Software Specification

G-API

Quickstart Guide
Version 2.1 (r8844)

GÖPEL electronic

Get the total Coverage!

GÖPEL electronic GmbH
Göschwitzer Str. 58/60
D-07745 Jena
Tel.: +49-3641-6896-597
Fax: +49-3641-6896-944
E-Mail: ats_support@goepel.com
http://www.goepel.com

G-API: Quickstart Guide

Version 2.1 (r8844)
Published 04/28/2021

Table of Contents

List of Figures

Chapter 1 Introduction

1.1 Introduction to the G-API

G–API stands for **G**oepel electronic **A**pplication **P**rogramming **I**nterface. It is the basic interface for accessing and configuring **GOEPEL electronic** devices.

1.1.1 Features of the G-API

Hardware abstraction
: Uniform addressing of different hardware (e.g. USB 3060 and PXI 3060) via the same API functions and libraries

Logical Interface Names
: Access to hardware interfaces via logical names

Parallel access

- of one application to several hardware interfaces
- of several applications to different hardware interfaces
- or of several applications to the same hardware interface

Asynchronous access

- Non-blocking execution of commands
- Evaluation of responses via callback functions

1.1.2 System requirements

System requirements:

- Microsoft Windows® 7
- Microsoft Windows® 10

1.1.3 System structure

The **G–API** consists of a multi-layer DLL hierarchy and a service that manages device access. The structure is shown in Figure 1.1, "G-API architecture". The user can access **G–API** commands over the DLLs by using the provided library and header files.

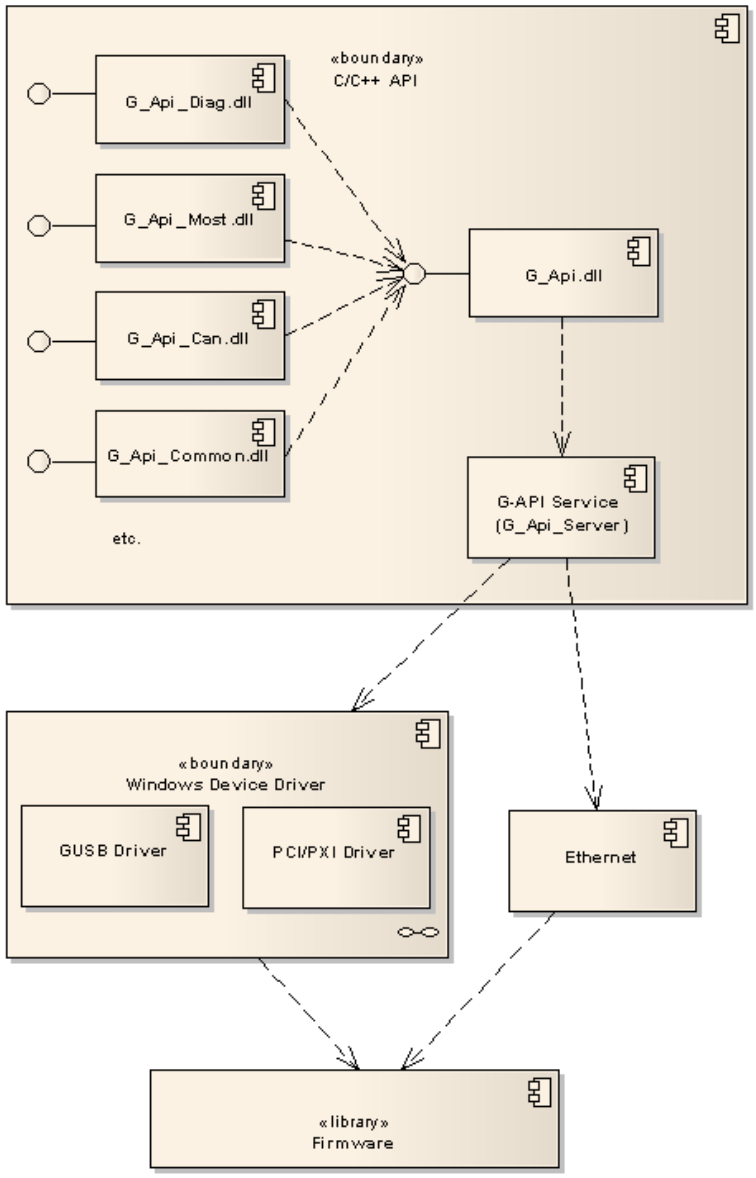

Figure 1.1 G-API architecture

1.1.4 Directory structure

During installation, the following directories are created as subdirectories in the installation path:

```
\bin ❶
\doc ❷
\explorer ❸
\samples ❹
```

❶ *bin:* This directory contains all header and library files *(*.h, *.lib)* and the **G–API** service *G_Api_Server.exe*.
❷ *doc:* Contains this quickstart guide and the reference manual.
❸ *Explorer:* Contains all files belonging to the **G–API** explorer.
❹ *samples:* This is the directory containing example applications as C source code.

Chapter 2 Installation

This chapter describes the installation process of the **G–API** under Microsoft Windows®.

For installing **G–API**, administrator rights are needed.

After starting the installer *g-api-Setup-*.exe* (* stands for the version number), follow the instructions given by the installer.

Figure 2.1 Installer Welcome

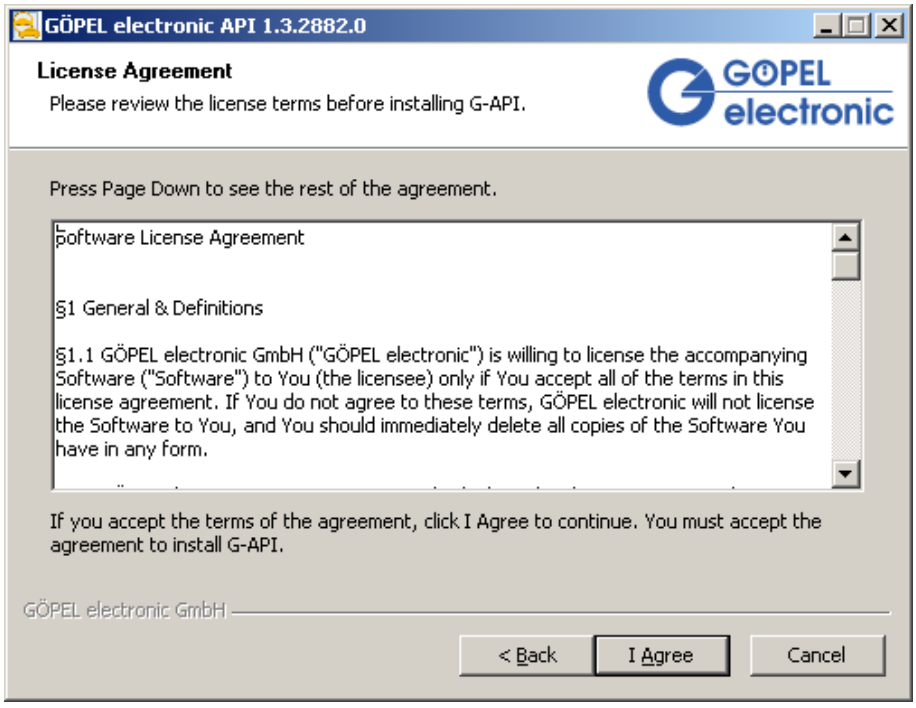

Figure 2.2 License terms

Please read the license terms and click on ***I AGREE*** for confirming that you have read and do agree to the licence terms.

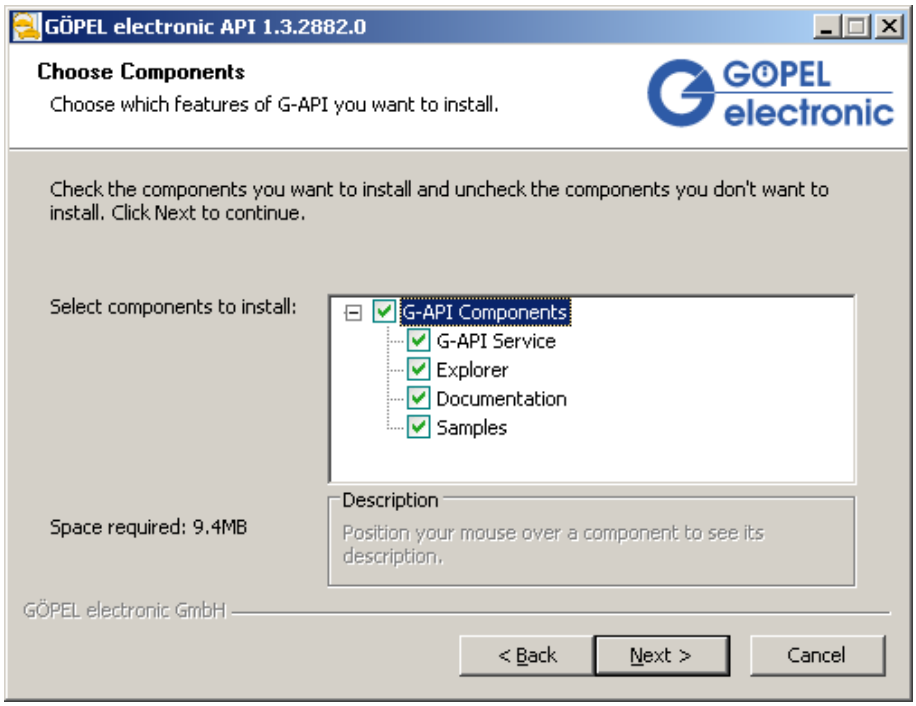

Figure 2.3 Component selection

Select the components to be installed. It is recommended to keep all components selected.

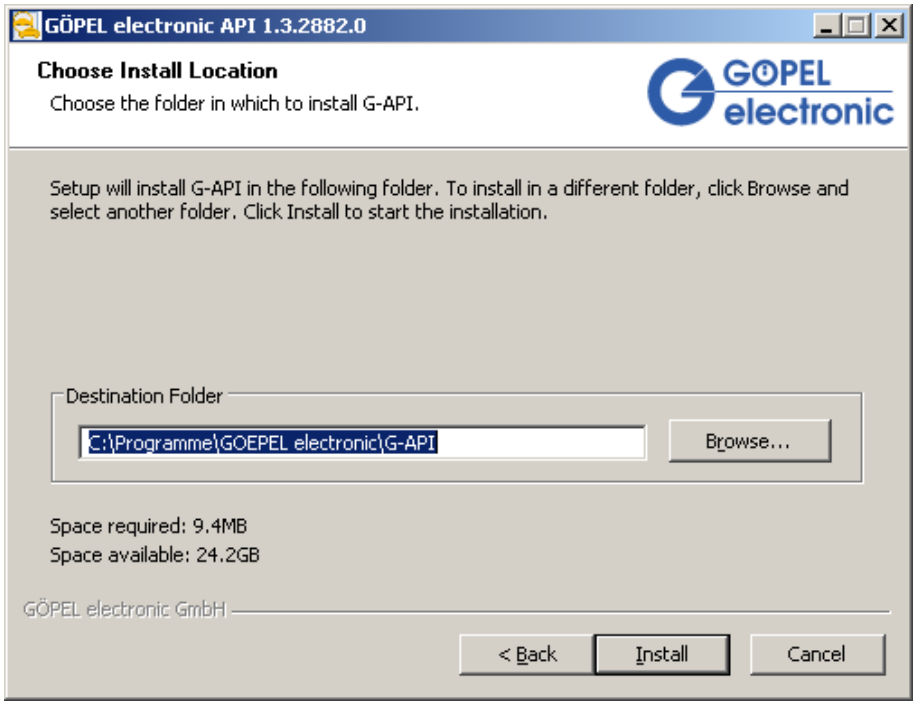

Figure 2.4 Installation directory

Select a target path or confirm the default installation directory.

