ibaVision

Automated Image Processing

Manual

Issue 3.0

Measurement Systems for Industry and Energy

www.iba-ag.com

Manufacturer

iba AG

Koenigswarterstr. 44

90762 Fuerth

Germany

Contacts

Main office	+49 911 97282-0
Fax	+49 911 97282-33
Support	+49 911 97282-14
Engineering	+49 911 97282-13
E-mail	iba@iba-ag.com
Web	www.iba-ag.com

The content of this publication has been checked for compliance with the described hardware and software. Nevertheless, discrepancies cannot be ruled out, and we do not provide guarantee for complete conformity. However, the information furnished in this publication is updated regularly. Required corrections are contained in the following regulations or can be downloaded on the Internet.

The current version is available for download on our web site www.iba-ag.com.

Version	Date	Revision - Chapter / Page	Author	Version SW
3.0	08-2022	Plug-in architecture	DG, st	3.0

Contents

1 About this documentation 5

1.1 Target group and previous knowledge 5

1.2 Notations 5

1.3 Used symbols 6

2 Introduction 7

2.1 Integration of ibaVision 7

2.2 The plug-in concept 8

2.3 Working principle 8

2.4 License information 14

3 System requirements 15

3.1 Software 15

3.2 Hardware 15

3.3 Notes about switching from ibaVision v2 to ibaVision v3 16

4 Installation 17

4.1 User account for Auto-logon 20

5 Configuration 22

5.1 Basic steps 22

5.2 HDevelop application 22

5.3 Python script 23

5.4 .NET plug-in 23

5.5 ibaVision 23

5.5.1 Start ibaVision 23

5.5.2 ibaVision Status Window 23

5.5.2.1 General tab 24

5.5.2.2 Event log tab 26

5.5.3 Configuration dialog 26

5.5.3.1 General information 27

5.5.3.2 Main window 28

5.5.3.3 Plugin settings 29

5.5.3.4 Offline video processing 30

5.5.3.5 ibaPDA inputs 34

5.5.3.6 Video inputs 37
5.5.3.7 Initialization, Main, Cleanup procedures 40
5.5.3.8 ibaPDA outputs 47
5.5.3.9 Video outputs 50
5.6 ibaCapture 54
5.7 ibaPDA 57
5.7.1 ibaVision input modules 57
5.7.2 ibaVision output module 59
6 Developing ibaVision programs in HDevelop 60
6.1 Accessing video data from ibaCapture in HDevelop 60
6.2 Writing log messages from the HDevelop program 63
6.3 Example HDevelop program 63
7 Notes on Python installation 70
7.1 Installing Python packages without direct internet access 70
7.2 Known issues 71
8 Creating a Python script 72
8.1 Introduction 72
8.2 Python script tutorial 72
8.2.1 Script header 73
8.2.2 Script procedure 75
9 Support and contact 76

1 About this documentation

This documentation describes the function and application of the software *ibaVision*.

1.1 Target group and previous knowledge

This manual is aimed at qualified professionals who are familiar with handling electrical and electronic modules as well as communication and measurement technology. A person is regarded as professional if he/she is capable of assessing safety and recognizing possible consequences and risks on the basis of his/her specialist training, knowledge and experience and knowledge of the standard regulations.

1.2 Notations

In this manual, the following notations are used:

Action	Notation
Menu command	Menu *Logic diagram*
Calling the menu command	*Step 1 – Step 2 – Step 3 – Step x* Example: Select the menu *Logic diagram – Add – New function block*.
Keys	<Key name> Example: <Alt>; <F1>
Press the keys simultaneously	<Key name> + <Key name> Example: <Alt> + <Ctrl>
Buttons	<Key name> Example: <OK>; <Cancel>
Filenames, paths	`Filename`, `Path` Example: `Test.docx`

1.3 Used symbols

If safety instructions or other notes are used in this manual, they mean:

Danger!

The non-observance of this safety information may result in an imminent risk of death or severe injury:

- Observe the specified measures.

Warning!

The non-observance of this safety information may result in a potential risk of death or severe injury!

- Observe the specified measures.

Caution!

The non-observance of this safety information may result in a potential risk of injury or material damage!

- Observe the specified measures

Note

A note specifies special requirements or actions to be observed.

Tip

Tip or example as a helpful note or insider tip to make the work a little bit easier.

Other documentation

Reference to additional documentation or further reading.

2 Introduction

ibaVision serves as a link between the iba system and project-specific image processing solutions. With version 3, *ibaVision* introduces a new plug-in architecture that allows building solutions for an even wider array of projects. Independent of the used plug-in, the main feature of *ibaVision* consists of processing image data automatically. Results are presented either as data (numeric/text) or as new images. All these results can be processed, visualized and analyzed with the familiar iba tools in the usual way. In general, automatic inspection images can provide information that is hidden to conventional sensors. Thus, *ibaVision* provides ways to enhance previously unavailable process information thus offering new possibilities for analysis and automated monitoring.

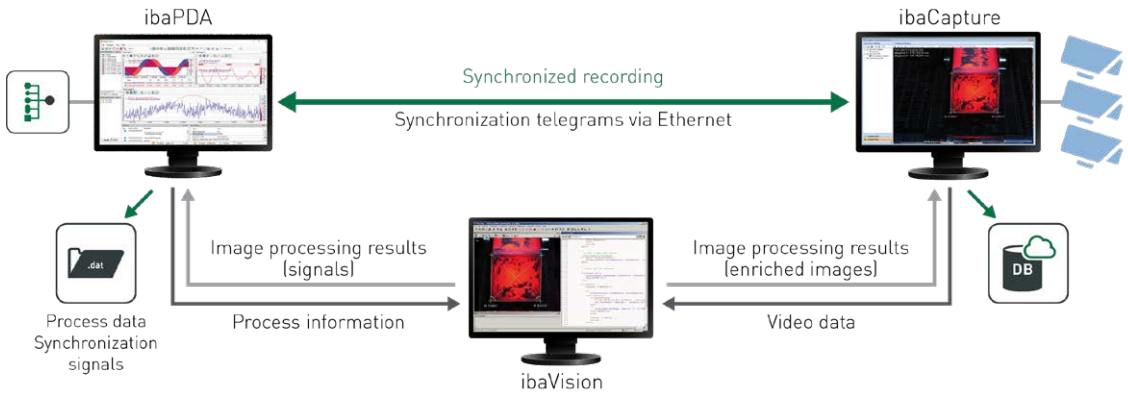
Intelligent image processing

ibaVision programs convert visual process information into numerical or logical values. In this way, for example, the spacing, geometry or position of products can be determined and used for quality testing or identification of parts during the ongoing process. In addition, identification features, such as bar codes, numbers or other machine-readable symbols can be captured. Numeric or textual values are recorded online in *ibaPDA* and can, like other process signals, be visualized and displayed as trends. The user can display process data side-by-side with static images or live video streams and can recognize emerging trends, process deviations or failures. In addition, optical warning messages can be integrated into HMI system displays. In case tolerances for a quality feature are exceeded, this can immediately be displayed, for example, using a signal light.

2.1 Integration of ibaVision

All data and image interfaces that are defined in *ibaVision* programs are automatically listed and can be flexibly linked to signals in the iba system. Both signal and image information can be used bidirectionally. The resulting values are recorded in *ibaPDA*. If necessary, process signals which are already available in *ibaPDA*, can be sent to *ibaVision* in order to adapt the running image analysis.

Cameras that are configured in *ibaCapture* can be used as image sources. Finally, resulting images from the image analysis which may contain markers or other additional visual information, can be encoded into a video stream. With *ibaCapture* such streams can again be recorded using a virtual camera.



2.2 The plug-in concept

The introduction of *ibaVision* v3.0 features a new plug-in concept to choose the right tool for every job. Initially, *ibaVision* will be delivered with two plug-ins that work out-of-the-box. The first plug-in provides the HALCON integration, which has already been available in all previous versions of *ibaVision*. The second plug-in executes Python scripts. This allows using the extensive selection of libraries that are available for the Python programming language for solving image processing tasks. Beyond the included plug-ins, it is also possible to program plug-ins that execute custom applications within *ibaVision*. This offers unlimited possibilities for image processing applications in the iba-system.

HALCON

HALCON by MVTec is a widely-used product that specialists all over the world use to create image processing applications. Commercial programming libraries such as HALCON offer the advantage of constant development of functions in new releases. In addition, technical support and training can be obtained. In order to create HALCON applications, a license for using HDevelop is required. To run the *ibaVision* program, a HALCON runtime license is required (included with *ibaVision* or purchased separately depending on the order option). The HALCON library provides a large set of functions and is a reliable tool for a wide range of image processing requirements.

Python

Due to the extensive number of available libraries, Python can also be used to create image processing applications. Python itself and most of the available libraries do not require purchasing licenses. This allows using Python immediately with every purchased *ibaVision* license. One available library for Python is OpenCV, which is an open-source image processing toolkit. Enthusiasts around the world use the available functionalities to build solutions. The OpenCV website is the starting point for documentation and help from other users.

2.3 Working principle

General remarks

ibaVision runs as a Windows process and thus requires a logged on user account. In order to grant automatic start of the system, create a user account which automatically logs on at system start. The installation of *ibaVision* configures Windows to start *ibaVision* automatically after logging in.

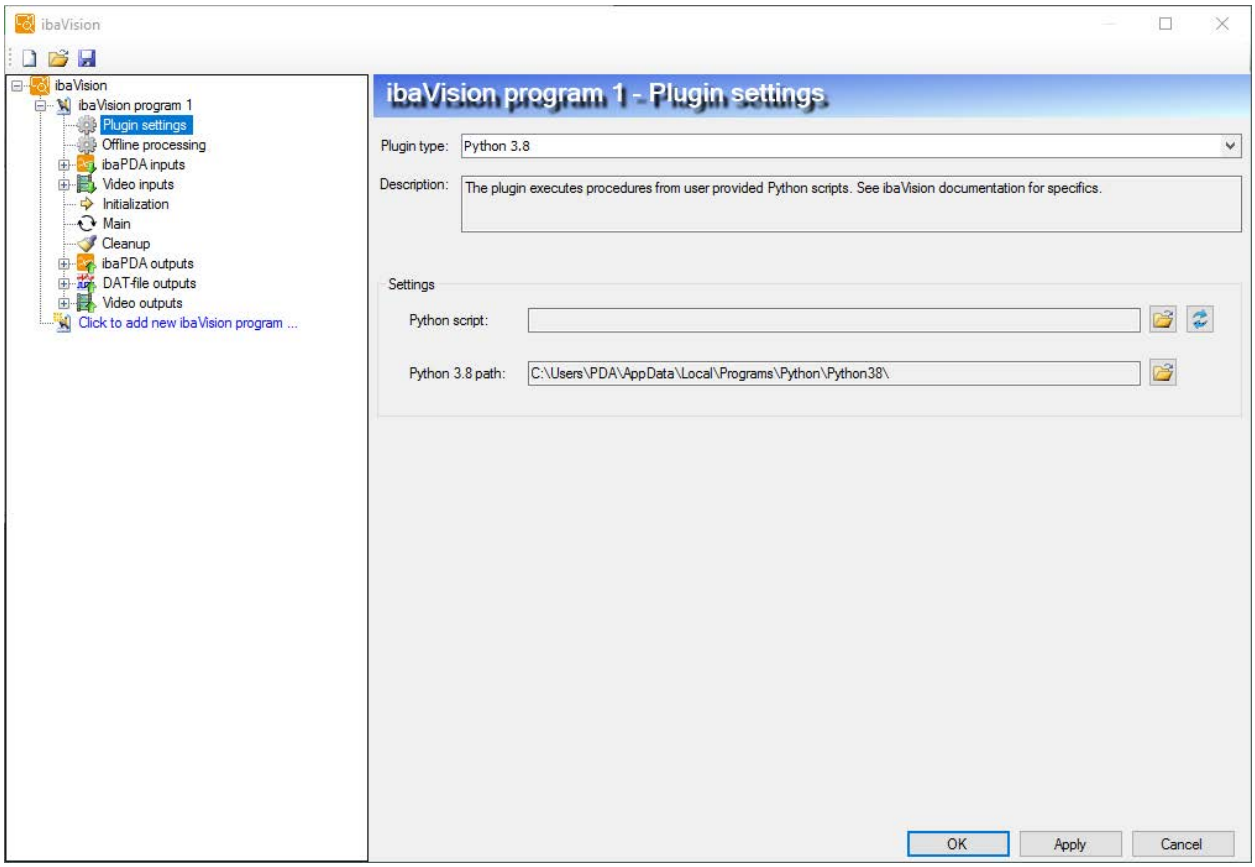
Within the main *ibaVision* application, multiple *ibaVision* programs can be run (2 *ibaVision* programs with the basic license), currently limited to 16 *ibaVision* programs per application.

Plugin selection

Due to the plug-in concept, you have to select the appropriate plug-in for your machine vision application.

- MVTec Halcon
  The processing of the image data is carried out with Halcon library functions. The application for every specific image processing task must be created by the user or a 3rd party with the development environment HDevelop. A description of how to develop an image processing application is not in the scope of this manual. Please refer to the Halcon SDK documentation.
- Python
  Scripts can be written in the Python programming language and utilize the full spectrum of exisiting Python libraries. Users have to provide their own installation of Python 3.8 to use the Python plug-in. Please refer to the Python documentation and chapter ↗ *Notes on Python installation*, page 70.
- Custom plug-in
  ibaVision functionalities can be extended by user-created custom plug-ins in the .NET framework. Plug-ins can, for example, execute custom machine vision processing algorithms or act as a custom video source.

  A code example is available on GitHub:
  https://github.com/iba-ag/ibaVision-DotNet-Plugin-Example



When the plug-in type is selected, further settings for the plug-in are required, like the path to the script and the program folder, see also chapter ↗ *Plugin settings*, page 29.

Structure of an ibaVision program

An *ibaVision* program consists of three fundamental procedures:

Initialization
The Initialization procedure should be used to open handles, for example to external hardware, internal software models, etc. It is executed once when the processing is started.

Main
The Main procedure is the procedure that will be executed in a continuous loop until the user decides to stop the *ibaVision* program. This is where the actual image processing should be implemented. The Main procedure must not contain infinite loops.

Cleanup
The Cleanup procedure can be used to close handles opened in the Initialization procedure. It will be executed after the *Main procedure* has been executed the last time. Then the *ibaVision* program is terminated.

Each time an *ibaVision* program executes one of these three procedures, it first sets all iconic and control inputs. The sources have to be specified. Next, the actual procedure is executed and finally the resulting iconic and control outputs are retrieved and passed on to the according output modules.

Example structure of an *ibaVision* program (simplified):

```
Run(Initialization)

while(Program execution not stopped)
{
 Run(Main)
}

Run(Cleanup)
```

For configuration details, refer to chapter ↗ *Initialization, Main, Cleanup procedures*, page 40

Sources and types of iconic and control inputs

Input parameters need to be connected to value sources. The following value types are possible:

- Control input
  - A constant value (integer, real or string)
  - A signal value coming from *ibaPDA* (analog, digital or string)
  - A control output from a procedure that has been executed before.

    For example:

    The Main procedure can use a control input that is the control output of the Initialization procedure. The Main procedure can also feedback control outputs: the value of a control output of the Main procedure can be used as a control input for the next iteration of the Main procedure. For this case, the user has to define an initial value for the control input.
  - Image timestamp: The timestamp information of the image from an *ibaCapture* video input that is currently processed

    In offline processing mode, these control inputs are also available:

- DAT file signal value
- DAT file info field value
- path to processed DAT file

- Iconic input
  - A constant image file
  - A frame coming from an *ibaCapture* camera
  - An iconic output from a procedure that has been executed before.

    For example:

    The Main procedure can use an iconic input that is the iconic output of the Initialization procedure. The Main procedure can also feedback iconic outputs: the value of an iconic output of the Main procedure can be used as an iconic input for the next iteration of the Main procedure. For this case, the user has to define an initial value for the iconic input.

For configuration details, refer to chapters ↗ *ibaPDA inputs*, page 34 and ↗ *Video inputs*, page 37.

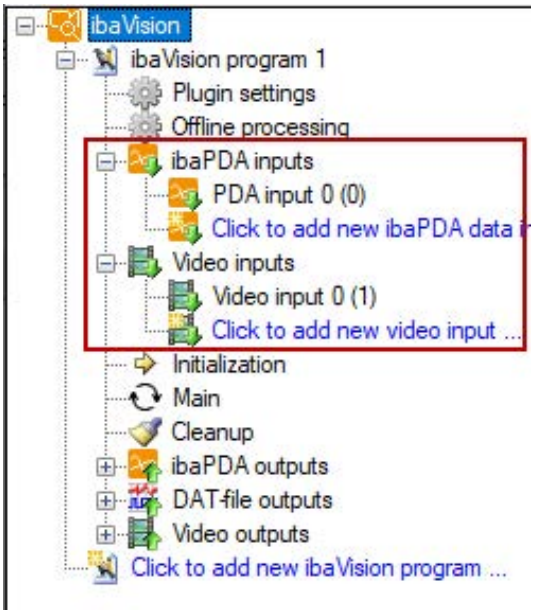
Iconic and control outputs

The following output parameters are parsed and available to use.

- Control output parameters can be used to:
  - Send values as visual signals to *ibaPDA* (analog, digital or string)
  - Set the value of a control input parameter
- Iconic output parameters can be used to:
  - Send a frame to a virtual camera in *ibaCapture*
  - Set the value of an iconic input parameter

For configuration details, refer to chapter ↗ *ibaPDA outputs*, page 47 and ↗ *Video outputs*, page 50.

Receiving data from ibaPDA and/or ibaCapture



Signals can be received from *ibaPDA*, e.g. in order to be used as control input variable and videos can be received from *ibaCapture* as image source for image processing.

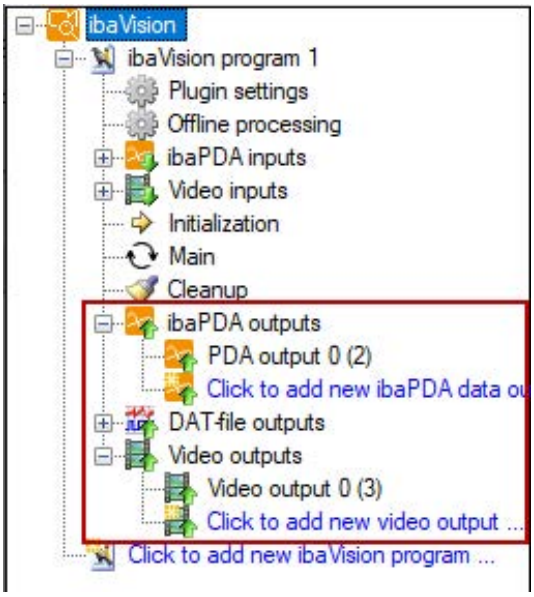
Signals from *ibaPDA* can be received via an *ibaPDA* input module. The necessary signals have to be defined in this module.

The videos to be processed can be received from *ibaCapture* via a Video input module. The desired camera connected to the *ibaCapture* Server can be selected and configured.

If the *ibaVision* program is designed to grab images through the plug-in-specific image acquisition functions, a video input module is not necessary.

For configuration details, refer to chapters ↗ *ibaPDA inputs*, page 34 and ↗ *Video inputs*, page 37.

Sending data to ibaPDA and/or ibaCapture



It is also possible to send signals to *ibaPDA*, such as measured values or other variables retrieved from the*ibaVision* program. These values can be acquired and recorded in *ibaPDA* as visual signals.

Signals can be sent to *ibaPDA* via an *ibaPDA* output module, where the desired variables can be selected. Possible signal types are analog and digital signals.

Processed images from iconic output parameters, e. g. which contain additional visual information, can be sent to *ibaCapture* and displayed and recorded as virtual camera. This video stream is sent via a Video output module, where the desired iconic output parameter can be selected.

For configuration details, refer to chapters ↗ *ibaPDA outputs*, page 47 and ↗ *Video outputs*, page 50.

Configuration in ibaPDA

To receive analog and digital signals in *ibaPDA* from an *ibaVision* program, an *ibaVision* input module has to be created in *ibaPDA*.

Within these modules, an *ibaPDA* output module configured in *ibaVision* has to be selected that will transmit the signals. The signals configured in the *ibaPDA* output module in the *ibaVision* program can automatically be mapped to the signals in the *ibaVision* input module in *ibaPDA*.

To send analog and digital signals from *ibaPDA* to an *ibaVision* program, *ibaVision* output modules have to be created in *ibaPDA*. Within this *ibaVision* output module, the *ibaPDA* input module configured in *ibaVision* has to be selected in order to receive the signals. The signals configured in the *ibaPDA* input module in the *ibaVision* program are mapped automatically to the signals in the *ibaVision* output module in *ibaPDA*. To define the signal value that is sent to *ibaVision*, an expression has to be defined in *ibaPDA* for each signal.

For configuration details, refer to chapter ↗ *ibaPDA*, page 57

Configuration in ibaCapture

Configuration in *ibaCapture* is necessary, when recording processed images from *ibaVision* as video stream. For this purpose, a camera with camera type Virtual camera (ibaVision) has to be created and configured.

For configuration details, refer to chapter ↗ *ibaCapture*, page 54.

2.4 License information

The basic *ibaVision* license allows running 2 parallel *ibaVision* programs. For applications with more *ibaVision* programs a license extension with the add-on license "ibaVision 2-Program-Add-On" is required. Each extension allows running 2 additional *ibaVision* programs.

The maximum number of allowed *ibaPDA* output signals is now limited to 128 signals. The number of *ibaPDA* output signals can be extended by purchasing the appropriate add-on license.

To run *ibaVision* with the HALCON plug-in, a HALCON runtime license is required. The HALCON runtime license can be purchased as a bundle with *ibaVision* or separately.

Order no.	Name	Description
38.100000	ibaVision	Application for image recognition tasks Executes image processing applications (up to 2 in parallel) with interfaces to images and data recorded/ provided by ibaCapture and ibaPDA.
38.100001	ibaVision with HALCON Runtime License	Application for image recognition tasks including HALCON runtime license Executes HALCON applications (up to 2 in parallel) with interfaces to images and data recorded/ provided by ibaCapture and ibaPDA. A USB dongle with “Halcon RTL bundle” license is part of this product.
38.100002	ibaVision 2-Program-Add-On	License extension for 2 additional program tasks.
38.100003	ibaVision-128-Signal-Add-On	License extension to send 128 more output signals to ibaPDA

3 System requirements

3.1 Software

- Operating system: Windows 10, Windows Server 2012/2012 R2/2016/2019

  Note: Depending on the used plug-in, there may be further restrictions of supported operating systems.
- *ibaVision* license (only WIBU licenses are supported for ibaVision, CodeMeter Runtime will be installed during the installation of *ibaVision*)
- HALCON plug-in: HALCON v20.11 steady (support is also available for v18.11, v13 and v12. Functionality may be limited)
- Python plug-in: Python v3.8 (for working with image data, *NumPy library* needs to be installed)

  Note: The Python plug-in is not supported on 32-bit or on systems that have 32-bit installations of HALCON and will not be selectable in the installer.
- To use *ibaCapture* input or output modules in *ibaVision*, an *ibaCapture* Server is required (for recording video encoded in H.265, *ibaCapture* v5.0.0 or newer is required)

  For *ibaCapture* input modules, *ibaCapture* Player needs to be installed. *ibaCapture* Player v5.1.6 or newer is required. It is recommended to use the latest version of *ibaCapture* Player. Older versions may not support all available features.
- *ibaPDA* v8.0.0 or later, if data communication is required

3.2 Hardware

- PC with Intel Core-CPU of the 2nd generation or later (e.g. Intel Core i7-2x00K CPU)
- 4 GB RAM

Requirements for video encoding in GPU

- NVIDIA encoding
  - NVIDIA GPU: A list of supported GPUs with their features is available in the NVENC Support Matrix: https://developer.nvidia.com/video-encode-decode-gpu-support-matrix#Encoder
    Please also see chapter ↗ *Video outputs*, page 50 for detailed information.
- Intel Quick Sync in HW-accelerated mode
  - Intel® HD Graphics 3000 or higher for H.264 encoding
  - Intel® HD Graphics 530 or higher for H.265/HEVC encoding

If no supported GPU is available, video encoding will be performed by the PC's CPU.

3.3 Notes about switching from ibaVision v2 to ibaVision v3

ibaVision v3 supports the same HALCON functionalities as v2 and all configurations and programs are forward (but not backward) compatible between v2 and v3. Despite this, it is still highly advised to create backup copies of all v2 configurations before trying to load them into v3 for the first time.

