ROBOTICS

Product manual

FlexLoader Vision

Trace back information:
Workspace Robots and Applications version a67
Checked in 2020-12-03
Skribenta version 5.3.075

Product manual
FlexLoader Vision

R20C

Document ID: 3HAC051771-001

Revision: F

The information in this manual is subject to change without notice and should not be construed as a commitment by ABB. ABB assumes no responsibility for any errors that may appear in this manual.

Except as may be expressly stated anywhere in this manual, nothing herein shall be construed as any kind of guarantee or warranty by ABB for losses, damage to persons or property, fitness for a specific purpose or the like.

In no event shall ABB be liable for incidental or consequential damages arising from use of this manual and products described herein.

This manual and parts thereof must not be reproduced or copied without ABB's written permission.

Keep for future reference.

Additional copies of this manual may be obtained from ABB.

Original instructions.

Table of contents

Overview of this manual 9
Network security 11

1 Getting started 13

2 What is FlexLoader Vision? 15

3 Overview of the user interface 17

3.1 Menu 17
3.2 Tool box 19
3.3 Alarm 20
3.4 System information 21

4 Teachin 23

4.1 Introduction 23
4.2 Navigation 24
4.3 Detail 25
4.4 Image 29
4.4.1 Image settings for 2D cameras 29
4.4.2 Image settings for 3D cameras 31
4.5 Positions 36
4.6 Parameters 38
4.7 Tools 48
4.8 Grip 54
4.9 Supervision 61
4.10 Mechanics 64
4.11 Test & Save 68

5 Advanced teachin settings 71

5.1 Advanced image settings 71
5.2 Advanced position settings 75
5.3 Advanced parameter settings 76
5.3.1 Standard analysis 76
5.3.2 Blob analysis 83
5.3.3 3D analysis 85
5.3.4 AddOn analysis 87
5.4 Advanced grip settings 93
5.5 Advanced supervision settings 98

6 Operation 101

6.1 Introduction 101
6.2 Starting the system 102
6.3 Stopping the system 103
6.4 Operation information 104
6.5 SmartPhone connection 106

7 Camera 109

7.1 2D camera settings 109
7.2 3D camera settings 115

8 Settings 119

8.1 Organizer 119
8.2 Mechanics 121
8.3 Service 124
8.4 Gripping Limitations 127

8.5 Master/Slave 131

9 Safety, SCADA and Browser 135

10 Troubleshooting 137

10.1 Problems with gripping accuracy 137
10.2 Identification problems 140
10.3 Communication problems 141
10.4 Camera problems 142

11 Installation 143

11.1 PC installation 143
11.2 Basler camera mechanical and electrical installation 147
11.3 Cirrus 3D camera mechanical and electrical installation 150
11.3.1 Mechanical installation 150
11.3.2 Electrical installation 152
11.3.3 Field of view geometry 153

12 Commissioning 157

12.1 Configuration editor 157
12.1.1 Introduction 157
12.1.2 Main settings 158
12.1.3 Camera settings 160
12.1.4 Robot settings 166
12.1.5 Tool settings 169
12.1.6 Configure 171
12.2 User accounts 173
12.3 Network setup 174
12.4 Robot system 183
12.5 User-defined mechanical equipment 186
12.6 SCADA and Safety 192
12.6.1 Introduction 192
12.6.2 Configuration 195
12.7 Other system settings 206

13 Maintenance 209

13.1 General maintenance 209
13.2 Backup and recovery of the hard disk 211
13.3 Upgrading FlexLoader Vision 219

14 Robot integration 221

14.1 Introduction 221
14.2 Calibrating the robot 222
14.3 RAPID program 230
14.3.1 Overview 230
14.3.2 FlexLoader application functionality 231
14.3.3 FlexLoader Vision interface 241
14.3.4 FlexLoader Vision Lite functionality 243
14.3.5 FlexLoader Library Add-in 245
14.3.6 FlexLoader assistance and utility functionality 252
14.3.7 FlexLoader machine tool interface functionality 255
14.3.8 WorldZones 257
14.4 Communication information 258
14.4.1 Communication between FlexLoader Vision and the robot 258
14.4.2 Communication between the robot and the PLC 262
14.4.3 Communication between Master and Slave 263

15 FlexLoader Vision Lite 267
15.1 Introduction 267
15.2 Operation 270
15.3 FlexLoader Vision Lite teachin 273
15.3.1 Detail 274
15.3.2 Load 275
15.3.3 Unload 277
15.3.4 Options 281
15.4 Commissioning 284
15.4.1 Robot 284
15.4.2 FlexLoader Vision Lite master detail 289
15.5 FlexLoader Vision Lite configuration 290
A Technical information 299
A.1 File structure in FlexLoader Vision 299
A.2 Logging the operation 301
A.3 Localization 303
A.4 Manually restoring a backup 304
B FlexLoader RAPID reference 305
B.1 FlexLoader data type prefix 305
B.2 FlexLoader Vision interface 306
B.3 FlexLoader application functionality 313
B.4 FlexLoader Vision Lite functionality 320
B.5 FlexLoader conveyor system control 321
B.6 FlexLoader assistance and utility functionality 322
B.7 FlexLoader machine tool interface functionality 329
C FlexLoader Library Add-in reference 333
C.1 Instructions 333
C.1.1 MT_ClearAllPartInfo - Clears all part tracker information 333
C.1.2 MT_ClearPartInfo - Resets part information of a part tracker 334
C.1.3 MT_CopyPartInfo - Copies part info between trackers 335
C.1.4 MT_CreateAlarm - Creates new alarm or message 337
C.1.5 MT_ErrWrite - Creates error log message in local language 338
C.1.6 MT_GetPartInfo - Gets any kind of part tracker data from a part tracker 339
C.1.7 MT_GoHome - Moves robot to home zone 341
C.1.8 MT_InitializeMessageHandling - Initiates the alarms and messaging module 342
C.1.9 MT_Log - Creates a log message in log file 343
C.1.10 MT_LogEnable - Enables or disables logging 344
C.1.11 MT_MovePartInfo - Moves part info between part trackers 345
C.1.12 MT_MoveRobotTo - Moves robot to a specific zone 347
C.1.13 MT_MoveToStart - Moves robot to closest start position 348
C.1.14 MT_MoveToZoneMenu - Shows manual menu to move between zones 349
C.1.15 MT_ResetAlarms - Resets all or a specific alarm 350
C.1.16 MT_SetAlarm - Sets an alarm to active 351
C.1.17 MT_SetPartInfo - Sets part information of a part in a specific tracker 352
C.1.18 MT_SetPartState - Set state of a part in a part tracker 353
C.1.19 MT_UpdateMessageTextsIfNeeded - Reads messages into message array if needed 354
C.2 Functions 355
C.2.1 MT_GetActiveAlarm - Get an active alarm data 355
C.2.2 MT_GetCurrentZone - Get zone where robot is positioned now 356
C.2.3 MT_GetMessage - Get complete message data 357
C.2.4 MT_GetNumberOfActiveAlarms - Gets number of active alarms 358
C.2.5 MT_GetPartState - Gets current part state of a part in specific tracker 359
C.2.6 MT_GetText - Gets a selected text line from a message 360
C.2.7 MT_GetViaPosition - Gets corresponding via position for a zone 362

C.2.8 MT_IsAlarmActive - Check if an specific alarm is active 363
C.2.9 MT_UIBoolEntry - Show an yes no question in localized language 364
C.2.10 MT_UIDNumEntry - Shows a localized dnum entry 368
C.2.11 MT_UIMessageBox - Show a message box in localized language 372
C.2.12 MT_UINumEntry - Shows a num entry box in localized language 376
C.3 Data types 380
C.3.1 mtaxisrange - specifies an axis with an axis range 380
C.3.2 mtloglevel - define a log level 381
C.3.3 mtmsgdata - alarm and message definition data 382
C.3.4 mtpartdata - Describes a part 384
C.3.5 mtpartdeviation - Description 385
C.3.6 mtpartdimension - Description 386
C.3.7 mtpartproperties - Description 387
C.3.8 mtpartstate - Describes a part state 388
C.3.9 mtparttracker - describes all data for a part tracker 389
C.3.10 mtrange - describes a robot axis range 390
C.3.11 mtstationdata - describes a station 391
C.3.12 mttargetdata - describes a robot zone 392

Overview of this manual

About this manual

FlexLoader Vision is a vision system designed to guide industrial robots during materials handling. This manual describes FlexLoader Vision, how to set up, calibrate, operate, troubleshoot, backup and recover the system. This manual also describes how the ABB robot works together with FlexLoader Vision and how to integrate the robot with the vision system. Finally, this manual describes how to configure and operate FlexLoader Vision Lite. FlexLoader Vision Lite is a simplified and limited, but very efficient and user-friendly version of FlexLoader Vision, designed specifically for lathe applications.

Organization of this manual

This manual basically consists of three different parts.

Chapter 1 is a quick start guide describing the first steps when using FlexLoader Vision.

Chapter 2-12 describe how to set up, calibrate, operate and troubleshoot FlexLoader Vision. Here you can also find technical information and information on communication, configuration, backup and recovery and how to set up the configuration editor of FlexLoader Vision.

Chapter 13 describes the interaction of FlexLoader Vision with the robot and the RAPID program structure.

Chapter 14 describes how to use and configure the simplified version of FlexLoader Vision, FlexLoader Vision Lite, that is specially developed for lathe applications.

The appendix gives more detailed information on communication, backup recovery and RAPID reference.

Usage

This manual should be used during:

- installation and robot integration
- calibration of FlexLoader Vision
- detail teachin
- operation
- backup and recovery
- troubleshooting.

Who should read this manual?

This manual is intended for:

- installation personnel
- FlexLoader Vision users
- FlexLoader Vision integrators
- PLC programmers
- service technicians.

Continues on next page

Prerequisites

The reader should have basic knowledge of:

- robot programming
- Microsoft Windows systems
- PLC programming
- mechanical and electrical installation work.

Trademarks

FlexLoader is a trademark of ABB.

References

- The product manual of the robot used together with FlexLoader Vision
- The programming manual of the robot used together with FlexLoader Vision
- The product manual of the camera/s used together with FlexLoader Vision
- The integration manual of the camera/s used together with FlexLoader Vision

Revisions

Revision	Description
A	First edition.
B	Complete revision with content and illustration update.
C	Clarifications on PROFINET setup
D	Updated RAPID documentation with regard to the FlexLoader Library Add-in. Minor updates related to new FlexLoader Vision functionality.
E	Updated for R20A. Improvements for SCADA and VisionMaster functionality as well as minor updates.

Network security

Network security

This product is designed to be connected to and to communicate information and data via a network interface. It is your sole responsibility to provide, and continuously ensure, a secure connection between the product and to your network or any other network (as the case may be).

You shall establish and maintain any appropriate measures (such as, but not limited to, the installation of firewalls, application of authentication measures, encryption of data, installation of anti-virus programs, etc) to protect the product, the network, its system and the interface against any kind of security breaches, unauthorized access, interference, intrusion, leakage and/or theft of data or information. ABB Ltd and its entities are not liable for damage and/or loss related to such security breaches, any unauthorized access, interference, intrusion, leakage and/or theft of data or information.

This page is intentionally left blank

1 Getting started

Basic steps for integrators

Follow the steps below for getting started with a FlexLoader Vision system.

	Action
1	Become familiar with this manual.
2	Install PC and camera. See *Installation on page 143*.
3	If necessary, make appropriate configuration changes. See *Configuration editor on page 157*.
4	Use the **Camera** -> **Live** window to adjust the camera (2D: focus, aperture; 3D: image acquisition parameters) for good contrast and visibility of part features. See *Camera on page 109*.
5	If necessary, copy and load all necessary RAPID code on the robot. See *Robot system on page 183*.
6	Calibrate all camera work objects. See *Calibration on page 110*, *Calibration on page 115* and *Calibrating the robot on page 222*.
7	Use a simple detail and perform a basic teachin. See *Teachin on page 23*.
8	Adjust the robot program to fit the needs of the robot cell. See *RAPID program on page 230* and *FlexLoader RAPID reference on page 305*
9	Perform a first pick with the robot at low speed. See *Operation on page 101*.
10	Make final teachin and refinements to to robot code.

This page is intentionally left blank

2 What is FlexLoader Vision?

FlexLoader Vision is a vision system that guides ABB industrial robots during materials handling with outstanding performance. It is a proven system offering high speed, short teachin times, reliable production and a user-friendly interface.

ABB offers a software that identifies the workpiece location and orientation from the picking area and guides the robot with precision in the robot cell. The FlexLoader Vision can handle an unlimited number of workpieces in various sizes and with complex geometries without any need for mechanical fixtures, thus reducing costs and complexity.

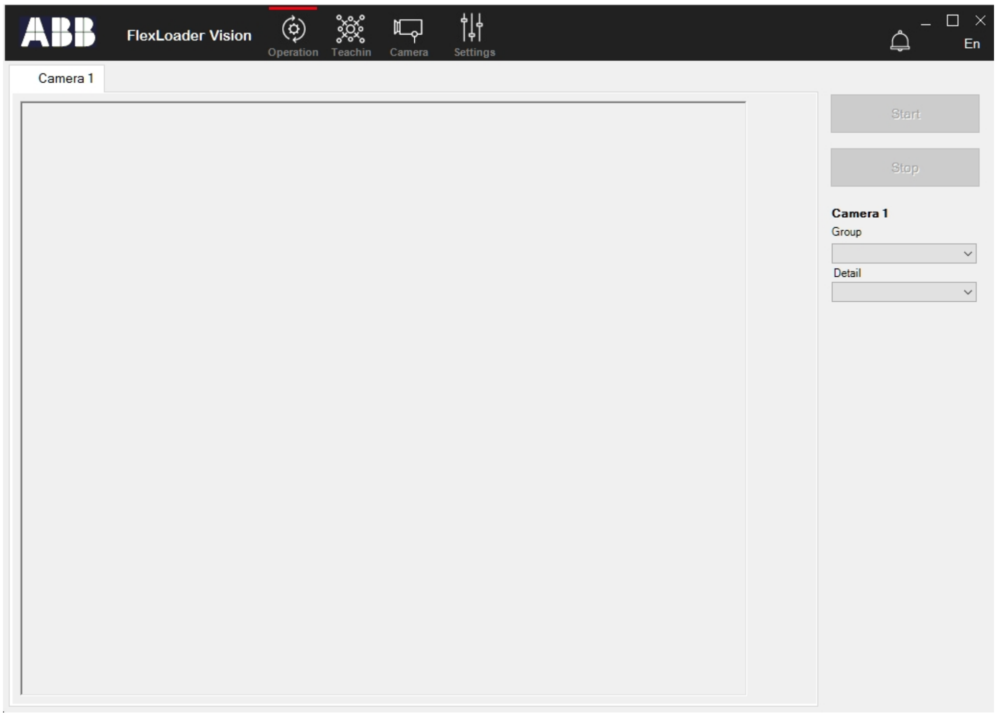
The FlexLoader Vision is open to communicate with a variety of camera sensors. It can be used for both 2D and semi-oriented 3D applications.

This page is intentionally left blank

3 Overview of the user interface

3.1 Menu

When you start FlexLoader Vision you will see the following window.



xx1800000203

At the top of the FlexLoader Vision window is a menu showing a number of menu icons that represent various system activities.

	Operation	Used to start and stop the system. Also shows information on the current operation.
	Teachin	Used to teach the system how to handle details.
	Lite	Used to perform the detail teachin in FlexLoader Vision Lite (option). Note that the **Lite** tab is not available in the standard edition.
	Camera	Used to calibrate the system and make camera related settings.
	Settings	Used to configure, service and maintain the system.
	Safety	Used to display safety information. (if configured by integrator)

Continues on next page

	SCADA	Used to display cell information. (if configured by integrator)
🌐	**Browser**	Used to display a web page. (if configured by integrator)
🔴	**Alarms**	Used to display the alarm overview.

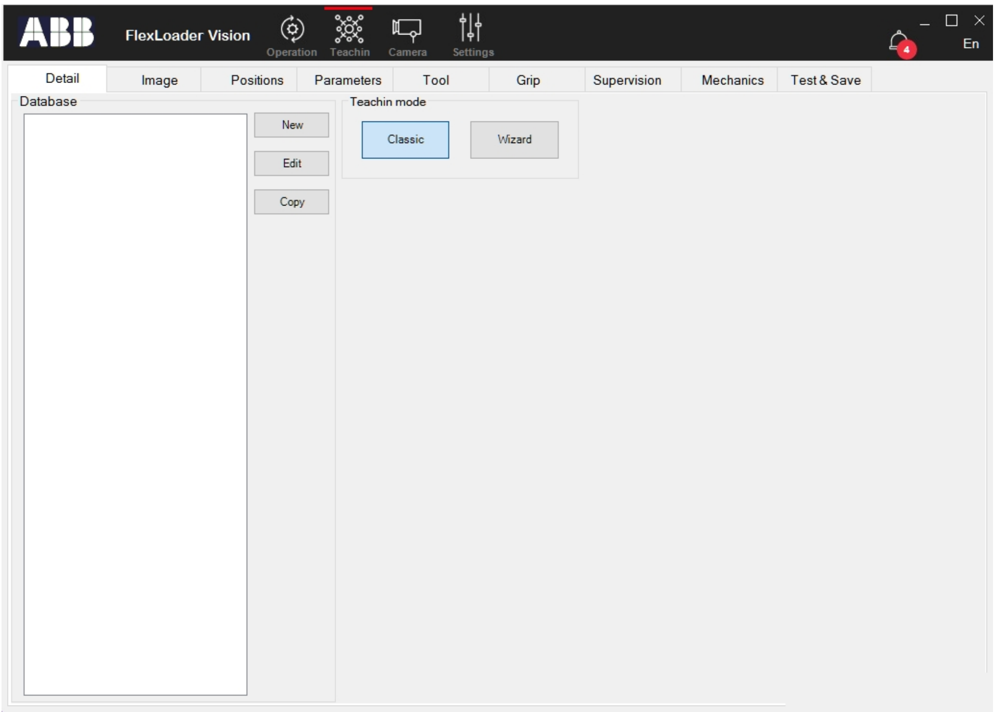
3.2 Tool box

FlexLoader Vision displays images at many different locations. To the right of each image there is a tool box. Not all the tools are displayed for every image – only the tools that can be used for the concerned operation are displayed. When a tool is selected, the button is highlighted and the tool is used when you click somewhere in the image. The functions of each of the icons in the tool box are described below.

🔍	Reset zoom	Used to zoom in or out of an image and view the image at normal scale again.
⊕	Zoom in	Used to zoom in on an image.
⊖	Zoom out	Used to zoom out of an image.
✋	Pan	If you have zoomed in on an image you can use this tool to move around the image. Click the mouse button and drag the mouse to pan. Then release the mouse button.
⌗	Define position Define region	Used to define a position, area or gripper limit. Click the mouse button, drag the mouse diagonally to draw a box of the desired size, then release the mouse button. Also used to define the region of interest for a 3D camera.
⌖	Define grip	Used to define a grip. Click the mouse button to place the grip point at the current mouse position.
✏	Draw	Used to draw. Click the mouse button to draw a point at the current mouse position. Release the mouse button to stop drawing. To change the drawing size, adjust the slider as required.
⌫	Clear	Used to delete objects. Click the mouse button to delete the drawing at the current position. Release the mouse button to stop deleting. To change the delete size, adjust the slider as required.
╲	Draw line	Used to draw lines. Click the mouse button where you want the line to start, drag the mouse to the point where you want the line to end, then release the button.
□	Draw box	Used to draw boxes. Click the mouse button where you want the box to start, drag the mouse to the point where you want the box to end, then release the button.
○	Draw circle	Used to draw circles. Click the mouse button where you want the circle to start, drag the mouse to the point where you want the circle to end, then release the button.
↶	Undo	Used to undo the last drawing action.
┤	Brush size	Used to changing the brush size for drawing. The brush size is displayed as a ring and the size can be adjusted with the mouse scroll wheel.

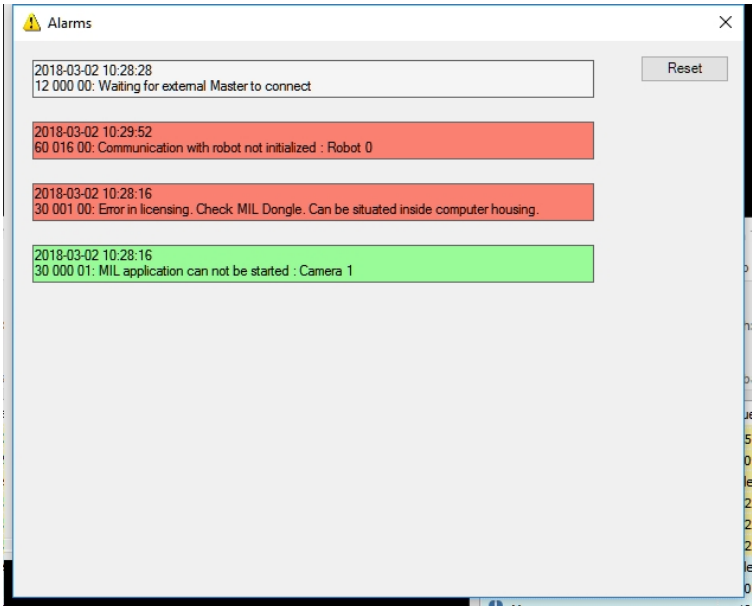
3.3 Alarm

The system continuously carries out tests to check that all the units are working. If a unit is not working it triggers an alarm that appears as a notification at the top of the window, in the menu bar. It is always the most recent alarm that is displayed.



xx1800000204

Click on the alarm notification to open a list of all active alarms and show more information about what has caused the alarms.



2018-03-02 10:28:28
12 000 00: Waiting for external Master to connect

2018-03-02 10:29:52
60 016 00: Communication with robot not initialized : Robot 0

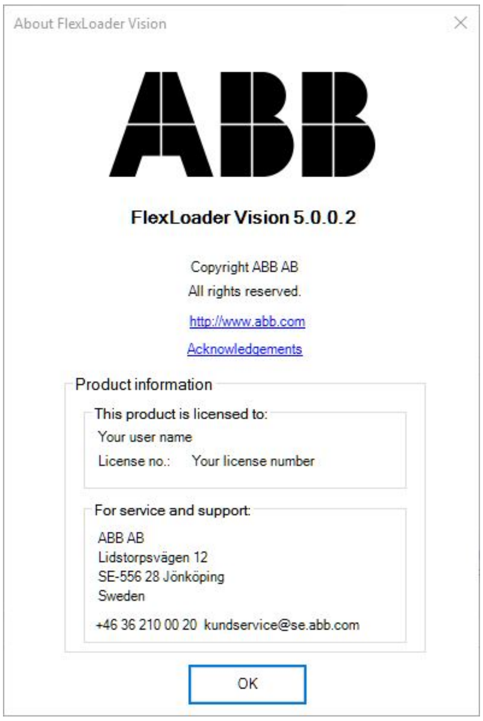
2018-03-02 10:28:16
30 001 00: Error in licensing. Check MIL Dongle. Can be situated inside computer housing.

2018-03-02 10:28:16
30 000 01: MIL application can not be started : Camera 1

xx1800000205

3.4 System information

The below window appears when you click on the ABB logotype at the top left of the screen. It contains information about the current version of software, license information and contact information for service and support.



About FlexLoader Vision

FlexLoader Vision 5.0.0.2

Copyright ABB AB
All rights reserved.

http://www.abb.com

Acknowledgements

Product information

This product is licensed to:
Your user name
License no.: Your license number

For service and support:

ABB AB
Lidstorpsvägen 12
SE-556 28 Jönköping
Sweden

+46 36 210 00 20 kundservice@se.abb.com

xx1800000206

This page is intentionally left blank

4 Teachin

4.1 Introduction

Before the system can be used it must be taught about the details it will handle. During the so called teachin process you tell the system what kind of detail it is to look for and how the robot is to grip or handle the identified details, as well as how the feeding equipment is to be set. The teachin process is divided into a number of stages that you will find on different tabs in the **Teachin** window: **Detail**, **Image**, **Positions**, **Parameters**, **Tool**, **Grip**, **Supervision**, **Mechanics** and **Test & Save**.

The settings available differ depending on how FlexLoader Vision is configured, for example depending on whether you use a 2D or a 3D camera.

For information on the advanced settings on each tab, see *Advanced teachin settings on page 71*.

4.2 Navigation

At the bottom right of each tab there are two buttons: << **Back** and **Next** >>. These allow you to move quickly between the different tabs. To change something in the previous tab, press the << **Back** button.

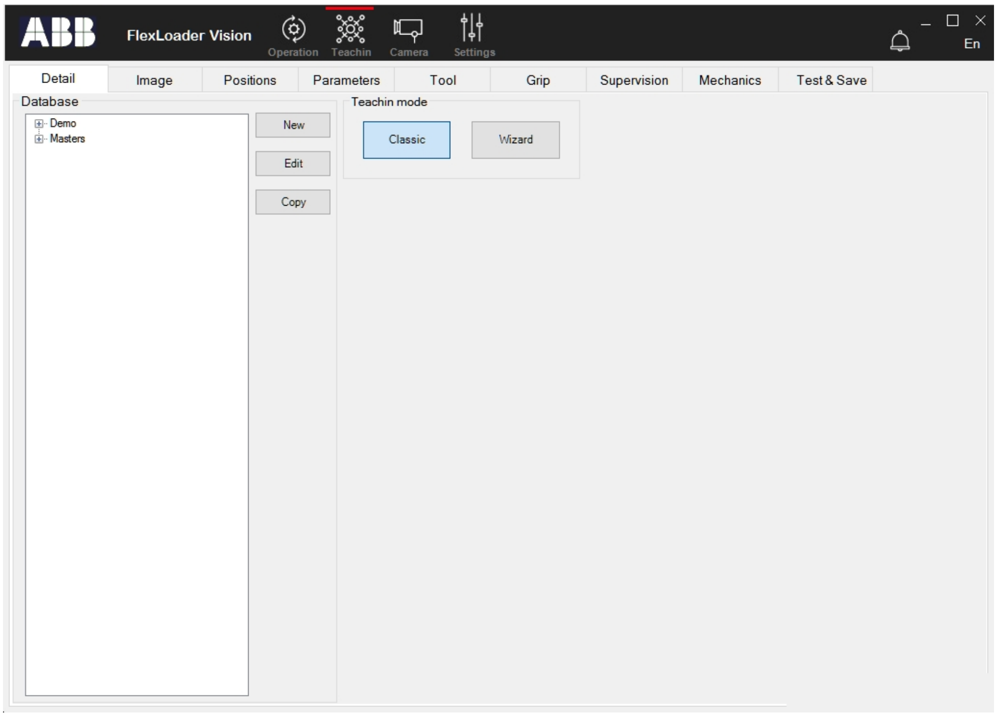
You can also get extra help during the teachin process: If you click the **Wizard** button, help text will guide you through the teachin process step by step in a preset order.

In **Classic** mode the help text is not shown, and you can move freely between the different tabs in the teachin process.

4.3 Detail

The first tab on the **Teachin** menu is **Detail.**

All the details that the system has been taught are saved in a database. The database is divided into various groups. Each group can contain details. An unlimited number of groups and details can be saved in the database.



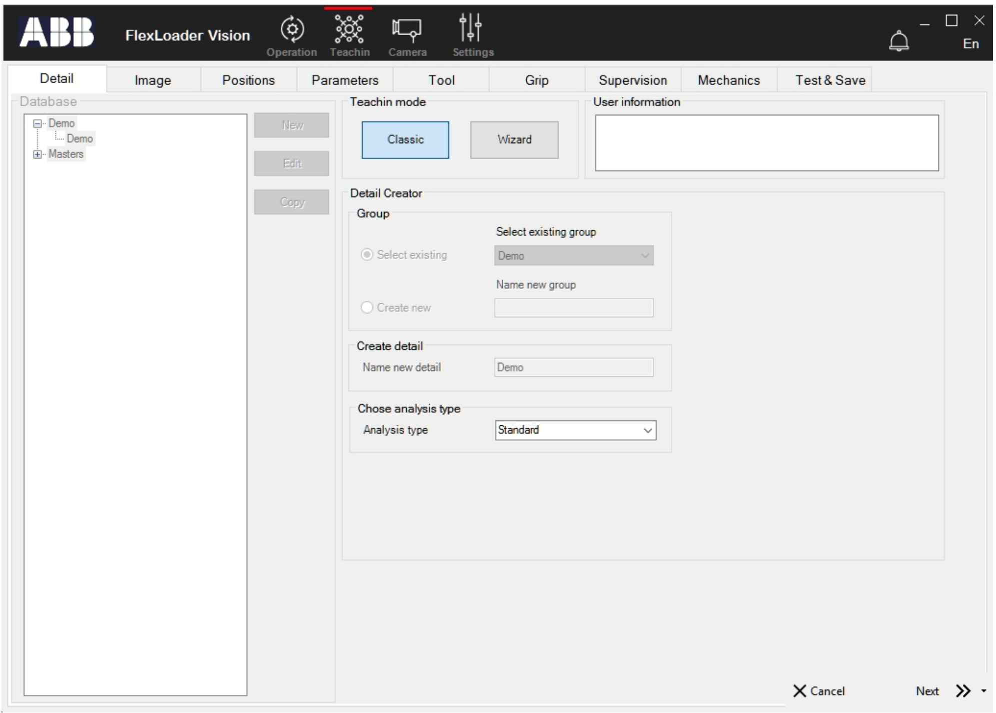
xx1800000207

Continues on next page

To the left is a pane called **Database**. This shows the various groups in the database. Clicking on the plus sign (**+**) in front of a group expands that group so you can see the details in the group.

Creating a new detail

Click **New** to create a new detail in the database.



Select existing group: Demo
○ Select existing
Name new group:
○ Create new
Name new detail: Demo
Analysis type: Standard

xx1800000208

Here you can choose if you want the new detail to be saved in an existing group, or if you want to create a new group. The detail should also be given a unique name. Several details can share the same name, but they must all be placed in different groups.

If the system has more than one camera you will also see a box headed **Select Camera**. This is where you choose which camera to use in the teachin process. This box will not appear if there is only one camera connected to the system.

Selecting analysis type

This drop-down list only appears if blob searching has been enabled for the system. It lets you choose between the two search methods **Standard** and **Blob**. In FlexLoader Vision, so called blobs are details that may sometimes, within specified ranges, vary in shape.

Editing a detail

To modify a detail, first select it and then click on **Edit**. Double-clicking on the detail in the database box also selects the detail for editing.

Continues on next page

Copying a detail

Very often a system will handle many types of details that are similar. The **Copy** button can be used to quickly create a copy of a detail that can be modified when teaching the system to use a similar detail. This lets you reuse many settings that are identical.

To copy a detail, select the detail in the database pane and click **Copy**.

Enter a new group for the detail, or leave as suggested (current group). Enter a new name for the detail, or leave as suggested (current name).

SmartPhone connection

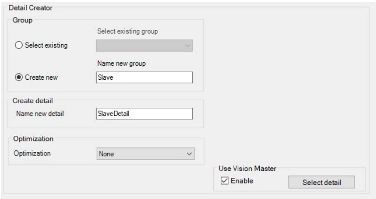
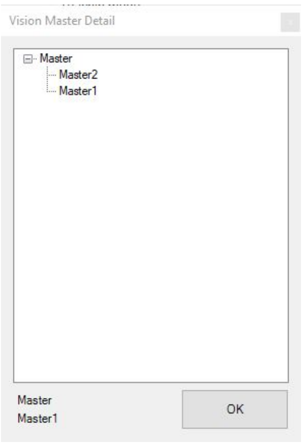
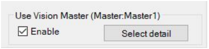
For SmartPhone connected systems you can enable data reporting and enter the expected cycle time for the detail (see *Smartphone connection on page 207*).

VisionMaster functionality

Some applications use the same teachin for several parts, but with different user-defined parameters. Examples are raw parts with common features but minor differences that are handled by the user-defined parameters.

For these parts, the teachin can rely on a vision master that defines the image related settings, e.g. camera settings, positions, grippers, and gripping points.

Only parameters visible on the **Mechanics** tab are accessible to the user.

	Action	Note
1	Create the detail that will act as vision master.	
2	Create a new part and enable the vision master functionality by selecting the **Enable** check box.	 Detail Creator Group ○ Select existing — Select existing group: ● Create new — Name new group: Slave Create detail Name new detail: SlaveDetail Optimization Optimization: None Use Vision Master ☑ Enable
3	Select a master detail by first clicking **Select detail**, and then selecting a master detail by double-clicking. Confirm by clicking **Ok**.	 Vision Master Detail ⊟ Master Master2 Master1 Master Master1
4	The selected master detail is now displayed for information.	 Use Vision Master (Master:Master1) ☑ Enable

The use of the VisionMaster functionality must be enabled manually.

Continues on next page

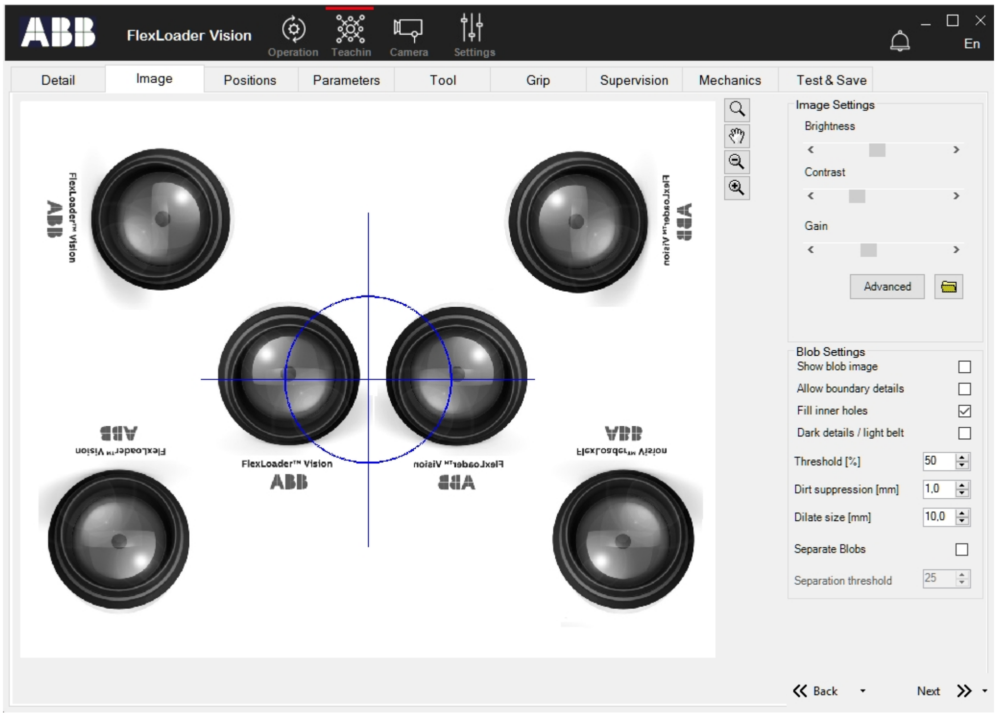
See *VisionMaster functionality on page 208*.

4.4 Image

4.4.1 Image settings for 2D cameras

Image settings

For 2D systems, the image can be adapted by adjusting brightness, contrast and gain. The effect on the image is seen immediately.



Show blob image □
Allow boundary details □
Fill inner holes ☑
Dark details / light belt □
Threshold [%] 50
Dirt suppression [mm] 1,0
Dilate size [mm] 10,0
Separate Blobs □
Separation threshold 25

xx1800000209

Fast and reliable identification of teachin details requires the clearest possible contrast between the background and the detail. If the contrast between background and detail is poor this will result in reduced system performance. Place a number of details in different positions underneath the camera and adjust the camera settings so that the contrast is as clear as possible.

Blob settings

To obtain the search image for so called blobs (details that may sometimes, within specified ranges, vary in shape), the image is divided into zones that are considered detail (marked as black) and zones where the background can be seen (white). To activate blob image display, select **Show blob image**.

Selecting **Allow boundary details** makes it possible to pick details even if they are touching the image edge. As this can lead to collisions, this setting is disabled by default.

Fill inner holes is selected by default. For details with important hole patterns, it may be necessary to deselect this box.

Continues on next page

Dark details/light belt can be selected to indicate if the conveyor is light in color and the details dark, or vice versa. The resulting search image is updated continuously to permit rapid checking.

With the **Threshold** value you can adjust the details to appear as black as possible and the background as white as possible.

With the **Dirt suppression** setting you can filter out individual black marks on the conveyor, as these are often just dirt and nothing else.

By specifying a **Dilate size** value, details with irregular outer contours are slightly smoothed out.

There are occurrences where some details are placed on top of each other and are considered as one big blob. It might be desirable to split these large blobs for further analysis. To do so, select **Separate blobs** and set a threshold value for this. Preferably add a number of details with varying overlaps under the camera. The threshold must be set so that the split is performed correctly, meaning that details with the permitted overlaps are split, but details with too large overlaps are not split.

4.4.2 Image settings for 3D cameras

Image settings

The image settings for 3D cameras are found on the **Image** tab on the **Teachin** menu.

The parameters **Analysis depth** and **Seek depth** determine which height area, within the scanner range, should be used to evaluate an image.

For some 3D cameras it takes a long time to obtain a new image, so FlexLoader Vision would be difficult to work with if the camera took new pictures the whole time. Therefore the 3D images are not automatically updated. Use the **New image** button or short cut **Ctrl+Shift** to take a new image.

The 3D camera does not need settings for brightness, contrast and gain (like the 2D camera does), so these settings are not visible on the 3D camera user interface. The grey scale in the image below represents height, and lighter colors mean that the object is higher up.

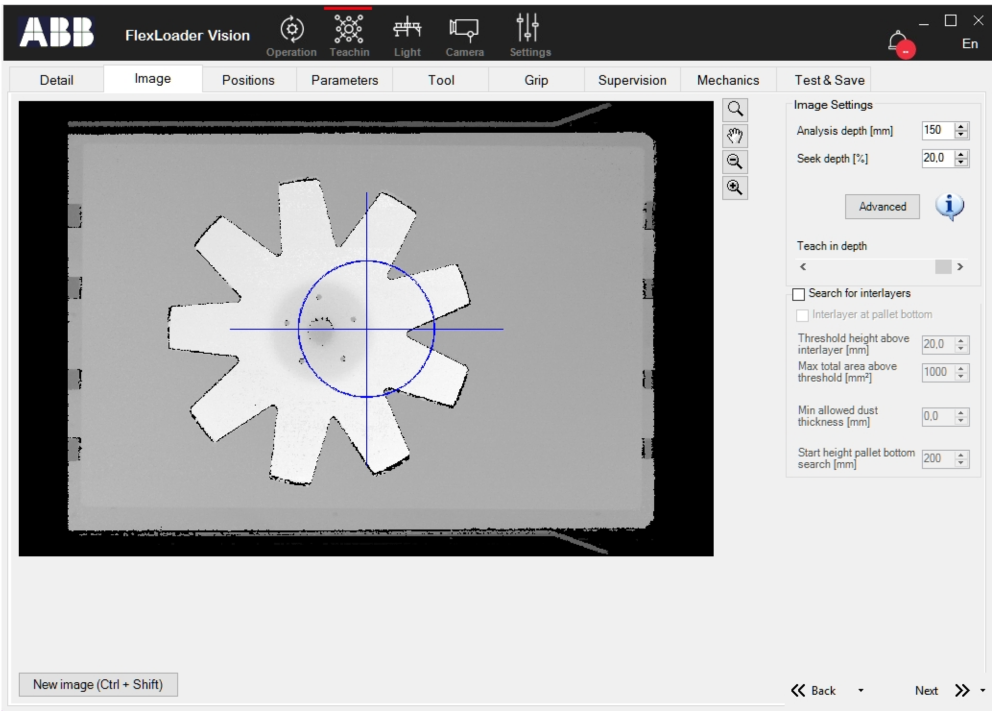


Image Settings

Analysis depth [mm] 150

Seek depth [%] 20,0

Teach in depth

□ Search for interlayers

□ Interlayer at pallet bottom

Threshold height above interlayer [mm] 20,0

Max total area above threshold [mm²] 1000

Min allowed dust thickness [mm] 0,0

Start height pallet bottom search [mm] 200

xx1800000210

Analysis depth

The analysis depth indicates how far down FlexLoader Vision is to search for details from the highest located point. This dimension directly affects what is visible in the image in FlexLoader Vision. The highest point in the image is white and then it gets darker, down to black, at the specified analysis depth. The upper edge of the pallet collar can lie at a greater height than the highest located point and is therefore also completely white.

The highest point in the image is determined within the cameras region of interest, see *Region of interest on page 118*.

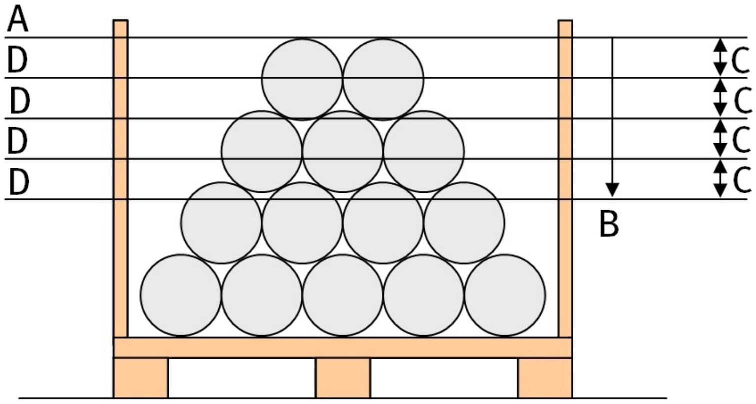
Continues on next page

The highest point also sets an upper limit for the allowed height of details. All details more than 50 mm above the highest point will be discarded.

Seek depth

With this parameter the analysis depth can be split up. With detail identification FlexLoader Vision searches from above at several heights until a part has been found or the whole analysis depth has been analyzed. The analysis depth is thereby layered from above and down for best possible search results.

The stated **Seek depth** (percentage of the analysis depth) represents the first search level for FlexLoader Vision. If no detail is found on the first search level, the next level is added and a new analysis carried out. This is repeated until a detail has been identified or the whole analysis depth has been analyzed. Note that the number of search levels depends on the specified seek depth. Do not use an unnecessary low percentage, because that will require more analysis time for FlexLoader Vision.



xx1800000300

Pos.	Description
A	Highest located point
B	Analysis depth
C	Seek depth
D	Search level

Figure 4.1: Illustration of seek depth and analysis depth

Teachin depth

After taking a picture, the teachin depth can be adjusted with a live update in the image.

Continues on next page

The teachin depth determines at which depth, that is, at which level, the image is captured when teaching parts. The teachin depth can never be greater than the value of the selected analysis depth.

Tip

It can be very useful to adjust the teachin depth to get clearer edges if there is only a minor height difference.

Use custom Acquisition ID

In most cases, the 3D sensor will be capable of imaging parts with one single setting of acquisition parameters.

This behavior can be changed, which enables the operator to select customized acquisition parameters as defined in the 3D sensor.

If enabled in the sensor settings, the user can select the acquisition ID to be default (as specified in the sensor configuration) or to be customized (by entering the desired acquisition ID value.

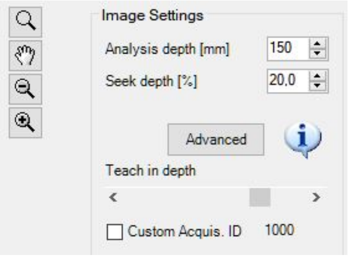


Image Settings

Analysis depth [mm] 150

Seek depth [%] 20,0

□ Custom Acquis. ID 1000

xx1900001041

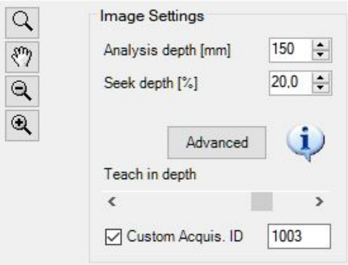


Image Settings

Analysis depth [mm] 150

Seek depth [%] 20,0

☑ Custom Acquis. ID 1003

xx1900001042

Interlayer search

If the function to search for interlayers is activated, FlexLoader Vision will look for interlayers when no details are identified. When an interlayer is identified, FlexLoader Vision will send information to the robot that an interlayer is found and at what height it is (flagged as PICK_INTERLAYER).

An interlayer is here defined as a sufficiently flat surface as defined below within the cameras region of interest, see *Region of interest on page 118*.

Continues on next page

Several checks are performed to ensure that an interlayer can be picked. Some parameters can be adjusted to modify search behavior.

Search for interlayers

Activates the function to search for interlayers.

Interlayer at pallet bottom

If this setting is selected, FlexLoader Vision will send commands to the robot to pick an interlayer when the bottom of the pallet is detected (flagged as PICK_INTERLAYER_AND_CHANGE_BIN).

Threshold height above interlayer

FlexLoader Vision starts interlayer detection by fitting a plane to all data points within the specified height of the search volume (see camera settings). After fitting the plane, a check is performed if anything is still present on top of the plane (i.e. the interlayer). This parameter specifies the height at which FlexLoader Vision is to search for remaining objects. Its value must be less than the height of the parts to be picked.

Max total area above threshold

Indicates the total area allowed to protrude above the specified threshold height. If the protruding surface exceeds this value, no interlayer is found. The contour lines for these areas are marked green in FlexLoader Vision during analysis.

Min allowed dust thickness

All areas in the threshold image that are narrower than this value will be filtered out prior to analysis.

Start height pallet bottom search

Interlayers or empty flat areas satisfying interlayer detection criteria are considered as pallet bottom if their height is below the **Start height pallet bottom** value.

After the interlayer analysis, the letter **I** together with a blue ring, will appear in the image to indicate that an interlayer has been found. If a blue cross appears on top of this, the analysis was performed without finding a valid interlayer. Note that no

Continues on next page

detail was found in the case below. Areas protruding the threshold height above the interlayer are marked in green.

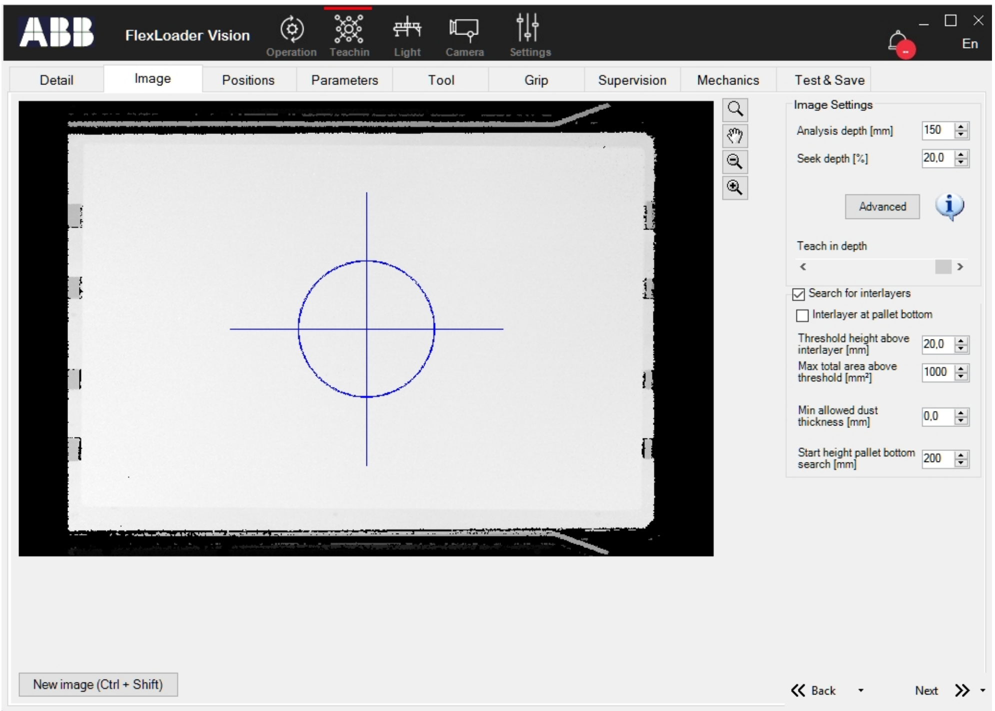


Image Settings

Analysis depth [mm] 150

Seek depth [%] 20,0

Teach in depth

☑ Search for interlayers

□ Interlayer at pallet bottom

Threshold height above interlayer [mm] 20,0

Max total area above threshold [mm²] 1000

Min allowed dust thickness [mm] 0,0

Start height pallet bottom search [mm] 200

xx1800000211

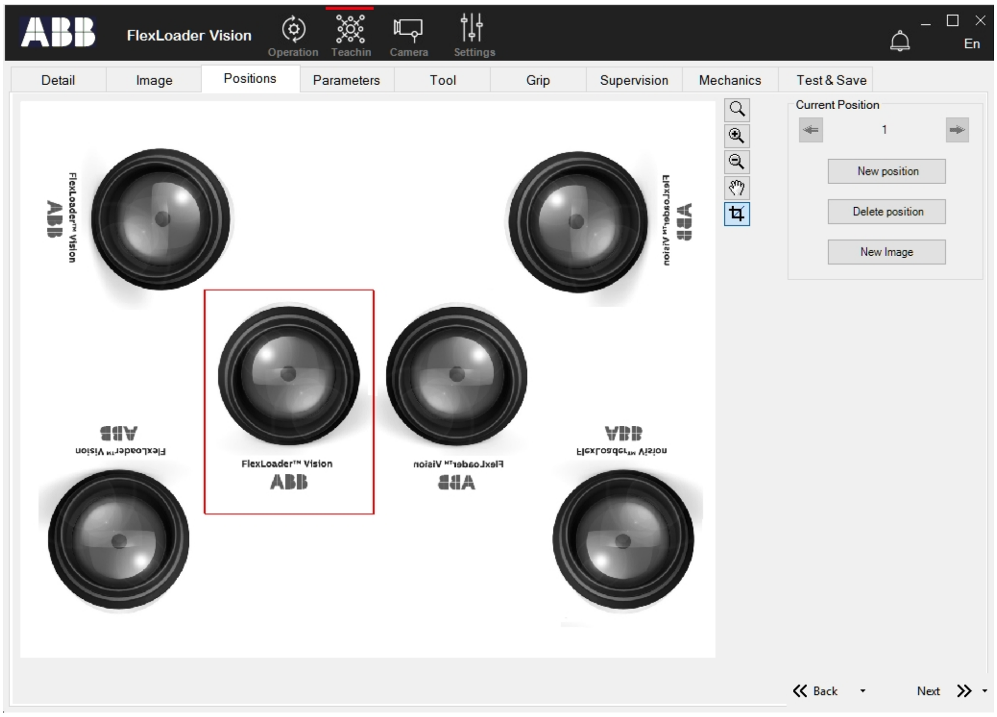
Pallets without interlayer

If searching for interlayers is deactivated, FlexLoader Vision will not search for interlayers when no details are identified. Instead, the highest point in the cameras region of interest is evaluated in order to decide what information is sent to the robot, see *Region of interest on page 118*.

If the highest point is below the minimum z-value in the camera settings (see *Image size on page 117*), the robot is informed to change the pallet (flagged as CHANGE_BIN). Else, the robot is informed that no details were found.

4.5 Positions

A detail can often lie in a number of different positions. These must be defined on the **Positions** tab. In the example below we teach the system to recognize the text FlexLoader Vision. The text can lie in two positions, either the right way up or mirrored. Clicking on the button **New position** creates a new position. This position must be defined by drawing a red box around the detail in the field of view that corresponds to the position. This is done using the Define position tool. Note that a position will be identified even if it is lying in a different angle (rotated) than it was at teachin.

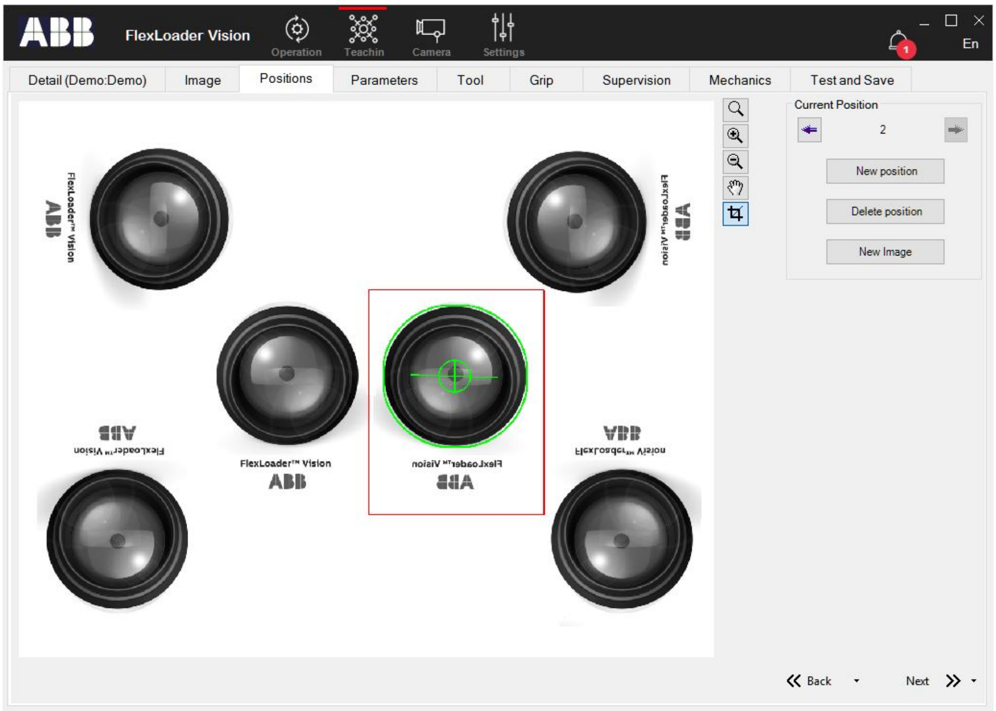


xx1800000212

The next position is defined by clicking on **New position** again and drawing a box around the detail with the mirrored text.

Continues on next page

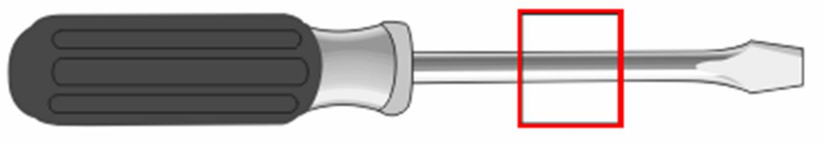
If the analysis type **Blob** is used for the detail, the outer contour and the found angle of the detail are drawn instead.



xx1800000213

By clicking on the **Current position** arrows you can browse through all the teachin positions.

There are certain factors to bear in mind when defining a position. All positions must be defined in a way that makes it possible to create a clear geometric model of the position, i.e. a model that unambiguously identifies that position.



xx1800001031

The box defined in the screwdriver image above is an example of an unclear position. The same geometric section shown above exists in many different places along the shaft of the screwdriver and could result in incorrect positioning when the detail is gripped or handled. It is often best to define a box around the entire position.

However, you should avoid drawing a box that is larger than necessary around the detail. The information that exists outside the detail could affect the analysis results in a way that makes it more difficult to set the parameters.

If a position is not wanted you can remove it by clicking on **Delete position**.

If it is difficult to draw a box around a position you can adjust the location of the detail under the camera and then click on **New Image**. A new image will be taken and the content of the box will be redefined.

4.6 Parameters

Settings for 2D standard analyses

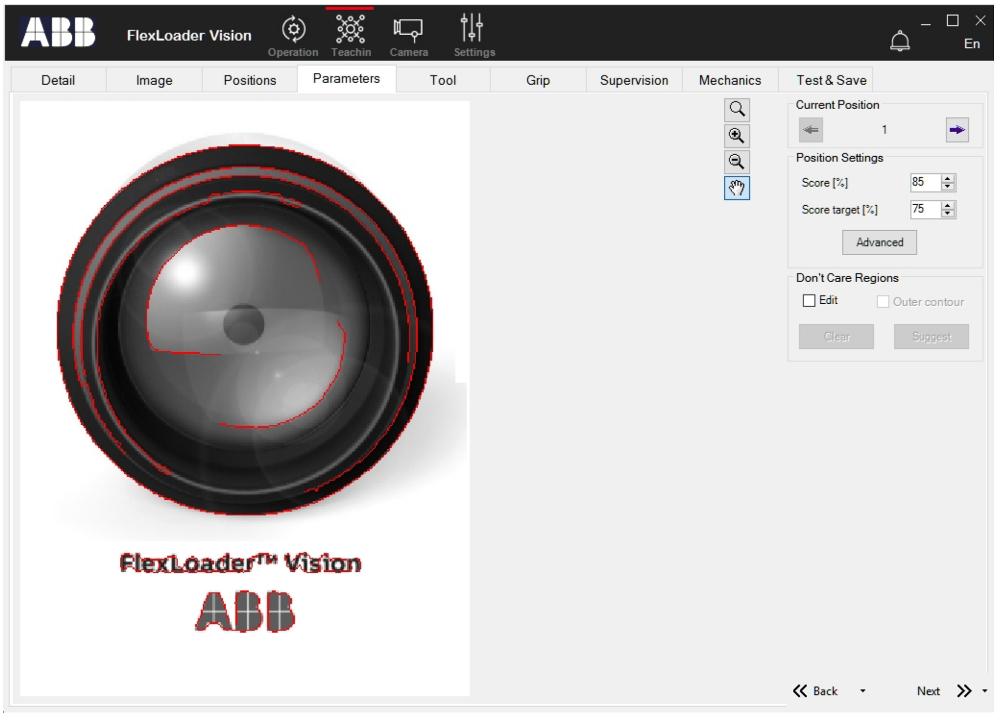
Introduction

The way a detail is identified and the requirements that must be met before a detail is assumed to have been identified can be adjusted in a number of ways. The basic threshold values for identification are set on the **Parameters** tab.

Note that the parameters differ between the analysis types **Standard** and **Blob** (if blob searching is enabled). The standard analysis is the alternative that you will use in the majority of the cases. The blob analysis is used only in exceptional cases where the shape of the detail (blob) may vary within certain specified ranges.

To identify a position, the system searches for edges in the image. The edges found are marked with red lines on the image. All the edges that are found are then joined together to form a geometric model that is compared with the teachin geometric model.

Before adjusting the search settings you need to understand how the search results are calculated when comparing the teachin position and the found position. The main parameters that determine the search results are called **score** and **score target.**



Score [%] 85

Score target [%] 75

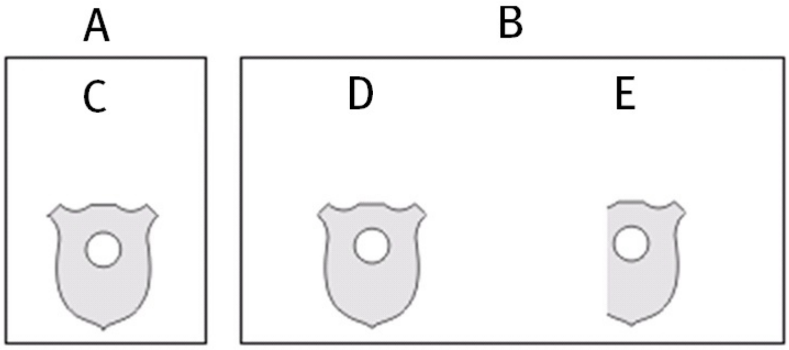
xx1800000214

Continues on next page

Score

Score is a measure of how well the found edges match the teachin edges.

The image below shows an example in which a number of edges have been missed in one of the two found positions (E). This has a negative effect on the **Score** value.



xx1800001032

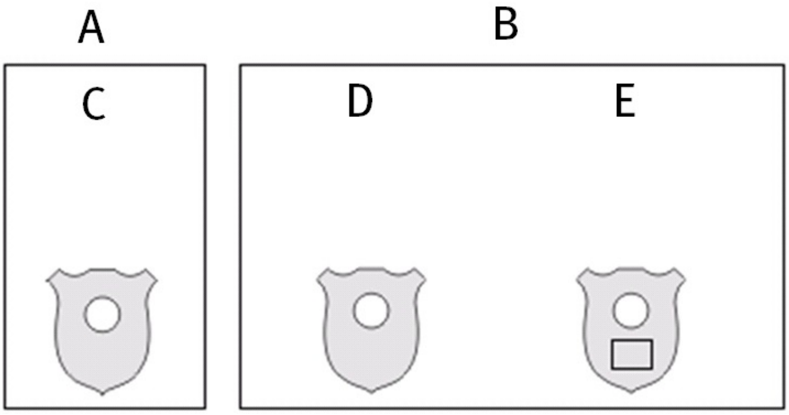
Pos.	Description
A	Teachin position
B	Found positions
C	Position 1
D	Position 1, **Score** 100%
E	Position 1, **Score** 75%

Figure 4.2:

Score target

Score target is a measure of found edges that do not match the teachin position, in other words: extra edges. Found edges that do not match those in the teachin position will reduce the score target.