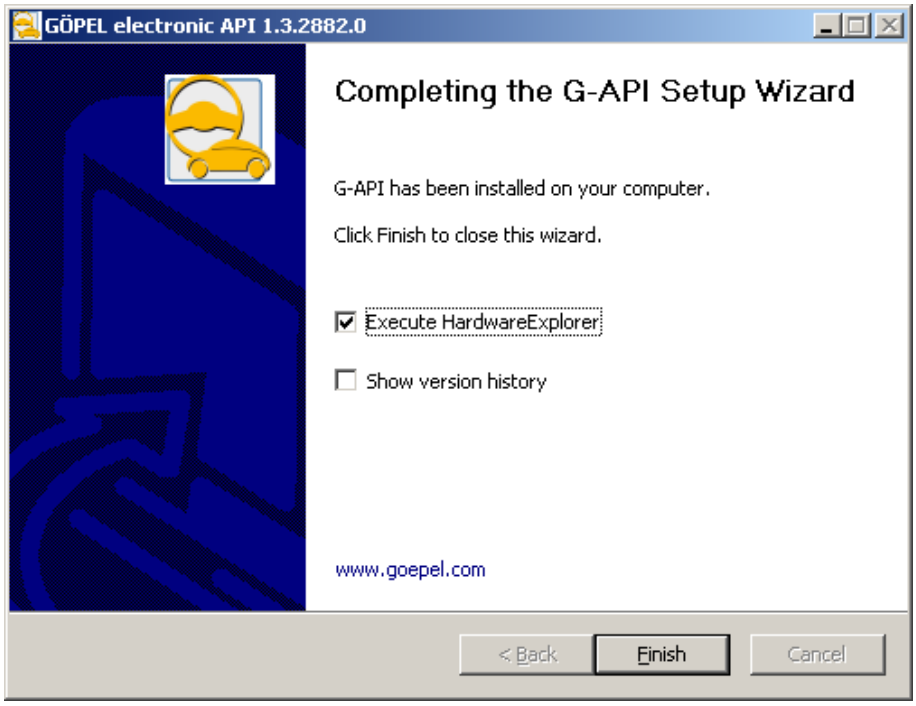

Figure 2.5 Finish

In the last dialog of the installer, you have the option to start the **HardwareExplorer** and show the latest changes and additions in the **Version History** .

Make your choice and complete the installation by clicking ***FINISH***.

The installer can also be run from the command line. It provides several switches for customizing the installation:

- **/?** : shows the options dialog
- **/S** : silent install
- **/L** : create an installation log file (*install.log*) in the installation directory
- **/INSTDIR="C:\MyInstallDir"** : installation directory
- **/NODESKTOPICONS** : do not create desktop icons

Usage: *G-API-Setup-x.y.z.exe [/?] [/S] [/L] [/INSTDIR="C:\MyInstallDir"] [/NODESKTOPICONS]*

Chapter 3 HardwareExplorer

3.1 Introduction

The **GOEPEL electronic HardwareExplorer** is a tool for configuring **GOEPEL electronic** devices. It provides a graphical representation of available devices and offers possibilities for testing and configuring the device interfaces.

At each start, the HardwareExplorer scans the system for connected **GOEPEL electronic** devices. All devices that could be found are displayed in the tree on the left hand side of the screen, while further information about the selected device or interface is diplayed on the right hand side.

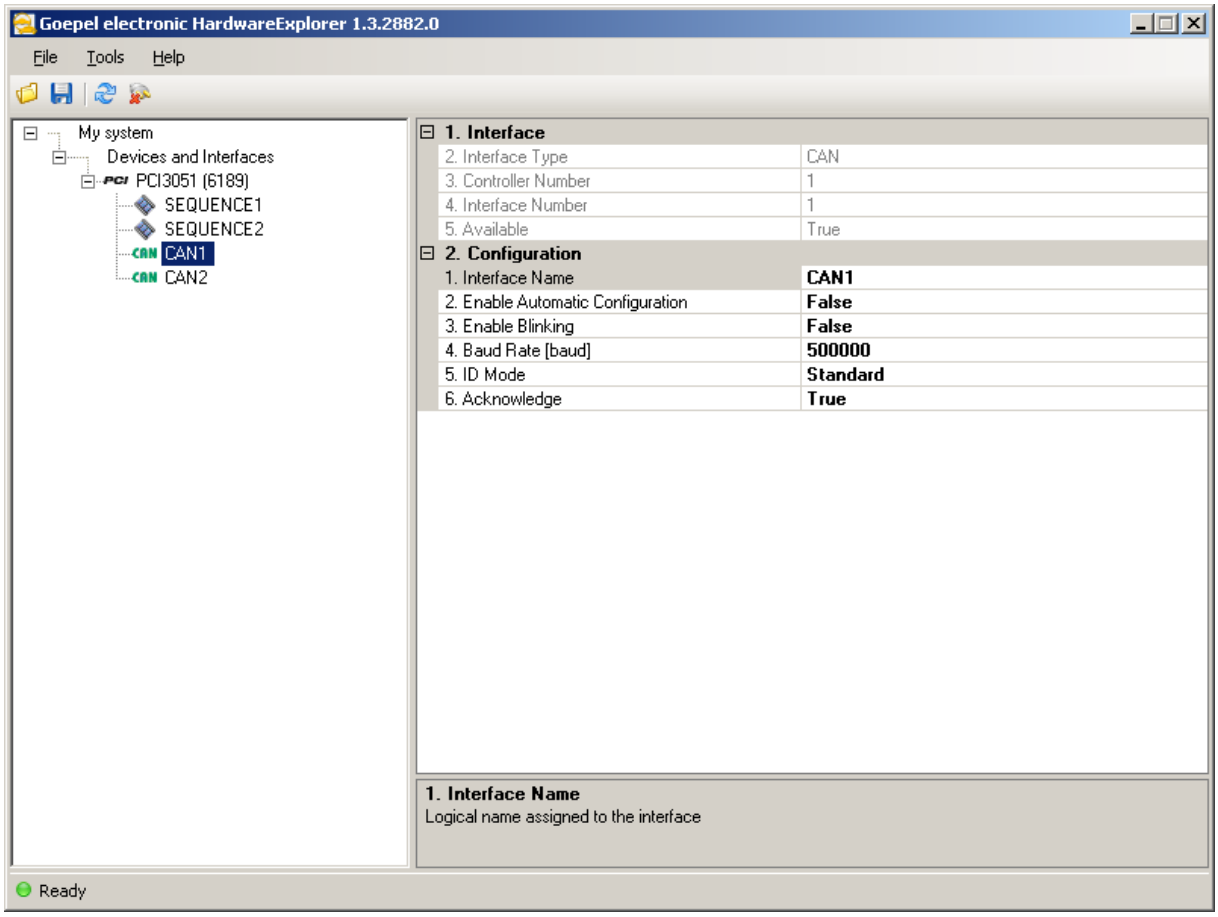

Figure 3.1 HardwareExplorer - Main Screen

Every device is listed with its type, followed by its serial number in braces. Interfaces belonging to this device are listed right below with a symbol signalizing the interface type followed by the logical name of the interface.

When a device is disconnected from the system, it will appear greyed out. Nevertheless, its interface configuration can be edited and is stored.

For maintaining clarity, all disconnected devices can be hidden from the device list by pressing  in the toolbar above the device list.

If you wish to delete all disconnected devices from the device list, go to ***Tools*** → ***Remove Disconnected Devices*** . Attention! The configuration data for all interfaces of disconnected devices will be erased!

By default the HardwareExplorer scans for PCI, PXI, USB and Ethernet devices. The scan for Ethernet devices will include every IP address of every network adapter that was detected. If this behaviour is not desired, you can select which network adapter IP addresses should be scanned under ***Tools*** → ***Preferences*** .

3.2 System Configuration

The topmost configuration page shows the system configuration. It consists of the system name (the name of the computer per default) a system description and information about the **GOEPEL electronic firmware package** .

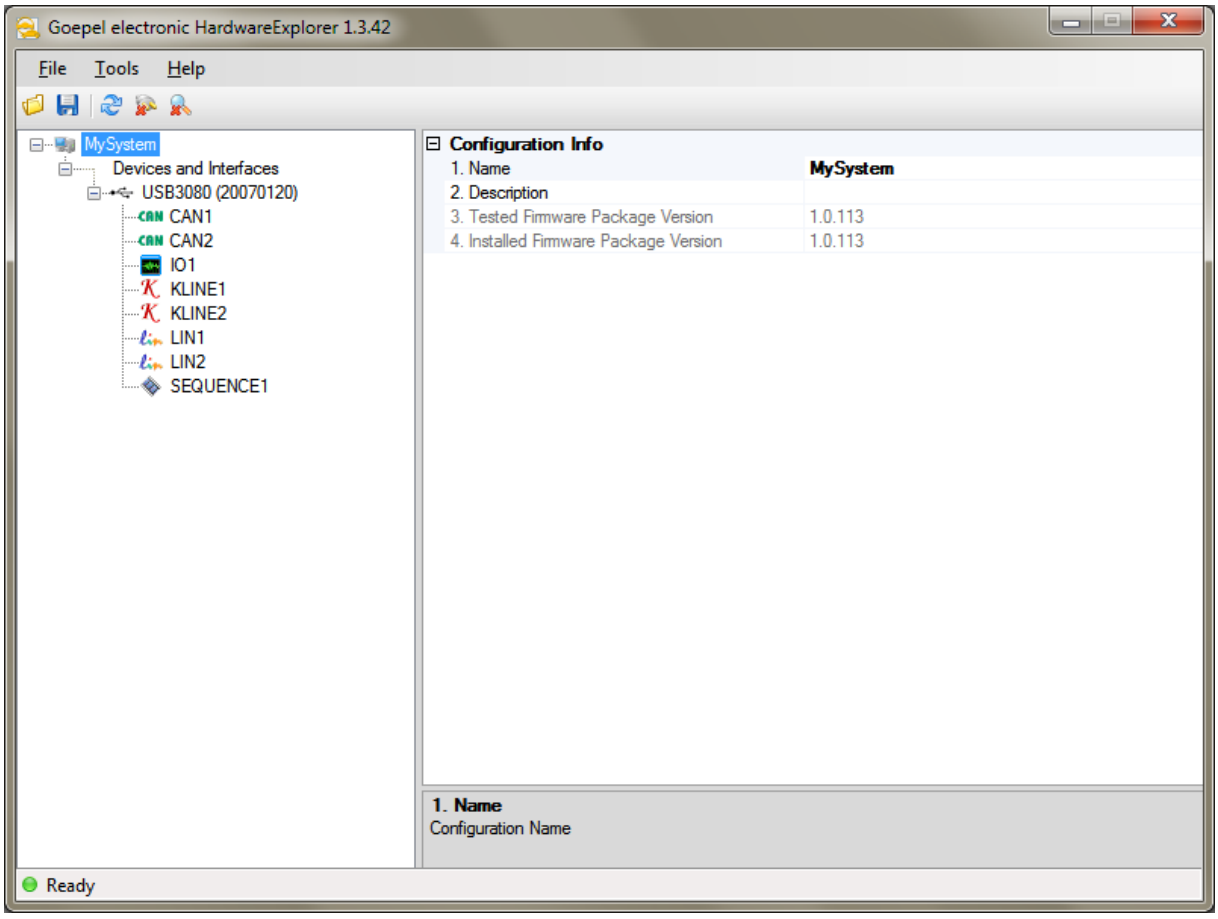

Figure 3.2 System Configuration

A **GOEPEL electronic firmware package** contains a set of firmware binary files for several devices and is available as a separate installer.

3.3 Device Configuration

When a device is selected in the left part of the window, the device configuration window is shown. It contains hardware and firmware information and furthermore you can select whether this device shall use a dedicated firmware version or not. To use this feature, a **GOEPEL electronic** firmware package has to be installed that matches the firmware package the **G–API** was tested with.

If enabled, the **G–API** will check for the dedicated firmware version everytime a port is opened to an interface of the device. If the firmware does not match the tested firmware, it will automatically be flashed to the dedicated version.

Information about the installed and needed firmware package can be found in the Section 3.2, "System Configuration" section.