The communication to *ibaPDA* has been changed extensively and *ibaPDA* v8.0.0 or newer is required to work with *ibaVision* v3. When upgrading from *ibaVision* v2/*ibaPDA* v7, the I/O-configuration in *ibaPDA* has to be adapted manually after updating both *ibaPDA* and *ibaVision*.

4 Installation

Depending on the installed HALCON version, *ibaVision* can be installed as 32-bit or 64-bit version. The 32-bit *ibaVision* version will be installed with a 32-bit HALCON library and the 64-bit *ibaVision* version will be installed with a 64-bit HALCON library. During installation you have to select whether to install the 32-bit or the 64-bit version of *ibaVision*.

Starting with HALCON v20.11, 32-bit versions of HALCON for Windows are no longer available.

When multiple HALCON versions are installed on the system, the *ibaVision* installer will evaluate the %HALCONROOT% environment variable to determine the appropriate version.

Selecting the Python plug-in will only be possible with a 64-bit version of *ibaVision*.

Note

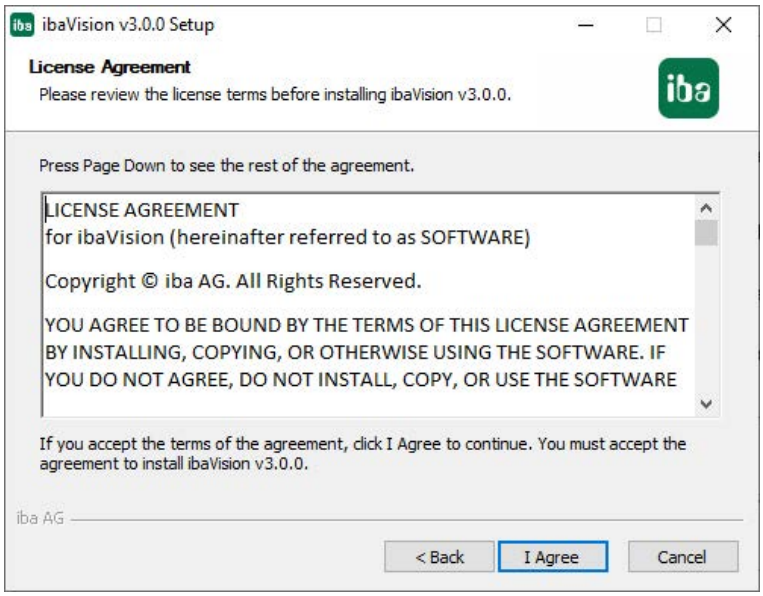
If *ibaVision*, *ibaPDA* or *ibaCapture* are installed on different PCs then separate dongles are required for the corresponding licenses. If *ibaVision* is installed on the same PC as *ibaPDA* or *ibaCapture* then the *ibaVision* license is stored on the common iba dongle.

ibaVision runs as a Windows process, not as service. When using *ibaVision* a user must be logged on at all times. iba recommends to create a user account which automatically logs on at system start, see chapter ↗ *User account for Auto-logon*, page 20

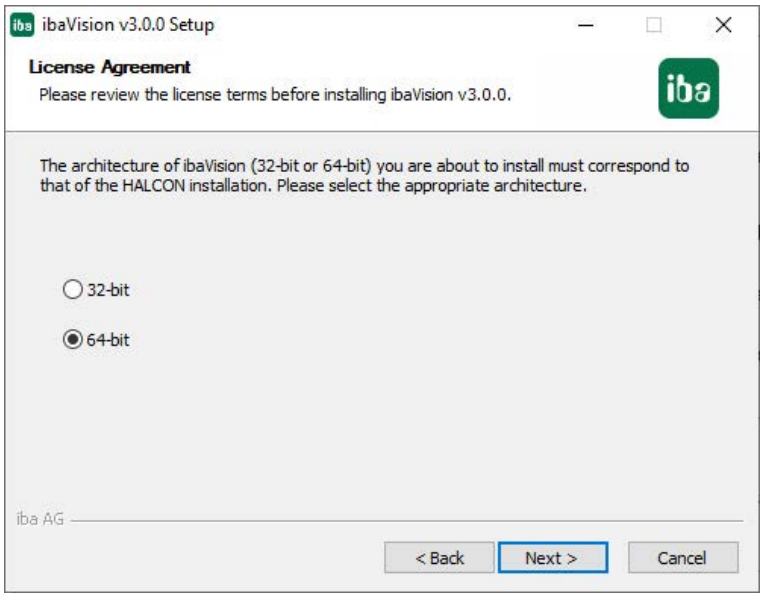
Proceeding

1. To install *ibaVision*, execute the file "ibaVision_setup_vx.y.z.exe" in the directory "\01_iba_Software\ibaVision" on the delivered data medium.
2. Follow the instructions of the installation wizard.

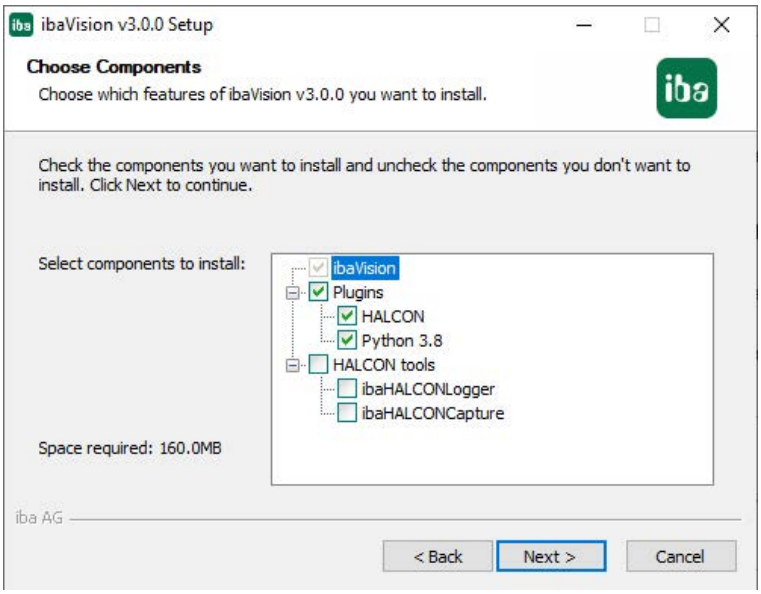
3. Agree to the terms of this license.



4. Select the appropriate architecture of *ibaVision* (must correspond to that of the HALCON installation)

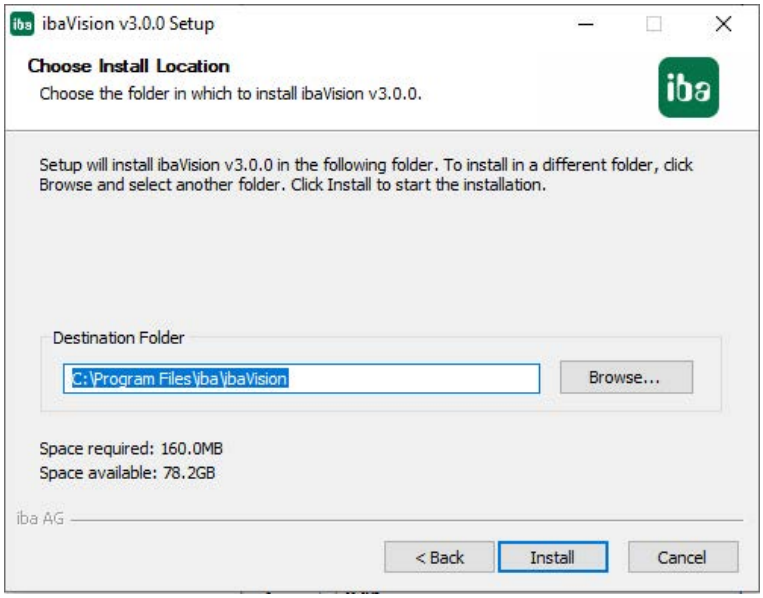


5. Select the components you want to install.



- ibaHALCONLogger installs an operator for the currently active HALCON version that allows writing log messages to the log of the *ibaVision* process (see chapter ↗ *Writing log messages from the HDevelop program*, page 63 for more information).
- ibaHALCONCapture installs image acquisition interfaces for HALCON that can be used when developing in HDevelop (see chapter ↗ *Accessing video data from ibaCapture in HDevelop*, page 60 for more information).

6. Select a directory for the program.



7. Click on <Install> and the installation process starts.
8. At the end, click on <Finish> to complete the installation.

4.1 User account for Auto-logon

In order to grant automatic restart of the system, create a user account, which automatically logs on at system start.

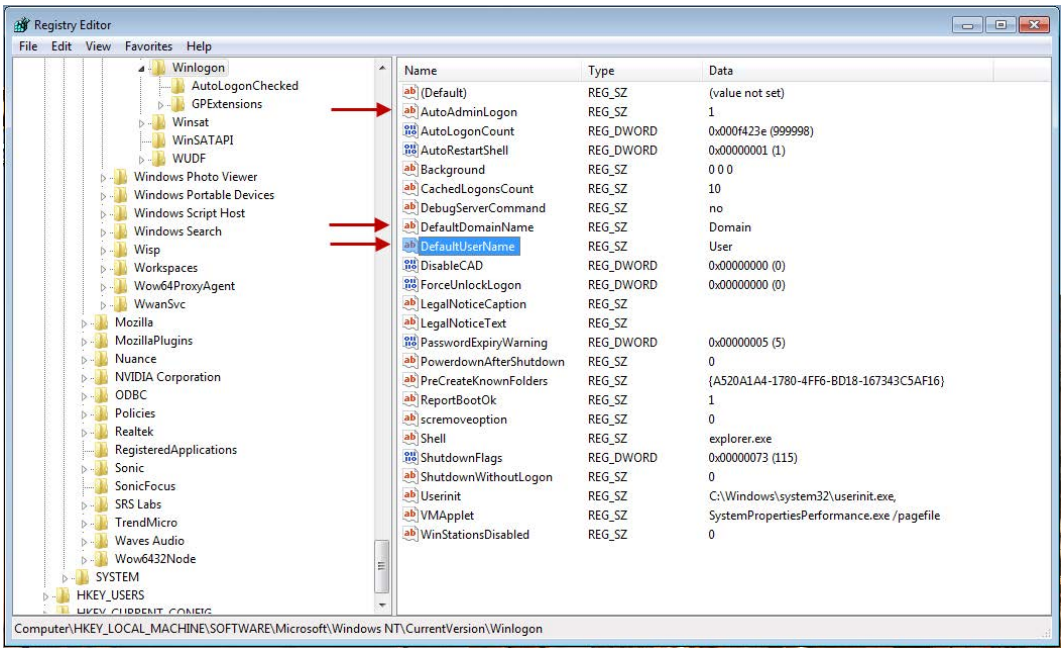
How to create a user account (local user) for Auto-logon under Windows 7 or higher

1. Click on the <Start> button, enter the command **netplwiz** in the search box and press <ENTER>.
   The dialog for the user account configuration opens.
2. In the "User" tab, uncheck the option "Users must enter user name and password".
3. Click on <Accept>.
4. When the dialog "Automatic login" opens, enter the user name for automatic login under Windows 7. Enter also the corresponding password.
5. Click <OK> and close the dialog "User accounts" by clicking <OK>.

How to create a user account for domain users

If the user is logged-in as domain user, the selection field „Users must enter user name and password" is not shown as default in the "User account" dialog and thus cannot be deactivated for individual users. If the appropriate setting has been done in the Windows Registry, the selection field is shown.

1. Click on the <Start> button, enter the command **regedit** in the search field and press <Enter>.
   The Registry Editor is opened.
2. Navigate in the Registry Editor to the path "HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\ Windows NT\CurrentVersion\ Winlogon"



3. Change the value for "AutoAdminLogon" to 1.
4. Click again on the <Start> button, enter the netplwiz command in the search field and press <Enter>.
   The dialog for configuring the user accounts is opened.
5. Select the user you want to log in automatically and deactivate the checkbox „Users must enter user name and password".
6. In the following dialog, you will be asked to enter the user password. Enter the password and confirm with <OK>.
7. Call once again the Registry Editor.
8. Change the setting for "DefaultDomainName" to the domain the user belongs to.
9. Enter under "DefaultUserName" the user that is to be logged in automatically.
10. Close the Registry Editor and reboot the computer. Now, the user should be logged in automatically.

5 Configuration

5.1 Basic steps

The basic steps give an overview of the general procedure. For configuration details, please refer to the following chapters.

1. First of all, the machine vision program has to be created. You may use different applications:
   a.) Create a HDevelop program (a HALCON SDK license is required for this step).
   b.) Create a Python script.
   c.) Create your own .NET plug-in.
2. Add a new *ibaVision* program in the configuration dialog of the *ibaVision* application and select the corresponding plug-in, which matches the used script. See chapter ↗ *Plugin settings*, page 29.
3. For every procedure type in *ibaVision* (Main, Initialization, Cleanup), select the corresponding procedure of the used script. See chapter ↗ *Initialization, Main, Cleanup procedures*, page 40.
4. Configure an *ibaPDA input* module in the *ibaVision* program if signals from *ibaPDA* should be used as control input. See chapter ↗ *ibaPDA inputs*, page 34.
5. Configure a *Video input* module in the *ibaVision* program to receive video streams from a camera connected to an *ibaCapture* Server. See chapter ↗ *Video inputs*, page 37.
6. Connect the control and iconic input parameters in every procedure to a value source. See chapter ↗ *Initialization, Main, Cleanup procedures*, page 40.
7. Configure an *ibaPDA output* module in the *ibaVision* program, if a control output parameter (e.g. a measured value from the image processing) should be recorded in *ibaPDA* as visual signal. On the *ibaPDA* side, configure an *ibaVision input* module to receive the signal. See chapter ↗ *ibaPDA outputs*, page 47.
8. Configure a *Video output* module in the *ibaVision* program, if a video stream that has been processed shall be recorded by *ibaCapture*. On the *ibaCapture* side, configure a virtual camera to receive the video stream. See chapter ↗ *Video outputs*, page 50.

5.2 HDevelop application

When a HDevelop program is used, it has to meet several requirements in order to work properly with *ibaVision*. *ibaVision* uses hdev-scripts without further processing, a compilation is not necessary.

- The necessary parameters must be specified in the HDevelop program, control variables for numbers, texts, etc. and iconic variables for images, regions, etc.
- Since an *ibaVision* program requires the procedures Initialization, Main and Cleanup, the HDevelop program must be structured accordingly. The procedures Initialization, Main and Cleanup need to be defined in HDevelop.

- The HALCON image processing application must be available as *.hdev file.
- HDevelop operators starting with **dev_** will be ignored when running the program in *ibaVision*.

The completed HDevelop program can be loaded in *ibaVision*. The usual start-up procedure "main" of the hdev application will not be executed by *ibaVision*. Also see chapter ↗ *Example HDevelop program*, page 63 for an example.

5.3 Python script

When a Python script is used, it has to follow certain definitions in order to allow *ibaVision* detecting its features and procedures.

- The necessary parameters must be specified in the Python script, control variables for numbers, texts, etc. and iconic variables for images, regions, etc.
- Functions need to be defined which can be selected to be executed as Initialization, Main and Cleanup procedures in *ibaVision* programs.
- The Python image processing application must be available as *.py file.

When the *ibaVision* program is executed, only code within the defined procedures for Initialization, Main and Cleanup will be executed. Also see chapter ↗ *Creating a Python script*, page 72.

5.4 .NET plug-in

Creating .NET plug-ins is not within the scope of this manual. An example plug-in and the required documentation is provided in a GitHub repository:
https://github.com/iba-ag/ibaVision-DotNet-Plugin-Example

5.5 ibaVision

5.5.1 Start ibaVision

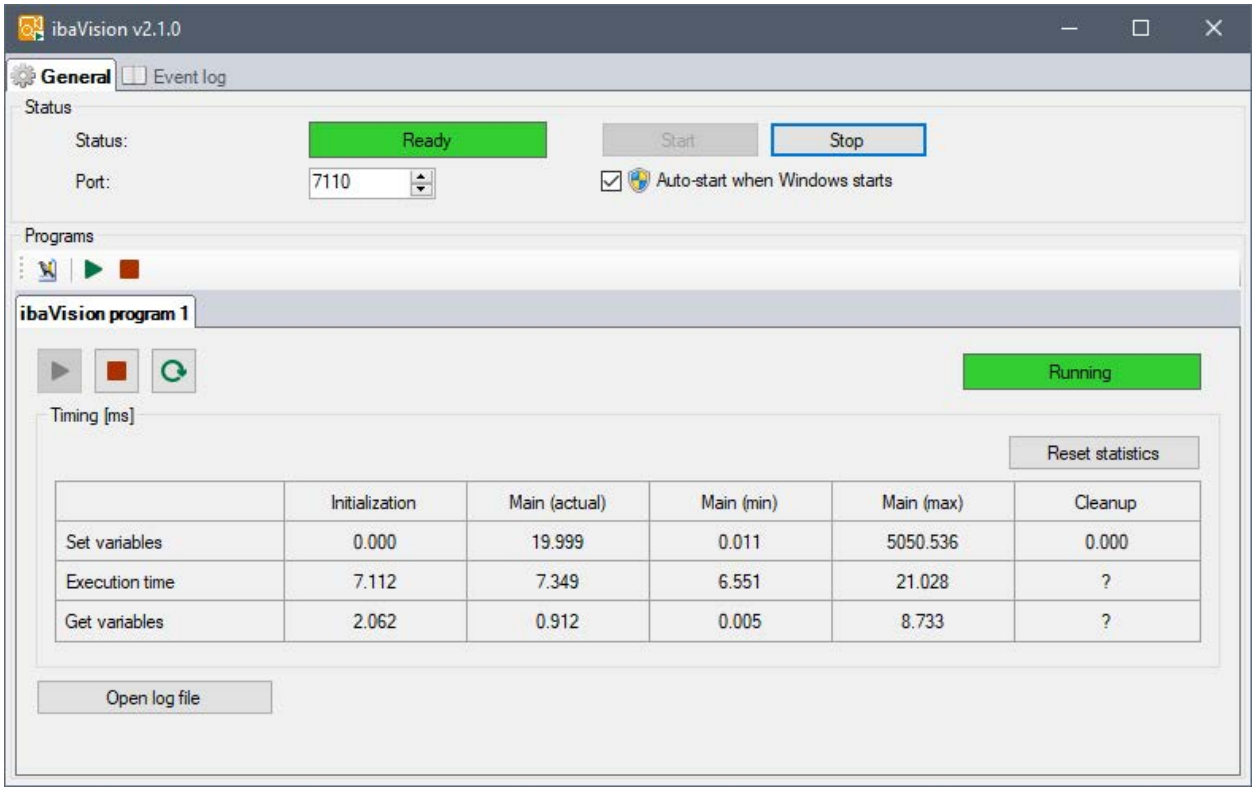
The ibaVision icon appears in the Windows tray after installation. To open the status window …

- Double-click the ibaVision icon or
- Right-click the ibaVision icon and select *Status* from the context menu.

5.5.2 ibaVision Status Window

The status window shows diagnostics and provides control elements for the execution of the ibaVision programs.

5.5.2.1 General tab



Status section

Status
General status of *ibaVision*

Button <Start>
General functions of *ibaVision* will be initialized, like opening interprocess communication channels, checking if the external programs and libraries are available, etc.

Button <Stop>
Stop the general functions.

Port
The default port number is 7110. If you want to manually change this value, stop the program with the <Stop> button. Enter the desired port number and restart the program with the <Start> button.

Note

If the port number is modified manually, the firewall settings may need to be adapted as well.

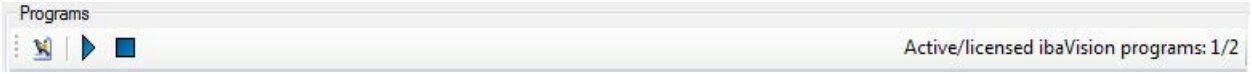
Auto-start when Windows starts
This option has to be activated, if *ibaVision* is supposed to start automatically with every system start.

Programs section
The programs section displays information on the currently configured *ibaVision* programs. The toolbar contains three buttons:


opens the configuration dialog where the ibaVision configuration can be edited


starts all configured ibaVision programs


stops all configured ibaVision programs



Operating elements and status information of the *ibaVision* programs are available in the section below. Each *ibaVision* program is enumerated in a corresponding tab. The name of the *ibaVision* is shown in the tab.


Starts the currently selected *ibaVision* program. The Initialization procedure is executed and the Main procedure is executed in a loop until the user stops the program.


Stops the currently selected *ibaVision* program. The Main procedure will be finished and the Cleanup procedure will be executed.


Restarts the currently selected *ibaVision* program.

Status — Current status of the *ibaVision* program: Stopped, Starting, Started, Stopping or Error.

<Open log file> — Opens this program's current log file in the default text editor. Each *ibaVision* program generates individual log files.

Timing — Each procedure execution consists of setting the values of iconic and control variables, running the procedure and getting the values of iconic and control variables. *ibaVision* constantly times each of these steps for the three procedures and displays the results here in microseconds. In addition to the actual values, the minimum and maximum values for the Main procedure execution are also displayed. To reset the minimum and maximum values, click the <Reset statistics> button.

The maximum size for *ibaVision* log files can be set in the Windows registry by changing the "MaxLogFileSize" entry under

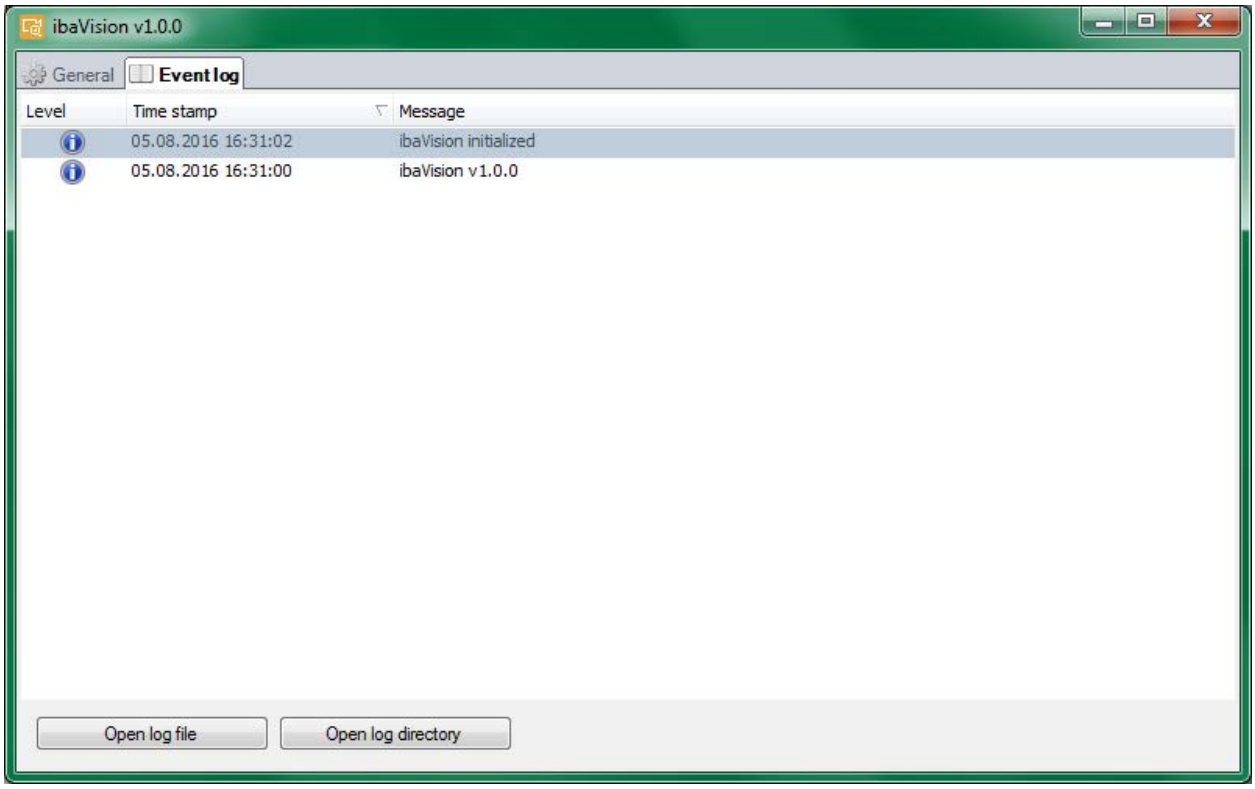
HKEY_LOCAL_MACHINE\SOFTWARE\iba\ibaVision

Name	Type	Data
(Default)	REG_SZ	(value not set)
MaxLogFileSize	REG_DWORD	0x00080000 (524288)
uninstallBusy	REG_DWORD	0x00000000 (0)

The value is in units of megabyte and the default size is 524288 MB.

