TECHNICAL INFORMATION

SICK Vision Suite Programming Manual 1.1

SICK
Sensor Intelligence.

Contents

1 Preface

Introduction

SICK AG has taken every possible care in preparing this manual. We however assume no liability for the content, completeness or quality of the information contained therein. The content of this manual is regularly updated and adapted to reflect the current status of the software. We furthermore do not guarantee that this product will function without errors, even if the stated specifications are adhered to.

Under no circumstances can we guarantee that a particular objective can be achieved with the purchase of this product.

Insofar as permitted under statutory regulations, we assume no liability for direct damage, indirect damage or damages suffered by third parties resulting from the purchase of this product. In no event shall any liability exceed the purchase price of the product.

Please note that the content of this manual is neither part of any previous or existing agreement, promise, representation or legal relationship, nor an alteration or amendment thereof. All obligations of SICK AG result from the respective contract of sale, which also includes the complete and exclusively applicable warranty regulations. These contractual warranty regulations are neither extended nor limited by the information contained in this manual. Should you require further information on this product, or encounter specific problems that are not discussed in sufficient detail in the manual, please contact your local dealer or system installer.

Trademarks

Microsoft and Windows are trademarks or registered trademarks of Microsoft Corporation. All other products or company names mentioned in this manual are used solely for purposes of identification or description and may be trademarks or registered trademarks of the respective owners.

Copyright

Status: April 2020

Contact

Visit our web site www.sick.com where you will find all the latest information about our software and hardware products.

Address	SICK AG Erwin-Sick-Str. 1 D-79183 Waldkirch, Germany
T	+49 7681 202-0
E	info@sick.com
W	www.sick.com

2 SICK Vision Suite

SICK Vision Suite is a comprehensive software package from SICK AG that can be used with GenICam-compliant industrial cameras. SICK Vision Suite provides all necessary tools to open cameras in an application with graphical user interface, to parametrize them, to capture images, etc. or to program your own application. This manual describes how to use SICK Vision Suite for programming.

2.1 Notes on usage

This symbol indicates hints with useful information for better understanding and using features and functions.

This symbol indicates important warnings for product safety to prevent damage.

This symbol indicates important warnings for personal safety to prevent injury.

2.2 Directories under Windows

Setup directory

The setup directory of SICK Vision API and SICK LibIMG is stored in the system environment variable SICKvision_api. On Windows this is typically the directory "C:\Program Files\SICK\SICKVisionSuite\sdk".

Include directories

The headers for SICK Vision Suite are located in the directories:

```
$(SICKvision_api)\api\include\vision_api
$(SICKvision_api)\libimg\include\libimg
```

Library directories

The libraries must be differentiated for 32 and 64-bit systems. They are located in the directories:

32-bit
`$(SICKvision_api)\api\lib\x86_32` `$(SICKvision_api)\libimg\lib\x86_32`
64-bit
`$(SICKvision_api)\api\lib\x86_64` `$(SICKvision_api)\libimg\lib\x86_64`

2.3 Directories under Linux

Under Linux, the default directories for the various components are used when installing the Debian packages.

Include directories

The headers for SICK Vision Suite are located in the include directories under "/usr/include/":

```
/usr/include/SICKVisionSuite-[version]/
/usr/include/SICKVisionSuite_libimg-[version]/
```

Library directories

The libraries are located in the library directory under "/user/lib/":

```
/usr/lib/SICKVisionSuite-[version]/
/usr/lib/SICKVisionSuite_libimg-[version]/
```

3 Creating projects

When you create an application with SICK Vision Suite, there are several ways to create the project:

- Use a development environment (IDE) to open a CMake project directly, e.g. Qt Creator or Visual Studio 2017 or higher (recommended).
- Use predefined projects from the "samples" folder as a starting point.
- Use CMake to create your own projects.

3.1 CMake

CMake is a cross-platform open source tool for creating, testing and building software. CMake is used to control the software compilation process with simple platform- and compiler-independent configuration files. These files allow you to create Makefiles and projects for many development environments and compilers.

Further information about CMake can be found at https://cmake.org/

3.2 Creating a project for Qt Creator with CMake

- CMake setup in Qt Creator
- Opening samples
- Creating your own projects

3.2.1 CMake setup in Qt Creator

Detailed information about the setup of CMake with Qt Creator can be found at https://doc.qt.io/qtcreator/creator-project-cmake.html#adding-cmake-tools

If CMake has been installed correctly, the installed version will automatically be displayed in Qt Creator under "Options > Kits > CMake".

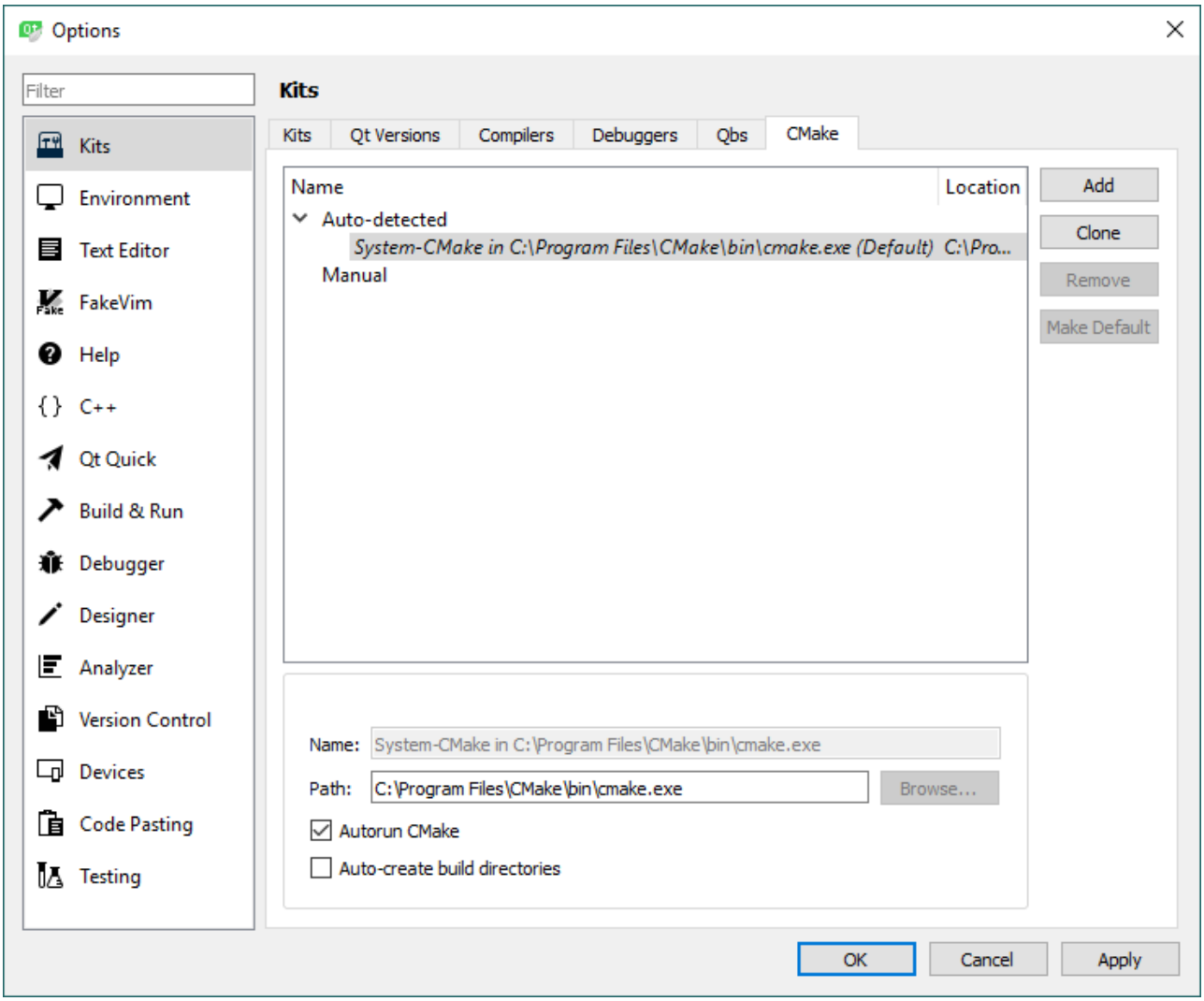

Fig. 1: CMake in Qt Creator

For some compilers the CMake settings have to be checked. Open the tab "Options > Kits > Kits". The CMake settings can be adjusted under "CMake Generator".

Compiler	CMake Generator
MinGW	MinGW Makefiles
MSVC	NMake Makefiles

3.2.2 Opening samples

You can open the provided samples with Qt Creator by opening the CMakeLists.txt files as a project. Before you start, it is recommended to copy the folder ".../SICK/SICKVisionSuite/sdk/samples/source/" into a user directory, e.g. under "Documents". Now open the solution via "File > Open File or Project" and select the CMake file "./source/CMakeLists.txt". A complete project folder with all samples is compiled. Now these can be built and executed individually. Alternatively, you can select the CMakeLists.txt files in the subdirectories. Only the selected sample will be created.