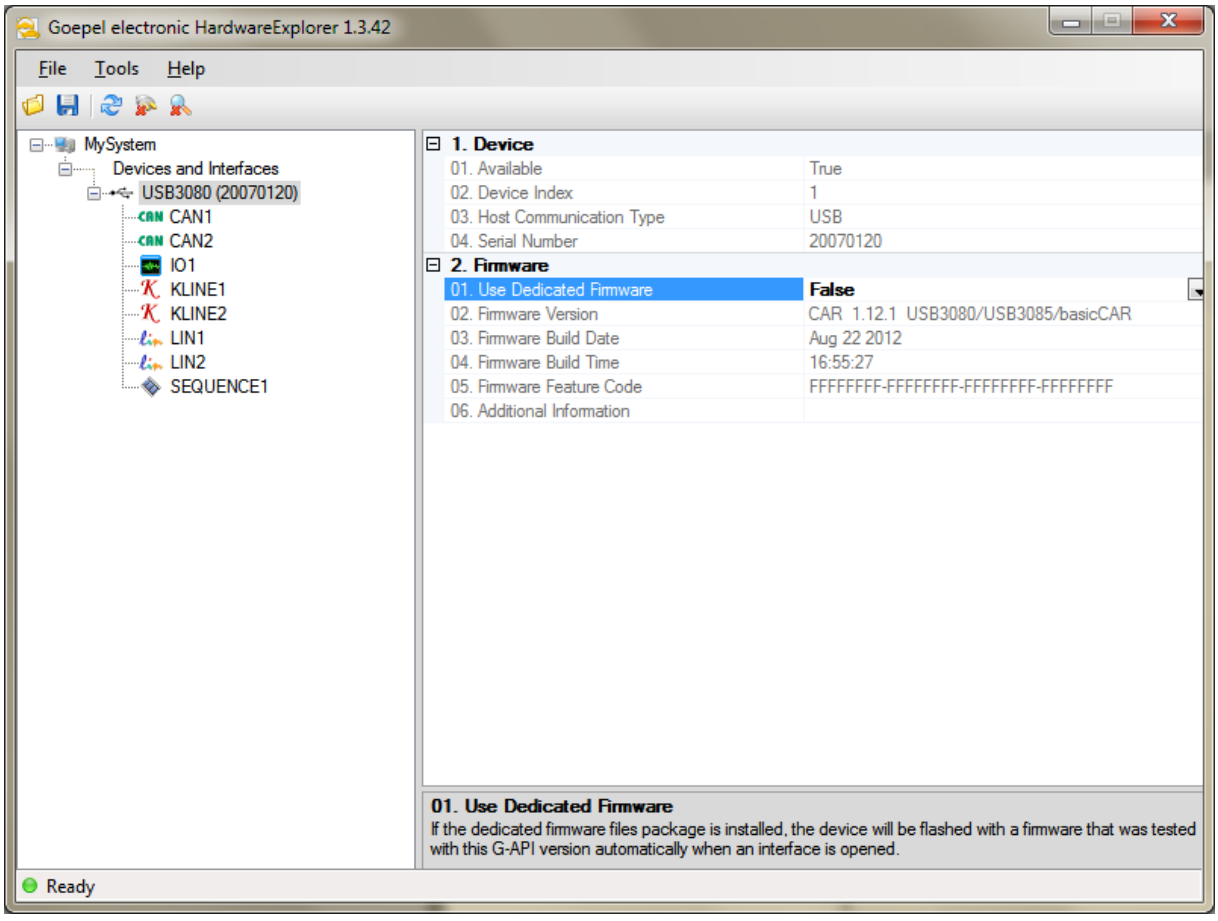

Figure 3.3 System Configuration

3.4 Interface Configuration

By selecting an interface in the device list, the interface properties are shown on the right hand side of the HardwareExplorer. Depending on the interface type, a part of the available properties may vary.

Property values displayed in **bold** letters can be edited.

3.4.1 Common Properties

These properties are available for every interface type.

Interface Type
Type of the seleceted interface

Controller Number
Number of the controller the interface is implemented on

Interface Number
Number of the selected interface

Since every controller may have multiple interfaces, this value represents the interface index for the controller the interface is implemented on.

Available
Interface availability

True - Interface is available

False - Interface is currently not available (disconnected)

Interface Name
Logical name assigned to the interface

This name is used to uniquely identify the interface within your applications. Use this name when connecting to the interface with **G–API** command `G_Common_OpenInterface`.

Enable Automatic Configuration
Enable automatic configuration

If set to **True** , the interface configuration will be applied to the interface when calling **G–API** command `G_Common_OpenInterface`. The interface is configured with the defined property values.

For example, if you want to use a CAN interface with a baud rate other than default (= 500000 baud), you have to enter the desired baud rate value under **Baud Rate** and set **Enable Automatic Configuration** to **True** . The next time you call `G_Common_OpenInterface` with the interface name of the selected interface, the changed baud rate value will be applied.

For compatibility reasons, this feature is set to **False** by default, which means that the values you entered are not applied to the interface automatically. When using the **Automatic Configuration Feature** of the HardwareExplorer, make sure that this value is set to **True** .

3.4.2 CAN Interface Configuration

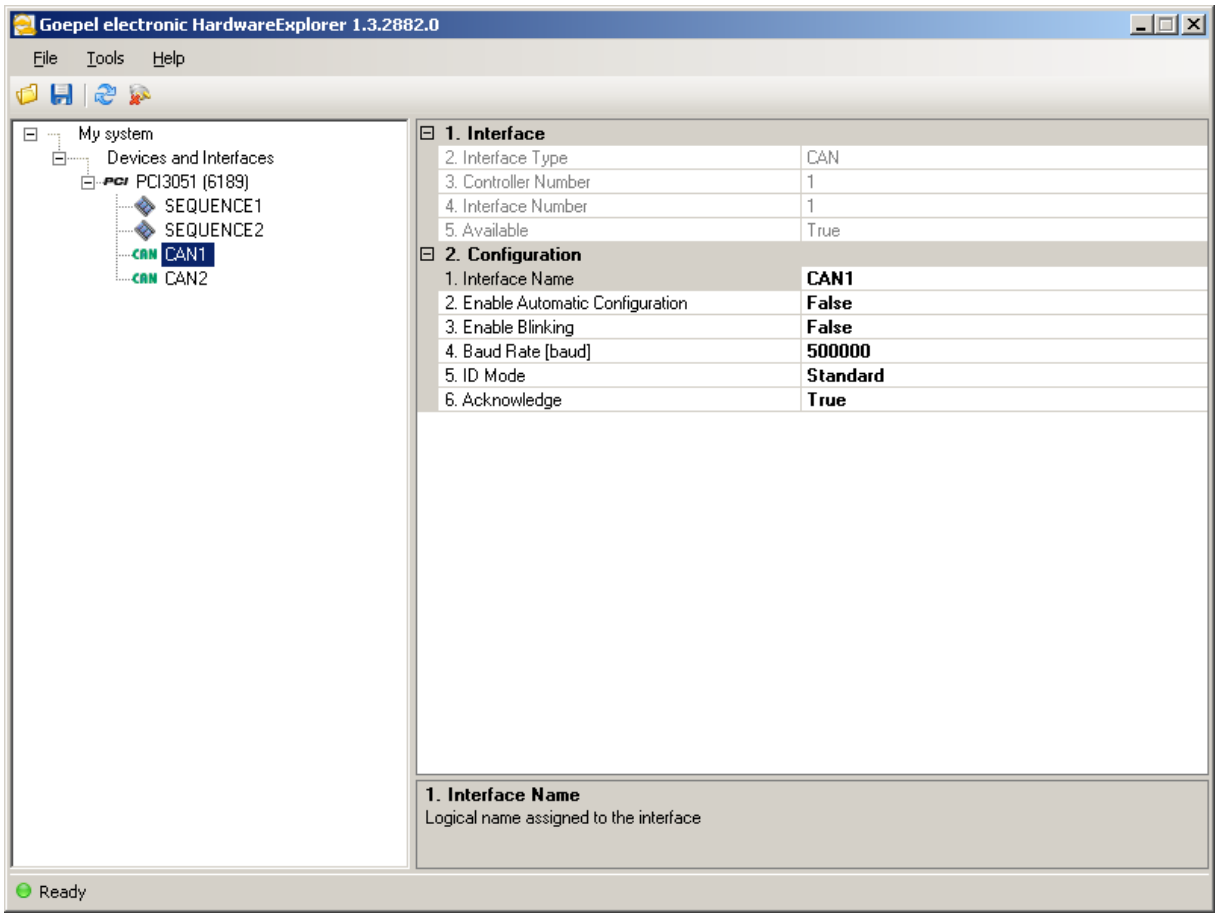

Figure 3.4 CAN Interface Properties

Enable Blinking
If set to **True** , an LED of the device will blink periodically. This feature is mainly used for debug purposes or for identifying a device in a multi-device-setup.

Baud Rate [baud]
Baud Rate value in baud

IDMode
CAN Identifier mode

The following values are possible:

`Standard`
Only Standard identifiers (11 Bit) are used

`Extended`
Only extended identifiers (29 Bit) are used

`Mixed`
Use standard (11 Bit) and extended (29 Bit) identifiers

Acknowledge
If set to **False** , no acknowledge will be sent if a CAN frame is received. This feature is used for invisibly tracing bus communication.

If set to **False**, the interface cannot transmit any data.

3.4.3 LIN Interface Configuration

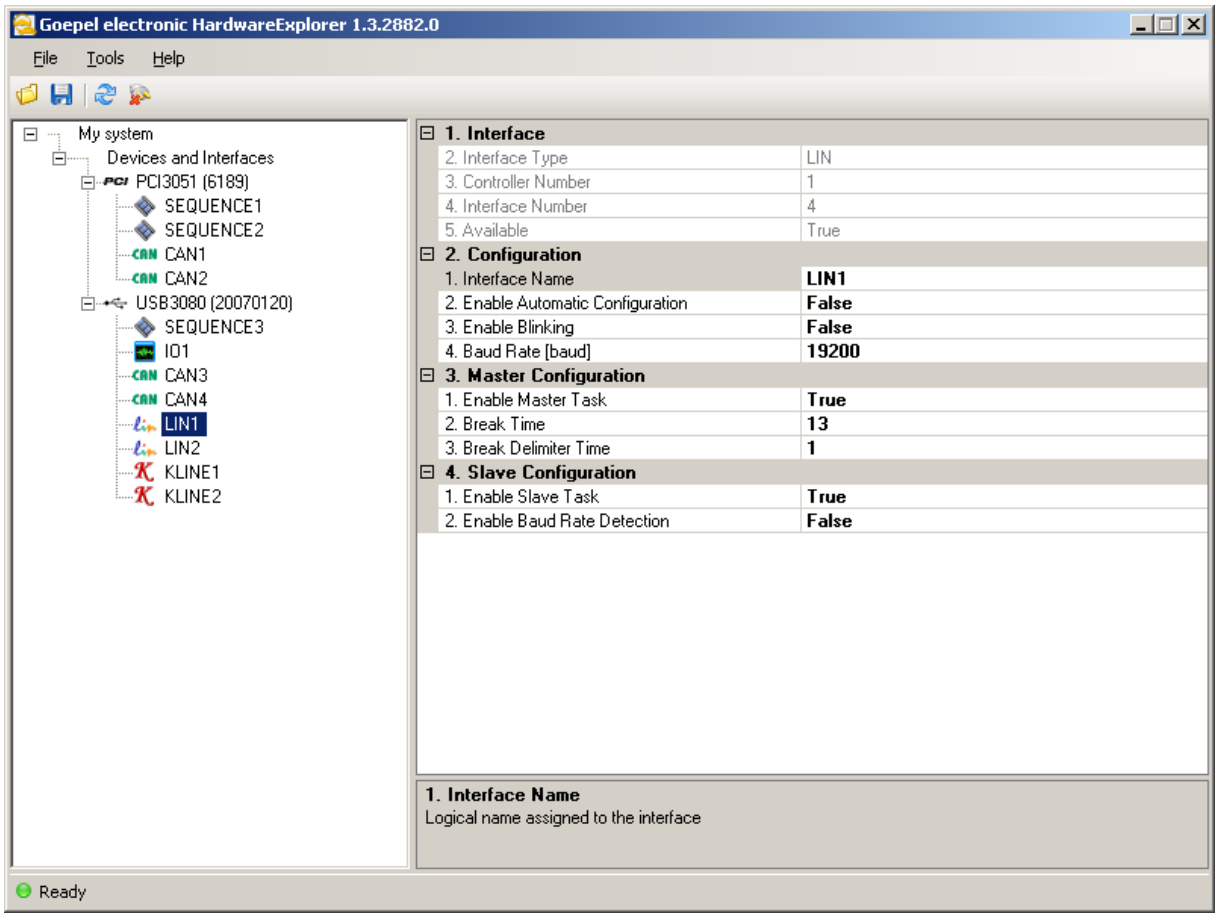

Figure 3.5 LIN Interface Properties

Enable Blinking
If set to **True**, an LED of the device will blink periodically. This feature is mainly used for debug purposes or for identifying a device in a multi-device-setup.