5.5.2.2 Event log tab

The Event log tab displays the most important log messages generated by *ibaVision*.



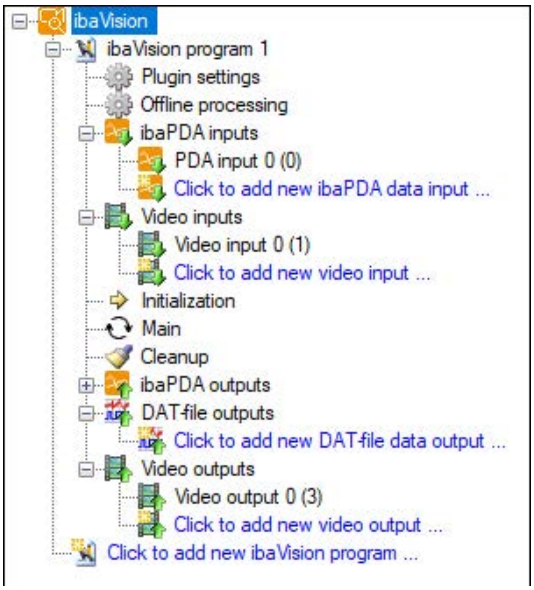
Clicking the button <Open log file> will open the current log file in the default text editor, the button <Open log directory> will open the directory containing the current and previous log files in Windows Explorer.

5.5.3 Configuration dialog

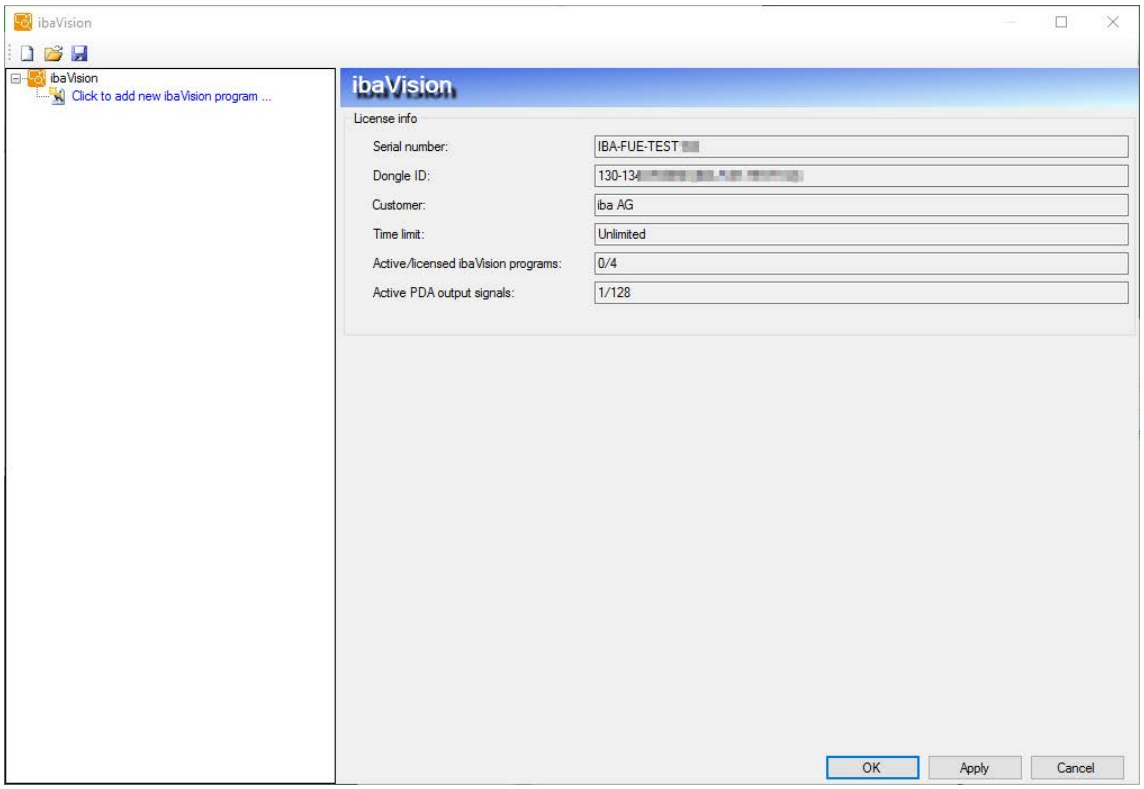
In the configuration tree on the left, all configured *ibaVision* programs are listed under the main node *ibaVision programs*.

To add a new program, click on the link *Click to add a new ibaVision program….* . A new *ibaVision* program node will be created with the following subnodes:

- ibaPDA inputs
- Video inputs
- Initialization
- Main
- Cleanup
- ibaPDA outputs
- Video outputs

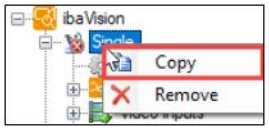


5.5.3.1 General information

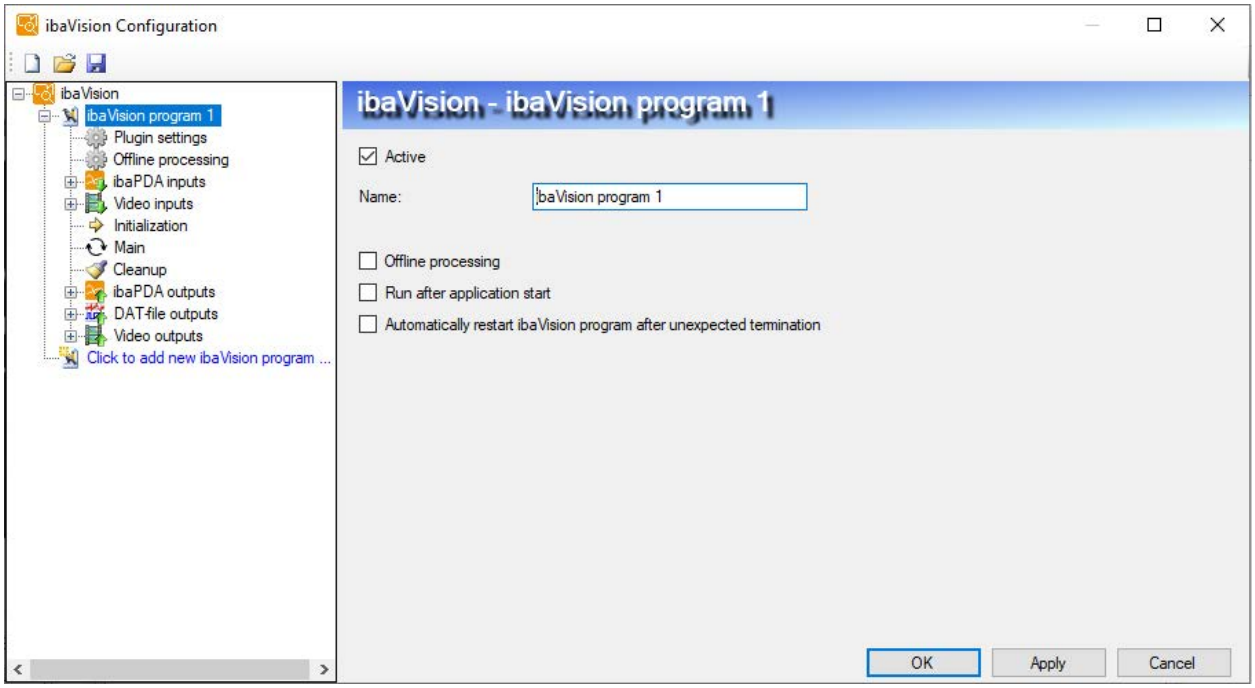


The main node *ibaVision* displays information regarding your license, like serial number of the license, license serial number, customer, time limit and active/licensed *ibaVision* programs.

A copy of the *ibaVision* program and the module configurations can be created via the context menu, opened by right-clicking on the node in the configuration tree.



5.5.3.2 Main window



Active
You can determine, whether the *ibaVision* program will be validated and started when applying the configuration. You also may temporarily deactivate a program, if the program is not needed without having to delete the configuration. This can be useful when only a limited number of active *ibaVision* programs are licensed.

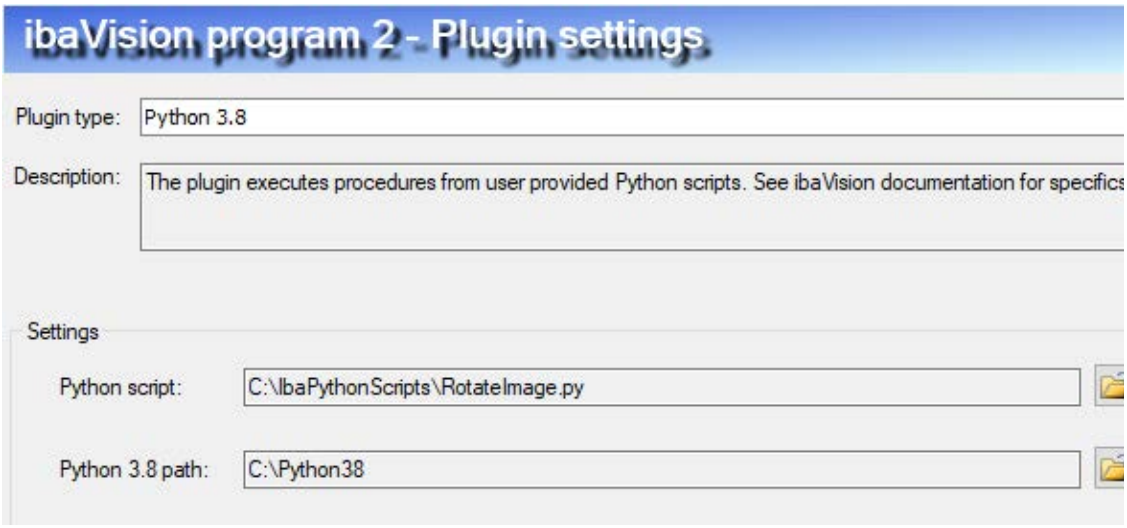
Name
Enter an appropriate name for the program.

Offline processing
Offline video processing can be enabled here. For a detailed description see also chapter
↗ *Offline video processing*, page 30

Run after application start
When this option is selected, the *ibaVision* program will automatically start when the main *ibaVision* application starts.

Automatically restart ibaVision program after unexpected termination
In case an error occurs during the execution of the *ibaVision* program, its execution will usually be stopped. If this option is enabled, *ibaVision* will attempt to restart the terminated *ibaVision* program automatically.

5.5.3.3 Plugin settings



Select the plug-in type for this *ibaVision* program in the dropdown menu and configure the appropriate settings for the plug-in.

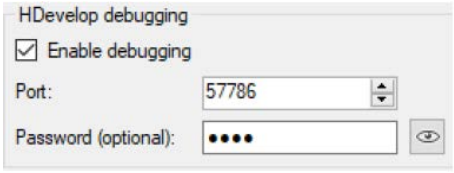
HALCON plug-in

HDevelop program
Enter the path to the HDevelop program that will be executed with this *ibaVision* program. Clicking the button will open a Windows dialog where you can select the file. When changes have been made to the HDevelop program, click the button to reload the HDevelop program. After reloading, the input/output settings may also need to be adapted.

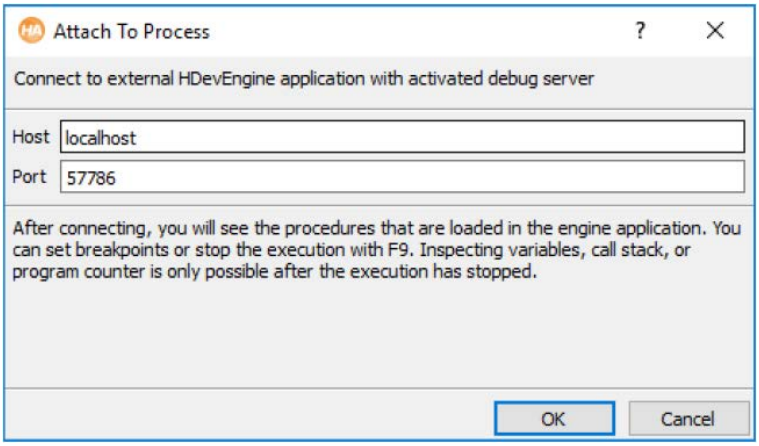
External procedure path
Certain Halcon programs rely on external procedures. Enter here the path to the required external procedures. If the procedure could not be found, Halcon will attempt to load external procedures from this path when clicking the reload button.

Use Halcon XL
Enable this option to run the *ibaVision* program with the Halcon XL variant of the HdevEngine.

HDevelop debugging
When the option *Enable debugging* is enabled, it is possible to debug HDevEngine applications remotely when using Halcon 13 or newer. Set a port and optionally set a password to restrict access to the debugging feature.



To attach to a running *ibaVision* program, open HDevelop and select 'Attach To Process...' in the *Execute* menu. A form will pop up where you enter the host on which *ibaVision* is running and the configured port. If a password has been set, this will be requested in a second step. After connecting to the *ibaVision* program, it is possible to step through your Halcon script and debug accordingly.



Click on <OK> or <Apply> to apply the new configuration. The configuration will be checked during the following validation. <OK> closes the validation dialog, provided that no warning or error messages have appeared during the validation process.

Python plug-in

Python script

Enter the path to the Python script with the procedures that you want to execute. Clicking the button will open a Windows dialog where you can select the *.py file. When you have a file loaded already and have made changes to it externally, click the button to reload it. After reloading, the input/output settings may need to be adapted.

Python 3.8 path

ibaVision does not provide its own Python interpreter or Python modules. To use the plugin, you need to have a working Python 3.8 setup installed on the system already. When creating a new program, *ibaVision* will attempt to determine the correct Python 3.8 home path from the Windows environment variable. If this process fails for some reason or there are multiple Python 3.8 installations on the system, the path can be changed here. Clicking the button will open a Windows dialog where you can select the correct Python 3.8 home directory. If this path is changed, a complete restart of *ibaVision* is required to reload all references.

5.5.3.4 Offline video processing

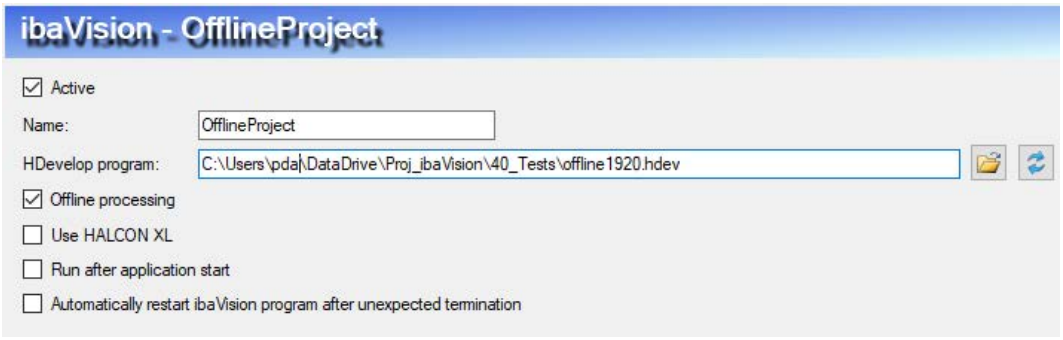
Sometimes it is not possible to process a video stream in real-time or only certain periods of a stream need to be processed. Offline video processing allows to postpone the processing and define the time periods to be processed. The resulting video output is sent to the *ibaCapture* Server using the timestamps of the source video. Thus the synchronization in *ibaAnalyzer* can be ensured.

The time periods are defined by evaluating *ibaPDA* DAT files that are written to a monitored directory. Offline processing will begin at the start of each DAT file and run through the whole length of each DAT file. For the time periods to process, video will be read from the *ibaCapture* storage of the selected camera.

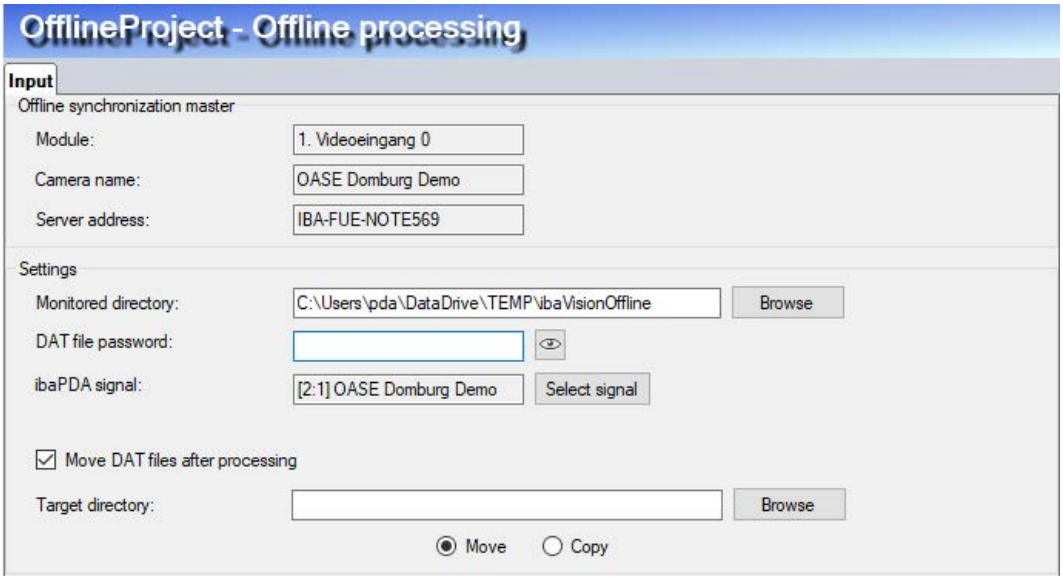
It is also possible to write back numeric, digital and text outputs to the DAT file currently being processed. For this purpose a DAT file output module must be configured. See chapter ↗ *DAT file output module*, page 33.

Set up

Offline video processing can be activated on the general node in the *ibaVision* program configuration. It is recommended to add and configure all other modules before opening the new node 'Offline processing'.



The *Offline processing* node provides information about the offline synchronization master and some additional settings.



Offline synchronization master

The offline synchronization master is automatically chosen from the configured video inputs. The master is the video input with the lowest frame rate when editing the configuration.

Note

To correctly synchronize the offline processed video in *ibaAnalyzer*, it is important that the video output of the *ibaVision* program is sent to the *ibaCapture* Server of the offline synchronization master.

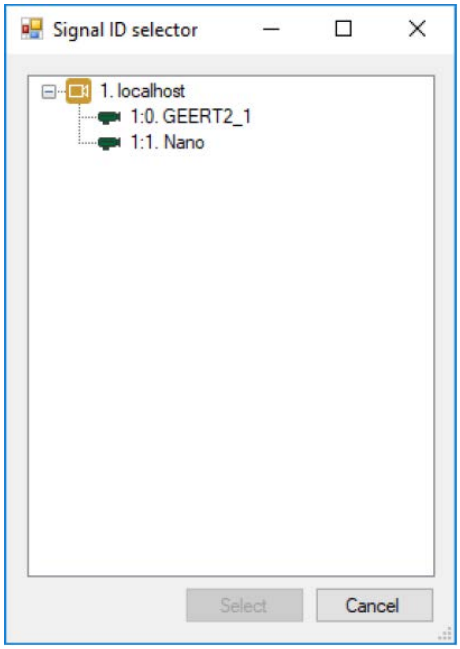
Settings

Monitored directory

Select a directory which is monitored for new DAT files. These DAT files will be parsed to acquire the time periods which should be processed.

ibaPDA signal

Select an *ibaPDA* signal to specify the correct time period. Open a sample DAT file and select the desired signal. It is recommended to choose the signal which matches with the offline synchronization master.



Move DAT files after processing

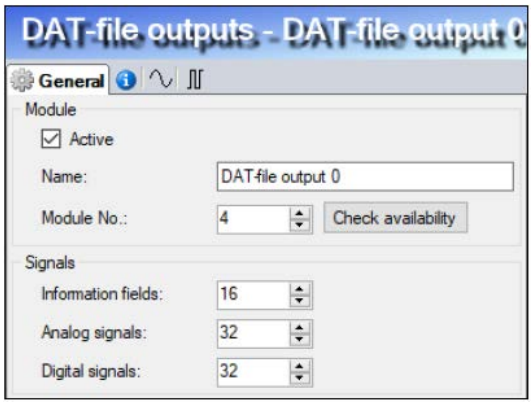
When you check this option, it is possible to move or copy DAT files after they have been processed. Select a target directory where the DAT files should be moved or copied to.

Offline input sources

When configuring the procedures of an offline program, there are two new sources for control inputs, namely the DAT file signal and information field input sources. This makes it possible to use data from the provided DAT files as input values. Currently, the DAT file signal input is only allowed in the Main procedure. See also chapter ↗ *Control input tab*, page 43

5.5.3.4.1 DAT file output module

To add a new *DAT file output* module, click on the link *Click to add new DAT file output …* under the *DAT file output* node.

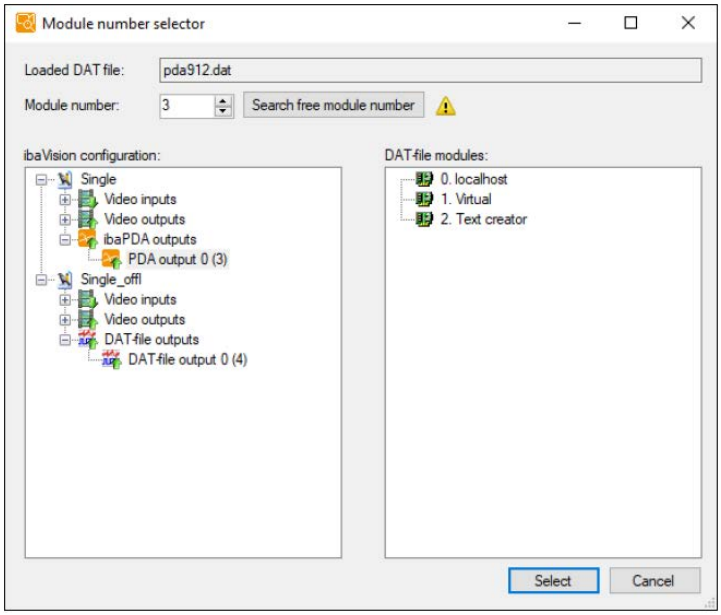


Enter a name and a module number. Make sure that the module number is unused. Otherwise existing data could be overwritten. You can check if a module number ia already used with the <Check availability> button, see description below.

In the *Signals* section, configure the number of information fields and analog or digital signals. To define text channels, select the type STRING for an analog signal.

<Check availability>
Clicking the button opens the dialog *Module number selection*. The *module number selector* checks the availability of the currently selected module number against both the *ibaVision* program configuration and the opened DAT file.

If the number is already in use a warning sign is shown next to the button <Search free module number> and a warning message will appear when clicking <Select>. A click on the button <Search free module number> searches for the lowest unused module number.