Further information about opening projects under Qt Creator can be found at https://doc.qt.io/qtcreator/creator-project-opening.html

3.2.3 Creating your own projects

The easiest way to create your own project is to copy and customize a sample. To do this, change the project name in the CMakeLists.txt file and add additional source files if necessary:

CMakeLists.txt

```
...
project (new_project_name)
...
add_executable (${SAMPLE_TARGET_NAME}
    ${PROJECT_NAME}.cpp
    new_source_file.cpp
)
...
```

Copy the ".../SICK/SICKVisionSuite/sdk/samples/source/vision_api/_cmake_scripts/" folder next to your project folder. Now you can open the project like any other sample in Qt Creator.

A second option is to create the project using the Qt Creator wizard.

- Select "File > New File or Project".
- Select the required project type. For example, use "Non-Qt Project > Plain C++ Application" to create a simple console application, use "Application > Qt Widgets Application" or "Application > Qt Quick Application" to create an application with Qt interface.
- Follow the steps of the project wizard.
- Select the "CMake" type in the "Build system" step.
- Follow the further steps of the project wizard and finish it. This creates the project.
- Now adapt the CMakeLists.txt file for the use with SICK Vision Suite.

CMakeLists.txt

```
...
# use C++14 or higher
set(CMAKE_CXX_STANDARD 14)
...
# set path to SICK Vision API CMake Finder
set(CMAKE_MODULE_PATH ${CMAKE_MODULE_PATH} "C:/Program
Files/SICK/SICKVisionSuite/sdk/api/cmake_finder")
...
# find SICK Vision API API package
find_package(sick_vision_api REQUIRED)
...
# link SICK Vision Suite libraries
target_link_libraries(${PROJECT_NAME} PRIVATE sick_vision_api)
...
```

Further information about creating projects under Qt Creator can be found at https://doc.qt.io/qtcreator/creator-project-creating.html

3.3 Creating a project for Visual Studio with CMake

The CMake support is directly integrated since Visual Studio 2017. This way, you can open folders containing a CMakeLists.txt file directly via "File > Open > Folder". More information can be found at https://docs.microsoft.com/de-de/cpp/build/cmake-projects-in-visual-studio. Select the appropriate Visual Studio version.

For all Visual Studio versions ≤ 2015 without direct CMake support it is possible to create Visual Studio projects via the CMake GUI. Proceed as follows:

1. Open the CMake GUI.

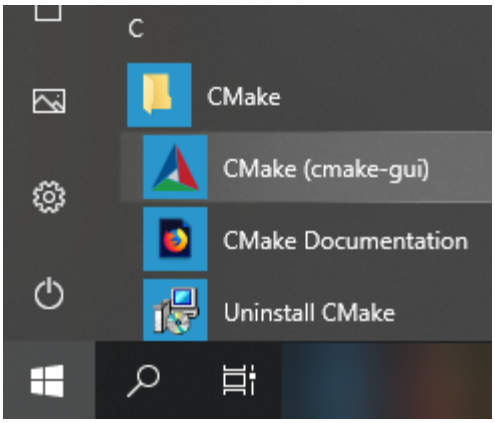

Fig. 2: Open the CMake GUI

2. Specify the path to the folder containing the CMakeLists.txt file and a build directory for which you have write permissions.

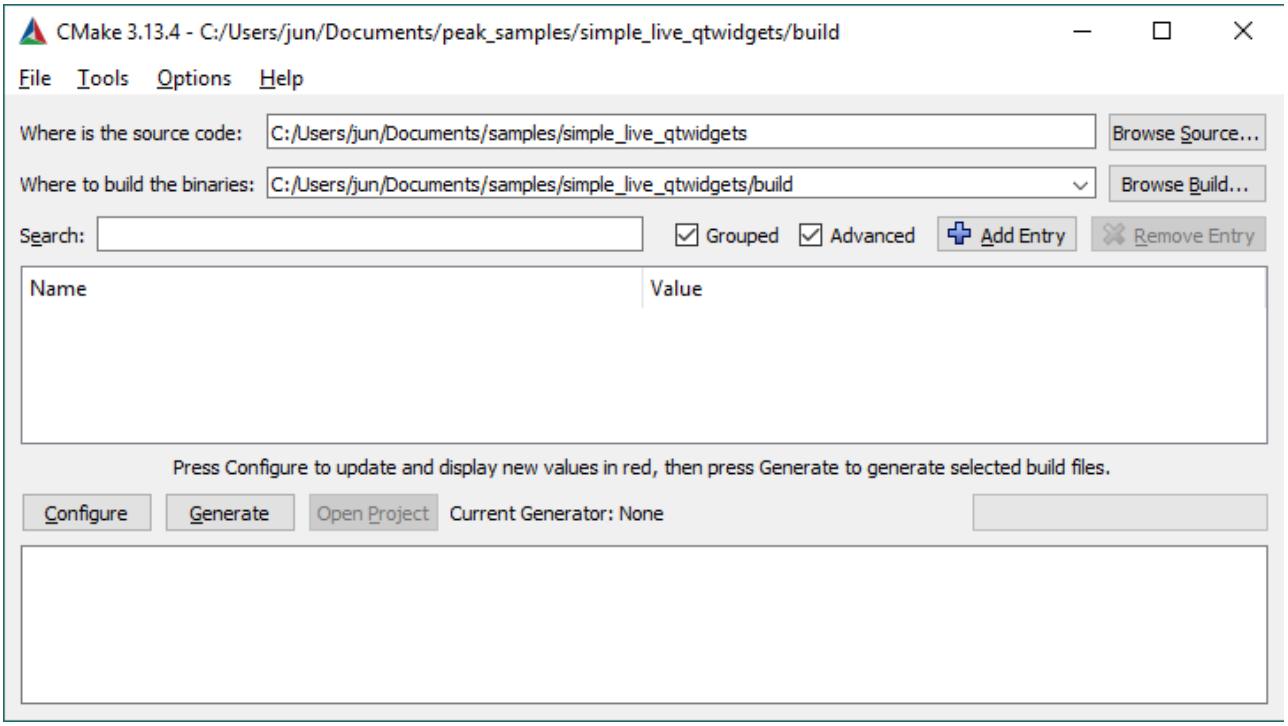

Fig. 3: Specify paths and build directory

3. Click on “Configure”.

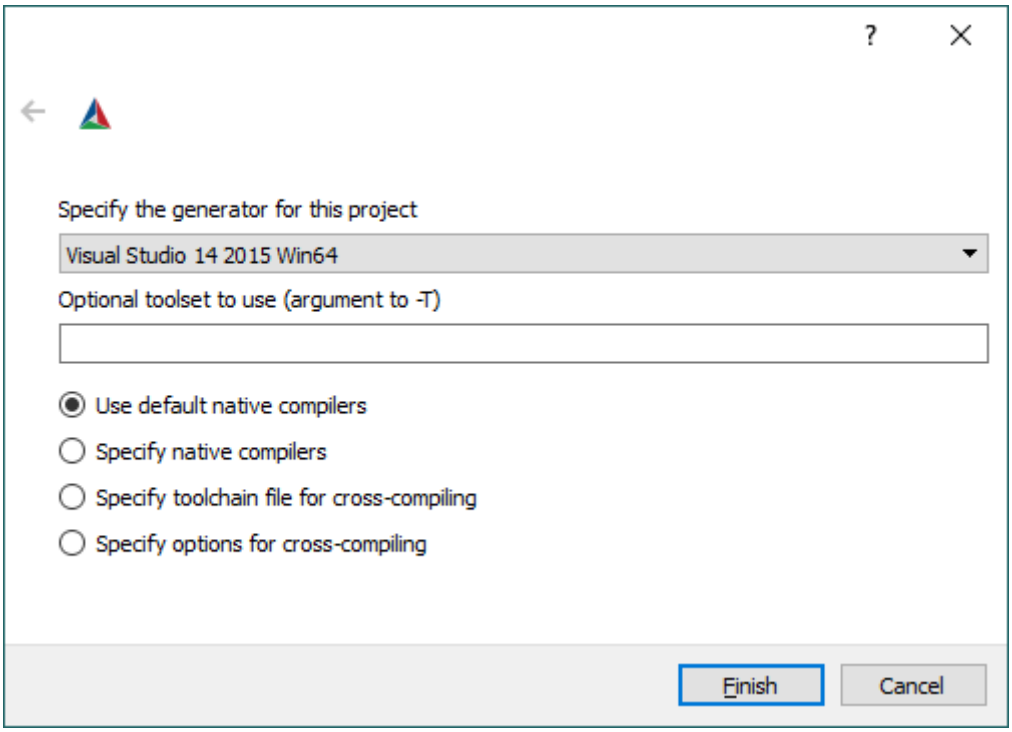

Fig. 4: Select compiler

4. Select the required compiler.
5. Click on “Finish”.
   Now CMake configures the project. For this, CMake searches e.g. the paths to the required components (see also Correcting configuration settings).
6. Then click on "Configure".
7. Click on "Generate" to create the Visual Studio project. Afterwards the project can be opened directly in Visual Studio via "Open Project". The required project files are created in the build directory, including the Visual Studio Solution (*.sln).