The image below shows an example in which extra edges have been identified in the found position to the right (E). In this case the **Score target** value is reduced to 85%. However, the **Score** value for both found positions is 100% because the found positions still have a 100% geometrical match with the teachin position.



xx1800001033

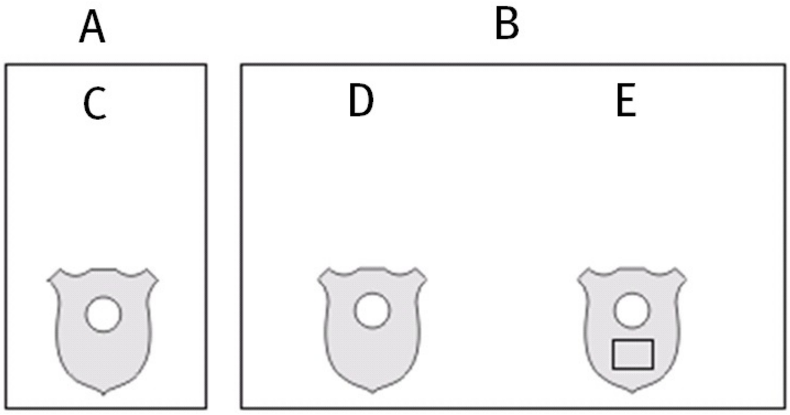
Continues on next page

Pos.	Description
A	Teachin position
B	Found positions
C	Position 1
D	Position 1, **Score** 100%, **Score target** 100%
E	Position 1, **Score** 100%, **Score target** 85%

Don't Care Regions

When a position has been defined you can mask out certain parts of the position by marking them as areas that the system should ignore during the analysis. For example, a position may contain some parts that can appear very different to the camera depending on their orientation. This is very common when the light reflections from the detail vary from detail to detail. By masking these parts out you can ensure that they do not affect the search results.

In the image below (with no masking) there are extra edges in the found target to the right (E). These extra edges have a negative effect on the **Score target**, and in this case the result is 85%.



xx1800001033

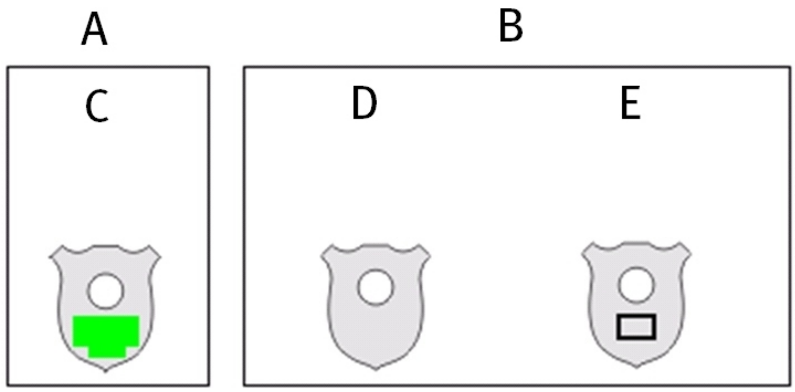
Pos.	Description
A	Teachin position
B	Found positions
C	Position 1
D	Position 1, **Score** 100%, **Score target** 100%
E	Position 1, **Score** 100%, **Score target** 85%

Figure 4.3: No masking

In the image below, parts of the teachin position have been masked out. During the search, any edges that are found in this area are excluded from the calculations. In this example it means that the **Score target** value for the two teachin positions is unaffected by the extra edges in the position to the right (E).

Continues on next page

To mask a position, select **Edit** in the **Don't Care Regions** pane.



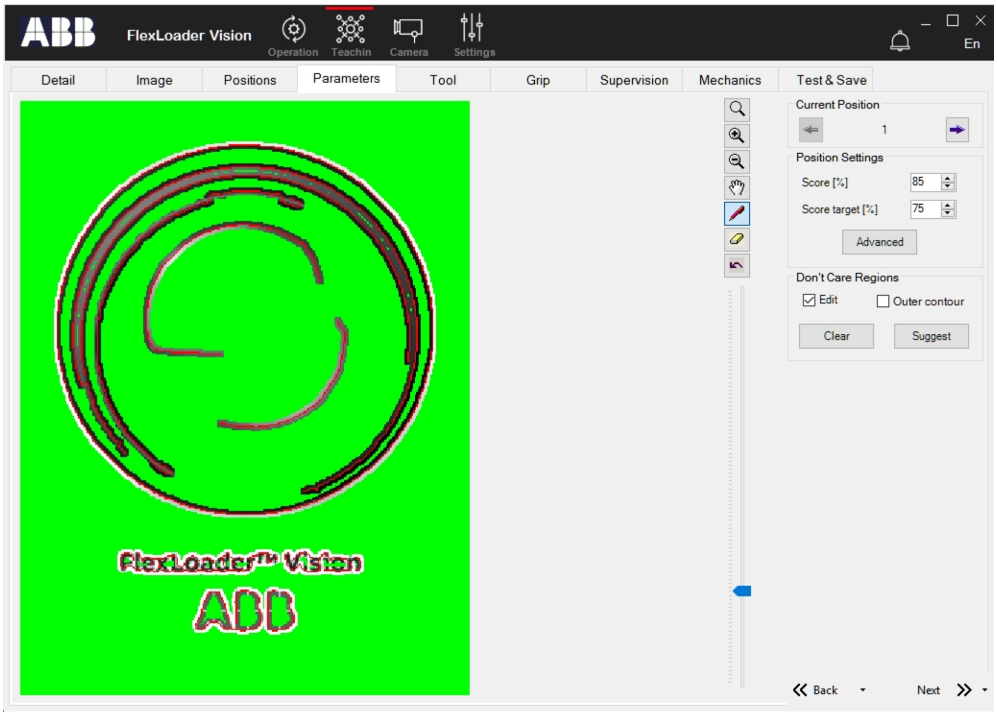
xx1800001034

Pos.	Description
A	Teachin position
B	Found positions
C	Position 1
D	Position 1, **Score** 100%, **Score target** 100%
E	Position 1, **Score** 100%, **Score target** 100%

Figure 4.4: Masking applied

If you want FlexLoader Vision to suggest a masked area, click **Suggest**. Select **Outer contour** to obtain a suggestion only for outer contour lines.

Use the **Current position** arrows to browse through all the teachin positions.



Current Position: 1

Position Settings

Score [%] 85

Score target [%] 75

Don't Care Regions

☑ Edit ☐ Outer contour

xx1800000215

Continues on next page

AddOnAnalysis

When a position has been defined, further analyses can be specified (if enabled for the system). These can, for example, help differentiate very similar positions. Click **AddOn Analysis** to specify the extra analyses.

Settings for 2D blob analyses

Introduction

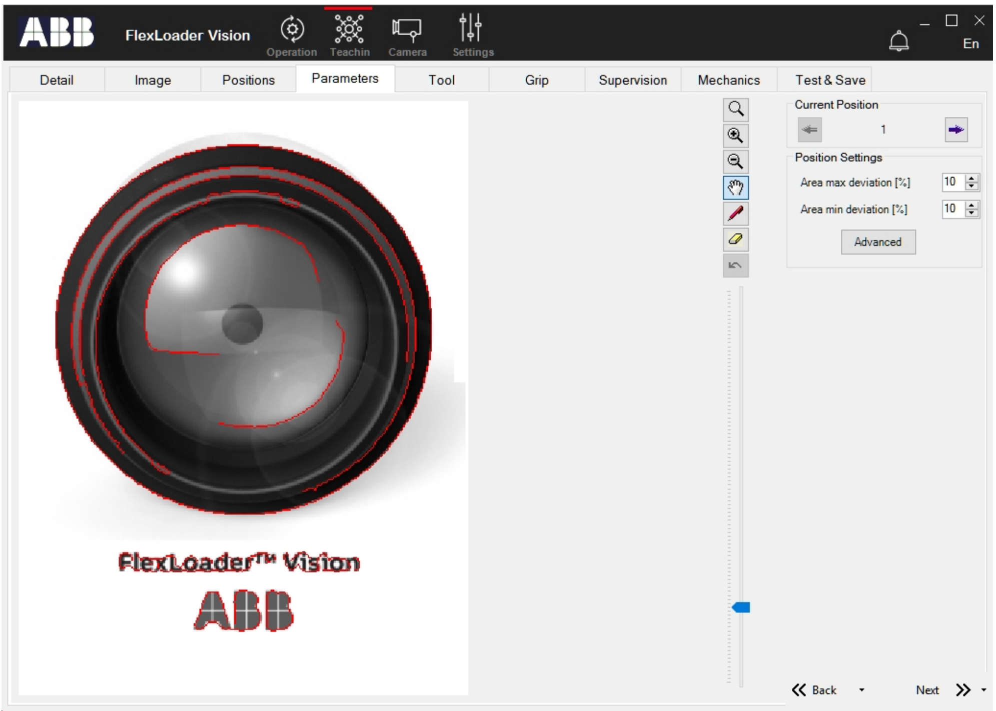
The standard analysis is the analysis alternative that you will use in the majority of all cases. But sometimes, when the shape of the detail may vary within certain ranges, it can be useful to do a blob analysis.

During identification of a position, the system analyzes the properties of each blob found in the image and compares them with the corresponding property of the teachin detail. The properties that are evaluated are area, perimeter, elongation and compactness. The analyses of all properties except area can be deselected.

For all properties, an upper (**Area max. deviation**) and lower (**Area min. deviation**) limit is specified as a percentage. The property requirements for the two positions must be completely different in at least one aspect, otherwise a warning is displayed to the user.

In many cases, it is sufficient to specify the area settings to identify details. The area settings are therefore directly accessible on the user interface.

In some cases, these may need to be supplemented with perimeter, elongation and compactness in order to differentiate the details better. These settings are accessed by clicking on **Advanced** and described in *Advanced teachin settings on page 71*.



Current Position 1

Position Settings

Area max deviation [%] 10

Area min deviation [%] 10

xx1800000216

Continues on next page

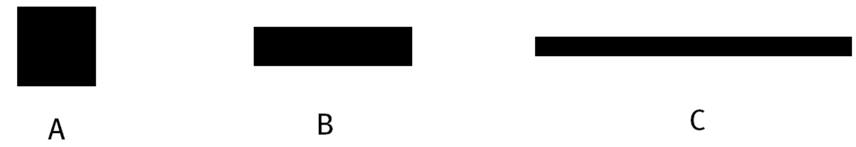
In order to understand the meaning of these parameters, descriptions and some examples in which different types of details are set against each other, are listed below.

Area and perimeter

The values for area and perimeter are evaluated directly from the image.

The value of the elongation is the relationship between the blob's length and width. For the sake of simplicity, it is assumed that the area is [length × width], and that the perimeter is [2 × (length × width)]. This applies if the blob's length and width is the same along the entire blob, but also applies for elongated thin blobs, even if they are curved.

The following image shows the difference in elongation for a blob with the same area.



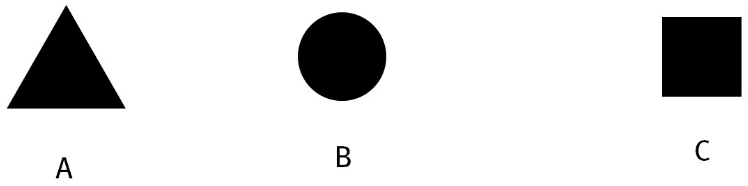
xx1800000301

Pos.	Area	Elongation
A	1	1
B	1	4
C	1	16

Compactness

The value of compactness is a measurement of how close all parts of a blob are to each other. A circular shaped blob is the most compact, with a value of compactness of 1.0. The more the shape deviates from a circle, the greater the value of compactness. Compactness is based on area and perimeter and is calculated as (perimeter × perimeter)/(4 × π × area).

The following image shows the difference in compactness for a blob with the same area.



xx1800000302

Pos.	Area	Compactness
A	1	1.65
B	1	1.00

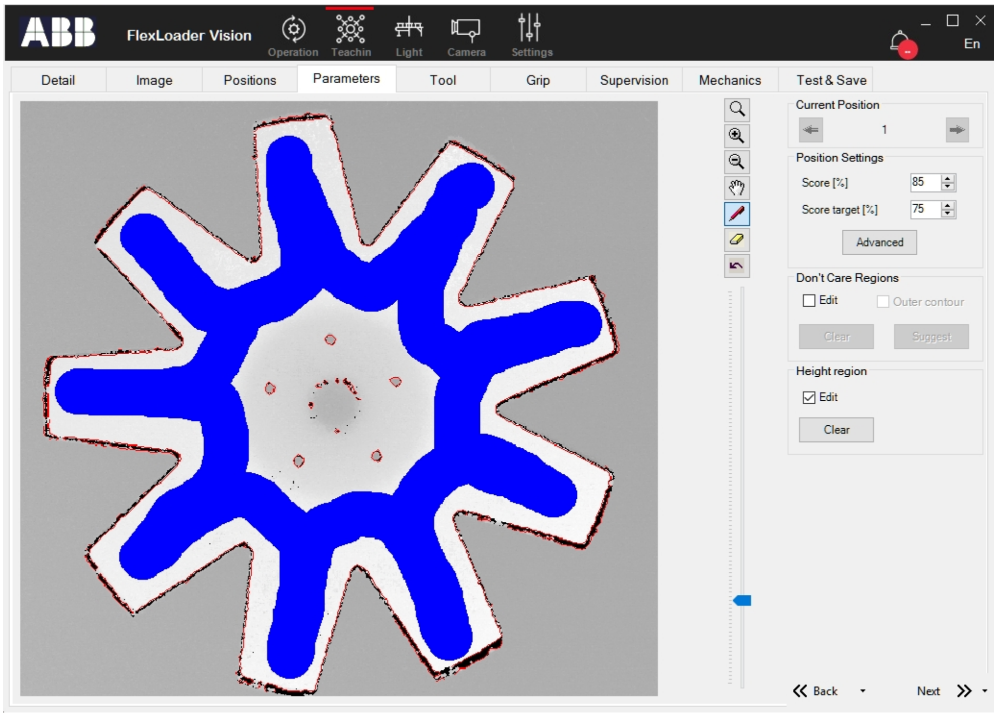
Continues on next page

Pos.	Area	Compactness
C	1	1.27

Settings for 3D analyses

3D analysis uses the same search principles as the 2D standard analysis to identify a detail. This means that the position settings and the masking work in exactly the same way.

3D analysis includes finding the height and angle in the x and y axes for the detail. To make this possible, a height region is defined. This is done by selecting **Edit** under the heading **Height region** and then drawing a blue area on the detail in the image. A height region must always be defined when using a 3D camera.



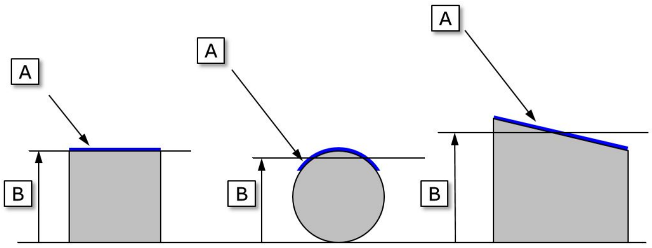
xx1800000217

When a detail has been identified by FlexLoader Vision, the height region value is used to establish a plane that lies on the detail. The average height of the plane is the height of the detail, and the angle of the plane gives the angles in the x and y axes. The height region must be drawn on a horizontal planar surface. Note that the height region should not be drawn up to the edge of the detail, because then the blue area might end up outside of the detail, if the detail leans more than it did at teachin.

However, in certain cases the height region must be drawn on a curved surface. A common example is lying cylinders. Then it is necessary to know that the average height of the located detail surface actually is below the top of the detail. For the

Continues on next page

robot to pick the detail correctly, this must be compensated for with the gripper settings on the **Grip** tab.



xx1800000304

Pos.	Description
A	Height region
B	Located height

Handling angle variations

By default, the averaged height of the height region will be used to only control the gripping height. Small angular deviations (inclination) will not influence the gripping position or tool rotation, and the rotation around x- and y-axis is locked to 0°.

When locking of X- and Y-angles has been turned off, the gripping position and tool rotation will follow the inclination of the part within reasonable limits (approximately +/- 10°), see *Detail Settings on page 85*.

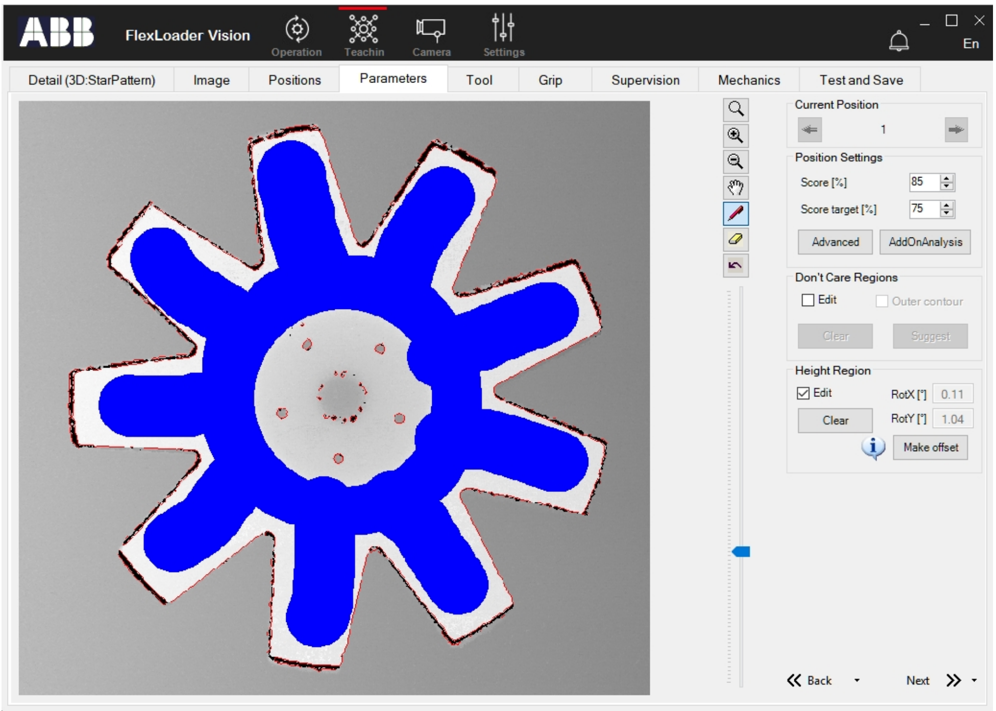
Note that a horizontal height region is normally required during teachin. The horizontal orientation is used as default offset when calculating gripping positions and gripper orientations.

Continues on next page

Inclined height regions

For some parts it might be impossible to find a horizontal height region. FlexLoader Vision helps the operator to identify and solve this situation.

The angle of the height region is displayed directly after drawing it, provided that locking of X- and Y-angles has been turned off, see *Detail Settings on page 85*.



Current Position: 1

Score [%]: 85

Score target [%]: 75

Don't Care Regions: □ Edit □ Outer contour

Height Region: ☑ Edit

RotX [°]: 0.11

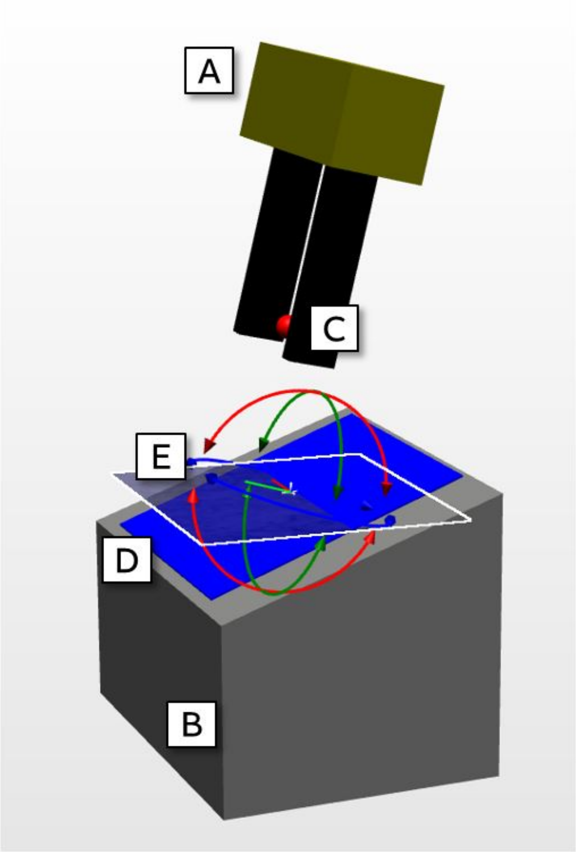
RotY [°]: 1.04

xx1800002706

If needed, the height region can then be offset by the appropriate value by clicking **Make Offset**. This offset value is used to rotate the height region prior to further

Continues on next page

analysis. The values are transferred to the **Offset height region** behavior in the **Grip** tab, see *Rotation behavior on page 57*.



xx1800002703

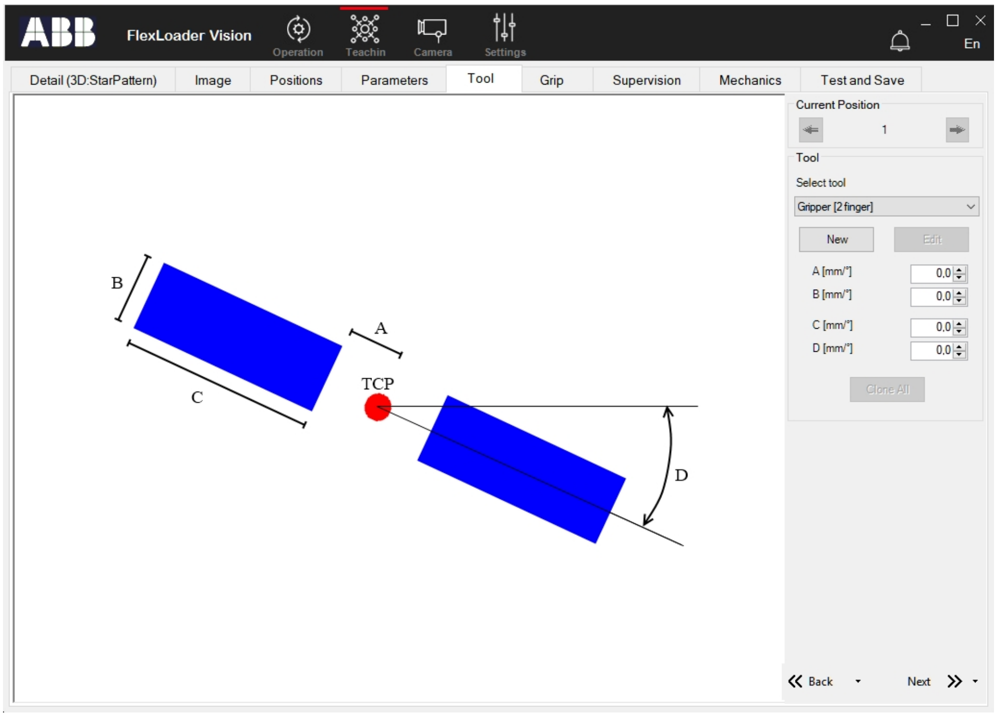
Pos.	Description
A	Gripper
B	Part
C	Tool center point
D	Height region
E	Offset height region

As soon as another image has been taken, or after leaving teachin, the rotation information information is no longer available.

4.7 Tools

Introduction

A gripper must be defined for each teachin position. The gripper information is used to simplify the visual definition of the grip point, and to enable the system to supervise tools and thereby avoiding collisions between the gripper and the details during operation. Select the gripper to be used for the current position on the **Tool** tab.



Current Position: 1

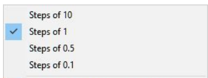
Tool

Select tool: Gripper [2 finger]

A [mm/°] 0,0
B [mm/°] 0,0
C [mm/°] 0,0
D [mm/°] 0,0

xx1800000218

The system has a number of predefined gripper types. These are defined by certain parameters, and when you select a gripper you must enter the dimensions of the gripper. The dimensions that are needed are shown by a schematic illustration of the tool type. The red dot represents the tool center point (TCP). The dimensions can be set in steps of different sizes. Select step size by right-clicking with the mouse to bring up the following menu.



xx1800000305

This lets you choose the size of the adjustment to be made. The currently selected step size is marked with a tick to the left.

If none of the predefined tools are suitable you can draw a new tool.

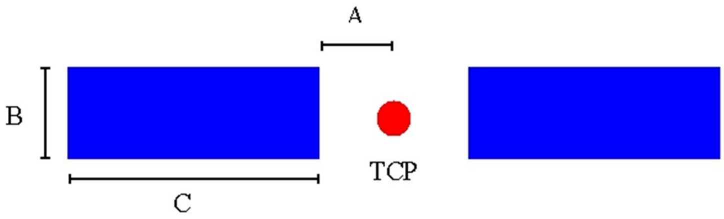
It is possible to have different tools linked to different positions. Usually, however, the same tool is used for all positions, and if this is the case you can easily link a defined tool to all of the positions by clicking **Clone all**.

Continues on next page

By clicking on the **Current position** arrows you can browse through all the teachin positions.

Predefined tools

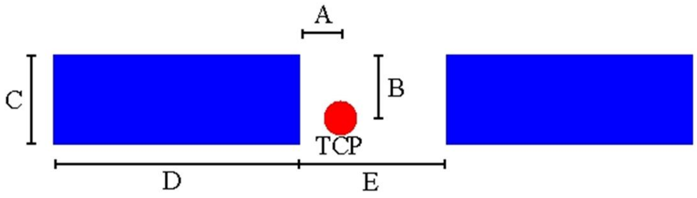
Gripper [2 finger]



xx1800001035

Corresponds to a gripper with two gripping fingers and the tool center point (TCP) located between the fingers. The indicated dimensions are with the fingers open.

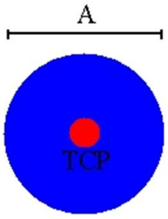
Gripper [2 finger advanced]



xx1800001036

Corresponds to a gripper with two gripping fingers and offset tool center point (TCP). The indicated dimensions are with the fingers open.

Suction cup

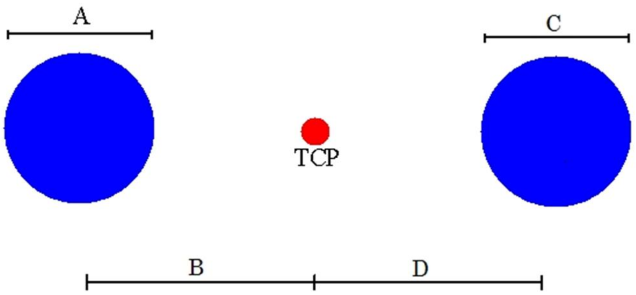


xx1800001037

Corresponds to a tool with a single suction cup and the tool center point (TCP) in the center of the suction cup. Note that when a suction cup is used, tool supervision is not active. This is because one suction cup must always be positioned on the detail.

Continues on next page

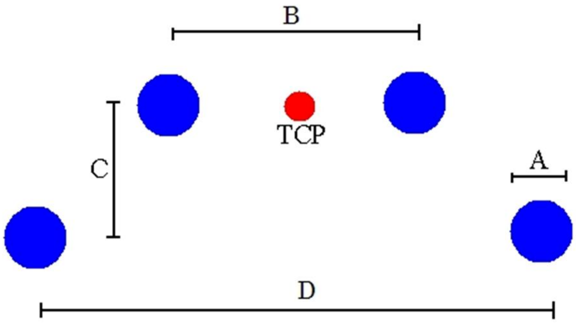
DoubleSuctionCup



xx1800001038

Corresponds to a gripper with double suction cups, which can be of different sizes. The tool center point (TCP) can be offset between the suction cups. Note that when double suction cups are used, tool supervision is not active. This is because the suction cups must always be positioned on the detail.

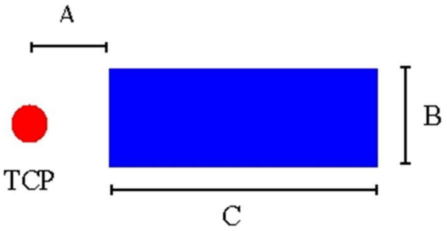
Four suction cups



xx1800001039

Corresponds to a tool with four suction cups and with the tool center point (TCP) located between the two upper suction cups. All suction cups are the same size. Note that when the four suction cups are used, tool supervision is not active. This is because the suction cups must always be positioned on the detail.

Single-sided gripper

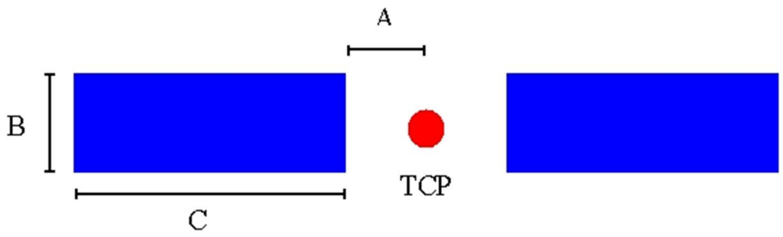


xx1800001040

Corresponds to a gripper with a single gripping finger and the tool center point (TCP) offset to the left of the finger.

Continues on next page

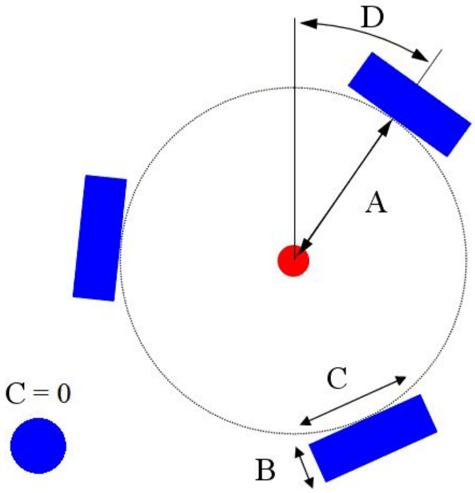
Gripper [2 finger] rotated



xx1800001041

The same as a two-finger gripper, but rotated 90° to fit alternative adapters on the robot arm.

Gripper [3 finger]



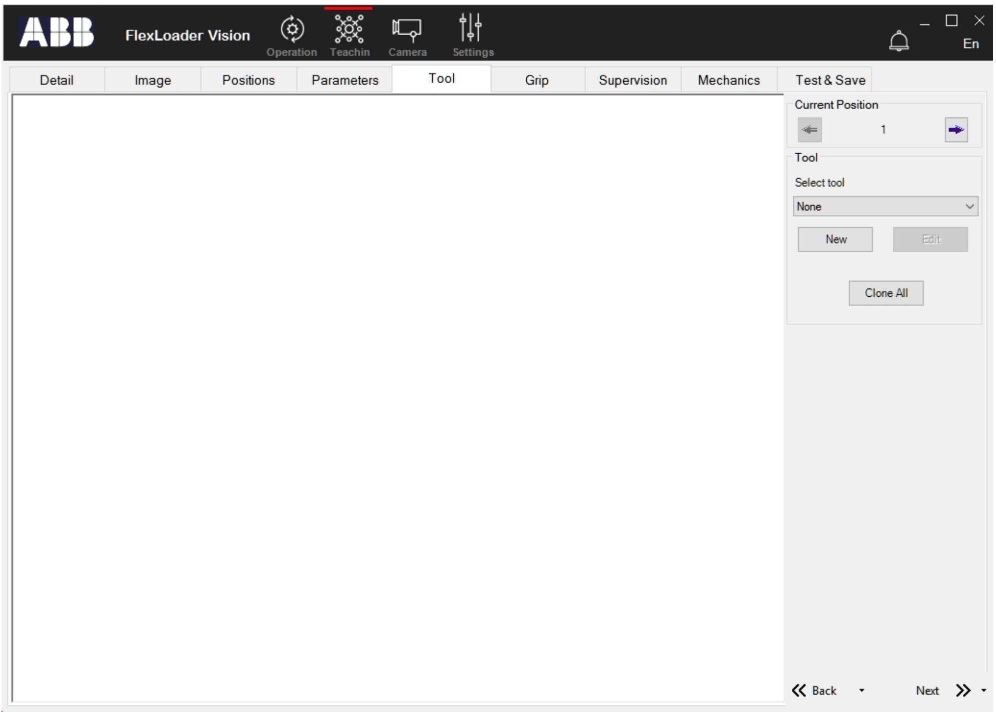
xx1800001042

A three finger gripper that can be turned relative to the robots wrist coordinate system.

Continues on next page

Creating your own tool

If none of the predefined tool types are suitable it is also possible to draw the tool you intend to use.

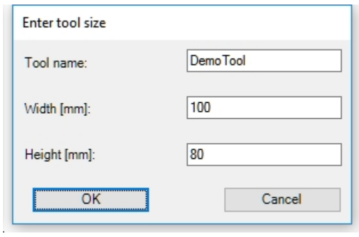


Current Position: 1

Select tool: None

xx1800000219

Click the **New** button under the gripper to bring up the following dialog box:



Enter tool size

Tool name: DemoTool

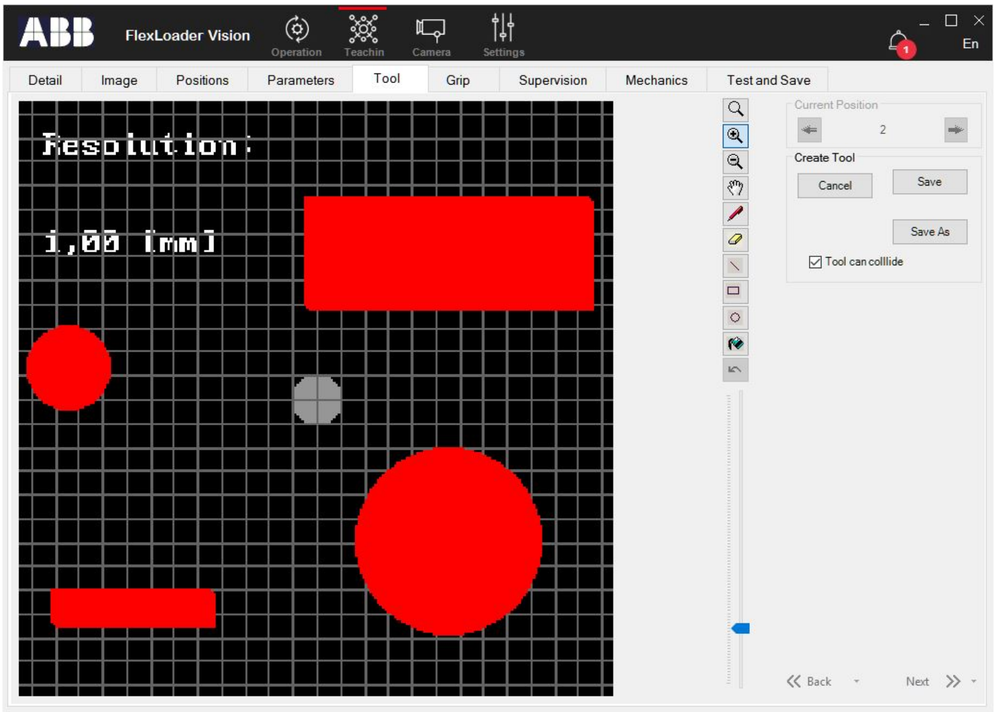
Width [mm]: 100

Height [mm]: 80

xx1800000220

Continues on next page

Enter the name of the new tool and its external dimensions. The external dimensions are needed to specify the accuracy for drawing the new tool. A small tool is easier to draw accurately than a large tool.



Resolution:

1,00 [mm]

Current Position: 2

☑ Tool can colllide

xx1800000221

The above information will appear during drawing. The image appears against a grid, with the grid resolution shown in the top left corner. The grey box in the middle of the image represents the tool center point (TCP).

Use the drawing tools to draw the desired tool. The entire tool should be marked in red, see the example in the image above.

When you have finished drawing the tool, click **Save** to exit the drawing function.

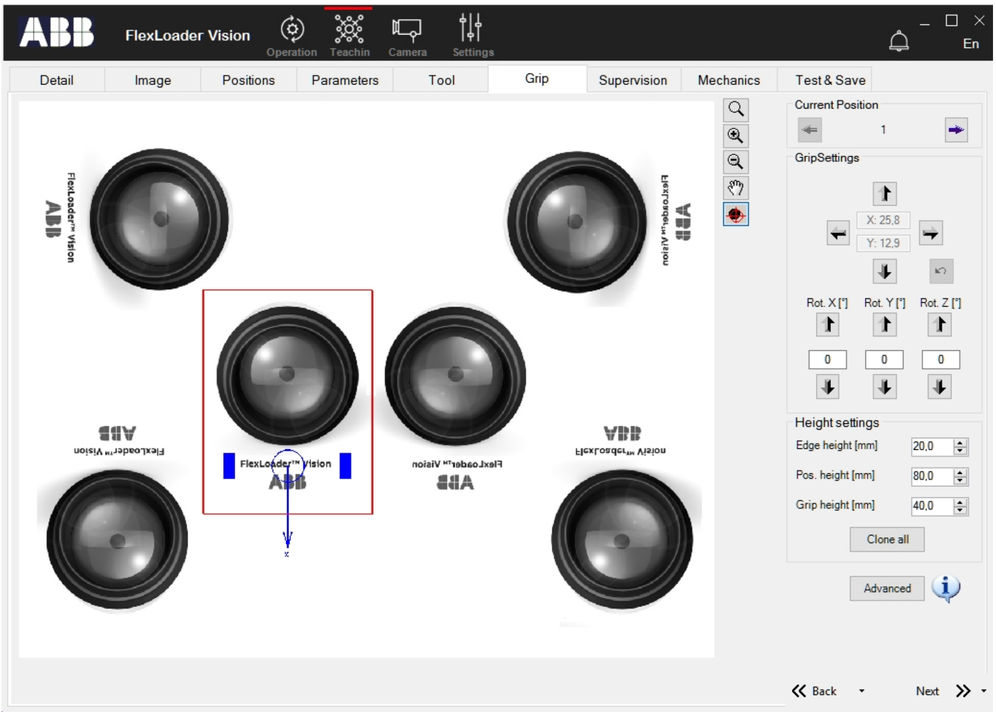
If you want to modify a tool you have defined previously, select the tool and click **Edit**. When the tool has been modified you can either save the tool by clicking **Save**, or save it with a different name by clicking **Save As**.

4.8 Grip

Grip settings for 2D cameras

Introduction

The grip point for each defined position should be marked on the **Grip** tab.



Current Position: 1

GripSettings: X: 25,8 Y: 12,9

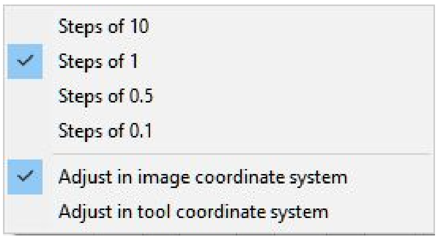
Rot. X [°]: 0 Rot. Y [°]: 0 Rot. Z [°]: 0

Height settings

- Edge height [mm]: 20,0
- Pos. height [mm]: 80,0
- Grip height [mm]: 40,0

xx1800000222

This is done by selecting the tool in the **Select tool** drop down list, defining the grip point (by specifying the grip and height settings) and marking the tool position in the image. Once the position has been marked you can fine-tune the adjustments with the arrows. Fine-tuning can be carried out in steps of different sizes. Select the step size by right-clicking with the mouse to bring up the following menu.



- Steps of 10
- ✓ Steps of 1
- Steps of 0.5
- Steps of 0.1
- ✓ Adjust in image coordinate system
- Adjust in tool coordinate system

xx1800000306

At teachin you should try to position the detail so that the grip point is located in the center of the image. This is particularly important for details with high edges or large grip height. If the center of perspective (the point in the field of view where the camera is looking vertically down on the conveyor) does not align with the image center, the detail should be placed so that the grip point is located close to the center of perspective.

Continues on next page

This lets you choose the size of the adjustment to be made. The currently selected step size is marked with a tick to the left.

You can also choose which coordinate system to use for movement; the image coordinate system or tool coordinate system. If the image coordinate system is chosen, movement will always be directly related to the directions of the arrows. If the tool coordinate system is chosen, movement will be in relation to the current rotation of the tool.

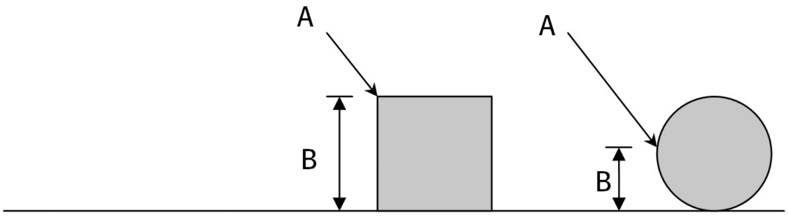
The current setting applies to all adjustments available under the tab.

The rotation x, rotation y and rotation z settings control the rotation of the robot when gripping or handling the position. When you adjust rotation z, the adjustment is illustrated graphically in the image. When you adjust rotation x and rotation y, you have to do a gripping test to see how the setting works in reality.

Note that the gripper rotation during blob searches is not unequivocally determined, but can be rotated 180 degrees.

Height settings

The edge height must be specified for each position. The edge height is the difference in height between the visible edges and the calibration plate. The visible edges are those edges that are marked with red lines in the image.



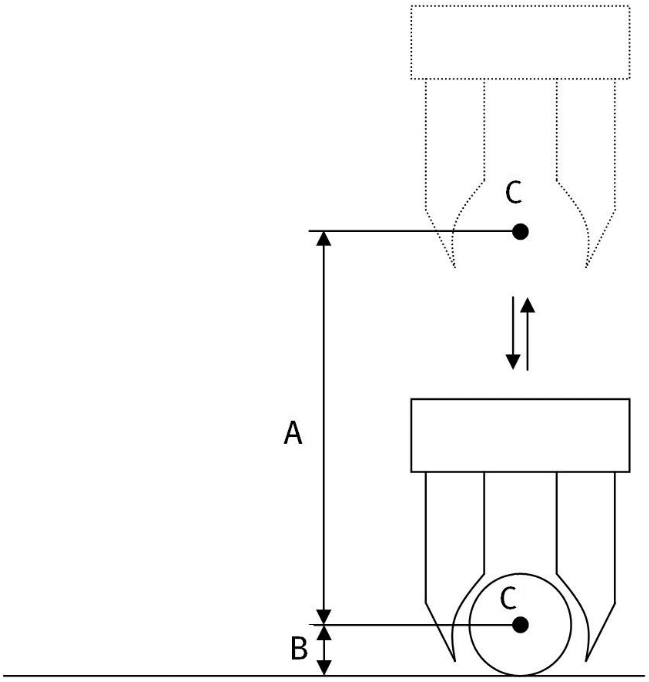
xx1800001043

Pos.	Description
A	Edges
B	Edge height

The **Edge height** value directly affects the positioning accuracy during gripping and handling of the position. It is therefore very important that the height is correctly set in the system.

Continues on next page

The **Pos. height** value specifies the height of the point above the grip height where the tool is to be positioned before gripping.



xx1800001044

Pos.	Description
A	Positioning height
B	Grip height
C	Tool center point

The **Grip height** value specifies the height at which the position is to be gripped or handled.

You can click **Clone all** to use these settings for all positions if all the positions have the same edge height, positioning height and grip height.

Grip settings for 3D cameras

Introduction

The grip points are positioned in the same way as for 2D. However, there are some setting differences as follows.

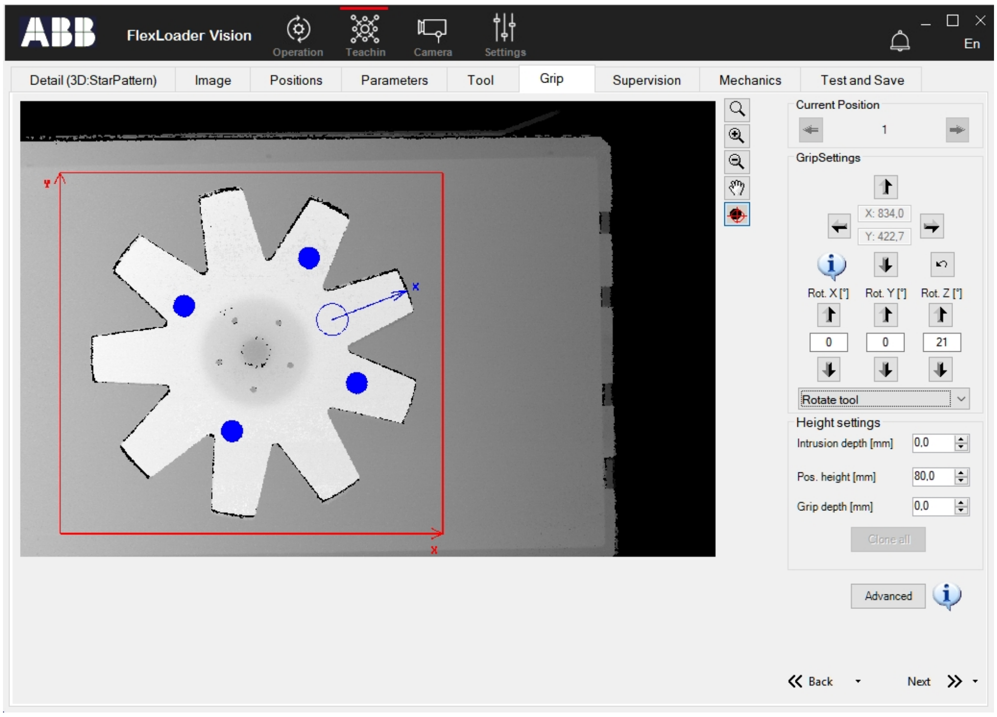
Note that the perspective concerns do not exist when the 3D camera is used, so there are no problems with this when the grip point is positioned.

Rotation Z sets the robot tool rotation around z alignment of the detail. There is a direct connection with the image of how the grip device rotates, just like with the 2D camera.

There is no direct connection in the image for x and y rotation. However, the x- and y-axes for the detail are drawn in the image. To understand how the grip device

Continues on next page

will rotate, bear in mind that the z-axis rotation occurs first, then rotation around the y-axis and finally rotation around the x-axis.



Current Position: 1

GripSettings

X: 834,0

Y: 422,7

Rot. X [°]: 0

Rot. Y [°]: 0

Rot. Z [°]: 21

Rotate tool

Height settings

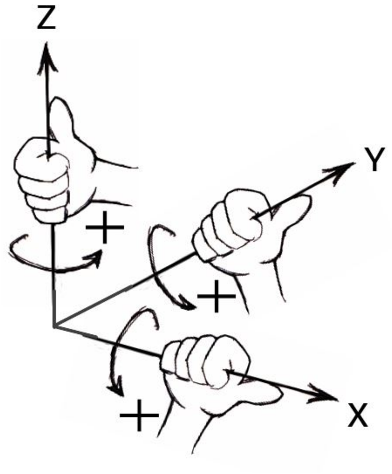
Intrusion depth [mm]: 0,0

Pos. height [mm]: 80,0

Grip depth [mm]: 0,0

xx1800000223

The positive direction of rotation can be easily remembered by the right hand rotation rule.



xx1800000307

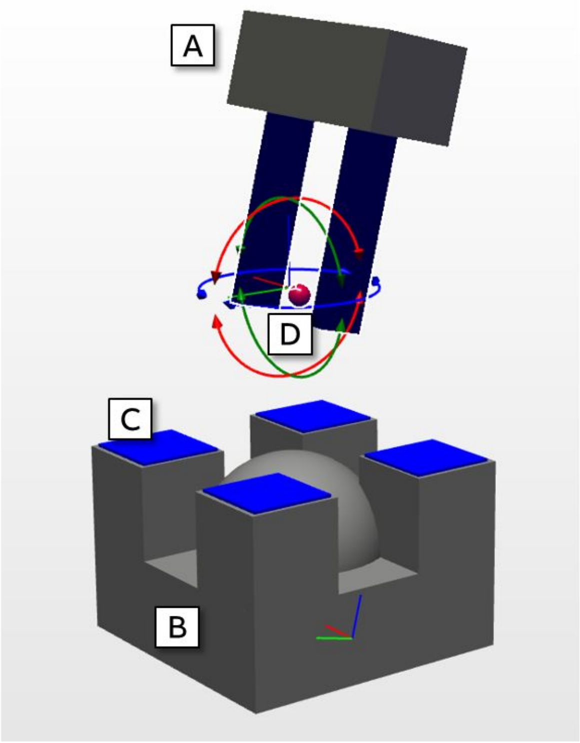
Rotation behavior

The inclination of the height region may control the gripping position and tool rotation. The operator can specify an additional rotation to be applied if needed.

Select **Rotate Tool** in order to use this additional rotation to rotate the tool around the tool center point.

Continues on next page

In this case the teachin must be done with horizontal height regions (or x- and y-axis angles must be locked).



xx1800002702

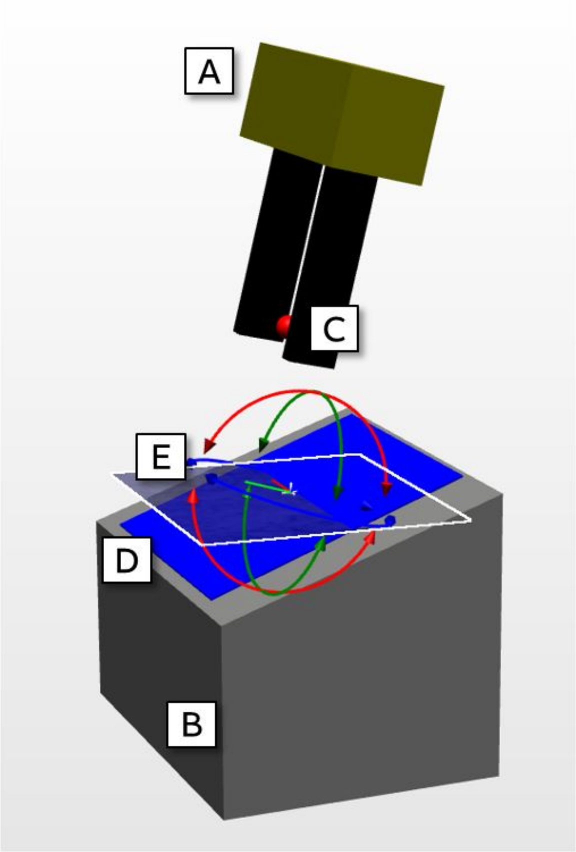
Pos.	Description
A	Gripper
B	Part
C	Height region
D	Tool center point

Select **Offset height region** in order to use this additional rotation to modify the height region to be horizontal (a horizontal orientation is used as default offset when calculating gripping positions and gripper orientations).

The **Make Offset** function can be used during teachin, see *Inclined height regions on page 46*.

Continues on next page

In this case the tool cannot be additionally rotated around the tool center point.



xx1800002703

Pos.	Description
A	Gripper
B	Part
C	Tool center point
D	Height region
E	Offset height region

Height settings

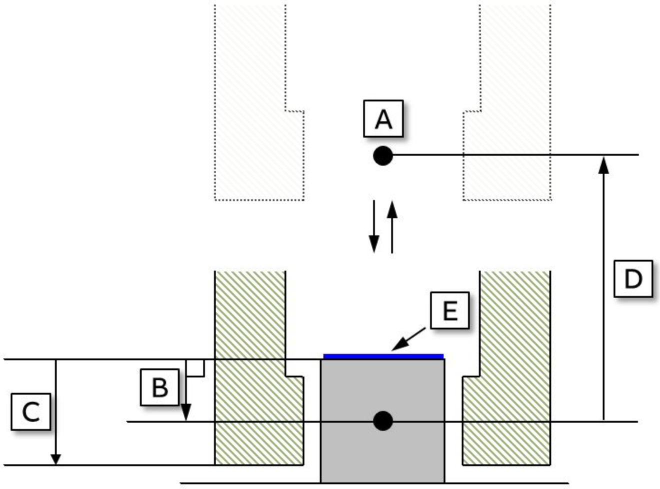
The **Intrusion depth** value specifies how deep the grip device goes below the located detail height at the picking position. The depth is measured positively from the located detail height and down. This value is used to determine at what height a check for gripper collision occurs.

Positioning height has the same function as for 2D cameras.

The **Grip depth** value specifies how deep the TCP of the grip device goes below the located detail height at the picking position. The depth is measured positively

Continues on next page

from the located detail height and down. This value is used to determine what grip height is to be sent to the robot.



xx1800000308

Pos.	Description
A	Tool center point
B	Grip depth
C	Intrusion depth
D	Positioning height
E	Height region

Note

The highest point determined within the cameras region of interest (see *Region of interest on page 118*) sets an upper limit for the allowed height of details. All details more than 50 mm above the highest point will be discarded.

4.9 Supervision

Supervision settings for 2D cameras

Introduction

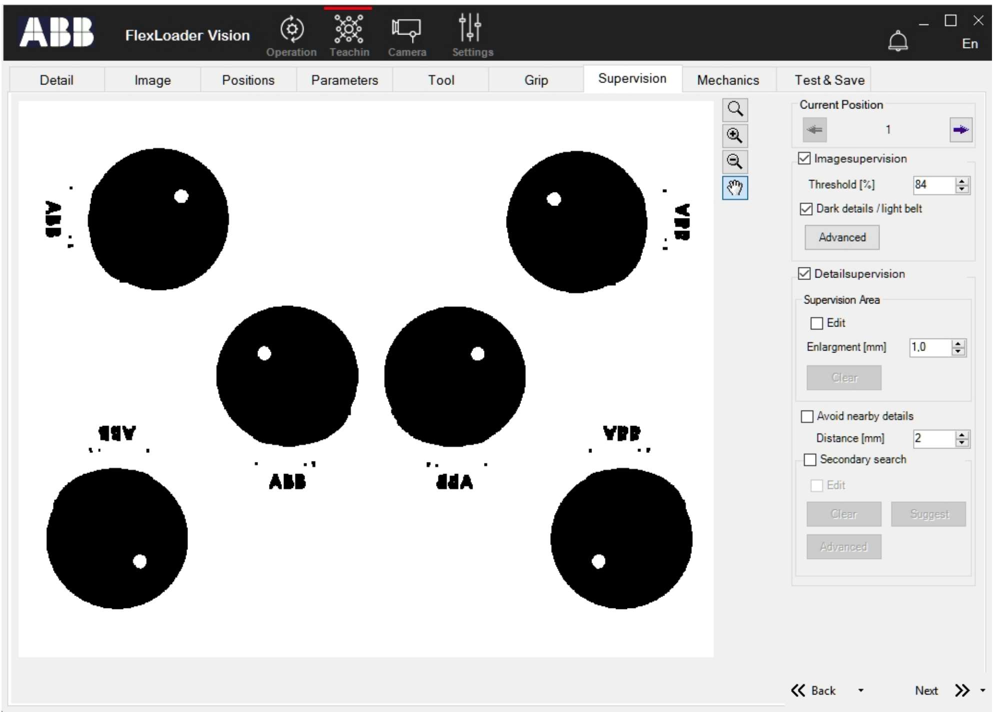
On the **Supervision** tab, you can adjust the settings for collision sensing. The patented technology for collision supervision prevents collisions between the gripper and the adjacent details to the greatest possible degree.

Collision supervision is achieved by calculating a collision risk image and then comparing this image with the desired tool positions. The collision risk image can be calculated in two ways, which can be used independently or in combination: **Image supervision** and **Detail supervision.**

If the settings are incorrect or if collision risk image calculation is disabled, collisions could occur during picking.

Image supervision

In **Image supervision**, data from the entire camera image is used to provide information about the collision risk image. The image is built up of zones where there is a collision risk (marked in black) and zones where there is no risk of collision (white). To activate image supervision, select **Image supervision**. Then adjust the **Threshold** value so that the details appear as black as possible and the background as white as possible. Use **Dark details/light belt** to indicate if the details are dark and the conveyor is light in color, or vice versa. The resulting collision risk image is updated continuously to permit rapid checking.



☑ Imagesupervision
Threshold [%] 84
☑ Dark details / light belt

☑ Detailsupervision
Supervision Area
☐ Edit
Enlargment [mm] 1,0

☐ Avoid nearby details
Distance [mm] 2
☐ Secondary search
☐ Edit

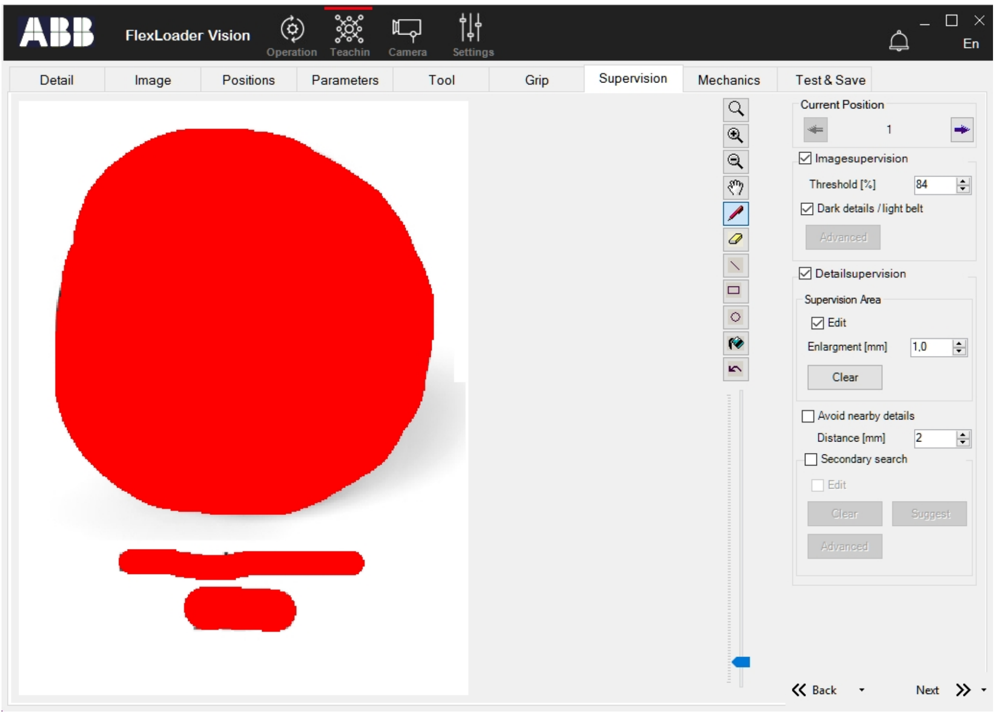
xx1800000224

Continues on next page

Detail supervision

Detail supervision can be used to provide a more advanced form of collision supervision. It is based on the details that are found by the system and therefore does not apply to details that are not found, for example details at the edge of the field of view or incorrect details. Therefore this technology is not comprehensive. Detail supervision is mainly useful when parts of details are difficult or impossible to make black. Detail supervision also permits more effective ways of picking more details.

To activate detail supervision, select **Detail supervision**. In order for the detail supervision to work fully, all parts of the detail that could cause a collision must be marked. Click **Edit** in the **Supervision area** pane to open an image of the teachin detail and mark all parts of the detail that may cause a collision in red, for each position. This information is used during operation. Even the parts of the position that are not visible are included in the collision sensing.



Current Position 1
☑ Imagesupervision
Threshold [%] 84
☑ Dark details / light belt
☑ Detailsupervision
Supervision Area
☑ Edit
Enlargment [mm] 1,0
☐ Avoid nearby details
Distance [mm] 2
☐ Secondary search
☐ Edit

xx1800000225

Avoid nearby details

When **Detail supervision** is enabled a further advanced setting can be used: **Avoid nearby details**. This setting can be used for the detail that is to be picked. If it is closer to any other detail than the specified distance it will not be picked. This is to avoid disturbing or knocking over other details on the camera belt. If all pickable details must be picked, this setting should be disabled.

Continues on next page

Secondary search

In certain cases the detail supervision is not sufficient to identify all collision risks, because only the details that have been found for picking are included in the result. The detail supervision can therefore be extended with a secondary search. This search is based on a user-defined masking that identifies more details in the field of view in order to improve the collision protection further. The details found in a secondary search will not be picked.

To activate the secondary search, select **Secondary search**, click **Edit** and add a mask to the details (in blue).

Secondary search cannot be activated for details with the analysis type **Blob**.

Note

Note that a secondary search significantly improves the collision protection, but does not provide an absolute protection.

Supervision settings for 3D cameras

Introduction

The supervision settings for 3D cameras are similar to the 2D camera settings, apart from the following details.

Image and detail supervision

The image supervision settings are always used when the 3D camera is run and are adjusted automatically.

The detail supervision works in exactly the same way with a 3D camera as with a 2D camera. Note that the drawn red area is always placed at the height where the detail has been identified.

Allowed overlap

If **Allowed overlap** is activated, the gripper is allowed to collide with objects in the image to a certain extent. The degree of collision permitted is specified in square millimetres. When tool collision is detected, FlexLoader Vision will compare the collision area with the specified, permitted overlap area.

If the collision covers less than the overlap area, the robot is permitted to pick the detail. For example, this can be used if the gripping fingers are permitted to knock away nearby details when picking.