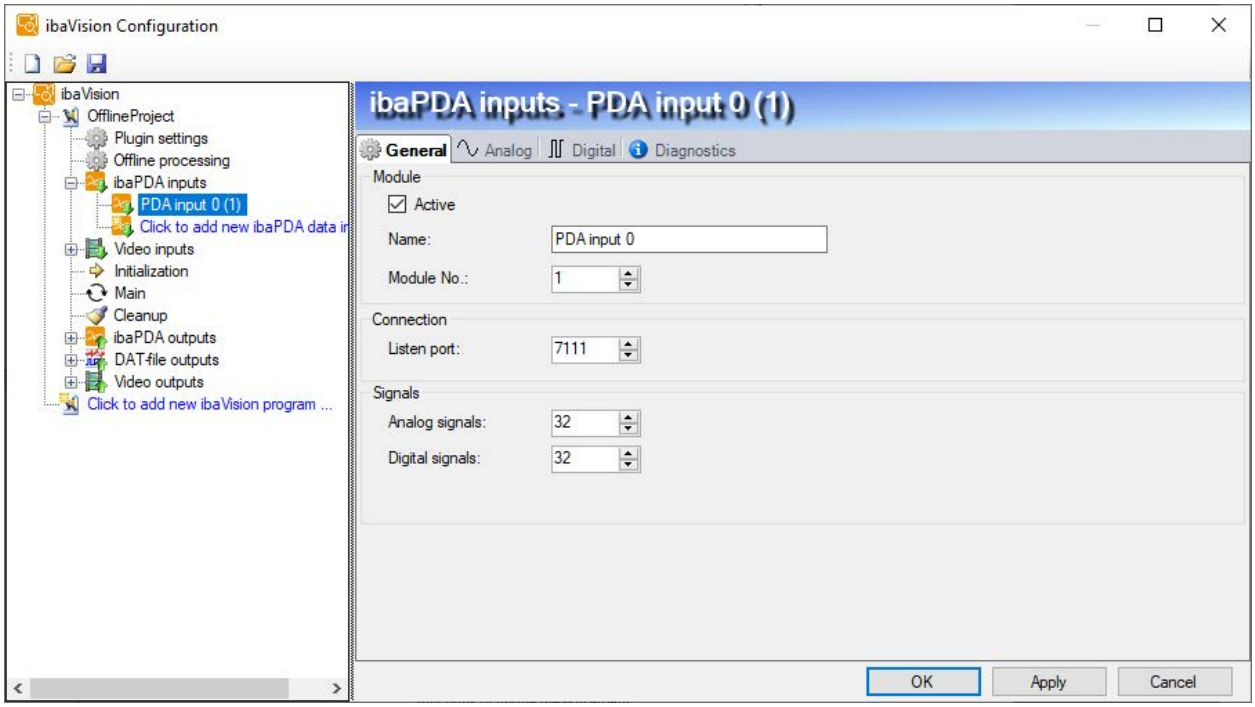
Note

When making use of DAT file output modules, it is advisable to back up the DAT files before processing. Accidently overwriting of previously existing data cannot be undone.

5.5.3.5 ibaPDA inputs

To add a new module *ibaPDA inputs*, click on the link *Click to add new ibaPDA input module …*

General tab



Module

Active
Here you can select if the module is active or not. Only active modules will be considered during validation of the configuration.

Name
Enter an appropriate name here. The name will be used for diagnostic purposes and log messages. The name is also displayed when setting up the module in the *ibaPDA* I/O-configuration.

Module No.
Logic module number for unique module reference.

Connection

Listen port
Network port for connections by *ibaPDA*. If another firewall than the Windows firewall is used, make sure that connections will be accepted on the defined port.

Signals

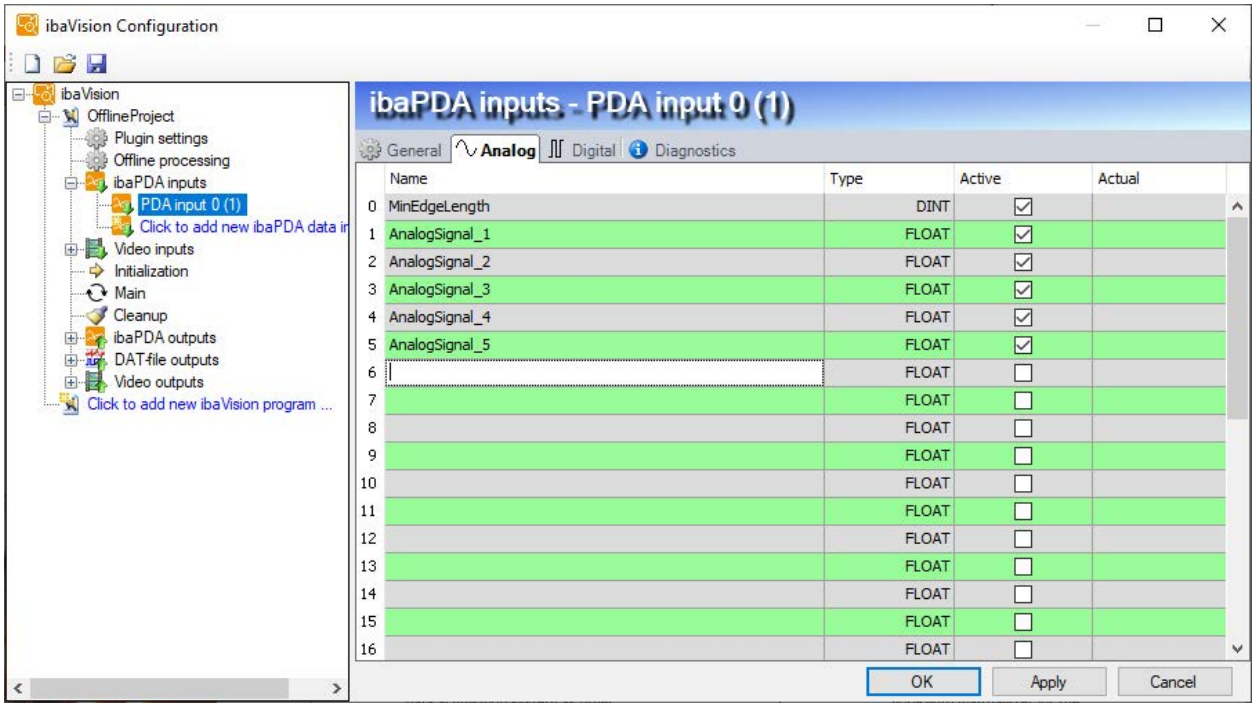
Analog signals

Defines the number of analog signals that will be received via this module. The same amount of lines will be displayed in the *Analog* tab. When the value is „0“, the *Analog* tab will not be displayed.

Digital signals

Defines the number of digital signals that will be received via this module. The same amount of lines will be displayed in the *Digital* tab. When the value is „0“, the *Digital* tab will not be displayed.

Analog tab



Enter the analog signals you want to use from *ibaPDA*. The individual columns in the signal list have the following meaning:

Name

Here you can enter a signal name.

Tip

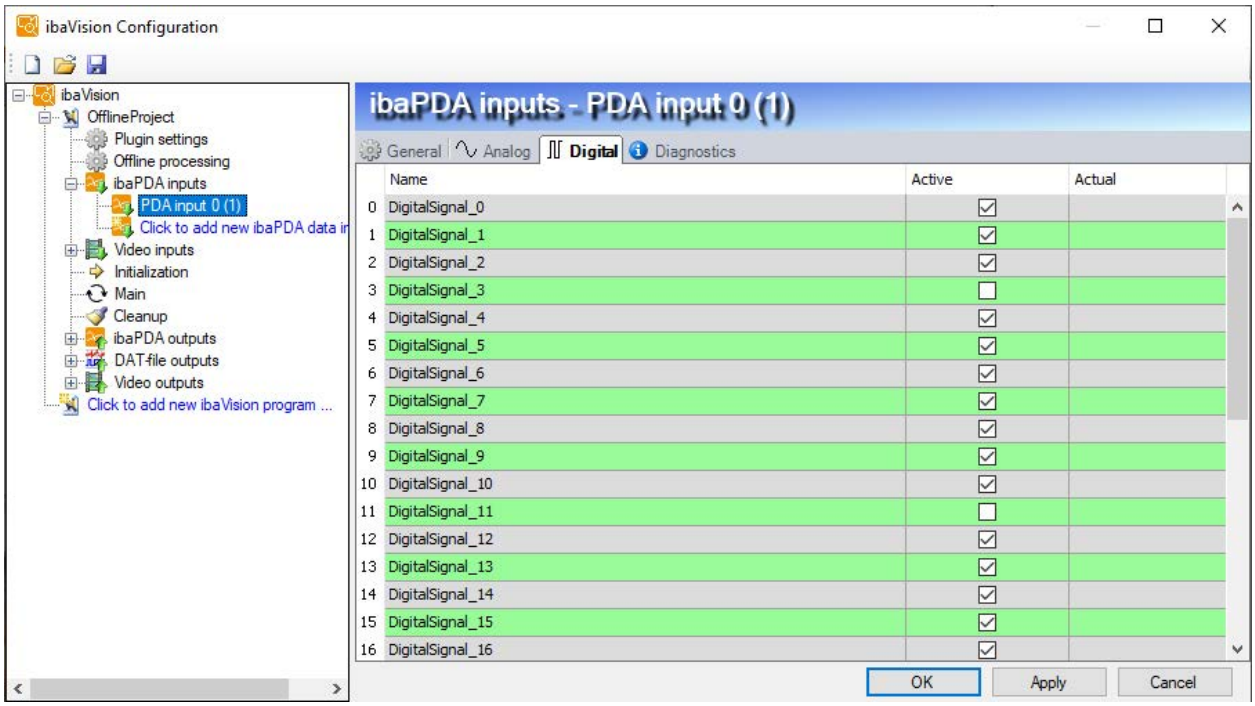
If you enter a signal name ending with a number and click on the column header (as long as the cursor is still in the name field) then all empty fields below will be filled with that name with an increasing number like an index. You may use this function in any row of the table. Fields that already have names will not be overwritten.

Type
Data type of the signals, available data types: BYTE, INT, DINT, FLOAT, STRING.

Active
If this option is selected, the signal will be used in the current configuration. Clicking the column header will apply the active state of the selected signal to subsequent signals.

Actual
Display of the currently acquired values.

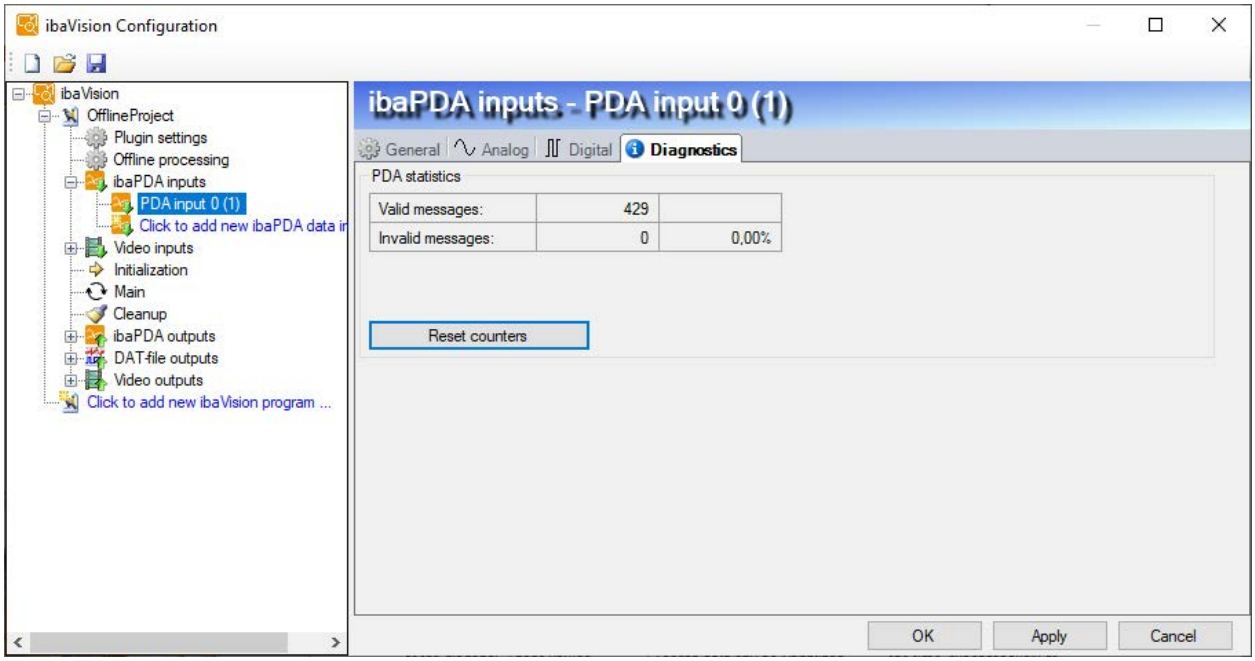
Digital tab



Enter the digital signals you want to use from *ibaPDA*. The individual columns in the signal list have the following meaning:

Name, Active, Actual
see *Analog tab*

Diagnostics



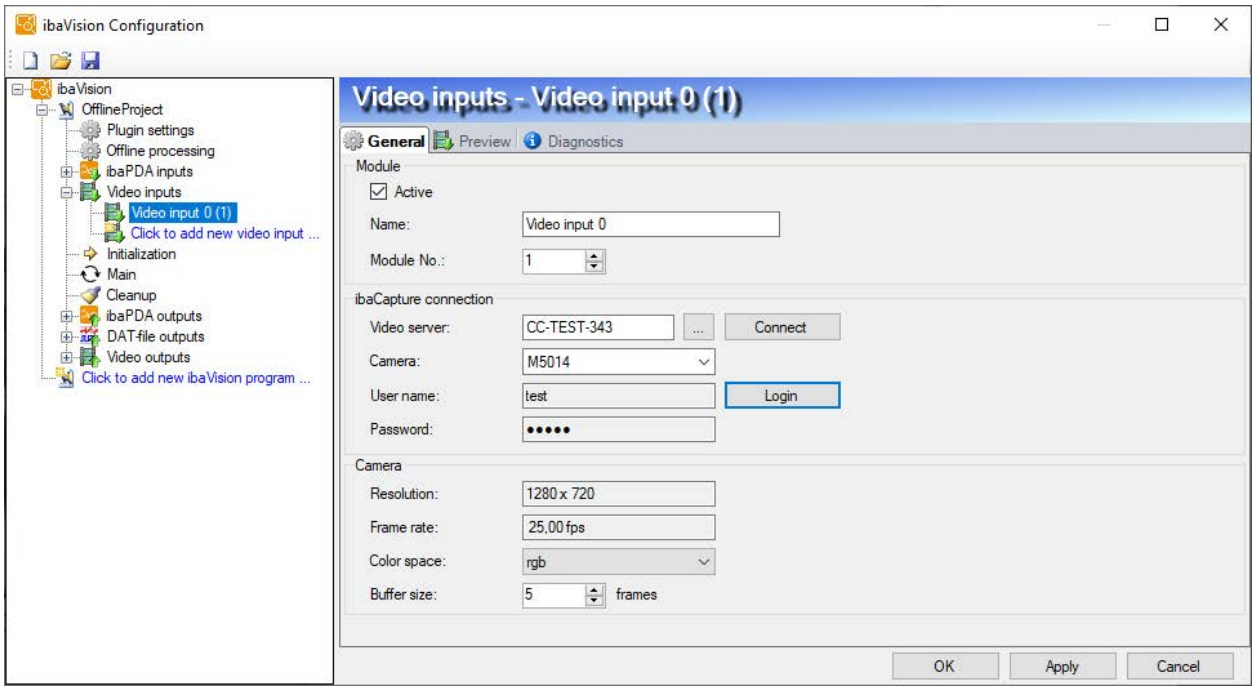
The *Diagnostics* tab displays counters for valid and invalid messages.

A click on <Reset counters> will reset the counters.

5.5.3.6 Video inputs

To add a new module Video inputs, click on the link *Click to add new video input ...*.

General tab



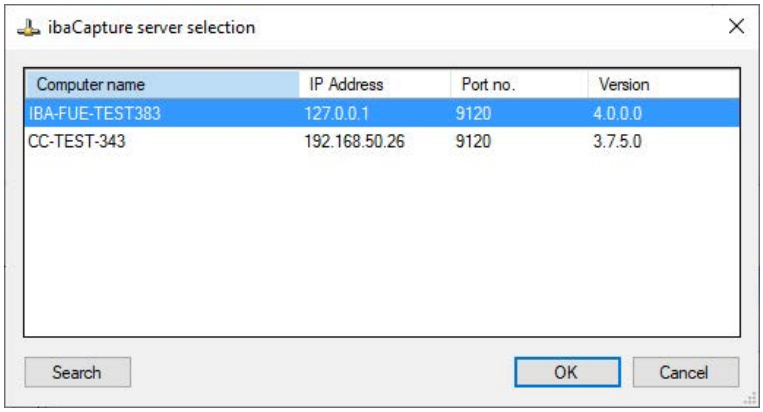
Module

See chapter ↗ *ibaPDA inputs*, page 34

ibaCapture connection

Video server

Hostname of the *ibaCapture* Server, which is used as image source. A connection can be made by manually typing and clicking <Connect>. Alternatively, a click on the browse button [...] opens a dialog window that shows available *ibaCapture* Servers.



Select the desired *ibaCapture* Server. The connection will be established by clicking <OK>.

Camera

Select a camera which images will be used as iconic input from the dropdown list.

Login

Click the <Login> button to open the authentication dialog for the selected *ibaCapture* Server.

Camera

Resolution, Frame rate

Resolution and frame rate of the selected camera are retrieved from the *ibaCapture* Server.

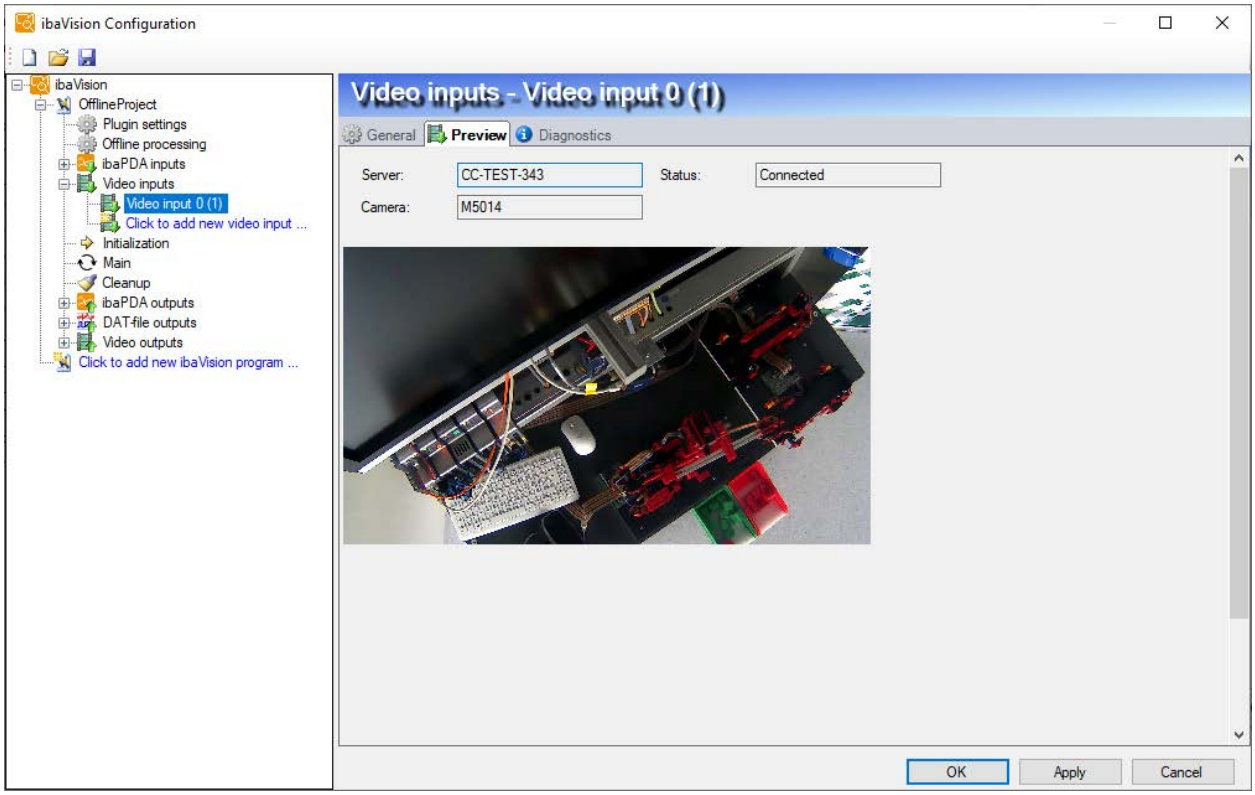
Color space

Select a color space from the dropdown list. The images from the video input will be passed to the procedure in the selected color space. Available color spaces: rgb, yuv or gray.

Buffer size

Enter the number of frames that are stored in a ring buffer from which *ibaVision* can retrieve video frames.

Preview tab

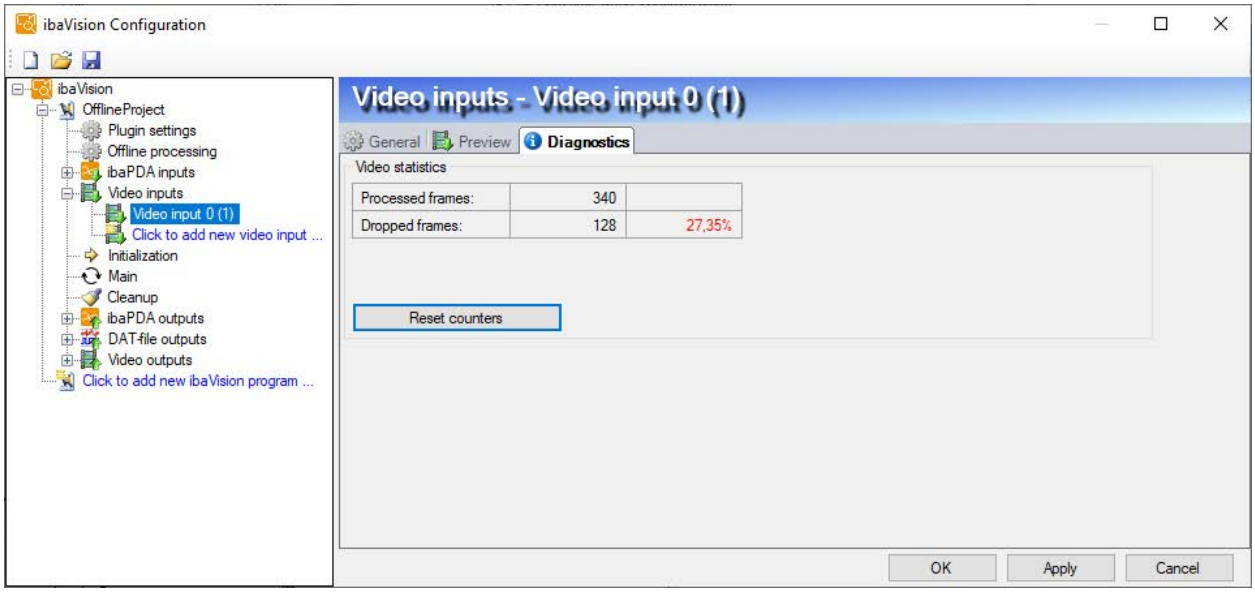


The *Preview* tab allows the user to check the video stream of the selected camera. The *Preview* tab shows the camera's name, the server and the connection status.

Note

During execution of an *ibaVision* program, video frames from *ibaCapture* are stored in an internal ring buffer. In case no new frames are present in the ring buffer at the time when the iconic input parameters need to be set, the program will wait 1 second for a new frame. If the operation times out, no new input frame will be set.

Diagnostics tab



The *Diagnostics* tab displays counters for processed frames and dropped frames. A click on <Reset counters> will reset the counters.

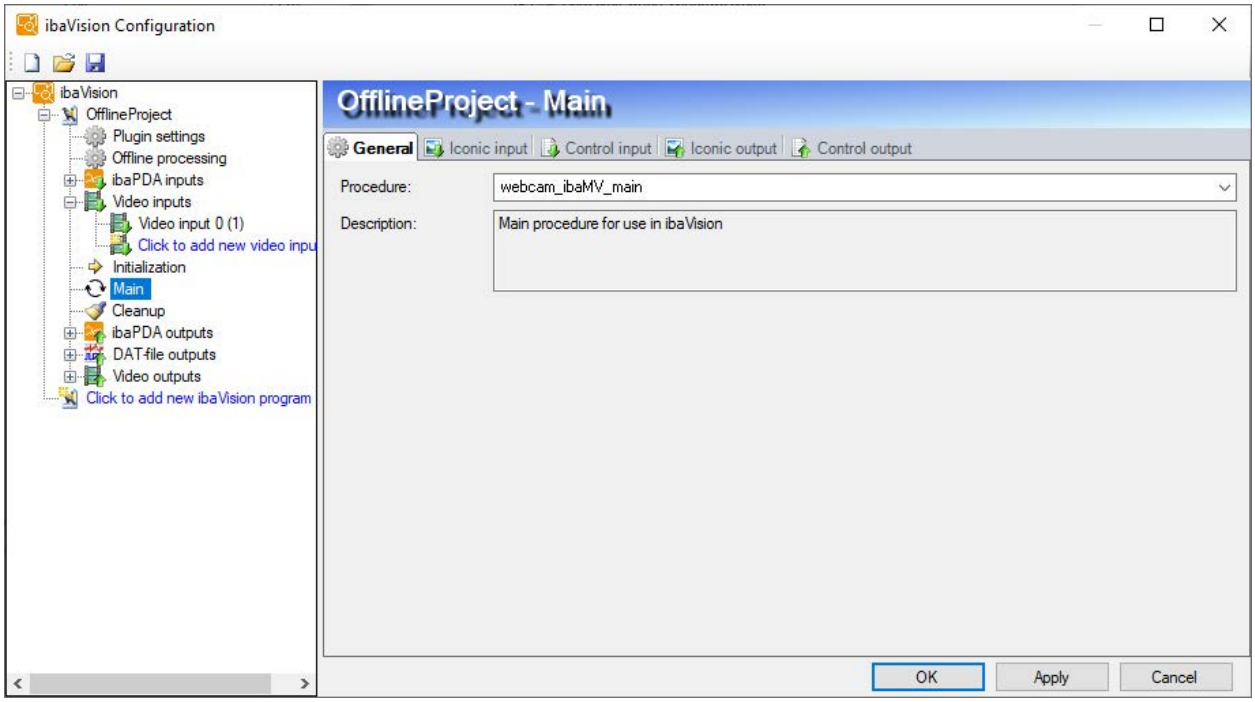
When a video source provides frames faster than they can be processed, frames will start to be dropped from the buffer.