Baud Rate [baud]
Baud Rate value in baud

Enable Master Task
If set to **True**, the interface acts as the LIN Master and is able to send frame headers.

Break Time
Duration of the 'Break' that signals the start of a new LIN frame (in bit times).

Break Delimiter Time
Time between the end of the 'Break' and the beginning of the start bit (in bit times)

Enable Slave Task
If set to **True**, the interface is able to receive and to respond to frame headers.

Enable Baud Rate Detection
If set to **True**, the slave task will detect the baud rate automatically by analyzing the 'SyncByte'.

3.4.4 K-Line Interface Configuration

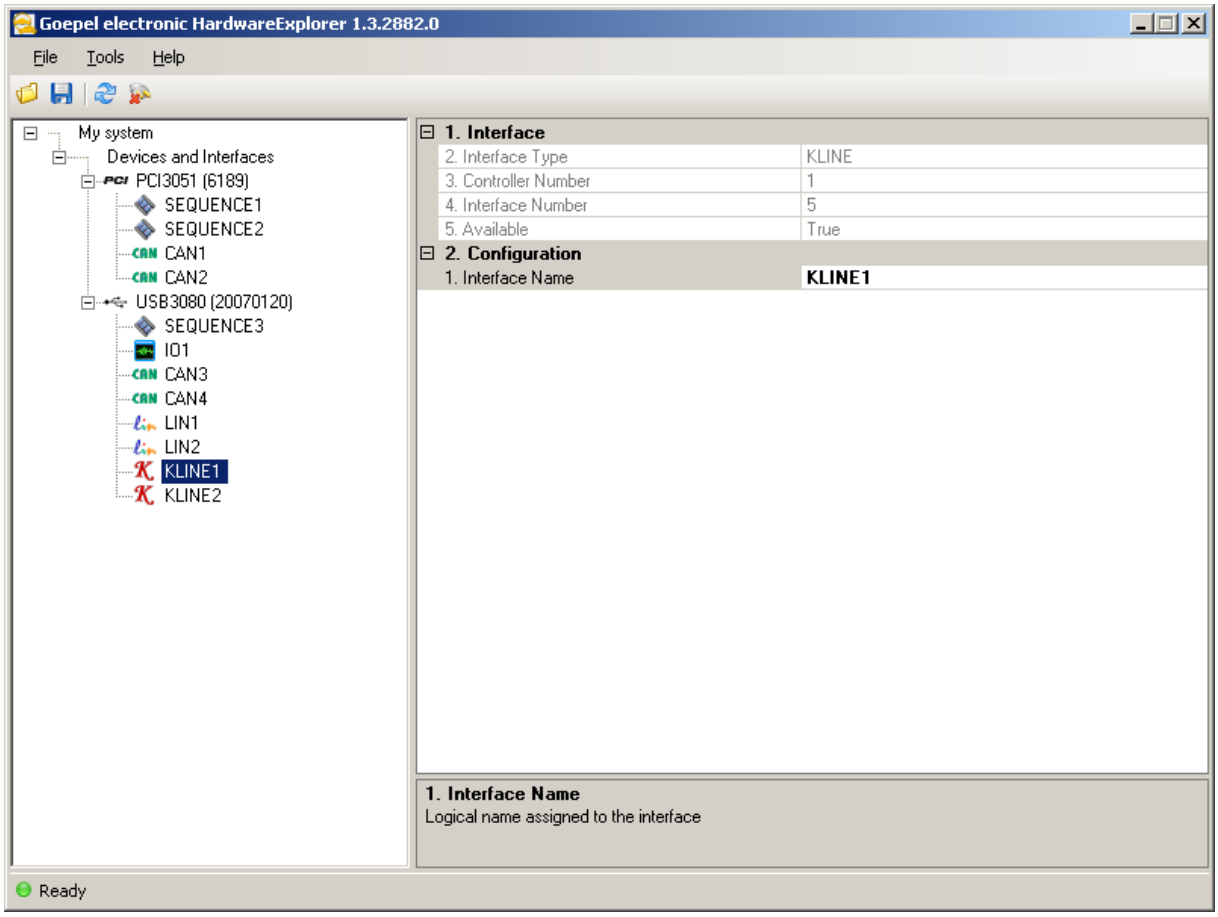

Figure 3.6 K-Line Interface Properties

Since the properties of the K-Line interface depend strongly on the type of diagnostic protocol, there are no common K-Line interface properties.

Use the **G–API** K-Line commands to configure the interface in your application. A description of these commands can be found in the **Reference Manual** .

3.4.5 IO Interface Configuration

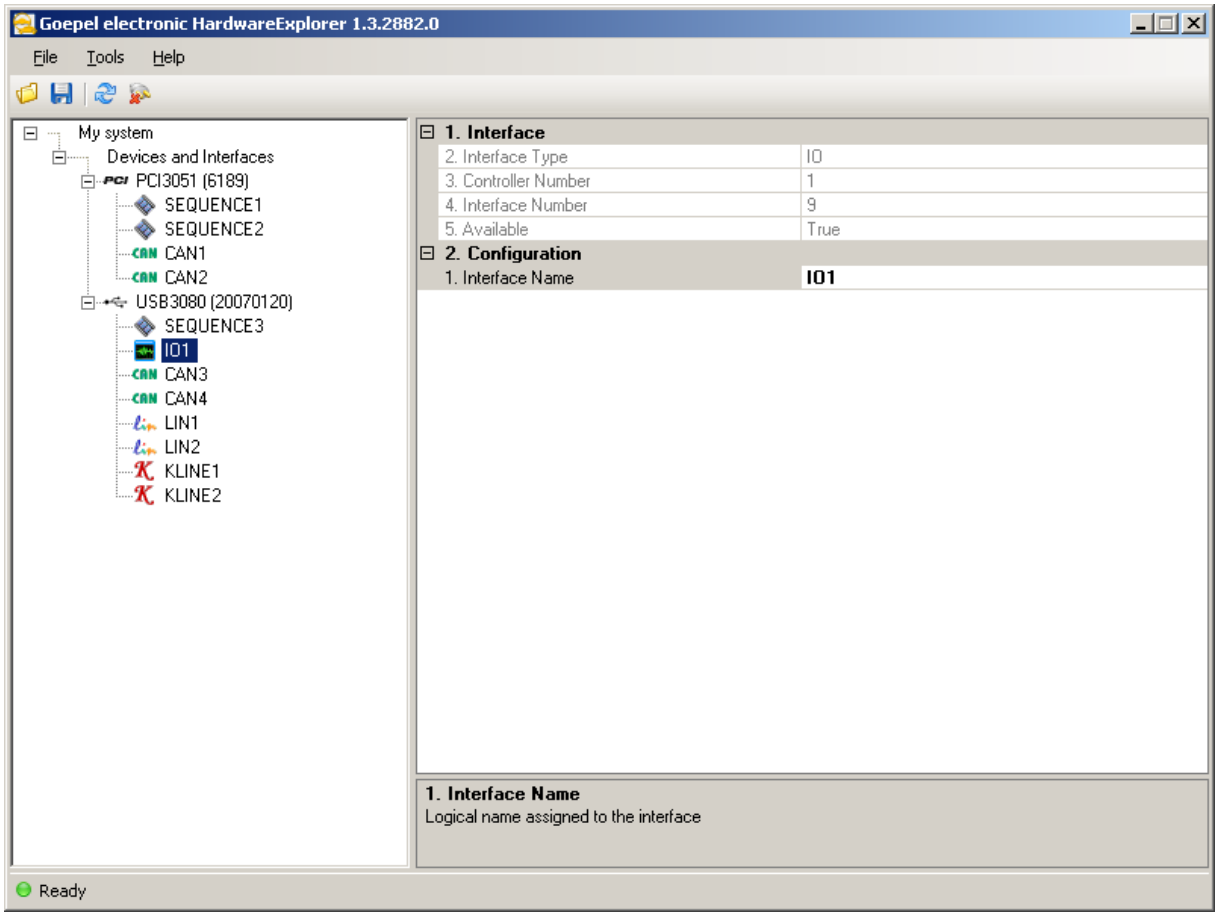

Figure 3.7 IO Interface Properties

The IO interface provides access to the inputs and outputs, relays and internally generated signals.

Use the **G-API** IO commands to gather information about the available resources and for controlling these.

3.4.6 Sequence Interface Configuration

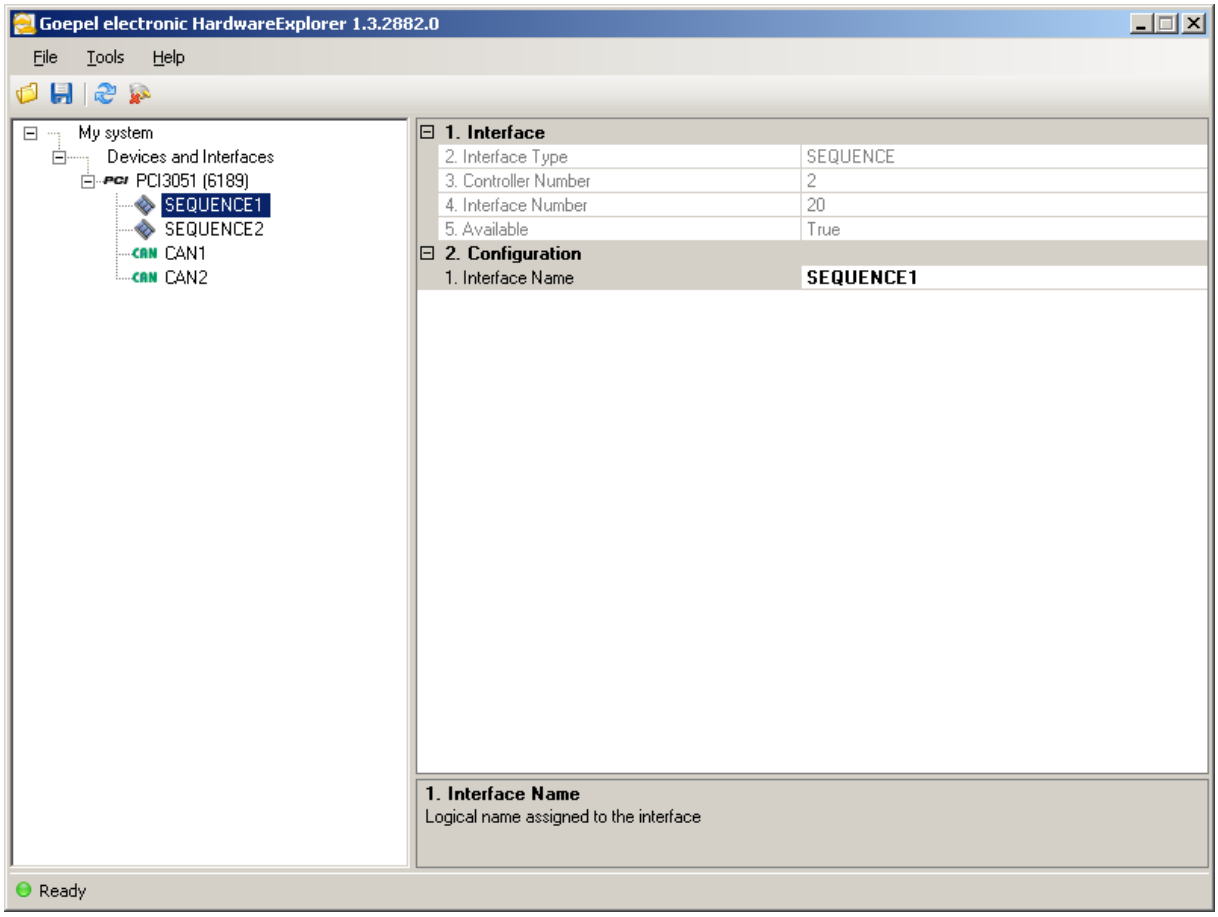

Figure 3.8 Sequence Interface Properties

The Sequence interface offers the possibility to record and play back command sequences.

This can be useful for example if a large number of commands have to be executed for more than one time.

You first need to allocate a sequence handle with **G-API** command `G_Sequence_Replay_Allocate`.

After calling command `G_Sequence_Replay_Recording_Start`, all commands to interfaces other than the sequence interface are recorded.

To stop recording, simply call `G_Sequence_Replay_Recording_Stop`. Now, you can control the recorded sequence by calling `G_Sequence_Replay_Playback_Start` and `G_Sequence_Replay_Playback_Stop`.