4.10 Mechanics

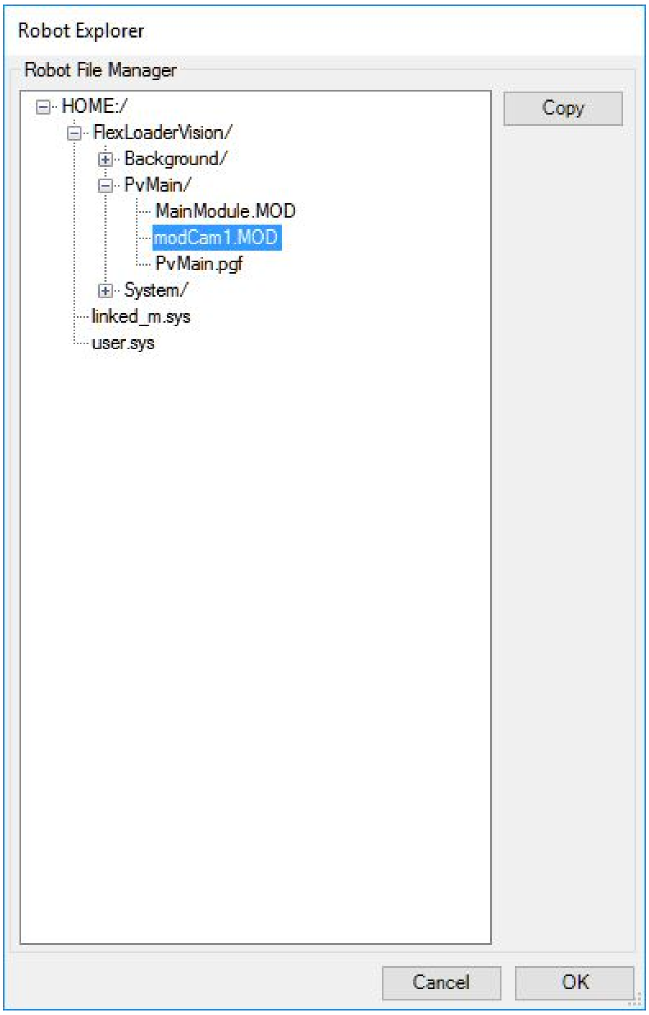
Introduction

What you see on the **Mechanics** tab varies from system to system, depending on the application and the type of robot that the system is configured for. See the product documentation of the concerned products for more information.

Robot program

When the system starts and stops, the program specified under **Robot program** is automatically loaded by the robot. Normally the program is loaded each time the system starts. If you want the robot to start with the program that is currently loaded in the robot, select **Run with currently loaded program**.

To specify a program, enter the path to the program manually in the **Robot program name** field or click **Browse** to browse for the program in the file manager. In the file manager, the **HOME** file system of the robot is shown together with available folders and files. Select the program to be used for the detail and click **OK** to use it. The path is now updated for the selected program.



xx1800000273

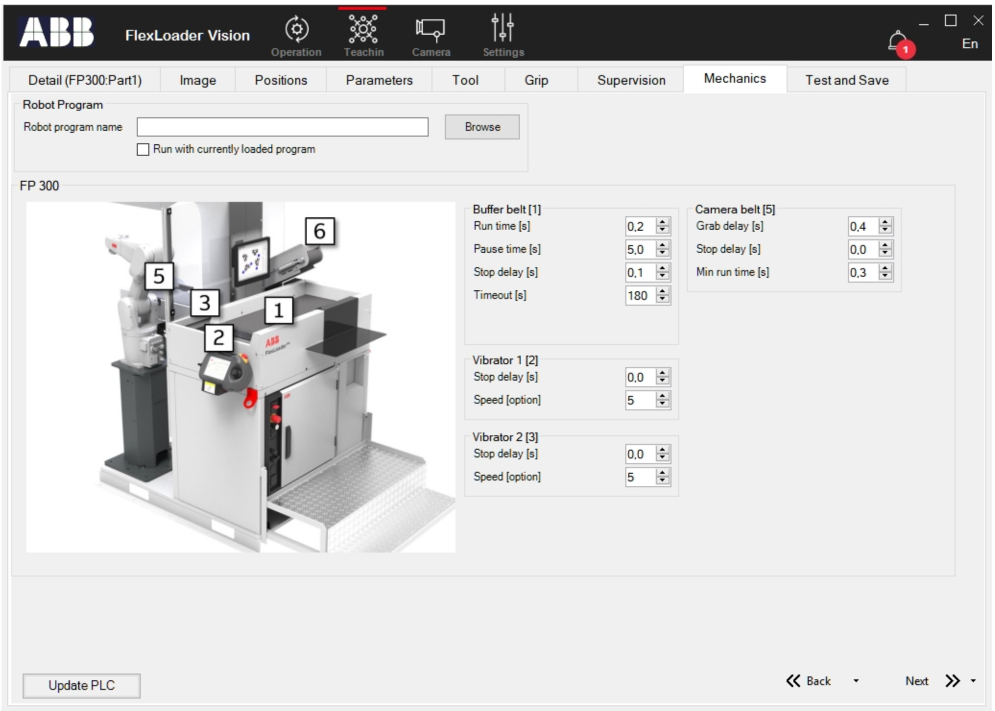
Continues on next page

To copy a program, select the program folder and click **Copy**. Enter a new name for the program folder and click **OK**. The copy is placed in the same folder as the original.

The module name itself can be changed from the FlexPendant (**Program Editor** -> **Modules** -> select module -> **File** -> **Change declaration**) or from RobotStudio.

FlexLoader FP 300

If FlexLoader Vision is used in FlexLoader FP 300, it is possible to adjust the parameters that govern mechanical control. On the **Mechanics** tab you will see an image of FlexLoader FP 300 and you will be able to set the parameters. The available parameters and their functions are described in the product manual of FlexLoader FP 300.



Robot Program
Robot program name
☐ Run with currently loaded program

Buffer belt [1]
Run time [s] 0,2
Pause time [s] 5,0
Stop delay [s] 0,1
Timeout [s] 180

Camera belt [5]
Grab delay [s] 0,4
Stop delay [s] 0,0
Min run time [s] 0,3

Vibrator 1 [2]
Stop delay [s] 0,0
Speed [option] 5

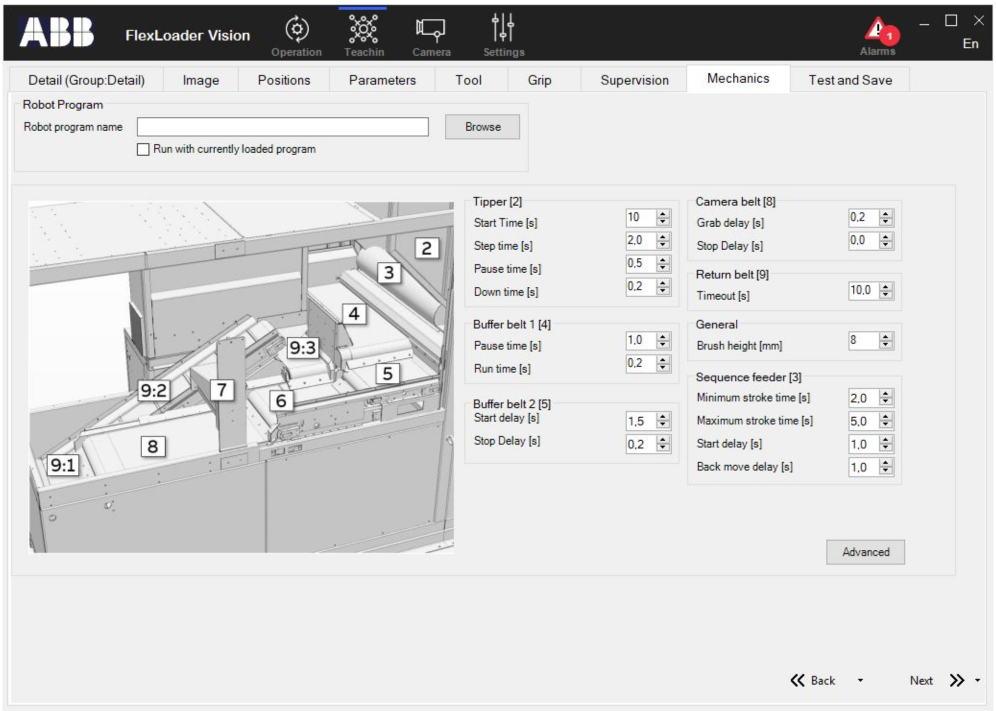
Vibrator 2 [3]
Stop delay [s] 0,0
Speed [option] 5

xx1800000227

Continues on next page

FlexLoader FP 400

If FlexLoader Vision is used in FlexLoader FP 400 it is possible to adjust the parameters that govern mechanical control. On the **Mechanics** tab you will see an image of FlexLoader FP 400 and you will be able to set the parameters. The available parameters and their functions are described in the product manual of FlexLoader FP 400.



Robot Program

Robot program name

□ Run with currently loaded program

Tipper [2]

Start Time [s]	10
Step time [s]	2,0
Pause time [s]	0,5
Down time [s]	0,2

Buffer belt 1 [4]

Pause time [s]	1,0
Run time [s]	0,2

Buffer belt 2 [5]

Start delay [s]	1,5
Stop Delay [s]	0,2

Camera belt [8]

Grab delay [s]	0,2
Stop Delay [s]	0,0

Return belt [9]

Timeout [s]	10,0

General

Brush height [mm]	8

Sequence feeder [3]

Minimum stroke time [s]	2,0
Maximum stroke time [s]	5,0
Start delay [s]	1,0
Back move delay [s]	1,0

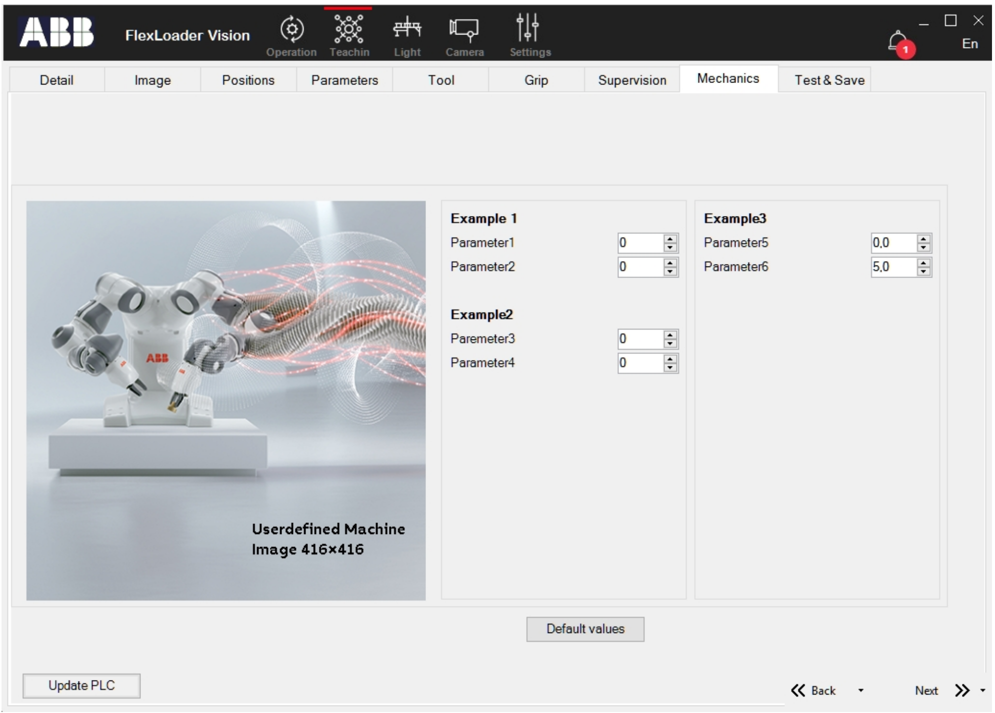
xx1800000226

Continues on next page

User-defined settings

If specified by the integrator, FlexLoader Vision can be used to control user-defined mechanical equipment. If so, the screen shows an image, two groups of parameters with a maximum of 32 values, and an **Update PLC** button to transfer these values to PLC. Refer to the product manual for the user-defined equipment for more information.

User-defined settings can also be used to transfer parameters to robot variables. This can with advantage be used when working with parametrized robot programs.



Example 1

Parameter1 0

Parameter2 0

Example2

Paremeter3 0

Parameter4 0

Example3

Parameter5 0,0

Parameter6 5,0

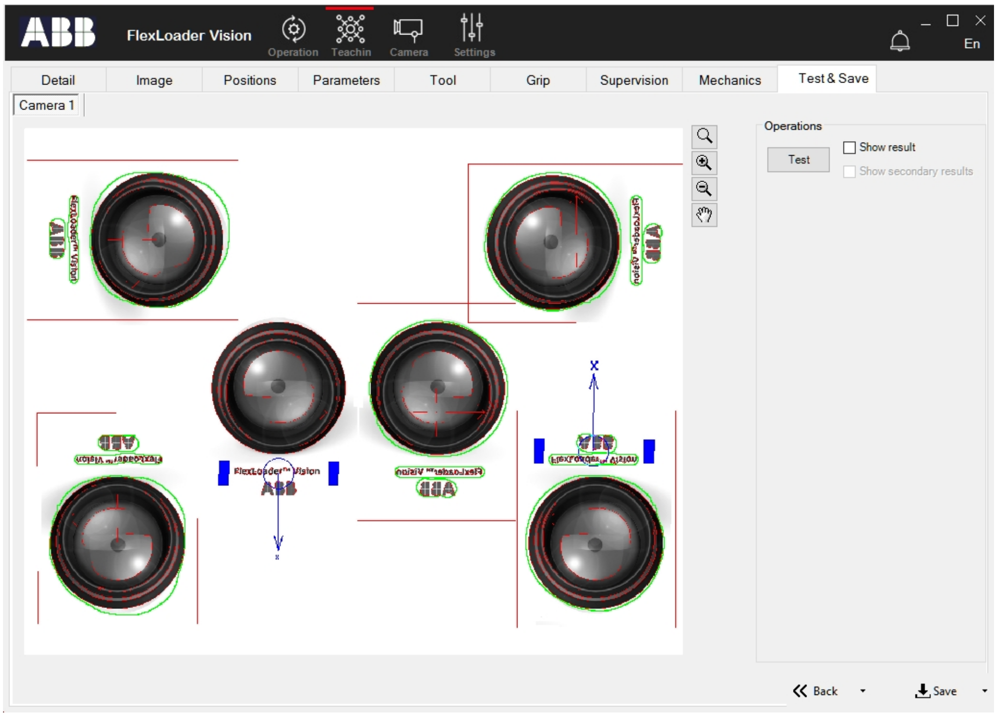
xx1800000228

For more information, see *User-defined mechanical equipment on page 186*.

4.11 Test & Save

On the **Test & Save** tab, the details are saved in the database. Before a detail is saved you have the opportunity to test how well the identification works.

Place a number of details in different positions under the camera and click **Test**. The identified positions and associated grip positions will be shown in the image.

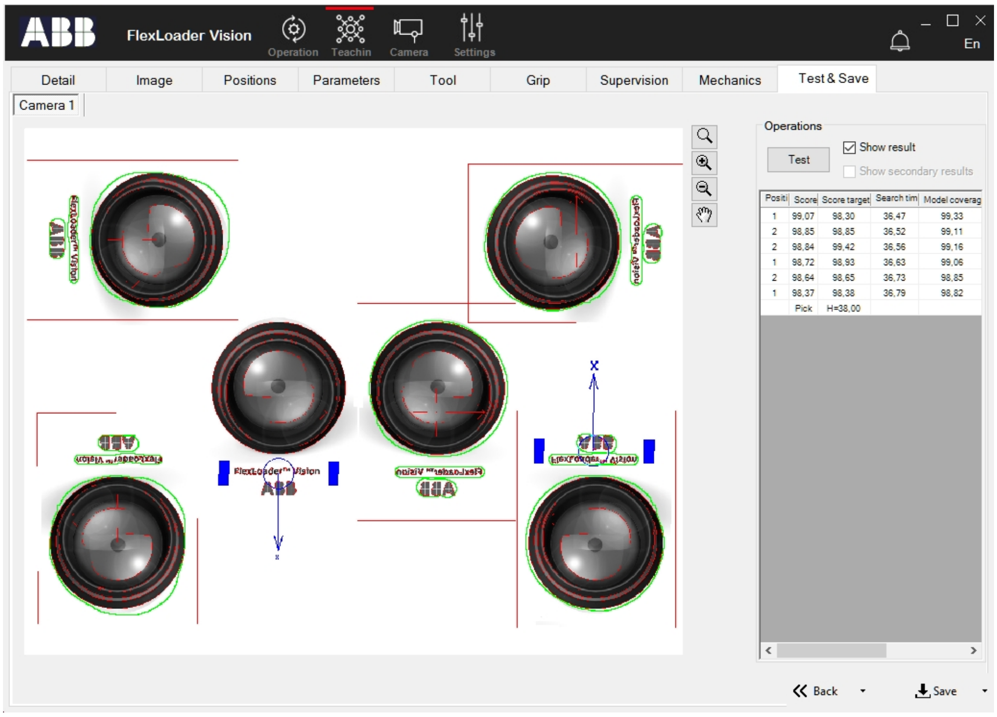


xx1800000229

In order to fine-tune the settings it is usually necessary to obtain more information about the identification results. Clicking on **Show result** opens a window with

Continues on next page

detailed test results for each identified detail. When a row is selected the associated detail is indicated by a blue circle in the field of view.



☑ Show result
□ Show secondary results

Positi	Score	Score target	Search tim	Model coverag
1	99,07	98,30	36,47	99,33
2	98,85	98,85	36,52	99,11
2	98,84	99,42	36,56	99,16
1	98,72	98,93	36,63	99,06
2	98,64	98,65	36,73	98,85
1	98,37	98,38	36,79	98,82
	Pick	H=38,00		

xx1800000230

The information shown in the image is the same as in normal operation.

The results of the secondary search can be displayed by selecting **Show secondary results.**

Click **Save** to save the detail in the database when all of the settings have been made. The **Detail** tab appears again.

If the **Blob** analysis type (optional setting) is selected for the detail, the properties of the found blobs are shown in percent in relation to the teachin position.

When FlexLoader Vision has identified a pickable detail, a new line is displayed in the list of test results. The new line is marked **Pick** and it specifies the data that would have been sent to the robot in normal operation. The data corresponds to the coordinates of the robot’s tool center point (TCP).

This page is intentionally left blank

5 Advanced teachin settings

5.1 Advanced image settings

Introduction

The teachin steps described in the previous section are usually adequate to provide a fully functional teachin setup. However, under certain circumstances it may be necessary to make additional settings for optimal teachin. Some of the additional settings available are described in the following chapter.

If it is difficult to obtain an image with good contrast, additional settings for adjusting the image can be accessed by clicking on **Advanced** on the **Image** tab.

Here the user can also configure detail specific movement supervision to prevent the details from moving after the images are taken.

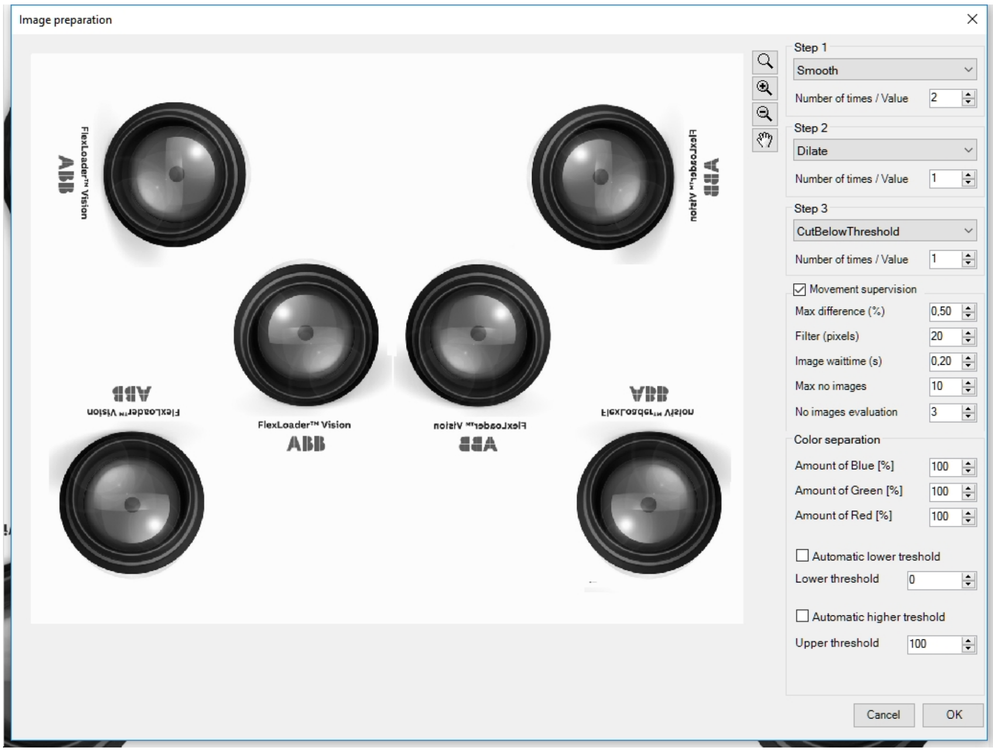


Image preparation

Step 1
Smooth
Number of times / Value 2

Step 2
Dilate
Number of times / Value 1

Step 3
CutBelowThreshold
Number of times / Value 1

☑ Movement supervision
Max difference (%) 0,50
Filter (pixels) 20
Image waittime (s) 0,20
Max no images 10
No images evaluation 3

Color separation
Amount of Blue [%] 100
Amount of Green [%] 100
Amount of Red [%] 100

□ Automatic lower treshold
Lower threshold 0

□ Automatic higher treshold
Upper threshold 100

xx1800000231

Filtering

With the advanced image settings you can radically change the appearance of an image by adding one or more filters before the image is analyzed further. This can be especially useful in cases where the image is very uniform or the detail is not clearly defined against the background, etc.

Up to three filter steps can be applied one after the other. Note that the results can differ widely depending on the order in which the filters are applied. This gives a large number of possible combinations. The effect of a filter is immediately seen

Continues on next page

in the image, which makes it easy to see how the different filters affect the image processing.

Filter name	Description
Smooth	Makes the image fuzzier and softens up the edges.
Sharpen	Makes the edges in the image more distinct.
SobelEdge	Creates a new image that reflects the edges in the original image.
VerticalEdge	Similar to the **SobelEdge** filter but primarily reproduces the vertical edges in the image.
HorizontalEdge	Similar to the **SobelEdge** filter but primarily reproduces the horizontal edges in the image.
LaplacianEdge	Similar to the **SobelEdge** filter, but the edges are made thinner and softer. This can be useful if the original image has a very high contrast.
CutBelowThreshold	Converts all of the pixels below a certain light level to black. This light level is set as a threshold value by the users.
CutAboveThreshold	Converts all of the pixels above a certain light level to white. This light level is set as a threshold value by the users.
BinzarizeAtThreshold	Divides the image into areas of black and white. All of the pixels above a certain light level are converted to white and the rest are converted to black. This light level is set as a threshold value by the users.
Invert	Inverts the image, so that all the light pixels are made dark and vice versa.
Erode	Reduces the size of light areas by deleting a small area along the entire edge.
Dilate	Increases the size of the light areas by adding a small area along the entire edge.
BlockPickedParts	A specialized filter that simplifies the identification of details in several cases. If the coordinates of a detail are sent to the robot, a circle with the specified parameter will be drawn in the subsequent images at the location where the detail is identified. This filter selection is only valid for the first filtering set. The **BlockPickedParts** filter is typically used for details in trays, where the shape of the empty tray often is similar to a detail and can erroneously be identified as a detail. Another scenario is when a machined detail that is very similar to the raw part is put back at the same position as the raw part was picked. Resetting the picked parts collection is done by the robot, by sending the **ResetPickedPositions** command.

Movement supervision

The movement supervision helps prevent details from moving after the image has been taken. This prevents, as far as possible, the robot from gripping for details that have shifted positions, or from colliding with details that have moved under the gripper.

The movement supervision takes two images of the image field with a small time difference and compares them. If there are differences between these images, the

Continues on next page

system assumes that something has moved in the image and therefore takes a new image.

Max difference (%)

This parameter indicates what the largest difference can be between the following images. If the images differ more, a new image will be taken. The smaller the details are in relation to the image field, the lower the value must be.

Filter (pixels)

The differences between the images are filtered to minimize the disruptive influence of, for example, noise, dirt or glare. The greater the value, the more the details must be moved for the movement supervision to be activated.

Image waittime (s)

This is the waiting time between two images. The slower the details roll, the longer the time period should be.

Max no images

Specifies the maximum amount of images to take. When the system has taken the maximum number of images, the images will be analyzed, regardless of whether any movement has been detected or not.

No images evaluation

Specifies the number of images that should be compared. Every picture is compared to all other pictures.

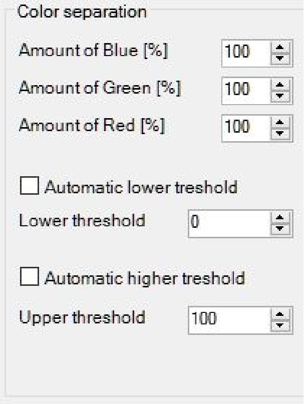
Color settings

If a color camera is used, the **Color separation** pane is displayed. Here you can determine how to combine the three colors applied to the black and white image that FlexLoader Vision is to analyze. This function can be used to enhance or weaken certain selected colors, or to increase or reduce the contrast between certain colors.

For each included color (blue, red, green) the proportion that is used to generate the final black-white image is set. Note that it is also possible to subtract colors from each other by giving negative percentages.

Continues on next page

The calculated black-white image can then be rescaled, either by allowing FlexLoader Vision to automatically determine the upper and lower limits (thresholds), or by specifying certain values.



Color separation

Amount of Blue [%] 100

Amount of Green [%] 100

Amount of Red [%] 100

☐ Automatic lower treshold

Lower threshold 0

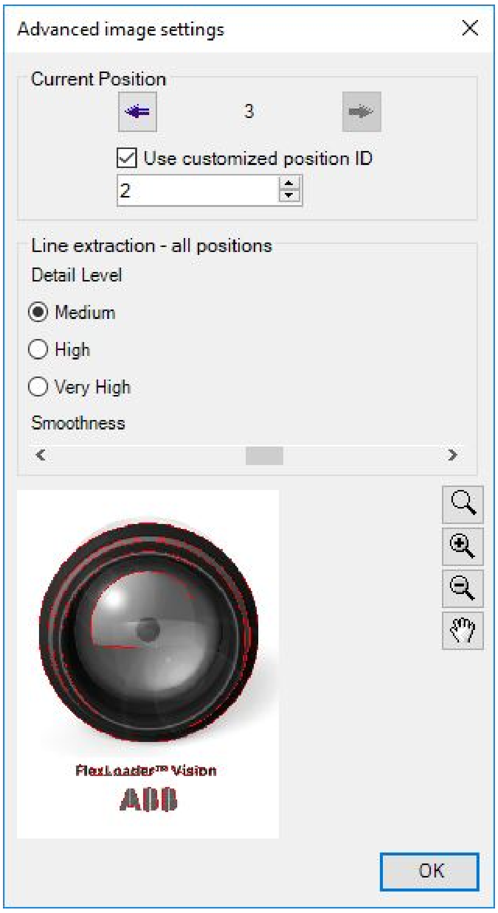
☐ Automatic higher treshold

Upper threshold 100

xx1800000232

5.2 Advanced position settings

The advanced position settings allow you to control how the system calculates edges and the geometrical shape of a position. The settings are accessed by clicking on **Advanced** on the **Positions** tab.



Advanced image settings

Current Position

3

☑ Use customized position ID

2

Line extraction - all positions

Detail Level

● Medium

○ High

○ Very High

Smoothness

xx1800000233

The user can select the option **Use customized position ID**, so that a different position ID can be entered. This customized ID will be sent to the robot instead of the actual position number. This feature can be useful when the same physical positions is taught several times for slightly different appearances of a part.

The **Detail level** settings govern how the edges are extracted from the image. Normally you should not change these settings, but if you have an image with low contrast it may be beneficial to change this setting to **High**. The slider can also be used to make further adjustments to edge detection. **Very high** should normally only be used in exceptional cases, and then only on images with very low contrast. If this parameter is set to **Very high** it can make edge detection less reliable and extremely sensitive to lighting or the presence of dirt. The effect of changing the settings is updated continuously in the image.

Note that the **Smoothness** settings affect every position in the system.

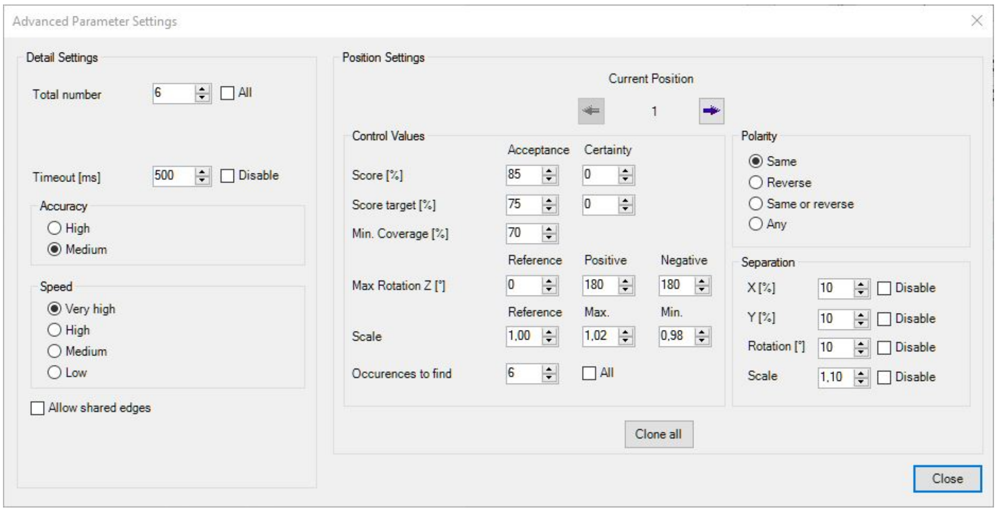
5.3 Advanced parameter settings

5.3.1 Standard analysis

Introduction

Clicking on **Advanced** on the **Parameters** tab when the standard analysis type is used gives access to additional settings for optimizing the identification of details.

The advanced parameter settings are divided into those that apply to the detail, i.e. all positions, and those that only apply to the current position. You can browse through the defined positions with the **Current position** arrows. If you want to apply the same settings to all positions this can be done quickly by clicking on **Clone all**.



Advanced Parameter Settings

Detail Settings
- Total number: 6 □ All
- Timeout [ms]: 500 □ Disable
- Accuracy: ○ High ● Medium
- Speed: ● Very high ○ High ○ Medium ○ Low
- □ Allow shared edges

Position Settings
- Current Position: 1

Control Values

	Acceptance	Certainty
Score [%]	85	0
Score target [%]	75	0
Min. Coverage [%]	70	

	Reference	Positive	Negative
Max Rotation Z [°]	0	180	180

	Reference	Max.	Min.
Scale	1,00	1,02	0,98

- Occurences to find: 6 □ All

Polarity: ● Same ○ Reverse ○ Same or reverse ○ Any

Separation

X [%]	10	□ Disable
Y [%]	10	□ Disable
Rotation [°]	10	□ Disable
Scale	1,10	□ Disable

xx1800000234

Detail Settings

The following settings apply at detail level, which means that they affect all the teachin positions for the detail.

Total number

The **Total number** setting can be used to limit the number of positions that are to be found. Searching for all the positions available takes more time than just searching for one. Searching for just one detail is faster, but could on the other hand result in finding a detail that cannot be gripped. A suitable starting value is 6. If the search takes too long, this is one of the parameters you could adjust.

To send

The **To send** value (if available) specifies how many results are sent to the robot at most. Sending too many details would mean that the robot would have to process and check considerably more coordinates than it is able to pick. In most cases the value of **Total number** should be set relatively high, while the value of **To send** can be set relatively low. This is because a number of the details that are found may be impossible to grip due to collision detection.

Continues on next page

The robot can also sort and select which details are to be picked when several occurrences are sent to the robot, for example based on the signals from the machine used in production.

Timeout

The **Timeout** parameter specifies a maximum period of time used for searching. An image taken against a dirty conveyor belt often contains a lot of noise. This noise results in a large number of edges being found, even in areas where there are no details. Processing all these edges can take relatively long. With the **Timeout** parameter, the maximum search time can be limited so that the identification time does not affect the robot cycle time. If no timeout is set at all, the search could take several minutes in the worst case.

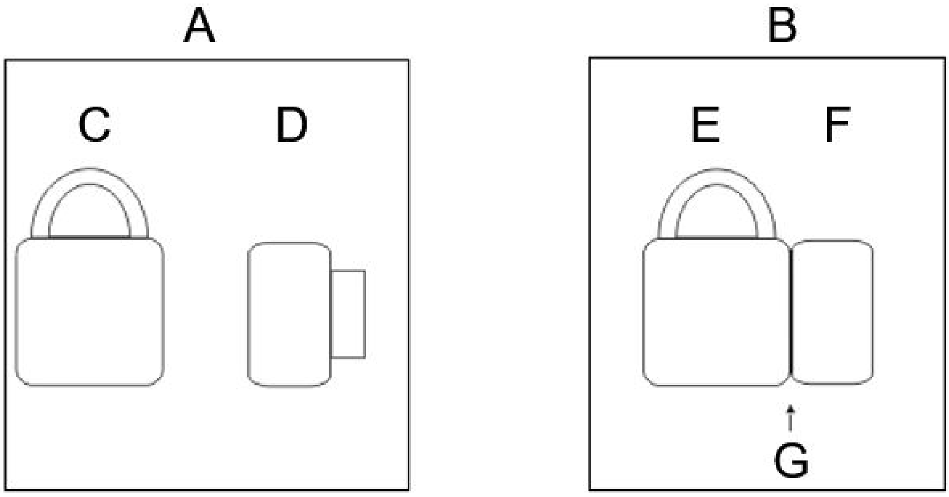
Accuracy and Speed

The **Accuracy** and **Speed** settings can be used to customize the search. However, you should remember that a higher speed will have a negative effect on the accuracy, and a higher accuracy will have a negative effect on the speed. The speed and accuracy settings should therefore be balanced against each other and adapted to suit the current application. Setting both the speed and accuracy settings to **High** will have the same result as setting them both to **Medium**.

To achieve a longer cycle time, the accuracy can be increased and the speed decreased. To achieve shorter cycle times, the speed should be increased and the accuracy decreased. In these cases the handling tools should be designed so that they are more tolerant of varying results.

Allow shared edges

If shared edges are allowed, a found edge can be counted as a result for two or more found positions. If shared edges are not allowed, shared edges are counted as belonging to the result that meets the requirements most closely.



xx1800000962

Pos.	Description
A	Teachin position

Continues on next page

Pos.	Description
B	Found positions
C	Position 1
D	Position 2
E	Position 1, score 100%
F	Position 2, score 75%
G	Shared edges

In the image above, allowing shared edges would mean that the positions in box B are found, with scores of 100% (E) and 75% (F) respectively. If you do not allow shared edges, position 1 will still be found, with a score of 100%, while the score for position 2 will be further reduced. This is because an additional edge is missing from the found position when compared to the teachin position.

Position Settings

The following settings apply at position level, i.e. they only affect the current position. It is easy to switch between positions: just click the arrows to change position.

Score

The **Acceptance** value specifies the minimum value (%) for identification of a position. It corresponds to the **Score** value previously specified on the **Parameter** tab. When the system is searching for a position and finds it with a score of 85%, for instance, it will continue searching for a better match until all other possibilities have been ruled out, or until it reaches timeout. To speed up the identification further, use the **Certainty** setting. The **Certainty** parameter is used to end the search process as soon as a position that meets the certainty setting is found.

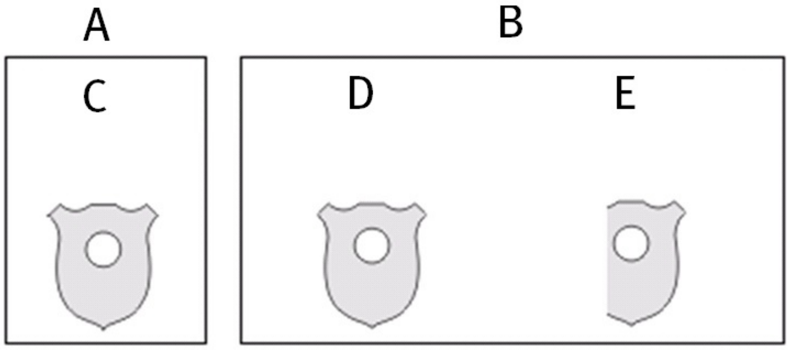
Score target

The **Acceptance** value specifies the minimum value (%) for identification of a position. It corresponds to the **Score target** value, previously specified on the **Parameter** tab. When the system is searching for a position and finds it with a score target of 85%, for instance, it will continue searching for a better match until all other possibilities have been ruled out, or until it reaches timeout. To speed up the identification further, use the **Certainty** setting. The **Certainty** parameter is used to end the search process as soon as a position that meets the certainty setting is found.

Continues on next page

Min. coverage

The **Min. coverage** parameter lets you adjust the system so that a certain proportion of the position must be visible in order to be considered a match.



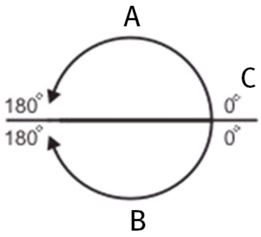
xx1800001032

Pos.	Description
A	Teachin position
B	Found position
C	Position 1
D	Position 1, coverage 100%
E	Position 1, coverage 75%

The image shows how the minimum coverage affects the search result. Note that the found position with 75% coverage will also result in a lower **Score** value.

Max. Rotation Z

The **Max. Rotation Z** parameter can be used to restrict the system to search for a position within a given range of angles.



xx1800001046

Pos.	Description
A	Positive
B	Negative
C	Reference

Zero degrees corresponds to the angle of the detail during teachin. By changing the reference point and/or the positive and negative directions you can control the permitted range of angles for identification. Often you may just want to set limits

Continues on next page

on a certain angle range in the robot's coordinate system. In these cases you should define new grip limitations instead of using this parameter.

Scale

By using the **Scale** parameter you can instruct the system to look for a position within a given size range. If the maximum setting is set at 1.02, the minimum setting is set at 0.98 and the reference setting is set at 1, it means that a position can be found within a size interval of ± 2% relative to the teachin position. Often this parameter can be used to instruct the system not to find positions that differ in size from the teachin position. In this case it is a good idea to set the tolerance quite tightly, e.g. 1.01 for maximum and 0.99 for minimum if the reference setting is 1.

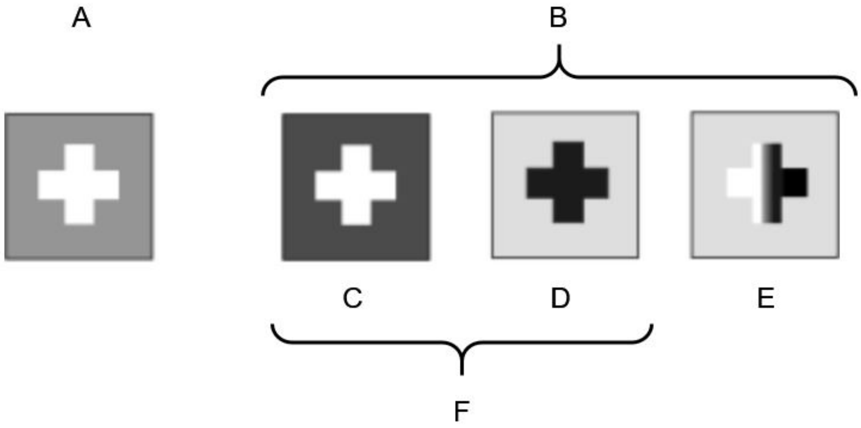
Occurrences to find

The **Occurrences to find** parameter in the **Positions Settings** pane interacts with the **Total number** parameter in the **Detail Settings** pane. If the **Total number** set for a detail is lower than the **Occurences to find** value that is set for positions, the maximum number that can be found will be decided by the **Total number** parameter for the detail.

If a detail teachin has two positions specified, each of which can have four occurrences, and the **Total number** to find is set at six, the maximum number of occurrences that can be found is six.

Polarity

Polarity specifies how the transfer between dark and light areas occurs. This makes it possible to control whether to have the same transfer between the different areas during operation as during teachin, or not.



xx1800000309

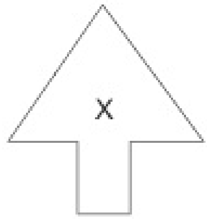
Pos.	Description
A	Teachin position
B	Found positions
C	**Polarity: Same**
D	**Polarity: Reverse**

Continues on next page

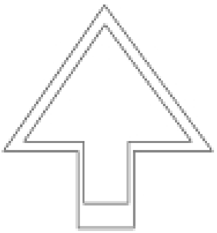
Pos.	Description
E	**Polarity: Any**
F	**Polarity: Same or reverse**

Separation

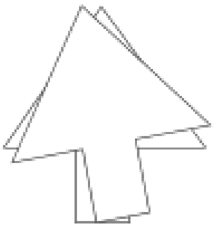
With the parameters for separation (distance) you can control how close different occurrences of details may be. You can request a minimum distance in x or y coordinates, in scale factor, or in rotation angle. You can also select to ignore separation requirements for certain parameters by selecting the **Disable** checkbox.



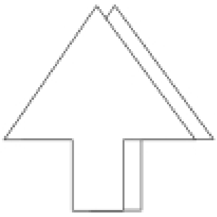
Teachin detail example.



Two occurrences of details that have the same angle and position, but differ in size. If the difference is bigger than the specified scale value, or if the **Disable** checkbox has been selected, both occurrences are found.



Two occurrences of details that have the same position and size, but differ in angle. If the difference is bigger than the specified angle value, or if the **Disable** checkbox has been selected, both occurrences are found.



Two occurrences of details that have the same angle and size, but differ in the x or y position. If the difference is bigger than the specified x or y position value, or if the **Disable** checkbox has been selected, both occurrences are found.

The separation parameters works differently for different analysis steps. The initial separation check is performed by comparing the centers of the boxes surrounding each occurrence. The second separation check is based on the actual TCP position that is determined in the serach.

Continues on next page

For each position, only occurrences separated according to the conditions defined above will be found during the initial search.

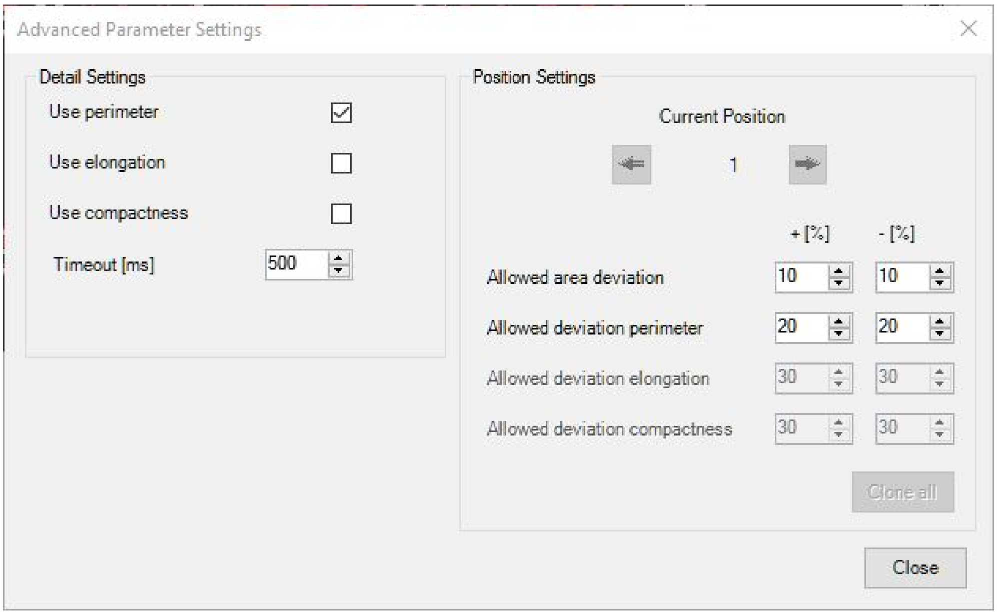
Different positions can initially be found without taking separation into account, but a filter is then applied which removes occurrences that violate the defined separation rules.

When locating parts in 3D, a filter is applied to all occurrences found at different seek depths, which also removes occurrences that violate the defined separation rules.

5.3.2 Blob analysis

Introduction

Clicking on **Advanced** on the **Parameters** tab when the blob analysis type is used, gives access to additional settings for optimizing the identification of blobs.



Advanced Parameter Settings

Detail Settings

Use perimeter ☑

Use elongation □

Use compactness □

Timeout [ms] 500

Position Settings

Current Position 1

	+ [%]	- [%]
Allowed area deviation	10	10
Allowed deviation perimeter	20	20
Allowed deviation elongation	30	30
Allowed deviation compactness	30	30

xx1800000235

Detail Settings

The following settings apply at detail level, i.e. they affect all the teachin positions for the detail.

Use perimeter

Use perimeter must be selected if the perimeter is to be included in the properties that are analyzed and compared with the teachin details. Tick this setting to set the permitted deviations for perimeter.

Use elongation

Use elongation must be selected if the elongation is to be included in the properties that are analyzed and compared with the teachin details. Tick this box to set the permitted deviations for elongation.

Use compactness

Use compactness must be selected if the compactness is to be included in the properties that are analyzed and compared with the teachin details. Tick this box to set the permitted deviations for compactness.

Continues on next page

To send

The **To send** value (if available) specifies how many results are sent to the robot. Sending too many details would mean that the robot would have to process and check considerably more coordinates than it is able to pick. In most cases the value of **Total number** should be set relatively high, while the value of **To send** can be set relatively low. This is because a number of the details that are found may be impossible to grip due to collision detection.

The robot can also sort and select which details are to be picked when several occurrences are sent to the robot, for example based on the signals from the machine used in production.

Timeout

The **Timeout** parameter specifies a maximum period of time used for searching. An image taken against a dirty conveyor belt often contains a lot of noise. This noise results in a large number of edges being found, even in areas where there are no details. Processing all these edges can take relatively long. With the **Timeout** parameter, the maximum search time can be limited so that the identification time does not affect the robot cycle time. If no timeout is set at all, the search could take several minutes in the worst case.

Position Settings

The following settings apply at position level, i.e. they only affect the current position. It is easy to switch between positions by clicking on the arrows to change position.

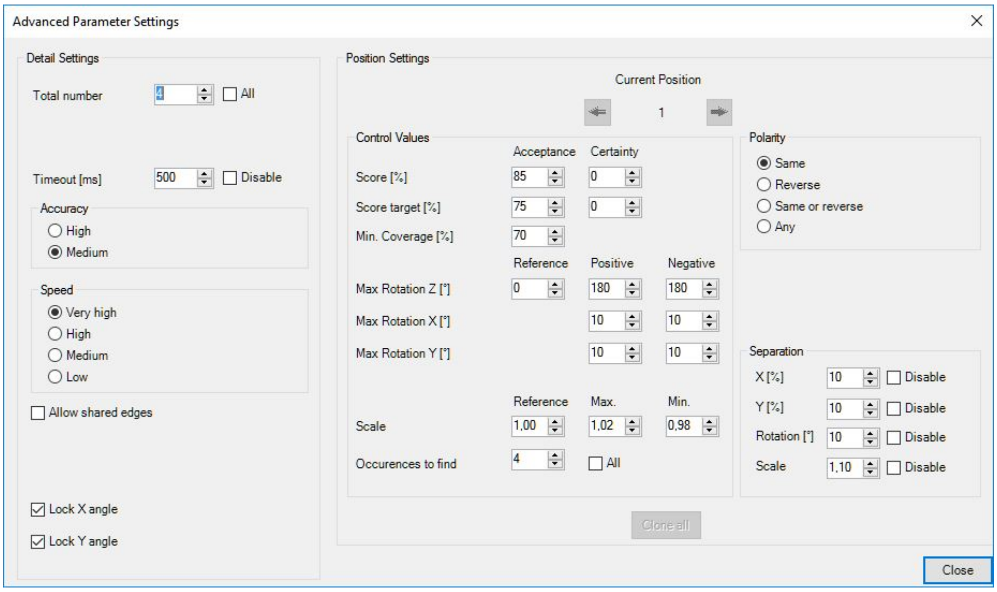
Permitted deviation for different properties

For all properties, an upper (maximum deviation) and lower (minimum deviation) limit is specified as a percentage. The requirements for the two positions must be completely different in at least one aspect, otherwise a warning is displayed.

5.3.3 3D analysis

Introduction

The advanced parameter settings for 3D analysis are very similar to those for the standard analysis (described above). The difference is that there are some extra settings for 3D analyses. These are described below.



Advanced Parameter Settings

Detail Settings

Total number: 4 □ All

Timeout [ms]: 500 □ Disable

Accuracy: ○ High ● Medium

Speed: ● Very high ○ High ○ Medium ○ Low

□ Allow shared edges

☑ Lock X angle

☑ Lock Y angle

Position Settings

Current Position: 1

Control Values

	Acceptance	Certainty
Score [%]	85	0
Score target [%]	75	0
Min. Coverage [%]	70	

	Reference	Positive	Negative
Max Rotation Z [°]	0	180	180
Max Rotation X [°]		10	10
Max Rotation Y [°]		10	10

	Reference	Max.	Min.
Scale	1,00	1,02	0,98
Occurences to find	4	□ All	

Polarity: ● Same ○ Reverse ○ Same or reverse ○ Any

Separation

X [%]	10	□ Disable
Y [%]	10	□ Disable
Rotation [°]	10	□ Disable
Scale	1,10	□ Disable

xx1800000236

Detail Settings

Lock X angle

Locks the rotation around the x-axis to 0 degrees. However, the additional rotation stated on the **Grip** tab will be even if you select this box. This means that the robot will always pick with the same rotation around x, that is stated on the **Grip** tab. Note that this concerns the x-axis for the detail.

Lock Y angle

Locks the rotation around the y-axis to 0 degrees. However, the additional rotation stated on the **Grip** tab will be added even if you select this box. This means that the robot will always pick with the same rotation around y, that is stated on the **Grip** tab. Note that this concerns the y-axis for the detail.

Control values

Max Rotation X

Specifies the maximum rotation around the x-axis of the detail. If the detail's angle in the x-axis together with the additional rotation that is specified on the **Grip** tab exceeds the maximum rotation, the detail will be marked as unpickable.

Continues on next page

Max Rotation Y

Specifies the maximum rotation around the y-axis of the detail. If the detail’s angle in the y-axis together with the additional rotation that is specified on the **Grip** tab exceeds the maximum rotation, the detail will be marked as unpickable.

5.3.4 AddOn analysis

Introduction

The so called AddOn analyses are extra analyses that can be enabled by the integrator. They can be used to further investigate different positions found by the system. The AddOn analyses are extremely flexible and can be used in many different ways. Possible applications include refining the search results, distinguishing between almost identical positions and checking whether parts of the detail exist or not.

To access the AddOn analyses (if enabled), click **AddOnAnalysis** on the **Parameters** tab on the **Teachin** menu.

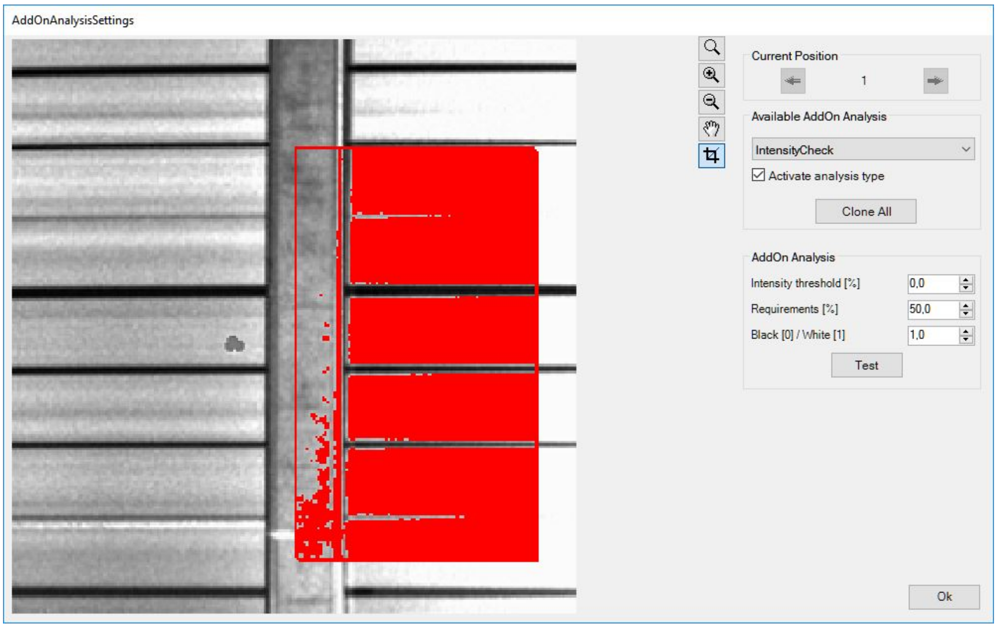
The AddOn analyses are defined for every teachin position. Select position with the **Current position** arrows and then select one of the available analysis types in the drop-down list. Confirm that you want to use the analysis type by selecting **Activate analysis type**. You can activate as many analyses as you would like.

If the conditions for any of the activated AddOn analyses are not met for a found detail, the detail will not be picked by the robot. Different parameters can be specified for each analysis type. In addition to this, you can specify the area in which the AddOn analyses are to be performed by drawing a red rectangle on the position image.

After specifying all parameters and the area where the AddOn analysis is to be performed, you can verify the settings by clicking **Test**. The result is shown in the image (certain areas are marked in red).

Intensity check

The analysis type **Intensity check** (if enabled) is used to specify that selected areas should be light or dark. This could be used, for example, to check for material, openings, or labels.


AddOnAnalysisSettings

Current Position: 1

Available AddOn Analysis: IntensityCheck

☑ Activate analysis type

AddOn Analysis
Intensity threshold [%]: 0,0
Requirements [%]: 50,0
Black [0] / White [1]: 1,0

xx1800000237

Continues on next page

Parameter	Description
Black[0] / White [1]	Indicates whether the operator is interested in the light (1) or dark areas (0). During testing, the pixels above the intensity threshold (1) or those below the intensity threshold (0) are marked.
Intensity threshold	Indicates at which brightness level the pixels are included in the analysis. 100% corresponds to a completely white pixel and 0% to a completely black pixel.
Requirements	Specifies what percentage of the pixels must meet the conditions above.
Logg Off [0] / Logg On [1]	Results will be logged continuously to a file (Addon-Analysis.log). Useful for identifying clusters of values and setting well defined requirements.

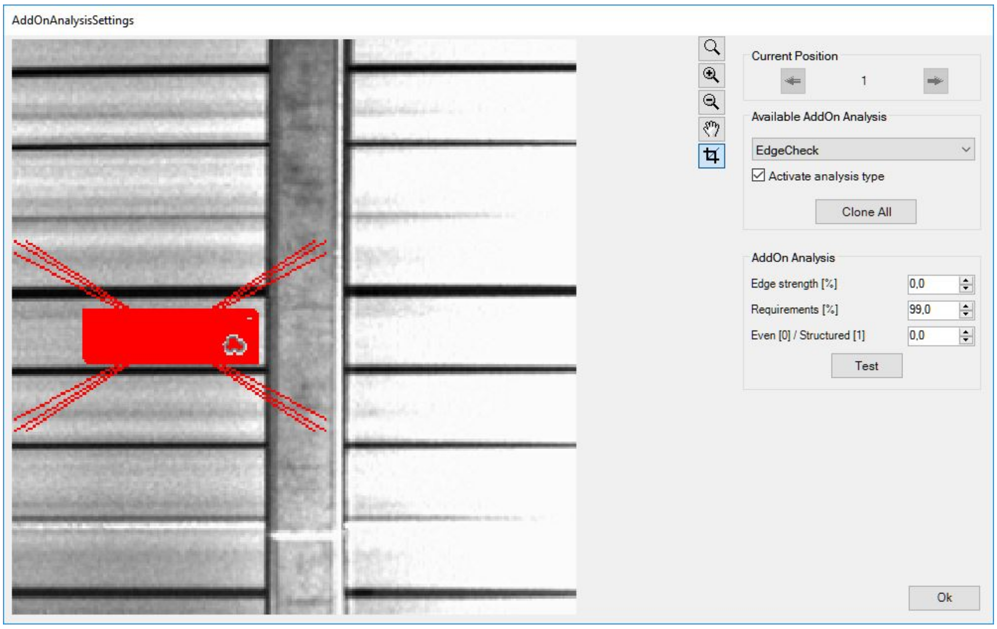
Example: **Black[0] / White [1]** = 1, **Requirements** = 40%, **Intensity threshold** = 80%

An analysis with the above settings would include all pixels within the red box with an brightness intensity above 80%. If more than 40% of the pixels in the red box meet the conditions, the analysis is approved.

The actually obtained result is shown for information.

EdgeCheck

The analysis type **EdgeCheck** (if enabled) is used to ensure that the selected parts of the detail position consist of a certain amount of edges, i.e. visible structures. This could be used, for example, to check for holes, embossing, or large defects on a flat surface.



EdgeCheck

☑ Activate analysis type

Edge strength [%] 0,0

Requirements [%] 99,0

Even [0] / Structured [1] 0,0

xx1800000238

Continues on next page

Parameter	Description
Even [0] / Structured [1]	Indicates whether the operator is interested in flat areas or structured areas of the surface. During testing, the pixels with an edge strength below the edge intensity threshold (1) or those above the edge intensity threshold (1) are marked.
Edge strength	Specifies how visible and distinctive an edge in the image must be. 100% corresponds to an immediate transition from black to white in the image, while 0% corresponds to an even brightness in the area. The threshold indicates at which edge strength the pixels in the analysis are calculated.
Requirements	Specifies what percentage of the pixels must meet the conditions above.
Standard [0] / Deriche [1]	Specifies how edges are extracted from the image. Standard extraction works best for well defined edges on smooth backgrounds. Deriche extraction is more suitable for extracting edges on slightly varying background.
Smoothness [%]	Specifies how much the background will be smoothed before applying the Deriche edge extraction.
Logg Off [0] / Logg On [1]	Results will be logged continuously to a file (Addon-Analysis.log). Useful for identifying clusters of values and setting well defined requirements.

Example: **Even [0]/ Structured [1]** = 1, **Requirement** = 10%, **Edge strength** = 50%

An analysis with the above settings would include all pixels within the red box with an edge strength above 50%. If more than 10% of the pixels in the red box meet the conditions, the analysis is approved.

The actually obtained result is shown for information.

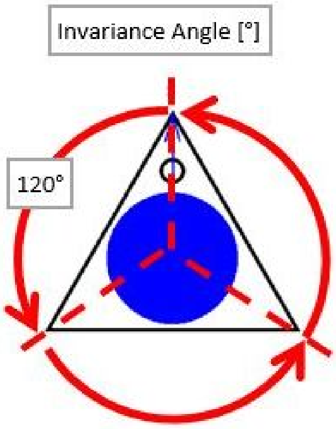
InvariantEdgeFinder

The analysis type **InvariantEdge** (if enabled) is used to find features on parts with a certain rotational symmetry. It helps establishing which of the several possible rotations the part has and modifies the gripping result according to the found feature.

The parameter settings correspond to the **EdgeCheck** parameters, always using the Deriche edge extraction.

Continues on next page

In order to accept a valid pickable part, the condition must be fulfilled exactly once for the possible angles.



xx2000000635

Parameter	Description
Parameters as in **EdgeCheck**	
Invariance angle [°]	Specifies the rotation invariance for the selected parts (120° for equilateral triangle, 90° for squares, 60° for hexagons).

The actually obtained result is shown for information.

Tip

The gripping point (TCP) has to placed exactly in the rotation point.

CircleFinder

The analysis type **CircleFinder** (if enabled) is used to find circle features with a specified diameter.

Parameter	Description
Diameter [mm]	Specifies the desired diameter of the circle to be found.
Max deviation [%]	Specifies accepted deviations from the nominal diameter. Max allowed deviations are -50% and +100%.
Smoothnes [%]	Specifies how much the edges shall be smoothed prior to circle analysis. A typical value is 50%.
Detail level [1,2,3]	Specifies how distinct the extracted edges can be. 1: Very strong, 2: Medium, 3: Very weak.
Acceptance [%]	Required degree of correspondence between image and circle.
Number	Required number of circles to be found.

The actually obtained result is shown for information.

Continues on next page

CodeFinder

The analysis type **CodeFinder** (if enabled) is used to find numerical barcodes or matrix codes of a specified type in the image.

Parameter	Description
Code type	101: EAN13, 102: EAN8, 103: Code39, 104: UPCA, 105: UPCE, 106: Interleaved2to5, 107: Code93, 108: Code128, 109: Codebar, 110: Industrial25 201:DATAMATRIX, 202:QR, 203:PDF417, 204:AZTEC,205:MICROPDF417, 206:MICROQR, 207:MAXICODE
Requirement	Specifies what the code should be (numerical). Set to 0 to accept any successful code read.

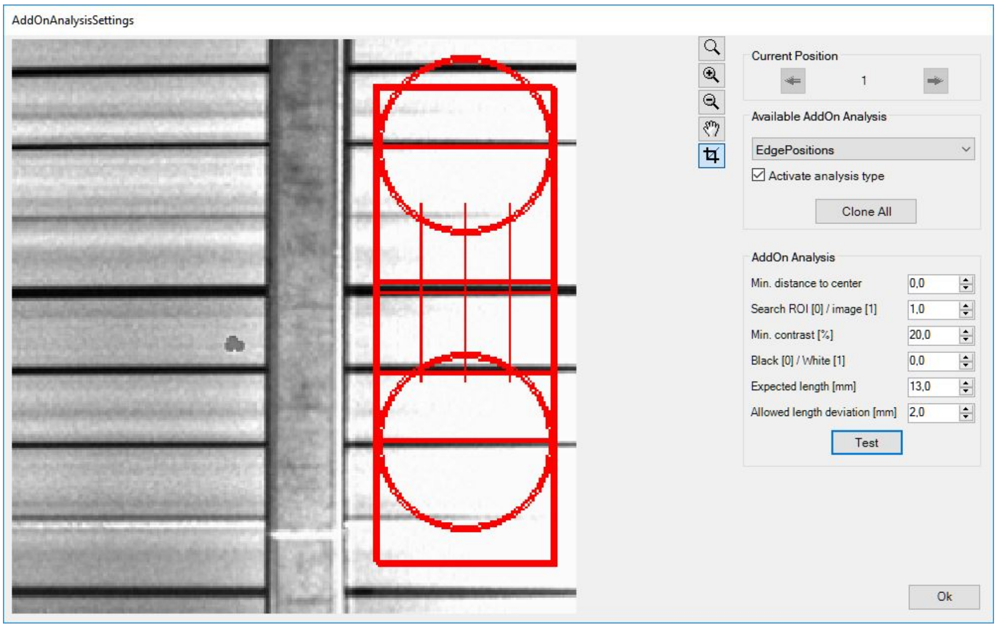
The actually obtained result is shown for information.