5.5.3.7 Initialization, Main, Cleanup procedures

The configuration of the Initialization, Main, and Cleanup procedure are similar. The configuration is described in the following using the example of the Main procedure.

Click on the *Main node* to configure the Main procedure.

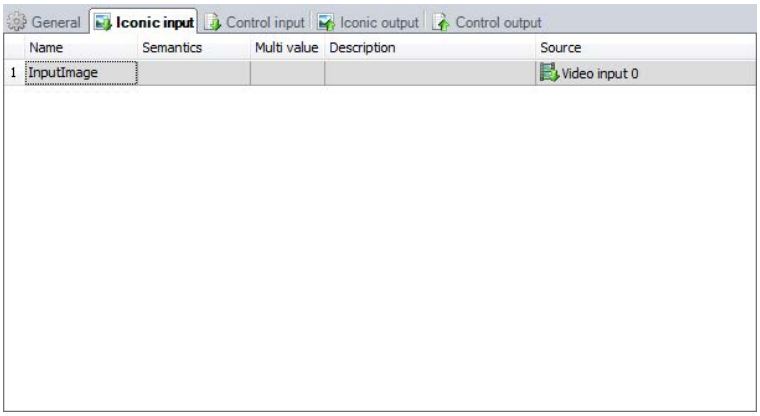
General tab



Procedure
Select the procedure to be used as Main procedure from the dropdown list. The description of the procedure is displayed in the *Description* textbox provided the procedure contains a description.

With the procedure, all configured iconic input, control input, iconic output and control output variables associated with the procedure will be loaded. The variables are displayed in their respective tabs.

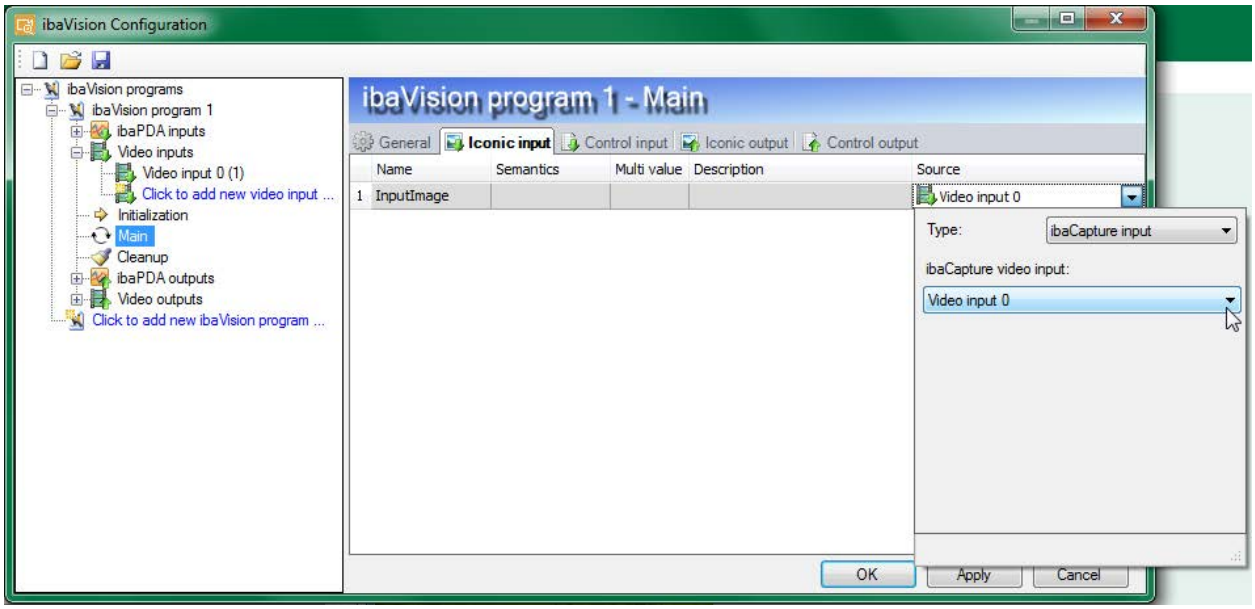
Iconic input tab



The *Iconic input* tab shows the iconic inputs configured in the selected procedure. All input variables need to be connected to a source. The tab contains the following columns:

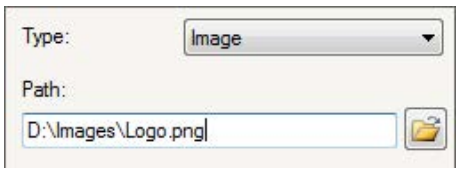
Name, Semantics, Multi value, Description
Display of the appropriate property if defined in the script.

Source
Select the source for the value of this variable.



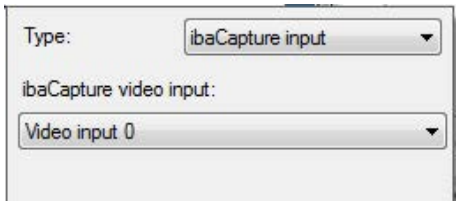
A click in a Source field opens a dialog where the source can be configured. First select the type, 3 types are available:

- Image



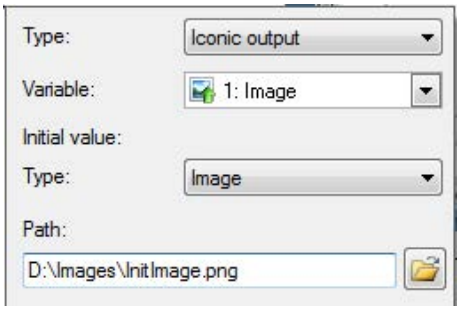
Enter the path manually to the image file or use the file browser with the button. Note that an image file is loaded only once, at the start of the ibaVision program. If the content of the image file changes during execution of the ibaVision program, the updated content will not be considered.

- ibaCapture input



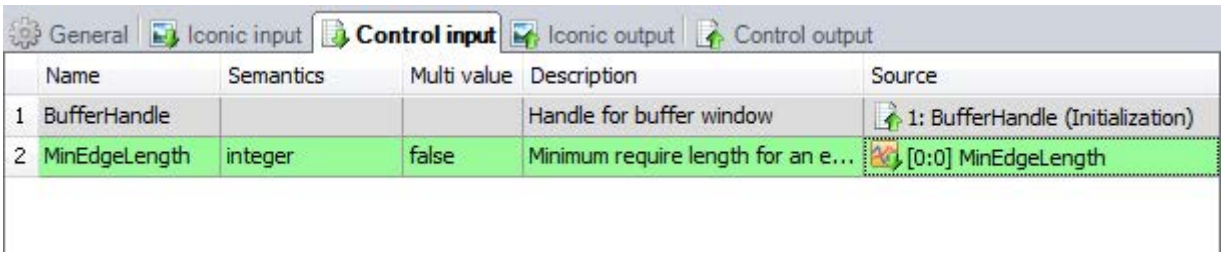
The dropdown list under ibaCapture video input lists all configured Video input modules. Select the desired module.

- Iconic output



Select the desired variable from the dropdown list. The available iconic output variables are listed in a tree structure grouped by procedure type. If necessary, an initial value can be configured for the variable. This will be the case, e.g. when the iconic output from the Main procedure is used in the Main procedure as iconic input. An initial value is not necessary, when the iconic output from the Initialization procedure is used in the Main procedure as iconic input.

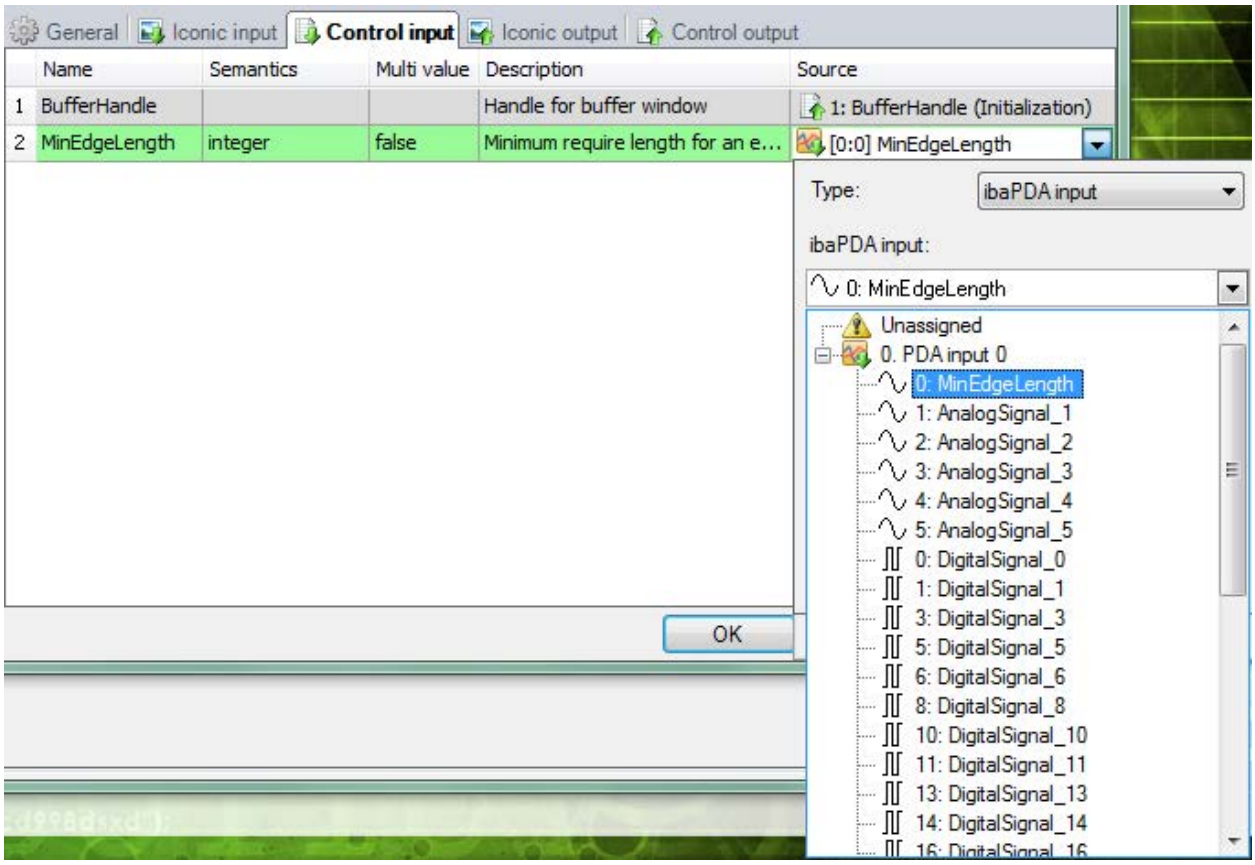
Control input tab



The *Control input* tab shows the control inputs configured in the selected procedure. The input variables need to be connected to a source here. The tab contains the following columns:

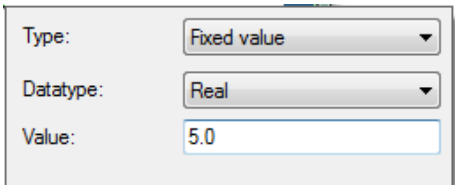
Name, Semantics, Multi value, Description
Display of the appropriate property if defined in the script.

Source
Select the source for the value of this variable.



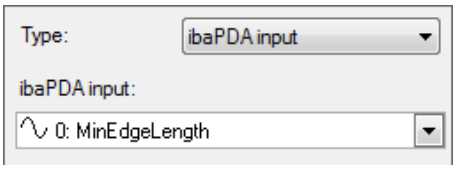
A click in the source field opens a dialog where the source can be configured. First select the type, these types are available:

- Fixed value



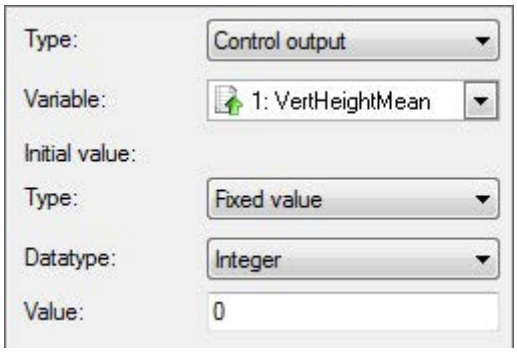
Select the data type from the dropdown list and enter the value in the textfield.

- ibaPDA input



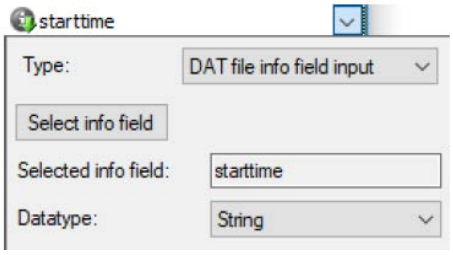
Select a signal from the dropdown list with all configured ibaPDA input signals.

■ Control output



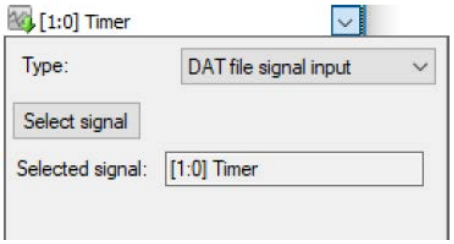
Select the desired variable from the dropdown list. The available control output variables are listed in a tree structure grouped by procedure type. If necessary, an initial value can be configured for the variable. This will be the case, e.g. when the control output from the Main procedure is used in the Main procedure as control input. An initial value is not necessary when its control input was a control output of a previously executed procedure.

■ DAT file info field input



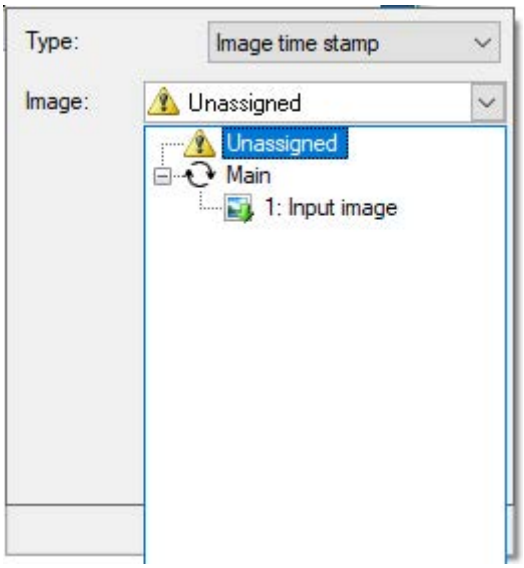
Select the desired info field. DAT file signal input is only allowed in the Main procedure.

■ DAT file signal input



Select the desired signal.

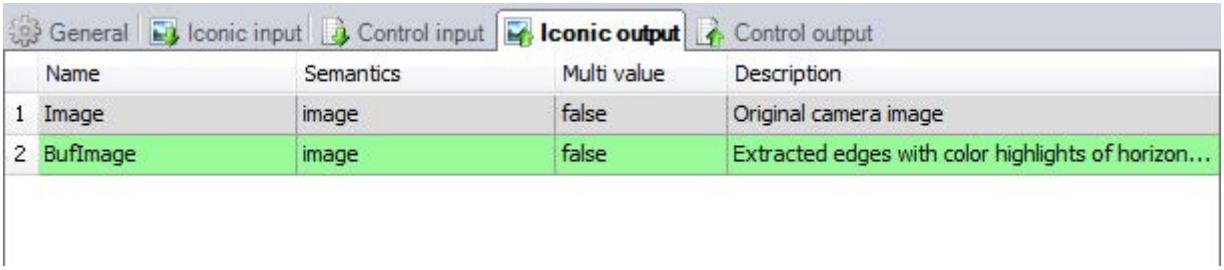
- Image time stamp



Select an iconic input that is fed by an ibaCapture input. The program will receive a tuple/array with the timestamp of the current image with these indices:

0 - year
1 - month
2 - day
3 - hour
4 - minute
5 - second
6 - millisecond

Iconic output tab

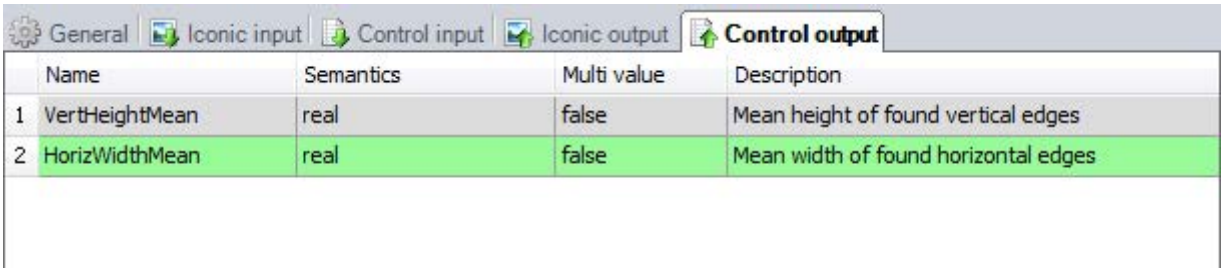


	Name	Semantics	Multi value	Description
1	Image	image	false	Original camera image
2	BufImage	image	false	Extracted edges with color highlights of horizon...

The *Iconic output* tab shows the iconic output variables configured in the selected procedure. The tab contains the following columns:

Name, Semantics, Multi value, Description
Display of the appropriate property defined in the script.

Control output tab



	Name	Semantics	Multi value	Description
1	VertHeightMean	real	false	Mean height of found vertical edges
2	HorizWidthMean	real	false	Mean width of found horizontal edges

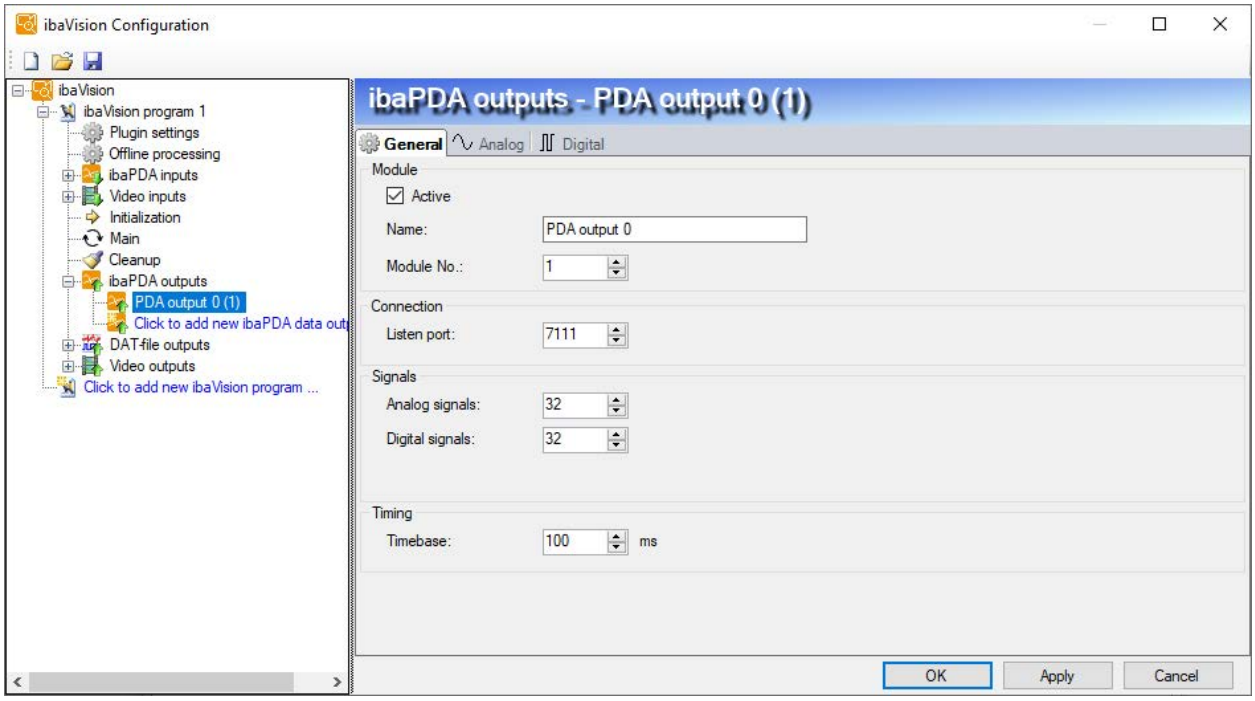
The *Control output* tab shows the control output variables configured in the selected procedure. The tab contains the following columns:

Name, Semantics, Multi value, Description
Display of the appropriate property defined in the script.

5.5.3.8 ibaPDA outputs

To add a new *ibaPDA outputs* module, click on the link *Click to add new ibaPDA data output …* under the *ibaPDA outputs* node.

General tab



Module

See chapter ↗ *ibaPDA inputs*, page 34

Connection

Listen port

Port for the connection to *ibaPDA*.

Signals

Analog signals

Defines the number of analog signals to be transmitted to *ibaPDA*. The same amount of lines will be displayed in the *Analog* tab. When the value is „0“, the *Analog* tab will not be displayed. Text signals are analog signals of the type STRING.

Digital signals

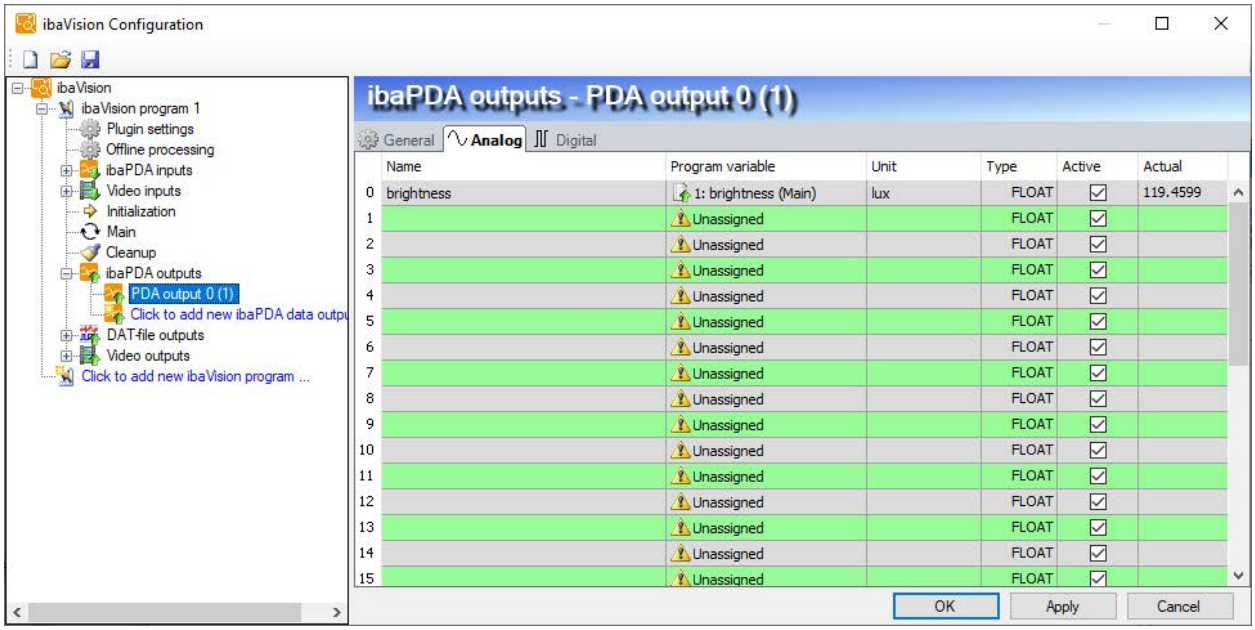
Defines the number of digital signals to be transmitted to *ibaPDA*. The same amount of lines will be displayed in the *Digital* tab. When the value is „0“, the *Digital* tab will not be displayed.