On devices providing a file system the recorded sequence can be stored permanently. The **G-API** command `G_Sequence_Replay_Autoplay_Enable` can then be used to enable the automatic start of the sequence at power on of the device.

3.4.7 MOST Interface Configuration

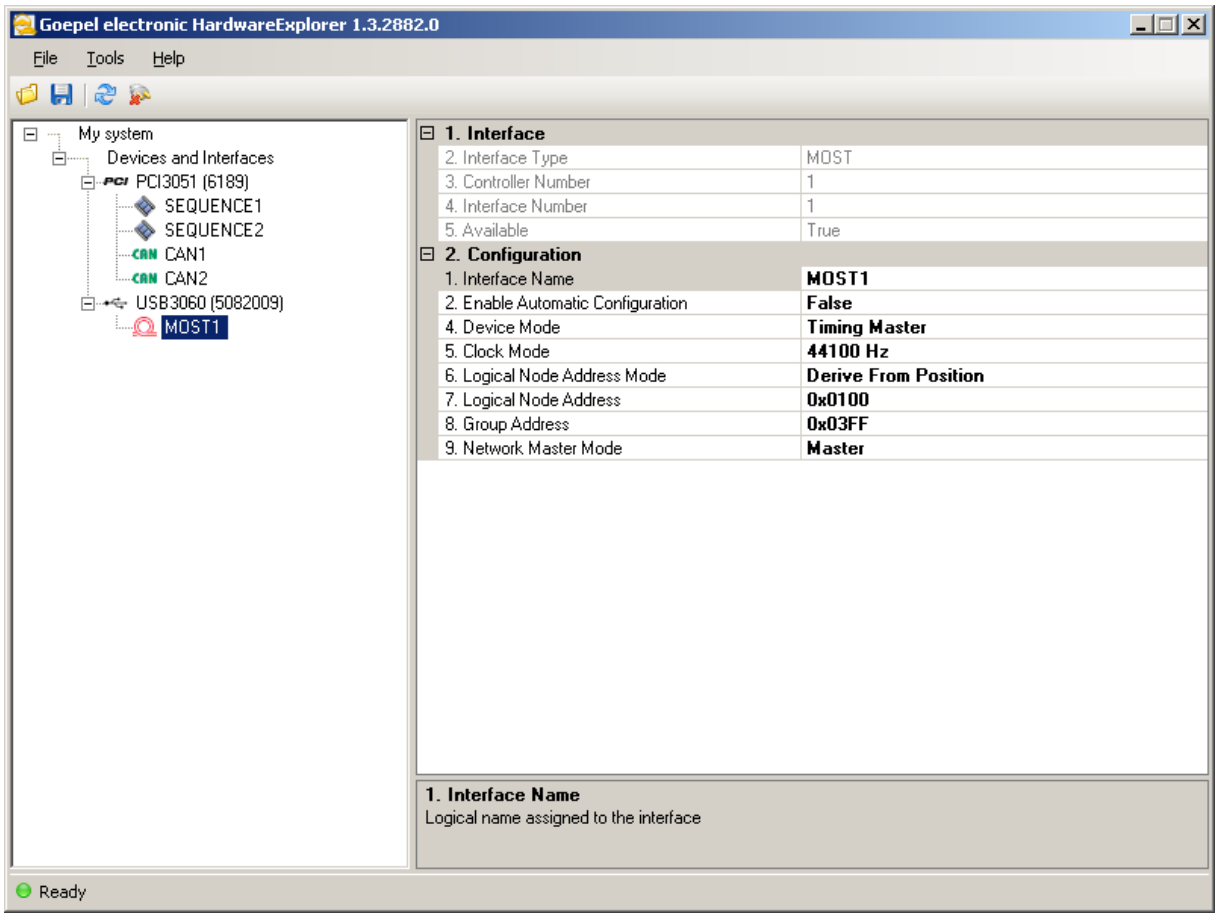

Figure 3.9 MOST Interface Properties

Device Mode

The following values are possible:

`Timing Master`
: The **Timing Master** generates the system clock for the frames and blocks. The timing slaves synchronize onto the system clock.

`Timing Slave`
: The **Timing Slave** synchronizes onto a given system clock.

`Spy`
: The interface is not visible in the net but can listen to the entire bus communication.

Clock Mode

The following values are possible:

`44100 Hz`
: The system clock is set to 44100 Hz.

`48000 Hz`
: The system clock is set to 48000 Hz.

Logical Node Address Mode

The following values are possible:

`Derive From Position`
The logical node address is derived from the position in the MOST ring (logical node address = **0x100** + **NodePosition**).

`Static Address`
The address given in **Logical Node Address** is used.

Logical Node Address
Logical node address (only relevant if **Logical Node Address Mode** is set to `Static Address` (0x0010..0x02FF, 0x0500..0x0FEF)

Group Address
Address of the group the interface belongs to (0x0300..0x03FF)

Network Master Mode
The following values are possible:

`Master`
The interface controls the system state and manages the central registry.

`Slave`
The interface is controlled by a network master.

3.4.8 FlexRay Interface Configuration

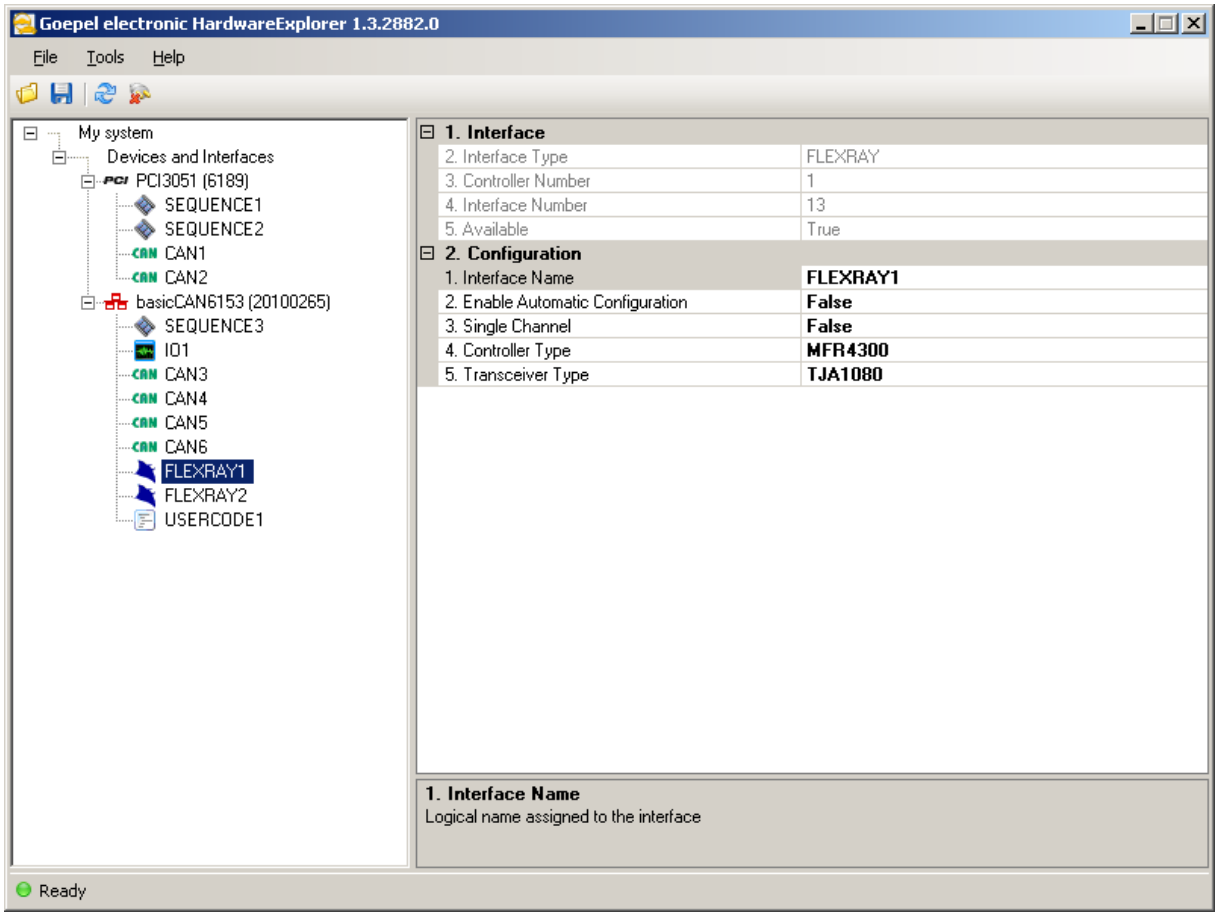

Figure 3.10 FlexRay Interface Properties

The FlexRay interface parameters are used to identify the communication module that is used on the device. This is necessary, because not all devices provide an automatic detection of the communication module.

Single Channel

If set to **True** , the **single channel mode** of the FlexRay controller is enabled. Otherwise it will be set to **dual channel mode** .

Controller Type

The following values are possible:

`MFR4300`
The FlexRay Communication Controller Module **MFR4300** is used on the device.

`MFR4310`
The FlexRay Communication Controller Module **MFR4310** is used on the device.

Transceiver Type

The following values are possible:

`TJA1080`
The transceiver **TJA1080** is used.

3.4.9 Net2Run Interface Configuration

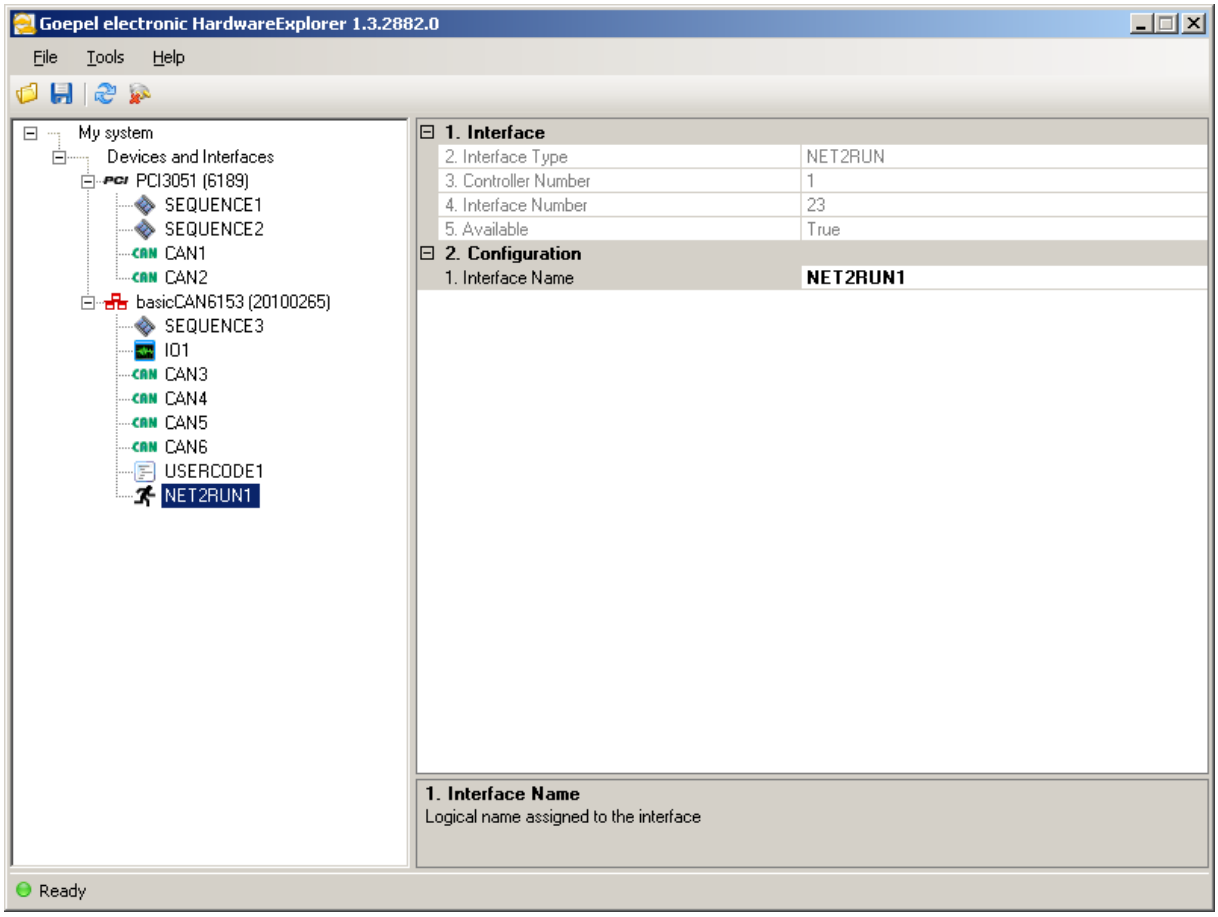

Figure 3.11 Net2Run Interface Properties

The Net2Run interface allows signal based communication. It represents an abstraction layer, that is used to address signals on different busses by name and to translate physical values to internal values and vice versa.