EdgePositions

The analysis type **EdgePositions** (if enabled) is used to check the length and width of a detail and to adjust its center position.

First, the length of the object is measured by locating the end points according to the operator settings. Next, the center position of the found detail is adjusted to the center of the detail, regardless of where it was initially found.

In the red box, the original center position is drawn with a dash in the center. The found end points are drawn with circles around a dash that marks the end point. The revised center position is drawn with several plus signs (+).


AddOnAnalysisSettings

Current Position: 1

Available AddOn Analysis: EdgePositions

☑ Activate analysis type

Clone All

AddOn Analysis

Min. distance to center: 0,0
Search ROI [0] / image [1]: 1,0
Min. contrast [%]: 20,0
Black [0] / White [1]: 0,0
Expected length [mm]: 13,0
Allowed length deviation [mm]: 2,0

Test

Ok

xx1800000239

Continues on next page

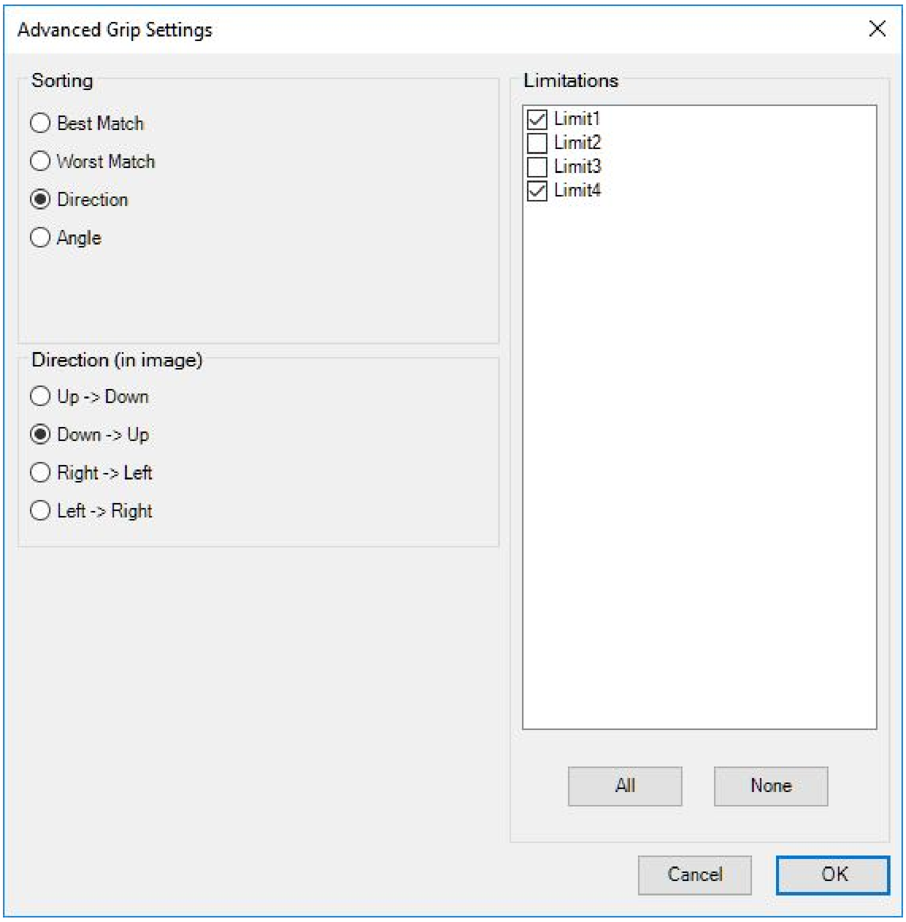
Parameter	Description
Min. distance to the center	Indicates how many pixels from the center the search for the end point is to start. With this parameter, the search can skip over uninteresting areas. The area in the center is also used as an average value of brightness that can be used in the edge search outwards.
Search ROI [0] / image [1]	Indicates how the red box affects the edge searching: The following values are permitted: 0: Edge searching is only performed within the red box. 1: Edge searching is performed in the entire image, longitudinally along the red box, over the entire area as the image permits. This setting may be useful if the first search does not position the detail correctly longitudinally.
Min. contrast [%]	Indicates the minimum brightness change that is required to assume there is an edge.
Black [0] / White [1]	Indicates whether the background is darker than the detail (0) or lighter than the detail (1).
Expected length [mm]	Indicates the expected length of the detail in mm.
Allowed length deviation [mm]:	Indicates by how much the length of the detail may deviate up or down to still be approved.

The actually obtained result is shown for information.

5.4 Advanced grip settings

Introduction

Clicking on **Advanced** on the **Grip** tab gives access to additional grip adjustment settings. These allow you to adjust the sorting and to select gripping limitations.



Advanced Grip Settings

Sorting
○ Best Match
○ Worst Match
● Direction
○ Angle

Direction (in image)
○ Up -> Down
● Down -> Up
○ Right -> Left
○ Left -> Right

Limitations
☑ Limit1
□ Limit2
□ Limit3
☑ Limit4

xx1800000240

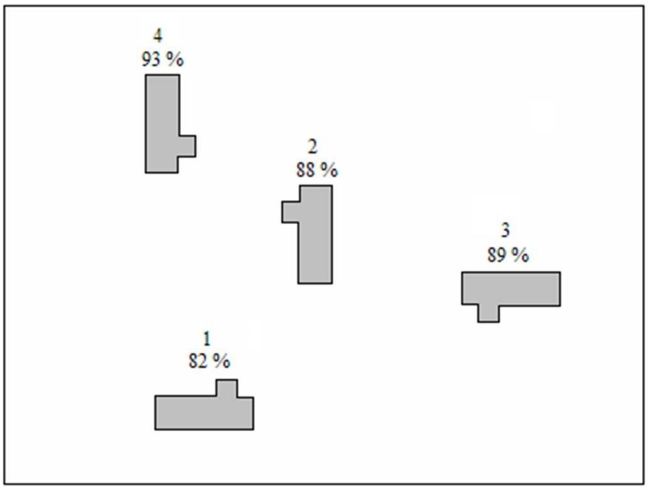
Sorting

The position that has the highest identification accuracy is normally sent first to the robot. In some cases, however, it may be preferable to let the robot pick from a certain direction or in a certain way first. With the sorting settings you can control how the coordinates are sorted and sent.

- **Worst match**: The coordinates of the position that gives the worst identification match are sent first. Because the location for the current worst

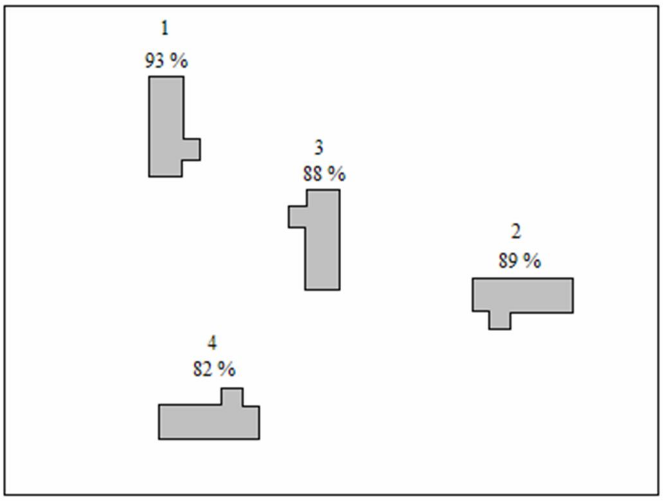
Continues on next page

match often varies slightly within the field of view, this setting appears to sort positions randomly.



xx1800000310

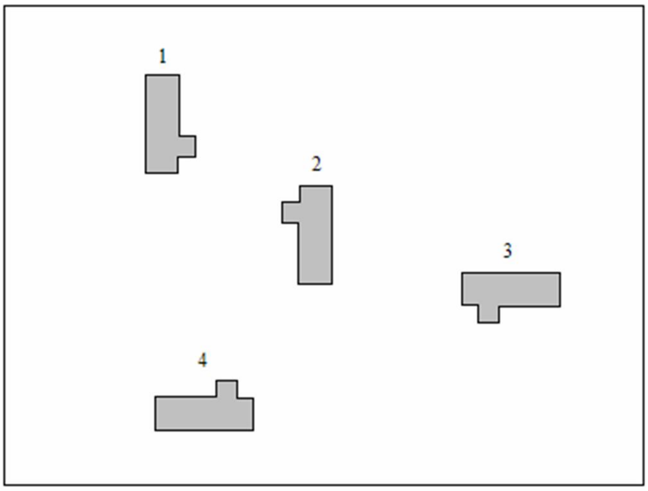
- **Best match**: The coordinates of the position that gives the best identification match are sent first. Because the location for the current best match often varies slightly within the field of view, this setting appears to sort positions randomly.



xx1800000311

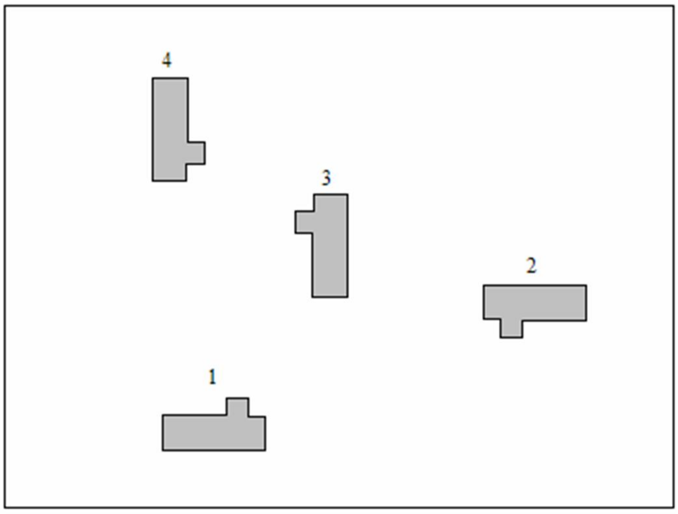
Continues on next page

- **Direction – Up -> Down**: The coordinates of the position at the top of the field of view are sent first.



xx1800000312

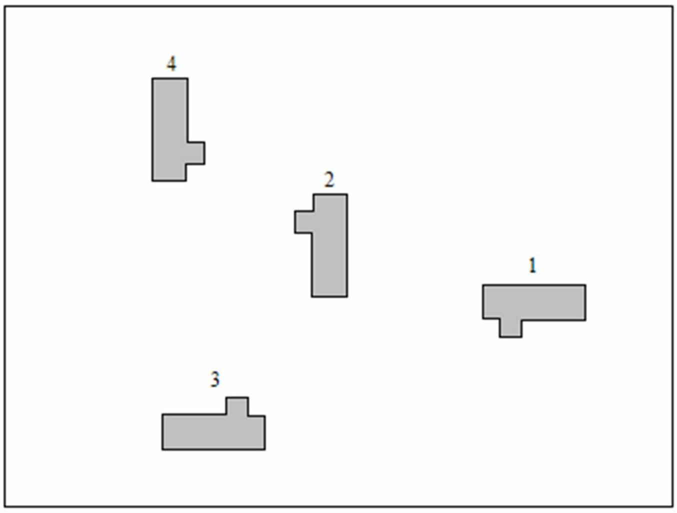
- **Direction – Down -> Up**: The coordinates of the position at the bottom of the field of view are sent first.



xx1800000313

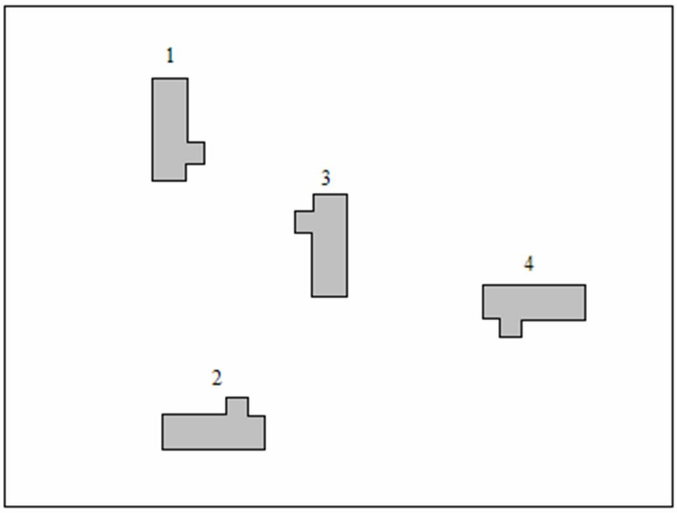
Continues on next page

- **Direction – Right -> Left**: The coordinates of the position at the far right of the field of view are sent first.



xx1800000314

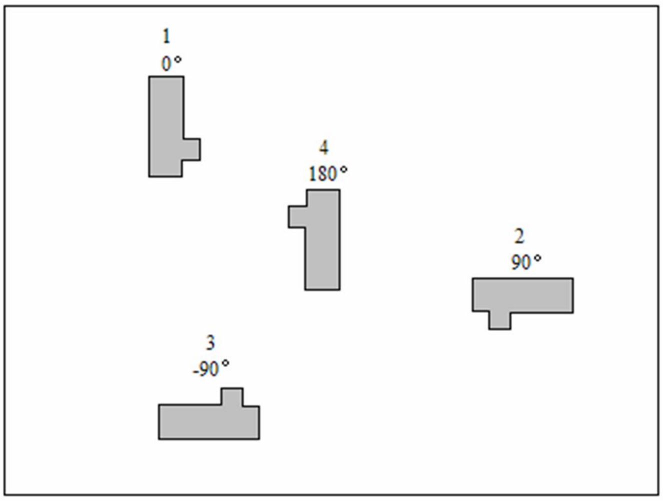
- **Direction – Left -> Right**: The coordinates of the position at the far left of the field of view are sent first.



xx1800001047

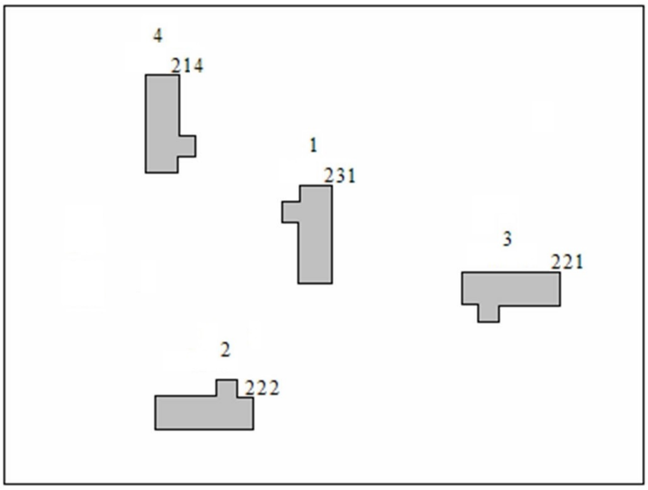
Continues on next page

- **Angle**: The coordinates of the position that is closest to the selected angle are sent first. If the selected angle is 0, the coordinates are sent in the order shown in the image below.



xx1800001048

- **Highest match**: This sorting condition is only available for 3D cameras. The detail located at the highest level is sent to the robot first. The located height of each detail is shown in millimetres in the image.



xx1800001053

Limitations

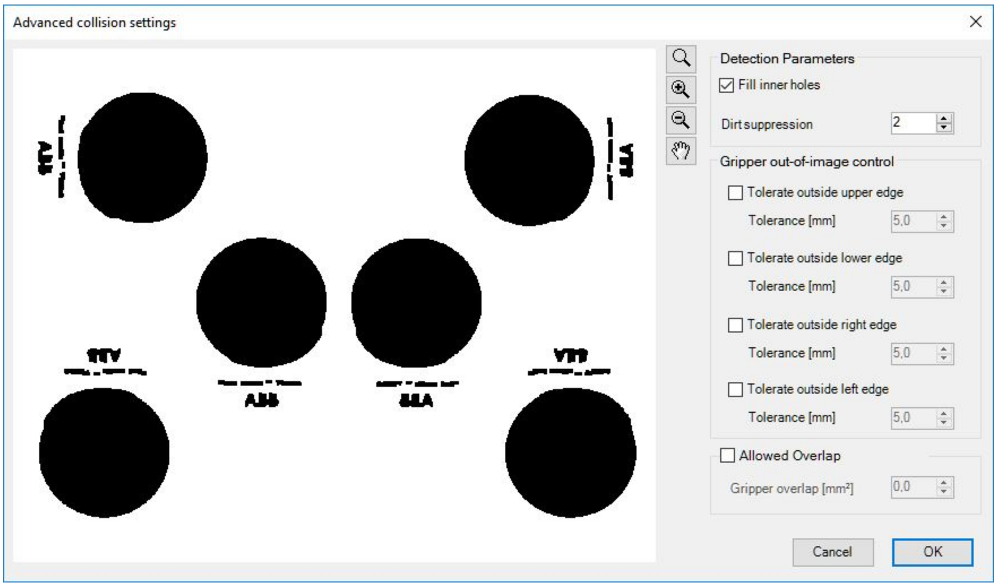
In some cases it may be desirable to prevent the robot from gripping or handling details when they are lying in certain positions and/or angles. This is done by specifying gripping limitations for the detail. Before you can activate the gripping limitations they must be defined in the system. For more information on this, see *Gripping Limitations on page 127*.

When this is done, you can select the limitations that you want to apply to the detail from this list. The number of limitations that can be activated for a detail is unlimited.

5.5 Advanced supervision settings

Advanced image supervision settings

Click **Advanced** under **Image supervision** on the **Supervision** tab to access additional image supervision settings.



Advanced collision settings

Detection Parameters

☑ Fill inner holes

Dirt suppression: 2

Gripper out-of-image control

□ Tolerate outside upper edge

Tolerance [mm]: 5,0

□ Tolerate outside lower edge

Tolerance [mm]: 5,0

□ Tolerate outside right edge

Tolerance [mm]: 5,0

□ Tolerate outside left edge

Tolerance [mm]: 5,0

□ Allowed Overlap

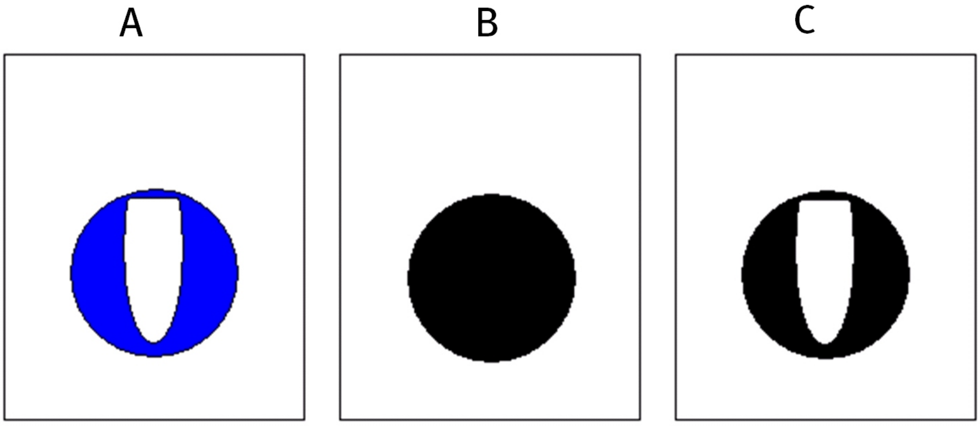
Gripper overlap [mm²]: 0,0

xx1800000241

Detection parameters

Fill inner holes

The default setting for collision detection is to fill the inner holes of details. However, in some cases this can give an inaccurate image of the detail.



xx1800001054

Pos.	Description
A	Teachin detail
B	Inner holes filled
C	Inner holes not filled

Continues on next page

The image shows an example in which the result of filling the inner holes gives a defective collision detection image for the detail.

Dirt suppression

The **Dirt suppression** setting filters out black marks on the conveyor, as these often are only just dirt. Dirt suppression can thus improve the reliability of the collision detection even on dirty conveyors. The higher dirt suppression level you set, the stronger filter you get. However, note that if the value is set too high it may also reject details, which may lead to collisions with the robot during gripping.

Gripper out-of-image control

If a position is close to the edge of the image it can often mean that the location of the gripper would fall outside the field of view. If the gripper moves outside the field of view this is normally regarded as a collision, since the system does not know what is beyond the field of view. In some cases, however, it can be perfectly safe to move the gripper outside certain areas of the field of view, as long as the operator knows that there is no collision risk there.

The **Gripper out-of-image control** can be used to set dimensional tolerances for positioning of the gripper outside the edges of the image. These tolerances can be activated on the chosen edges. To do so, select the concerned **Tolerate outside** boxes and enter the required tolerance dimensions. The designations of the edges (right, left, upper, lower) refer to the camera image that is seen on the screen.

Allowed overlap

If **Allowed overlap** is activated, the gripper is allowed to collide with objects in the image to a certain extent. The degree of collision permitted is specified in square millimetres. When tool collision is detected, FlexLoader Vision will compare the collision area with the specified, permitted overlap area.

If the collision covers less than the overlap area, the robot is permitted to pick the detail. For example, this can be used if the grip fingers are permitted to knock away nearby details when picking.

Advanced parameter settings for secondary search

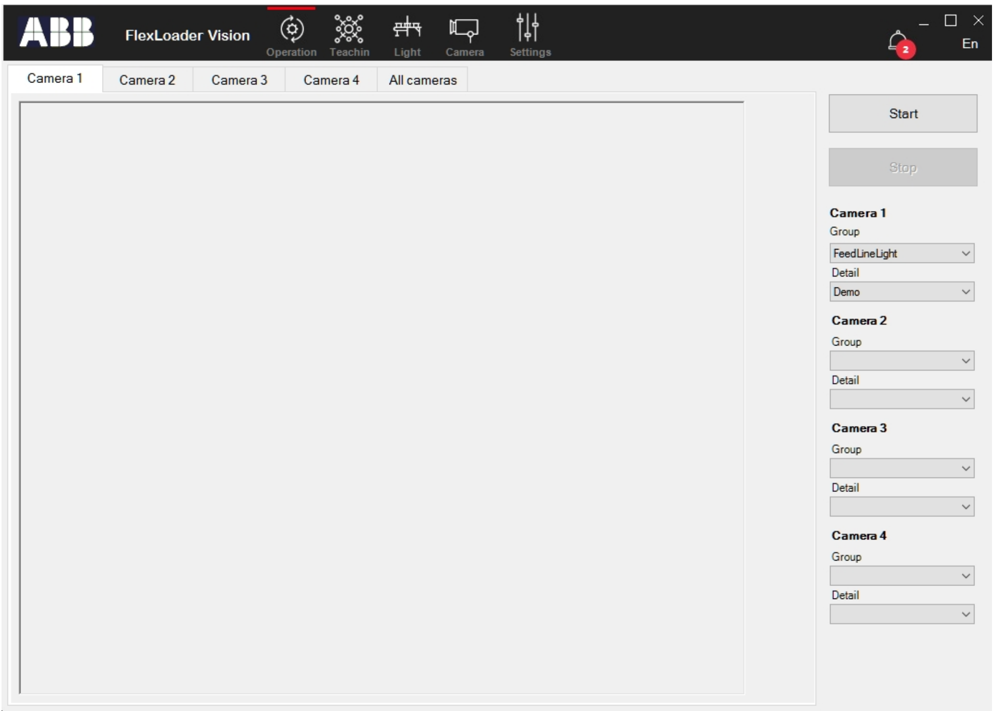
Click **Advanced** under **Detail supervision** > **Secondary search** on the **Supervision** tab to open and edit the advanced parameter settings for secondary searches. Enter lower agreement values for the teachin model in order to find more details than those approved for picking.

This page is intentionally left blank

6 Operation

6.1 Introduction

The system is started and stopped on the **Operation** menu. The appearance of this tab varies a little depending on how many cameras are connected to the system.



Camera 1
Group: FeedLineLight
Detail: Demo

Camera 2
Group:
Detail:

Camera 3
Group:
Detail:

Camera 4
Group:
Detail:

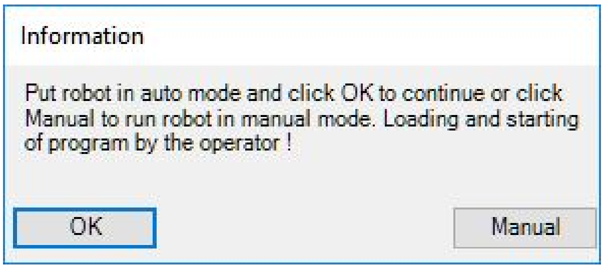
xx1800000242

The drop-down lists are now provided for the group and detail for each camera. The **Group** drop-down list for camera 1 is used to select the group for camera 1, the **Group** drop-down list for camera 2 is used to select the group for camera 2, and so on.

6.2 Starting the system

To start the system, select a group and a detail in the drop-down lists in the **Operation** window in FlexLoader Vision and click **Start**. The specified program is now downloaded to the robot, which then starts automatically if it is set to auto mode.

If the robot is not in auto mode, the following dialog box will appear when you click **Start**.



Information

Put robot in auto mode and click OK to continue or click Manual to run robot in manual mode. Loading and starting of program by the operator !

xx1800000274

If you want to continue with the robot in auto mode, put the robot in auto mode and click **OK**.

If you want to continue with the robot running at reduced speed, click **Manual**. Note that FlexLoader Vision does not load the program that is specified for the detail if you select **Manual**, instead the correct program must be loaded and started manually.

We recommend to start the system at least once with the robot in auto mode in order to load all modules correctly.

Note that the program must be started from the beginning.

6.3 Stopping the system

To stop the system, click **Stop** in FlexLoader Vision. If the robot is in **Auto** mode, it will also be stopped. Note that clicking **Stop** does not make the robot stop immediately. Instead it will stop at the end of its cycle or after a certain time. To stop the robot immediately, you must use the stop button on the robot or, if an emergency situation has arisen, use the emergency stop.

If FlexLoader Vision is configured so that the robot should control the stop situation, the stop button behaves differently. In this case FlexLoader Vision does not actively stop the robot when the operator clicks **Stop**. Instead, only **DOF_EndCycle** is sent to the robot, whereupon FlexLoader Vision awaits the robot to call the procedure **StopFlexLoader Vision** which in turn will stop everything as usual (as described above).

In case the user clicks **Stop** in FlexLoader Vision again, while FlexLoader Vision is awaiting the call to **StopFlexLoader Vision**, FlexLoader Vision will enforce a stop. This means that FlexLoader Vision will no longer be waiting for the robot, but will instead stop as usual (as in the case where FlexLoader Vision is configured to control the robot stop).

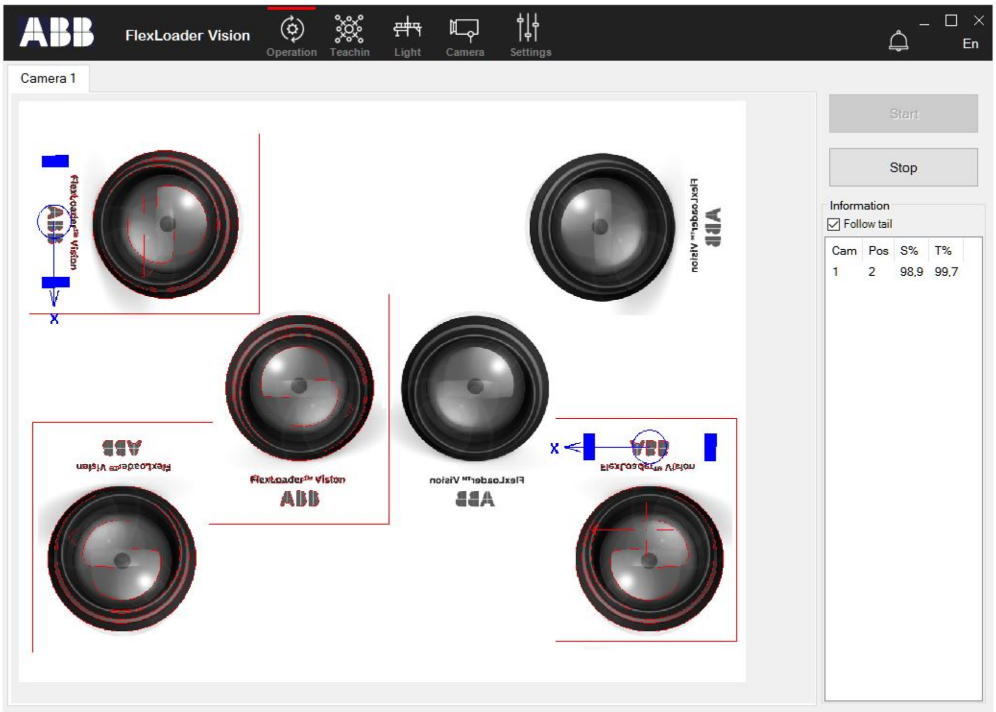
Letting the robot control the stop situation is, for example, useful when running special cycles that cannot be stopped in the middle of operation. In this case, the robot normally knows when a stop is appropriate.

If FlexLoader Vision is used with a FlexLoader FP 400, the FP 400 will also stop.

If FlexLoader Vision is used with a user-defined PLC, a stop signal can be defined that will be sent to the user-defined PLC.

6.4 Operation information

The following information is displayed during operation.



Cam	Pos	S%	T%
1	2	98,9	99,7

xx1800000243

The field of view always shows the latest image taken. If the system has more than one camera, a separate tab will appear for each camera, as well as a tab for all cameras.

A red box and the found contours are drawn over those details that have been identified. If the detail is to be picked, the gripping tool is drawn in blue. If an identified detail is not to be picked, it is marked with a cross and a letter. The letters have the following meanings:

Letter	Meaning	Description
C	Coverage	The detail does not meet the set requirements for minimum position coverage.
L	Limitation	The detail lies within an active limitation.
P	Proximity	The detail lies too close to a previously picked detail.
T	Tool	A tool collision would occur if the detail were to be picked.

Collision risk images are drawn with green contours.

Displayed to the right of the image are the latest results that have been sent to the robot. The most recent result always appears at the bottom. The results are shown as follows:

Parameter	Description
Cam	Camera number

Continues on next page

Parameter	Description
Pos	Position number
S%	Score %
T%	Score target %

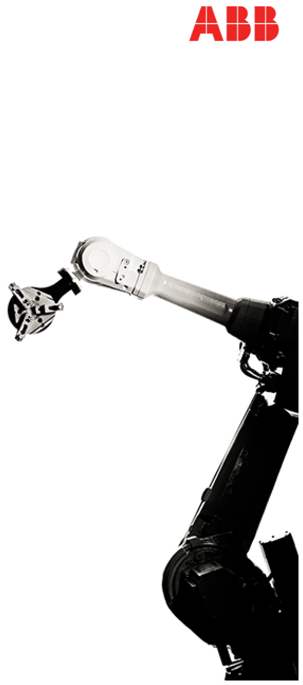
In the screenshot above FlexLoader Vision the results are as follows:

- **Cam**: 1
- **Pos**: 2
- **S%**: 98.9
- **T%**: 99.74

This means that a detail has been found in position 2 in an image taken by camera 1. It has been identified with a 98.9% score certainty and a 99.74% score target certainty.

6.5 SmartPhone connection

Production data from FlexLoader Vision can optionally be visualized with the FlexLoader Vision SmartPhone app. Currently the app is limited to systems with one camera.

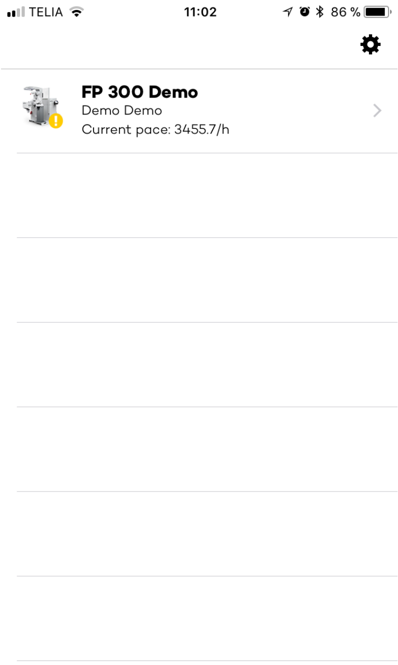


xx1800000511

The main screen shows all FlexLoader systems that are associated with the customer account and their current production status.

Continues on next page

This page can be manually updated. Just swipe down the page and release.

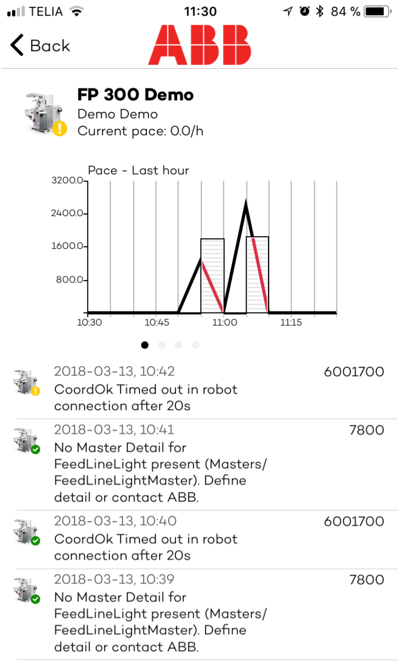


FP 300 Demo
Demo Demo
Current pace: 3455.7/h

xx1800000512

Select one of the FlexLoader systems to display the most recent production statistics and alarms.

This page can be manually updated. Just swipe down the page and release.



FP 300 Demo
Demo Demo
Current pace: 0.0/h

2018-03-13, 10:42 CoordOk Timed out in robot connection after 20s	6001700
2018-03-13, 10:41 No Master Detail for FeedLineLight present (Masters/ FeedLineLightMaster). Define detail or contact ABB.	7800
2018-03-13, 10:40 CoordOk Timed out in robot connection after 20s	6001700
2018-03-13, 10:39 No Master Detail for FeedLineLight present (Masters/ FeedLineLightMaster). Define detail or contact ABB.	7800

xx1800000513

This page is intentionally left blank

7 Camera

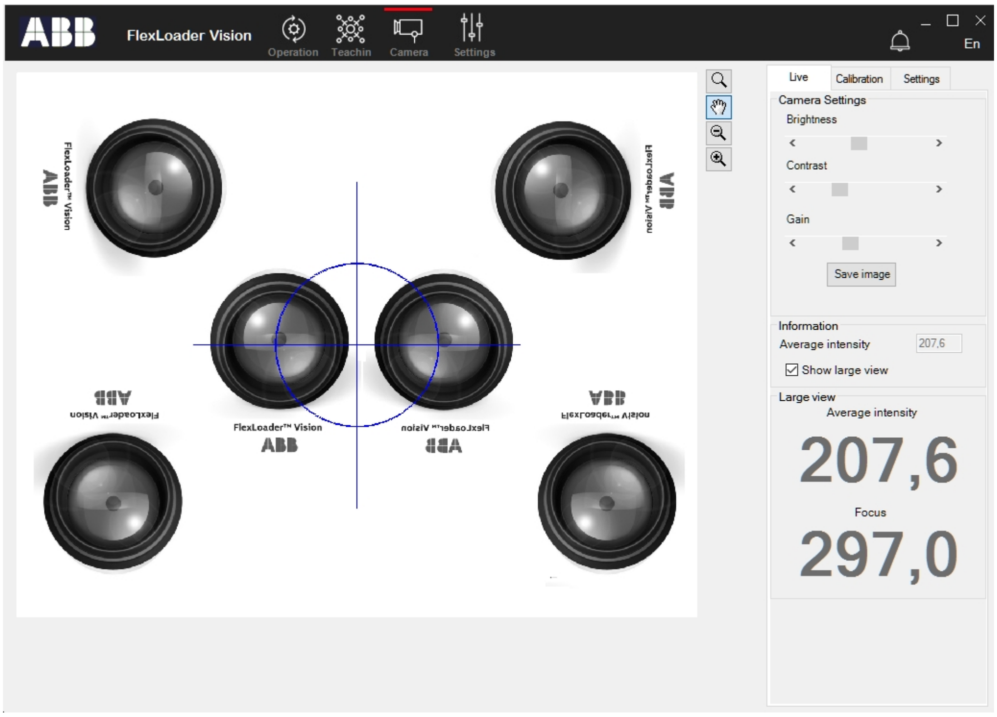
7.1 2D camera settings

Introduction

The **Camera** window has three tabs: **Live**, **Calibration** and **Settings**.

Live

The **Live** tab on the **Camera** menu shows the current camera image.



Average intensity 207,6

☑ Show large view

Average intensity
207,6

Focus
297,0

xx1800000246

Camera settings

Here you can also adjust the camera settings. It is a quick and easy way of checking the different settings. Click **Save image** to save the image for analysis on another computer, etc.

Information

If **Show large view** is selected, the display of **Average intensity** (brightness) and **Focus** (a higher value usually corresponds to a better focus) is enlarged. These two values help setting up the cameras for servicing and maintenance.

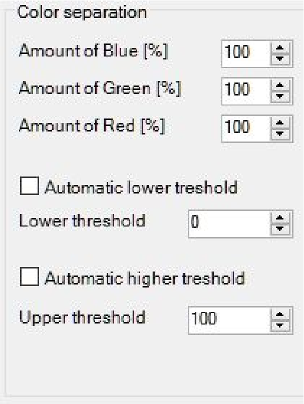
Color settings

If a color camera is used, the **Color separation** pane is displayed. Here you can determine how to combine the three colors applied to the black and white image that FlexLoader Vision is to analyze. This function can be used to enhance or weaken certain selected colors, or to increase or reduce the contrast between certain colors.

Continues on next page

For each included color (blue, red, green) the proportion that is used to generate the final black-white image is set. Note that it is also possible to subtract colors from each other by giving negative percentages.

The calculated black-white image can then be rescaled, either by allowing FlexLoader Vision to automatically determine the upper and lower limits (thresholds), or by specifying certain values.



Color separation

Amount of Blue [%] 100

Amount of Green [%] 100

Amount of Red [%] 100

☐ Automatic lower treshold

Lower threshold 0

☐ Automatic higher treshold

Upper threshold 100

xx1800000232

Calibration

Introduction

In order for FlexLoader Vision to send the correct coordinates to the robot, the system must be calibrated together, i.e. the coordinate systems of FlexLoader Vision and the robot must be made to correspond. For this purpose, the supplied calibration plate and calibration tool are used.

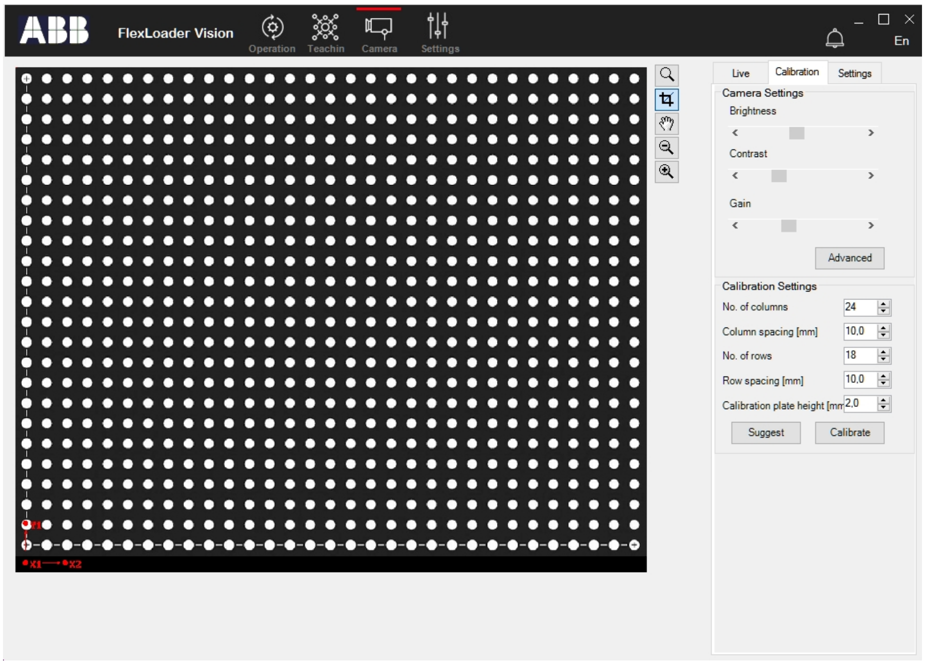
The following information describes how to calibrate FlexLoader Vision. For information about how to calibrate the robot, see the separate robot product manual.

To start with, FlexLoader Vision must be calibrated. For this purpose, the supplied calibration plate is used.

Continues on next page

Calibrating a 2D camera

1 Select the **Calibration** tab on the **Camera** menu in FlexLoader Vision.



No. of columns 24
Column spacing [mm] 10,0
No. of rows 18
Row spacing [mm] 10,0
Calibration plate height [mm 2,0

xx1800000244

2 Select the camera to be calibrated at the bottom of the screen under **Active camera** (this setting is only available if there are several cameras installed). Then place the plate underneath the camera so that the point with coordinates x=0, y=0 appears at the bottom left corner of the image. The image also shows a small coordinate system that illustrates how the calibration plate should be orientated.

3 Make sure that the dots fill the whole image and go as far out to the edges as possible. No dots may be only partially visible in the field of view. If it is not possible to adjust the plate so that no partial dots are seen, the partially visible row must be covered.

4 Under **Camera Settings**, adjust **Brightness**, **Contrast** and **Gain** to create as sharp a contrast as possible between the dots and the background.

5 Adjust the number of columns (vertical) under **No. of columns** and the number of rows (horizontal) under **No. of rows**. Click **Suggest** if you want the system to display suggested values for the number of visible rows and columns.

6 Enter the spacing between each dot. On the standard calibration plate, these values are printed on the plate. If the spacing is not known it must be measured. FlexLoader Vision cannot give suggestions for the current row or column spacings.

7 To start the calibration, click **Calibrate**.

Continues on next page

Note

Note that the calibration plate must not move during the calibration. If it moves, you must start over with the calibration.

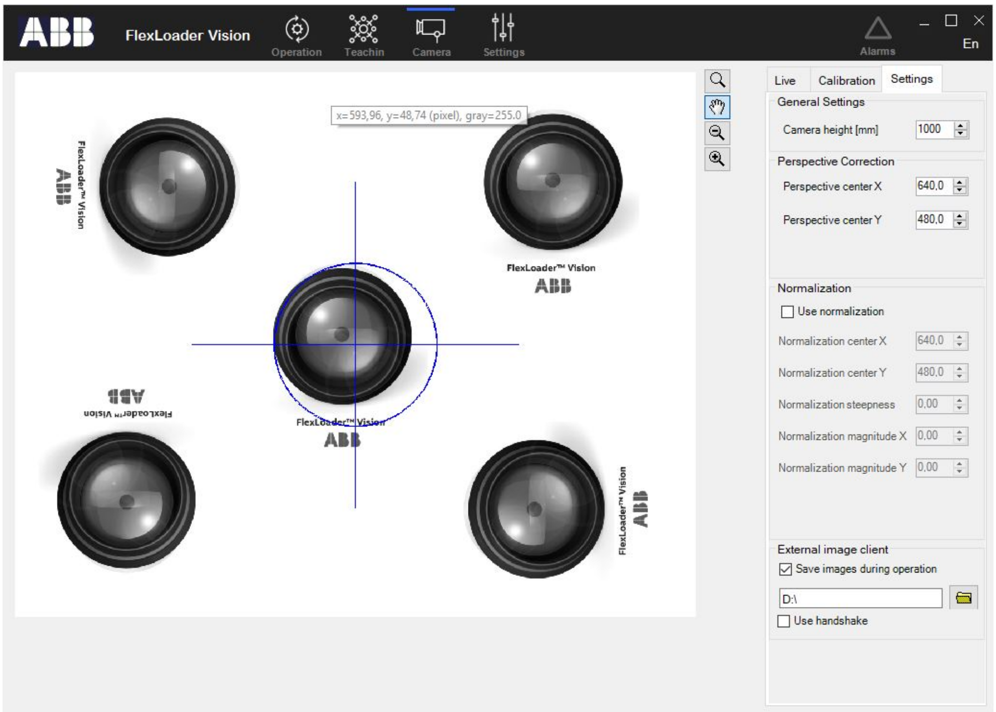
Note

It is possible to reduce the area that is used for calibration. This can be useful in several cases, e.g. when the field of view is large and there are visible objects outside of the calibration plate.

Select the define position tool and draw a red rectangle around the valid calibration dots. Then perform the calibration as described above.

Settings

General Settings



General Settings

Camera height [mm] 1000

Perspective Correction

Perspective center X 640,0

Perspective center Y 480,0

Normalization

☐ Use normalization

Normalization center X 640,0

Normalization center Y 480,0

Normalization steepness 0,00

Normalization magnitude X 0,00

Normalization magnitude Y 0,00

External image client

☑ Save images during operation

D:\

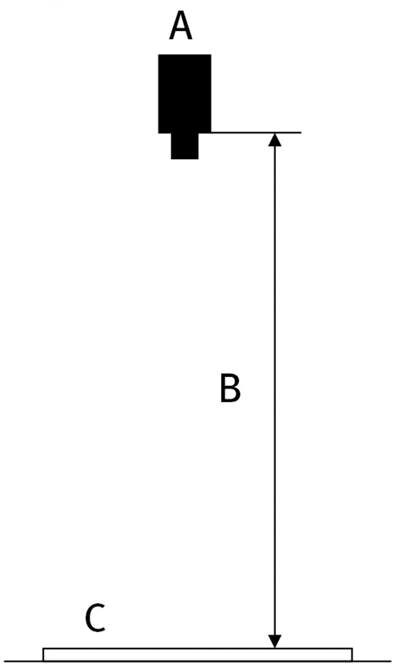
☐ Use handshake

xx1800000248

Under **General Settings**, enter the correct camera height to ensure that the system sends the correct coordinates to the robot. The camera height is measured in mm

Continues on next page

from the edge of the camera (not the lens) to the calibration plate, see the image below.



xx1800001057

Pos.	Description
A	Camera
B	Camera height
C	Calibration plate

Perspective Correction

Perspective correction is always used to take into account the different edge heights of details above the calibration plate. From the camera viewpoint, details that are in the same location but have different edge heights appear to be in different locations in the field of view (known as parallax error). FlexLoader Vision therefore has a built-in perspective correction technology that automatically compensates for this effect.

The perspective center (the point in the field of view where the camera is looking vertically down on to the conveyor) must be specified in order for the automated correction to work. There are two ways of finding the perspective center:

- Place a mirror on the conveyor underneath the camera. Look for the camera lens in the image (a dark round spot). Right-click on the image and select **Show cursor position**, followed by **Position in pixels.** Point at the middle of the camera lens spot and note the x and y coordinates. Enter these coordinates as **Perspective center X** and **Perspective center Y**.
- Attach the calibration device on the robot, move it directly above the picking area and tilt it so that the tip can be clearly seen in the camera image. Then move the robot tip up and down in camera coordinate system, while you

Continues on next page

watch the tip in the camera image. Try to find a position in the x and y axes where the robot tip does not appear to move in the camera image as the tip is moved up and down. Enter these coordinates as **Perspective center X** and **Perspective center Y**.

Normalization

Normalization can be used if you have an image that is unevenly lit, i.e. when the center of the image is light but the corners are dark. Normally there is no need to use normalization; it is only likely to be needed when you have a large field of view and a light background.

Normalization center X and **Normalization center Y** must be set for the point you deem to be the lightest in the image. The current x and y coordinates are displayed as you move the cursor over the image. (Right-click on the image and select **Show cursor position**, followed by **Position in pixels**). Move the cursor to the lightest point, read the coordinates and enter them.

Normalization steepness is set between 0 and 10. This value is used to adjust the correction at the edges of the image. A low value results in a correction that decreases towards the center of image. A high value results in a stronger correction at the edges, but hardly any correction closer in towards the center. Usually **Normalization steepness** should be set around 1.

Normalization magnitude X and **Normalization magnitude Y** control how large the correction should be in the x axis and y axis, respectively. A high value results in a large correction for each axis, while a low value results in a smaller correction. Usually the correction factor should be between 0.5 and 0.9.

Export images to external clients

Images taken by FlexLoader Vision can be exported to external clients for further analysis, e.g. image analysis through Matrox Design Assistant or similar software.

This functionality is activated by checking **Save images during operation** and selecting a folder where the images are saved. Please check **Use handshake** if the external client supports the functionality.

The following sequence is used (X (1,2,3,4) represents the camera number):

- If handshake is used, FlexLoader Vision awaits the creation of the **READ_COMPLETE_CAMX.txt** file before saving the image file (maximum wait time 2 s).
- Every time an image is taken during operation, a image is saved with the name **EXPORT_CAMX.tif**. As soon as the image is saved, the **READ_COMPLETE_CAMX.txt** file is deleted.
- If handshake is used, the file **CREATED_CAMX.txt** is written to the disk.
- If handshake is used, the external client awaits the creation of the file **CREATED_CAMX.txt**
- The external client reads the image file.
- The external client deletes the image file and the handshake file (if used).
- If handshake is used, the client writes the file **READ_COMPLETE_CAMX.txt**.

7.2 3D camera settings

Introduction

The **Camera** window for 3D camera has the same three tabs as for 2D cameras, but the settings differ.

Live

For 3D cameras, the **Live** tab is only used to save images (as the 3D camera does not need settings for brightness, contrast and gain). Therefore only the **Save image** button is visible.

Calibration

In order for FlexLoader Vision to send the correct coordinates to the robot, the system must be calibrated together, i.e. the coordinate systems of FlexLoader Vision and the robot must be made to correspond. For this purpose, the supplied calibration plate and calibration tool are used.

The following sections describes how to calibrate FlexLoader Vision. For information about how to calibrate the robot, see the separate robot product manual.

Start by calibrating the coordinate system **wCamera1** (or for example **wCamera2** for camera 2) in the robot. The coordinate system should lie with the x-axis along the long side of the pallet, the y-axis along the short side of the pallet, and the z-axis pointing upwards (for example, it can lie on the guides for the pallets).

Fit the calibration cone on the robot and ensure that the tool center point (TCP) for it is used in all calibration positions. If the calibration positions do not exist they must be created to facilitate the calibration. For more information on this, see *Calibrating for use with a 3D camera on page 228*.

All positions should lie in **wCamera1** (the 3D camera is camera 1) and be within the robot's range and the camera's measurement range. The calibration cone should point directly upwards at each point. If the camera is inclined in relation to **wCamera1**, inclined cone positions should be considered.

FlexLoader Vision needs 9 to 30 calibration points (typically 15) in order to perform calibration calculations. Prepare the RAPID module **Calibration3D.sys** with all robot positions used during calibration (see routine **Calibration3DUpdatePositions**).

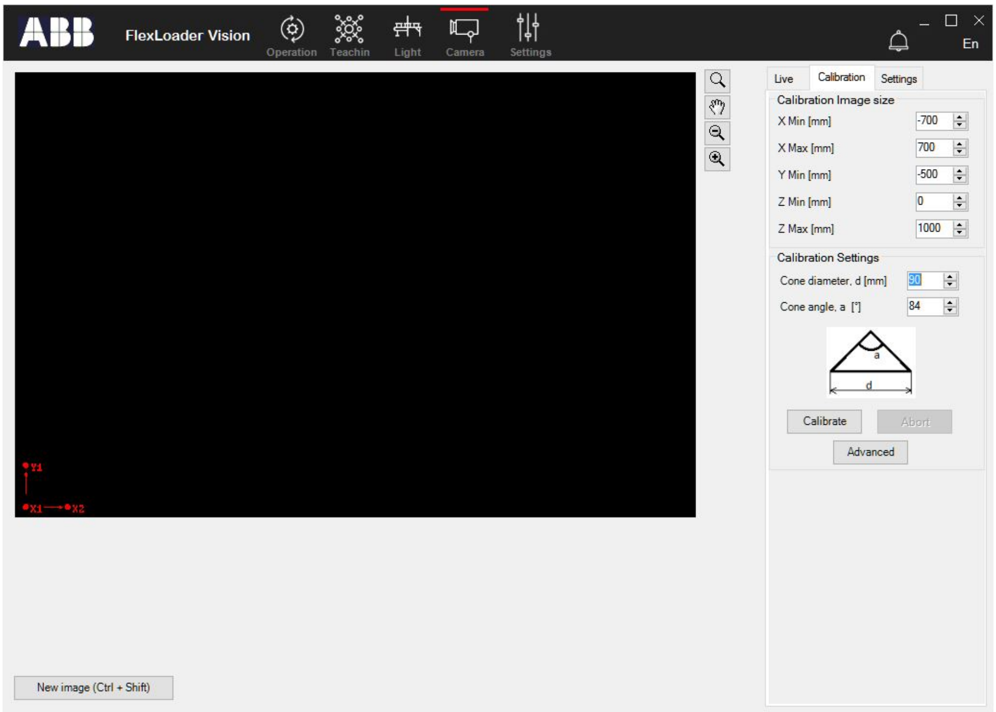
It is strongly recommended to use 5 points as far down as possible, one in each corner of the area where the pallets will stand and 1 in the middle. Then a further 5 positions located in the same way at a medium height and, finally, a further 5 located in the same way at a height close to the maximum height. Remember that the positions must always be within the measurement volume of the camera.

In FlexLoader Vision, select **Camera** and the tab **Calibration**. Enter cone diameter and cone angle of the robot's calibration cone. Click **Calibrate.**

After a confirmation prompt, FlexLoader Vision will reset the current calibration. This means that the displayed part of the camera field of view may have moved considerably.

Continues on next page

The operator is asked to start the calibration routine in the robot. The calibration can be performed more or less automatically. See *Calibrating for use with a 3D camera on page 228* for more information.



Calibration Image size

X Min [mm] -700
X Max [mm] 700
Y Min [mm] -500
Z Min [mm] 0
Z Max [mm] 1000

Calibration Settings

Cone diameter, d [mm] 90
Cone angle, a [°] 84

xx1800000245

FlexLoader Vision will provide visual feedback about cone identification progress. A blue circle with a cross indicates where the cone was found. The values to the right show the current robot coordinates and the identified camera coordinates. The coordinates can also be entered manually. See *Robot integration on page 221*.

When everything is done, the information on calibration quality is displayed. The operator can choose to accept the new calibration or revert to the previous one.

Note that the image size needs to be adjusted to fit the new calibration.

In some cases, a temporary manual adjustment of the characteristics of the 3D camera is required in order to perform a calibration. Click **Advanced** on the **Calibration** tab and adjust the image in order to obtain a clear and smooth image of the calibration cone.

If you need more information on how to calibrate the camera, refer to the FlexLoader Vision camera integration manual.

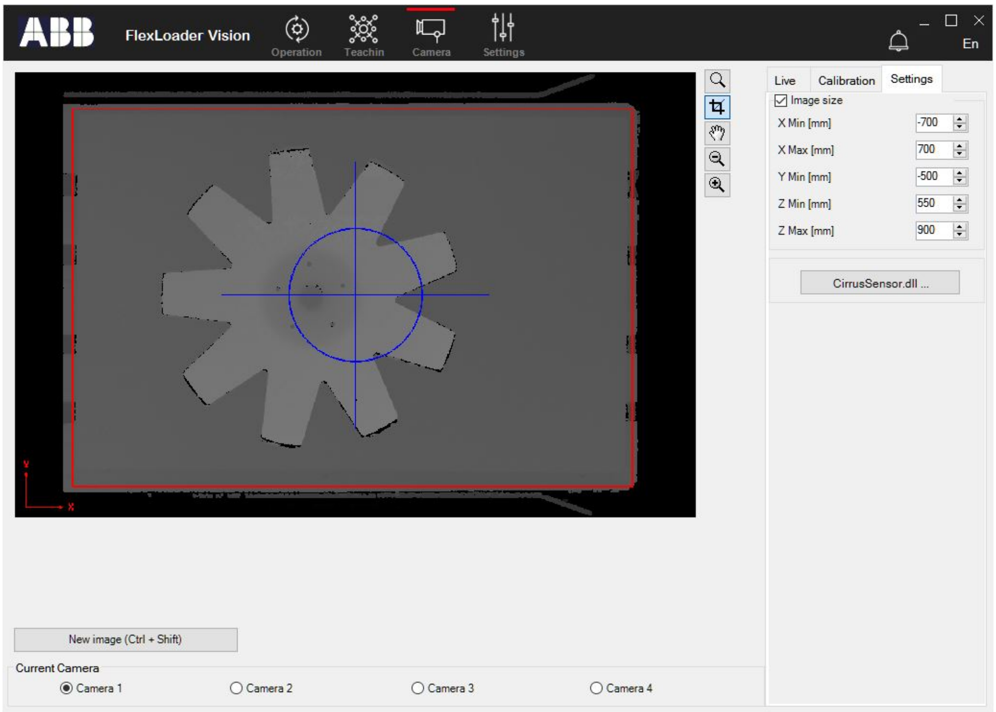
Continues on next page

Settings

Introduction

A 3D camera differs from a normal 2D camera. The image that comes into FlexLoader Vision is rectified, which means that it does not have any perspective faults. The grey scale corresponds to the height so that it is possible to find the z coordinates.

Some settings must be made in order for FlexLoader Vision to search in an efficient way.



☑ Image size

X Min [mm]	-700
X Max [mm]	700
Y Min [mm]	-500
Z Min [mm]	550
Z Max [mm]	900

Current Camera

● Camera 1 ○ Camera 2 ○ Camera 3 ○ Camera 4

xx1800000249

Image size

Depending on the application, the interesting area for detail search can vary quite a lot. The **Image size** settings let you determine exactly which area, i.e. what volume, should be displayed and analyzed by FlexLoader Vision.

It is important that the measurement volume is slightly larger in the x- and y-axis than the actual pick volume. For a pallet it is important for example that the whole pallet collar is always visible in the image.

When operating without interlayers, the lowest height value (**Z min**) should be specified so that the floor or bottom of the pallet is not shown in the image. The FlexLoader Vision image must always contain sufficient height information so that the last layer of details can be found but not the surface that they are lying on. The robot will only receive **CHANGE_BIN** flags.

In contrast, when operating with interlayers, the lowest height value must be adjusted so that the floor or bottom of the pallet is shown in the image. The robot will then receive **PICK_INTERLAYER_AND_CHANGE_BIN** flags.

Continues on next page

The highest height value (**Z max**) must be specified so that the highest detail always lies below this height. The aspect ratio of the defined area is always constant which means that the **Y max** value is calculated automatically in FlexLoader Vision.

Note

All values for the measurement volume are measured in the coordinate system of the robot (i.e. for camera 1)).

Region of interest

The region of interest is a very important parameter when using a 3D camera. When FlexLoader Vision starts searching for a detail, the highest point is identified first. FlexLoader Vision then continues the search downwards until the detail is located, or as far down as is set by the detail analysis depth.

When searching for the highest point, the pallet collar must not be included. Therefore there is a region of interest which specifies within which area FlexLoader Vision is to search. Note that the details that are outside the region of interest will not be included in the search for the highest point. The region of interest should lie just inside the pallet collar.

If there is no pallet collar or other edges or guides for the details, a region of interest that is large enough for the whole area where the details are to be found must be drawn.

Note

The highest point determined within the cameras region of interest sets an upper limit for the allowed height of details. All details more than 50 mm above the highest point will be discarded.

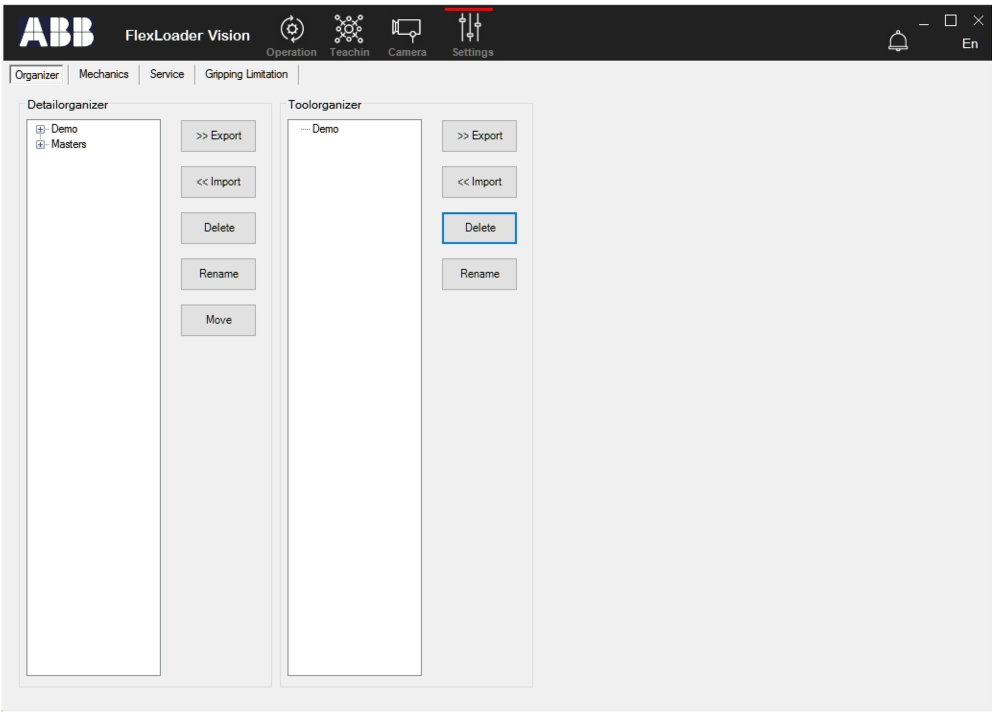
To define the region of interest, first click . Click the mouse button, drag the mouse diagonally to draw a box of the desired size, then release the mouse button. This will draw the red rectangle that shows the region of interest.