Timing

Timebase

Time interval at which this *ibaPDA output module* will send data packets to *ibaPDA*.

Analog tab



Enter the analog signals you want to send to *ibaPDA*. The individual columns in the signal list have the following meaning:

Name

Here you can enter a signal name and additionally two comments, if you click on the icon on the signal name field.

Tip

If you enter a signal name ending with a number and click on the column header (as long as the cursor is still in the name field) then all empty fields below will be filled with that name with an increasing number like an index. You may use this function in any row of the table. Fields that already have names will not be overwritten.

Program variable

Select the desired program variable from the dropdown list. The available program variables are listed in a tree structure grouped by procedure type. The value will be converted to the data-type selected in the *Type* column.

Unit

Enter a measurement unit.

Type

Data type of the signals, available data types: BYTE, INT, DINT, FLOAT, STRING.

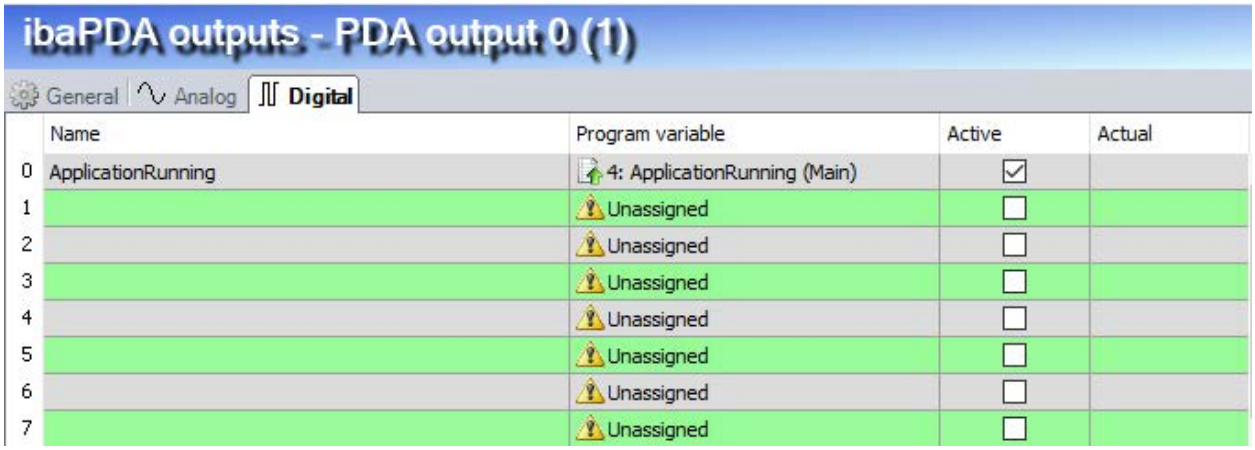
Tip

A click in the column header will apply the datatype of the selected signal to subsequent signals.

Active
If this option is selected, the signal will be used in the current configuration. Clicking the column header will apply the active state of the selected signal to subsequent signals.

Actual
Display of the currently acquired values.

Digital tab

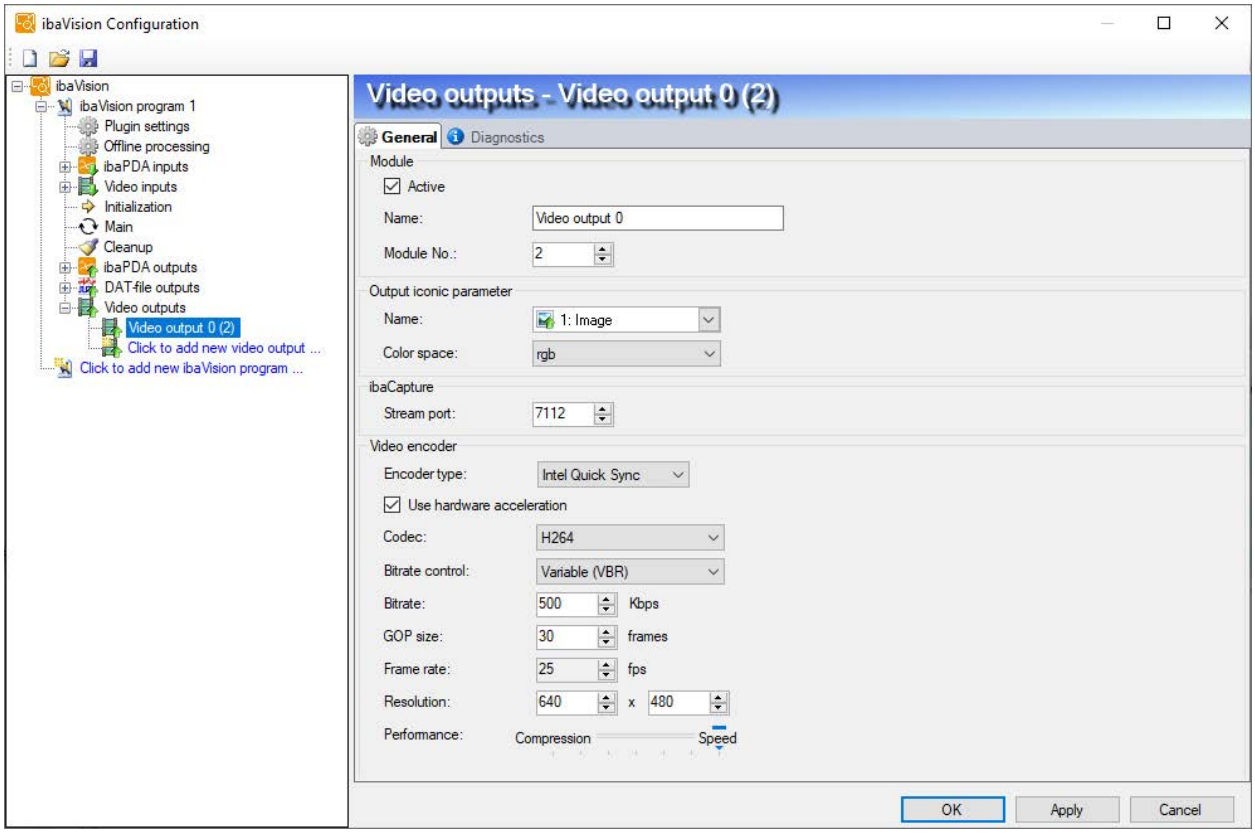


Enter the digital signals you want to send to *ibaPDA*. The individual columns in the signal list have the following meaning:

Name, Program variable, Active, Actual
see *Analog* tab

5.5.3.9 Video outputs

To add a new *Video outputs* module, click on the link *Click to add a new video output…* under the Video outputs node.



Module

See *ibaPDA input* module

Output iconic Parameter

Name

Select the output iconic parameter that is used for video output from the dropdown list.

Color space

Select the color space for the parameter, available color spaces are: rgb or gray.

ibaCapture

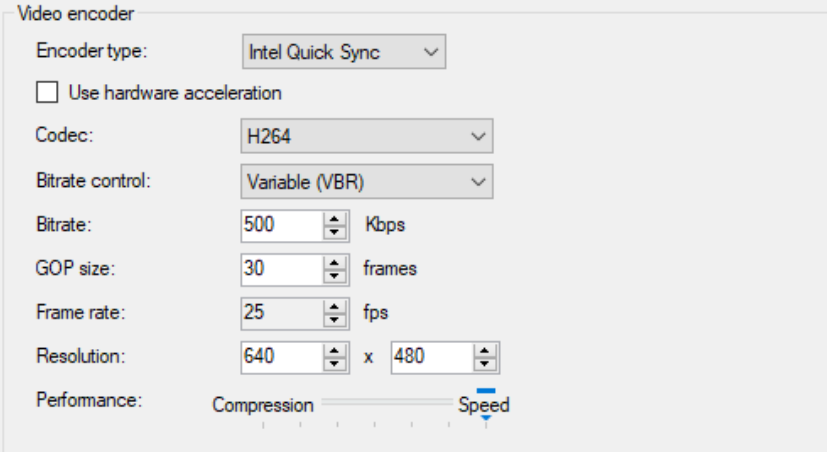
Stream Port

Port for connections to *ibaCapture*.

Video encoder

ibaVision supports Intel Quick Sync Video encoding and NVIDIA encoding. The desired encoder can be selected from a dropdown menu. The corresponding configuration options will appear accordingly.

5.5.3.9.1 Intel Quick Sync video encoding



Use hardware acceleration
If a supported Intel GPU is available on the system, hardware acceleration for video encoding can be enabled by checking this box. If the box is unchecked, video encoding will be done on the CPU.

Codec
ibaCapture supports H.264 encoding as well as H.265 (or HEVC). Select the encoding type in the dropdown menu:

- H.264
- HEVC (H.265)
  When using a hardware encoder, make sure that the device supports this codec. Software-encoding is only possible on 64 bit systems.

Bitrate control

- VBR (Variable Bit Rate)
  With VBR, the bitrate is always set according to the complexity of the taken image. That means, that a high bandwidth is required for a high activity and a low bandwidth for low activity in the video.
- CBR (Constant Bit Rate)
  If you select CBR, you can configure a fixed maximum bitrate and hence make sure that the demand for the bandwidth remains calculable. Thus, the maximum value will not be exceeded, no matter how much activity is in the video.

Bitrate
If VBR is configured for the bitrate control, you can configure here the average bitrate. If CBR is selected, you can configure here the maximum permitted bitrate.

GOP size
GOP (Group Of Pictures) defines the number of pictures after which a full picture, a so called keyframe will be stored. The pictures between the keyframes are derived from the previous keyframe, in between, only changes of the pictures are stored. With a high GOP, possible encoding errors exist longer, with a low GOP, the bitrate and hence the memory requirements of the video stream tend to grow. The default setting 30 is suitable for most of the cameras.

Frame rate
A frame rate of 25 fps usually works for the human eye to get the impression of smooth movement.

Resolution
Image resolution can be set here. The images from the iconic outputs will be scaled to the set resolution. The aspect ratio will not be preserved.

Performance slider
The performance can be adjusted with the slider to prioritize either compression or speed. Please note that a high speed causes low compression and produces higher-bandwidth data. High compression requires high computing capacity and lowers the speed.

5.5.3.9.2 NVIDIA encoding

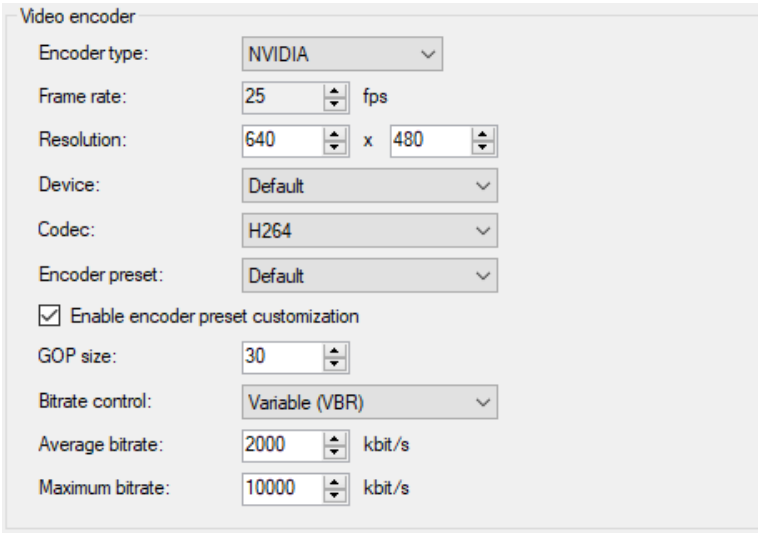
To use NVIDIA encoding, it is necessary that a supported NVIDIA GPU is installed into the *ibaVision* PC. The support matrix can be found here https://developer.nvidia.com/video-encode-decode-gpu-support-matrix.

Important: Pay attention to the column "Max # of concurrent sessions" in the support matrix. If the value is 2 or 3, only the given amount of streams per system will be supported. This is also true if multiple NVIDIA GPUs are installed. Only GPUs from the Quadro series (value "Unrestricted") will allow more concurrent streams. For encoding of video outputs in *ibaVision*, one video output requires one stream.

So if you plan to encode more than two video outputs, it is necessary to install an appropriate GPU. Running ibaCapture with GigE cameras encoded via NVIDIA also counts into the number of sessions.

In case you are not sure about your application, contact your local iba support for clarification.

Please note that when using an NVIDIA GPU, hardware accelerated Intel® Quick Sync Video encoding will be disabled. If a video output is configured to use Intel Quick Sync, *ibaVision* will automatically switch to software-based Intel® Quick Sync Video encoding.



Frame rate, resolution
see Intel Quick Sync encoding

Device
GPU on which the encoding will be performed (in case multiple GPUs are installed)

Codec
ibaVision supports H.264 and H.265 (or HEVC) encoding. The main benefit of H.265 encoding is a reduction of the required storage space compared to H.264 encoding. However, this comes at the cost of an increased demand for processing power (both during encoding and decoding). Select the encoding type in the dropdown menu:

- H.264
- HEVC (H.265)
  Device compatibility for HEVC can also be checked in the support matrix. (link at the top of this chapter).

Encoder preset
The NVIDIA encoder API provides different presets. Their names indicate their intended use. If a preset is not supported by the GPU, it reverts to the default preset.

Enable encoder preset customization
When checking this option, you can set the following properties:

GOP size
GOP (Group Of Pictures) defines the number of pictures after which a full picture, a so called keyframe will be stored. The pictures between the keyframes are derived from the previous keyframe, in between, only changes of the pictures are stored. With a high GOP, possible encoding errors exist longer, with a low GOP, the bitrate and hence the memory requirements of the video stream tend to grow. The default setting 30 is suitable for most of the cameras.

Bitrate control
- VBR (Variable Bit Rate)
  With VBR, the bitrate is always set according to the complexity of the taken image. That means, that a high bandwidth is required for a high activity and a low bandwidth for low activity in the video.
- CBR (Constant Bit Rate)
  If you select CBR, you can configure a fixed maximum bitrate and hence make sure that the demand for the bandwidth remains calculable. Thus, the maximum value will not be exceeded, no matter how much activity is in the video.

Average / maximum bitrate
If VBR is configured for the bitrate control, you can configure here the average and maximum bitrate. If CBR is selected, you can configure the average bitrate here.

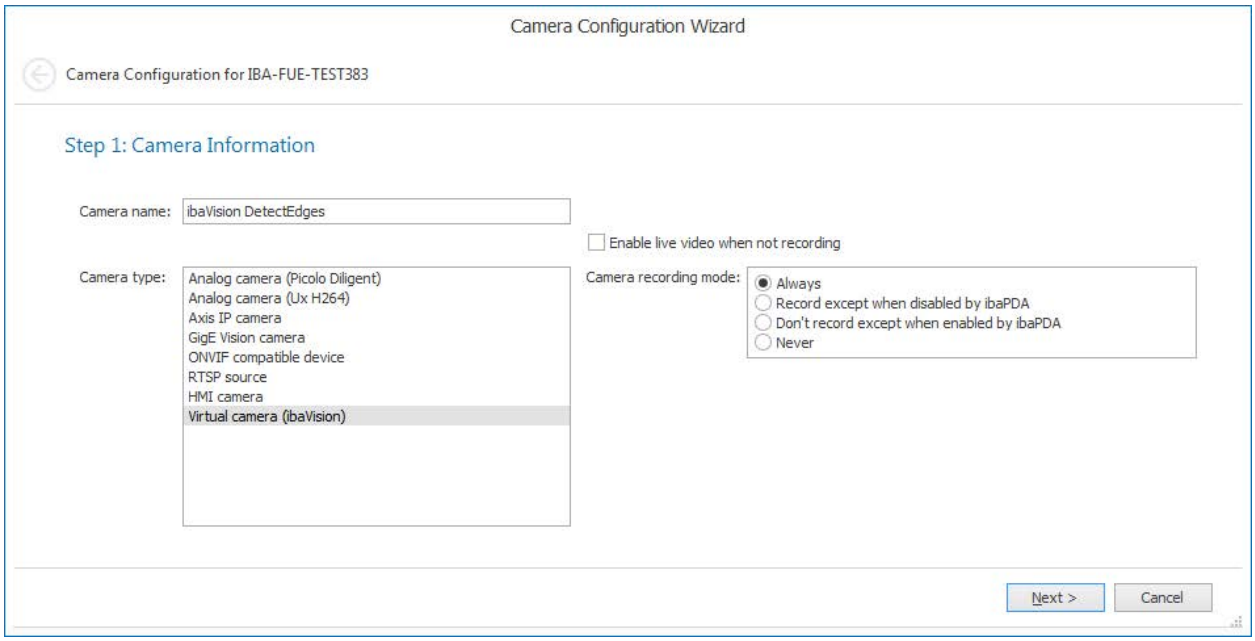
Note

Demosaicing of 8-bit Bayer pixel formats is executed on the GPU with the use of the NVIDIA Performance Primitives (NPP) library v9.2. Therefore, make sure your GPU driver supports CUDA 9.2.

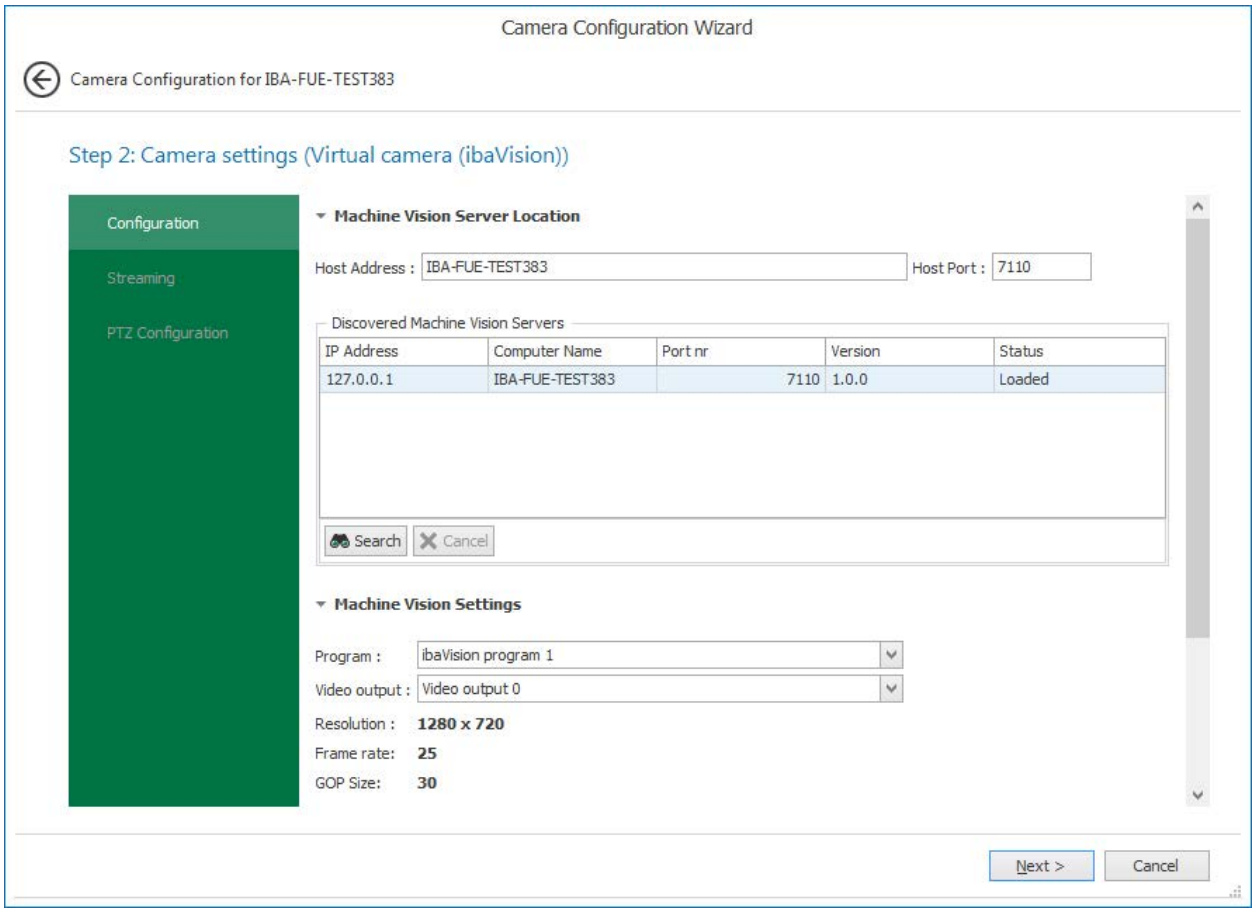
5.6 ibaCapture

Video output modules can transmit images (videos) to *ibaCapture*, which can be recorded as virtual camera. Since all relevant video parameters have already been configured in *ibaVision*, only the *Video output* module that will be accessed must be selected.

Open *ibaCapture* Manager and add a new camera. The camera configuration wizard starts. Select *Virtual camera* (ibaVision) as camera type and enter a camera name.



- Enable live video when not running
- Camera recording mode
  - *Always* (default): By default, video is always recorded, irrespective of whether a recording control was enabled in *ibaPDA* or not. Even if the recording control is enabled and camera recording is disabled in *ibaPDA*, *ibaCapture* will record videos.
  - *Record except when disabled by ibaPDA*: Video is recorded unless the recording control is enabled in *ibaPDA* and recording is disabled for the relevant camera (output signal = true).
  - *Don't record except when enabled by ibaPDA*: Video is not recorded unless the recording control is enabled in *ibaPDA* and recording is enabled for the relevant camera (output signal = true).
  - *Never*: No videos are recorded by *ibaCapture*, irrespective of whether a recording control was enabled in *ibaPDA* or not. Only a live image is provided. In this case the "storage settings" are deactivated, since no memory space is required.



Specify the camera settings in step 2 of the camera configuration wizard.

Host Address
Hostname of the computer running *ibaVision*.

Port
Port for general communication as configured in the *ibaVision* status window, see chapter ↗ *ibaVision Status Window*, page 23.

Search
When clicking the button <Search>, the local network will be searched for all servers running *ibaVision*. The available servers are listed in the grid. *ibaCapture* Manager will search the local network for all *ibaVision* instances. The available Video output modules are listed in the grid below.

Program
All *ibaVision* programs available on the specified server are listed in the dropdown list. Select the desired *ibaVision* program from the list.

Video output
Select the desired Video output module from the dropdown list.

Resolution
Video resolution as configured in *ibaVision*.

Frame rate
Video frame rate in frames per second as configured in *ibaVision*.

GOP size
GOP size as configured in *ibaVision*.

Click on <Next> and complete the configuration.

5.7 ibaPDA

Data can be transmitted between *ibaPDA* and *ibaVision* in both directions.

ibaPDA output modules configured in *ibaVision* can transmit analog, digital and text signals and to *ibaPDA*. To receive and measure this data in *ibaPDA*, *ibaVision input* modules need to be configured in *ibaPDA*.

Reversely, *ibaPDA* can transmit data to *ibaVision*. For this purpose *ibaVision output* modules must be configured in *ibaPDA*, which transmit data (analog and digital signals) to *ibaPDA input* modules in *ibaVision*.

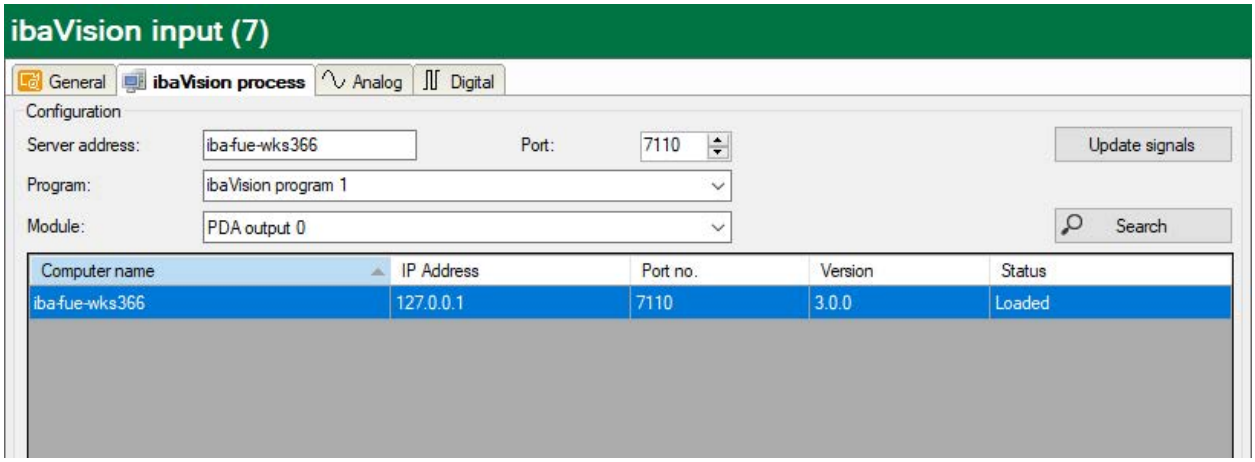
The modules *ibaVision V2 input* and *ibaVision V2 text input* module must be used when working with the previous version ibaVision-V2. These modules are not subject of this documentation.