There are no specific Net2Run properties to be configured within the HardwareExplorer, because Net2Run is configured with a separate application, called **Net2Run Network Runtime Configurator .**

3.4.10 UserCode Interface Configuration

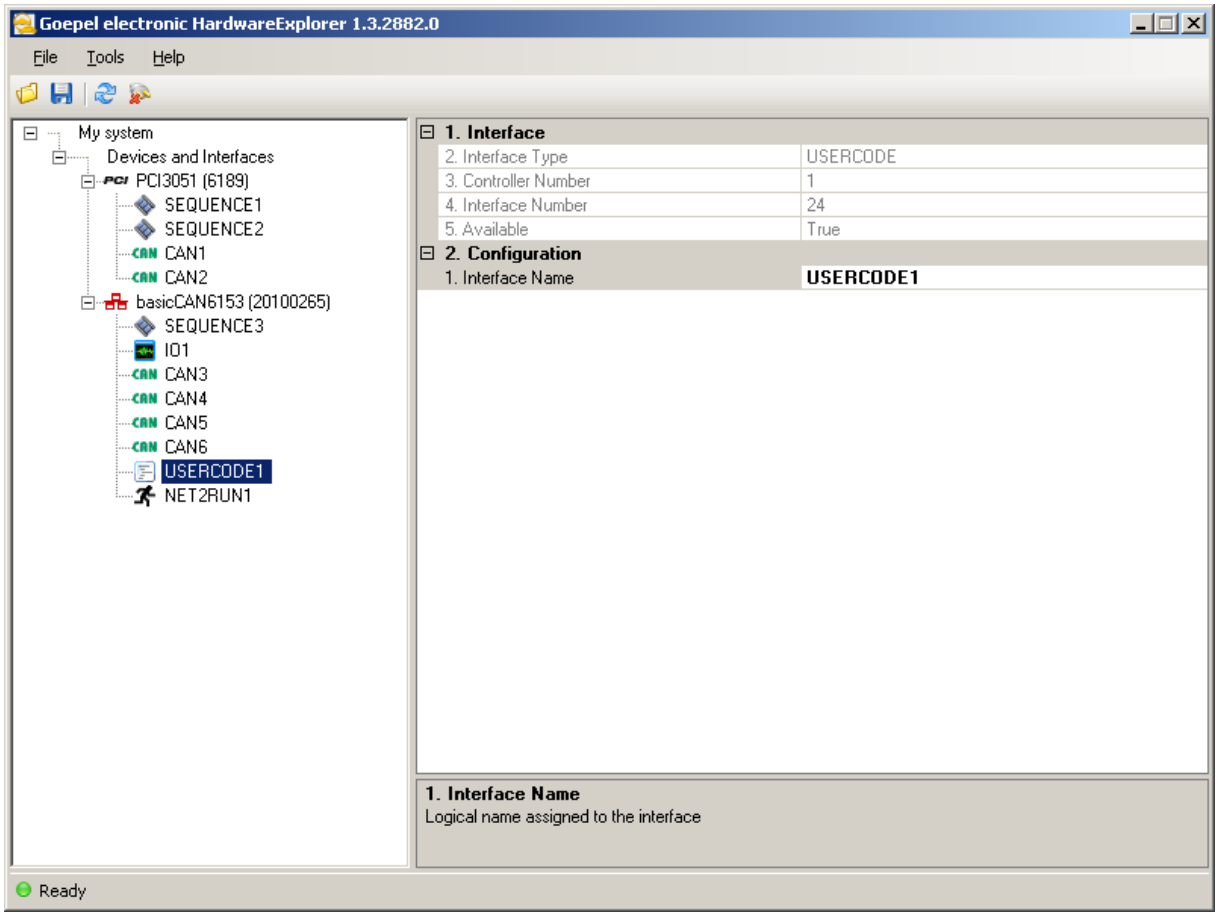

Figure 3.12 UserCode Interface Properties

The UserCode interface allows access to the part of the device, where applications written by the user can be stored and controlled.

Compared to the **Sequence Interface** , the **UserCode Interface** allows more sophisticated activities. Here, an entire application can be programmed by using the command set of the **Onboard API** .

It is possible to react to specific events or states within the application. After the application is compiled, it can be downloaded to the UserCode area, where it can be started, stopped and even be debugged.

Just like a sequence, a UserCode application can be configured to start automatically after a power on reset.

3.4.11 LVDS FrameGrabber Interface Configuration

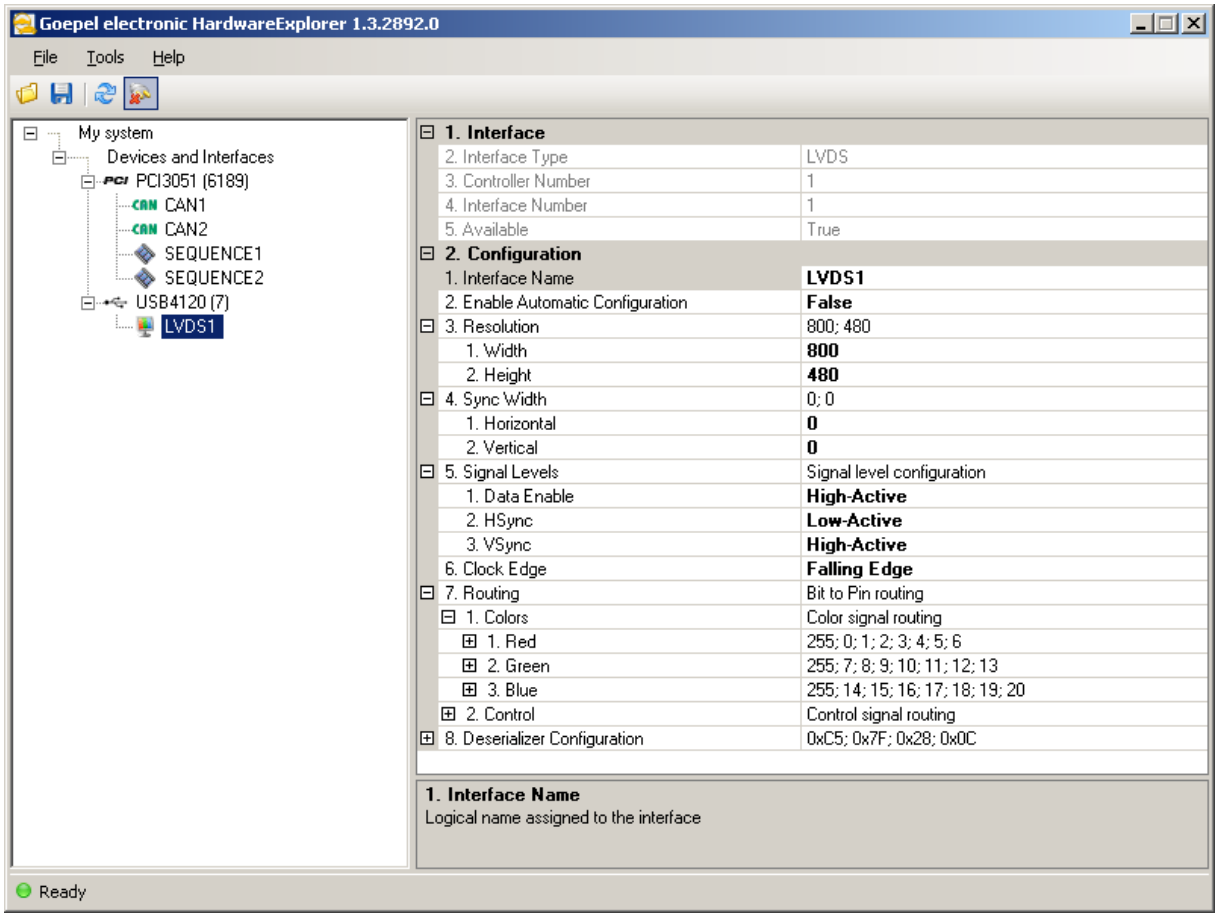

Figure 3.13 LVDS FrameGrabber Interface Properties

Resolution
Resolution of the grabbed image in pixels

Sync Width
Horizontal and vertical synchronization width

Signal Levels
Signal levels for signals **Data Enable** , **HSync** and **VSync**

The following values are possible:

`Low-Active`
The selected signal is **low active**

`High-Active`
The selected signal is **high active**

Clock Edge
Edge selection for the clock signal

The following values are possible:

`Falling Edge`
The falling edge of the clock signal will be used for synchronization

`Rising Edge`
The rising edge of the clock signal will be used for synchronization

Routing

Routing of hardware pins of the LVDS deserializer to bits of image signals, including color and control pins

For Example, if **Bit 0** of the color **Red** is connected to the **3rd pin** of the deserializer, the value for ***Routing*** → ***Colors*** → ***Red*** → ***Bit0*** has to be **3** .

Unused signal bits need to be set to **255** !

3.5 Menu Options

3.5.1 Menu Bar

Figure 3.14 Menu Bar

The menu bar located in the upper half of the main window offers access to common features of the HardwareExplorer.

File Menu

File → Open Configuration
Load configuration from file

A snapshot of a device configuration can be loaded from a file. This is useful for restoring a configuration backup or when working with different configurations for one device.

If the file contains configuration data for currently connected devices, the current configuration of these devices will be overwritten.

File → Save Configuration
Save configuration to file

A snapshot of the current device configuration is saved to a file.

File → Exit
Closes the program

Tools Menu

Tools → Refresh Configuration
Scans for newly connected or disconnected devices and updates the device list accordingly.

Tools → Remove Disconnected Devices
Removes all devices that are currently not connected from the device list.

Tools → Reset G-API Service
Resets (actually restarts) the G-API Service that is used for communication with **GOEPEL electronic** devices.

Tools → Preferences
Opens the preferences dialog

The preferences dialog gives access to configuration parameters of the HardwareExplorer. It is divided into categories that are shown in the list on the left side of the dialog.

General Preferences

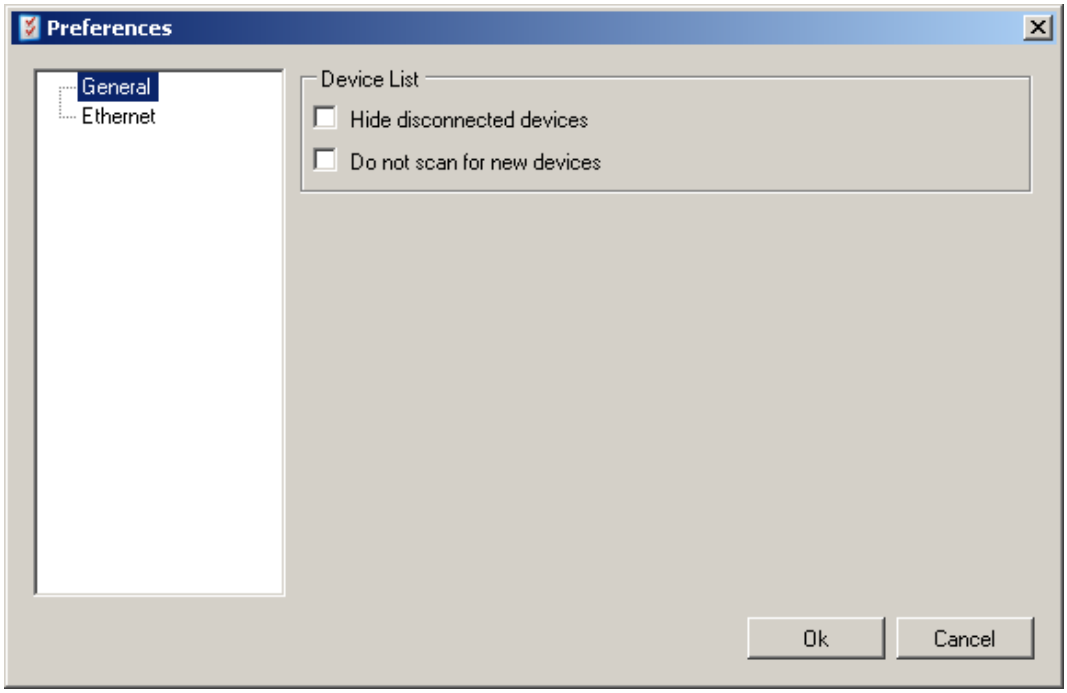

Figure 3.15 General Preferences

Device List

Hide disconnected devices
If checked, all disconnected devices are hidden (not deleted!) from the device list.