8 Settings

8.1 Organizer

Introduction

The **Organizer** tab on the **Settings** menu allows you to manage the details and tools in the database.



xx1800000250

Detail organizer

In the **Detail organizer** pane you can see the groups that have been created in the system. Click the plus sign (+) next to a group to show the teachin details included in that group. Click the minus sign (-) to hide them.

Export

Teachin details or groups in a system can easily be moved between different systems. Select the desired detail or group and click **Export**. Choose a location for the file and click **OK**. This saves all data associated with that detail to a file. This can be very useful when you have several identical systems, or if you want to carry out the teachin process on another computer.

Note

If the exported detail is to be used on another system, the camera geometry and settings on that system, i.e. the camera height, camera lens and aperture setting, must be the same as on the old system.

Continues on next page

Import

Click **Import** to import details that have been exported from another system. Select the exported file and click **OK.**The details are imported to the same group and have the same names as they did prior to the export.

Delete

Select a group or detail and click **Delete** to delete the name of the group or detail. Click **Yes** to confirm.

Rename

Select a group or detail and click **Rename**. Edit the name of the group or detail and click **OK**.

Move

Details can be moved from one group to another group. Select the detail concerned and click **Move**. Select the group that you want to move the detail to and click **OK**.

Tool organizer

The **Tool organizer** pane shows the tools that you have defined in the system. Note that the predefined tools are not visible here.

Export

User-defined tools can easily be moved between different systems. Select the desired tool and click **Export**. This saves all data associated with that tool to a file. This can be very useful when you have several systems with identical tools.

Note

If the exported tool is to be used on another system, the camera geometry and settings on that system, i.e. the camera height, camera lens and aperture setting, must be the same as on the old system.

Import

Click **Import** to import tools that have been exported from another system. Select the exported file and click **OK**. The details are imported to the same group and have the same names as they did prior to the export.

Delete

Select a tool and click **Delete** to delete the tool from the system. Click **Yes** to confirm that you want to delete it.

If the tool is used for one or more teachin details you will be asked to confirm the deletion once more. Click **Delete** to confirm the deletion. If you choose to delete it, you will need to define a new tool for the details that used the deleted tool before the system can run those details again.

Rename

Select a group or detail and click **Rename**. Edit the name of the group or detail and click **OK**.

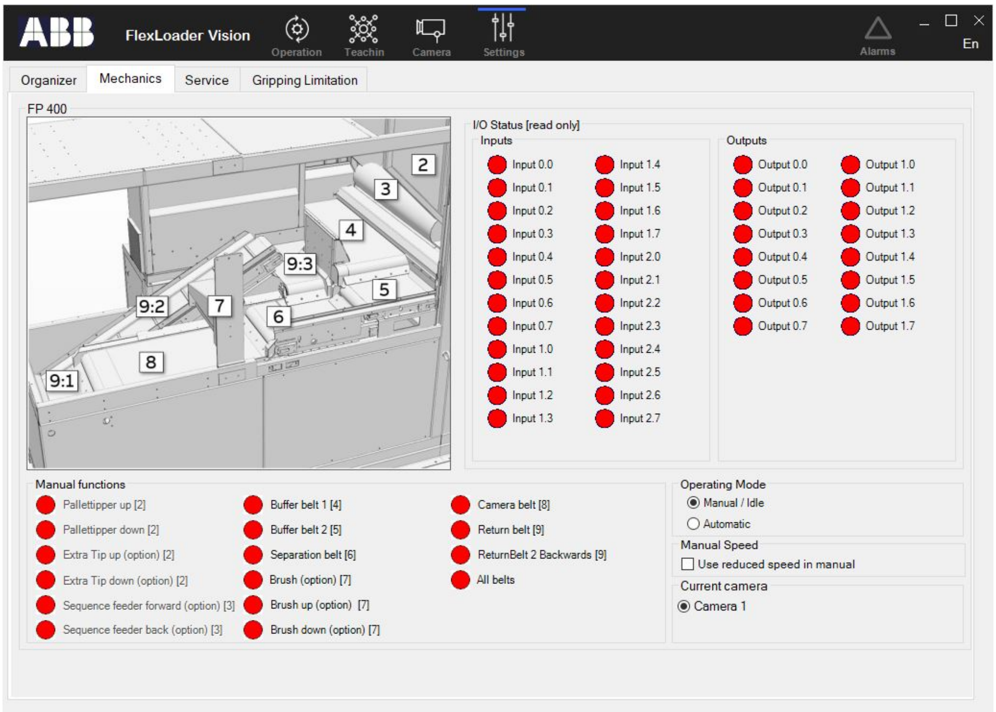
8.2 Mechanics

What information you see on the **Mechanics** tab depends on which type of equipment FlexLoader Vision is used with. For example you can see the input and output status, as well as various modes, functions and settings.

Note that the functions cannot be operated manually during production.

When used with FlexLoader FP 400

Below follows an example of what it may look like when FlexLoader Vision is used together with FlexLoader FP 400.



I/O Status [read only]

Inputs

● Input 0.0
● Input 0.1
● Input 0.2
● Input 0.3
● Input 0.4
● Input 0.5
● Input 0.6
● Input 0.7
● Input 1.0
● Input 1.1
● Input 1.2
● Input 1.3
● Input 1.4
● Input 1.5
● Input 1.6
● Input 1.7
● Input 2.0
● Input 2.1
● Input 2.2
● Input 2.3
● Input 2.4
● Input 2.5
● Input 2.6
● Input 2.7

Outputs

● Output 0.0
● Output 0.1
● Output 0.2
● Output 0.3
● Output 0.4
● Output 0.5
● Output 0.6
● Output 0.7
● Output 1.0
● Output 1.1
● Output 1.2
● Output 1.3
● Output 1.4
● Output 1.5
● Output 1.6
● Output 1.7

Manual functions

● Pallettipper up [2]
● Pallettipper down [2]
● Extra Tip up (option) [2]
● Extra Tip down (option) [2]
● Sequence feeder forward (option) [3]
● Sequence feeder back (option) [3]
● Buffer belt 1 [4]
● Buffer belt 2 [5]
● Separation belt [6]
● Brush (option) [7]
● Brush up (option) [7]
● Brush down (option) [7]
● Camera belt [8]
● Return belt [9]
● ReturnBelt 2 Backwards [9]
● All belts

Operating Mode
◉ Manual / Idle
○ Automatic

Manual Speed
□ Use reduced speed in manual

Current camera
◉ Camera 1

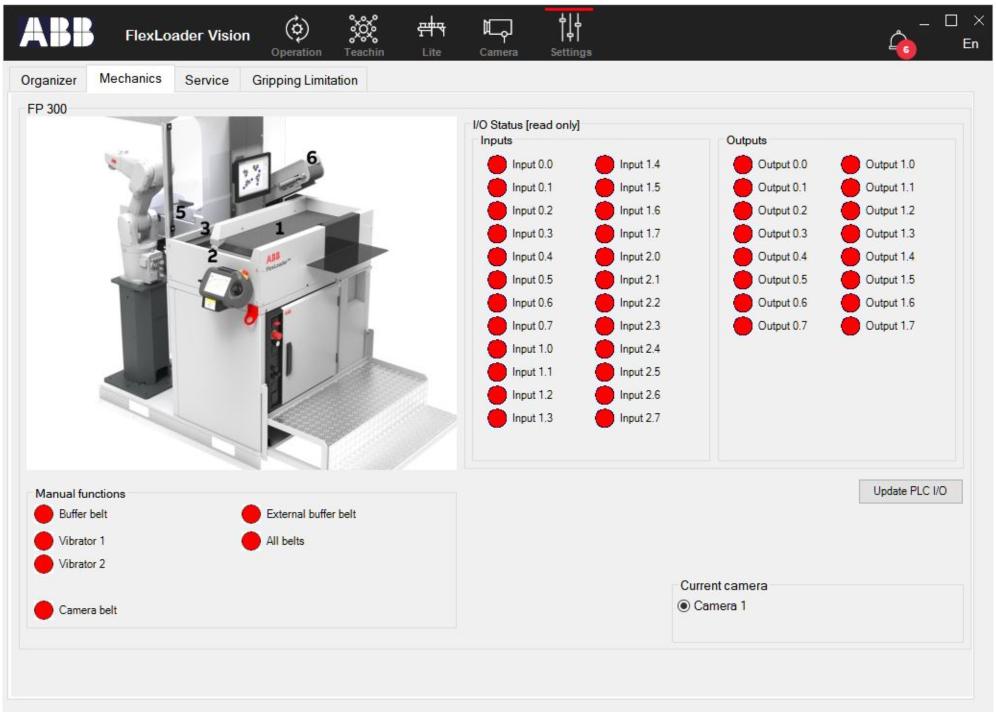
xx1800000256

For more information on the FlexLoader FP 400, refer to the FlexLoader FP 400 product manual.

Continues on next page

When used with FlexLoader FP 300

Below follows an example of what it may look like when FlexLoader Vision is used together with FlexLoader FP 300.



Current camera
◉ Camera 1

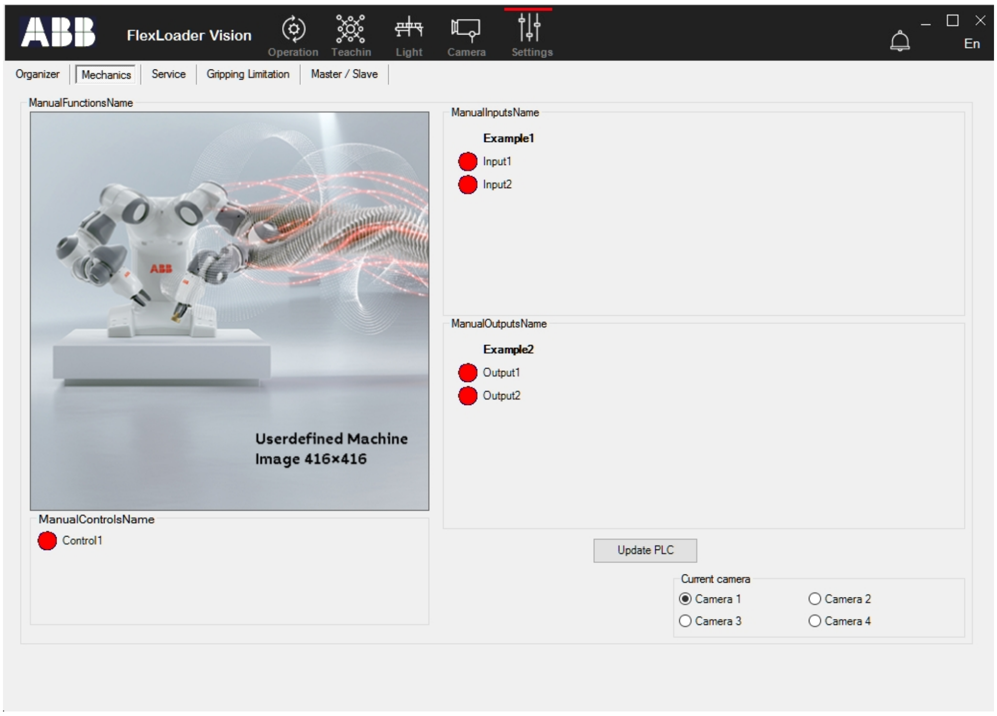
xx1800000590

For more information on the FlexLoader FP 300, refer to the FlexLoader FP 300 product manual.

Continues on next page

When used with user-defined mechanical equipment

The display for user-defined mechanical equipment varies depending on which equipment is used together with FlexLoader Vision. Below follows an example of what the interface may look like. For more information, refer to the product manual of the user-defined equipment.



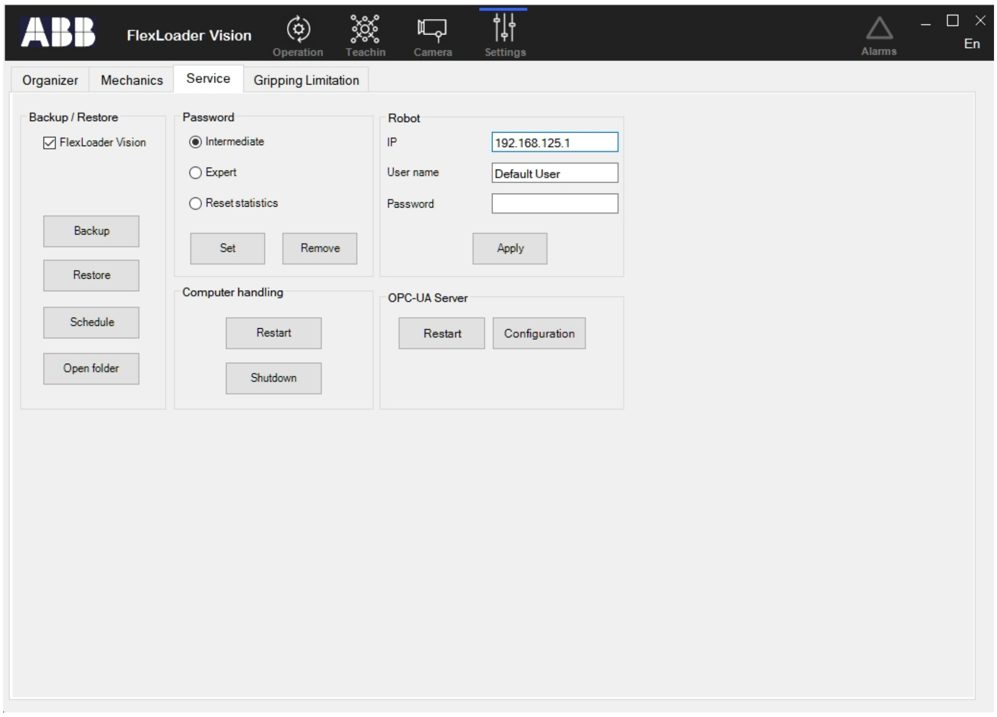
Current camera
● Camera 1 ○ Camera 2
○ Camera 3 ○ Camera 4

xx1800000257

For more information, see *User-defined mechanical equipment on page 186*.

8.3 Service

The **Service** tab in the **Settings** window allows you to backup and restore FlexLoader Vision, set and edit passwords for system functions, change the IP address of the robot and restart the computer.



☑ FlexLoader Vision

● Intermediate
○ Expert
○ Reset statistics

IP: 192.168.125.1
User name: Default User
Password:

xx1800000253

Backup and restore FlexLoader Vision

A system backup includes all files associated with FlexLoader Vision.

To perform a backup, click **Backup** on the **Settings** menu > **Service** tab. Select where you want to save the backup file and click **OK**.

To restore the system from a backup file, click **Restore**. FlexLoader Vision must be restarted for the restored values to apply. If the system cannot be restored automatically, you can restore the system manually. For more information on how to do a manual restore, see *Manually restoring a backup on page 304*.

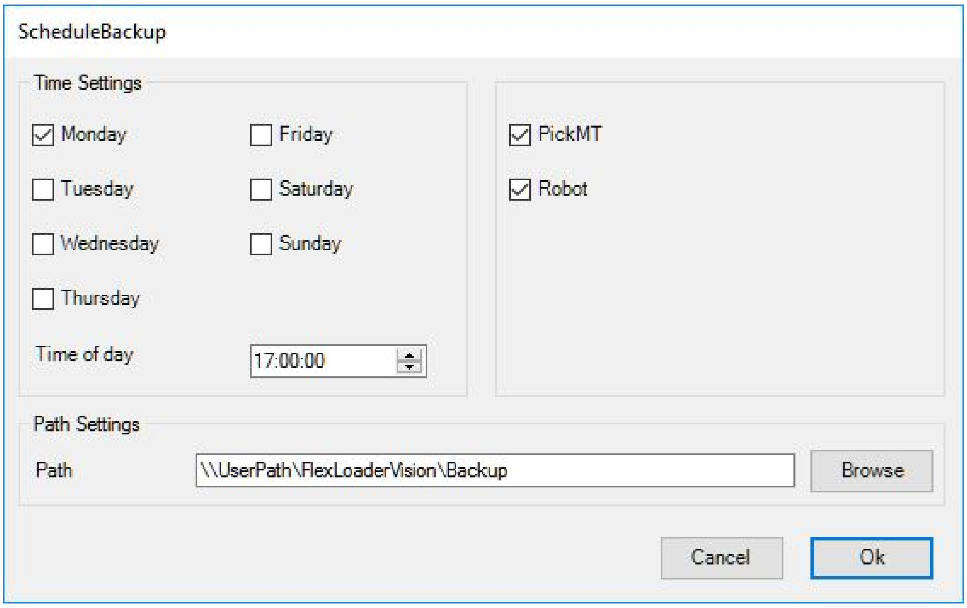
Note

Additional backup data may be created for 3D cameras in a separate folder (e.g. **Camera_1_CirrusBackup**). These files have to be manually copied to the sensor if needed.

To schedule a backup procedure, click **Schedule**. Select which day/s and which time of day you want the backup to be performed. The backup can be scheduled to up to once a day. Select what units you want to include in the backup process in the pane to the right. Under **Path Settings**, enter the path to the folder where

Continues on next page

you want to save the backup files or click **Browse** and browse for the correct folder. Click **OK** to apply the settings.



ScheduleBackup

Time Settings

☑ Monday
□ Tuesday
□ Wednesday
□ Thursday
□ Friday
□ Saturday
□ Sunday

Time of day: 17:00:00

☑ PickMT
☑ Robot

Path Settings

Path: \\UserPath\FlexLoaderVision\Backup

xx1800000254

In the image above, FlexLoader Vision is backed up every Monday at 17:00. The backup files are saved in the folder C:\FlexLoader Vision\Backup.

For information on how to do a complete hard disk backup or recovery, see *Backup and recovery of the hard disk on page 211*.

Password

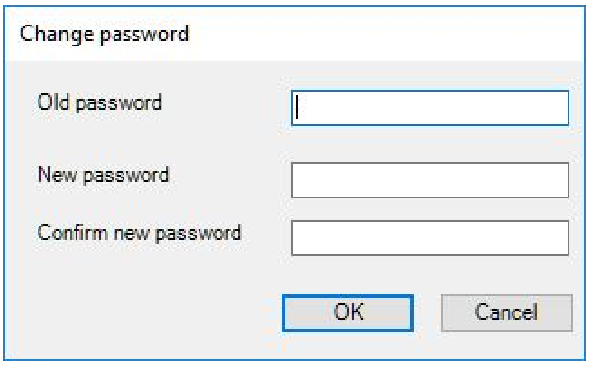
Some of the system functions can be password-protected.

- If the system is password-protected on the **Intermediate** level, you must enter a password to view the **Teachin** menu.
- If the system is password-protected on the **Expert** level, you must enter a password to view the **Settings** and **Calibration** tabs on the **Camera** menu and the complete **Settings** menu.

To set a password for one of the levels, select the desired level and click **Set**. Leave the **Old password** field blank, and then enter the new password in the fields **New password** and **Confirm new password**. Click **OK** to confirm the change.

Continues on next page

To change a password, choose the desired level and click **Set**. Enter the current password in the **Old password** field, and then enter the new password in the fields **New password** and **Confirm new password**. Click **OK** to confirm the change.



xx1800000255

To delete a password, select the desired level and click **Remove**. Click **Yes** to confirm the deletion.

Robot data

Here you can change the IP address for the robot. You can also enter data for the robots user authentication system that is used for connecting to the robot.

Please enter a valid IP address and appropriate user credentials, then click **Apply**. FlexLoader Vision must be restarted for the new address to be activated.

Computer handling

Click **Restart** to restart the computer.

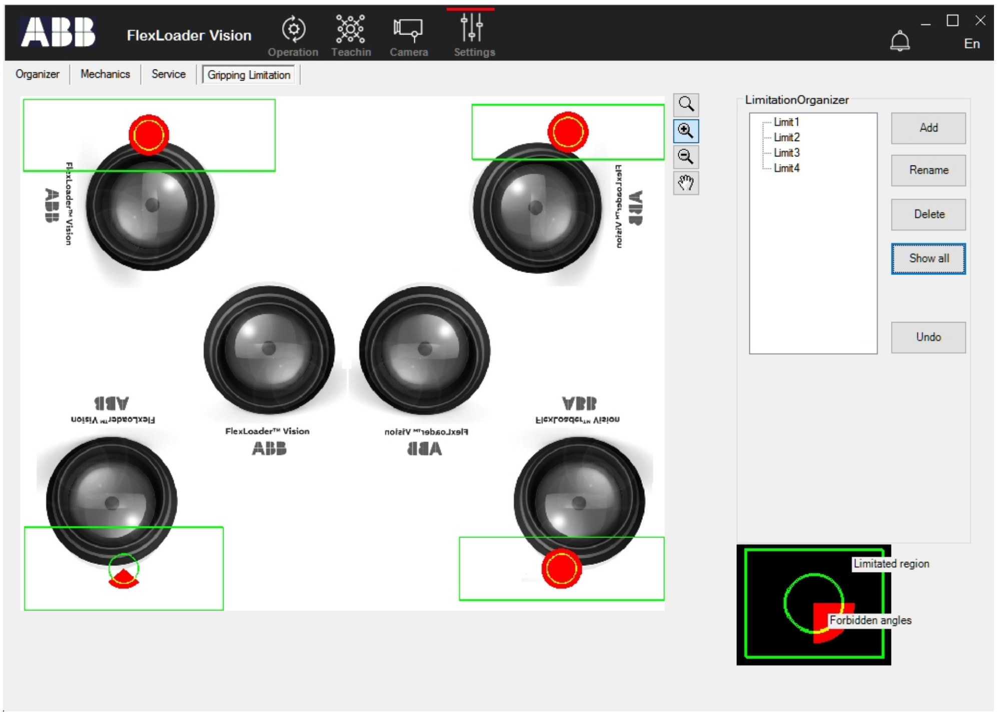
Click **Shutdown** to turn the computer off.

8.4 Gripping Limitations

Standard use

In some cases external circumstances can make it undesirable to pick or handle certain parts or certain orientations in the field of view. For example, you may wish to avoid picking or handling parts that have a tendency to roll if they are lying at certain angles.

The gripping limitations can be used to define an unlimited number of limitations for the system. These limitations can be applied during the teachin process of a detail.

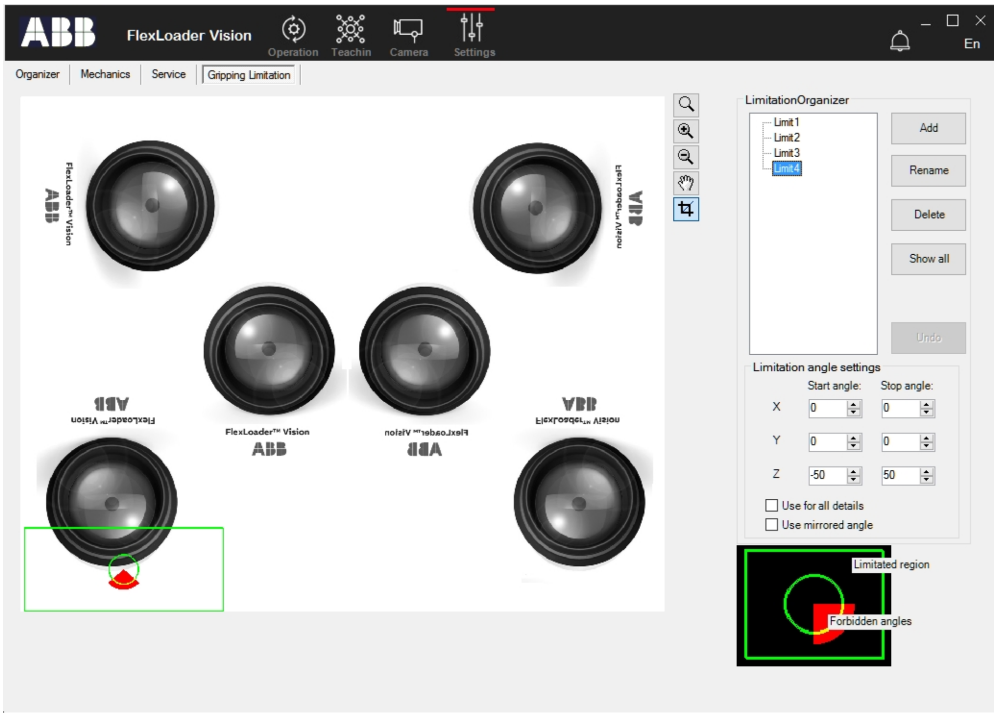


xx1800000251

The list to the right in the image above shows the limitations that are currently defined for the system.

Continues on next page

Click **Add** to add a gripping limitation for the system. Enter a suitable name for the limitation and click **OK**.



Limitation angle settings

	Start angle:	Stop angle:
X	0	0
Y	0	0
Z	-50	50

☐ Use for all details
☐ Use mirrored angle

xx1800000252

Use the Define position tool to define a box around the area where you want the limitation to apply. Use the **Start angle** and **Stop angle** settings for the x, y and z rotations to specify that the limitations only apply to certain angles within the selected area.

The z angles that are limited are marked in red, see image above. The section available for each defined tool may not be shown in the red area. At the bottom right of the image there is also explanatory text regarding the angles.

If several different limitations are defined for the system you can show individual limitations by clicking on their names in the field to the right. If you want to see all defined limitations, click **Show all**.

The **Rename** button can be used to change the name of a limitation.

To remove a limitation, click **Delete**. Note that if you delete a defined limitation that has been enabled for one or more details, it will cease to apply to all those details. You will still be able to run the detail in the system, but without the deleted limitation. If a defined limitation has been enabled for one or more details you will be asked for confirmation before it is deleted.

The check box **Use for all details** activates the selected limit for all details in the system. Note that as long as it is selected, it is not possible to deselect the limitation of a single detail.

If **Use mirrored angle** is checked, the limited area will also contain a reflection of the limited angle. The reflected section of the limitation will also appear in the image so that it is easy to understand what is covered.

Continues on next page

During operation, details are first located and the gripper image is positioned on the detail. A detail is regarded as limited if all of the following conditions are fullfilled.

- The TCP is inside the limitation rectangle.
- The gripper x-axis (blue arrow) is pointing towards the red angular range.
- The gripper inclination (rotX and rotY) are inside the defined range.

Extended use

The concept of gripping limitations can be extended in a number of specialized ways.

In some cases, certain areas of the image contain information that disturbs the image analysis. In such areas the gripping limitations can be used to change the image content. The gripping limitation areas can be replaced by white, black or distorted areas.

BlackRegion and **WhiteRegion**

To replace the gripping limitation area by a white area, give the area a name starting with **WhiteRegion**.

To replace the gripping limitation area by a black area, give the area a name starting with **BlackRegion**.

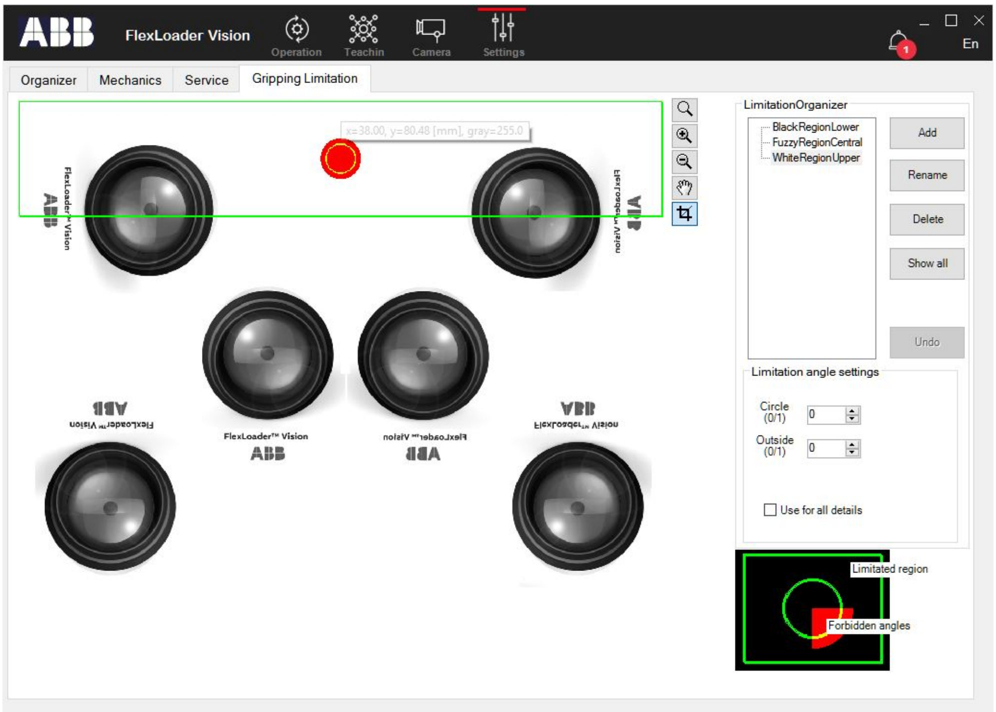
When the gripping limitation area is named this way, the **Limitation angle settings** pane is enabled.

The circle parameter determines if the area is rectangular (**Circle** = 0) or rounded (**Circle** = 1).

The outside parameter determines if the area is drawn inside the region (**Outside** = 0) or outside the region (**Outside** = 1).

Continues on next page

Select **Use for all details** if you want to enable the setting for all details.



Circle (0/1) 0

Outside (0/1) 0

□ Use for all details

xx1800000965

FuzzyRegion

To replace the gripping limitation area by a distorted unrecognizable representation of its content, give the area a name starting with **FuzzyRegion**.

When the gripping limitation area is named this way, the **Limitation angle settings** pane is enabled.

The **Erode (N)** parameter determines how many times bright objects are first eroded (typically 1–3).

The **Dilate (N)** parameter determines how many times bright objects are finally dilated (typically 1–3).

Select **Use for all details** if you want to enable the setting for all details.

Suitable parameters must be found in each project.

8.5 Master/Slave

Introduction

If required, FlexLoader Vision can communicate with the surrounding equipment as Slave. As Slave, part of the FlexLoader Vision functionality is controlled via commands from the external equipment. A number of parameters must be set, depending on the connection type and communication type, as described below. For this purpose, the **Master** / **Slave** tab appears on the **Settings** menu.

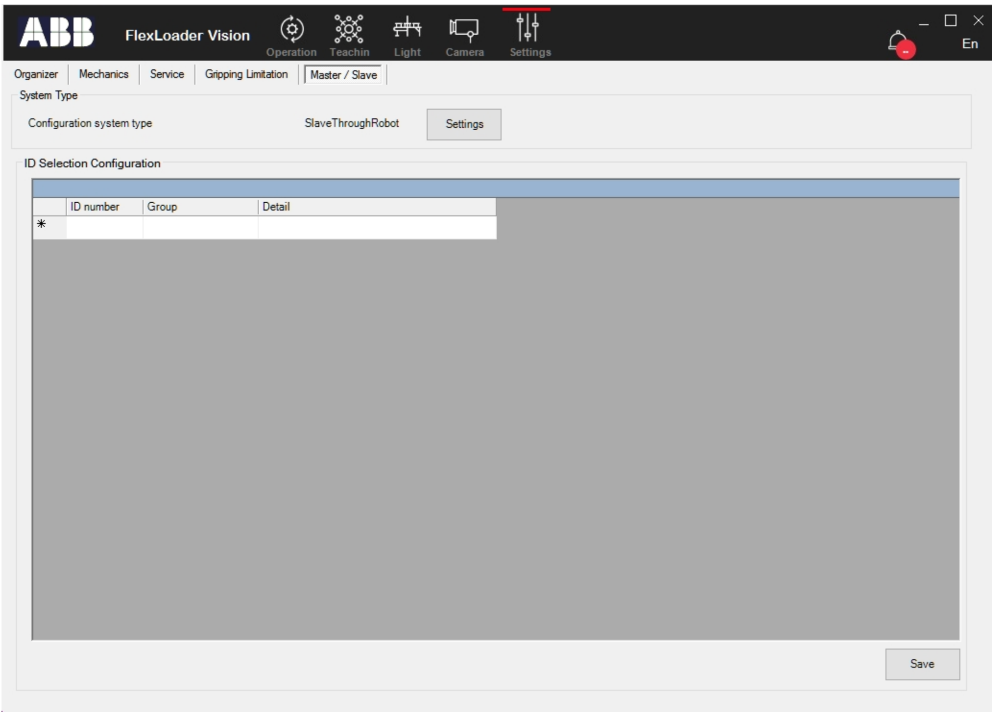
Master

The implementation of a Master function is preferably done in the robot program belonging to the detail. Communication with the surroundings occurs via the robot's I/O system.

Slave (through robot)

FlexLoader Vision communicates using one of the robot's I/O units, for example through a Profibus-Slave card with Master equipment.

The **SlaveThroughRobot** settings are used to set the translation between the ID number and the FlexLoader Vision detail. The Master equipment can choose one of the teachin details for operation according to a translation table that links the ID number of the Master equipment to a group and a detail in FlexLoader Vision. This is done by entering an ID number in the **ID Selection Configuration** pane, choosing a group from a drop-down list in the **Group** column, and then choosing one of the available details from the drop-down list in the **Detail** column.



Configuration system type SlaveThroughRobot

ID number	Group	Detail

xx1800000258

Continues on next page

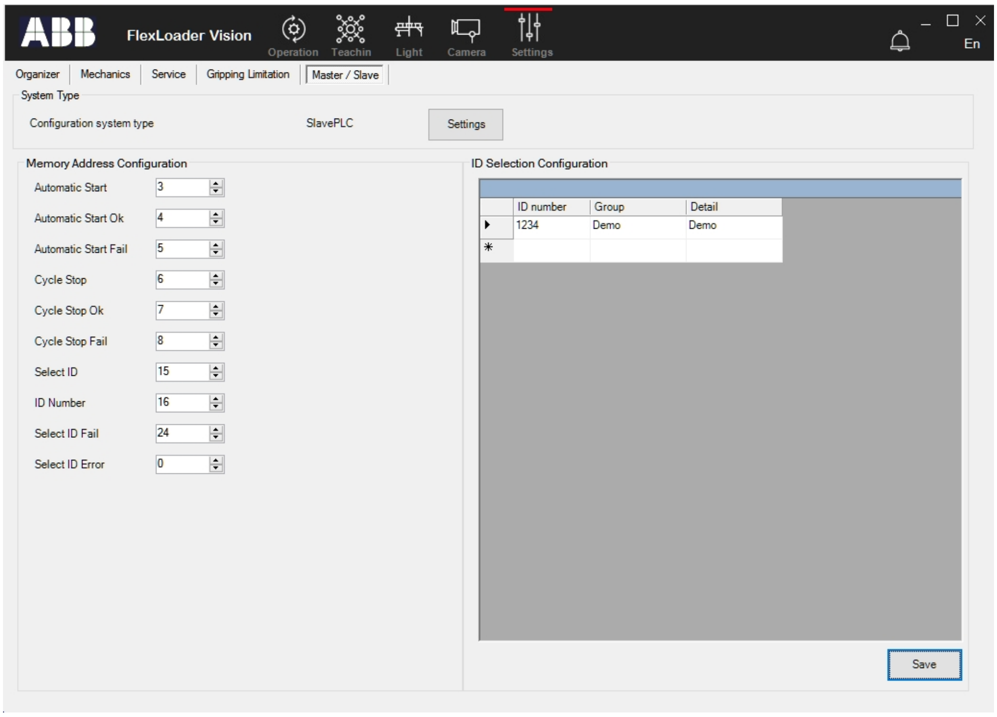
For communication with the Master control system, special signal names in the robot must be used, see *Communication information on page 258*.

Slave (PLC)

FlexLoader Vision communicates via the network with a Master PLC (Siemens S7).

The **SlavePLC** settings are used to set the translation between the ID number and the FlexLoader Vision detail, and to set the communication parameters.

The Master equipment can choose one of the teachin details for operation according to a translation table that links the ID number of the Master equipment to a group and a detail in FlexLoader Vision. This is done by entering an ID number in the **ID Selection Configuration** pane, choosing a group from a drop-down list in the **Group** column, and then choosing one of the available details from the drop-down list in the **Detail** column.



System Type

Configuration system type: SlavePLC

Memory Address Configuration

Automatic Start	3
Automatic Start Ok	4
Automatic Start Fail	5
Cycle Stop	6
Cycle Stop Ok	7
Cycle Stop Fail	8
Select ID	15
ID Number	16
Select ID Fail	24
Select ID Error	0

ID Selection Configuration

ID number	Group	Detail
1234	Demo	Demo

xx1800000259

To set the communication settings for PLC, click the **Settings** button. Enter the IP address for the PLC and click **OK**.

To communicate with the Master control system, the PLC memory addresses must be entered for each command and associated confirmation signal. The memory addresses are assumed to be “bytes” (8-bit memory locations), with the exception of a selection of ID-numbers and ConfirmIDs, both of which are “double word” (32-bit memory locations).

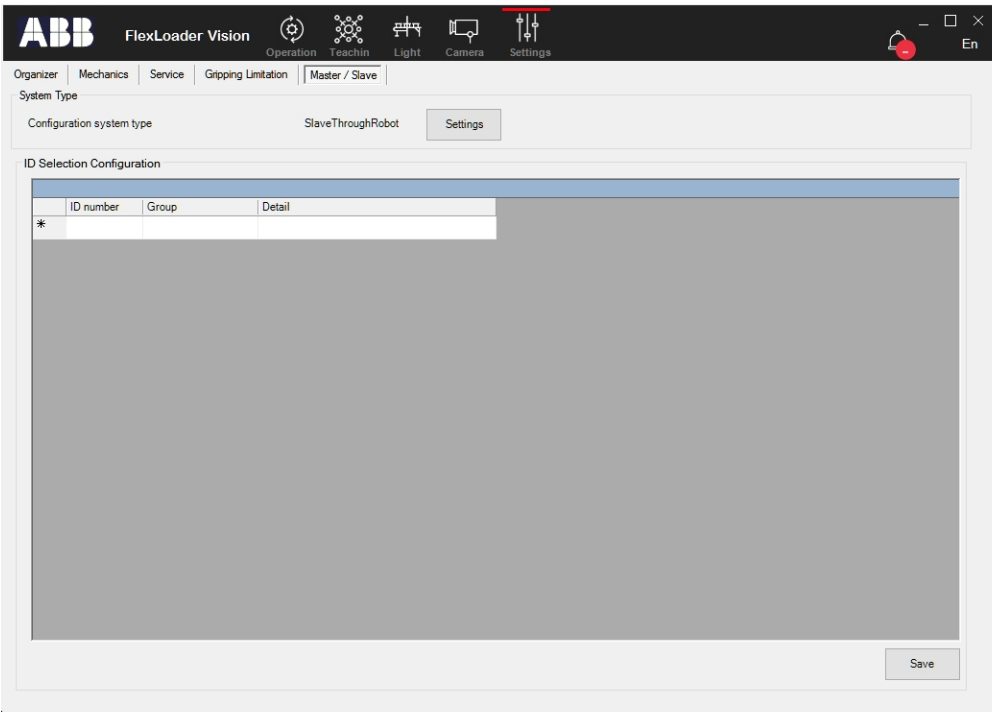
Slave (through TCP/IP)

FlexLoader Vision communicates with the Master software via network TCP/IP socket communication.

Continues on next page

The **SlaveThroughTCPListener** settings are used to define the IP adress and the port number for the communication.

This form of master/slave communication is used for operation with the Barcode handler, the Master handler, and the Batch handler.



System Type

Configuration system type: SlaveThroughRobot

ID Selection Configuration

	ID number	Group	Detail
*			

xx1800000258

This page is intentionally left blank

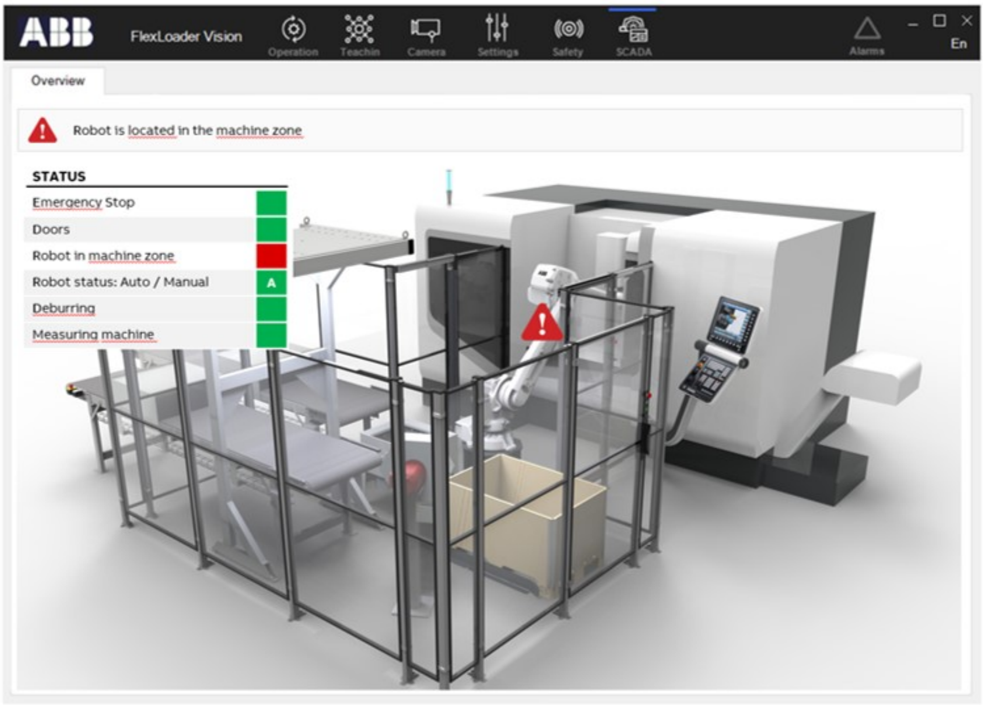
9 Safety, SCADA and Browser

Safety

The **Safety** menu tab (if enabled and configured) shows an overview of the robot cells safety related data. Integrators can configure the graphical appearance and select which data to show.

The data is retrieved from the robot controller.

The safety visualization of a robot cell might look like the example below.



Robot is located in the machine zone

STATUS	
Emergency Stop	🟩
Doors	🟩
Robot in machine zone	🟥
Robot status: Auto / Manual	A
Deburring	🟩
Measuring machine	🟩

xx1800002967

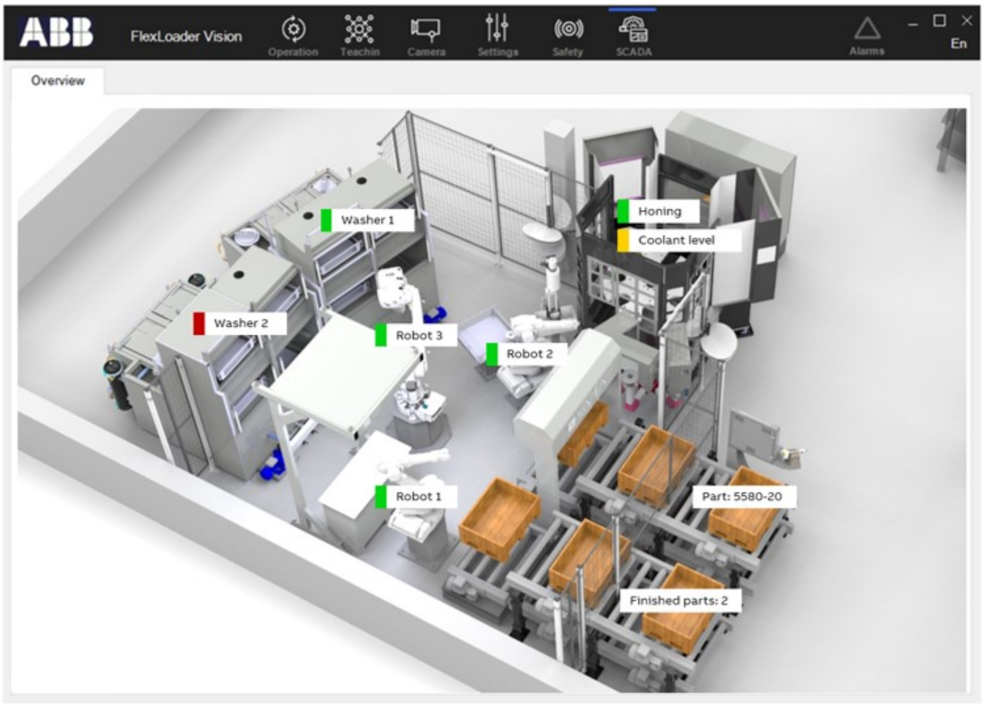
SCADA

The **SCADA** menu tab (if enabled and configured) shows an overview of the robot cell and its related data. Intuitive information display enables improved operator interaction with the cell. Integrators can configure the graphical appearance and select which data to show.

The data can be retrieved from the robot controller and OPC-UA hosts.

Continues on next page

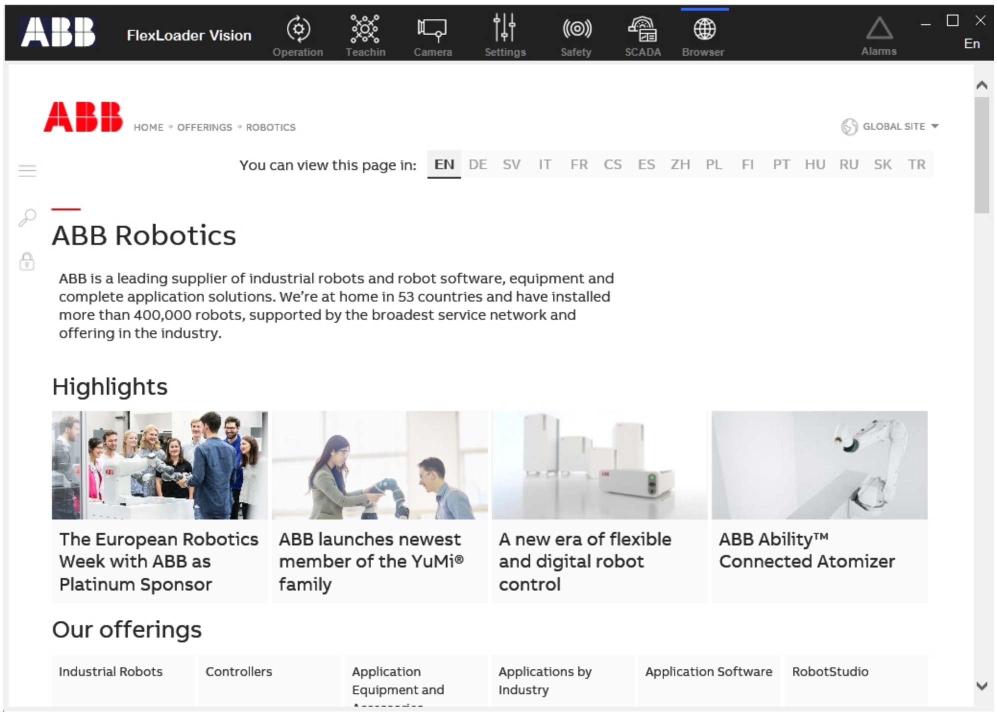
The SCADA visualization of a robot cell might look like the example below.



xx1800002966

Browser

The **Browser** menu tab (if enabled and configured) shows a web page, e.g. the information page of connected equipment or other process information. Integrators can configure which data to show.



You can view this page in: EN DE SV IT FR CS ES ZH PL FI PT HU RU SK TR

ABB Robotics

ABB is a leading supplier of industrial robots and robot software, equipment and complete application solutions. We're at home in 53 countries and have installed more than 400,000 robots, supported by the broadest service network and offering in the industry.

Highlights

The European Robotics Week with ABB as Platinum Sponsor

ABB launches newest member of the YuMi® family

A new era of flexible and digital robot control

ABB Ability™ Connected Atomizer

Our offerings

Industrial Robots | Controllers | Application Equipment and Accessories | Applications by Industry | Application Software | RobotStudio

xx1800002965

10 Troubleshooting

10.1 Problems with gripping accuracy

Introduction

There can be many different reasons behind problems with gripping accuracy, for example problems with the teachin, problems with the calibration, problems with the robot, problems with leaving details. A number of common fault sources and remedies are described below.

Gripping with general poor accuracy, without a visible pattern

- Faults up to +/-1 mm are possible and do not constitute a fault. Similarly, large faults in systems with large image fields may occur.
- Faults up to +/-5 mm can occur in 3D camera systems. This depends largely on form and surface properties of the details that are identified.
- Poor gripping accuracy could depend on the settings on the **Mechanics** tab on the **Teachin** menu. If the details easily start rolling, the image delay parameter needs to be increased. The camera belt must be still and the details must be stationary when the image is taken.
- Check the gripper fingers. Are they parallel? Does the gripper open and close smoothly? Is the gripper secured properly? Are the guide pins in position?
- Check that the teachin for the detail is correct: The light contrast must be good and the red lines must follow the edges of the detail carefully. Also check the parameter settings, especially **Score**, **Score target**, **Speed**, **Timeout** and **Accuracy**.
- Check that the settings for the perspective center and the camera height are correct.
- Recalibrate FlexLoader Vision and the robot. Check that the data for the calibration pattern is correctly specified. To prevent the calibration plate from moving during calibration it can be taped to the camera belt. Check that the dot for coordinate 0,0 is visible at the bottom left of the screen. Also check that the robot calibration has been carried out correctly (see the robot manual and *Robot integration on page 221*).
- Check that the robot TCP is correctly defined. For more information on this, see the robot manual.
- Verify that the motor parameters (calibration offset) are correct and valid.
- If you load a backup in the robot you have to recalibrate the camera afterwards if the last calibration was made after the backup file was created.
- If the camera has been moved, it must be recalibrated.
- If the camera lens has been adjusted, removed or otherwise changed, the camera must be recalibrated.
- Check that the camera, the lens and the fixing brackets are secured properly.
- The robot and its base must be properly bolted into place. Expander bolts can creep out of their holes if the floor is bad.

Continues on next page

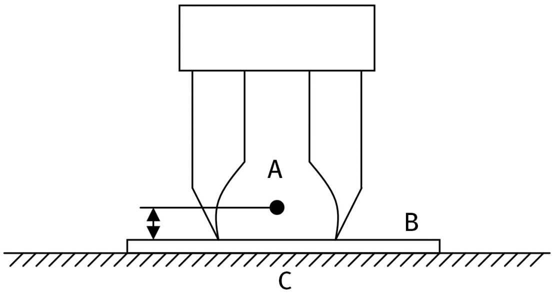
- If the robot has collided with a detail or been exposed to movement in the processing machine there is a risk that the robot's mechanics are changed so much that it has to be recalibrated. Read the robot manual for further information about what to do to check and remedy this. Note that it is normally not necessary to perform a fine calibration.

Gripping almost correctly, but always offset

- Note in which position the fault occurs. Check that the grip point is correctly set in that part of the detail.
- If the grip point is correctly set, the calibration between the robot and FlexLoader Vision can be offset. Recalibrate according to the above.

Positioned slightly wrong when gripping

- Check that the camera is perpendicular to the picking area. This is very important for the system.
- Check that the camera height (in mm) over the picking area matches the specified value. If not, change the value so that it corresponds with the reality.
- Check that the robot's coordinate system by the camera is correct. This is done by using the robot and jogging to a point on or just above the belt. Then the z-value specified in the coordinate system for the camera should be 0 (or close to 0).



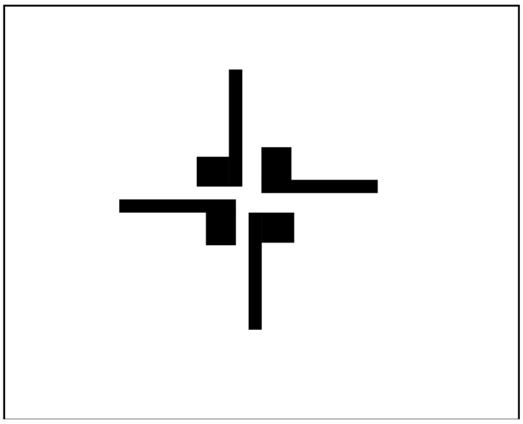
xx1800000323

Pos.	Description
A	TCP
B	Calibration plate
C	Camera belt

- Check if the intensity of the ambient light falling into the picking area is casting strange shadows from the detail. If so, screen off the light.
- Position four details in the middle of the picking area, according to the diagram below (so that they can be gripped by the robot). Place all four details in the same position but rotated by 90°. If the robot grips slightly wrong, but different on all these details, it may indicate that the TCP for the gripper is

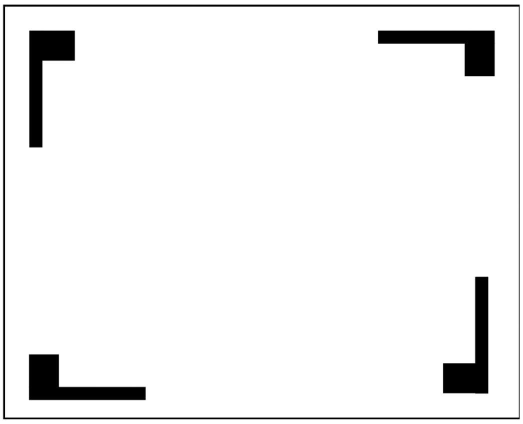
Continues on next page

incorrect in the x and y axes and a new calibration of the tool must be carried out.



xx1800000325

- Position four details in the corners of the picking area, according to the diagram below. Try to grip these details. If this leads to positioning faults it may indicate that the camera position needs to be adjusted according to the above or that the set camera height or machine height is incorrect.



xx1800000324

10.2 Identification problems

FlexLoader Vision does not find any details

- During operation: Check that the correct detail is selected.
- During teachin: If the detail is too light or shiny it can be difficult for the system to discern the detail from a conveyor that is light in color. This can result in no detail being found. The same problem can occur with a dark detail and a black conveyor. To test if the detail is too light, a matt black disc or similar can be placed temporarily underneath the camera and a new teachin carried out. If this works, the problem can be solved by changing to a dark conveyor. Note that the operation of previous teachin details may be affected by the new conveyor color.

FlexLoader Vision identifies an incorrect position

- Redefine the masked surface. Try to find an identification surface of the present position that markedly differs from other positions.

10.3 Communication problems

FlexLoader Vision PLC

The alarm **PLC could not be initialized** is displayed at start up

- Check that the PLC can be reached via the network: Run **ping 192.168.125.151** (adjust the IP address to the relevant system settings) in a command window. If the PLC does not respond, check the network wiring and the switch.
- In FlexLoader Vision ConfigurationEditor, check that the IP-address for the PLC is correct, see *Configuration editor on page 157*.

FlexLoader Vision robot

FlexLoader Vision has no contact with the robot

- Check that the robot can be reached via the network: Run **ping 192.168.125.1** (adjust the IP address to the relevant system settings) in a command window. If the robot does not respond, check the network wiring, switch and the network settings of the robot.
- Check that the robot has all necessary options and hardware installed to communicate via the network.
- Check that Windows firewall is configured correctly.

Messages about missing robot signals

- Check if the named signal is defined in the robot.

Messages about missing robot variables

- Check if the named variable is defined in the robot.

10.4 Camera problems

The system cannot connect to the camera

- Check that the camera can be reached via the network: Run **ping 192.168.2.2** (adjust the IP address to the relevant system settings and desired camera) in a command window. If the camera does not respond, check the network wiring, switch, and the network settings of the camera.

The camera image is far too light or too dark.

- Check the lighting.
- Go to the image settings and check the settings for contrast, brightness and gain.
- If the image is still too light or too dark, adjust the aperture. Do not forget to recalibrate the robot and vision system when you have adjusted the camera.

11 Installation

11.1 PC installation

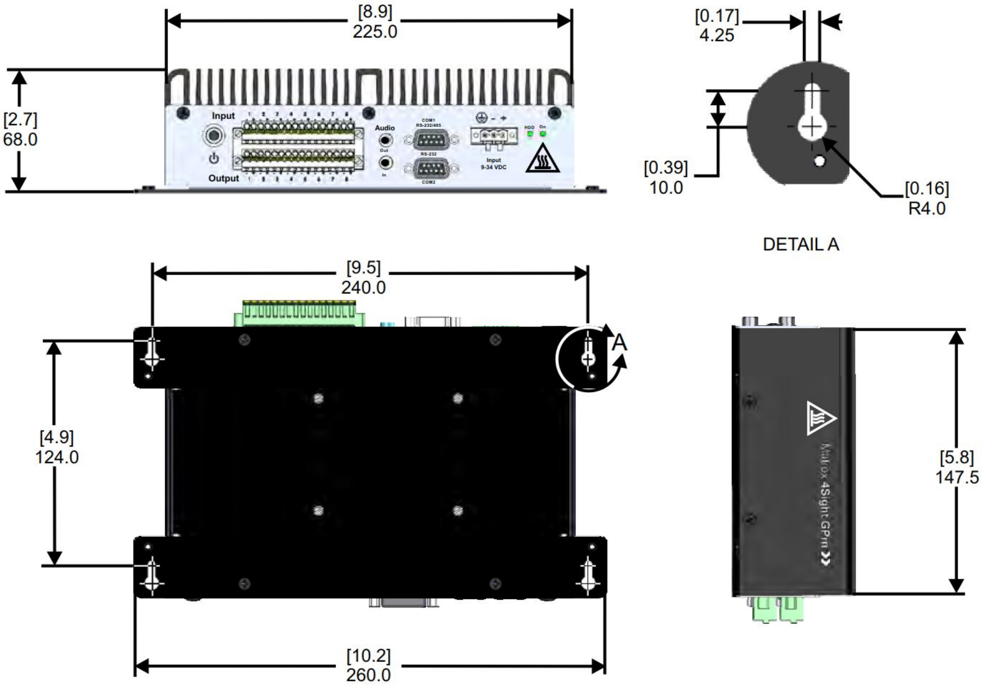
Mechanical installation

Note

When FlexLoader Vision is pre-installed as part of a function package or standard cell, all hardware installation is already done.

The PC shall be mechanically installed in an electrical cabinet. It can be oriented vertically or horizontally. Ensure sufficient air circulation around the heat sink.

PC dimensions with suitable screw location information is shown in the following picture.



xx1800002002

For detailed information about the PC please refer to the manufacturers manual. It is located in the FlexLoader Vision documentation folder on the PC.

Electrical installation

WARNING

Be sure to turn off the power to the unit and all peripherals, as well as unplug the power from the unit before adding or removing devices or cameras.

Connect other peripherals like mouse, display and keyboard to the PC.

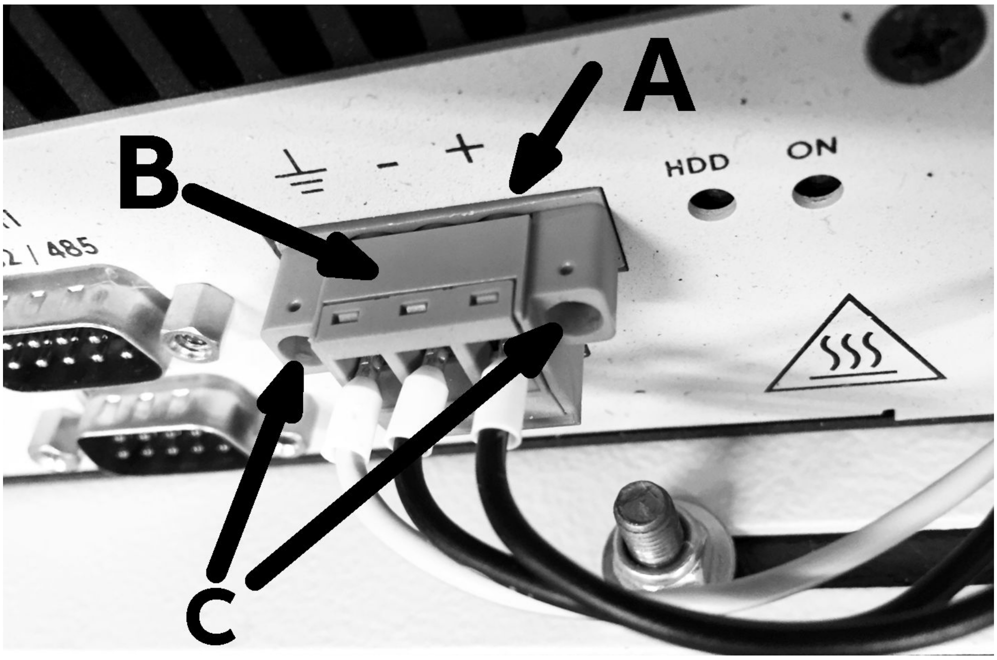
Continues on next page

Connect the PC to the robot with a network cable. Make the connection from connector LAN on the PC to the internal network connector LAN2 of the robot controller. If necessary use a network switch.

