Testo Camera Application Interface for testo 885/testo 890

Testo part number 0501.8985

Table of Content

1. Overview

This document describes the testo C++ Camera Application Interface for the thermal imagers testo 890/testo 885. This interface provides several C++ objects and functions to capture thermal images from the camera and for reading and changing camera parameters. This interface does not provide access to thermal images (bmt files) in a file system. For this purpose you need IrApi (part number 0501 8987 for testo 865/testo 868/testo 871/ testo 872 and 0501 8984 for all other cameras).

The target audience of this API are software developers who build customer specialized interface applications for the testo thermal imagers using Microsoft Visual Studio or a compatible compiler on Windows Systems.

1.1 Supported operating systems

The *Camera Interface* supports following operation systems:

- Windows 7 SP 1 (x86, x64)
- Windows 8.1 (x86, x64)
- Windows 10 (x86, x64)

1.2 Supported testo cameras

- t885 firmware version >1.64
- t890 firmware version >1.64

1.3 Recommended Configurations

- Windows 7 SP 1 (x86, x64), VisualStudio 2013, cmake 3.0.1, IRSoft 3.7 or higher
- Windows 8.1 x64 Pro, VisualStudio 2013, cmake 3.0.1, IRSoft 3.7 or higher
- Windows 10 x64 Pro, VisualStudio 2013, cmake 3.0.1, IRSoft 3.7 or higher

2. External dependencies and requirements

Required Software

- IR-Soft 3.6 or higher
  https://www.testo.com/irsoft
- Microsoft Visual C++ Redistributable Packages for Visual Studio 2013
  https://www.microsoft.com/de-de/download/details.aspx?id=40784
  (or installed Microsoft Visual Studio 2013)
- Cmake 3.0.1 or higher (optional needed to use the interface example and integrated cmake scripts)

  Note the API binaries can also be used without the cmake support by setting up the project manually.

Third party libraries

- open_cv (version: 2.4.8)
- gtest (only needed to run the test cases)

Compiler

The Camera API binaries and the external third parties are compiled with Visual Studio 2013 32bit.

Development Skills

The developer should be familiar with the following domains:

- C++, VisualStudio 20xx
- Image processing
- Thermography
- Cmake (optional)

3. Installation

Install IRSoft including the Microsoft Visual C++ Redistributable Packages or Visual Studio 2013. To use the integrated cmake scripts e.g. to generating the Visual Studio example project files install cmake.

Unpack the “campera_api.zip” in a target folder.

4. Package content

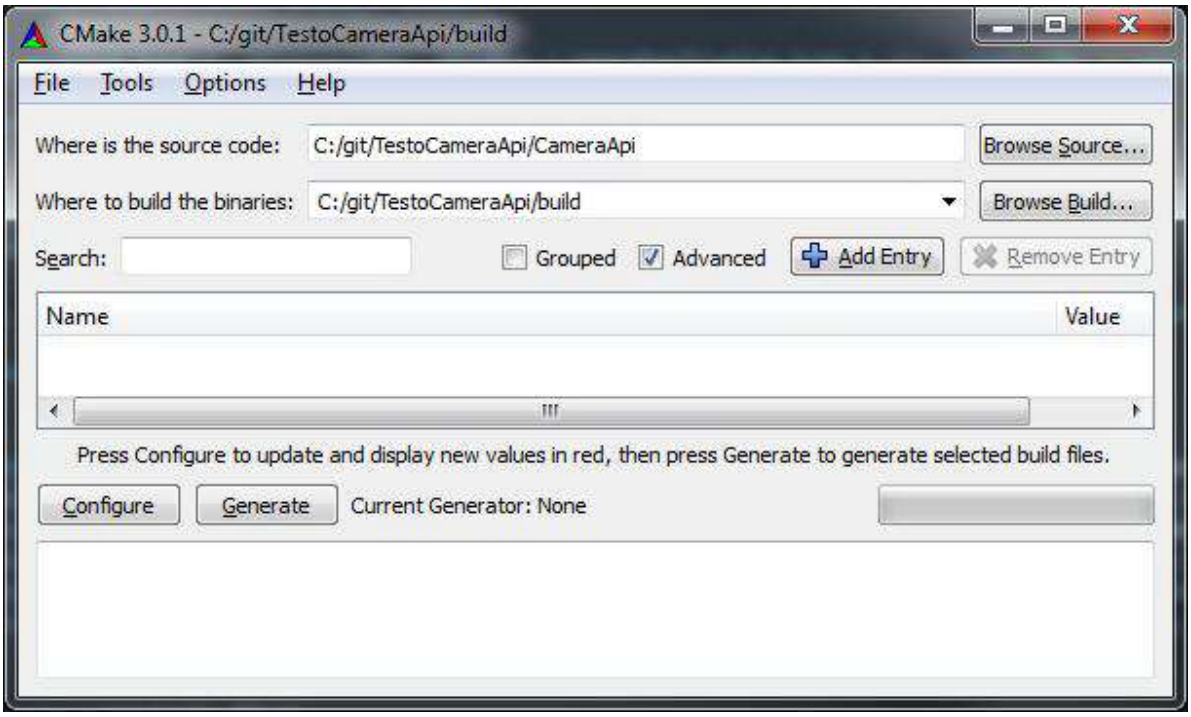
The package contains a “CameraApi” folder and a “thirdparty” folder.
Note: The camera interface binaries are located in the thirdparty\shared\tilib folder.