3.3.1 Correcting configuration settings

When CMake configures a project, CMake searches e.g. the paths to the required components. If not all components are found, an error message is displayed. In this case, correct the corresponding settings.

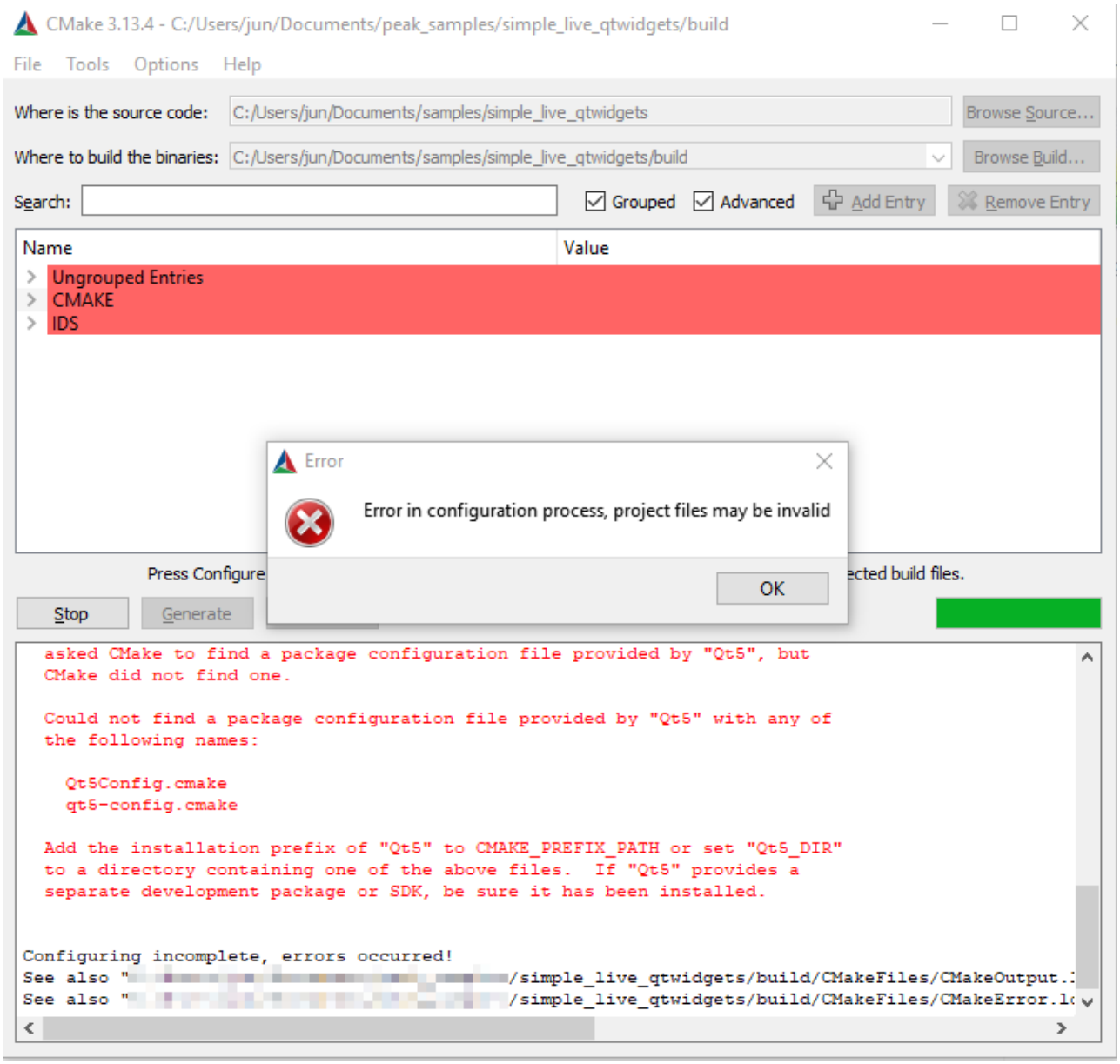

Fig. 5: Configuration error

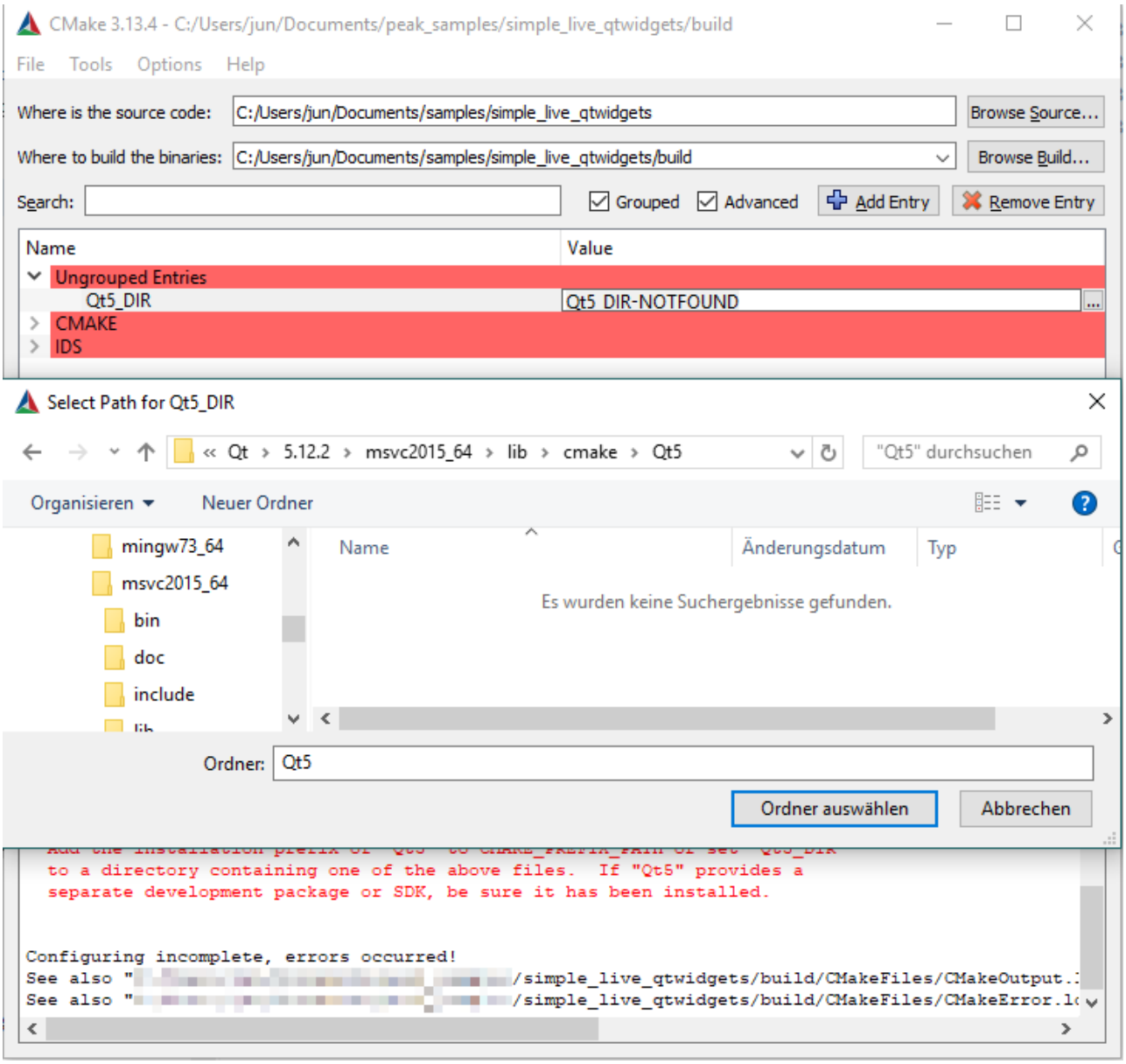

Fig. 6: Correct error

Then click "Configure" again. If no error message is displayed, the configuration was successful.

4 Basic principles

It is recommended to observe some basic principles when working with SICK Vision Suite. This makes programming easier and error cases can be better handled or avoided.

- Error handling with try/catch
- Shared pointer (C++)
- Descriptors

4.1 Error handling with try/catch

If an error occurs, SICK Vision Suite throws an exception (C++ exception). Use try-catch blocks for exception handling. A try-catch block marks some of the statements to be executed and defines an action when an exception occurs.