We recommend a power supply with 24 VDC, supporting the maximum power-up current of 3.6 A.

Prepare the power connector latch and connect both power supply and protective earth to it according to the designation shown on the PC.

Use the screws on the power connector latch to secure it into place in the power connector.



xx1800002004

Pos.	Description
A	Power connector
B	Power connector latch
C	Screws

Switch on power and check that the PC is booting as expected.

Note

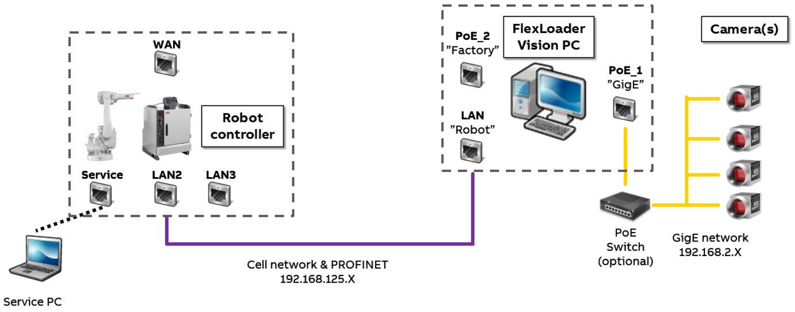
FlexLoader Vision software is always pre-installed on the PC.

Continues on next page

Network overview

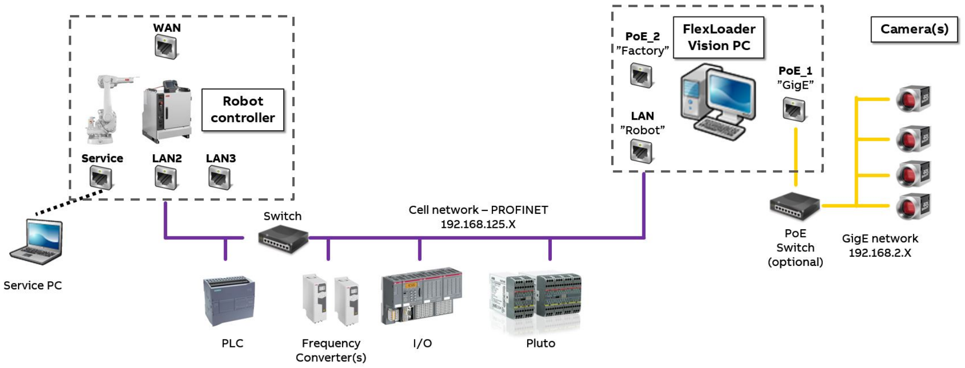
The following image shows the basic network setup between FlexLoader Vision and the robot on the internal robot network.

For detailed information on network settings see *Standard network IP settings on page 177*.



xx1800002351

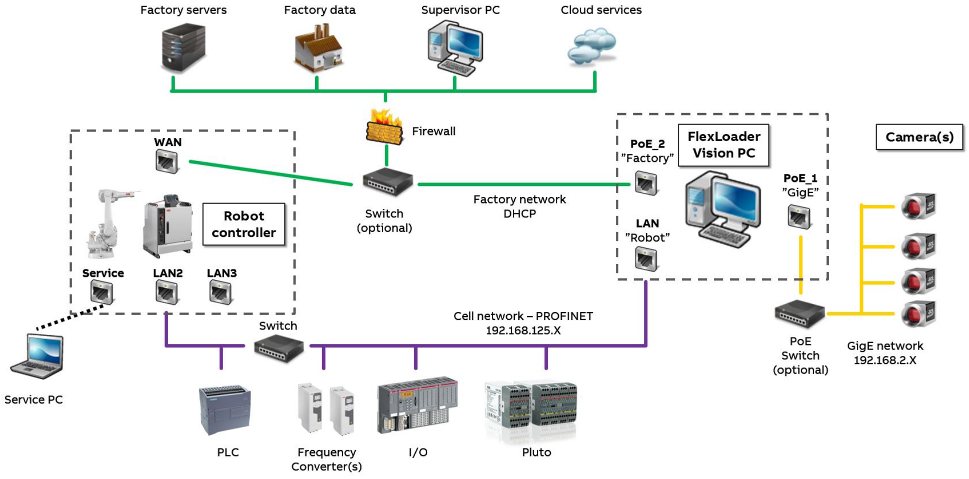
Feeders, safety units and I/O are typically connected to the same internal network, using PROFINET (robot option 888-2).



xx1800002352

Continues on next page

The robot and the FlexLoader Vision PC can be connected to a separate factory network as shown below.



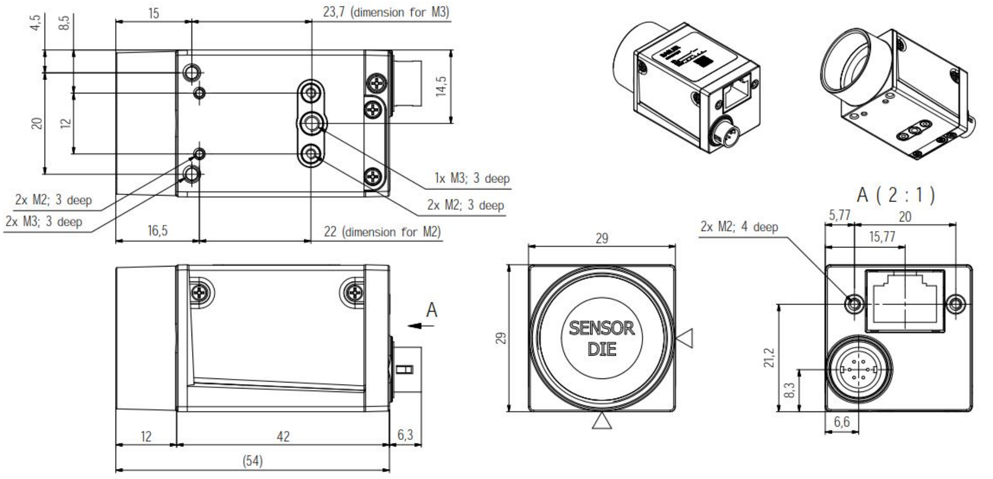
xx1800002353

For connections of an external PROFINET network see *External PROFINET connections on page 178*.

11.2 Basler camera mechanical and electrical installation

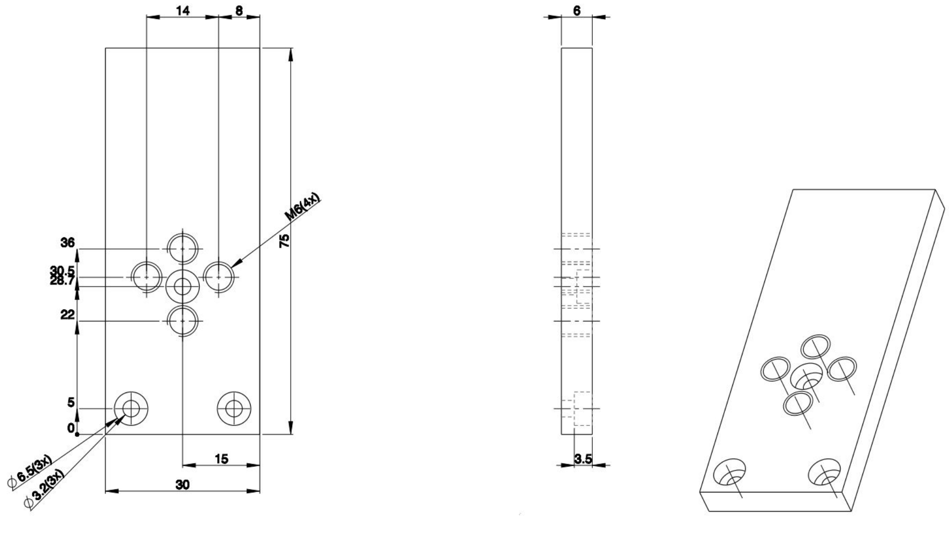
The camera shall be mechanically installed at a suitable location. It is normally oriented aiming downward or forward.

Camera dimensions with screwlocation information is shown in the following picture.



xx1800002001

Fit the camera to the isolating plastic mounting plate that is delivered with FlexLoader Vision and install it at the intended location.

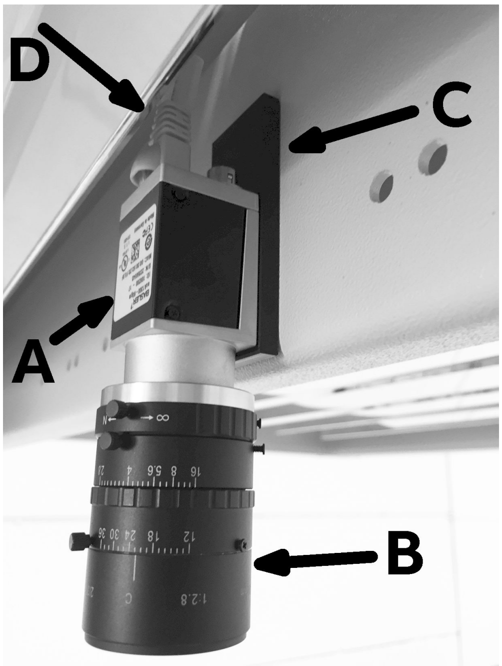


xx1800002006

Connect the camera to the **PoE1** network port of the PC with a network cable (cat6). We recommend the use of shielded network cables. The power to the camera is

Continues on next page

provided by PoE (Power over Ethernet). If more than one camera is used, a suitable PoE switch shall be used (IEEE 802.3af).



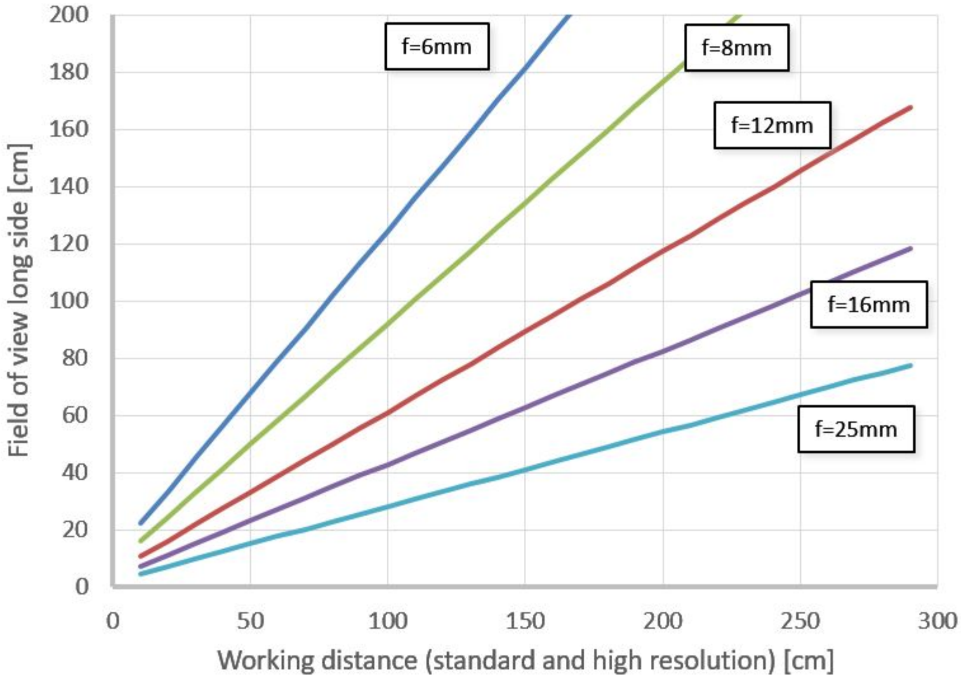
xx1800002003

Pos.	Description
A	Camera
B	Lens
C	Isolating plate
D	Ethernet cable connection

The field of view of the camera depends on the height of the camera (i.e. working distance) and the focal length of the lens. Approximate values for the size of the

Continues on next page

field of view (long side) for different focal lengths and camera height is shown in the following diagram.



xx1800002005

For detailed information about the Basler camera please refer to the manufacturers manual. It is located in the FlexLoader Vision documentation folder on the PC.

11.3 Cirrus 3D camera mechanical and electrical installation

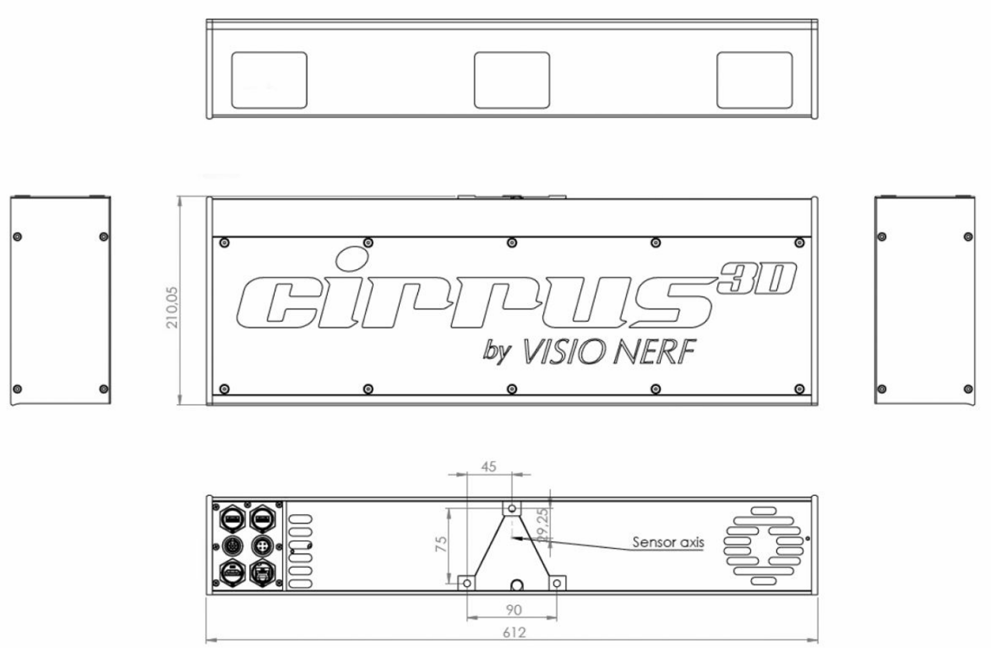
11.3.1 Mechanical installation

The camera shall be mechanically installed at a suitable location. It is normally oriented aiming downward.

Camera dimensions with screw location information is shown in the following picture. Ensure that the camera is fixed against a flat surface with screws of appropriate length.

Keep the air inlet and outlet free for air flow.

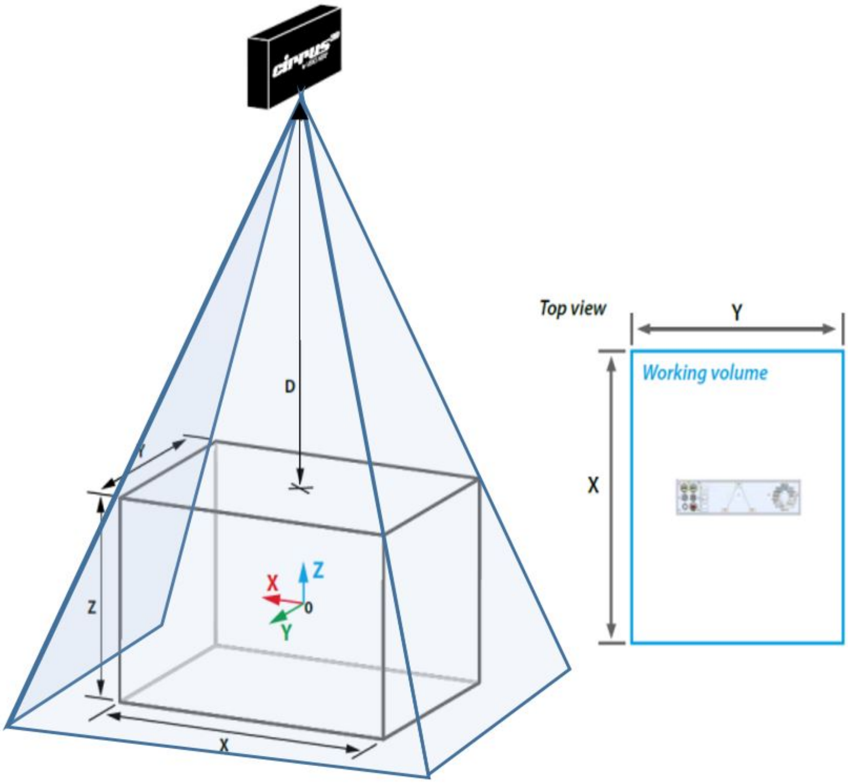
The Cirrus sensor contains no user serviceable part inside.



xx1800002007

Continues on next page

The general field of view of the 3D camera is fixed. For detailed geometry see *Field of view geometry on page 153*.



xx1800002009

Pos.	Description
X	Length of volume of view: 1200 mm
Y	Width of volume of view: 1000 mm
Z	Height of volume of view: 1000 mm
D	Stand-off distance to volume of view: 1900 mm
	Maximum distance: 2900 mm

Different volumes of view can be achieved by using other sensors in the Cirrus family.

Tip

Pallets shall be preferably placed as close as possible to the sensor, without violating the minimum stand-off distance. This will ensure maximum image resolution for a given pallet.

For detailed information about the Cirrus camera please refer to the manufacturers manual. It is located in the FlexLoader Vision documentation folder on the PC.

It is also available on the Cirrus desktop, in the folder **Documentation.**

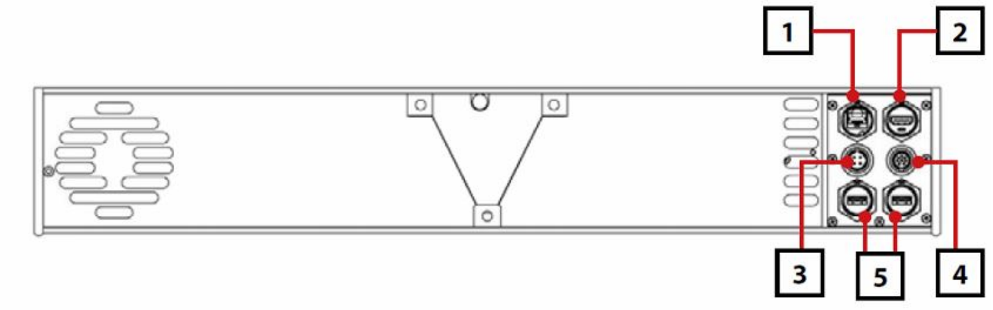
11.3.2 Electrical installation

Connect the camera to the **PoE1** network port of the PC with a network cable (cat6). We recommend the use of shielded network cables.

We recommend a stabilized power supply with 24 VDC, 10 A. Prepare the Power cable according to the following table.

Wire color	Designation
Black	0 V
Black	0 V
Red	24 VDC
Red	24 VDC

Connect the Power cable to the camera, then switch on the power.



xx1800002008

Pos.	Description
1	Network cable connector
2	Connector for HDMI (<= 1920×1080) cable
3	Connector for Power cable
4	Connector for I/O cable
5	USB device connector (optional)

If necessary, an optional HDMI display and USB mouse/keyboard can be connected to the camera.

Tip

Note that IP65 rating is achieved only if VisioNerf cables are used and if all connectors are used. If some connectors are unused, a protective cap should be installed to keep the IP65 rating.

When IP65 rating is not required, you can connect standard cables (USB, Ethernet, HDMI) to the Cirrus sensor.

The shield of the power supply must be properly connected to the ground by using a dedicated grounding clamp.

11.3.3 Field of view geometry

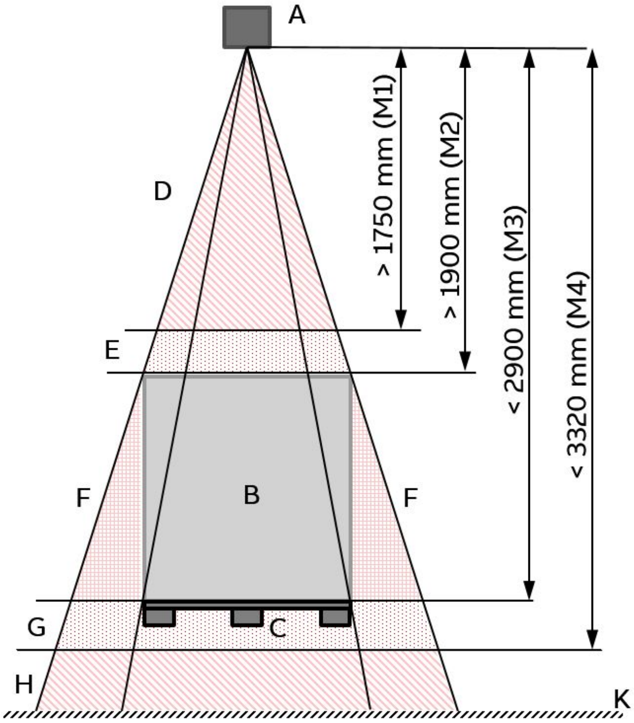
Introduction

The Cirrus sensor has a volume of view with fixed geometry. Objects within the volume of view can be measured normally.

Objects/surfaces situated in certain areas can influence the measurement and result in bad data quality and erroneous data points.

Basic geometry

The following schematic view is valid for the standard Cirrus sensor (Cirrus 1200).



xx1800002586

Pos.	Description
A	Cirrus sensor
B	Nominal volume of view Length 1200 mm, width 1000 mm, height 1000 mm
C	Bin pallet, not part of volume of view
D	Upper standoff volume. Never place any object or surface in this volume. Other object data will become corrupt.

Continues on next page

Pos.	Description
E	Upper standoff safety volume. Normally, do not place any object or surface in this volume. Measurement data may be obtained in some circumstances, but we strongly advise not to use this volume. Other object data may become corrupt.
F	Outer measurement volume. The size is varying with height. Measurement data may be obtained from this volume, but data points may be of less quality. A typical use of this volume is for oversized bins (length and width) with reduced height requirements.
G	Maximum distance safety volume. Normally, do not place any object or surface in this volume. Measurement data may be obtained in some circumstances, but we strongly advise not to use this volume. Other object data may become corrupt.
H	Maximum distance forbidden volume. Never place any object or surface in this volume. Other object data will become corrupt.
K	Floor

Alternative Cirrus sensors

A number of Cirrus sensors is available on request. They have different field of views and different standoff.

	Cirrus 1200	Cirrus 1600
Nominal volume of view	Length 1200 mm, width 1000 mm, height 1000 mm	Length 1600 mm, width 1200 mm, height 1200 mm
Minimum safe distance (M1)	1750 mm	2350 mm
Nominal standoff (M2)	1900 mm	2500 mm
Nominal maximum distance (M3)	2900 mm	3700 mm
Maximum safe distance (M4)	3320 mm	3850 mm

Recommendations

Bin placement close to sensor

Pallets shall be preferably placed as close as possible to the sensor (only leaving a small safety margin), without violating the minimum stand-off distance.

This will ensure maximum image resolution for a given pallet. Additionally, the influence of other light sources is reduced.

No surfaces beyond maximum distance forbidden volume

Ensure that no object or surface is located within the maximum distance forbidden volume. Else this will lead to corrupted measurement data.

A shield or similar device has to be used if bins are to be placed on conveyors or other elevated positions.

The impact of out-of-volume surfaces can be reduced (but not fully eliminated) by adjusting the region of interest box in the acquisition configuration.

Continues on next page

Shiny surfaces

Shiny surfaces pose problems to most 3D sensors, also the Cirrus. Avoid scanning of shiny objects in bins.

The reflection of bin walls in shiny objects will lead to erroneous data points in affected areas.

This page is intentionally left blank

12 Commissioning

12.1 Configuration editor

12.1.1 Introduction

FlexLoader Vision is able to handle a wide variety of machine types and working conditions, with one or more cameras, in various FlexLoader function packages and standard cells.

This section describes the available parameters.

CAUTION

In most cases, FlexLoader Vision is completely configured upon delivery. Only trained and experienced users should do changes to the FlexLoader Vision configuration. Errors in the configuration can result in malfunctioning of the system.

Note

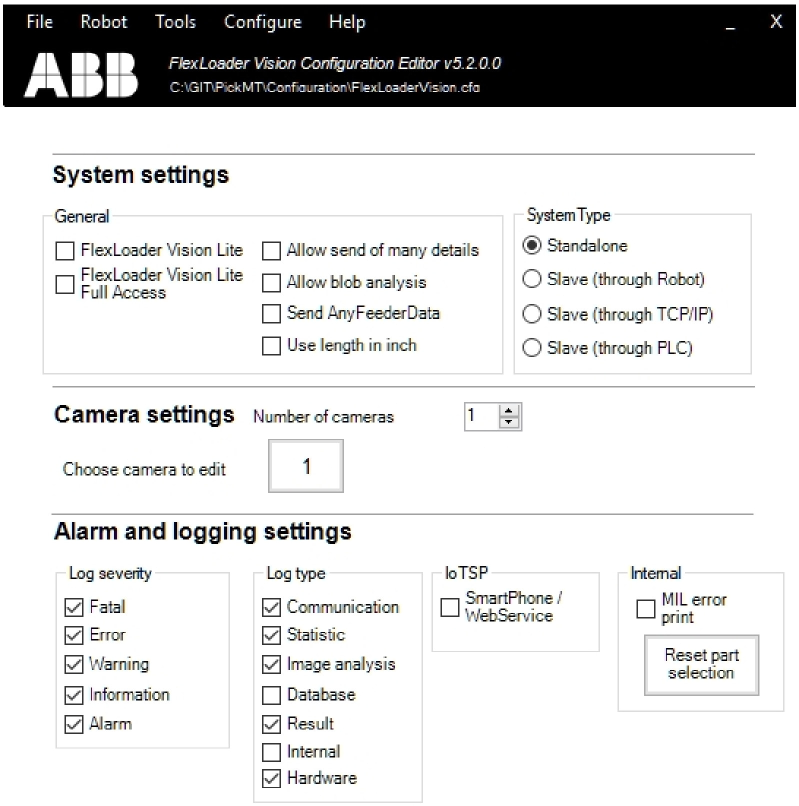
Some configuration features are used for testing and maintenance and are not to be used for production systems. Other features are used for handling of legacy functionality and should be avoided in new production systems.

12.1.2 Main settings

The main window is shown at startup. The default configuration file is loaded automatically.

The configuration file (**FlexLoaderVision.cfg**) is stored in the FlexLoader Vision configuration folder, typically located at **D:\Program Files\ABB Industrial IT\Robotics IT\FlexLoader Vision\Configuration\**.

Be sure to load and save the correct configuration file when specifying files manually.



FlexLoader Vision Configuration Editor v5.2.0.0
C:\GIT\PickMT\Configuration\FlexLoaderVision.cfg

System settings

General

□ FlexLoader Vision Lite
□ FlexLoader Vision Lite Full Access
□ Allow send of many details
□ Allow blob analysis
□ Send AnyFeederData
□ Use length in inch

System Type

● Standalone
○ Slave (through Robot)
○ Slave (through TCP/IP)
○ Slave (through PLC)

Camera settings

Number of cameras: 1

Choose camera to edit: 1

Alarm and logging settings

Log severity

☑ Fatal
☑ Error
☑ Warning
☑ Information
☑ Alarm

Log type

☑ Communication
☑ Statistic
☑ Image analysis
□ Database
☑ Result
□ Internal
☑ Hardware

IoTSP

□ SmartPhone / WebService

Internal

□ MIL error print

Reset part selection

xx1800000433

System settings	Description	Default
General		
FlexLoader Vision Lite	Enables the use of FlexLoader Vision Lite features. General teachin is not accessible.	No
FlexLoader Vision Lite Full Access	Enables the use of FlexLoader Vision Lite features. General teachin features are accessible as well.	No
Allow send of many details	Normally only one coordinate is sent to the robot. This flag enables sending of several coordinates to the robot (see *Advanced parameter settings on page 76*) Coordinates must be received and handled differently (see *RAPID program on page 230*).	No

Continues on next page

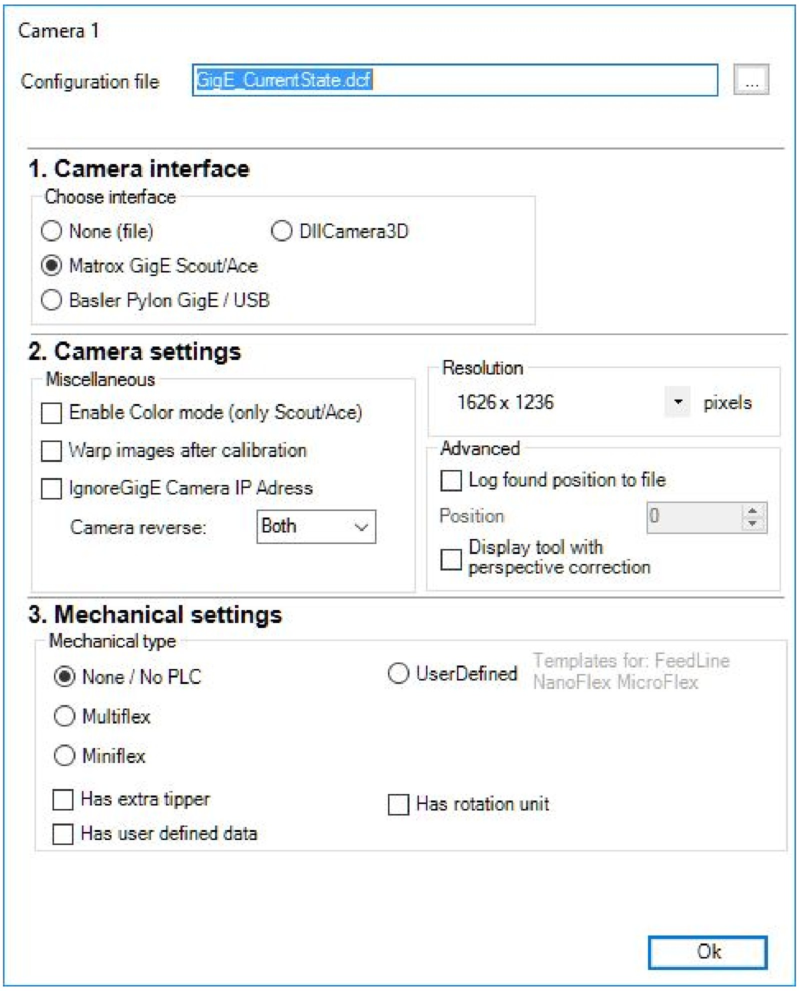
System settings	Description	Default
Allow blob analysis	Enable alternative way of identifying shapes based on their size and form. Limitations in accuracy and orientations apply.	No
Send AnyFeederData	Specialized use case. Transmits detail distribution within the field of view to the robot.	No
Use length in inch	Changes display of length units from metric to imperial system.	No
System Type	FlexLoader Vision can be externally controlled by other devices or software. Start, stop and selection of details is possible.	
Standalone	FlexLoader Vision operates as standalone system.	Yes
Slave (through robot)	FlexLoader Vision is controlled by the robot.	No
Slave (through TCP/IP)	FlexLoader Vision can be controlled by a TCP/IP based application.	No
Slave (through PLC)	FlexLoader Vision can be controlled by the PLC.	No

Camera settings	Description	Default
Number of cameras	Select the correct number of cameras. Click on the number associated to the camera to configure it.	1

Alarm and logging settings	Description	Default
Log severity	Selectable level of severity for logging.	All
Log type	Selectable types of logging.	All
IoTSP	Select **SmartPhone**/**WebService** if desired. This service needs an active subscription.	No
MIL error print	Used for system testing.	No
Reset part selection	Used for resetting saved selections for groups and details.	-/-

12.1.3 Camera settings

Click the camera number to open the camera settings window and adjust the settings for the corresponding camera.



Camera 1

Configuration file: GigE_CurrentState.dcf ...

1. Camera interface

Choose interface

○ None (file) ○ DllCamera3D
● Matrox GigE Scout/Ace
○ Basler Pylon GigE / USB

2. Camera settings

Miscellaneous

□ Enable Color mode (only Scout/Ace)
□ Warp images after calibration
□ IgnoreGigE Camera IP Adress
Camera reverse: Both

Resolution

1626 x 1236 pixels

Advanced

□ Log found position to file
Position 0
□ Display tool with perspective correction

3. Mechanical settings

Mechanical type

● None / No PLC ○ UserDefined
○ Multiflex
○ Miniflex
□ Has extra tipper □ Has rotation unit
□ Has user defined data

xx1800000434

Figure 12.1: Default settings for the camera

Camera interface	Description	Default
None (file)	Used for system testing.	No
Matrox GigE (Ace)	Used for standard Basler Ace cameras	Yes
Basler Pylon GigE/USB	Used for system testing.	No
DllCamera3D	Used for 3D camera communication.	No
Configuration file	Configuration file used for Matrox GigE and DllCamera3D.	GigE_Current-State.dcf

Camera settings	Description	Default
Enable color mode	Enables the use of color extraction and combination. Only possible with Matrox GigE Ace interface.	No

Continues on next page

Camera settings	Description	Default
Warp images after calibration	Can be used if the raw image from the 2D camera is significantly distorted due to lens distortion. This will rectify the image according to the calibration data. Changing this setting requires a renewed camera calibration. Using warped images takes additional processing time.	No
Ignore GigE camera IP adress	Used for system testing.	No
Camera reverse	Can be used to mirror camera images directly upon image acquisition. Useful if the camera orientation does not match the operator station. Only allowed with 2D camera. Changing this setting requires a renewed camera calibration.	None
Resolution	Used as default camera resolution if the camera interface **None** is selected. Used for system testing.	1280x1024
Log found position to file	Used for system testing. Activates logging of the specified position data to a separate file. Useful for testing of mechanical long term stability by teaching a fixed object within the field of view of the camera.	No
Display tool with perspective correction	Normally the gripper is drawn at a height location that corresponds to the mask height setting during teachin. Activate this setting if the gripper is to be drawn at a different height location corresponding to the actual grip height. The teachin for all details has to be renewed if this setting is changed.	No

3D camera settings

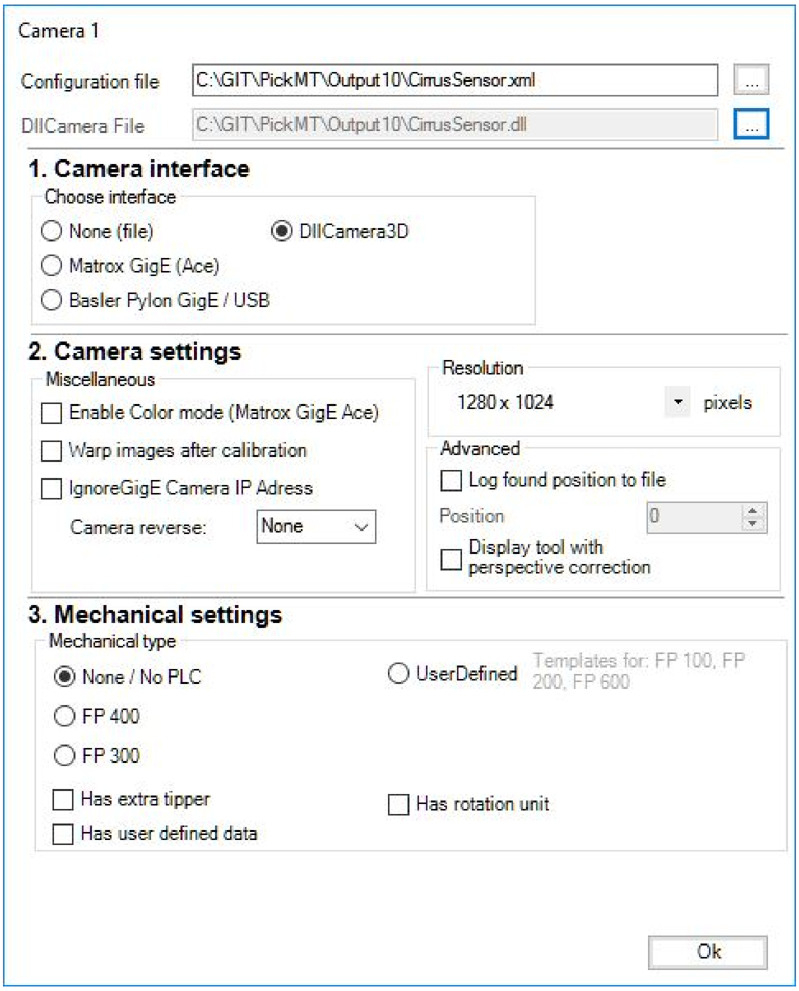
For 3D cameras, select **DllCamera3D**. Then specify the correct DLL file and XML configuration file. These files are provided by ABB.

Typically the files used are CirrusSensor.xml and CirrusSensor.dll.

3D camera settings

Camera interface	Description	Default
Matrox GigE Ace	Used for standard Basler Ace cameras	No
DllCamera3D	Used for 3D camera communication.	Yes
Configuration file	Configuration file used for DllCamera3D	Cirrus-Sensor.xml
DLL file	DLL file used for DllCamera3D	Cirrus-Sensor.dll

Continues on next page



Camera 1

Configuration file: C:\GIT\PickMT\Output10\CirrusSensor.xml

DllCamera File: C:\GIT\PickMT\Output10\CirrusSensor.dll

1. Camera interface

Choose interface

○ None (file) ● DllCamera3D
○ Matrox GigE (Ace)
○ Basler Pylon GigE / USB

2. Camera settings

Miscellaneous

□ Enable Color mode (Matrox GigE Ace)
□ Warp images after calibration
□ IgnoreGigE Camera IP Adress
Camera reverse: None

Resolution: 1280 x 1024 pixels

Advanced

□ Log found position to file
Position: 0
□ Display tool with perspective correction

3. Mechanical settings

Mechanical type

● None / No PLC
○ FP 400
○ FP 300
○ UserDefined Templates for: FP 100, FP 200, FP 600

□ Has extra tipper
□ Has user defined data
□ Has rotation unit

xx1800000439

Figure 12.2: 3D camera settings

Camera sharing settings

It is possible to share the same physical camera and use it as several logical cameras.

One single 2D camera (configured as Matrox GigE) can be shared to several logical cameras, and one or more Cirrus 3D (configured as DllCamera3D) can be shared to several Cirrus 3D.

For the 2D case, configure several cameras in the configuration editor, and select **Ignore GigE camera IP adress** for each camera. Configure the camera for IP address **192.168.2.2**.

Continues on next page

For the Cirrus 3D case, configure each camera as usual, with separate dll and xml files, but assign identical IP addresses to all cameras.

CAUTION

The application must ensure that no simultaneous image acquisitions are initiated on a single camera.

Simultaneous image acquisitions can lead to unpredictable and erroneous behavior.

Mechanical settings

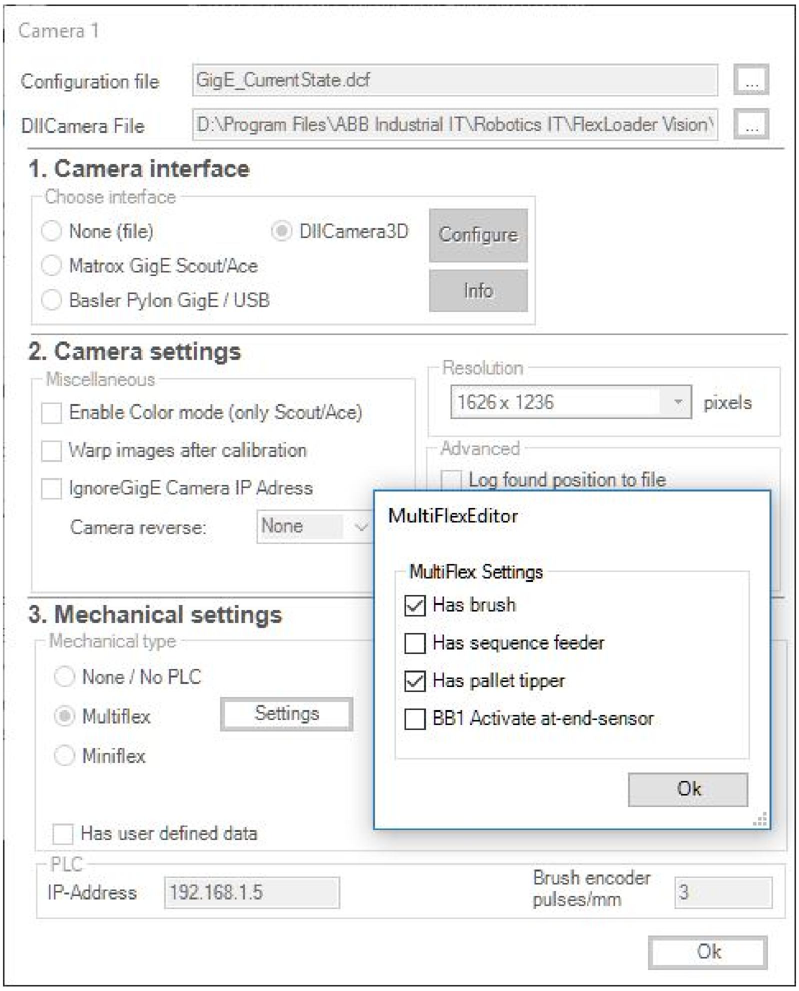
The mechanical settings determines visual information and communication settings. Information with regard to the mechanical feeder is shown during teachin and while working with system settings.

Communication information is used during system initialization and operation.

Mechanical settings	Description	Default
None/PLC	No data is exchanged with external PLC, and no information is shown on HMI.	Yes
FP 300	Selects FP 300 specific settings for this camera.	No
FP 400	Selects FP 400 specific settings for this camera.	No
UserDefined	Selects user-defined specific settings for this camera.	No
Has extra tipper	Can be selected for some feeders.	No
Has rotation unit	Can be selected for some feeders.	No
Has user defined data	Can be selected for some feeders.	No
Disable **Update PLC** during operation	The **Update PLC** button will normally not be available in TeachIn during operation. The button is only available when editing the same part that is running in operation or if no operation is ongoing.	No

Continues on next page

Each of the mechanical feeders have settings for their PLC IP address and feeder specific options.



Camera 1

Configuration file: GigE_CurrentState.dcf [...]

DllCamera File: D:\Program Files\ABB Industrial IT\Robotics IT\FlexLoader Vision\ [...]

1. Camera interface

Choose interface

- ○ None (file)
- ● DllCamera3D
- ○ Matrox GigE Scout/Ace
- ○ Basler Pylon GigE / USB

2. Camera settings

Miscellaneous

- □ Enable Color mode (only Scout/Ace)
- □ Warp images after calibration
- □ IgnoreGigE Camera IP Adress

Camera reverse: None

Resolution: 1626 x 1236 pixels

Advanced

- □ Log found position to file

3. Mechanical settings

Mechanical type

- ○ None / No PLC
- ● Multiflex
- ○ Miniflex

□ Has user defined data

PLC

IP-Address: 192.168.1.5

Brush encoder pulses/mm: 3

MultiFlexEditor

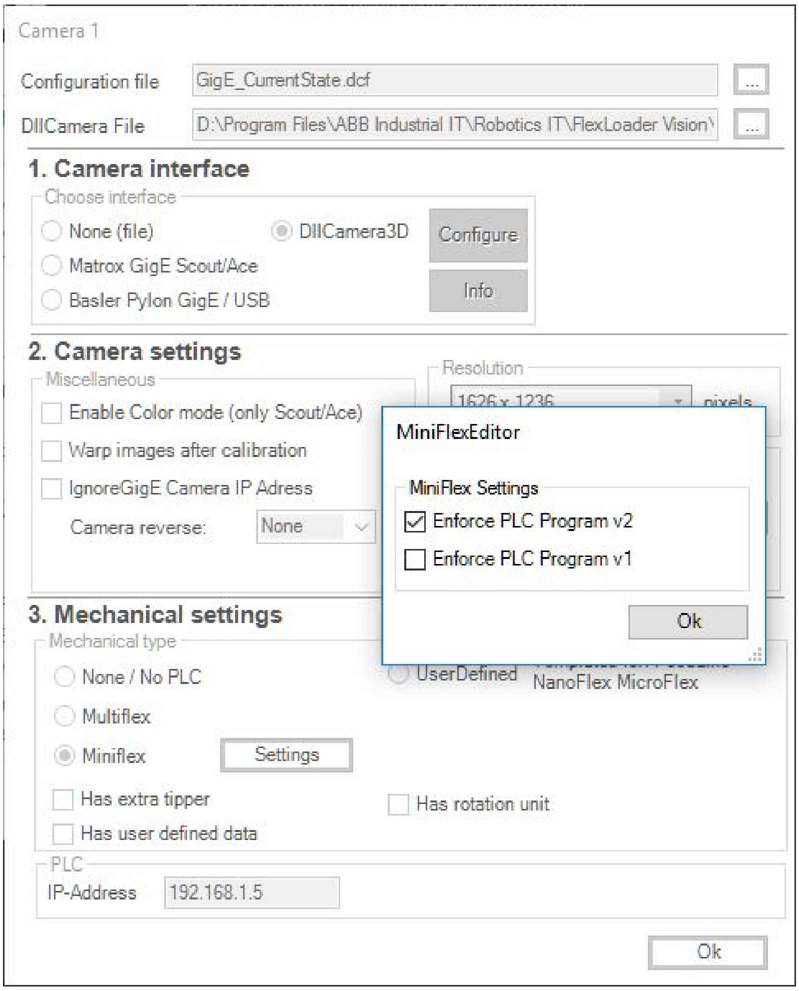
MultiFlex Settings

- ☑ Has brush
- □ Has sequence feeder
- ☑ Has pallet tipper
- □ BB1 Activate at-end-sensor

xx1800000440

Figure 12.3: Available settings for FP 400

Continues on next page



Camera 1

Configuration file: GigE_CurrentState.dcf ...

DllCamera File: D:\Program Files\ABB Industrial IT\Robotics IT\FlexLoader Vision\ ...

1. Camera interface

Choose interface

- ○ None (file)
- ● DllCamera3D
- ○ Matrox GigE Scout/Ace
- ○ Basler Pylon GigE / USB

2. Camera settings

Miscellaneous

- □ Enable Color mode (only Scout/Ace)
- □ Warp images after calibration
- □ IgnoreGigE Camera IP Adress

Camera reverse: None

Resolution: 1626 x 1236 pixels

MiniFlexEditor

MiniFlex Settings

- ☑ Enforce PLC Program v2
- □ Enforce PLC Program v1

3. Mechanical settings

Mechanical type

- ○ None / No PLC
- ○ Multiflex
- ● Miniflex
- ○ UserDefined

NanoFlex MicroFlex

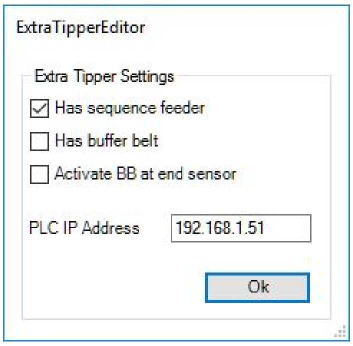
- □ Has extra tipper
- □ Has rotation unit
- □ Has user defined data

PLC

IP-Address: 192.168.1.5

xx1800000441

Figure 12.4: Available settings for FP 300



ExtraTipperEditor

Extra Tipper Settings

- ☑ Has sequence feeder
- □ Has buffer belt
- □ Activate BB at end sensor

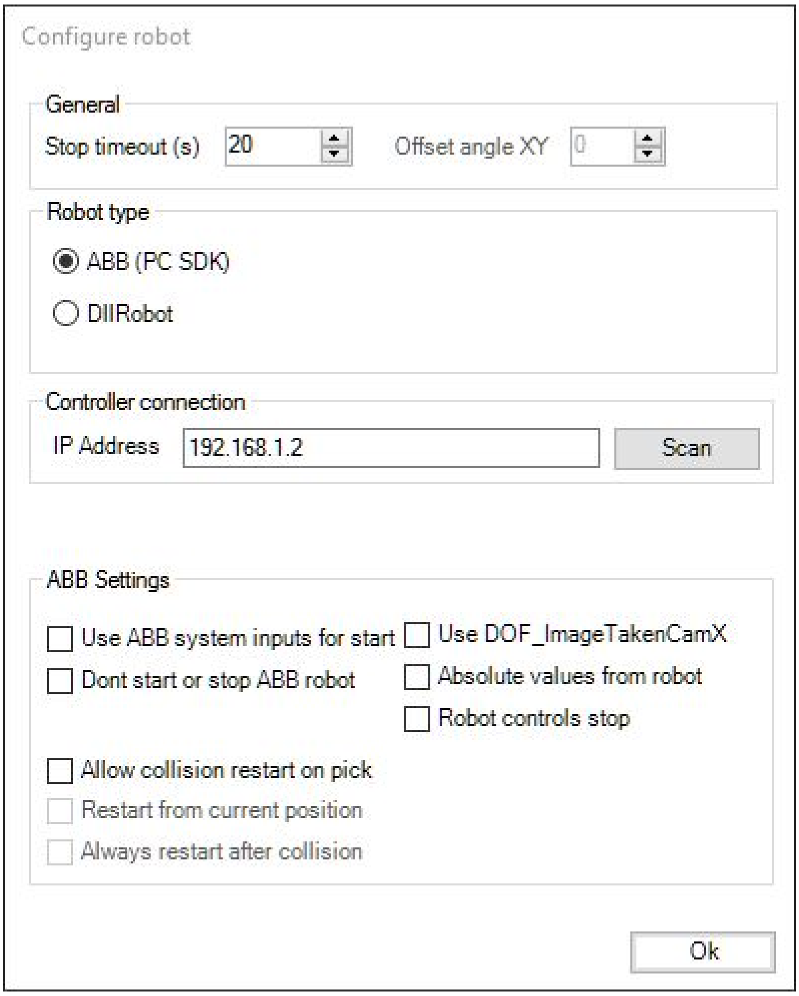
PLC IP Address: 192.168.1.51

xx1800000442

Figure 12.5: Available settings for extra tipper

12.1.4 Robot settings

The **Robot** menu is used to configure the robot properties and control behavior. To open the robot configuration settings, select **Robot** > **Configure**.


Configure robot

General

Stop timeout (s): 20

Offset angle XY: 0

Robot type

● ABB (PC SDK)

○ DllRobot

Controller connection

IP Address: 192.168.1.2

Scan

ABB Settings

□ Use ABB system inputs for start

□ Use DOF_ImageTakenCamX

□ Dont start or stop ABB robot

□ Absolute values from robot

□ Robot controls stop

□ Allow collision restart on pick

□ Restart from current position

□ Always restart after collision

Ok

xx1800000435

Robot configuration	Description	Default
Stop timeout [s]	When stopping the cell, the robot is allowed this amount of time to finish its task and stop. If the robot has not stopped after this time it will be forced to stop.	20
Offset angle (XY)	The grip angle of the located details (rotation in x-y plane) will be offset by this value.	0
Robot Type	The production systems work with the ABB robot. The **DllRobot** type is only used during system tests.	ABB (production system)
IP Address	Enter the robot IP address. Click **Scan** to detect an available robot system.	192.168.2.2
Use ABB system inputs for start	System signals are used for starting the robot. Can be useful in multitasking systems.	No

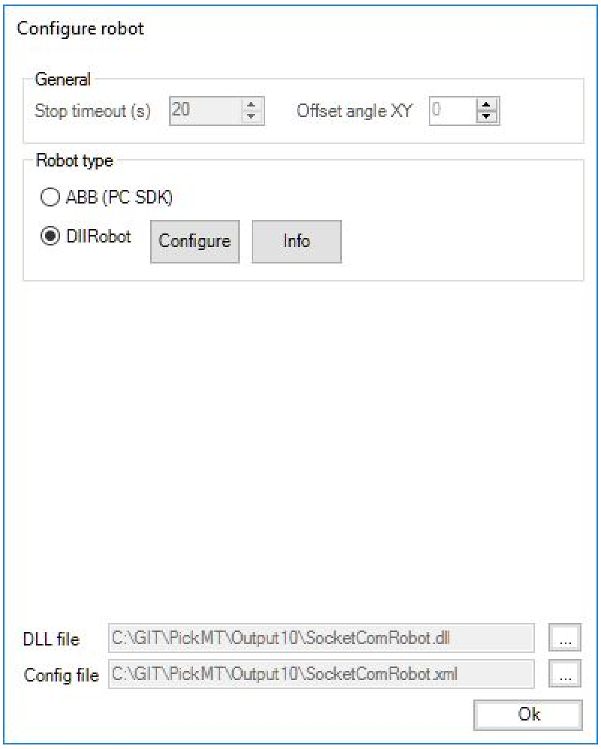
Continues on next page

Robot configuration	Description	Default
Don't start or stop ABB robot	Starting and stopping the robot is handled by another system component. Useful if an external master handles the overall task of selecting details and starting cell components.	No
Allow collision restart on pick	While picking a detail from the camera field of view the robot sometimes collides with another detail. In that case, it can attempt a restart (from main) in order to maintain the production process. A restart is only allowed if the robot is moving in one of the camera work objects, within acceptable coordinate ranges. A restart will not be attempted if the collision occurs within 3 minutes from start. A restart is only possible if the robot is able to clear its path (i.e. sufficiently soft collision).	No
Restart from current position	Instead of restarting from main, the robot restarts from its current position.	No
Always restart after collision	This setting allows restarting even within the 3 minutes limit mentioned above.	No
Use DOF_ImageTakenCamX	Message from FlexLoader Vision to the robot that the camera has finished the image acquisition process. Useful in applications with secondary camera checks and demands on short cycle times.	No
Absolute values from robot	Used when setting camera gain, exposure time, and contrast. If not checked, these values are sent as percentage (1–100). If checked, the values are sent as shown on the corresponding sliders on the user interface (1–255).	No
Robot controls stop	Allows the robot to control its stopping behavior while disregarding the specified timeout. After the stop command, FlexLoader Vision will wait indefinitely for the robot to stop, or enforces a stop when the operator clicks stop again. Useful in cells with extensive time demands on cycle stops.	No

When **DllRobot** is selected, the corresponding DLL file and XML configuration file must be specified. These files are provided by ABB. Typical settings are **SockekComRobot.xml** and **SockekComRobot.dll**. This configuration is only used for testing.

DllRobot	Description	Default
Config file	Configuration file used for DllCamera3D.	CirrusSensor.xml
DLL file	DLL file used for DllCamera3D.	CirrusSensor.dll

Continues on next page



Configure robot

General

Stop timeout (s) 20 Offset angle XY 0

Robot type

○ ABB (PC SDK)

● DllRobot

DLL file C:\GIT\PickMT\Output10\SocketComRobot.dll

Config file C:\GIT\PickMT\Output10\SocketComRobot.xml

xx1800000446

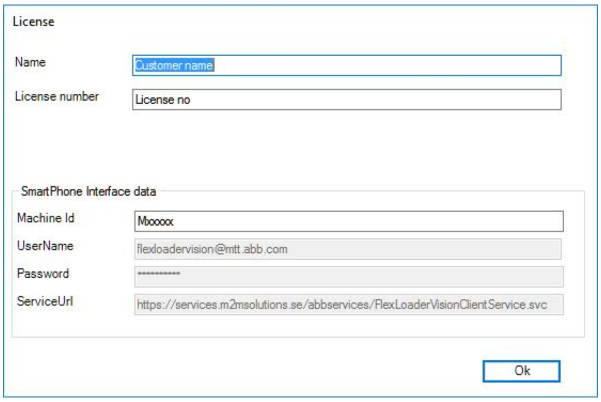
12.1.5 Tool settings

The **Tools** menu is used for further data configuration.

License data

License data	Description
Name	User name information
License number	License number information

SmartPhone interface data	Description
Machine ID	Machine ID used for app information display. Do not change if not instructed.
User name	User name for data reporting. Do not change if not instructed.
Password	Password for data reporting. Do not change if not instructed.
Service URL	URL for data reporting. Do not change if not instructed.



License

Name: Customer name

License number: License no

SmartPhone Interface data

Machine Id: Mxxxxx

UserName: flexloadervision@mtt.abb.com

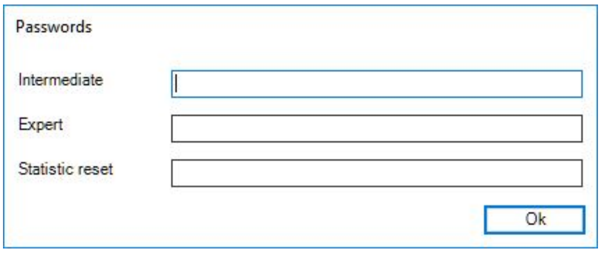
Password: ***********

ServiceUrl: https://services.m2msolutions.se/abbservices/FlexLoaderVisionClientService.svc

xx1800000437

Password settings

This dialog lets you specify the passwords for intermediate and expert user levels.



Passwords

Intermediate:

Expert:

Statistic reset:

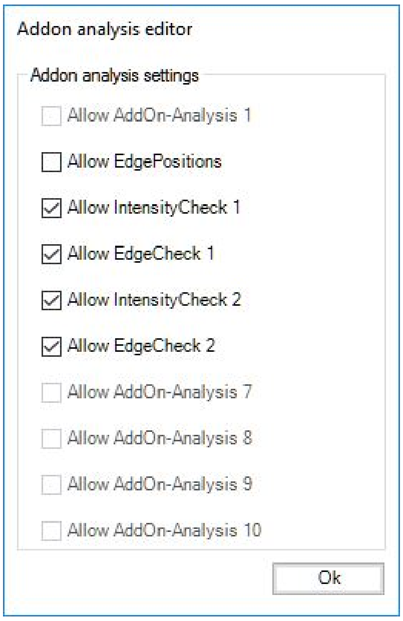
xx1800000438

Continues on next page

AddOn analysis

FlexLoader Vision provides additional methods of analysis. These methods can be activated depending on project needs.

For more information, see *AddOn analysis on page 87*.



Addon analysis editor

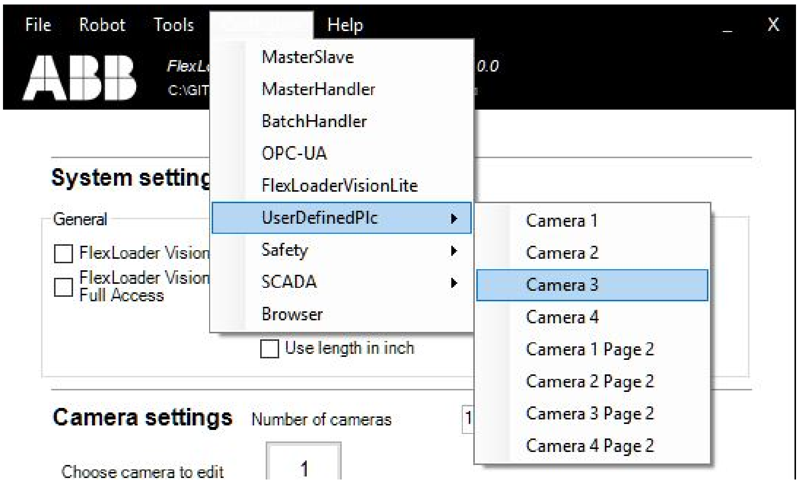
Addon analysis settings

- □ Allow AddOn-Analysis 1
- □ Allow EdgePositions
- ☑ Allow IntensityCheck 1
- ☑ Allow EdgeCheck 1
- ☑ Allow IntensityCheck 2
- ☑ Allow EdgeCheck 2
- □ Allow AddOn-Analysis 7
- □ Allow AddOn-Analysis 8
- □ Allow AddOn-Analysis 9
- □ Allow AddOn-Analysis 10

xx1800000436

12.1.6 Configure

The **Configure** menu is used for easy access to configuration files.



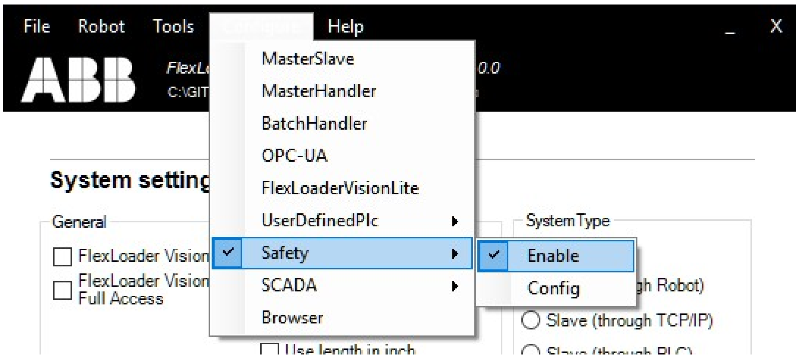
xx1800002956

Most configuration files can be easily accessed by simple selection, e.g. for configuration of user-defined mechanical equipment.

Select one of the files and configure according to commissioning instructions, see e.g. *User-defined mechanical equipment on page 186*.