Do not scan for new devices
If checked, only devices that are already in the device list are updated. When new devices are connected, they will not show up in the device list until this option is deactivated.

Ethernet Preferences

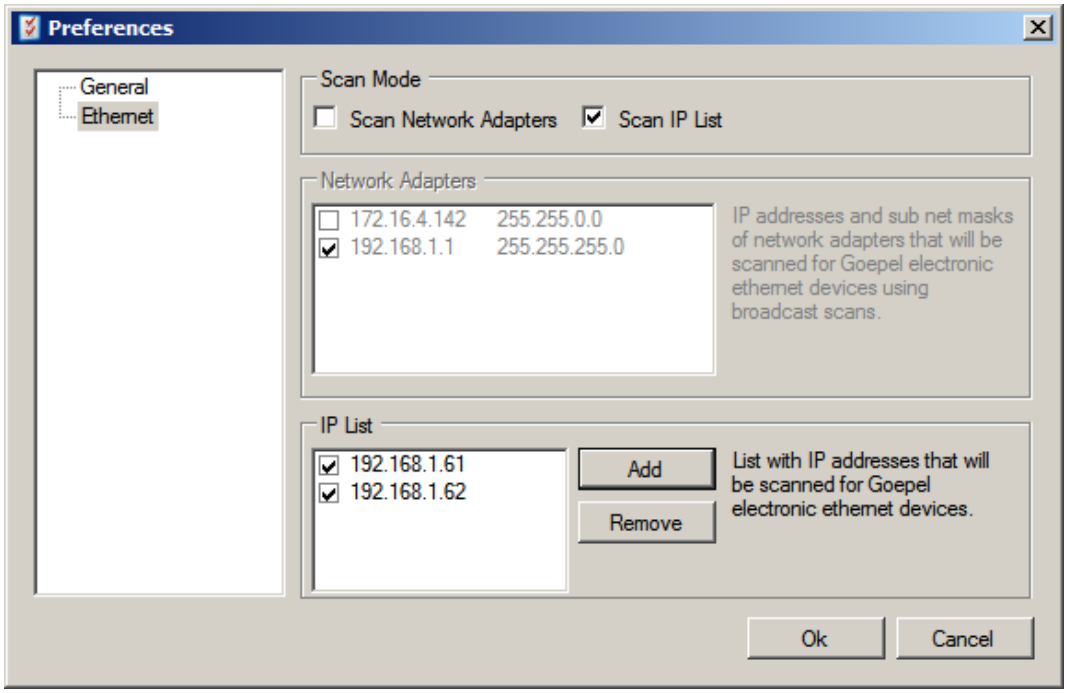

Figure 3.16 Ethernet Preferences

Scan Mode

Defines the mode for scanning for **GOEPEL electronic** ethernet devices.

Scan Network Adapters

A broadcast scan is performed for all selected ethernet adapters in the 'Network Adapters' list.

You can specify which adapters will be scanned by checking / unchecking the adapters in the 'Network Adapters' list.

Scan IP List

A list of user defined IP addresses that will be scanned.

IP addresses can be added to or removed from the list by using the dedicated buttons next to the 'IP List'.

You can specify which ip addresses will be scanned by checking / unchecking them in the 'IP List' list.

After the first install, all Scan Mode Options are unchecked. This means that no scan for **GOEPEL electronic** ethernet devices will be preformed.

Help Menu

Help → About

Display the **About Screen** containing version and contact information.

3.5.2 Tool Bar

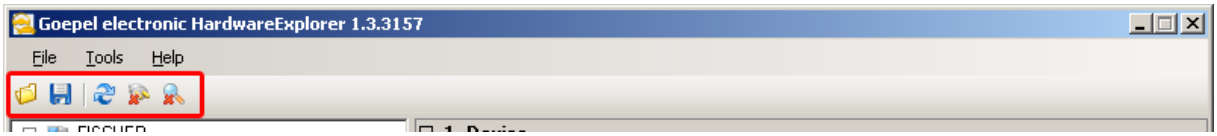

Figure 3.17 Tool Bar

The tool bar located below the menu bar offers one-click-access to the most common features of the HardwareExplorer.

Load configuration from file

A snapshot of a device configuration can be loaded from a file. This is useful for restoring a configuration backup, or when working with different configurations for one device.

If the file contains configuration data for currently connected devices, the current configuration of these devices will be overwritten.

Save configuration to file

A snapshot of the current device configuration is saved to a file.

Scans for newly connected or disconnected devices and updates the device list accordingly.

Hide disconnected devices from device list.

Do not scan for new devices

When activated, only the devices that are already in the device list are updated. If new devices are connected, they will not show up in the device list until this option is deactivated.

3.5.3 Device Context Menu

Open the **Device Context Menu** by right-clicking on a device in the device list.

Figure 3.18 Device Context Menu

Flash Firmware
Update the firmware of the device.

3.5.4 Interface Context Menu

Open the **Interface Context Menu** by right-clicking on an interface in the device list.

Figure 3.19 Interface Context Menu

Self Test
Perform a quick self test of the interface.

Reset Interface
Reset the interface to its initial state.

Read Firmware Version
Read the firmware version of the interface.

Chapter 4 Building an Application

4.1 Introduction

This chapter is a step by step tutorial of building a simple application with **Microsoft Visual C++ 2008 Express Edition .**

When working with other development environments, these steps may differ slightly.

4.2 Project Properties

After starting Visual C++, go to ***File*** → ***New*** → ***Project*** .

This will open a project wizard, where you first have to choose a project template.

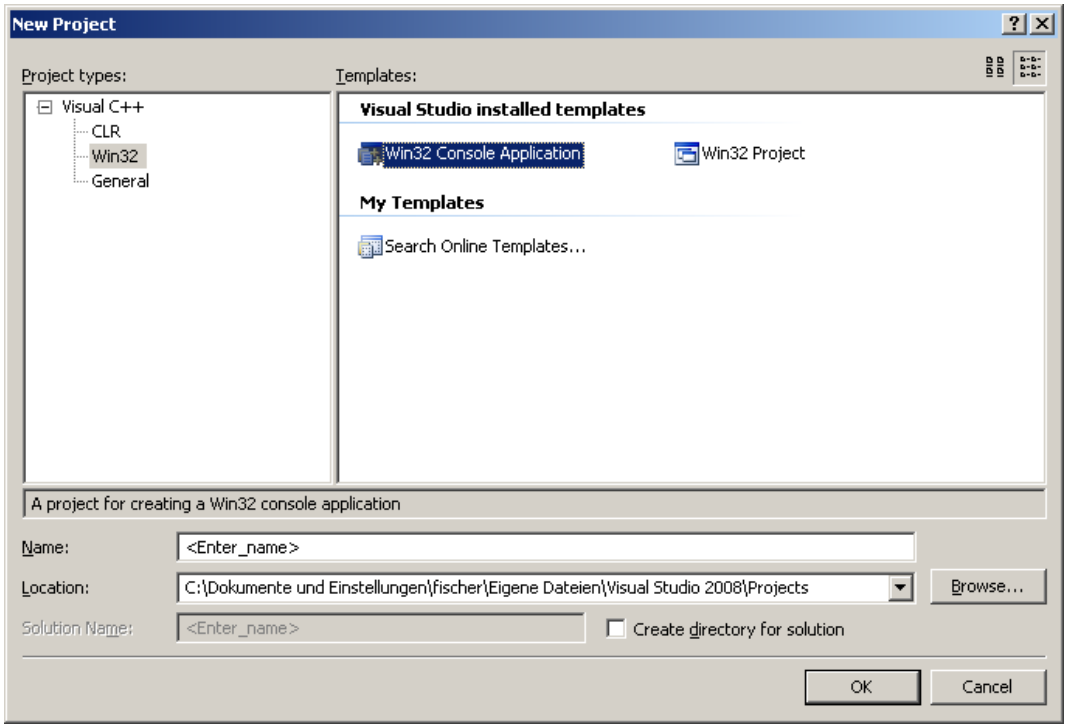

Figure 4.1 Project Template

Since we want to build a console application, we choose **Win32 Console Application .**

Enter a project name and click ***OK*** to proceed. In this example, our project is called **G-API Tutorial .**

In the following dialog, click ***NEXT*** for changing the standard application settings.

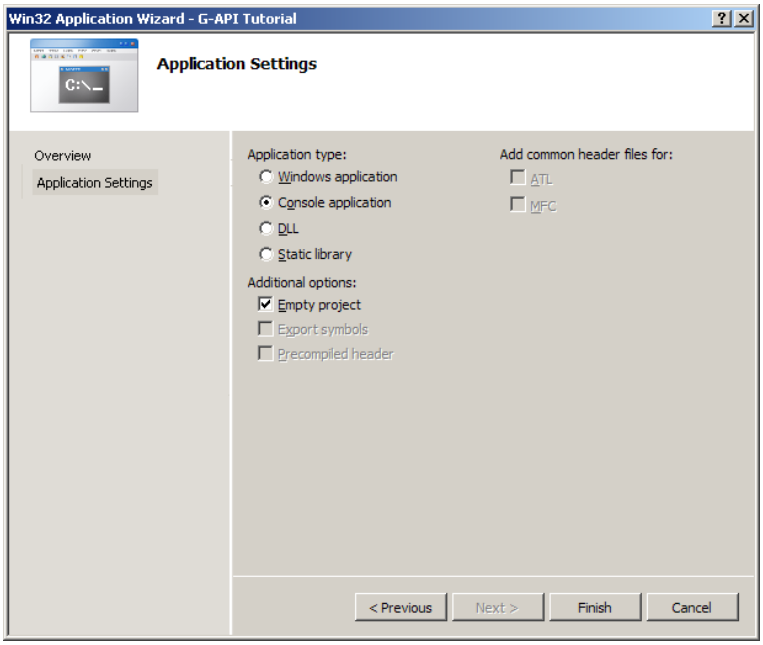

Figure 4.2 Application Settings

Under **Additional Options** uncheck **Precompiled Header** and check **Empty Project** for creating an empty project.

Close the project wizard by clicking ***FINISH***.

In the **Solution Explorer** window on the left, right click on **Source Files** and choose ***Add*** → ***New Item...*** and enter *tutorial.c* as file name.

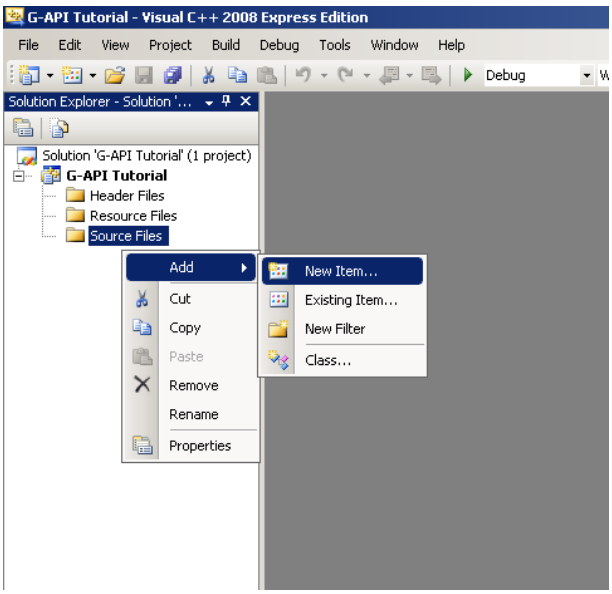

Figure 4.3 Solution Explorer

This will create the empty file *tutorial.c*, add it to the project and open it in the edit window.

Since our program needs to access functions contained in *g_api_common.dll* and *g_api.dll*, we need to include the library information for these files.

To do this, right click on **G-API Tutorial** in the **Solution Explorer** and choose ***Properties*** . This will open a dialog box with the project properties.