C++

```
try
{
    // do something here
    // ...
}
catch (const std::exception& e)
{
    std::cout << "EXCEPTION: " << e.what() << std::endl;
}
```

C#

```
try
{
    // do something here
    // ...
catch (System.ApplicationException e)
{
    Console.WriteLine("Exception: " + e.Message);
}
```

4.2 Shared pointer (C++)

SICK Vision Suite uses shared pointers, for example, for device lists, data streams, or nodes in the NodeMap of a device. Shared pointers (intelligent pointers, smart pointers) are special pointers that have additional features and functions compared to simple pointers, such as automatic buffer management. These functions are intended to reduce errors caused by incorrect use of pointers. Efficiency is not compromised.

It is recommended to use the "auto" placeholder of C++ if possible. This ensures that the intended shared pointers are automatically used in the correct way.

```
// open the first camera
auto device = deviceManager.Devices().at(0)-
>OpenDevice(vision_api::core::DeviceAccessType::Control);
```

4.3 Descriptors

Descriptors are deputy classes. SICK Vision Suite uses this concept for all modules that can be opened (Device, Interface, System). Descriptors make it possible to get information even if the corresponding module has not yet been opened. The functions of the opened modules are only available when the module has really been opened. This reduces errors in module handling.

C++

```
// device descriptor for the first camera
auto deviceDescriptor = deviceManager.Devices().at(0);

// open first camera by calling the open function of the device descriptor
auto device = deviceDescriptor-
>OpenDevice(vision_api::core::DeviceAccessType::Control);
```

C#

```
// device descriptor for the first camera
var deviceDescriptor = deviceManager.Devices()[0];

// open first camera by calling the open function of the device descriptor
var device = deviceDescriptor.OpenDevice(DeviceAccessType.Control);
```

5 Important classes and functions

A detailed documentation of the classes and functions that SICK Vision Suite provides can be found in the separate documentation for SICK Vision API and SICK LibIMG.

- Library
- DeviceManager
- Device
- DataStream
- Buffer
- BufferTo
- Image

5.1 Library

Namespace C#	`vision_api`
Namespace C++	`vision_api`
Include C++	`<vision_api/vision_api.hpp>`

Use the "Library " class in SICK Vision Suite to initialize and close the SICK Vision Suite library.

- Initialize
- Close

5.1.1 Initialize

Before classes and functions of SICK Vision Suite can be used, the library must be initialized.

C++

```
// initialize library
vision_api::Library::Initialize();
```

C#

```
// initialize library
vision_api.Library.Initialize();
```

5.1.2 Close

Before closing an application with SICK Vision Suite, the "Library" must be closed.

C++

```
// close library before exiting program
vision_api::Library::Close();
```

C#

```
// close peak library
vision_api.Library.Close();
```

5.2 DeviceManager

Namespace C#	`vision_api`
Namespace C++	`vision_api`
Include C++	`<vision_api/vision_api.hpp>`

The `DeviceManager` handles the module handling in SICK Vision Suite. It detects all available systems, interfaces and devices. An instance of the `DeviceManager` is created using the Instance function.

C++

```
// create a camera manager object
auto& deviceManager = vision_api::DeviceManager::Instance();
```

C#

```
// create a device manager object
var deviceManager = vision_api.DeviceManager.Instance();
```

- Update
- Update with selected CTI (GenTL)
- Opening a camera
- Further functions of the DeviceManager
- Examples for using the DeviceManager

5.2.1 Update

After an instance of the `DeviceManager` has been created, an `Update()` must be performed. The `DeviceManager` scans for available modules and stores them in the lists `Systems(), Interfaces()` and `Devices()`.

C++

```
// update the camera manager
deviceManager.Update();
```

C#

```
// update the deviceManager
deviceManager.Update();
```

A simple example how to use the `DeviceManager` can be found in the sample "open_camera".

5.2.2 Update with selected CTI (GenTL)

A GenTL Producer implementation is provided as a platform-dependent, dynamically loadable library. On Microsoft Windows platforms, this is a Dynamic Link Library (*.dll). The file extension of this library is *.cti (Common Transport Interface).

The `DeviceManager` offers the possibility to select a CTI and to perform the update only with this CTI. This ensures that only the selected transport layer is used on a system where multiple GenTL implementations are installed.

C++

```
// define the CTI that will be used
std::string cti = "C:\Program
Files\SICK\SICKVisionSuite\sick_gevgentl\64\sick_gevgentlk.cti>";

// add CTI to DeviceManager
deviceManager.AddProducerLibrary(cti);

// update the DeviceManager
deviceManager.Update(vision_api::DeviceManager::UpdatePolicy::DontScanEnvironmen
tForProducerLibraries);
```

C#

```
// define the CTI that will be used
var = "C:\Program
Files\SICK\SICKVisionSuite\sick_gevgentl\64\sick_gevgentlk.cti";

// add CTI to DeviceManager
deviceManager.AddProducerLibrary(cti);

// update the DeviceManager
deviceManager.Update(peak.DeviceManager.UpdatePolicy.DontScanEnvironmentForProdu
cerLibraries);
```

A simple example how to use the `DeviceManager` with a pre-selected CTI can be found in the sample "open_camera_select_cti".

5.2.3 Opening a camera

After the `DeviceManager` has scanned the system for available cameras, these are listed as descriptors in the `Devices()` list. The OpenDevice function can be used to open the device module. Here, the access type is passed, e.g. `Control, Exclusive` or `ReadOnly`.

C++

```
// open the first camera
auto device = deviceManager.Devices().at(0)-
>OpenDevice(vision_api::core::DeviceAccessType::Control);
```

C#

```
// open the selected device
var device = deviceManager.Devices()
[0].OpenDevice(vision_api.core.DeviceAccessType.Control);
```

A simple example how to use the `DeviceManager` for opening a camera can be found in the sample "open_camera".

5.2.4 Further functions of the DeviceManager

Using the `DeviceManager` the following callbacks can be registered and unregistered:

- `DeviceFound / DeviceLost`
- `InterfaceFound / InterfaceLost`
- `SystemFound / SystemLost`

C++

```
// register DeviceFound callback
auto callbackHandle = deviceManager.RegisterDeviceFoundCallback([](const
std::shared_ptr<vision_api::core::DeviceDescriptor>& foundDevice)
{
    std::cout << "DeviceFound: " << foundDevice->DisplayName() << std::endl;
});

// unregister DeviceFound callback
deviceManager.UnregisterDeviceFoundCallback(callbackHandle);
```

C#

```
// define DeviceFound callback
static void DeviceFoundDelegate(object o, DeviceDescriptor dev)
{
```

```
    Console.WriteLine("C# static function DeviceFound: {0}", dev.DisplayName());
}
...
// register DeviceFound callback
deviceManager.DeviceFoundEvent += DeviceFoundDelegate;

// unregister DeviceFound callback
deviceManager.DeviceFoundEvent -= DeviceFoundDelegate;
```

Also the following timeouts can be set:

- `DeviceUpdate`
- `InterfaceUpdate`
- `SystemUpdate`

5.2.5 Examples for using the DeviceManager

Example	Use of the DeviceManager
open_camera	Simple create and update, open a camera from the list `Devices()`
open_camera_by_serno	Create and update, open a camera by its serial number
open_camera_select_cti	Create and update for a selected CTI, open a camera from the list `Devices()`
device_tree	Create and update, structure of a module tree, open a camera from the list `Devices()`
walkthrough	Detailed module tree and open the first available camera

5.3 Device

Namespace C#	`vision_api.core`
Namespace C++	`vision_api::core`
Include C++	`<vision_api/vision_api.hpp>`

A `Device` is generated automatically when the OpenDevice function of a `DeviceDescriptor` is executed.

C++

```
// open camera by calling the open function of the device descriptor
auto device = deviceDescriptor-
>OpenDevice(vision_api::core::DeviceAccessType::Control);
```

C#

```
// open camera by calling the open function of the device descriptor
var device =
deviceDescriptor.OpenDevice(vision_api.core.DeviceAccessType.Control);
```

5.3.1 Changing settings

A `Device` has a `RemoteDevice` class that can access the device as such. With the `NodeMap` of this `RemoteDevice` it is possible to access camera settings.

C++

```
// get the remote device node map
auto nodeMapRemoteDevice = device->RemoteDevice()->NodeMaps().at(0);
```

C#

```
// get the remote device node map
var nodeMapRemoteDevice = device.RemoteDevice().NodeMaps()[0];
```

To change settings, the corresponding node of the setting must be found in the `NodeMap`. The name of the node and its type must be known. The various setting options and related node types can be found in the SICK Vision Reference. If the type and name of the node are known, it can be found using the FindNode function.

C++

```
// get the node for the frame rate
auto nodeFrameRate = nodeMapRemoteDevice-
>FindNode<vision_api::core::nodes::FloatNode>("AcquisitionFrameRate");
```

C#

```
// get the node for the frame rate
var nodeFramerate =
nodeMapRemoteDevice.FindNode<vision_api.core.nodes.FloatNode>("AcquisitionFrameR
ate");
```

Nodes can be of the following types:

Type	Description
Category	Category under which several nodes are grouped.
String	Character string
Boolean	Boolean value (**True** or **False**)
Integer	Integer number (64 bit)
Float	Floating point number
Enumeration	Discrete value list
EnumerationEntry	Entry in a value list
Command	Executable command

The handling of the nodes depends on their type. For example, `EnumerationNodes` have a `Entries` value list that contains all possible values. However, `IntegerNodes` and `FloatNodes` provide information about the permissible value range by querying the `Minimum()` and `Maximum()`.