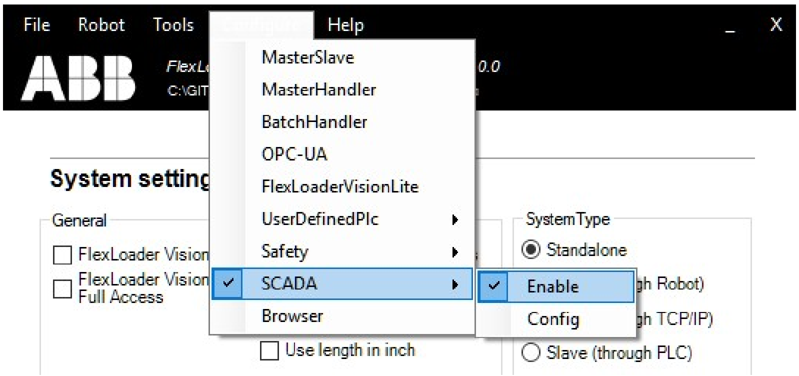
Functionality for safety, SCADA and browser information needs to be first enabled and then configured.

Enable the desired option and configure the file structure, see *SCADA and Safety on page 192*.



xx1800002957

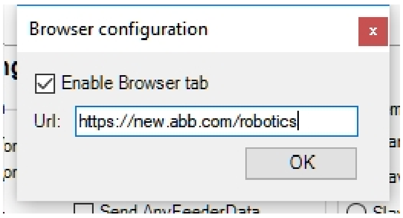
Continues on next page



☐ FlexLoader Vision
☐ FlexLoader Vision Full Access
✓ SCADA
✓ Enable
☐ Use length in inch
● Standalone
○ Slave (through PLC)

xx1800002958

Enable the **Browser** option if desired and configure the initial web page.


Browser configuration

☑ Enable Browser tab

Url: https://new.abb.com/robotics

xx1800002959

12.2 User accounts

General

Note

There are several default user accounts used in FlexLoader Vision. It is strongly recommended to change the default passwords to reduce the risk of unauthorized access to important system parameters.

Follow ABB’s advice for administration of user permissions.

Note

Please refer to the section on cyber security in the robot Integrator's guide for additional information.

FlexLoader Vision

The FlexLoader Vision system has two default user accounts:

For operation

User name	Password
FlexLoaderVision	FlexLoaderVision

For adminstration

User name	Password
FlexLoaderAdmin	FlexLoaderAdmin

Robot

Default credentials according to ABB default settings.

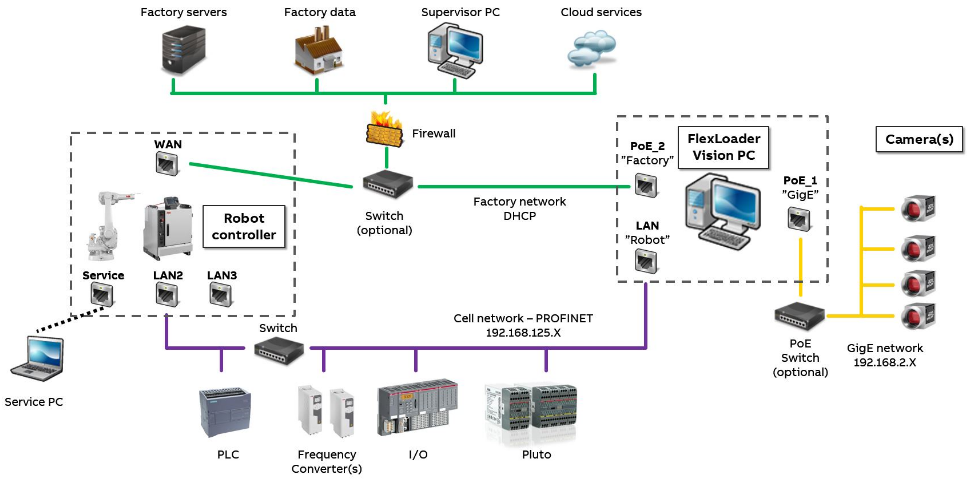
User name	Password
Default User	robotics

Follow ABB’s advice for administration of user permissions.

12.3 Network setup

Network overview

The standard network setup with its three main networks is configured as shown below.



xx1800002353

- The factory network connects to both the FlexLoader Vision PC and the robot. This network is optional.
- The camera network is reserved for FlexLoader Vision cameras.
- The internal cell network connects all internal components, mainly through PROFINET communication (using robot option 888-2).

PROFINET configuration

The FlexLoader function package has pre-configured PROFINET settings. If any changes are needed, at least some of the steps below must be performed.

Changes to PROFINET configurations shall only be done by trained technicians. Technicians shall have a good knowledge of I/O settings both in general and in RobotStudio.

	Action
1	Set your PROFINET network in the robot to **Private network** in RobotStudio. Use **Controller** -> **Configuration**. Modify IP settings to fit the pricate network.
2	Configure devices for your robot controller in RobotStudio I/O configurator (IOC). Use **Controller** -> **Configuration**. Import all necessary GSDML files or modify the IPPNIO.xml file. Locate devices with the RobotStudio ScanTool and add relevant devices to the controller. Modify names and addresses if necessary, see *Standard network IP settings on page 177*. All sub-modules are needed when adding PLUTO devices.

Continues on next page

	Action
3	If relevant, configure Safe I/O. As general recommendation, two channel signals can be defined as separate single channel signals that are combined by PreLogic in SafeMove. Ensure correct safety checksum by using **Controller** -> **Configuration** -> **Vendor Tool**. The generated checksum is then used in the SafeMove configuration.
4	Set device names on devices with RobotStudio ScanTool. Modify the devices names if necessary, see *Standard network IP settings on page 177*. Other tools can be used if desired.

The PROFINET controllers in FlexLoader function packages normally uses DCP (Discovery Configuration Protocol), i.e. that they will configure devices when they found them.

The key property for this method is the device name (same as station name). The controller is told which devices to handle and communicate with and which device names they have. With this information, the controller finds the actual devices with the right names on the network and automatically assigns IP addresses that are pre-configured in the controller.

ABB CI502 I/O nodes will get its name depending on the switches/knobs on the module, e.g. **ci502-pn-01** if knobs are put in positions 0 (x10H) and 1 (x01H). A cold start is needed to read new name. Do not use knob positions 0 and 0, as these are used for user-defined names.

Robot PROFINET devices such as I/O nodes and Pluto gateway will get their IP address as warm start as defined in robot I/O configurator.

Internal cell network

The internal cell network connects the FlexLoader Vision PC, robot, safety devices, I/O devices, frequency converters and other internal cell components to each other.

The PC network connection is named **Robot** on physical port **LAN**.

This network is the robots private network, and both robot **LAN2** and **LAN3** as well as the service port are connected to this network. The robot is acting as PROFINET master.

Tip

The FlexLoader Vision PC can only be connected to **LAN2**.

A separate network connection to the robot network can be made by using **LAN3** network isolation.

In this case, the service port remains internally connected to **LAN2**.

See robot documentation for further information.

Note

Do not connect the internal network directly to the factory network.

Continues on next page

Limitations

Currently, the PROFINET master functionality of the robot does not allow to configure a connection to a Siemens IDevice, i.e. direct connection to another Siemens PLC acting as PROFINET master is not possible.

The robot PROFINET functionality can be used on only one of the following connections: **WAN**, private network (service port, **LAN2**, **LAN3**), or isolated **LAN3** network.

The private network of two robot controllers cannot be directly connected.

For connecting two robot controllers or an external master PLC to the cell, see *External PROFINET connections on page 178*.

Note

Robot option 841-1 Ethernet/IP™ cannot be combined with the FlexLoader function packages and standard cells.

If Ethernet/IP™ communication is needed, the robot option 840-1 Ethernet/IP™ fieldbus adapter must be used instead.

Robot service connection

A robot service PC can be connected to the robot controller through the service port of the robot controller. For more information, see the robot product manual.

Factory network

The FlexLoader Vision PC can be connected to a factory network. The PC network connection is named **Factory** on physical port **PoE2**.

Please refer to the robot controller product manual for setting the robot IP configuration on the **WAN** port.

This network is controlled by the customer, who is responsible for appropriate network setup and network protection mechanisms.

Note

Please refer to the section on cyber security in the robot integrator's guide for additional information.

The use of the FlexLoader Vision App requires an active internet connection to the FlexLoader Vision PC.

Camera network

The FlexLoader Vision PC is connected to the cameras on a separate internal network.

The PC network connection is named **GigE** on physical port **PoE1**.

Continues on next page

A single camera is connected to a PoE port of the FlexLoader Vision PC. Several cameras are typically connected by means of a PoE switch.

Tip

If the factory network is not used, the **PoE2** port can be used to connect and power a second camera.

Standard network IP settings

Internal machine network

Equipment	IP address	Gateway	Device name
PC	192.168.125.150	192.168.125.1	-/-
Robot	192.168.125.1	192.168.125.1	irc5-pnio
PLC camera 1	192.168.125.151	192.168.125.1	e.g. plc-fp100
PLC camera 2	192.168.125.152	192.168.125.1	e.g. plc-fp300
PLC camera 3	192.168.125.153	192.168.125.1	e.g. plc-fp400
PLC camera 4	192.168.125.154	192.168.125.1	e.g. plc-fp600
Pluto gateway	192.168.125.160	192.168.125.1	gatepn
I/O board 1	192.168.125.161	192.168.125.1	ci502-pn-01
...	...	...	
I/O board 9	192.168.125.169	192.168.125.1	e.g. ci502-pn-09
Frequency converter 1	192.168.125.171	192.168.125.171	e.g. bufferbelt2, separationbelt, camerabelt, pallettipper
...	...	...	...
Frequency converter 9	192.168.125.179	192.168.125.179	
Project specific equipment 1	192.168.125.180	192.168.125.1	
...	...	...	
Project specific equipment 20	192.168.125.199	192.168.125.1	

Internal camera network

Equipment	IP address	Gateway
PC	192.168.2.1	Do not specify gateway.
Camera 1	192.168.2.2	Do not specify gateway.
Camera 2	192.168.2.3	Do not specify gateway.
Camera 3	192.168.2.4	Do not specify gateway.
Camera 4	192.168.2.5	Do not specify gateway.

Continues on next page

Factory network

DHCP is used as standard setting.

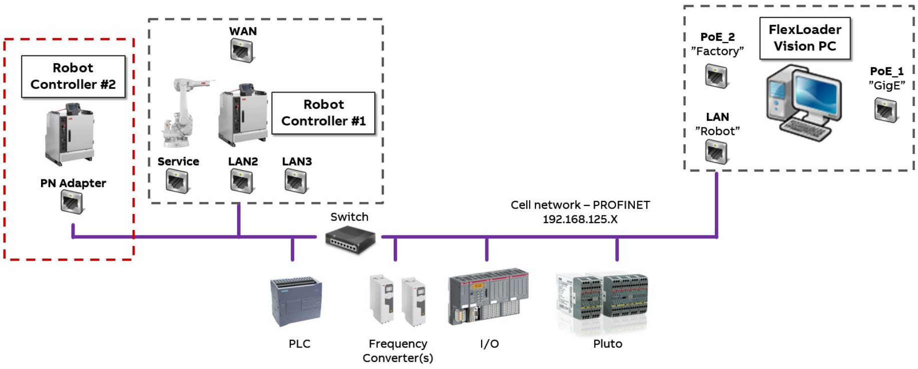
Note

The network configuration on the factory network can be freely adopted to the customers network.

External PROFINET connections

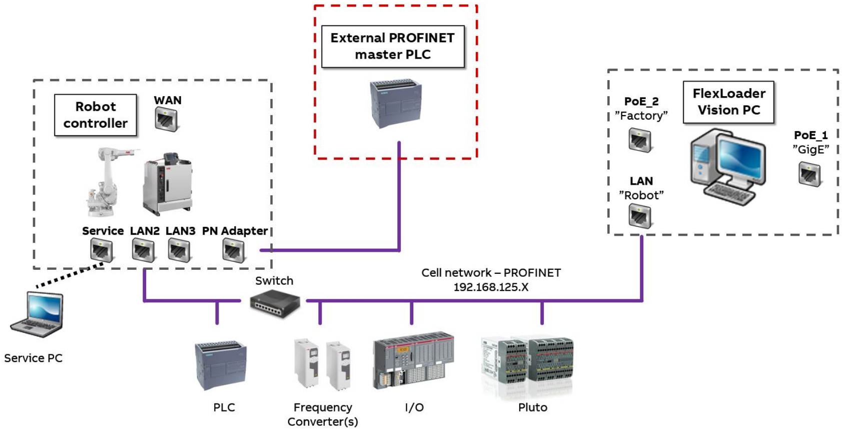
In some cases external PROFINET connections are needed during integration of a FlexLoader function package or standard cell.

A second robot controller can be connected through PROFINET by using a PROFINET fieldbus adapter in the second controller (robot option 840-3).



xx1800002906

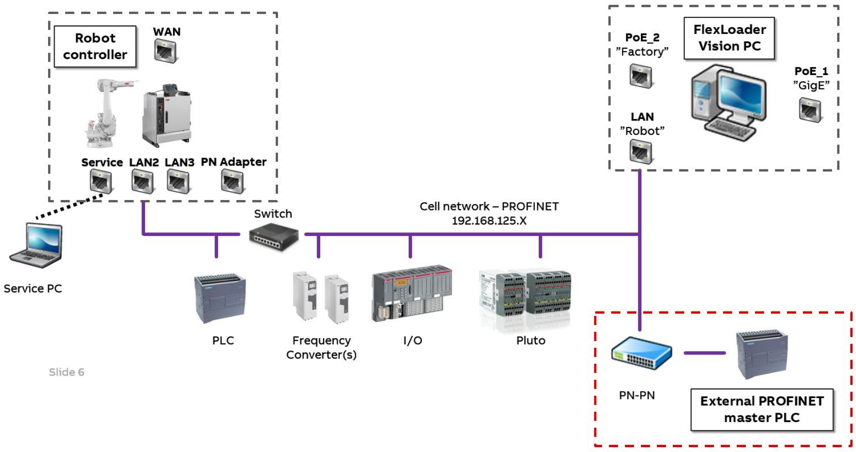
A master PLC be connected through PROFINET by using a PROFINET fieldbus adapter in the first controller (robot option 840-3).



xx1800002905

Continues on next page

Alternatively, the master PLC can be connected by using a PNPN-coupler.



xx1800002907

Note

The connections can in principle be routed through the WAN port as well, with all internal network components connected to **WAN** instead of **LAN2**.

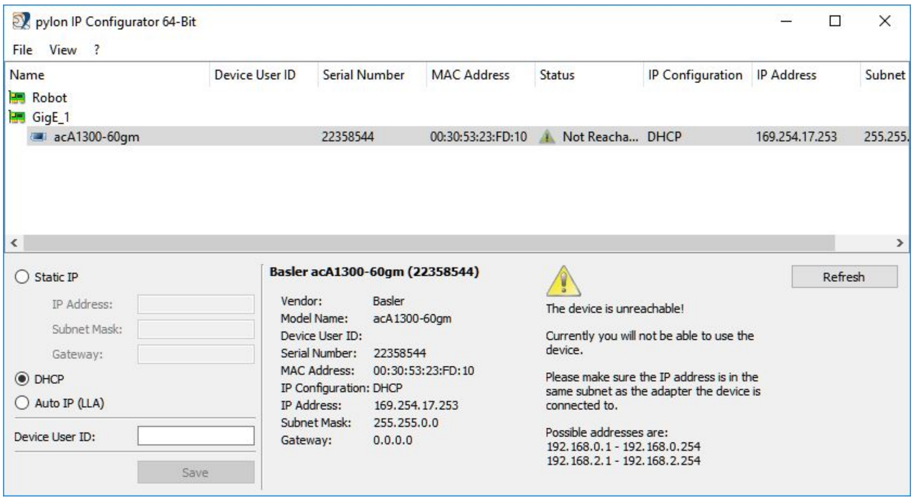
However, this will expose all components to an external network. We recommend to avoid this system setup.

Settings for Basler GigE cameras

Network setting for cameras

The following instructions assume that ABB's standard IP address settings are used. The same steps must be followed if alternative IP addresses are to be used. The only difference is in the addresses that must be entered.

1 Connect the camera to the computer. Start **Pylon IP Configuration Tool** and check that a connected camera can be seen. Select the camera.



Name	Device User ID	Serial Number	MAC Address	Status	IP Configuration	IP Address	Subnet
Robot							
GigE_1							
acA1300-60gm		22358544	00:30:53:23:FD:10	Not Reacha...	DHCP	169.254.17.253	255.255.

○ Static IP
IP Address:
Subnet Mask:
Gateway:
● DHCP
○ Auto IP (LLA)
Device User ID:

Basler acA1300-60gm (22358544)

Vendor:	Basler
Model Name:	acA1300-60gm
Device User ID:	
Serial Number:	22358544
MAC Address:	00:30:53:23:FD:10
IP Configuration:	DHCP
IP Address:	169.254.17.253
Subnet Mask:	255.255.0.0
Gateway:	0.0.0.0

The device is unreachable!

Currently you will not be able to use the device.

Please make sure the IP address is in the same subnet as the adapter the device is connected to.

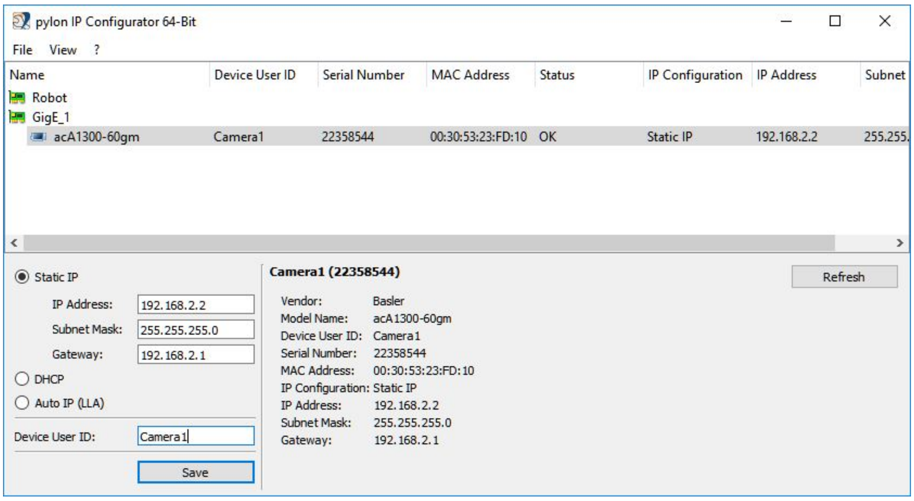
Possible addresses are:
192.168.0.1 - 192.168.0.254
192.168.2.1 - 192.168.2.254

xx1800000966

Continues on next page

2 Select the radio buttons in the **IP Configuration Protocol** pane according to the image below. Specify the correct **Device User ID** (**Camera1**, **Camera2**, **Camera3**, **Camera4** for the respective camera) and the correct data in the **Static IP address** pane (**Camera1 192.168.2.2**, **Camera2 192.168.2.3**, **Camera3 192.168.2.4**, **Camera4 192.168.2.5**, **Subnet Mask 255.255.255.0**, **GateWay 192.168.2.1**) as below.

3 Click **Save** and wait for the camera to be detected again.



Name	Device User ID	Serial Number	MAC Address	Status	IP Configuration	IP Address	Subnet
Robot							
GigE_1							
acA1300-60gm	Camera1	22358544	00:30:53:23:FD:10	OK	Static IP	192.168.2.2	255.255.

● Static IP

IP Address: 192.168.2.2

Subnet Mask: 255.255.255.0

Gateway: 192.168.2.1

○ DHCP

○ Auto IP (LLA)

Device User ID: Camera1

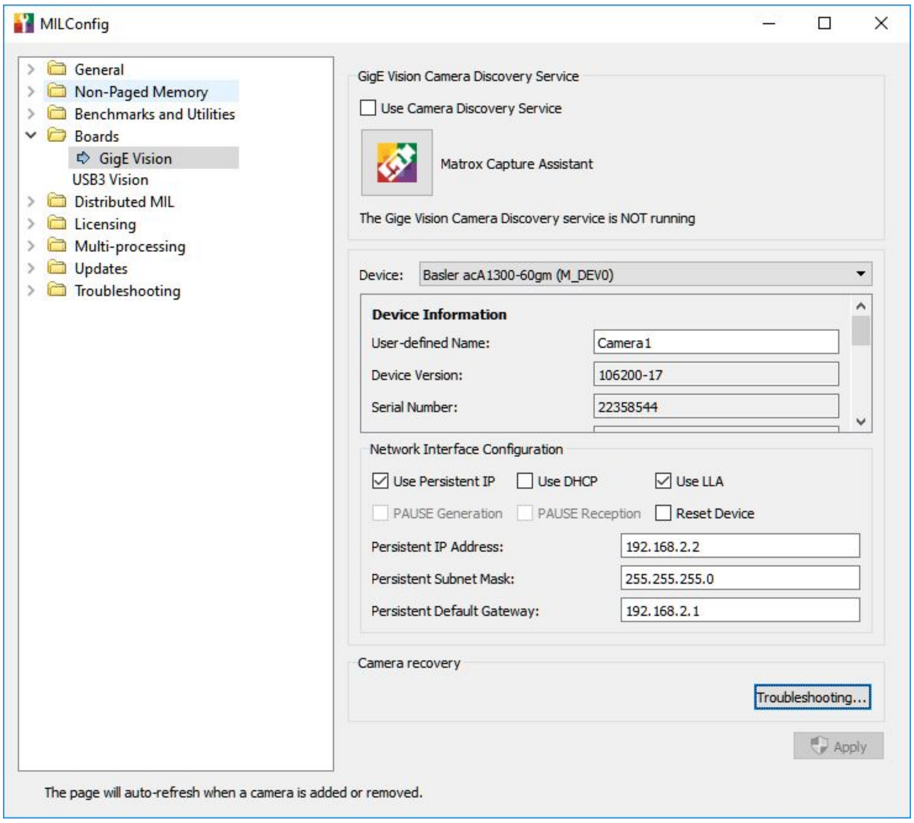
Camera1 (22358544)

Vendor:	Basler
Model Name:	acA1300-60gm
Device User ID:	Camera1
Serial Number:	22358544
MAC Address:	00:30:53:23:FD:10
IP Configuration:	Static IP
IP Address:	192.168.2.2
Subnet Mask:	255.255.255.0
Gateway:	192.168.2.1

xx1800000968

Continues on next page

Start the program **MilConfg**. Select **Boards** > **GigE Vision**. Deselect **Use Camera Discovery Service** and click **Apply**.



□ Use Camera Discovery Service

The Gige Vision Camera Discovery service is NOT running

Device: Basler acA1300-60gm (M_DEV0)

User-defined Name: Camera1

Device Version: 106200-17

Serial Number: 22358544

☑ Use Persistent IP □ Use DHCP ☑ Use LLA

□ PAUSE Generation □ PAUSE Reception □ Reset Device

Persistent IP Address: 192.168.2.2

Persistent Subnet Mask: 255.255.255.0

Persistent Default Gateway: 192.168.2.1

The page will auto-refresh when a camera is added or removed.

xx1800000967

General information about camera settings

The camera settings should first be made via the settings options in **Pylon Viewer**, but can also be programmed in Matrox's own software **IntelliCam.**

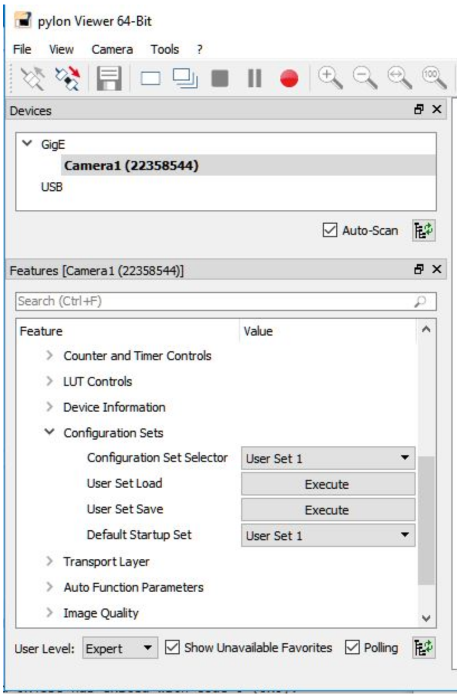
Note that the settings can differ slightly for different camera models and in newer versions of the software. Also note that there are several advanced settings options for the image field size, image preparation etc. These options should only be used by specially trained personnel.

Start **Pylon Viewer** and select the correct camera. Specify **Expert** or **Guru** as **User Level** and open **Configuration Sets** in the parameter tree. Set **ConfigurationSetSelector** to **User Set 1**. The changes are then made in this configuration setup so that the default values of the camera are saved.

Specify **User Set 1** as **Default Startup Set.**

Continues on next page

After making the desired changes to the configuration, the changes must be saved to the camera memory. This keeps the settings safe in the event of a power cut. To save to the camera memory, select **User Set Save** and click **Execute**.



Feature	Value
Configuration Sets	
Configuration Set Selector	User Set 1
User Set Load	Execute
User Set Save	Execute
Default Startup Set	User Set 1

User Level: Expert ☑ Show Unavailable Favorites ☑ Polling

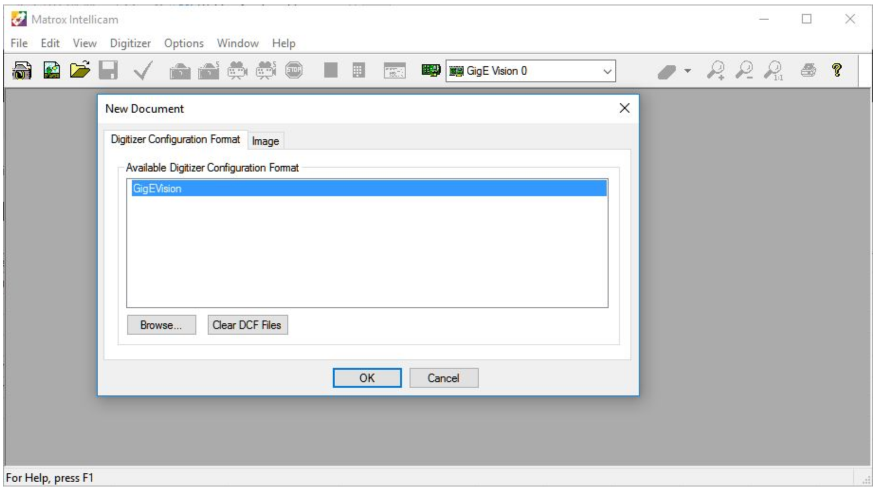
xx1800000303

Troubleshooting

If the camera is not detected by FlexLoader Vision, this may be due to several circumstances. First check the configuration of the IP settings for both the computer and the camera.

If the camera shows images in Baslers Pylon Viewer, but FlexLoader Vision still does not detect the camera, do as follows:

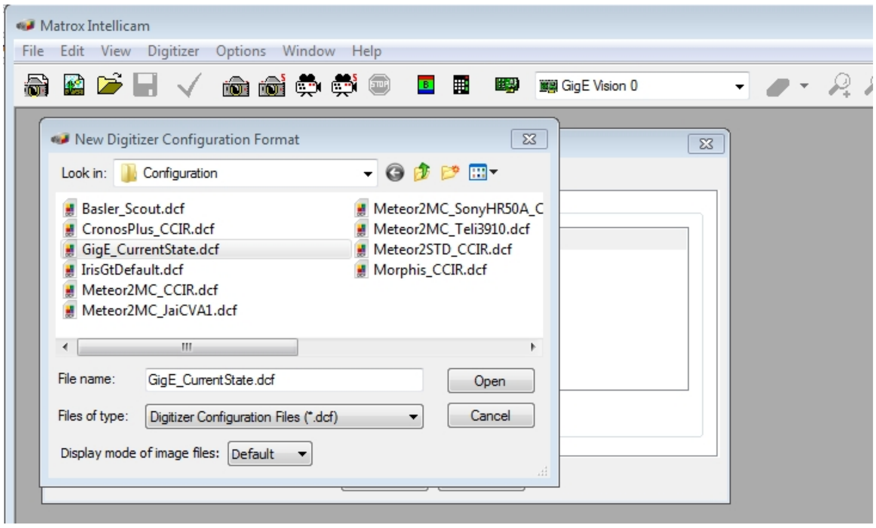
Start **Matrox Intellicam** (**Start** > **Matrox Imaging** > **Tools** > **IntelliCam**). Click on **File** > **New**, select **GigEVision** and click **OK**.



xx1800000980

Continues on next page

Click **File** > **Save as ...** and save the camera configuration file as follows in FlexLoader Vision’s configuration folder.



Look in: Configuration

Basler_Scout.dcf
CronosPlus_CCIR.dcf
GigE_CurrentState.dcf
IrisGtDefault.dcf
Meteor2MC_CCIR.dcf
Meteor2MC_JaiCVA1.dcf
Meteor2MC_SonyHR50A_C
Meteor2MC_Teli3910.dcf
Meteor2STD_CCIR.dcf
Morphis_CCIR.dcf

File name: GigE_CurrentState.dcf

Files of type: Digitizer Configuration Files (*.dcf)

Display mode of image files: Default

xx1800000981

Close **IntelliCam** and restart FlexLoader Vision. Check that the camera is detected correctly and shows images in FlexLoader Vision.

Settings for 3D cameras

See separate instructions for information about 3D cameras.

12.4 Robot system

Operating system and options

FlexLoader Vision is prepared for use with RobotWare 6.10.02 or above.

This RobotWare option is required for correct communication with RobotWebServices.

The following controller options are needed when using FlexLoader Vision with the ABB robot controller.

Robot option	Description	Note
685-2	RW 6	Robotware version
608-1	World Zones	Used for optional surveillance of robot motion
616-1	PC Interface	Mandatory for base communication
623-1	Multitasking	Mandatory for base communication
1554-1	Prepared for FlexLoader Vision	Base license for FlexLoader Vision

System program and configuration files

Note

Do a backup of the robot before you carry out the installation.

Continues on next page

There is a structure for the robot program files to manage the modularity of the different machine types in FlexLoader. Each file sub package could contain RAPID files such as .mod or .sys files.

It could also contain language files to handle localization of user messages and configuration files to define I/O:s and make RAPID modules load correctly.

Exactly which files are included in a sub package depends on what function that package should cover. The complete code package for a machine might include a lot of small files, especially regarding the configuration files. This is a result of the modularity, to have small functions put into their own section.

When a robot is prepared, a code package built for the application is placed in the robot HOME folder. If the package includes more than just the basic vision functionality, then an add-in (see appendix) containing a set of base functionality should be installed into the robot system before loading the program files.

Then all configuration files from the package are loaded at once by marking all of them and select to load and replace duplicates.

Restart the robot with a P-start after the configuration files are loaded. This will correctly load all programs.

Always adjust the I/O address mapping to match the actual project.

The programs on the robot can be structured into several functionality areas:

- FlexLoader application functionality - main program files (for example MainModule, PartCam1, Common) handling the actual movements, picking parts, cell flow.
- FlexLoader Vision interface - Handling all vision communication and vision related functionality. Most important modules Vision.sys and VisionCom.mod.
- FlexLoader Vision Lite functionality - Handling parametrization and machine handling code for FlexLoader Vision Lite.
- FlexLoader FlexLoader Library Add-in - Support functionality implemented as an RobotWare Add-in. Core parts of the add-in are Alarms&Messages, Part trackers, Stations, MoveVia and Logging.
- FlexLoader assistance and utility functionality - Handles for example tool handling, entry control, light indications, in- and out-feeders and options.
- FlexLoader machine tool interface functionality - Handles everything regarding machine communication. Intended to be replaced to fit current machine.

For detailed information see *FlexLoader RAPID reference on page 305*.

MultiMove systems

In MultiMove systems additional files can be used:

- Home\MainRob2.pgf, Home\MainRob3.pgf, Home\MainRob4.pgf

These are optional files. If present, they will be automatically loaded upon start of a detail from FlexLoader Vision.

Operation with previous RobotWare versions

FlexLoader Vision checks the correct RobotWare version during startup.

Continues on next page

This check can be disabled by creating the file **skiprwscheck.txt** in the **FlexLoaderVision\Bin** folder.

⚠ **WARNING**

Disabling this check may result in communication losses if used to communicate with older RobotWare versions.

Coordinate system for the robot gripper

When setting the gripping positions during teachin, FlexLoader Vision always draws a blue line independent of the areas that visualize the gripping fingers themselves. The blue line shows in which direction the positive x direction should be for the robot gripper. The z direction is always down or into the image.

When defining grippers, it is usually necessary to choose how x should point in relation to how the fingers are positioned. When defining the picking position, it is possible to rotate the entire gripper to match the detail correctly.

Depending on how the gripper and the cell look physically, it may be important to make sure these rotations are exactly correct.

In order to provide more setting options, it is also possible to make adjustments for the x direction of the robot in the underlying configuration. This allows rotation z in teachin to be 0 as a standard which seems more natural to the user.

Normally, there is no additional rotation imposed from the underlying configuration. In cases where standard double grippers are used, the mechanical design requires this direction to be rotated 180 degrees. In these cases, the underlying configuration is adjusted to +180 degrees (see parameter OffsetAngleXY, *Configuration editor on page 157*).

Further information

Further information on calibration, RAPID code and communication are given in the section on robot integration, see *Robot integration on page 221*.

12.5 User-defined mechanical equipment

Overview

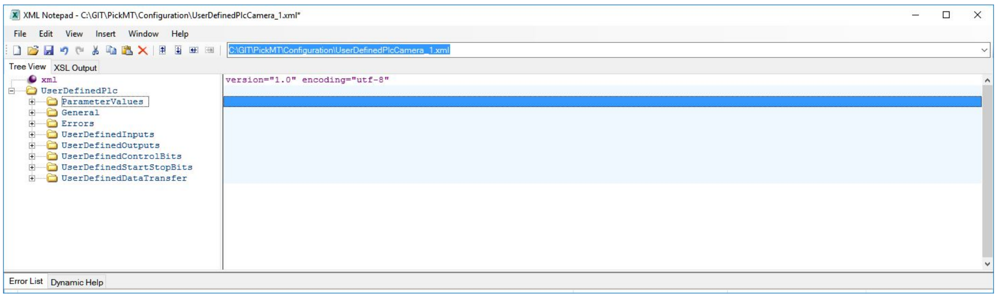
The concept of user-defined equipment provides a variety of settings for a complete integration of another piece of equipment with FlexLoader Vision. Communication can occur between FlexLoader Vision PC and an S7-PLC 1200 series. All data is written to datablock 1 (if not specified otherwise).

All parameters are configured in an XML file, **UserDefinedPlcCamera_1.xml**. The following examples are taken from configuration via the software **XML NotePad**, but any XML editor can be used.

> **Tip**
>
> As an option the user-defined parameters can be sent to an ABB robot. In this case, at least one of the 32 parameters must be defined as type 9 or 11 (see below).

Note that it is possible to mix robot and PLC parameters for the same user-defined equipment. However, there must be at least one PLC type parameter and at least one robot type parameter among the 32 basic parameters.

The following parameter areas can be configured:


XML Notepad - C:\GIT\PickMT\Configuration\UserDefinedPlcCamera_1.xml*
File
Edit
View
Insert
Window
Help
C:\GIT\PickMT\Configuration\UserDefinedPlcCamera_1.xml
Tree View
XSL Output
Error List
Dynamic Help

```
xml                          version="1.0" encoding="utf-8"
UserDefinedPlc
    ParameterValues
    General
    Errors
    UserDefinedInputs
    UserDefinedOutputs
    UserDefinedControlBits
    UserDefinedStartStopBits
    UserDefinedDataTransfer
```

xx1800000983

- General data, designation in FlexLoader Vision interface among other things.
- Up to 32 parameters that can be sent to PLC at each start-up.
- Up to 64 error messages that are connected to PLC.
- Up to 16 user-defined inputs.
- Up to 16 user-defined outputs.
- Up to 8 memory cells that can be used for manual control of the equipment.
- A start and stop bit for synchronized starts and stops between FlexLoader Vision and PLC.
- A free number of hidden parameters. None of these will be displayed in FlexLoader Vision. They are sent to the PLC in the same way as the usual parameters. This is useful for more general parameters which may need to be set from time to time.

If more than 32 parameter values are needed, additional data that is displayed on a second page on the user interface. Data for this page is defined in the XML file

Continues on next page

UserDefinedPlcCamera_1_Tab2.xml. Note that only the sections "General" and "user-defined parameters" (see below) are used in this case.

User-defined data

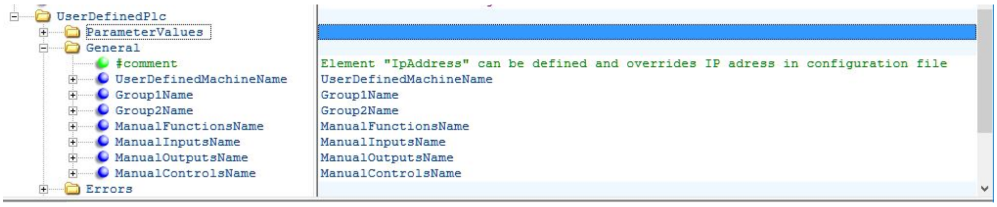
Normally, the mechanical setup is defined as a single feeder for each camera, e.g. as user-defined mechanical equipment.

In some cases, the mechanical setup is defined by a main feeder, e.g. a FlexLoader FP 600, and it is desirable to define additional data.

Configuration can then be done to include User-defined data, see *Mechanical settings on page 163*. The user defined data is configured in the same way as for a user-defined mechanical equipment. However, only the user-defined parameters and the reset bits will be applicable.

General

Here principal text is configured that is shown on FlexLoader Vision's user interface.



```
UserDefinedPlc
  ParameterValues
  General
    #comment                     Element "IpAddress" can be defined and overrides IP adress in configuration file
    UserDefinedMachineName       UserDefinedMachineName
    Group1Name                   Group1Name
    Group2Name                   Group2Name
    ManualFunctionsName          ManualFunctionsName
    ManualInputsName             ManualInputsName
    ManualOutputsName            ManualOutputsName
    ManualControlsName           ManualControlsName
  Errors
```

xx1800000984

On the **Mechanics** tabs (on the **Teachin** and the **Settings** menus), during teachin and servicing, the image **UserDefinedPlcImage.bmp**, is shown, which should be 416×416 pixels in size.

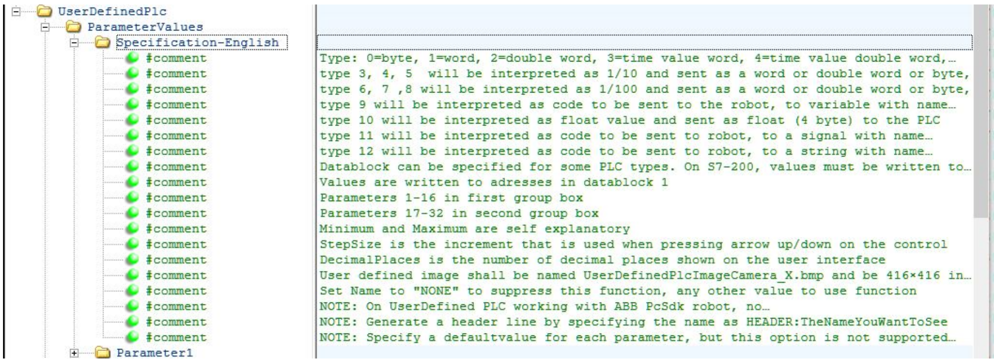
User-defined parameters

Each parameter that should be managed via FlexLoader Vision can be configured fully according to the user requirements.

Parameter	Description
Name	The text that is shown on the **Mechanics** tab for this parameter. Must be set to **NONE** if the parameter is not used. If the text starts with **HEADER:** the parameter is interpreted as a header. Text after **HEADER:** is in bold and the input field for parameter data is hidden in FlexLoader Vision. This parameter contains data for different languages. New languages can be added by adding new language tags.
PLCDataBlock	Data block in the PLC memory.
PLCAddress	Memory address in the PLC memory.

Continues on next page

Parameter	Description
Type	Specifies the size of the variable and how the sent value is to be interpreted: • 0 = integer as byte • 1 = integer as word • 2 = integer as double word • 3 = 1/10 value as word • 4 = 1/10 value as double word • 5 = 1/10 value as byte • 6 = 1/100 value as word • 7 = 1/100 value as double word • 8 = 1/100 value as byte • 9 = numerical value sent to robot variable • 10 = float value as float (4 bytes in PLC) • 11 = 0/1 value sent to robot signal • 12 = string sent to robot variable
Minimum	Minimum value that the user is permitted to set in FlexLoader Vision.
Maximum	Maximum value that the user is permitted to set in FlexLoader Vision
StepSize	Size of the value change that occurs when the user presses the arrow up/down.
DecimalPlaces	Number of decimals shown in FlexLoader Vision.
RobotID	Indicates the name of a robot variable that the value is written to.
DefaultValue	Start value for the parameter.



```
Type: 0=byte, 1=word, 2=double word, 3=time value word, 4=time value double word,…
type 3, 4, 5  will be interpreted as 1/10 and sent as a word or double word or byte,
type 6, 7 ,8 will be interpreted as 1/100 and sent as a word or double word or byte,
type 9 will be interpreted as code to be sent to the robot, to variable with name…
type 10 will be interpreted as float value and sent as float (4 byte) to the PLC
type 11 will be interpreted as code to be sent to robot, to a signal with name…
type 12 will be interpreted as code to be sent to robot, to a string with name…
Datablock can be specified for some PLC types. On S7-200, values must be written to…
Values are written to adresses in datablock 1
Parameters 1-16 in first group box
Parameters 17-32 in second group box
Minimum and Maximum are self explanatory
StepSize is the increment that is used when pressing arrow up/down on the control
DecimalPlaces is the number of decimal places shown on the user interface
User defined image shall be named UserDefinedPlcImageCamera_X.bmp and be 416×416 in…
Set Name to "NONE" to suppress this function, any other value to use function
NOTE: On UserDefined PLC working with ABB PcSdk robot, no…
NOTE: Generate a header line by specifying the name as HEADER:TheNameYouWantToSee
NOTE: Specify a defaultvalue for each parameter, but this option is not supported…
```

xx1800000986

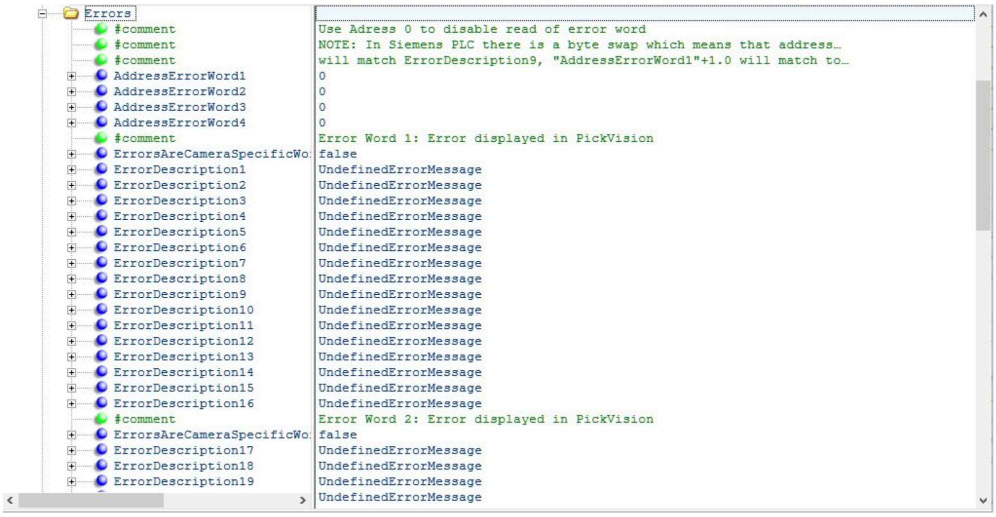
Error management

Errors are read as single bits from up to four word memories in PLC. It is possible to select whether the alarm for each word memory is to be camera specific or not. If they are camera specific, the text **camera X** appears in the alarm text. Also note that in a Siemens PLC, a byte swap occurs, which means that the start address for the word in the PLC corresponds to **ErrorDescription 9**. Other bytes in the PLC word memory start at **ErrorDescription 1**.

Parameter	Description
AddressErrorWord1	Enter memory location for error messages 1–16. Enter 0 if this address is not to be used.
AddressErrorWord2,3,4	As above, but for error messages 17–32, 33–48, 49–64.

Continues on next page

Parameter	Description
ErrorDescriptionX	Text that is shown if corresponding bit in corresponding **AddressErrorWord** is 1. This parameter contains data for different languages. New languages can be added by adding new language tags.



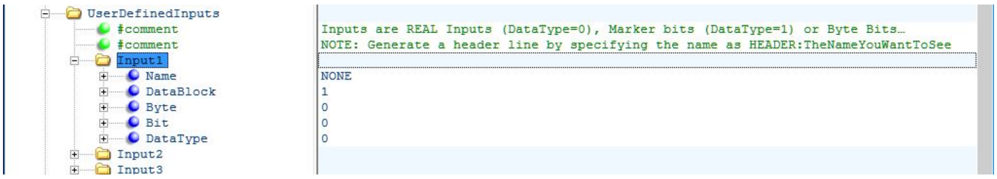
```
Errors
    #comment                      Use Adress 0 to disable read of error word
    #comment                      NOTE: In Siemens PLC there is a byte swap which means that address…
    #comment                      will match ErrorDescription9, "AddressErrorWord1"+1.0 will match to…
    AddressErrorWord1             0
    AddressErrorWord2             0
    AddressErrorWord3             0
    AddressErrorWord4             0
    #comment                      Error Word 1: Error displayed in PickVision
    ErrorsAreCameraSpecificWo:    false
    ErrorDescription1             UndefinedErrorMessage
    ErrorDescription2             UndefinedErrorMessage
    ErrorDescription3             UndefinedErrorMessage
    ErrorDescription4             UndefinedErrorMessage
    ErrorDescription5             UndefinedErrorMessage
    ErrorDescription6             UndefinedErrorMessage
    ErrorDescription7             UndefinedErrorMessage
    ErrorDescription8             UndefinedErrorMessage
    ErrorDescription9             UndefinedErrorMessage
    ErrorDescription10            UndefinedErrorMessage
    ErrorDescription11            UndefinedErrorMessage
    ErrorDescription12            UndefinedErrorMessage
    ErrorDescription13            UndefinedErrorMessage
    ErrorDescription14            UndefinedErrorMessage
    ErrorDescription15            UndefinedErrorMessage
    ErrorDescription16            UndefinedErrorMessage
    #comment                      Error Word 2: Error displayed in PickVision
    ErrorsAreCameraSpecificWo:    false
    ErrorDescription17            UndefinedErrorMessage
    ErrorDescription18            UndefinedErrorMessage
    ErrorDescription19            UndefinedErrorMessage
                                  UndefinedErrorMessage
```

xx1800000985

User-defined inputs

PLC inputs that are to be shown for the user are defined here. There are up to 16 user-defined inputs, which can be freely defined. These are used to show the status of the equipment in FlexLoader Vision. Inputs can also be defined as memory bits or datablock memories (datablock 1) in the PLC.

Parameter	Description
Name	Text that appears on FlexLoader Vision’s user interface. Enter **NONE** if the input is not used. This parameter contains data for different languages. New languages can be added by adding new language tags.
DataBlock	If **DataType** is set to 2 (ByteBits) the number of the datablock is specified here.
Byte	Byte for the physical input.
Bit	Bit for the physical input.
DataType	Determines whether FlexLoader Vision is to write actual PLC inputs (0), memory addresses in PLC (1) or datablock memories in PLC (2).



```
UserDefinedInputs
    #comment          Inputs are REAL Inputs (DataType=0), Marker bits (DataType=1) or Byte Bits…
    #comment          NOTE: Generate a header line by specifying the name as HEADER:TheNameYouWantToSee
    Input1
        Name          NONE
        DataBlock     1
        Byte          0
        Bit           0
        DataType      0
    Input2
    Input3
```

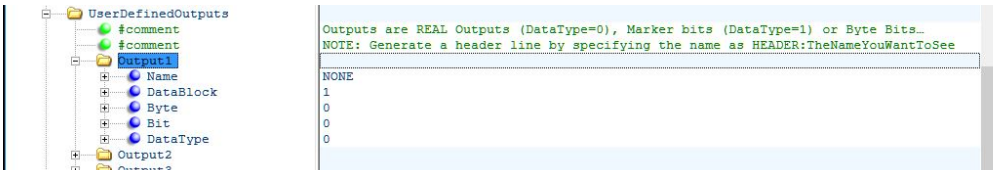
xx1800000987

Continues on next page

User-defined outputs

PLC outputs that are to be shown for the user are defined here. There are up to 16 user-defined outputs, which can be freely defined. These are used to show the status of the equipment in FlexLoader Vision.

Parameter	Description
Name	Text that appears on FlexLoader Vision's user interface. Enter **NONE** if the output is not used. This parameter contains data for different languages. New languages can be added by adding new language tags.
DataBlock	If **DataType** is set to 2 (**ByteBits**) the number of the datablock is specified here.
Byte	Byte for the physical input.
Bit	Bit for the physical input.
DataType	Determines whether FlexLoader Vision is to write actual PLC outputs (0), memory addresses in PLC (1) or datablock memories in PLC (2).



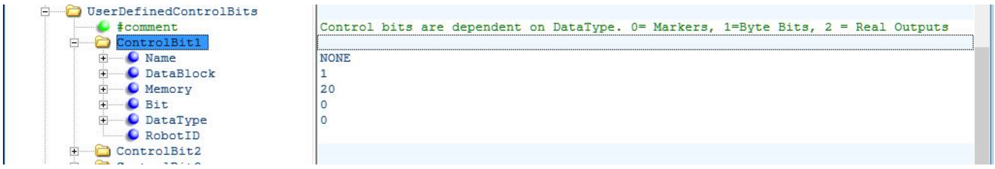
```
UserDefinedOutputs
    #comment          Outputs are REAL Outputs (DataType=0), Marker bits (DataType=1) or Byte Bits…
    #comment          NOTE: Generate a header line by specifying the name as HEADER:TheNameYouWantToSee
    Output1
        Name          NONE
        DataBlock     1
        Byte          0
        Bit           0
        DataType      0
    Output2
```

xx1800000988

General control bits

PLC cursors that are to be shown for the user are defined here. There are up to 8 user-defined cursors, which can be freely defined. These are used to allow manual control of the equipment from FlexLoader Vision.

Parameter	Description
Name	Text that appears on FlexLoader Vision's user interface. Enter **NONE** if the function is not used. This parameter contains data for different languages. New languages can be added by adding new language tags.
DataBlock	If **DataType** is set to 1 (**ByteBits**) the number of the datablock is specified here.
Memory	Address for the physical input.
Bit	Bit for the physical input.
DataType	Determines whether FlexLoader Vision is to write memory addresses in PLC (0), datablock memories in PLC (1) or actual PLC outputs (2).
RobotID	Indicates the name of a robot variable that the value is written to.



```
UserDefinedControlBits
    #comment          Control bits are dependent on DataType. 0= Markers, 1=Byte Bits, 2 = Real Outputs
    ControlBit1
        Name          NONE
        DataBlock     1
        Memory        20
        Bit           0
        DataType      0
        RobotID
    ControlBit2
```

xx1800000989

Continues on next page

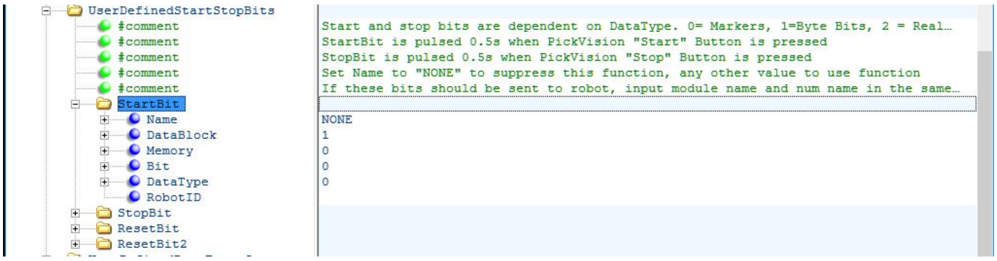
Start, stop and reset bit

PLC markers that are used at start and stop in FlexLoader Vision are defined here. These are used to synchronize the equipment with FlexLoader Vision.

The PLC marker is set to 1 for 0.5 s at start, stop or reset

Note that you can use dual reset bits. Both **ResetBit** and **ResetBit2** are available. This may be useful if reset needs to be sent to both the robot and the PLC.

Parameter	Description
Name	Text that appears on FlexLoader Vision's user interface. Enter **NONE** if the function is not used. This parameter contains data for different languages. New languages can be added by adding new language tags.
DataBlock	If **DataType** is set to 1 (**ByteBits**) the number of the datablock is specified here.
Memory	Address for the physical input.
Bit	Bit for the physical input.
DataType	Determines whether FlexLoader Vision is to write memory addresses in PLC (0), datablock memories in PLC (1) or actual PLC outputs (2).
RobotID	Indicates the name of a robot variable that the value is written to.



```
UserDefinedStartStopBits
    #comment        Start and stop bits are dependent on DataType. 0= Markers, 1=Byte Bits, 2 = Real…
    #comment        StartBit is pulsed 0.5s when PickVision "Start" Button is pressed
    #comment        StopBit is pulsed 0.5s when PickVision "Stop" Button is pressed
    #comment        Set Name to "NONE" to suppress this function, any other value to use function
    #comment        If these bits should be sent to robot, input module name and num name in the same…
    StartBit
        Name        NONE
        DataBlock   1
        Memory      0
        Bit         0
        DataType    0
        RobotID
    StopBit
    ResetBit
    ResetBit2
```

xx1800000982

12.6 SCADA and Safety

12.6.1 Introduction

Overview

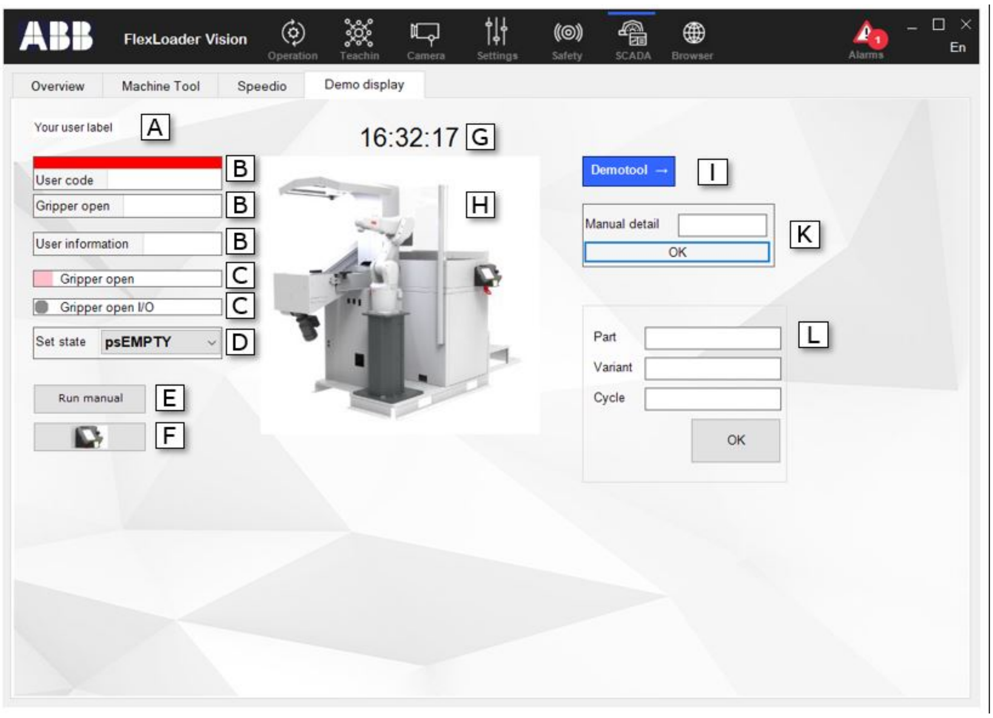
Various types of information from the robot controller and from OPC-UA servers can be visualized in different ways and support the operator of the robot cell. Sum alarms for the **Safety** and **SCADA** tab can be defined in order to call the operators attention to problems.

The information that is visible on these pages is configured by means of several XML configuration files. Language localization is fully supported by all configuration files. It is the integrators task to configure these information pages.

Configuration for both **Safety** and **SCADA** tab is identical and is therefore described together in this section.

Examples

The following image shows examples for various graphical elements.



Set state psEMPTY

xx1800002971

	Description		Description
A	Label	G	Clock
B	Textbox	H	Image
C	Signal	I	Navigation
D	ComboBox	K	InputBox
E	Button	L	Form

Continues on next page

	Description		Description
F	ImageButton		

Note

FlexLoader Vision can run on other PC:s while configuring the **SCADA** and **Safety** tab. Some warnings may appear.

The **SCADA** and **Safety** tab can be tested either with a virtual controller on the same PC (specify 127.0.0.1 as robot IP address) or with a real controller.

For testing on another PC, you must copy the \Bin folder and the \Configuration folder to the other PC.

Graphical element appearance

General

A variety of graphical elements is available for configuration and display, with different use cases.

Element	Description
Button ImageButton	Defines a button that can send a predefined value to a specific data variable. ImageButton is like a Button, with the ability to display an image instead of text.
Clock	Defines a clock with local time that is updated every second.
ComboBox	Defines a ComboBox for easy selection of predefined values.
Form	Collection of values which are transferred to data variables when clicking a single button.
Image	Defines an image display at desired location.
InputBox	Defines a small window with a button and an input field that can be used to send user information to a specific data variable.
Label BorderLabel	Defines a text label for general information. BorderLabel is like a Label, but with a thin border drawn around the text.
Navigation	Defines a blue navigation button that can be used to link to other information pages.
Signal	Defines a data-bound green/red indication, based on the status of a signal or variable.
Textbox	Defines a read-only data-bound display with title, with additional green/red indication for quick diagnosis.

Visibility conditions

Visibility is an important feature to enhance operator attention to presented data. Visibility can be controlled in two ways:

- Conditions for the value of the displayed data can control visibility (e.g. display of error information only when the error occurs).
- Conditions for other data, e.g. using common control data for highlighting groups of data values.

Continues on next page

Number of elements

The SCADA and safety pages can handle many data points for typical cell sizes.

These data points are continuously polled from the robot controller. Up to ~100 data points can be handled with normal polling times.

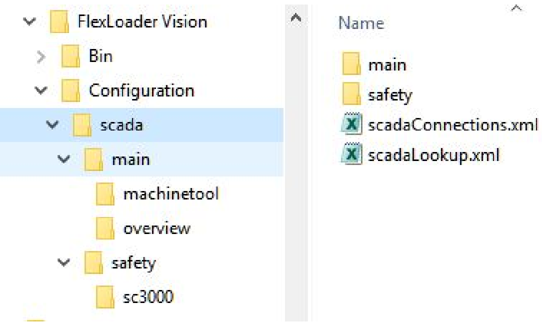
The system can handle much larger numbers of data points, at the cost of slower updates and higher communication load with the robot.

12.6.2 Configuration

Folder structure

The FlexLoader Vision **Configuration** folder contains the **Scada** folder. Information for the **Safety** tab is contained in the **Safety** folder. Information for the **SCADA** tab is contained in the **Main** folder.

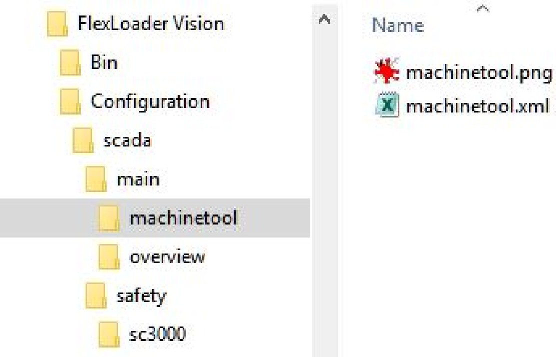
The **Scada** folder also contains general information for data lookup (**scadaLookup.xml**) and optional OPC-UA server connections (**scadaConnections.xml**).



xx1800002968

Each of the information pages is represented by a subfolder, in the example below the **SCADA** tab contains two information pages (**Overview** and MachineTool), whereas the **Safety** tab contains a single information page (**SC3000**).

Each information page folder contains the page definition XML file (with same name as folder name) and all necessary supporting image files.