Folder	**Notes of content**
CameraApi	Example projects for the camera API
CameraApi \configure	Intern config files for the example cmake scripts
CameraApi\doc	interface documentation
CameraApi\test	gtest example using the camera api
CameraApi\ labview_cwrapper	Example library that adapts the camera interface to labview
Thirdparty	Contains external in internal third parties
thirdparty\shared\gtest	gtest binaries 32Bit (build with Visual Studio 2013)
thirdparty\shared\opencv	opencv binaries 32Bit (build with Visual Studio 2013)
thirdparty\shared\tilib	library with the actual camera interface binaries
thirdparty\shared\tilib\include	Thermal image library headers
thirdparty\shared\tilib\include\ externinterface	**Header files for the camera interface**
thirdparty\shared\tilib\bin	Binaries files for the camera interface (DEBUG and RELEASE)

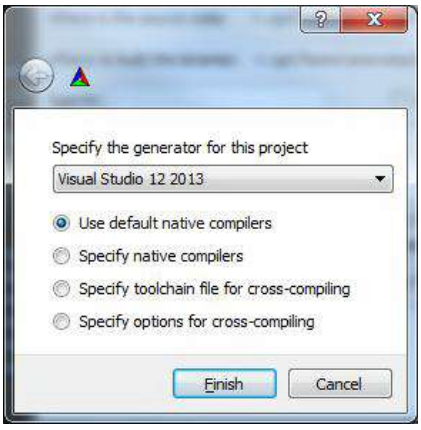
5. Setting up the Visual Studio 2013 example project

Setting up the Visual Studio 2013 example project with cmake

To build the example project **CameraApi** start cmake-gui and select the source folder “CameraApi” and choose a build folder (see below)



Hit the “Generate” button to generate the example project and choose Visual Studio 2013 generator.



After finishing the generation process you will find the file “CameraApi.sln” in the build project folder.

Open the CameraApi.sln file in Visual Studio and build the CameraApiTest. The example project can be extended with own projects using the camera interface.

Setting up projects without cmake

To build the example or any own project without cmake, all header folders and binary folder of the third party libraries (open_cv, gtest, tilib) must be includes in the project search paths.

6. Functional overview of the interface

Image data

- Thermal image format: float values represent temperature in °C
- Color mapping function with testo palettes
- Visual image format RGB 8-Bit

Thermal image parameter with read and write function

- Emission (float)
- Reflected temperature (RTC) (float)
- Measurement range (uint32_t)
- Humidity (float)
- Atmosphere temperature (float)
- Atmosphere correction enable disable (bool)

Read-only parameter

- Serial number
- Device type

For usage examples see the test functions that are included with the camera api.

7. Interface definition

```
class CamConnector
{
public:
  /** @function getListOfCameras
  ********************************************************************************************************
  get a list with serial and a list with camera type string of all found devices
  @param [out] vecSerials: list with serials
  @param [out] vecDeviceType: list with device types

  @retrun true if any camera was found
  ********************************************************************************************************/
  static bool getListOfCameras(std::vector<uint32_t>& vecSerials, std::vector<std::string>& vecDeviceType);

  /** @function open
  ********************************************************************************************************
  open a camera with a given serial
  @param [in] serial
  @retrun camera object
  ********************************************************************************************************/
  static IrCamera open(uint32_t u32Serial);
};

class IrCamera
{
```

```
  public:
    /**
    ****************************************************************************
    empty constructor only useable to create a dummy (not connected object)
    will not throw any exception
    ****************************************************************************/
    IrCamera();

    virtual ~IrCamera();

    uint32_t getSerial();
    std::string getDeviceType();

    /**
    ****************************************************************************
    captures one radiometric calculated ir frame from the camera
    framerate about 1fps

    @return float mat with temperature values for each pixel
    ****************************************************************************/
    cv::Mat captureIr();

    /**
    ****************************************************************************
    captures one visual frame from the camera

    @return 8-Bit RGB mat
    ****************************************************************************/
    cv::Mat_<cv::Vec<unsigned char, 3>> captureVis();

    /**
    ****************************************************************************
    Parameter
    ****************************************************************************/
    uint32_t getMeasurementRange();
    void setMeasurementRange(uint32_t u32MeasRange);
    uint32_t getNumberOfMeasurementRanges();

    float getEmissivity();
    void setEmissivity(float fValue);

    float getReflectedTemperature();
    void setReflectedTemperature(float fValue);

    bool getAtmosphereCurrectionState();
    void setAtmosphereCurrectionState(bool bEnable);

    float getHumidiy();
    void setHumidiy(float fValue);

    float getDistance();
    void setDistance(float fValue);
};
```

8. Example snippet

For more detailed information and running example please look into the test that is included with the camera API.

Open the camera and get a thermal image

```
// get a list of connected cameras
std::vector<uint32_t> vecSerials;
std::vector<std::string> vecDeviceType;
CamConnector::getListOfCameras(vecSerials, vecDeviceType);
// check if any camera was found
if(vecSerials.size() > 0)
{
  // open the 1. found camera
  IrCamera camera = CamConnector::open(vecSerials[0]);
  // get ir image
  cv::Mat matImageIr = m_camera.captureIr();
}
```