- Code example for EnumerationNode
- Code example for FloatNode

5.3.1.1 Code example for EnumerationNode

C++

```
// get the node for the trigger selector
auto nodeTriggerSelector = nodeMapRemoteDevice-
>FindNode<vision_api::core::nodes::EnumerationNode>("TriggerSelector");

// read the current entry
auto currentEntry = nodeTriggerSelector->CurrentEntry();

// get a list of all possible entries
auto entries = nodeTriggerSelector->Entries();

// set the selector to the first item of its possible entries list
nodeTriggerSelector->SetCurrentEntry(entries.at(0));

// or set the selector to a specific entry
nodeTriggerSelector->SetCurrentEntry("ExposureStart");
```

C#

```
// get the node for the trigger selector
var nodeTriggerSelector =
nodeMapRemoteDevice.FindNode<vision_api.core.nodes.EnumerationNode>("TriggerSele
ctor");

// read the current entry
var currentEntry = nodeTriggerSelector.CurrentEntry();

// get a list of all possible entries
var entries = nodeTriggerSelector.Entries();

// set the selector to the first item of its possible entries list
nodeTriggerSelector.SetCurrentEntry(entries[0]);

// or set the selector to a specific entry
nodeTriggerSelector.SetCurrentEntry("ExposureStart");
```

5.3.1.2 Code example for FloatNode

C++

```
// get the node for frame rate
auto nodeFrameRate = nodeMapRemoteDevice-
>FindNode<vision_api::core::nodes::FloatNode>("AcquisitionFrameRate");

// read the current value
auto value = nodeFrameRate->Value();

// get the value range
auto valueMin = nodeFrameRate->Minimum();
auto valueMax = nodeFrameRate->Maximum();
auto valueInc = nodeFrameRate->Increment();

// get the unit of the node
auto unit = nodeFrameRate->Unit();

// set a new frame rate
nodeFrameRate->SetValue(10);
```

C#

```
// get the node for frame rate
var nodeFrameRate =
nodeMapRemoteDevice.FindNode<vision_api.core.nodes.FloatNode>("AcquisitionFrameR
ate");

// read the current value
var value = nodeFrameRate.Value();

// get the value range
var valueMin = nodeFrameRate.Minimum();
var valueMax = nodeFrameRate.Maximum();
var valueInc = nodeFrameRate.Increment();

var unit = nodeFrameRate.Unit();

// set a new frame rate
nodeFrameRate.SetValue(10);
```

5.4 DataStream

Namespace C#	`vision_api.core`
Namespace C++	`vision_api::core`
Include C++	`<vision_api/vision_api.hpp>`

A `DataStream` is generated automatically when the OpenDataStream function of a `DataStreamDescriptor` is executed.

C++

```
// first data stream descriptor of the camera
auto dataStreamDescriptor = device->DataStreams().at(0);

// open the data stream
auto dataStream = dataStreamDescriptor->OpenDataStream();
```

C#

```
// first data stream descriptor of the camera
var dataStreamDescriptor = device.DataStreams()[0];

// open the data stream
var dataStream = dataStreamDescriptor.OpenDataStream();
```

5.5 Buffer

Namespace C#	`vision_api.core`
Namespace C++	`vision_api::core`
Include C++	`<vision_api/vision_api.hpp>`

A `buffer` is returned by a `DataStream` using the WaitForFinishedBuffer function and contains the image data transmitted by the `device`.

C++

```
// get buffer from data stream
// timeout value is set to 5000 ms
auto buffer = dataStream->WaitForFinishedBuffer(5000);
```

C#

```
// get buffer from data stream
// timeout value is set to 5000 ms
var buffer = dataStream.WaitForFinishedBuffer(5000);
```

5.6 BufferTo

Namespace C#	`vision_api.core`
Namespace C++	`vision_api::core`
Include C++	`<vision_api/converters/vision_api_buffer_converter_libimg.hpp>`

The BufferTo function converts inline a buffer transmitted via the camera data stream into an image of the SICK LibIMG (image processing library). The SICK LibIMG can be used to perform image operations (`PixelLine, Histogram, …`) or to get image information (`Width, Height, PixelFormat, …`). A detailed documentation of the classes and functions that SICK LibIMG provides can be found in the separate documentation for SICK LibIMG.

C++

```
// create SICK LibIMG image from buffer
auto image = vision_api::BufferTo<libimg::Image>(buffer);
```

C#

```
// Note that the BufferTo function currently does not exist in C#
// Create SICK LibIMG image
var image = new vision_api.libimg.Image((libimg.PixelFormatName)
buffer.PixelFormat(), buffer.BasePtr(), buffer.Size(), buffer.Width(),
buffer.Height());
```

In this way, conversion functions for other/own data types can be implemented via further specializations of the template function.

5.6.1 Examples for using the BufferTo function

Example	Use of the BufferTo function
get_first_pixel	Conversion of a `buffer` to an `image`
simple_live_qml	Conversion of a `buffer` to an `image` in `acquisitionworker.cpp`
simple_live_qtwidgets	Conversion of a `buffer` to an `image` in `acquisitionworker.cpp`
simple_live_windows_forms	Conversion of a `buffer` to an `image` in `AcquisitionWorker.cs`
simple_live_windows_wpf	Conversion of a `buffer` to an `image` in `AcquisitionWorker.cs`

5.7 Image

Namespace C#	`vision_api.libimg`
Namespace C++	`libimg`
Include C++	`<libimg/libimg.hpp>`

The Image class is part of the SICK LibIMG (image processing library). The SICK LibIMG can be used to perform image operations (`PixelLine, Histogram, …`) or to get image information (`Width, Height, PixelFormat, …`). A detailed documentation of the classes and functions that SICK LibIMG provides can be found in the separate documentation for SICK LibIMG.

An `image` can be created from a `buffer` using the BufferTo function.

C++

```
// create SICK LibIMG image from buffer
auto image = vision_api::BufferTo<libimg::Image>(buffer);
```

C#

```
// Note that the BufferTo function currently does not exist in C#
// Create SICK LibIMG image
var image = new vision_api.libimg.Image((libimg.PixelFormatName)
buffer.PixelFormat(), buffer.BasePtr(), buffer.Size(), buffer.Width(),
buffer.Height());
```

5.7.1 ConvertTo

The buffer transmitted via the data stream is often in a raw format, e.g. BayerRG8. However, a debayered RGB format is typically required for the display, e.g. RGB8. With the ConvertTo function an image can be converted inline from one pixel format to another (if necessary with debayering).

C++

```
// create SICK LibIMG image for debayering and convert it to RGB8 format
image = image.ConvertTo(libimg::PixelFormatName::RGB8);
```

C#

```
// Debayering to IPL RGB8 format
iplImg = iplImg.ConvertTo(libimg.PixelFormatName.BGR8);
```

5.7.2 Examples for using the Image class and the ConvertTo function

Example	Use oft the Image class and the ConvertTo function
get_first_pixel	Conversion of a `buffer` to an `image`
simple_live_qml	Conversion of a `buffer` to an `image` and conversion of the pixel format with `ConvertTo()` to RGB8 in `acquisitionworker.cpp`
simple_live_qtwidgets	Conversion of a `buffer` to an `image` and conversion of the pixel format with `ConvertTo()` to RGB8 in `acquisitionworker.cpp`
simple_live_windows_forms	Conversion of a `buffer` to an `image` and conversion of the pixel format with `ConvertTo()` to RGB8 in `AcquisitionWorker.cs`
simple_live_windows_wpf	Conversion of a `buffer` to an `image` and conversion of the pixel format with `ConvertTo()` to RGB8 in `AcquisitionWorker.cs`

6 Program sequence (capture 1st image)

6.1 Program start

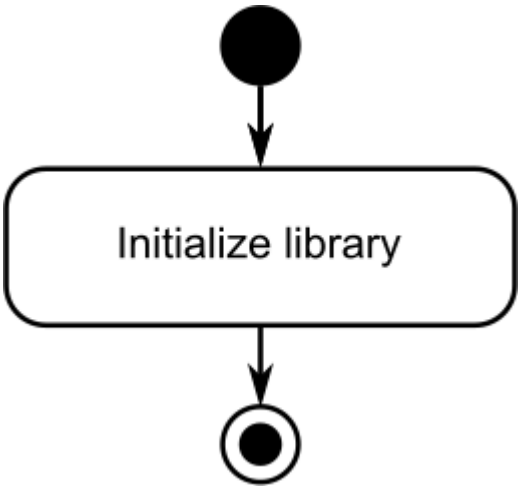

Fig. 7: Program start

```
// initialize library
vision_api::Library::Initialize();
```