5.7.1 ibaVision input modules

Open the I/O Manager and click on the link *Click to add module…* under the ibaCapture node. Select *ibaVision input* as module type.

In the *General* tab, enter general settings like module name, module number, etc. For detailed information, please refer to the *ibaPDA* manual.

In the *ibaVision process* tab, select the requested *ibaPDA output module* from *ibaVision*.



Server address
Name of the host running *ibaVision*. When the address has been entered, *ibaPDA* establishes a connection to the *ibaVision* instance running on this host and retrieves all *ibaVision* programs.

As an alternative to manually enter the server address, click the <Search> button and the local network will be searched for all hosts running *ibaVision*. The available hosts are listed in the grid below and can be selected.

Port
Port for general communication as configured in the *ibaVision* status window, see chapter
↗ *ibaVision Status Window*, page 23.

Program
All *ibaVision* programs available on the specified server are listed in the dropdown list. Select the desired *ibaVision* program from the list.

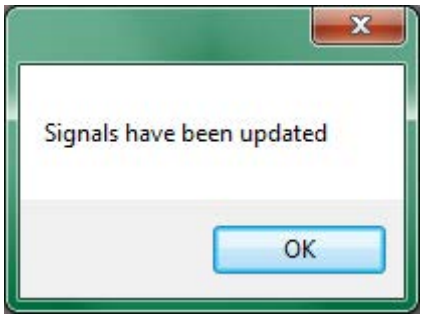
Module

All *ibaPDA output* modules configured in the selected *ibaVision* program are listed in the drop-down list. Select the module from the list whose data shall be received.

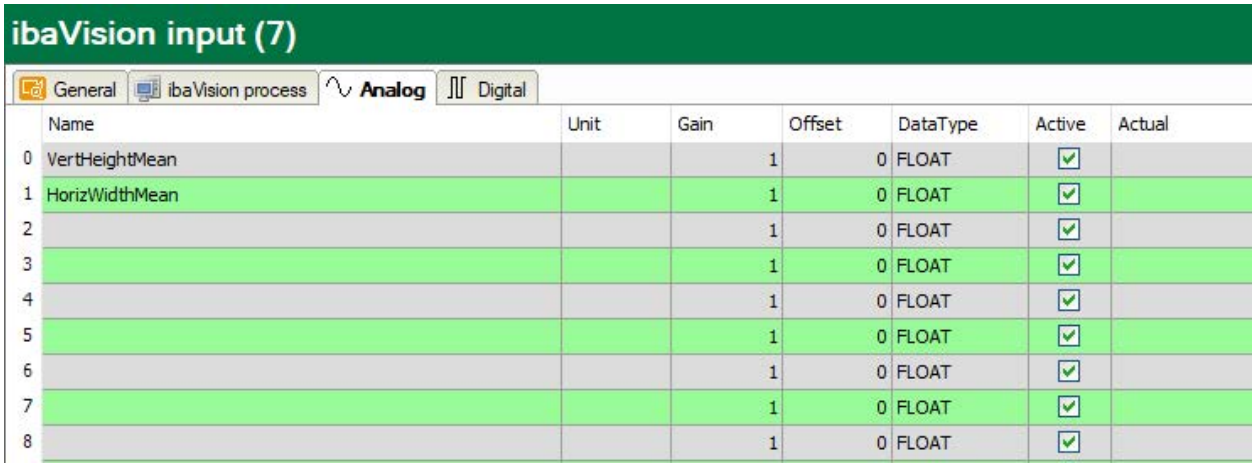
Update signals

When the desired *ibaPDA output* module is selected, click the <Update signals> button. The signals configured in the *ibaPDA output* module in *ibaVision* will automatically be mapped to the corresponding signals in the *ibaVision input* module in *ibaPDA*.

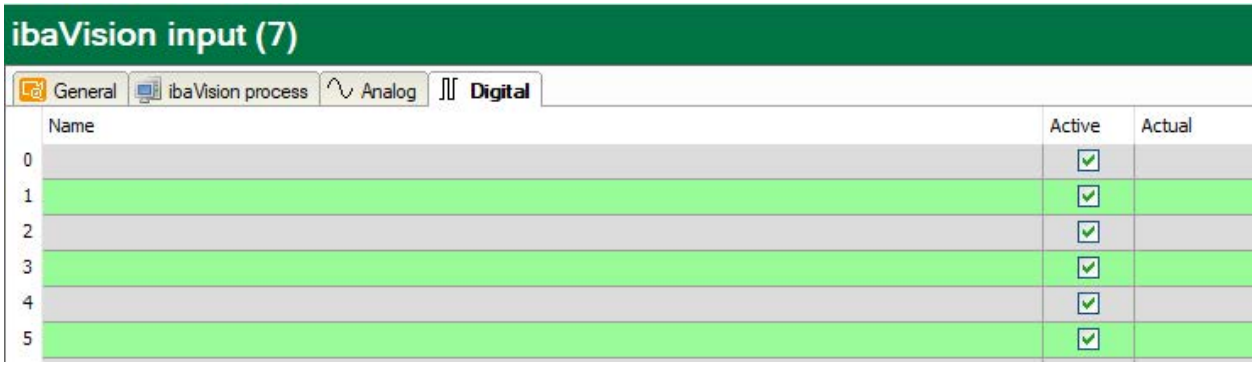
The following message confirms that the signals have been successfully mapped.



The signal mapping can be checked in the *Analog* and *Digital* tabs. The signals configured in the *ibaPDA output* module in *ibaVision* are listed here. No further action is necessary.



	Name	Unit	Gain	Offset	DataType	Active	Actual
0	VertHeightMean		1	0	FLOAT	☑	
1	HorizWidthMean		1	0	FLOAT	☑	
2			1	0	FLOAT	☑	
3			1	0	FLOAT	☑	
4			1	0	FLOAT	☑	
5			1	0	FLOAT	☑	
6			1	0	FLOAT	☑	
7			1	0	FLOAT	☑	
8			1	0	FLOAT	☑	



	Name	Active	Actual
0		☑	
1		☑	
2		☑	
3		☑	
4		☑	
5		☑	

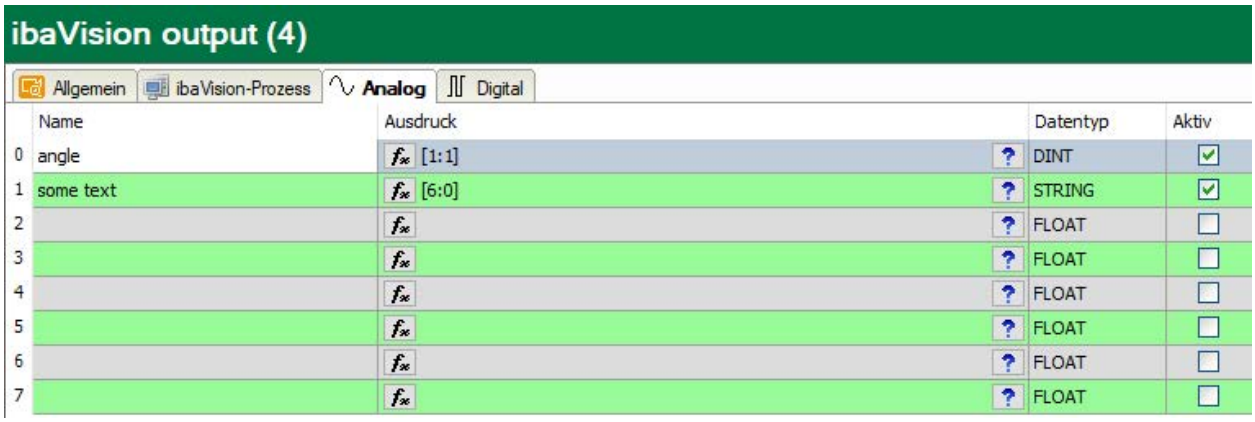
5.7.2 ibaVision output module

Change to the *Outputs* tab of the I/O Manager. Click *Click to add module …* under the *ibaCapture* node in the tree structure and select the *ibaVision output* type.

In the *General* tab, enter general settings like module name, module number, etc. For detailed information, please refer to the *ibaPDA* manual.

In the *ibaVision process* tab, select the requested *ibaPDA input* module from *ibaVision*. The configuration of an *ibaVision output* module is analogous to the *ibaVision input* module, see chapter ↗ *ibaVision input modules*, page 57.

After having updated the signals, an expression has to be configured which defines the signal value that is sent to *ibaVision*. Select a signal or enter an expression in the *Expression* column in the *Analog* and *Digital* tab. You can also open the expression editor with the fx symbol in order to create the desired expression. For detailed information, please refer to the *ibaPDA* manual.



6 Developing ibaVision programs in HDevelop

6.1 Accessing video data from ibaCapture in HDevelop

During the development of Machine Vision applications, image data is regularly used for tuning and debugging purposes.

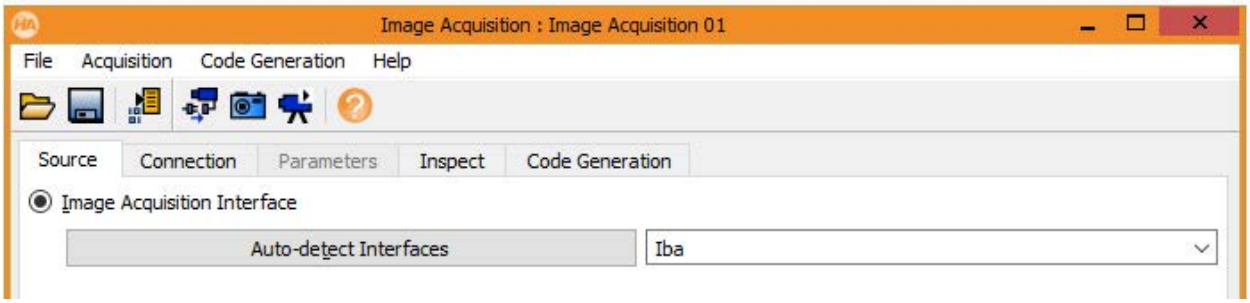
In order to access video data from *ibaCapture*, there is the option to install the extension “ibaHALCONCapture” (↗ *Installation*, page 17) with the *ibaVision* installer.

Note

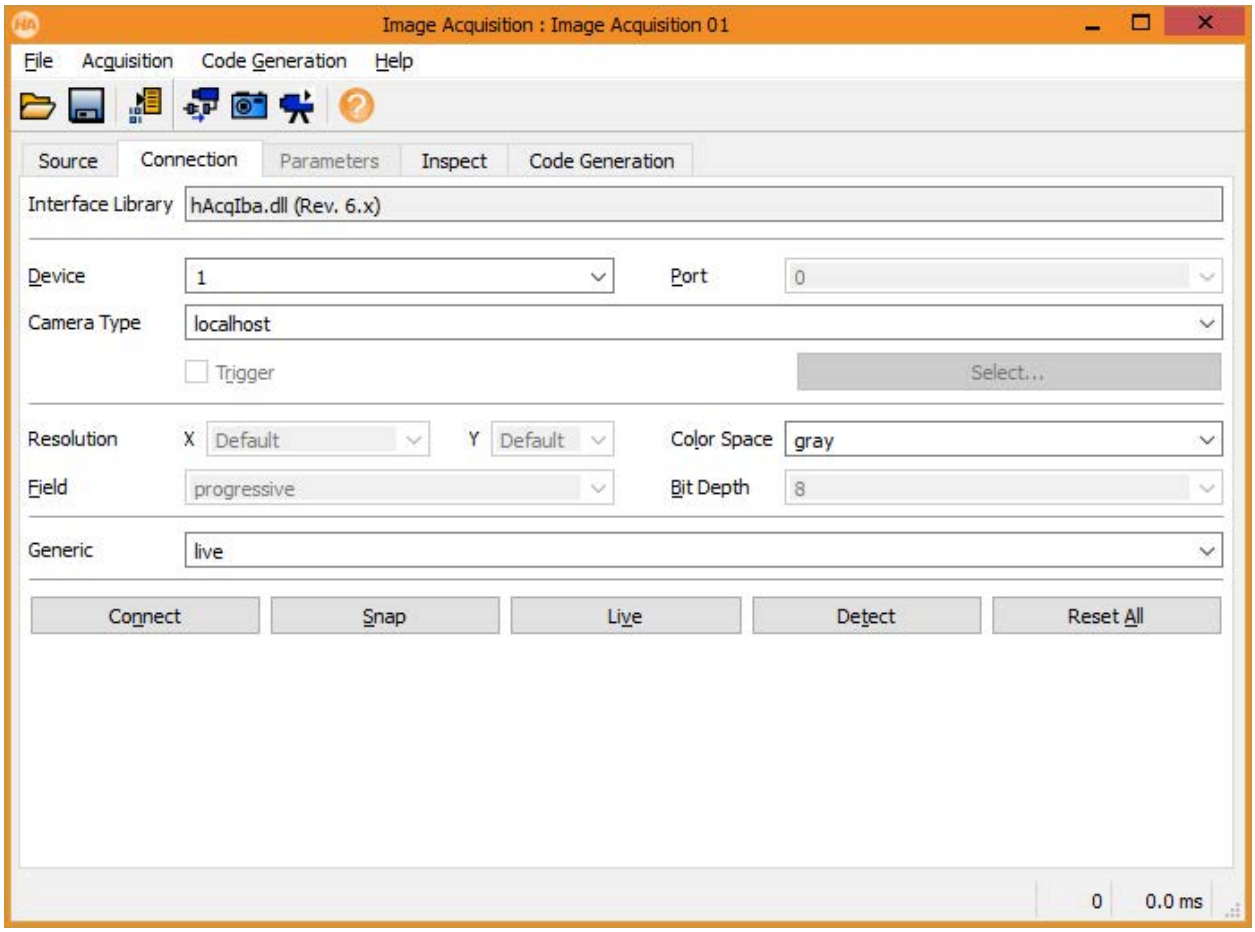
This acquisition interface is not required to execute HALCON programs in *ibaVision*. It is intended to allow access to *ibaCapture* video sources during development in HDevelop.

This is an HDevelop image acquisition interface that allows accessing video directly from *ibaCapture*. To use this interface it is required to also install *ibaCapture* Player on the HDevelop PC.

If the “ibaHALCONCapture” option has been installed, an “Iba” image acquisition interface is available in HDevelop and HDevelop XL.

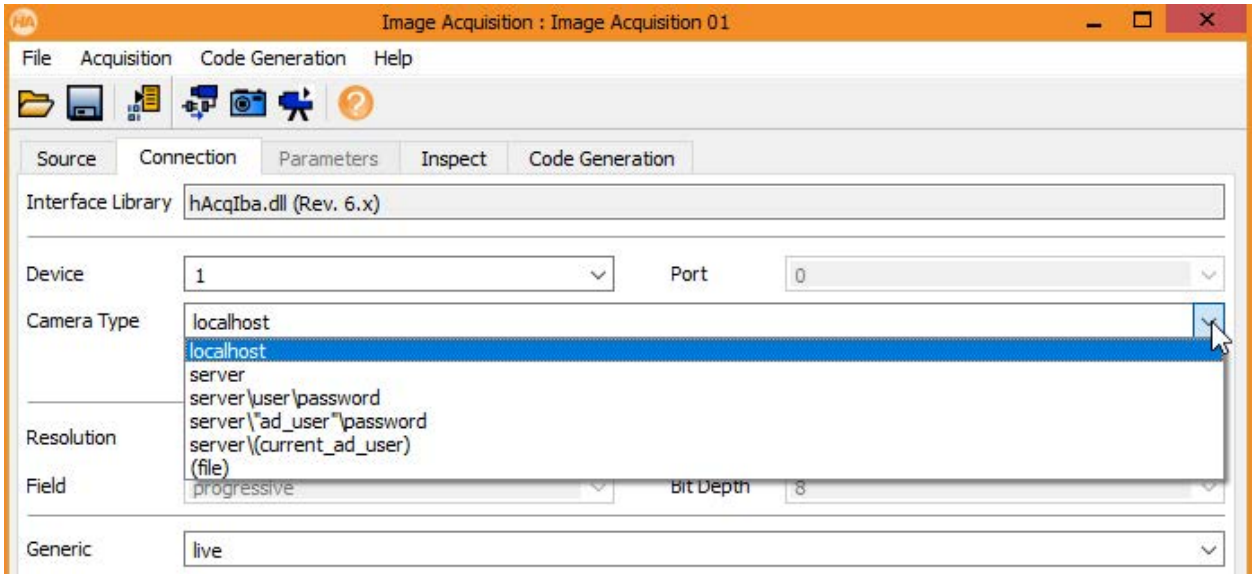


With this it is possible to access *ibaCapture* Server instances directly over the network for access to live video or stored video. Also, MP4 files exported from *ibaCapture* can be read with this interface.



When using the HDevelop Image Acquisition assistant, specify the server to connect to in the “Camera Type” input field. Possible values are hostname or IP address of the server.

The camera on the *ibaCapture* Server can be selected with the “Device” input field. This accepts either the camera id (numeric) or the camera name (text).



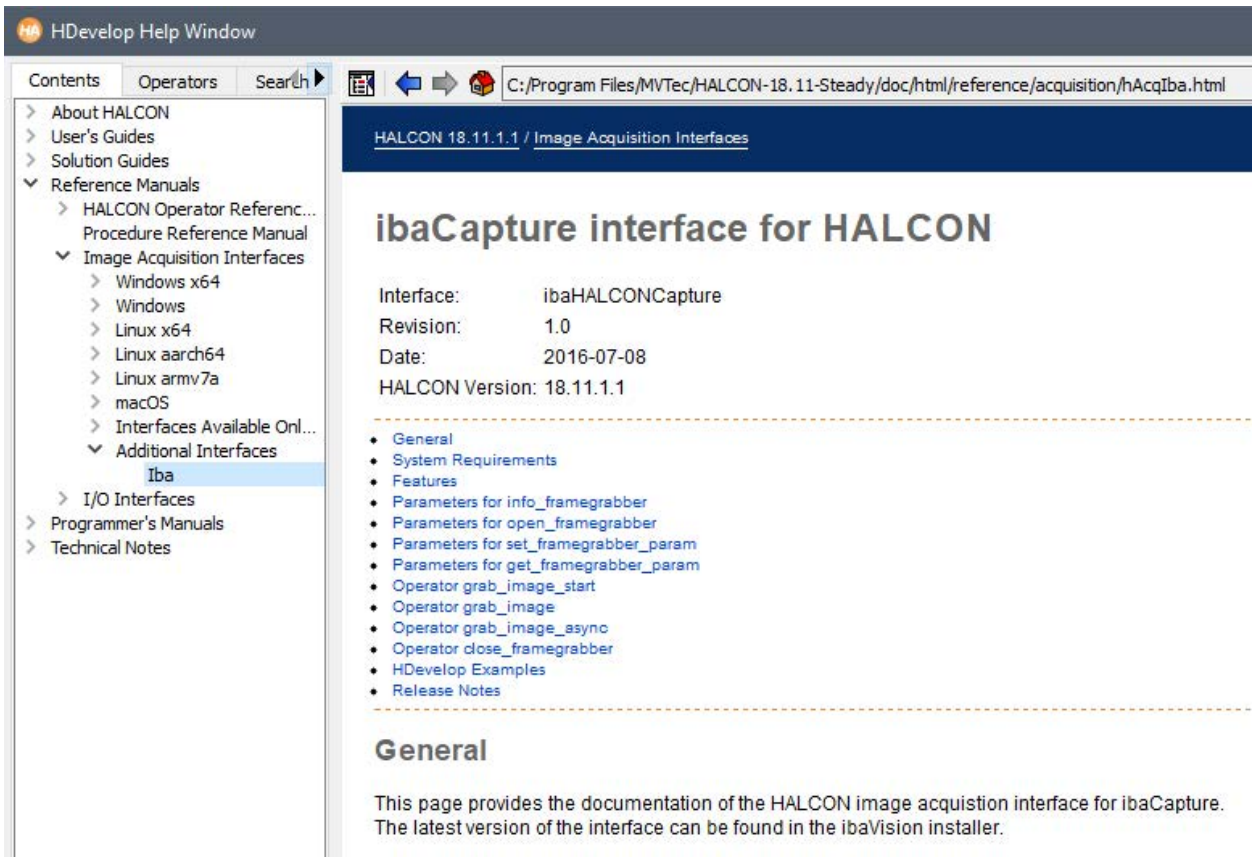
The “Generic” input field allows selecting the access mode.

Once all parameters have been set, it is possible to test the interface by using the “Connect” and “Snap”/”Live” buttons.

On the *Code Generation* tab, clicking “Insert Code” will generate the basic code to use the image acquisition interface in a Halcon program.

The HDevelop User’s Guide contains more information about the Image Acquisition Assistant.

As soon as the “ibaHALCONCapture” interface has been installed, it is also possible to open the interface documentation in the HDevelop Help Window.



Examples

Opening a live video stream from an ibaCapture Server:

```
open_framegrabber('Iba', 0, 0, 0, 0, 0, 0, 'progressive', 8, 'rgb', ‘live’,
'false', 'server\\user\\password', '1', 0, 0, AcqHandle)
```

If user management is not enabled on the *ibaCapture* Server, the user and password information can be omitted.

Opening a stream of recorded video from an ibaCapture Server:

```
open_framegrabber('Iba', 0, 0, 0, 0, 0, 0, 'progressive', 8, 'rgb', 'dd/MM/yyyy
HH.mm:ss;2', 'false', 'server', '1', 0, 0, AcqHandle)
```

In this example, the server does not have user management enabled. The timestamp to start replaying is given in the “Generic” parameter. Optionally, the replay speed can also be specified, this is done by adding “;2” to the start timestamp.

Reading video from an exported MP4 file:

```
open_framegrabber('Iba', 0, 0, 0, 0, 0, 0, 'progressive', 8, 'rgb', ';1', 'false',
'(file)', '<Path to file>', 0, 0, AcqHandle)
```

In this case, the "CameraType" parameter is set to “(file)”, the path to the MP4 file needs to be given in the "Device" parameter.

6.2 Writing log messages from the HDevelop program

Most iba software tools are capable of writing log files. These are often helpful when attempting to diagnose unexpected behavior.

As *ibaVision* programs also write log files, we also provide an operator to write custom log information coming from the HDevelop code.

In order to use this feature, *ibaVision* needs to be installed with the “ibaHALCONLogger” option enabled (see chapter ↗ *Installation*, page 17).

Once installed, the operators `iba_create_log` and `iba_write_log` will be available in HDevelop. A detailed usage description is available in the HDevelop operator documentation.

6.3 Example HDevelop program

To get started, the basic structure of an *ibaVision* program has been described in chapter ↗ *Working principle*, page 8. We will also provide an example program here where the most important *ibaVision* features will be included.

The example program is available on GitHub:
https://github.com/iba-ag/ibaVision-Example-Scripts.