Go to **C/C++** and enter the path to the G-API *bin* directory in the G-API directory.

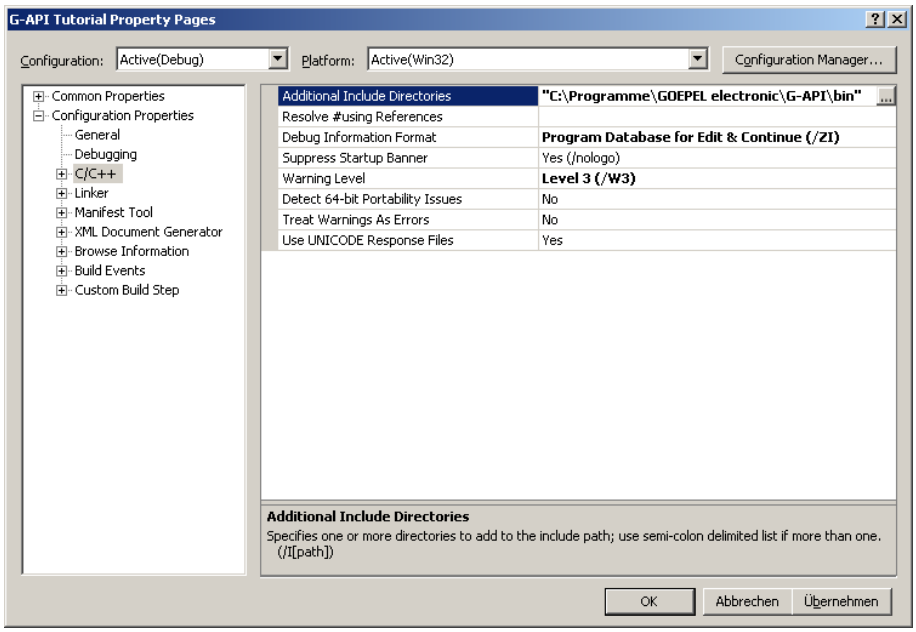

Figure 4.4 C/C++ Properties

Close the properties dialog, right click on **Resource Files** in the **Solution Explorer** and choose ***Add*** → ***Existing Item...*** .

Go to the *bin* directory of your G-API installation, select the files *g_api_common.lib* and *g_api.lib* and click ***ADD*** to add the files to the project.

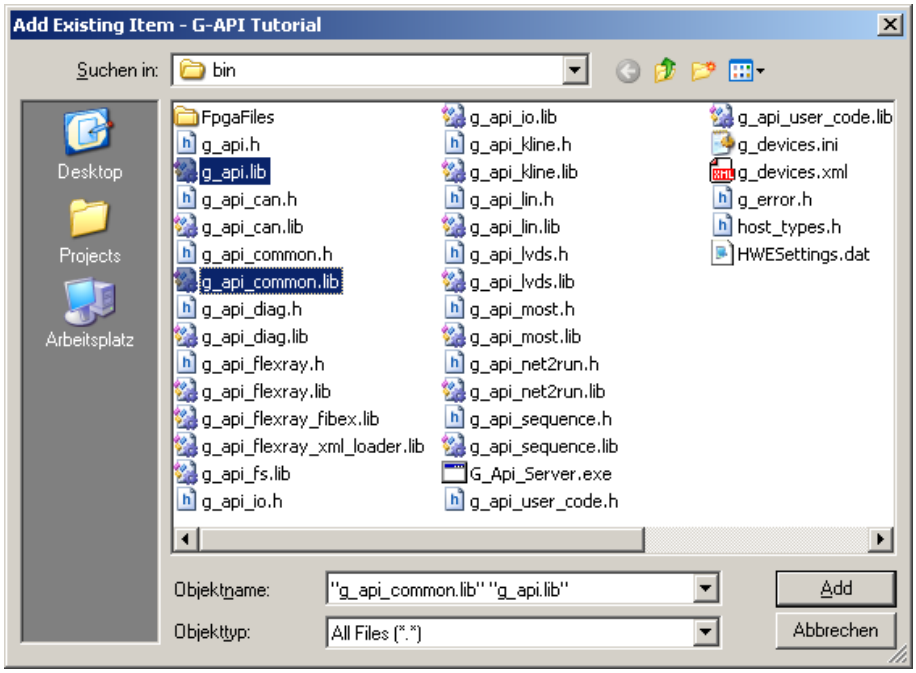

Figure 4.5 Add Library Files

If a dialog appears asking for a custom build rule simply click ***NO***.

If your project requires access to other G-API DLL files you need to include these as well.

At this point all project properties are set and the Solution Explorer should look like this:

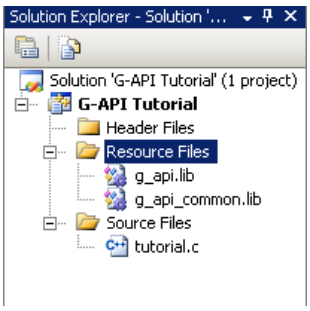

Figure 4.6 Solution Explorer After Configuration

4.3 Source Code

In this tutorial, we will build a simple application that reads the firmware version of an interface and displays it in a console window.

4.3.1 Include Headers

Our example application needs two header files to be included.

```
#include "g_api_common.h"
#include <stdio.h>
```

g_api_common.h is required for accessing functions of *g_api_common.dll*, *stdio.h* is a windows library header needed for the `printf` function.

4.3.2 Local Variables

The main function of our application has a set of local variables:

```
G_Error_t  rc;
G_PortHandle_t  portHandle;
char buffer[1024];
u32_t  length = 1024;
```

`rc`
Provides the return code of a G-API function call. If the function returns without error, the value of `rc` is `G_NO_ERROR`.

`portHandle`
The **port handle** is used to specify the connection of the application to the interface. It is returned by function `G_Common_OpenInterface` and used for subsequent G-API function calls.

`buffer[1024]`
The data buffer for storing the firmware information string. In our example it is a **char** array that can hold 1024 characters.

`length`
The variable `length` is used to pass the size of our buffer to the G-API. On function return, it contains the size of the returned string in characters.

4.3.3 Open Interface

At first, we need to open a port to an interface. The interface is specified by its name assigned in the **HardwareExplorer** .

In this example we will choose a CAN interface and assign the name `TUTORIAL`.

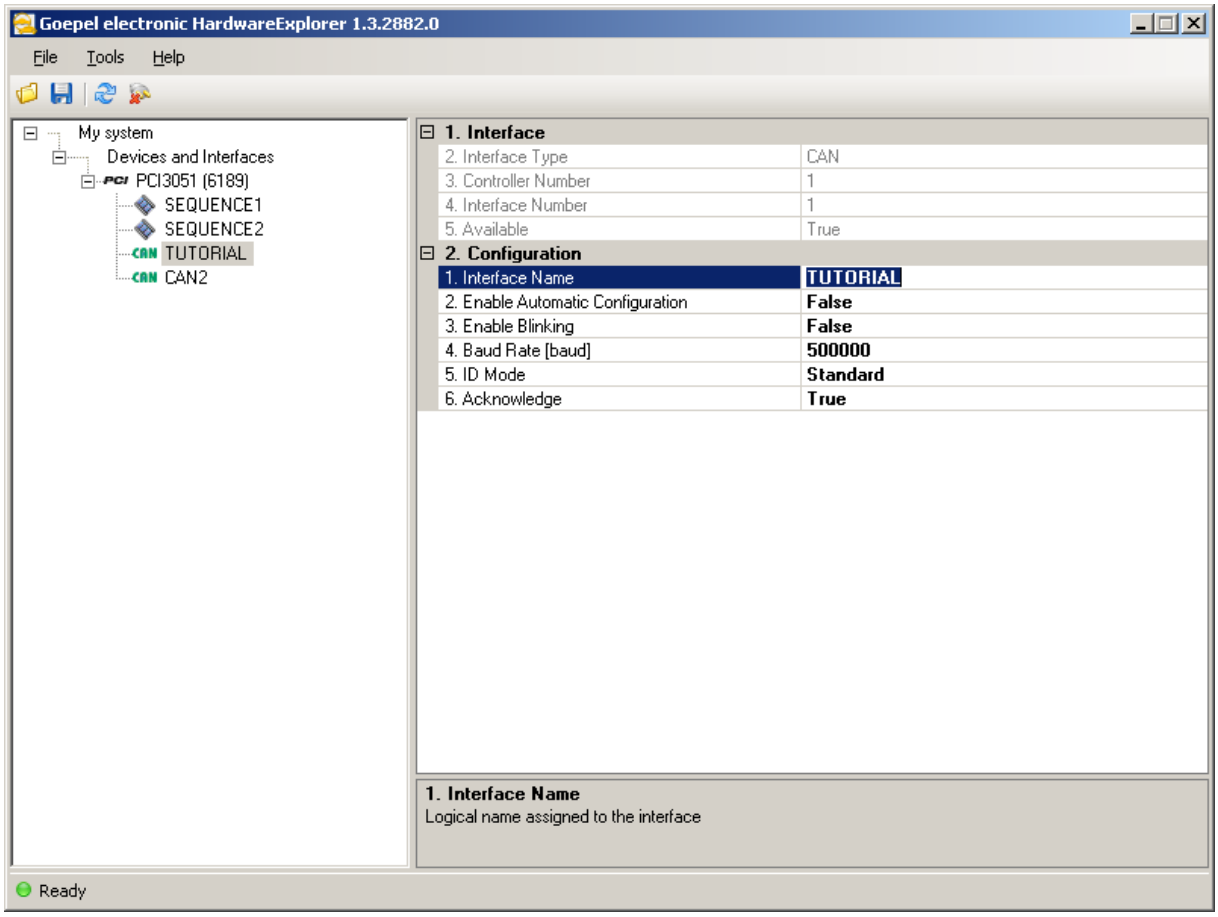

Figure 4.7 Assign Interface Name

The port is opened by calling `G_Common_OpenInterface`, passing the interface name and the pointer to our `portHandle` variable. On function return `portHandle` contains the **port handle** .

```
rc =
 G_Common_OpenInterface(
  "TUTORIAL",
  &portHandle
 );
```

4.3.4 Read Firmware Version

To read the firmware version of the interface we need to call `G_Common_GetFirmwareVersion`, passing the **port handle** , a pointer to our buffer and a pointer to our length variable.

Note that `length` is initialized with the size of our buffer on definition! On function return, `buffer` will contain the firmware version string and `length` will contain the length of the string in characters.

```
rc =
 G_Common_GetFirmwareVersion(
  portHandle,
  buffer,
  &length
 );
```

4.3.5 Print Firmware Version

A simple `printf` command is used to print the firmware version string to the console window.

```
printf("%s\n", buffer);
```

4.3.6 Close Interface

After the firmware version has been read we need to close the port to the interface.

```
rc = G_Common_CloseInterface(portHandle);
```

All resources of the connection are freed and the **port handle** becomes invalid.

Subsequent calls of G-API functions will need a new **port handle** that can be retrieved with `G_Common_OpenInterface`.

4.3.7 Error Handling

It is important to check if a G-API function call is successful by validating the return code `rc`.

```
if (rc != G_NO_ERROR) {
 printf("%s\n", G_GetLastErrorDescription());
 return;
}
```

If the return code is not `G_NO_ERROR`, a description of the last G-API error is queried with `G_GetLastErrorDescription` and printed to the console window. After that the program is closed.

4.3.8 The Whole Example

```
#include "g_api_common.h"
#include <stdio.h>


void main(void) {
 G_Error_t  rc;
 G_PortHandle_t  portHandle;
 char buffer[1024];
 u32_t  length = 1024;

 // open port to interface
 rc =
  G_Common_OpenInterface(
   "TUTORIAL",
   &portHandle
  );

 if (rc != G_NO_ERROR) {
  printf("%s\n", G_GetLastErrorDescription());
  return;
 }

 // read firmware version
 rc =
  G_Common_GetFirmwareVersion(
   portHandle,
   buffer,
   &length
  );

 if (rc != G_NO_ERROR) {
  printf("%s\n", G_GetLastErrorDescription());
  return;
 }

 // print firmware version to console window
 printf("%s\n", buffer);

 // close port
 rc = G_Common_CloseInterface(portHandle);

 if (rc != G_NO_ERROR) {
  printf("%s\n", G_GetLastErrorDescription());
  return;
 }
}
```