6.2 Discovering cameras

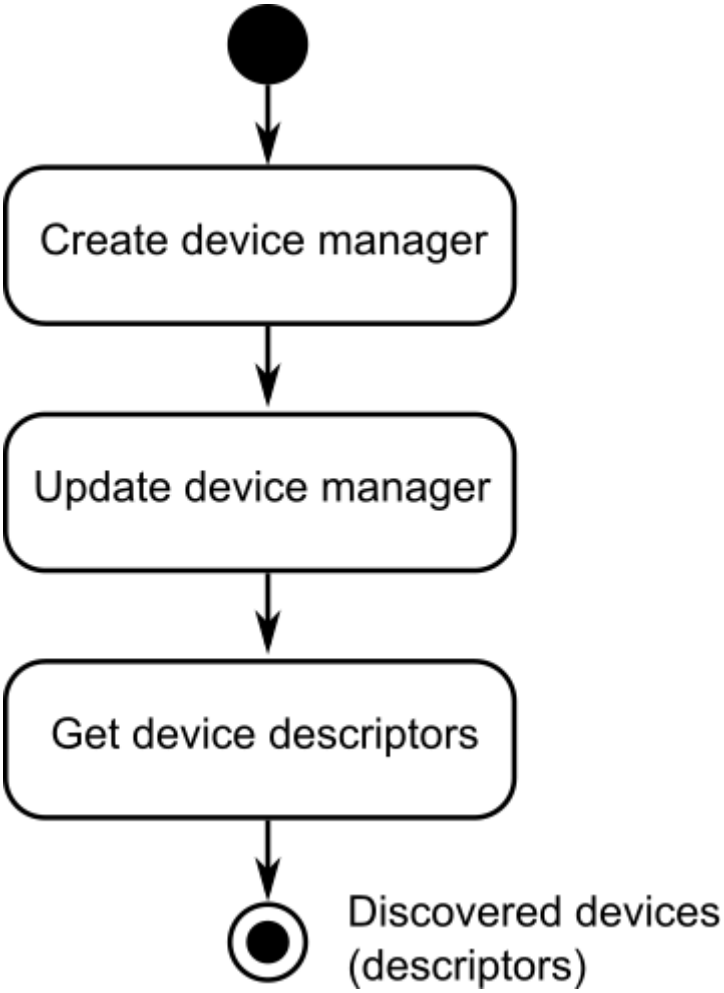

Fig. 8: Discover cameras

```
// create a device manager object
auto& deviceManager = vision_api::DeviceManager::Instance();

try
{
    // update the device manager
    deviceManager.Update();

    // get vector of device descriptors
    auto devices() = deviceManager->Devices();

   // ...
}
catch (const std::exception& e)
{
    std::cout << "EXCEPTION: " << e.what() << std::endl;
}
```

6.3 Opening a camera

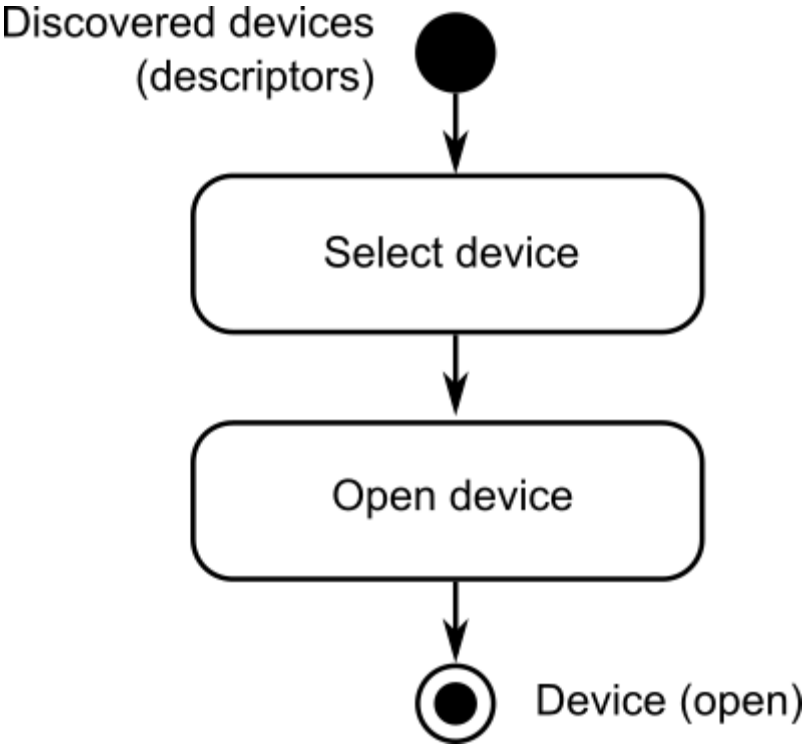

Fig. 9: Open a camera

```
try
{
    // ...

    // select a device to open
    size_t selectedDevice = 0;

    // open the selected device
    auto device = deviceManager.Devices().at(selectedDevice)-
>OpenDevice(vision_api::core::DeviceAccessType::Control);

    // ...
}
catch (const std::exception& e)
{
    std::cout << "EXCEPTION: " << e.what() << std::endl;
}
```

6.4 Adjusting settings

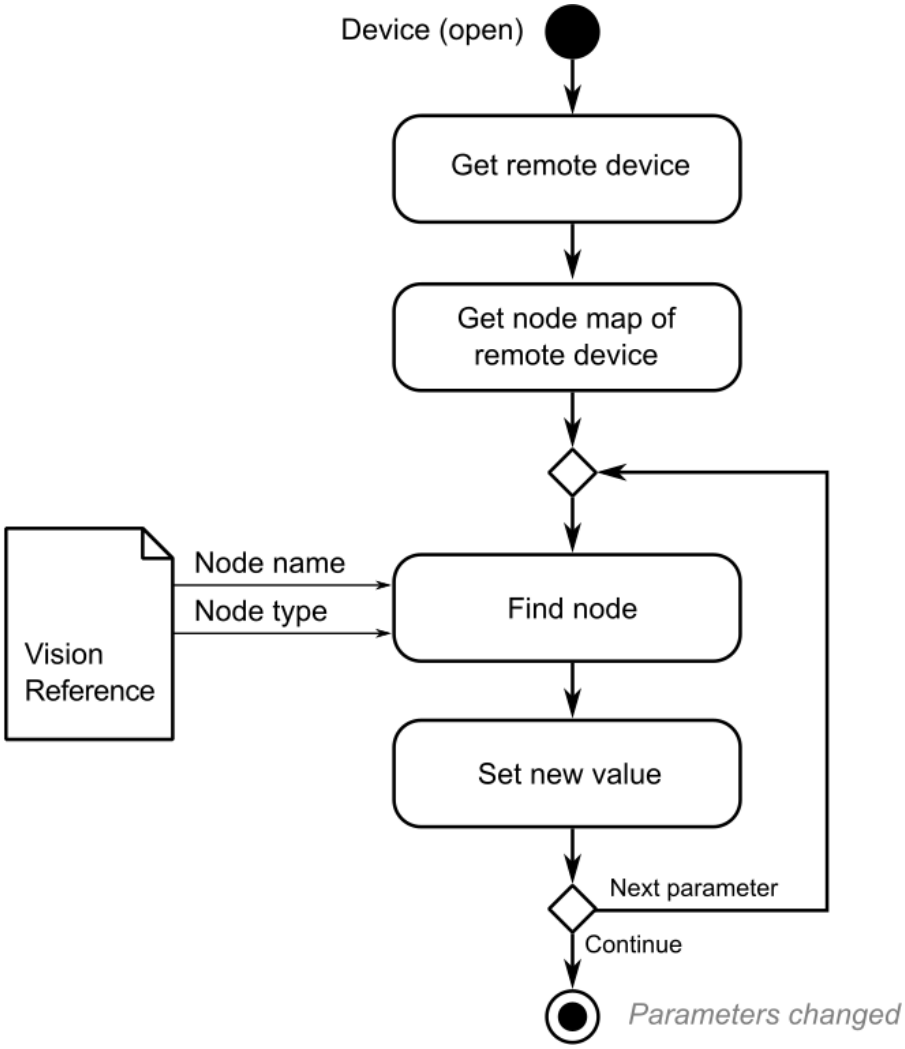

Fig. 10: Change parameters

```
try
{
    // ...

    // get the remote device node map
    auto nodeMapRemoteDevice = device->RemoteDevice()->NodeMaps().at(0);

    // find the parameter node, e.g. 'AcquisitionFrameRate', and set a new value
    nodeMapRemoteDevice-
>FindNode<vision_api::core::nodes::FloatNode>("AcquisitionFrameRate")-
>SetValue(10);

    // ...
}
catch (const std::exception& e)
{
    std::cout << "EXCEPTION: " << e.what() << std::endl;
}
```