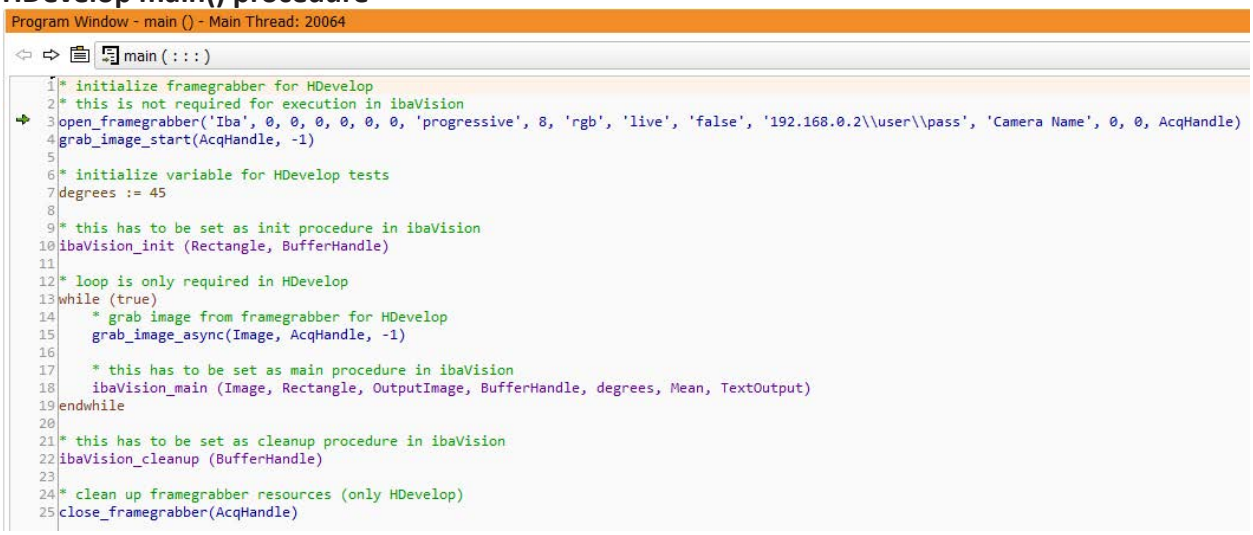
The listed code in this manual may not match the example from GitHub in all details.

The example program will read images from an *ibaCapture* Server, evaluate the average brightness value and rotate the image by a given angle. The value for the rotation angle can be passed dynamically from an *ibaPDA* server.

On the outputs, there will be a few numeric and text values as well as the rotated image. This stream of images can then be encoded for storage in an *ibaCapture* Server.

Listing

HDevelop main() procedure




```
1  * initialize framegrabber for HDevelop
2  * this is not required for execution in ibaVision
3  open_framegrabber('Iba', 0, 0, 0, 0, 0, 0, 'progressive', 8, 'rgb', 'live', 'false', '192.168.0.2\\user\\pass', 'Camera Name', 0, 0, AcqHandle)
4  grab_image_start(AcqHandle, -1)
5
6  * initialize variable for HDevelop tests
7  degrees := 45
8
9  * this has to be set as init procedure in ibaVision
10 ibaVision_init (Rectangle, BufferHandle)
11
12 * loop is only required in HDevelop
13 while (true)
14     * grab image from framegrabber for HDevelop
15     grab_image_async(Image, AcqHandle, -1)
16
17     * this has to be set as main procedure in ibaVision
18     ibaVision_main (Image, Rectangle, OutputImage, BufferHandle, degrees, Mean, TextOutput)
19 endwhile
20
21 * this has to be set as cleanup procedure in ibaVision
22 ibaVision_cleanup (BufferHandle)
23
24 * clean up framegrabber resources (only HDevelop)
25 close_framegrabber(AcqHandle)
```

This procedure is the entry point for program execution in HDevelop. It will however not be executed by *ibaVision* by default.

For this example, the main() procedure is structured in a way that allows it to be executed from both HDevelop and *ibaVision*.

These requirements are considered here to allow operation in both HDevelop and *ibaVision*:

- In *ibaVision*, the specification of three procedures (init, main and cleanup) is required. These are called from the HALCON main() procedure in lines 10, 18 and 22.
- Image acquisition from *ibaCapture* will be handled in *ibaVision*. For operation in HDevelop, a framegrabber instance is managed outside the three *ibaVision* procedures.
- The main procedure that is specified in *ibaVision* must not contain an infinite loop. While *ibaVision* is working, *ibaVision* will call the main procedure repeatedly to retrieve result values after every execution. This behavior is simulated with the while-loop (line 13) for execution in HDevelop.
- In the ibaVision_init() procedure a buffer window is created to allow flexible drawing operations for output images. The drawback of this approach is execution time. For time-critical applications where output images are required, other possibilities of dynamically creating images with HALCON operators should be considered.

To avoid runtime errors in HDevelop, on line 7 the input variable that *ibaVision* would receive from *ibaPDA* is initialized with a static value. In *ibaVision*, these can be set as static values but could also be read from *ibaPDA* inputs.

To actually execute this example in HDevelop, you need access to an *ibaCapture* Server in the network.

It is necessary to modify the parameters of open_framegrabber in line 3:

- ‘192.168.0.2\\user\\pass’ needs to be adapted to the actual network address of an *ibaCapture* Server. If user management is enabled, specify the credentials instead of “user” and “pass”. Otherwise only the server address needs to be specified.
- ‘Camera Name’ needs to be changed to a camera name that is actually available on the *ibaCapture* Server where HDevelop connects.
- The parameter ‘rgb’ is typically used for color cameras. If monochrome images should be acquired, set this to ‘gray’.

Other possibilities of accessing *ibaCapture* video data are described in the chapter ↗ *Accessing video data from ibaCapture in HDevelop*, page 60.

ibaVision_init() procedure

Program Window - ibaVision_init ()

ibaVision_init (: Rectangle : : BufferHandle)

```
1  global def tuple g_maincounter
2  global def tuple g_width
3  global def tuple g_height
4
5  * initialize global variables
6  g_maincounter := 0
7  g_width := 640
8  g_height := 640
9
10 set_system('width', g_width)
11 set_system('height', g_height)
12
13 * open log file
14 LogFileName := './log/ibaVision_example.txt'
15 iba_create_log(LogFileName, ' ==== ibaVision example log ==== ', 'True')
16 iba_write_log('Debug', 'Starting initialization procedure')
17
18 * try-catch block to handle exceptions internally
19 try
20     * create rectangular region for brightness evaluation
21     gen_rectangle1(Rectangle, 0, 0, g_height-1, g_width-1)
22
23     * initialize buffer for writing images
24     open_window(0, 0, g_width, g_height, 0, 'buffer', '', BufferHandle)
25 catch (Exception)
26     * log example for exceptions
27     dev_get_exception_data (Exception, 'error_message', Message)
28     iba_write_log ('Exception', 'Exception in ibaVision_init: ' + Message)
29     dev_get_exception_data (Exception, 'procedure', proc)
30     dev_get_exception_data (Exception, 'program_line', line)
31     iba_write_log ('Exception', 'Exception in ibaVision_init: procedure ' + proc + ', line ' + line)
32 endtry
33
34 return ()
```

The init procedure is intended to be run once before the actual image processing starts. All resources that need to be available during image processing can be initialized here.

Our example has the following sections:

Global variable definitions
(lines 1 to 3)

Global variables are defined here. Global variables can be used across multiple procedures in a HALCON program. See the HALCON Programmer’s Manual for more information on global variables.

Initialize global variables
(lines 6 to 8)

The global variables are initialized for program execution. Note that it would also be possible to read the initialization values from inputs of the init procedure instead of hard-coding values here.

Set system
(lines 10 and 11)

The operator *set_system* for width and height is used to maintain properly scaled image output. The values for width and height should be set to the width and height of the output image.

Open log file
(lines 14 to 16)

This creates a log file where messages from the HALCON program can be written to. Be aware that *ibaVision* needs to be installed with the ibaHALCONLogger option enabled to use logging.

Create rectangular region
(line 21)

A region is generated for use in the main procedure.

Initialize buffer for writing images
(line 24)

This operator creates a buffered window. This will be used for drawing the output images in the main procedure.

Try-catch-block
(Lines 19, 25 to 32)

The try-catch mechanism is a widely used programming technique for error handling. In case a so-called exception happens during the execution of the code in the try-block, its message will be logged in the catch-block, using the previously defined log file. More information on exception handling can be found in the HALCON Programmer's Manual.

Keep in mind that some programming tools like e.g., barcode models for barcode readers, only need to be initialized once and can be used indefinitely. Such initialization commands should always happen in the init procedure of *ibaVision* programs.

If the image processing does not require any initialization commands, the procedure can also be left empty except from the "return()" operator.

ibaVision_main() procedure

Program Window - ibaVision_main ()

ibaVision_main (Image, Rectangle : OutputImage : BufferHandle, degrees : brightness, textOutput)

```
1  global tuple g_maincounter
2  global tuple g_width
3  global tuple g_height
4
5  try
6      * count number of procedure executions
7      if(g_maincounter > 9999999)
8          g_maincounter := 0
9      else
10         g_maincounter := g_maincounter + 1
11     endif
12
13     * prepare input image
14     zoom_image_size(Image, ImageZoom, g_width, g_height, 'constant')
15     count_channels(ImageZoom, Channels)
16     if(Channels == 3)
17         rgb1_to_gray(ImageZoom, ImageGray)
18     else
19         copy_image(ImageZoom, ImageGray)
20     endif
21
22     * determine brightness
23     intensity (Rectangle, ImageGray, brightness, Deviation)
24
25     * rotate image
26     rotate_image (ImageZoom, ImageRotate, degrees, 'constant')
27
28     * build textOutput
29     textOutput := 'Maincounter: ' + g_maincounter$'10d' + ' | Brightness: ' + brightness$'8.2f' + ' | Channels: ' + Channels
30
31     * write image to buffer
32     clear_window(BufferHandle)
33     disp_obj (ImageRotate, BufferHandle)
34
35     * write text to buffer
36     disp_message(BufferHandle, textOutput, 'window', 24, 24, 'black', 'true')
37
38     * get image from buffer
39     dump_window_image(OutputImage, BufferHandle)
40 catch (Exception)
41     * log example for exceptions
42     dev_get_exception_data (Exception, 'error_message', Message)
43     iba_write_log ('Exception', 'Exception in ibaVision_main: ' + Message)
44     dev_get_exception_data (Exception, 'procedure', proc)
45     dev_get_exception_data (Exception, 'program_line', line)
46     iba_write_log ('Exception', 'Exception in ibaVision_main: procedure ' + proc + ', line ' + line)
47 endtry
48
49 return ()
```

This procedure will be called repeatedly by *ibaVision*. Before it is called, all input values (Image, Rectangle, BufferHandle, degrees) are updated with the current values from the respective data sources.

After the execution, the values from all output values (OutputImage, brightness, textOutput) will be forwarded to the defined output channels.

Our example has the following sections:

Global variable definitions
(lines 1 – 3)

To use defined global variables in this procedure, they need to be defined at the beginning.

Count number if procedure executions
(lines 7 – 11)

This is optional for the actual image processing but a very helpful tool for monitoring the execution of this program.

Prepare input image
(lines 14 – 20)

All input images will be resized to 640 x 640 pixels. This size was set globally in the init procedure. In case a color image (RGB) was used as input, it will be converted into a grayscale image.

Determine brightness
(line 23)

The used HALCON operator determines the average brightness of all pixels in a given region. The region in this case is defined by the object “Rectangle”, which also has a size of 640 x 640 pixels.

The output value “Mean” will then be passed to the calling procedure through the output parameters.

Rotate image
(line 26)

After determining the brightness, the image is rotated by the given number of degrees. While the value of the variable “degrees” is constant when executed in HDevelop, a dynamic value can be assigned in *ibaVision*, for example coming from an *ibaPDA* input module. That way, the rotation angle could be different for every new image.

Build textOutput
(line 29)

Maincounter, brightness and number of channels are combined into one string.

Write image to buffer
(lines 32, 33)

To prepare the output image, the rotated image is now written to the buffer window.

Write text to buffer
(line 36)

For demonstration purposes, the content of the variable “textOutput” is written to the buffer window.

Get image from buffer
(line 39)

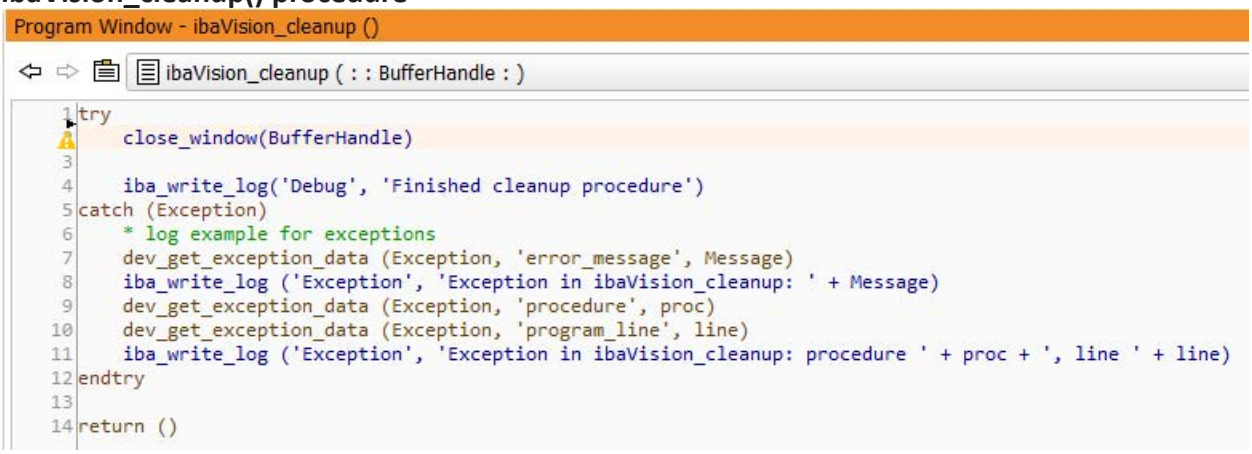
The current state of the buffer window with the recently painted image and string is written to the image object “OutputImage”. This will then be passed as an output parameter from the ibaVision_main procedure.

At the end there is a catch-block where messages from occurred exceptions will be logged.

Please keep in mind that this is just an example to show the possibilities of manipulating data and images in HALCON and communicate the results with other tools in the iba-system.

Other image processing tasks may require programs with significantly higher complexity.

ibaVision_cleanup() procedure



The cleanup procedure should be used to close all open resources. Typically, open resources will have been opened in the init procedure.

In this example, the buffer window needs to be closed.

To indicate that the execution of the cleanup procedure was successful, a final message will be written to the log.

Like in the other procedures the try-catch-block is used to catch and log exceptions.

7 Notes on Python installation

In order to use the Python plugin provided by *ibaVision*, Python needs to be installed separately.

This can either be done as a system-wide installation or limited to the user account.

In the Python v3.8 installer, it is at the moment necessary to check the “Add Python 3.8 to PATH” checkbox, otherwise *ibaVision* won’t be able to use the installation.

When Python has been successfully installed, another requirement is the installation of the numpy package using Python’s package manager “pip”. Without numpy installed, the Python plugin in *ibaVision* will always report errors.

The easiest way usually is to open a Commandline Prompt or the Powershell and to type

```
pip install numpy
```

Depending on the way Python was installed, it may be necessary to run this command with Administrator privileges. It however always requires internet access.

7.1 Installing Python packages without direct internet access

To install Python packages on a system without direct internet access, you first need to collect the packages on a system with Python installed and **with** internet access.

1. On the online system go to a folder of your choice and create a “requirements.txt” file. The content of the file should list the required packages. A minimum for ibaVision with Python would be:

   ```
   numpy>=1.16
   ```

2. Then create a subfolder “wheelhouse” and run this command from a shell:

   ```
   pip download -r requirements.txt -d wheelhouse
   ```

3. Copy the “requirements.txt” file to the “wheelhouse” folder.
4. Pack the contents of "wheelhouse” into an archive (e.g., zip); then transfer the archive to the target machine (that has no internet access).

5. Extract the contents of the archive on the target machine.
6. From the folder where the “wheelhouse” folder is now located, run this command:

```
pip install -r wheelhouse\requirements.txt –no-index –find-links wheelhouse
```

7.2 Known issues

Using special characters, for example German umlauts, in *ibaVision's* Python plugin currently does not work.

If special characters are passed through the *ibaVision* configuration, they will mostly be displayed as ‘??’.

In the parameter declarations of Python scripts, special characters cause validation errors. Currently, the only workaround is to avoid special characters.

8 Creating a Python script

8.1 Introduction

The Python plug-in allows users to execute procedures written in the Python programming language, similar to how MVTec HALCON scripts work. For each Python script *ibaVision* will launch and handle a native CPython interpreter and take care of all inter-ops. Since the scripts have no dependencies on iba software, users are free to create, debug and test the scripts in their Python IDE of choice as standalone Python programs. Each script requires a special header to allow *ibaVision* to import scripts correctly and allow it to interact with the inputs/outputs parameters.

Requirements

- ibaVision >=3.0.0 x64 (the Python plugin is not supported by ibaVision 32-bit)
- Python 3.8.x with Numpy >=1.16 module installed

Installation

- Install Python 3.8 from a source of your choice (e.g. https://www.python.org)
- Install Numpy through the “pip” package manager by opening a command prompt and typing “pip install numpy”

Plug-in selection

Select the Python plug-in in the *Plugin settings* node of the *ibaVision* program, see chapter ↗ *Plugin settings*, page 29.

8.2 Python script tutorial

The *ibaVision* installation includes an example script. The example script can be found on GitHub: https://github.com/iba-ag/ibaVision-Example-Scripts

The listed code in this manual may not match the example from GitHub in all details. The script uses the OpenCV (https://opencv.org) library to rotate an input image (for example, from an ibaCapture video output) by an angle specified as a control input and then returns the rotated image as an image output as well as the average brightness of the image as a control output.

To use the example script in *ibaVision*, the user needs to have the OpenCV module for Python configured on his system. The easiest and fastest way to accomplish this is to use the pip package manager. Open a command prompt and type “pip install opencv-python”. The Python package manager will automatically install all required binaries and dependencies needed to use OpenCV in a Python script.

8.2.1 Script header

For *ibaVision* to be able to import the script procedure correctly, the user needs to write a short description of the input and output parameters at the beginning of every script.

```
import numpy as np

import cv2

IbaVisionInterfaceVersion = "1.0.0"

ProcedureDescriptions = [

    {"Description":("Main", "ExecuteMain", "Rotates the image by a specified angle and returns the rotated image and the average brightness(range 0 to 255) as ouputs."),
    "InputsControl" : [(0,"Input rotation angle", "The rotation angle")],
    "InputsIconic" : [(0,"Input image", "The unrotated input image")],
    "OutputsControl" : [(0,"Image brightness", "The average brightness of the image")],
    "OutputsIconic" : [(0,"Output image", "The rotated output image")]}
    ]

# Inputs
InputsIconic = {}
InputsIconic["Main"] = []
InputsIconic["Main"].append((128, 128, 3, np.ones([128,128,3], dtype='uint8'))) # Add a dummy 128x128 RGB image that allows the script to run standalone.

InputsControl = {}
InputsControl["Main"] = []
InputsControl["Main"].append(0) # Add a dummy control value that allows the script to run standalone.

# Outputs
OutputsIconic = {}
OutputsIconic["Main"] = []
OutputsIconic["Main"].append((128, 128, 3, np.ones([128,128,3], dtype='uint8'))) # Add a dummy 128x128 RGB image that allows the script to run standalone.

OutputsControl = {}
OutputsControl["Main"] = []
OutputsControl["Main"].append(0) # Add a dummy control value that allows the script to run standalone.

```

IbaVisionInterfaceVersion (string)
Specifies which version of the internal Python script parser *ibaVision* will use to read this script. Later versions of *ibaVision* might include breaking changes to the script specifications, so to avoid problems with backwards compatibility, you always need to specify an interface version for the script.

Current versions:

“1.0.0“ - Initial release

ProcedureDescriptions (List<Dict<string, object>>)
This dictionary contains the descriptions of the procedures (Name, info text, inputs and outputs) and needs to be configured correctly for each script. The following keys and values must always be included for each procedure:

“Description” - Tuple<string name, string functionName, string descriptionText>

This Python tuple contains the name of the procedure, the name of the function that will be called by *ibaVision* and a description text to display in the UI.

“InputsIconic” - List<Tuple<int index, string name, string descriptionText>>

This list contains tuples that describe the image inputs. The index corresponds to the position of this input in the list. The name and description text can be chosen freely.

“InputsControl” - List<Tuple<int index, string name, string descriptionText>>

This list contains tuples that describe the control inputs. The index corresponds to the position of this input in the list. The name and description text can be chosen freely.

“OutputsControl” - List<Tuple<int index, string name, string descriptionText>>

This list contains tuples that describe the control outputs. The index corresponds to the position of this input in the list. The name and description text can be chosen freely.

“OutputsIconic” - List<Tuple<int index, string name, string descriptionText>>

This list contains tuples that describe the image outputs. The index corresponds to the position of this input in the list. The name and description text can be chosen freely.

InputsControl (List<object>)
Contains the control inputs from *ibaVision*. The inputs are automatically updated by *ibaVision* for each new frame before the script procedure is called. Python types that can be set by *ibaVision* are

int – Converted from 8-Bit, 16-Bit, 32-Bit and 64-Bit signed and unsigned integral types

string – Converted from Technostrings

float – Converted from single and double precision floating point numbers

InputsIconic (Dict<string procedureName, List<Tuple<int width, int height, int numChannels, numpy.ndarray pixelData>>>)
Contains the image inputs provided by *ibaVision*. The inputs are automatically updated by *ibaVision* for each new frame before the script procedure is called. The index of each image corresponds to the index defined in the description for this parameter. The images are provided as a 3D Numpy array with the shape (width, height, numChannels) where width and height are the width and height of the image and numChannels is the number of channels (e.g. 3 for an RGB image or 1 for a grayscale image). The data type must be “uint8”, i. e. an unsigned byte, which results in 8 bits per pixel color depth for a grayscale image and 24 bit per pixel color depth for an RGB image.

OutputsControl (List<object>)
Contains the control outputs that will be retrieved by *ibaVision* after each time that the script procedure has been executed. Supported Python types are

int – Will be converted to a 64-Bit signed integer internally

string – Will be converted to a string internally

float – Will be converted to a double precision floating point number internally

OutputsIconic Dict<string procedureName, List<Tuple<int width, int height, int numChannels, numpy.ndarray pixelData>>>)
Contains the image outputs that must be set by the script procedure during each execution and before returning control back to *ibaVision*. The index of each image corresponds to the index defined in the description for this parameter. The images must be a 3D Numpy array with the shape (width, height, numChannels) where width and height are the width and height of the image and numChannels is the number of channels (e.g. 3 for an RGB image or 1 for a grayscale image). The data type must be “uint8”, i.e. an unsigned byte, which results in 8 bits per pixel color depth for a grayscale image and 24 bit per pixel color depth for an RGB image.

8.2.2 Script procedure

```
1.   def ExecuteMain():
2.
3.       # Unpack input image in index 0 of iconic inputs
4.        # (width, height, numChannels, pixelData)
5.        width = InputsIconic["Main"][0][0]
6.        height = InputsIconic["Main"][0][1]
7.        numChannels = InputsIconic["Main"][0][2]
8.        pixelData = InputsIconic["Main"][0][3]
9.
10.       # Get input parameter in index 0 of control inputs
11.       rotationAngleInDegrees = InputsControl["Main"][0]
12.
13.       # Calculate average brightness of the image
14.       averageBrightness = np.sum(pixelData) / pixelData.size
15.
16.       # Rotate input image
17.       rotM = cv2.getRotationMatrix2D((width / 2, height / 2), rotationAngleInDegrees, 1)
18.       outputPixelData = cv2.warpAffine(pixelData, rotM, (width, height))
19.
20.       # Set rotated output image
21.       # (width, height, numChannels, pixelData)
22.       OutputsIconic["Main"][0] = (width, height, numChannels, outputPixelData)
23.
24.       # Set output control
25.       OutputsControl["Main"][0] = int(averageBrightness)
26.
```

ExecuteMain
This is the entry point for the procedure and that we have specified earlier in the header. *ibaVision* will execute this function for each frame.

Lines 5-11 unpack the input parameters into separate variables to improve readability.

Lines 11-18 calculate the average brightness of the image and use the OpenCV library to rotate it.

Lines 22-25 set the outputs for *ibaVision* to retrieve.

9 Support and contact

Support

Phone: +49 911 97282-14

Fax: +49 911 97282-33

Email: support@iba-ag.com

Note

If you need support for software products, please state the license number or the CodeMeter container number (WIBU dongle). For hardware products, please have the serial number of the device ready.

Contact

Headquarters

iba AG
Koenigswarterstrasse 44
90762 Fuerth
Germany

Phone: +49 911 97282-0

Fax: +49 911 97282-33

Email: iba@iba-ag.com

Mailing address

iba AG
Postbox 1828
D-90708 Fuerth, Germany

Delivery address

iba AG
Gebhardtstrasse 10
90762 Fuerth, Germany

Regional and Worldwide

For contact data of your regional iba office or representative please refer to our web site

www.iba-ag.com.