++, you can not use all compilers to link against the pre-compiled libraries (Windows users only, Linux users have to compile the library anyway).

Visual C++ 6.0 has been reported to link successfully against the pre-compiled binaries. The library should be recompiled if using MinGW. There is in general no guarantee, that you can link against the library using any other compilers than the ones KNI is built with.

Boost Dependencies under Windows

The KNI libraries come already pre-built for Windows with every release. If you want to modify the Linear Movement library code, you need to install Boost for your platform and tell the compiler where the Boost libraries are to be found. An easy way is to download an installer for Boost, for example from

www.boost-consulting.com/products/free

In order to let the compiler find the libraries, you need to:

1. Right-click on the Library-LM in the solution explorer of Visual Studio -> Properties
2. Add the Boost Path to the list In C++ -> General -> Additional Include Directories, for instance C:\Programme\boost\boost_1_34_1, depending on where you installed Boost to.

Python Module

The python module depends on the python-version it is compiled with. For compilation, the latest stable version of python will be used. This is, by the time of writing 2.4.x. Please contact us, if you have a different version of python or problems compiling the module.

License

The KNI is distributed under the terms of the GNU General Public License (NOT the GNU Library General Public License). This has the direct consequence that you have to distribute your application under the terms of the GNU GPL, too. Please make sure to read

<http://www.opensource.org/licenses/gpl-license.php> for more information.

Should this be a serious problem for your application/project, please contact support@neuronics.ch.