6.5 Opening a data stream and creating a buffer

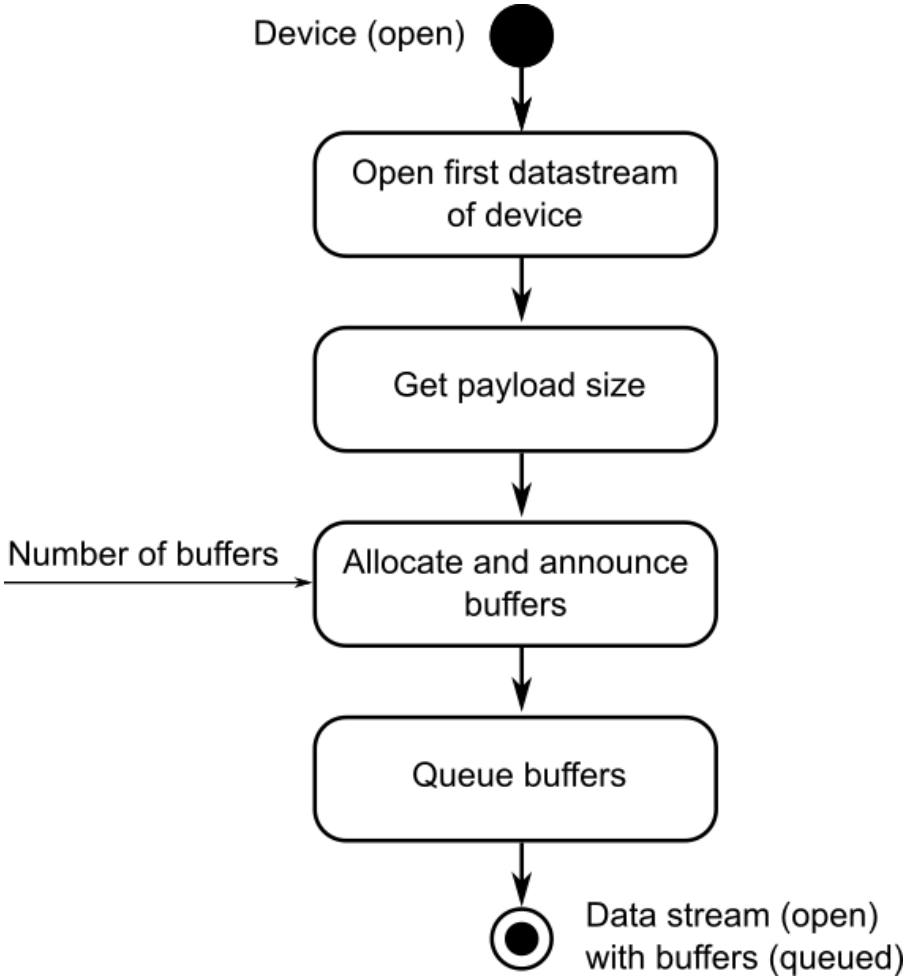

Fig. 11: Data stream and buffer

```
try
{
    // ...

    // open the first data stream
    auto dataStream = device->DataStreams().at(0)->OpenDataStream();

    // get payload size
    auto payloadSize = nodeMapRemoteDevice-
>FindNode<vision_api::core::nodes::IntegerNode>("PayloadSize")->Value();

    // get number of buffers to allocate
    // the buffer count depends on your application, here the minimum required
number for the data stream
    auto bufferCountMax = dataStream->NumBuffersAnnouncedMinRequired();

    // allocate and announce image buffers and queue them
    for (uint64_t bufferCount = 0; bufferCount < bufferCountMax; ++bufferCount)
    {
        auto buffer = dataStream-
>AllocAndAnnounceBuffer(static_cast<size_t>(payloadSize), nullptr);
        dataStream->QueueBuffer(buffer);
    }
```

```
    // ...
}
catch (const std::exception& e)
{
    std::cout << "EXCEPTION: " << e.what() << std::endl;
}
```

6.6 Starting image acquisition

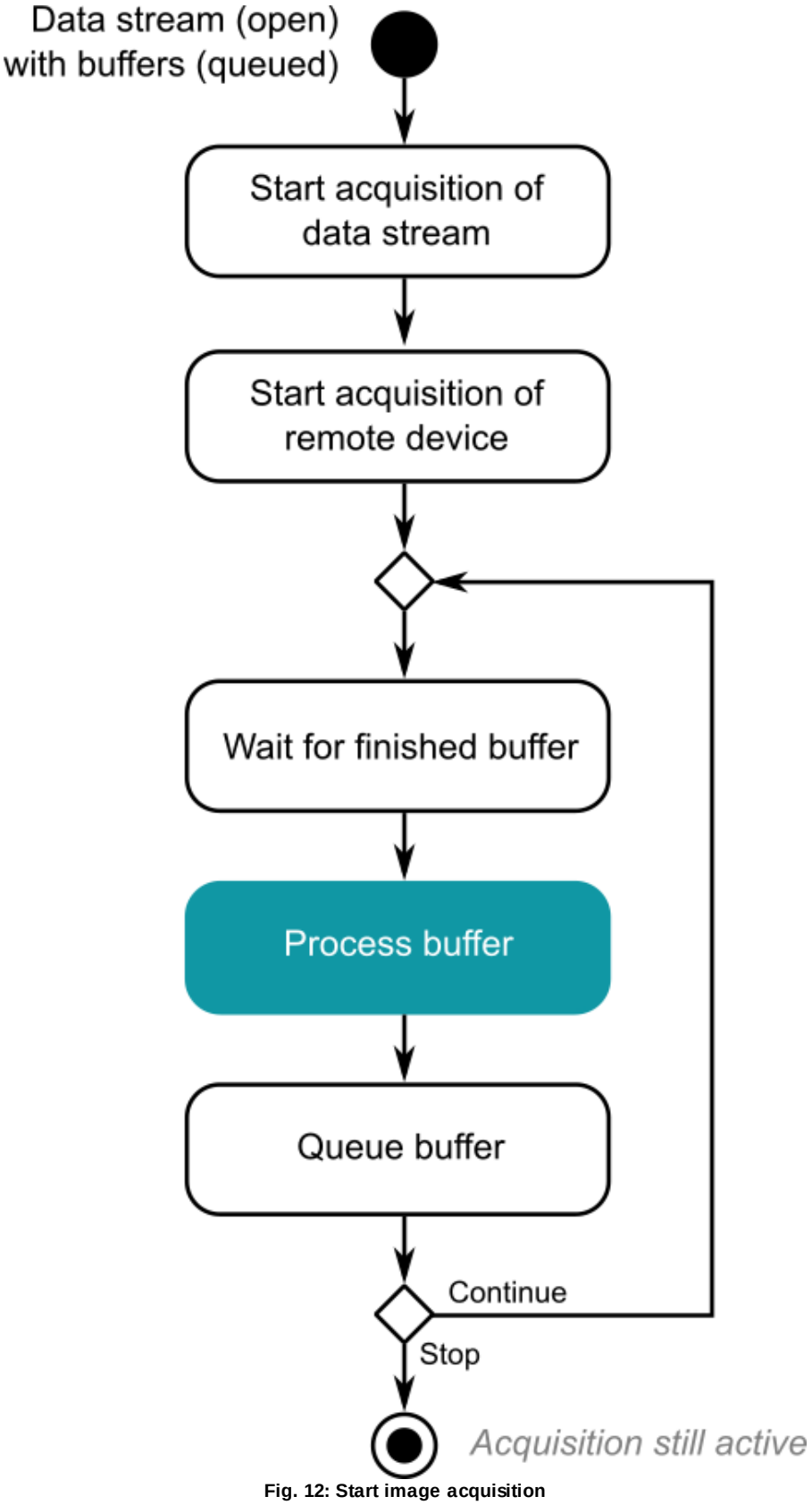

Fig. 12: Start image acquisition

```
try
{
    // ...

    // start acquisition with infinite number of images to acquire

    // start the data stream
    m_dataStream-
>StartAcquisition(vision_api::core::AcquisitionStartMode::Default,
vision_api::core::INFINITE_NUMBER);

    // start the device
    nodeMapRemoteDevice-
>FindNode<vision_api::core::nodes::CommandNode>("AcquisitionStart")->Execute();

    // the acquisition loop
    bool is_running = true;
    while (is_running)
    {
        // get buffer from data stream and process it
        auto buffer = dataStream->WaitForFinishedBuffer(5000);

        // buffer processing start
        // ....
        // buffer processing end

        // queue buffer
        dataStream->QueueBuffer(buffer);
    }

    // ...
}
catch (const std::exception& e)
{
    std::cout << "EXCEPTION: " << e.what() << std::endl;
}
```

Note on image acquisition
Typically image acquisition is done in a separate thread and not in the main thread, see also samples "SimpleLiveQml" and "SimpleLiveQtWidgets".

6.7 Stopping image acquisition and cleaning up buffers

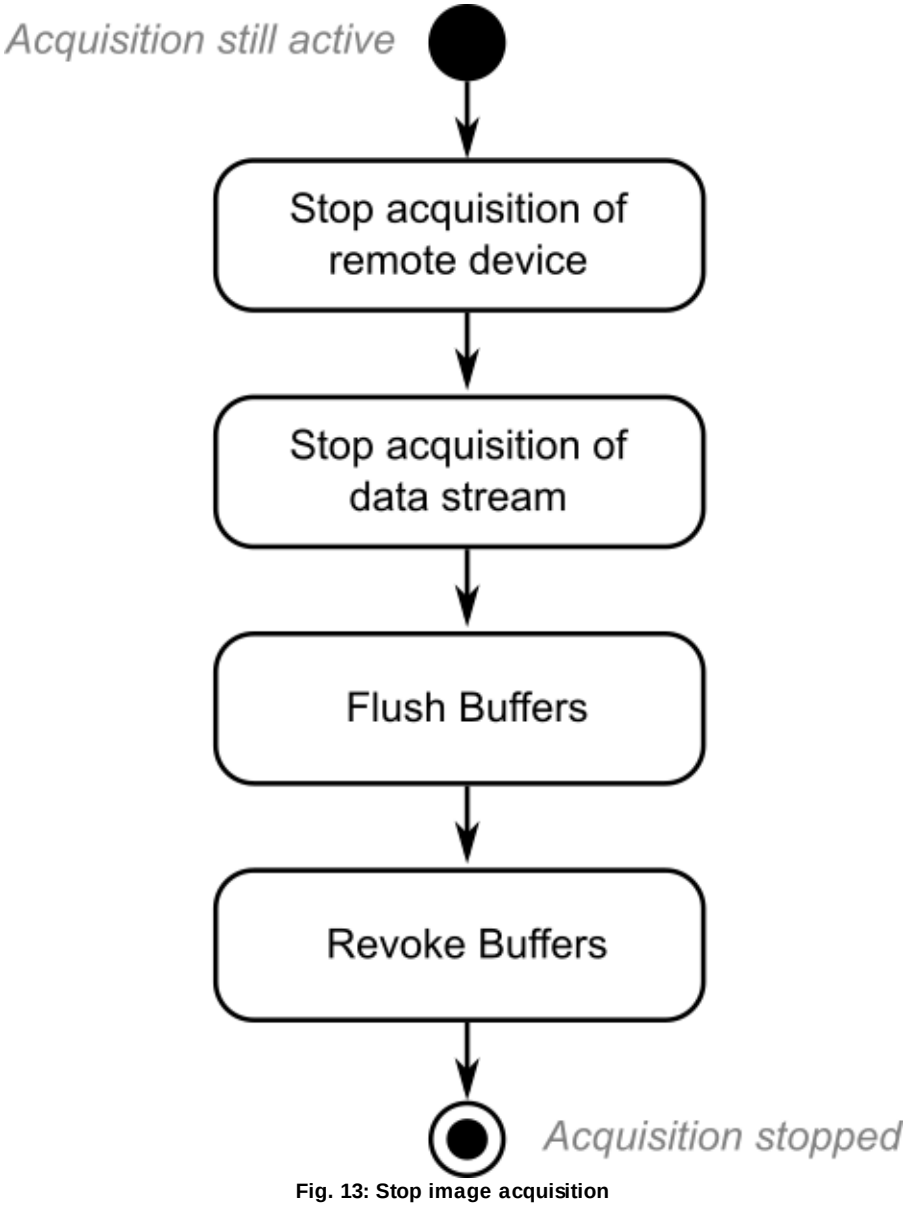

Fig. 13: Stop image acquisition

```
try
{
    // ...

    // stop acquisition

    // stop the data stream
    dataStream->StopAcquisition(vision_api::core::AcquisitionStopMode::Default);

    // stop the device
    nodeMapRemoteDevice-
>FindNode<vision_api::core::nodes::CommandNode>("AcquisitionStop")->Execute();

    // flush all buffers
    dataStream->Flush(vision_api::core::DataStreamFlushMode::DiscardAll);

    // revoke all buffers
```

```
    for (const auto& buffer : dataStream->AnnouncedBuffers())
    {
        dataStream->RevokeBuffer(buffer);
    }
}
catch (const std::exception& e)
{
    std::cout << "EXCEPTION: " << e.what() << std::endl;
}
```

6.8 Program end

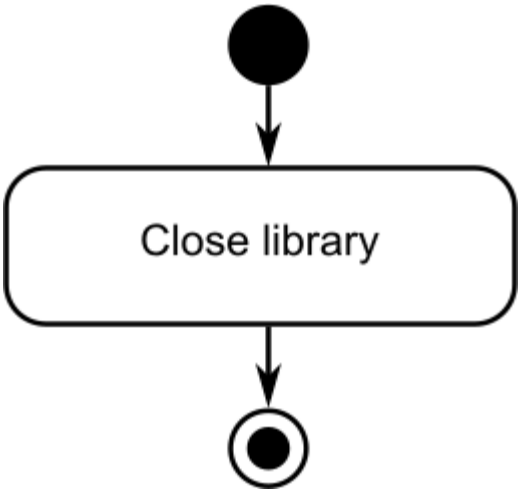

Fig. 14: Program end

```
try
{
    // ...
}
catch (const std::exception& e)
{
    std::cout << "EXCEPTION: " << e.what() << std::endl;
}

// close library before exiting program
vision_api::Library::Close();
```

- R -

- S -

- T -

8025624/0000/2020-04/en

Australia
Phone +61 (3) 9457 0600
1800 33 48 02 – tollfree
E-Mail sales@sick.com.au

Austria
Phone +43 (0) 2236 62288-0
E-Mail office@sick.at

Belgium/Luxembourg
Phone +32 (0) 2 466 55 66
E-Mail info@sick.be

Brazil
Phone +55 11 3215-4900
E-Mail comercial@sick.com.br

Canada
Phone +1 905.771.1444
E-Mail cs.canada@sick.com

Czech Republic
Phone +420 234 719 500
E-Mail sick@sick.cz

Chile
Phone +56 (2) 2274 7430
E-Mail chile@sick.com

China
Phone +86 20 2882 3600
E-Mail info.china@sick.net.cn

Denmark
Phone +45 45 82 64 00
E-Mail sick@sick.dk

Finland
Phone +358-9-25 15 800
E-Mail sick@sick.fi

France
Phone +33 1 64 62 35 00
E-Mail info@sick.fr

Germany
Phone +49 (0) 2 11 53 010
E-Mail info@sick.de

Greece
Phone +30 210 6825100
E-Mail office@sick.com.gr

Hong Kong
Phone +852 2153 6300
E-Mail ghk@sick.com.hk

Hungary
Phone +36 1 371 2680
E-Mail ertekesites@sick.hu

India
Phone +91-22-6119 8900
E-Mail info@sick-india.com

Israel
Phone +972 97110 11
E-Mail info@sick-sensors.com

Italy
Phone +39 02 27 43 41
E-Mail info@sick.it

Japan
Phone +81 3 5309 2112
E-Mail support@sick.jp

Malaysia
Phone +603-8080 7425
E-Mail enquiry.my@sick.com

Mexico
Phone +52 (472) 748 9451
E-Mail mexico@sick.com

Netherlands
Phone +31 (0) 30 229 25 44
E-Mail info@sick.nl

New Zealand
Phone +64 9 415 0459
0800 222 278 – tollfree
E-Mail sales@sick.co.nz

Norway
Phone +47 67 81 50 00
E-Mail sick@sick.no

Poland
Phone +48 22 539 41 00
E-Mail info@sick.pl

Romania
Phone +40 356-17 11 20
E-Mail office@sick.ro

Russia
Phone +7 495 283 09 90
E-Mail info@sick.ru

Singapore
Phone +65 6744 3732
E-Mail sales.gsg@sick.com

Slovakia
Phone +421 482 901 201
E-Mail mail@sick-sk.sk

Slovenia
Phone +386 591 78849
E-Mail office@sick.si

South Africa
Phone +27 10 060 0550
E-Mail info@sickautomation.co.za

South Korea
Phone +82 2 786 6321/4
E-Mail infokorea@sick.com

Spain
Phone +34 93 480 31 00
E-Mail info@sick.es

Sweden
Phone +46 10 110 10 00
E-Mail info@sick.se

Switzerland
Phone +41 41 619 29 39
E-Mail contact@sick.ch

Taiwan
Phone +886-2-2375-6288
E-Mail sales@sick.com.tw

Thailand
Phone +66 2 645 0009
E-Mail marcom.th@sick.com

Turkey
Phone +90 (216) 528 50 00
E-Mail info@sick.com.tr

United Arab Emirates
Phone +971 (0) 4 88 65 878
E-Mail contact@sick.ae

United Kingdom
Phone +44 (0)17278 31121
E-Mail info@sick.co.uk

USA
Phone +1 800.325.7425
E-Mail info@sick.com

Vietnam
Phone +65 6744 3732
E-Mail sales.gsg@sick.com

Detailed addresses and further locations at **www.sick.com**

SICK AG | Waldkirch | Germany | www.sick.com

SICK
Sensor Intelligence.