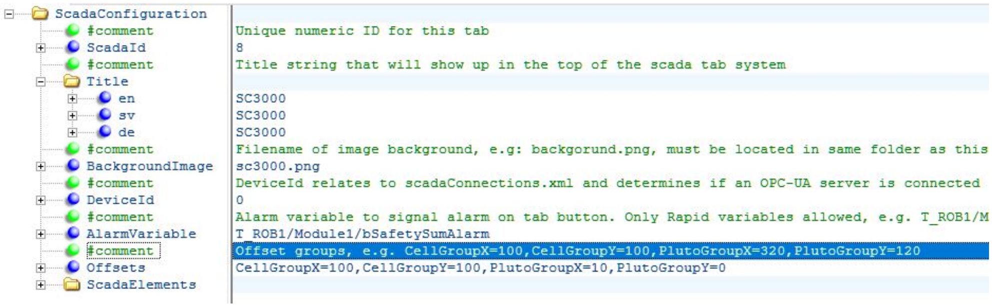


xx1800002969

Continues on next page

Information page

The base data for each information page is described in the first part of the page definition XML file.



```
ScadaConfiguration
  #comment              Unique numeric ID for this tab
  ScadaId               8
  #comment              Title string that will show up in the top of the scada tab system
  Title
    en                  SC3000
    sv                  SC3000
    de                  SC3000
  #comment              Filename of image background, e.g: backgorund.png, must be located in same folder as this
  BackgroundImage       sc3000.png
  #comment              DeviceId relates to scadaConnections.xml and determines if an OPC-UA server is connected
  DeviceId              0
  #comment              Alarm variable to signal alarm on tab button. Only Rapid variables allowed, e.g. T_ROB1/M
  AlarmVariable         T_ROB1/Module1/bSafetySumAlarm
  #comment              Offset groups, e.g. CellGroupX=100,CellGroupY=100,PlutoGroupX=320,PlutoGroupY=120
  Offsets               CellGroupX=100,CellGroupY=100,PlutoGroupX=10,PlutoGroupY=0
  ScadaElements
```

xx1800002970

XML element	Description
ScadaId	Unique numeric ID for this tab , e.g. 9. Do not use same ID in **Safety** and **SCADA** tab. The ScadaId also defines the information page order.
Title	Title string that will show up in the top of the information page, e.g. Machine Tool. The Title can be localized.
BackgroundImage	Filename of background image, e.g. sc3000.png. It must be located in same folder as the XML file. Images can be of jpg or png type.
AlarmVariable	Alarm variable to signal alarm on tab button. Only RAPID variables allowed, e.g. T_ROB1/Module1/bSafetySumAlarm. Allowed variables are of type bool and num (values <> 0 will be interpreted as alarm) An alarm from at least one of the information pages is sufficient to raise a visible alarm on the corresponding tab icon.
Offsets	Offset groups, e.g. CellGroupX=100,CellGroupY=100,PlutoGroupX=320,PlutoGroupY=120 Used to position and move several graphical elements together as a group.
ScadaElements	Collection of graphical elements, see *Graphical element configuration on page 198*.

Syntax rules

Data variables

The data variable element can be configured for both RAPID data and OPC-UA server data. The following rules apply:

Data source	Format	Example
RAPID variable	{task}/{module}/{symbol}	T_ROB1/MainModule/nVariable1

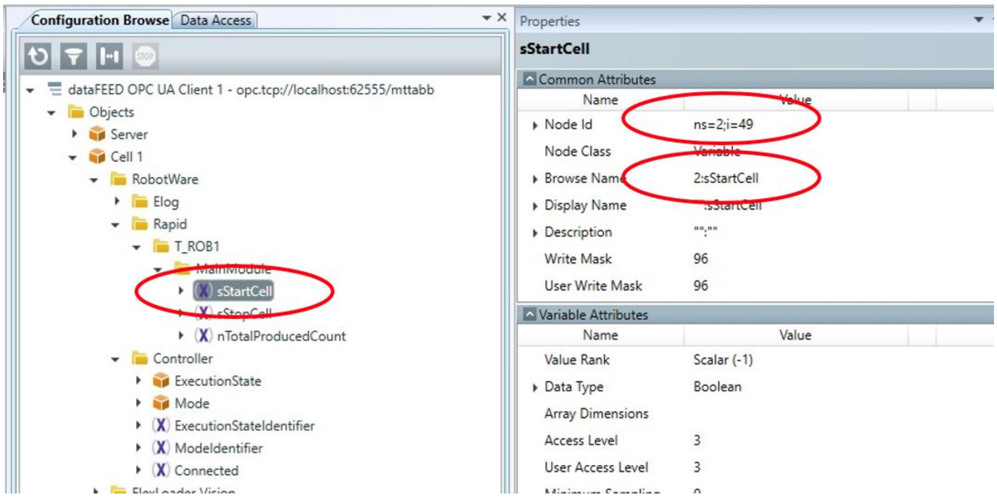
RAPID RECORD variable	{task}/{module}/{symbol}	T_ROB1/MainModule/rSomeRecordElement{1}
		T_ROB1/MainModule/rAnotherElement{1,4,3}
		RECORD variables are handled as strings. Indexing starts at element 1.
		If the indexed variable is a RECORD, it will be shown in standard notation, e.g. [0,0,0].

Continues on next page

Data source	Format	Example
RAPID signal	{signalname} Signal used instead of a variable IO|{signalname}	Signal1MachineA IO|Signal1MachineA as DataType bool in ElementType Signal

OPC-UA server variable
(see example below)
	ns={namespaceindex};s={string identifier}
	or
	orns={namespaceindex};i={identifier}
		ns=2;s=sStartCell (recommended)
		or
		ns=2;i=49 (identifier can change when variables are added)

Example for OPC-UA server variable definition:



sStartCell

Common Attributes

Name	Value
Node Id	ns=2;i=49
Node Class	Variable
Browse Name	2:sStartCell
Display Name	"":sStartCell
Description	"":""
Write Mask	96
User Write Mask	96

Variable Attributes

Name	Value
Value Rank	Scalar (-1)
Data Type	Boolean
Array Dimensions	
Access Level	3
User Access Level	3
Minimum Sampling	0

xx1800002992

Conditions

Conditions can be used for color indication (GreenCondition) and visibility control (VisibleCondition). Conditions can be specified in several ways. String data types are not supported in conditions.

Condition	Format	Example
Equal to X	value==X	value==1 value==false
Not equal to X	value!=X	value!=0 value!=true
Less than or equal to X	value(=X	value(=12
Less than X	value(X	value(5
Greater than or equal to X	value)=X	value)=4
Greater than X	value)X	value)100
Beetween X and Y	X(value(Y	2(value(5

When using a additional ConditionVariable, its value can be used in the following manner.

Condition	Format	Example
single use	conditionvalue==X	conditionvalue==1

Continues on next page

Condition	Format	Example
AND condition	value==X AND conditionvalue==X	value==10 AND condition-value==true
OR condition	not supported	not supported

Only single AND connection as shown above are supported.

Localized text

Text that is displayed can be localized to different languages. By default, the english version must be supplied and will be used if the target language is not available. Other languages can be added by the integrator.

XML example:

```
<Label>
  <en>Batch count</en>
  <sv>Batch antal</sv>
  <de>Batch Anzahl</de>
</Label>

// Language codes as defined in Language.xml
```

Graphical element configuration

General

All graphical elements have common properties that are used in the same way for all element types.

Element	Description	Not applicable
X	Specifies the horizontal pixel position of the graphical element in relation to the display window (if no offset is used) or in relation to the offset position.	-/-
Y	Specifies the vertical pixel position of the graphical element in relation to the display window (if no offset is used) or in relation to the offset position.	-/-
Element-Type	Set to one of TextBox, BorderLabel, Label, Signal, Image, Navigation, Clock, InputBox, Button, ImageButton, ComboBox, Form.	-/-
OffsetX	Use one of the offsets defined in the information page base data or set to 0. Can be omitted.	-/-
OffsetY	Use one of the offsets defined in the information page base data or set to 0. Can be omitted.	-/-
SizeX	Horizontal size of graphical element. Set to suitable value.	Navigation
Visible-Condition	Optional. Specifies condition for element visibility, e.g. value)70. If used, it requires the definition of DataType and DataVariable. For syntax see *Conditions on page 197*.	Clock

Continues on next page

Element	Description	Not applicable
DataType	Set to one of int, bool, string, signal. When used as Button, only int and bool are allowed. When used as ComboBox / Form, only string is allowed.	Clock
DataVariable	Specifies data variable source. For syntax see *Data variables on page 196*.	Clock
Condition-Variable	Optional. Specifies data variable source for an additional condition value. Variable must be bool or num For syntax see *Conditions on page 197*.	Clock, ComboBox
Label	Specifies the text shown. English text must always be specified. The text can be localized, see *Localized text on page 198*.	Image, Clock

XML examples are shown below.

Other properties

Some of the graphical elements have other properties that are used specifically for these elements.

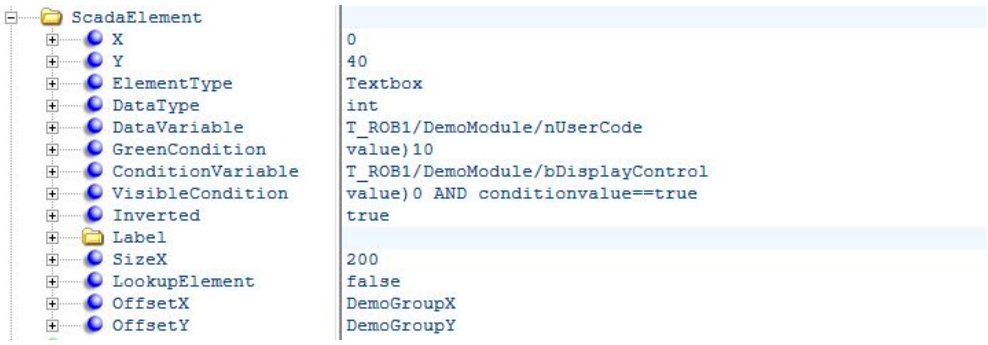
Element	Description	Applicable
LookupElement	Determines if lookup table shall be used for the values. Set to true or false. For definition of lookup tables see *Lookup tables on page 204*.	
GreenCondition	Defines the condition when the bar above the Textbox is shown in green. For syntax see *Conditions on page 197*. If omitted, no bar is shown at all.	TextBox
Inverted	Inverts green and red color display for TextBox and Signal.	TextBox, Signal
Source	For Image: File name of display image, e.g. arrow.png. It must be located in same folder as the XML file. Images can be of jpg or png type. For Button: Pre-defined value that is written to data variable. For ImageButton: Pre-defined value that is written to data variable. Additional image file name. For Navigation: The ScadaID of the information page that the navigation refers to. For ComboBox: Values or variables to be displayed in the selection drop down box. Values are separated by a semicolon.	Image, Button / Image-Button, Navigation, ComboBox

XML examples are shown below.

Continues on next page

Textbox example

Simple example for Textbox display.



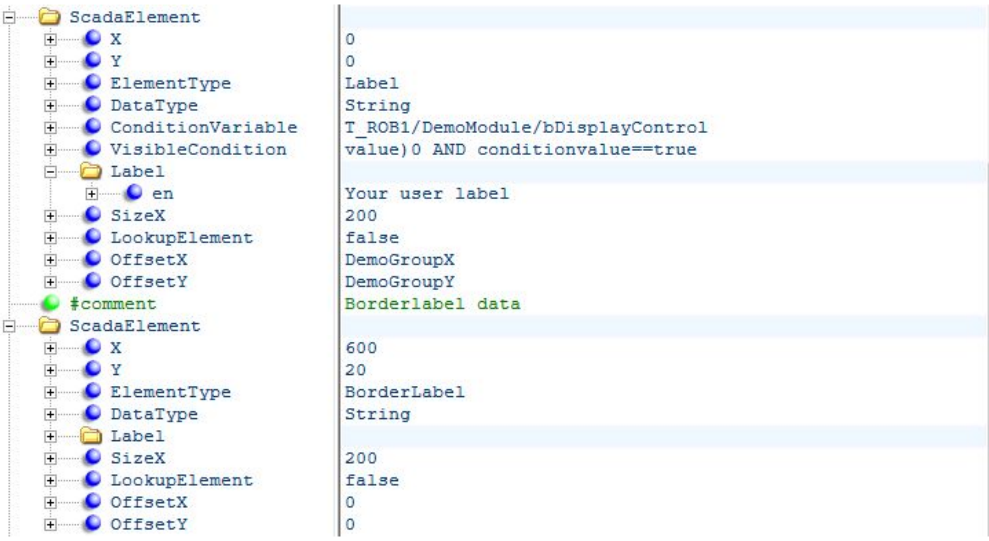
```
ScadaElement
    X                     0
    Y                     40
    ElementType           Textbox
    DataType              int
    DataVariable          T_ROB1/DemoModule/nUserCode
    GreenCondition        value)10
    ConditionVariable     T_ROB1/DemoModule/bDisplayControl
    VisibleCondition      value)0 AND conditionvalue==true
    Inverted              true
    Label
    SizeX                 200
    LookupElement         false
    OffsetX               DemoGroupX
    OffsetY               DemoGroupY
```

xx1800002979

Label / BorderLabel example

Simple example for Label / BorderLabel display.

BorderLabel has the same functionality and behavior as Label, with the difference that a border is drawn around a BorderLabel.



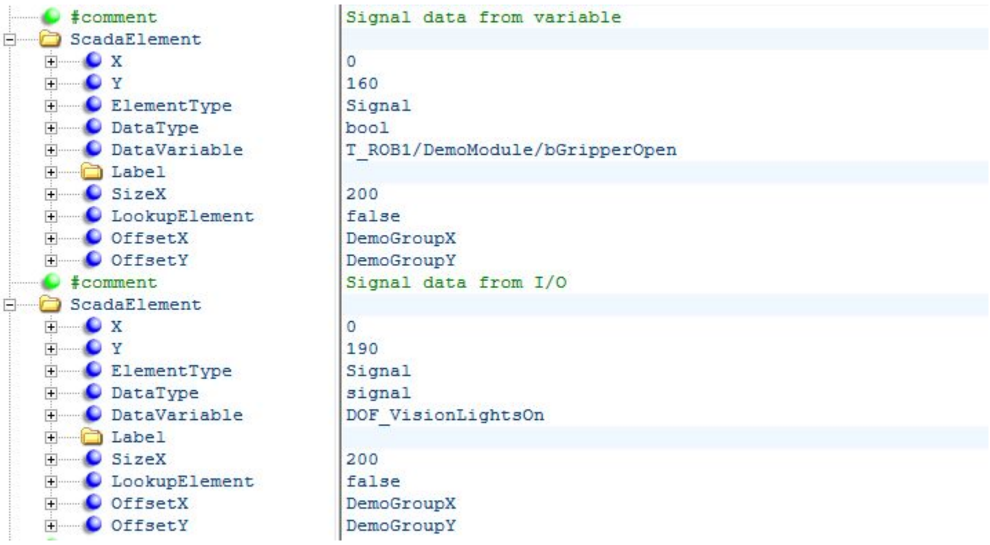
```
ScadaElement
    X                     0
    Y                     0
    ElementType           Label
    DataType              String
    ConditionVariable     T_ROB1/DemoModule/bDisplayControl
    VisibleCondition      value)0 AND conditionvalue==true
    Label
        en                Your user label
    SizeX                 200
    LookupElement         false
    OffsetX               DemoGroupX
    OffsetY               DemoGroupY
#comment                  Borderlabel data
ScadaElement
    X                     600
    Y                     20
    ElementType           BorderLabel
    DataType              String
    Label
    SizeX                 200
    LookupElement         false
    OffsetX               0
    OffsetY               0
```

xx1800002980

Continues on next page

Signal example

Simple example for Signal display.

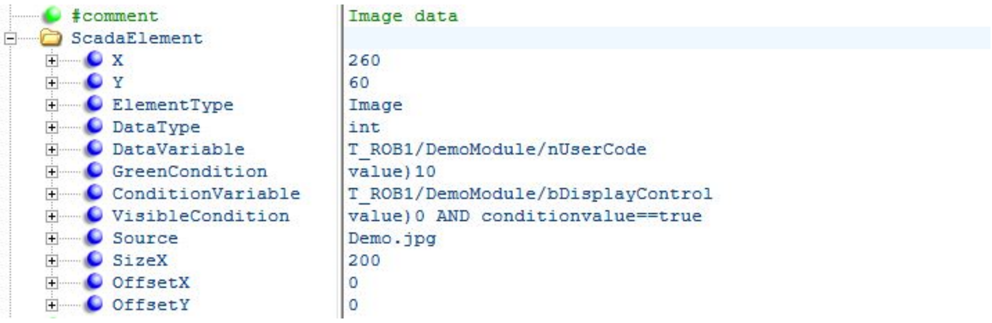


```
   #comment              Signal data from variable
⊟  ScadaElement
 ⊞   X                   0
 ⊞   Y                   160
 ⊞   ElementType         Signal
 ⊞   DataType            bool
 ⊞   DataVariable        T_ROB1/DemoModule/bGripperOpen
 ⊞   Label
 ⊞   SizeX               200
 ⊞   LookupElement       false
 ⊞   OffsetX             DemoGroupX
 ⊞   OffsetY             DemoGroupY
   #comment              Signal data from I/O
⊟  ScadaElement
 ⊞   X                   0
 ⊞   Y                   190
 ⊞   ElementType         Signal
 ⊞   DataType            signal
 ⊞   DataVariable        DOF_VisionLightsOn
 ⊞   Label
 ⊞   SizeX               200
 ⊞   LookupElement       false
 ⊞   OffsetX             DemoGroupX
 ⊞   OffsetY             DemoGroupY
```

xx1800002981

Image example

Simple example for Image display.

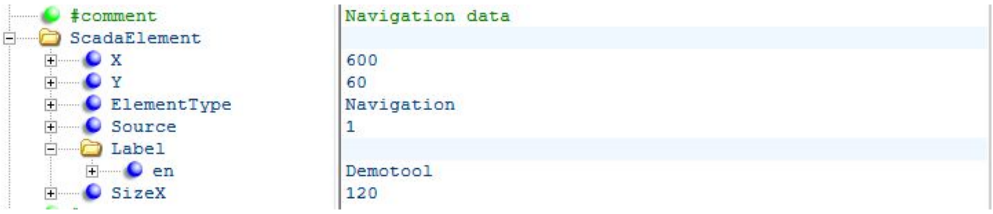


```
   #comment              Image data
⊟  ScadaElement
 ⊞   X                   260
 ⊞   Y                   60
 ⊞   ElementType         Image
 ⊞   DataType            int
 ⊞   DataVariable        T_ROB1/DemoModule/nUserCode
 ⊞   GreenCondition      value)10
 ⊞   ConditionVariable   T_ROB1/DemoModule/bDisplayControl
 ⊞   VisibleCondition    value)0 AND conditionvalue==true
 ⊞   Source              Demo.jpg
 ⊞   SizeX               200
 ⊞   OffsetX             0
 ⊞   OffsetY             0
```

xx1800002982

Navigation example

Simple example for Navigation display.



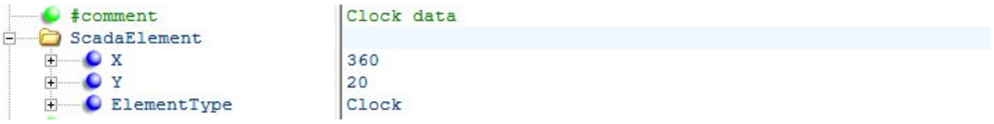
```
   #comment              Navigation data
⊟  ScadaElement
 ⊞   X                   600
 ⊞   Y                   60
 ⊞   ElementType         Navigation
 ⊞   Source              1
 ⊟   Label
  ⊞    en                Demotool
 ⊞   SizeX               120
```

xx1800002983

Continues on next page

Clock example

Simple example for Clock display.

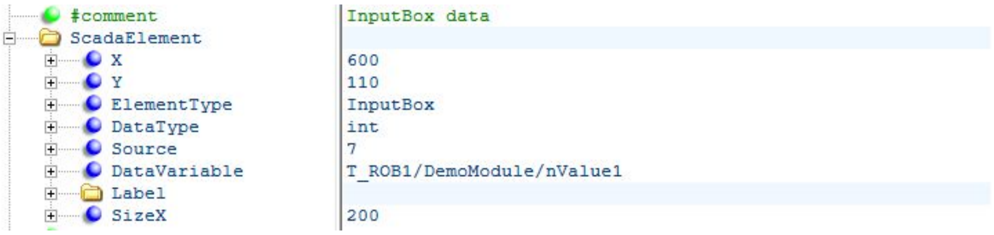


```
   #comment              Clock data
⊟  ScadaElement
   ⊞  X                  360
   ⊞  Y                  20
   ⊞  ElementType        Clock
```

xx1800002984

Inputbox example

Simple example for Inputbox display.

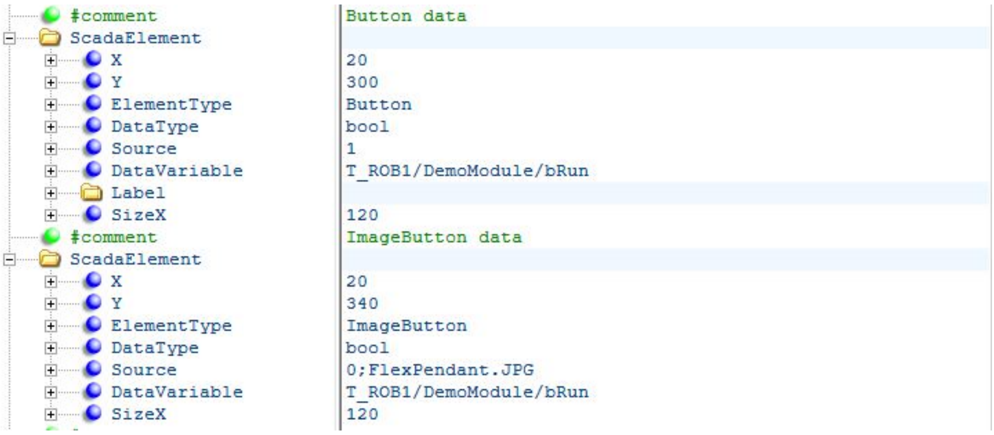


```
   #comment              InputBox data
⊟  ScadaElement
   ⊞  X                  600
   ⊞  Y                  110
   ⊞  ElementType        InputBox
   ⊞  DataType           int
   ⊞  Source             7
   ⊞  DataVariable       T_ROB1/DemoModule/nValue1
   ⊞  Label
   ⊞  SizeX              200
```

xx1800002985

Button / ImageButton example

Simple example for Button display.

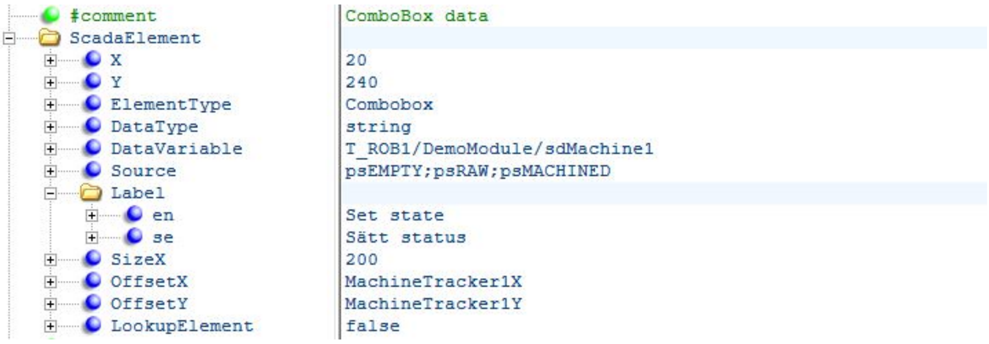


```
   #comment              Button data
⊟  ScadaElement
   ⊞  X                  20
   ⊞  Y                  300
   ⊞  ElementType        Button
   ⊞  DataType           bool
   ⊞  Source             1
   ⊞  DataVariable       T_ROB1/DemoModule/bRun
   ⊞  Label
   ⊞  SizeX              120
   #comment              ImageButton data
⊟  ScadaElement
   ⊞  X                  20
   ⊞  Y                  340
   ⊞  ElementType        ImageButton
   ⊞  DataType           bool
   ⊞  Source             0;FlexPendant.JPG
   ⊞  DataVariable       T_ROB1/DemoModule/bRun
   ⊞  SizeX              120
```

xx1800002986

ComboBox example

Simple example for ComboBox display.



```
   #comment              ComboBox data
⊟  ScadaElement
   ⊞  X                  20
   ⊞  Y                  240
   ⊞  ElementType        Combobox
   ⊞  DataType           string
   ⊞  DataVariable       T_ROB1/DemoModule/sdMachine1
   ⊞  Source             psEMPTY;psRAW;psMACHINED
   ⊟  Label
      ⊞  en              Set state
      ⊞  se              Sätt status
   ⊞  SizeX              200
   ⊞  OffsetX            MachineTracker1X
   ⊞  OffsetY            MachineTracker1Y
   ⊞  LookupElement      false
```

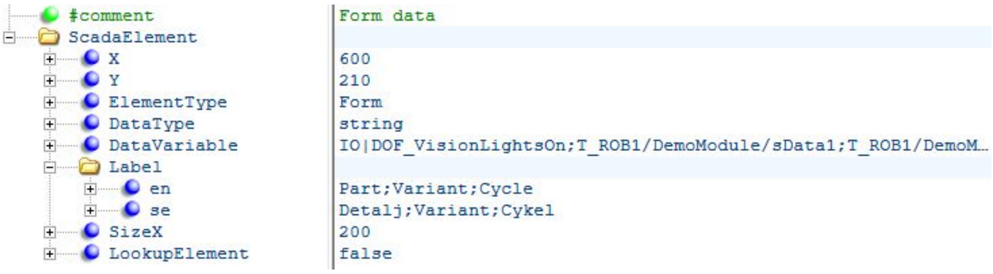
xx1900001058

Continues on next page

Form example

Simple example for Form display.

The Ok button for data transfer is added automatically.



#comment	Form data
ScadaElement	
X	600
Y	210
ElementType	Form
DataType	string
DataVariable	IO|DOF_VisionLightsOn;T_ROB1/DemoModule/sData1;T_ROB1/DemoM...
Label	
en	Part;Variant;Cycle
se	Detalj;Variant;Cykel
SizeX	200
LookupElement	false

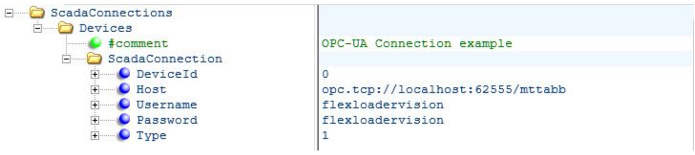
xx2000000628

Data connections

Valid data sources for visualization can be data available in the robot controller and data available from OPC-UA servers.

Data from the robot controller is directly addressed in the DataVariable. Data from OPC-UA servers require a ScadaConnection definition. These are defined in the **scadaConnections.xml** file. Every information page can use only one of the defined OPC-UA servers.

Currently the system is limited to a single OPC-UA server, using any DeviceId >0.



ScadaConnections	
Devices	
#comment	OPC-UA Connection example
ScadaConnection	
DeviceId	0
Host	opc.tcp://localhost:62555/mttabb
Username	flexloadervision
Password	flexloadervision
Type	1

xx1800002960

XML element	Description
DeviceID	Currently not used. Set to 0. Reference ID number that is used in ScadaElement definitions, e.g. 0.
Host	OPC-UA server supplying the data, e.g opc.tcp://localhost:62555/mttabb
Username	User name for login to OPC-UA server.
Password	Password for login to OPC-UA server.

```
<ScadaConnections>
  <Devices>
    <!-- OPC-UA Connection example-->
    <ScadaConnection>
      <DeviceId>0</DeviceId>
      <Host>opc.tcp://localhost:62555/mttabb</Host>
      <Username>flexloadervision</Username>
      <Password>flexloadervision</Password>
    </ScadaConnection>
  </Devices>
</ScadaConnections>
```

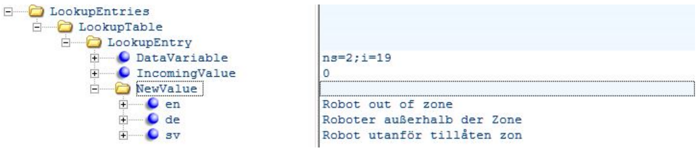
Continues on next page

Lookup tables

Many systems report information, e.g. status or error codes, by numerical values. It is preferable to display these values as text messages to the operator for better understanding

The system provides lookup tables as a method to translate numerical values from data variables to localized text messages.

Each LookupEntry specifies how a given value for a specific data variable shall be changed to a new value.



```
LookupEntries
  LookupTable
    LookupEntry
      DataVariable        ns=2;i=19
      IncomingValue       0
      NewValue
        en                Robot out of zone
        de                Roboter außerhalb der Zone
        sv                Robot utanför tillåten zon
```

xx1800002961

XML element	Description
DataVariable	Specifies data variable source. For syntax see *Data variables on page 196*.
Incoming-Value	Value that shall be changed to an new value
NewValue	Specifies the text shown. English text must always be specified. The text can be localized, see *Localized text on page 198*.

```
<LookupEntries>
  <LookupTable>
    <LookupEntry>
      <DataVariable>ns=2;s=DataValueBox</DataVariable>
      <IncomingValue>0</IncomingValue>
      <NewValue>
        <en>Robot out of zone</en>
        <de>Roboter außerhalb der Zone</de>
        <sv>Robot utanför tillåten zon</sv>
      </NewValue>
    </LookupEntry>
  </LookupTable>
</LookupEntries>
```

Practical tools

Some tools simplify the design of the information pages.

Press F5 while displaying an information page, and the **Safety** and **SCADA** tab definitions will be refreshed and re-initialized. Changes to XML files, images or folder structure will thus have immediate effects.

Click inside any information page, and the coordinates for this point are shown. These coordinates can be used for placement of the graphical elements.

Error handling

Data must be available from either the robot controller or the OPC-UA hosts.

Continues on next page

If the robot controller or a OPC-UA host is not available, a graphical indication will be displayed in the lower right corner.

If a robot or OPC-UA variable is not accessible or undefined, no value is shown. For a Textbox element with GreenCondition, a grey bar will be displayed. For a Signal element, a grey indication will be shown.

Special cases

The polling time is normally fixed to its standard value (1s for data variables, 5s for condition variables). However, it can be changed if needed.

For user defined polling times of data variables, please create a file named **polltime.txt** in the **FlexLoaderVision\Bin** folder.

The file must contain a number specifying the poll time in seconds.

The poll time for condition variables cannot be changed.

12.7 Other system settings

Installed software

A number of programs is pre-installed on the computer in order to run FlexLoader Vision. The following programs are required:

- FlexLoader Vision.
- Robot Communication Runtime.
- Basler Pylon Driver (for configuration and advanced setting of the camera).
- Matrox Imaging Library.
- If necessary, software for other camera types (e.g. 3D camera).

Several utilities are pre-installed or ready to install for ease-of-use and interactions with other FlexLoader function packages and standard cells.

- ABB RobotStudio (for configuration and advanced setting of the robot).
- ABB PlutoManager (for configuration and advanced setting of FlexLoader safety solutions).
- Several utilities for viewing documentation, editing configuration files and enabling remote access.
- Please refer to the release notes for details.

Licensing

Licensing agreements

By installing and using FlexLoader Vision you agree to Microsoft and Matrox licensing agreements.

Licensing FlexLoader Vision

In order to run FlexLoader Vision, the robot license option 1554-1 Prepared for FLexLoader Vision is required.

Contact ABB for licensing on existing previous installations.

Licensing Matrox Imaging Library

Licensing for Matrox Imaging Library is done through Matrox hardware. Normally, no USB dongle or license code is needed.

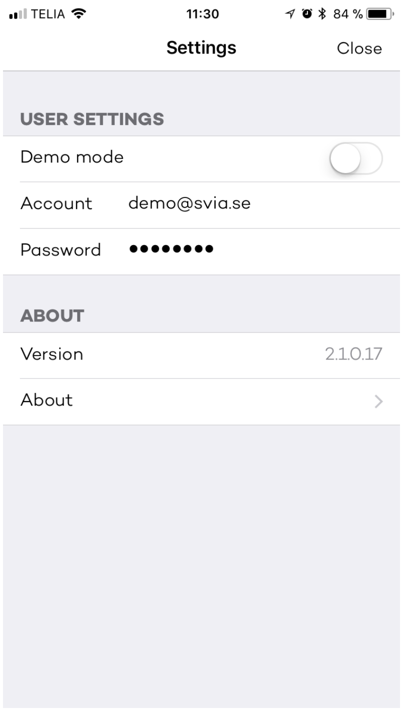
Contact ABB for licensing on non-Matrox hardware.

Continues on next page

Smartphone connection

Download and install the app. Start the app and enter the settings.

Turn off the demo mode and enter your customer login as obtained from ABB.



USER SETTINGS

Demo mode ○

Account demo@svia.se

Password ••••••••

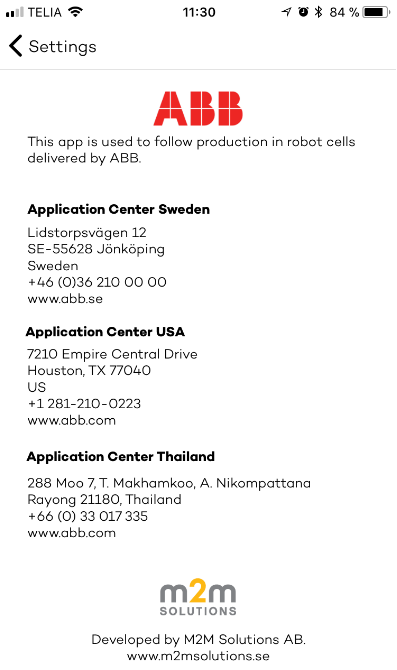
ABOUT

Version 2.1.0.17

About >

xx1800000514

For more information contact any of the application centers shown in the app.



This app is used to follow production in robot cells delivered by ABB.

Application Center Sweden

Lidstorpsvägen 12
SE-55628 Jönköping
Sweden
+46 (0)36 210 00 00
www.abb.se

Application Center USA

7210 Empire Central Drive
Houston, TX 77040
US
+1 281-210-0223
www.abb.com

Application Center Thailand

288 Moo 7, T. Makhamkoo, A. Nikompattana
Rayong 21180, Thailand
+66 (0) 33 017 335
www.abb.com

Developed by M2M Solutions AB.
www.m2msolutions.se

xx1800000515

Continues on next page

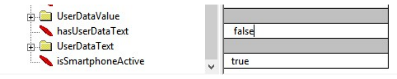
Make sure that the FlexLoader Vision PC is connected to the internet and that the SmartPhone settings are enabled and configured in the configuration editor (see *License data on page 169*).

In case of connectivity problems ensure that network configuration is correct. A common problem is that a default gateway has been configured for the **Robot** or **GigE** connection. Normally, only the outbound network connection is to have a default gateway specified.

Data is stored locally in case of lost network connectivity. Upon reconnection, this data is reported to the app.

The data and time settings on the PC must be correctly set and match the SmartPhone settings.

Using a SmartPhone connection with FlexLoader Vision Lite requires an additional setting in **FlexLoaderVisionLite.xml**. In the section **BaseData**, set the value **isSmartphoneActive** to **true**.



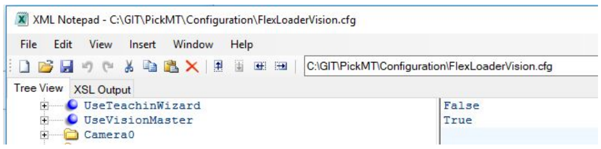
UserDataValue	
hasUserDataText	false
UserDataText	
isSmartphoneActive	true

xx1800000516

VisionMaster functionality

Use of the VisionMaster functionality must be enabled manually.

Edit **FlexLoaderVision.cfg** and set the **UseVisionMaster** entry to True.



XML Notepad - C:\GIT\PickMT\Configuration\FlexLoaderVision.cfg
File
Edit
View
Insert
Window
Help
C:\GIT\PickMT\Configuration\FlexLoaderVision.cfg
Tree View
XSL Output

UseTeachinWizard	False
UseVisionMaster	True
Camera0	

xx2000000627

13 Maintenance

13.1 General maintenance

This section covers maintenance of both FlexLoader Vision and options that are not included in all installations. If the installation contains further parts, maintenance instructions may also be found in other documentation.

FlexLoader Vision is made of components that have a minimal maintenance requirement. However, there are some components that require scheduled maintenance.

Always make sure that FlexLoader Vision is turned off during maintenance.

Below is an overview of the maintenance intervals and the corresponding corrective actions.

Maintenance intervals	Corrective action
Every 3 months	Check the cables and the cable rack. Check the connections.
When necessary	Clean the camera optics. Clean the PC screen.
If the system has been changed	Backup the robot program. Backup the vision system.
After major system updates	Create recovery image of vision PC.
Automatic	Windows updates (performed as long as PC has internet connectivity)

Check the cables and cable racks

What: Check the cables and cable racks.

When: Every 3 months.

How: Inspect the entire cable rack (fixing points, dirt deposits, wear) and remedy any breaks. Check all cables. Replace any damaged cables. Extend cables that rub against sharp edges.

Check the connections

What: Check the connections.

When: Every 3 months.

How: Inspect all connections and make sure that they are secured properly and that there is no risk of play.

Backup the vision system

What: Do a backup of the details stored in the vision system.

When: When you have made a change to the system.

How: Create a backup on the **Service** tab on the **Settings** menu and save it onto a secure media.

Continues on next page

Backup the robot program

What: Do a backup of the robot program.

When: When you have made a program change.

How: Follow the robot manual and save the copy onto a secure media.

Recovery image of the vision system

What: Do a complete recovery image of the vision system.

When: After major system changes, upgrades or updates.

How: Follow instructions in the FlexLoader Vision product manual.

Clean the camera optics

What: Clean the camera optics.

When: When the image quality is reduced by dirt.

How: Wipe the front of the object using a clean cloth. If this is not sufficient, use pure alcohol or similar. Ensure not to leave any streaks on the lens.

> **Note**
>
> If you happen to change the camera position, perform a new calibration.

Clean the computer screen

What: Carefully wipe the screen off using a damp cloth.

When: As necessary.

Windows updates

What: Download and install of Windows updates.

When: Windows updates are downloaded and installed by standard Windows functionality as long as the PC has internet connectivity.

How: Normally no user interaction is necessary. If the system needs to restart this is performed by standard windows functionality.

> **Note**
>
> Feature updates shall not be installed on the PC. This is defined by deselecting the "Defer feature updates" switch.

13.2 Backup and recovery of the hard disk

Introduction

This section describes how to backup and restore the entire FlexLoader Vision PC hard disk.

The backup can be used to restore the PC in case of a hard disk failure. Backups are normally made after an initial system installation and after major system changes (e.g. upgrades).

The normal file backup function should be used for continuous backup of FlexLoader Vision working data. A complete hard disk recovery is only to be performed by trained personal that is authorized to handle backup and restore operations.

Tip

We strongly recommend to keep valid backups of both the PC hard disk as well as continuous backup of FlexLoader Vision data.

Tip

We strongly recommend to duplicate the USB memory and/or to save the backup files at a safe location.

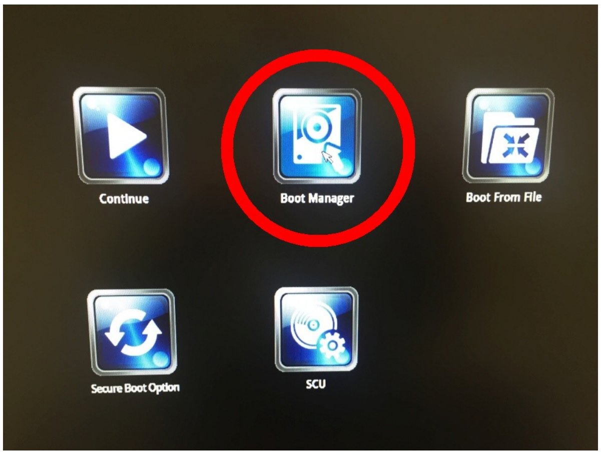
The backup and restore tool used here is **Redo Backup**, which is available under the GNU public license, but any other tool for hard disk imaging can be used.

Backup

To perform a backup, turn off the computer and put the backup USB in an available USB port (preferably the USB3 port).

Enter the boot device manager by pressing Esc when the startup screen is shown. If the computer boots into Windows, you need to restart and try again.

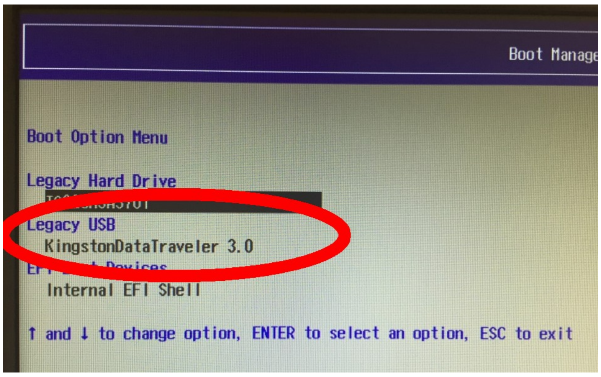
Select **Boot manager**.



xx1800000385

Continues on next page

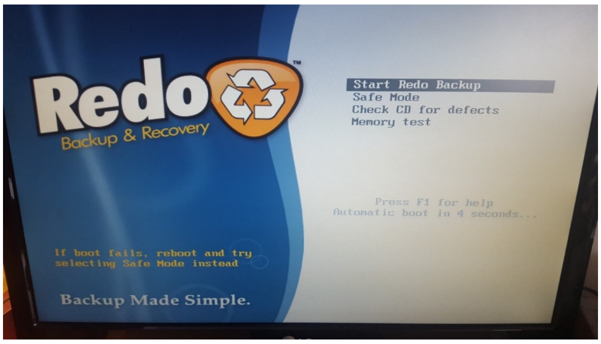
Choose to boot from the USB device



↑ and ↓ to change option, ENTER to select an option, ESC to exit

xx1800000386

Click **Start Redo Backup** (or wait).

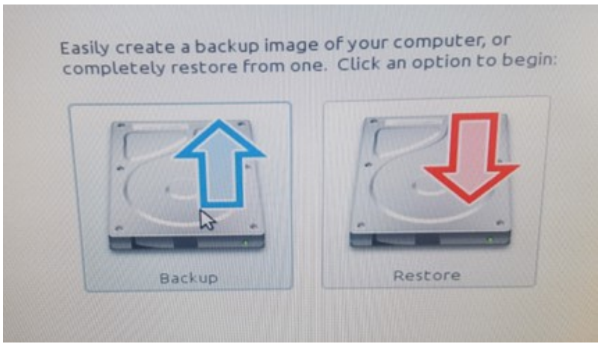


Press F1 for help
Automatic boot in 4 seconds...

If boot fails, reboot and try selecting Safe Mode instead

xx1800000387

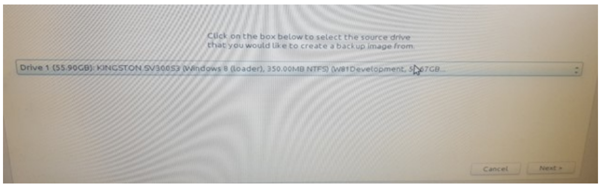
Click **Backup**.



Easily create a backup image of your computer, or completely restore from one. Click an option to begin:

xx1800000388

Step 1: Select the source device containing Windows and FlexLoader Vision in the drop-down list and click **Next**.



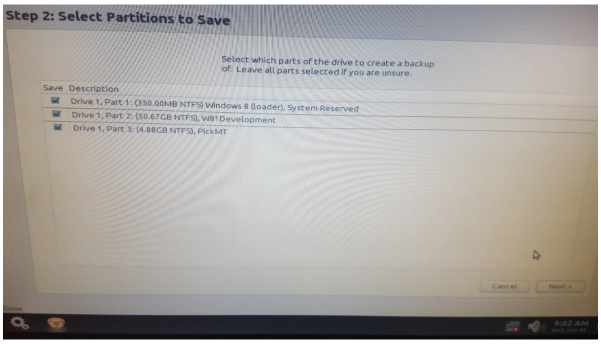
Click on the box below to select the source drive that you would like to create a backup image from.

Drive 1 (55.90GB): KINGSTON SV300S3 (Windows 8 (loader), 350.00MB NTFS) (W81Development, 55.67GB...

xx1800000396

Continues on next page

Step 2: Check that all partitions on the hard drive are selected (all check boxes are selected) and click **Next.**



Step 2: Select Partitions to Save

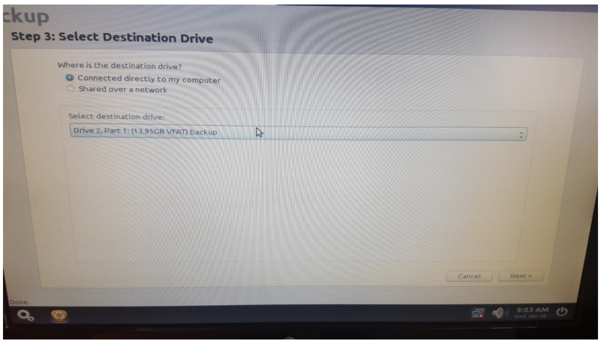
Select which parts of the drive to create a backup of. Leave all parts selected if you are unsure.

Save	Description
☑	Drive 1, Part 1: (350.00MB NTFS) Windows 8 (loader), System Reserved
☑	Drive 1, Part 2: (50.67GB NTFS), W81Development
☑	Drive 1, Part 3: (4.88GB NTFS), PickMT

Done.

xx1800000397

Step 3: Select **Connected directly to my computer**. Then select the USB device (**Drive 2, select largest Part**) as destination drive in the drop-down list and click **Next.**



ckup

Step 3: Select Destination Drive

Where is the destination drive?

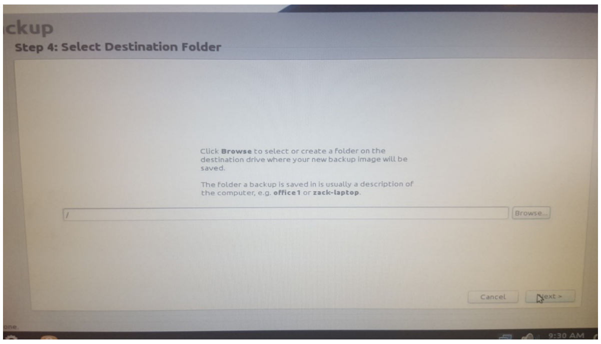
● Connected directly to my computer
○ Shared over a network

Select destination drive:
Drive 2, Part 1: (13.95GB VFAT) backup

Done.

xx1800000398

Step 4: Click **Next.** There is no need to specify a destination folder.



ckup

Step 4: Select Destination Folder

Click **Browse** to select or create a folder on the destination drive where your new backup image will be saved.

The folder a backup is saved in is usually a description of the computer, e.g. **office1** or **zack-laptop**.

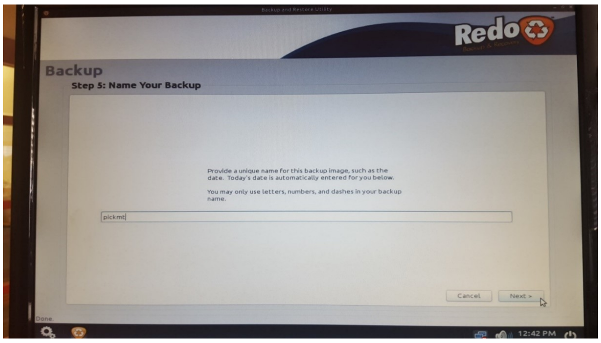
/

one.

xx1800000399

Continues on next page

Step 5: Name your backup **flexloadervision** (so that you can easily overwrite it, in case you need to refresh the backup). Click **Next.**



Step 5: Name Your Backup

Provide a unique name for this backup image, such as the date. Today's date is automatically entered for you below.

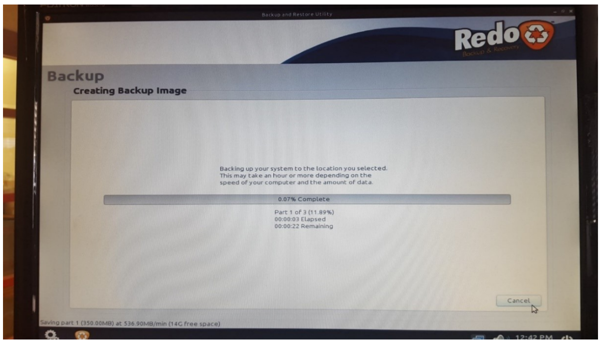
You may only use letters, numbers, and dashes in your backup name.

pickmt

Done.

xx1800000400

Finally, click **Yes** to start the backup.



Creating Backup Image

Backing up your system to the location you selected. This may take an hour or more depending on the speed of your computer and the amount of data.

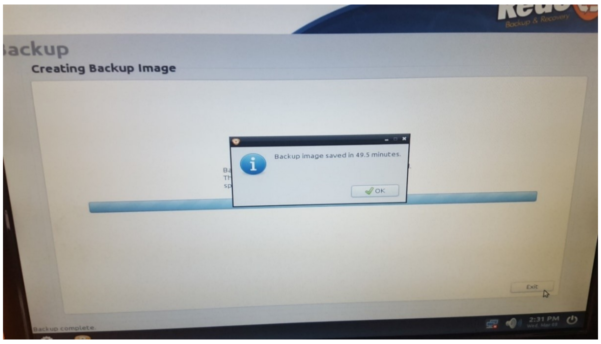
0.07% Complete

Part 1 of 3 (11.89%)
00:00:03 Elapsed
00:00:22 Remaining

Saving part 1 (350.00MB) at 536.90MB/min (14G free space)

xx1800000401

When the pop-up message that the back up is complete is displayed, click **OK.**



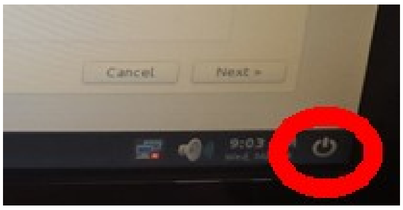
Creating Backup Image

Backup image saved in 49.5 minutes.

Backup complete.

xx1800000402

Remove the USB memory and restart the computer.



xx1800000403

Continues on next page

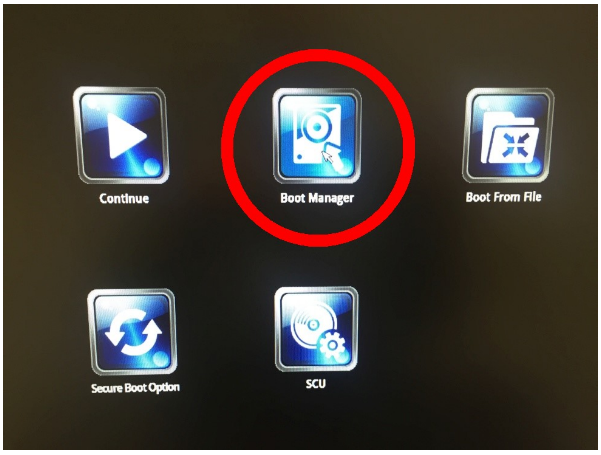
Save the backup file to a safe location, preferably not on the same computer as the software.

Restore

To restore the backup, turn off the computer and put the backup USB in an available USB port (preferably the USB3 port).

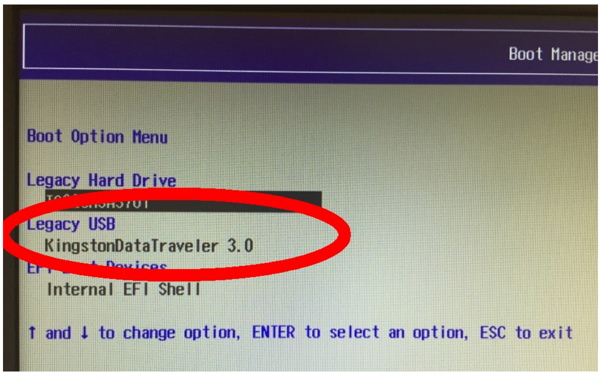
Enter the boot device manager by pressing Esc when the startup screen is shown. If the computer boots into Windows, you need to restart and try again.

Select **Boot manager**.



xx1800000385

Choose to boot from the USB device.

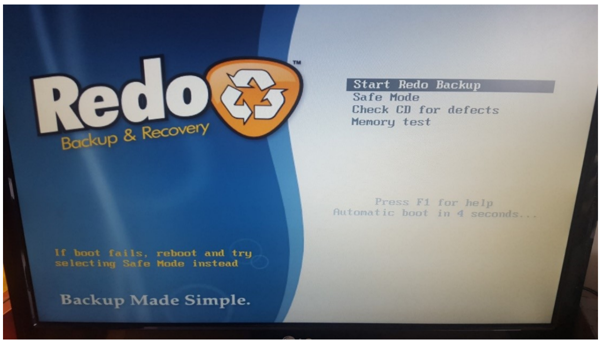


```
                                         Boot Manage

Boot Option Menu

Legacy Hard Drive
  [illegible]
Legacy USB
  KingstonDataTraveler 3.0
EFI Boot Devices
  Internal EFI Shell

↑ and ↓ to change option, ENTER to select an option, ESC to exit
```

xx1800000386

Continues on next page

Click **Start Redo Backup** (or wait).

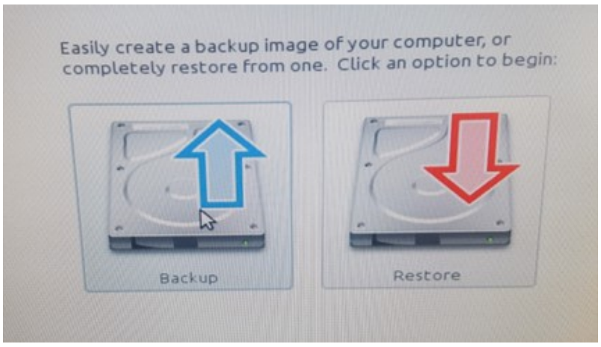


Press F1 for help
Automatic boot in 4 seconds...

If boot fails, reboot and try selecting Safe Mode instead

xx1800000387

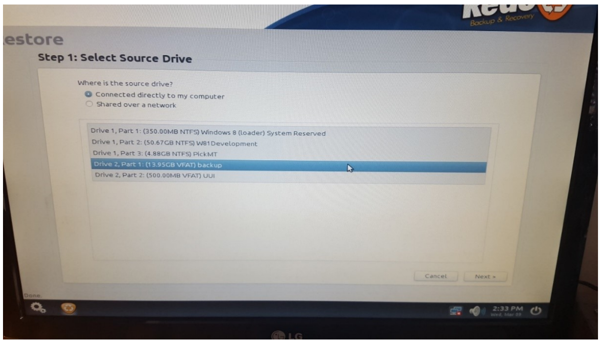
Click **Restore.**



Easily create a backup image of your computer, or completely restore from one. Click an option to begin:

xx1800000388

Step 1: Select the USB device (**Drive 2**, **Part 1**) as source drive.



Step 1: Select Source Drive

Where is the source drive?

● Connected directly to my computer
○ Shared over a network

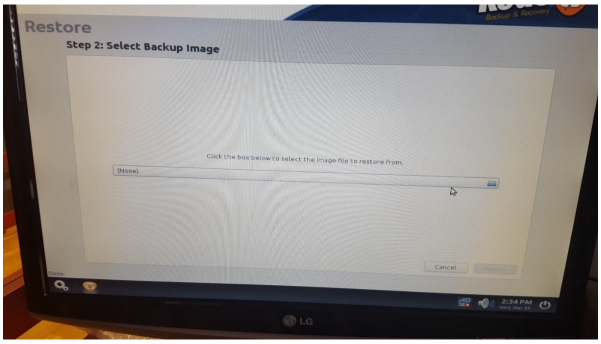
Drive 1, Part 1: (350.00MB NTFS) Windows 8 (loader) System Reserved
Drive 1, Part 2: (50.67GB NTFS) W81Development
Drive 1, Part 3: (4.88GB NTFS) PickMT
Drive 2, Part 1: (13.95GB VFAT) backup
Drive 2, Part 2: (500.00MB VFAT) UUI

Done.

xx1800000389

Continues on next page

Step 2: Browse to the **flexloadervision** backup file and click **Open**.



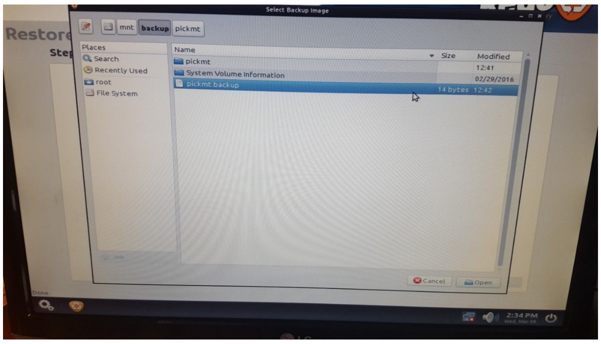
Restore

Step 2: Select Backup Image

Click the box below to select the image file to restore from.

(None)

xx1800000390

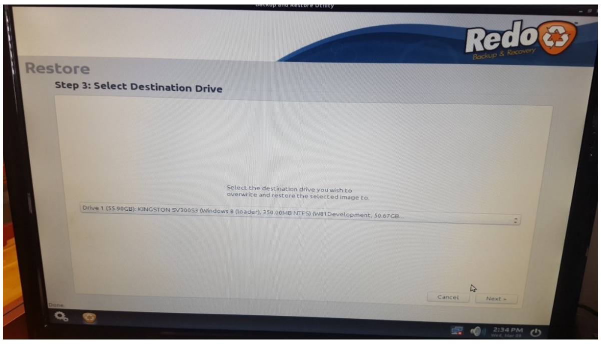


Select Backup Image

Name	Size	Modified
pickmt		12:41
System Volume Information		02/29/2016
pickmt.backup	14 bytes	12:42

xx1800000391

Step 3: Select **Drive 1** as destination drive in the drop-down list and click **Next**.



Restore

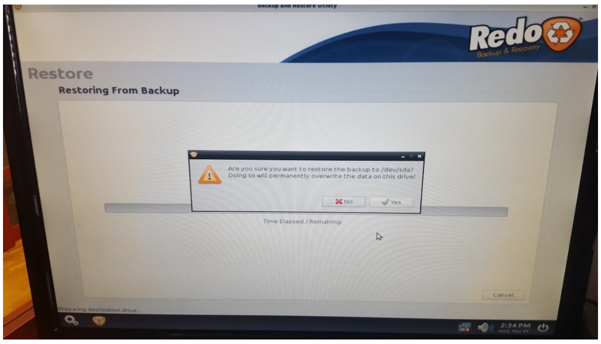
Step 3: Select Destination Drive

Select the destination drive you wish to overwrite and restore the selected image to.

Drive 1 (55.90GB): KINGSTON SV300S3 (Windows 8 (loader), 350.00MB NTFS) (W81Development, 50.67GB...

xx1800000392

Click **Yes** to confirm that you want to overwrite the FlexLoader Vision harddrive.



Restore

Restoring From Backup

Are you sure you want to restore the backup to /dev/sda?
Doing so will permanently overwrite the data on this drive!

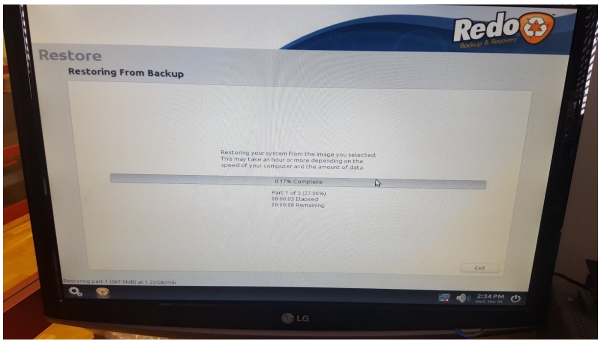
Time Elapsed / Remaining:

Preparing destination drive...

xx1800000393

Continues on next page

The restore procedure starts and a progress bar appears.



Restoring your system from the image you selected.
This may take an hour or more depending on the
speed of your computer and the amount of data.

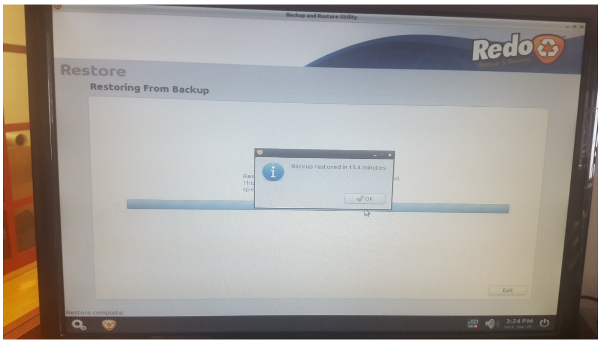
0.17% Complete

Part 1 of 3 (27.06%)
00:00:03 Elapsed
00:00:08 Remaining

Restoring part 1 (367.0MB) at 1.22GB/min

xx1800000394

In a couple of minutes the process is completed and a pop-up message that the restore procedure is complete is displayed. Click **OK**.

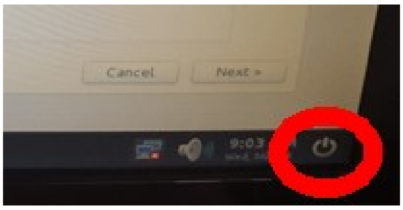


Backup restored in 15.4 minutes.

Restore complete.

xx1800000395

Remove the USB memory and restart the computer:



xx1800000403

13.3 Upgrading FlexLoader Vision

Upgrading the software

The version numbering system for FlexLoader Vision follows the Major.Minor.Revision pattern, e.g. FlexLoader Vision5.0.1 indicates that 5 is a major upgrade, 0 is a minor upgrade and 1 is a revision.

If the upgrade only involves a revision to the current version, all you need to do is replace **FlexLoaderVision.exe**. This file can be found in the bin directory of FlexLoader Vision, normally **D:\Program Files\ABB Industrial IT\Robotics IT\FlexLoaderVision\Bin**.

In the case of major or minor upgrades, the database and configuration file must also be updated. In most cases, however, a complete new installation is required, which may also mean that the details must be taught again. For more information on this, see the version history of FlexLoader Vision.

Note

Always make a complete backup before carrying out an upgrade.

Restoring a backup of FlexLoader Vision

To restore FlexLoader Vision from a backup, click the **Settings** icon in FlexLoader Vision and select the **Service** tab. Click **Restore** and enter the path to the backup file. When the backup has been restored, FlexLoader Vision must be restarted. For more information, see *Backup and restore FlexLoader Vision on page 124*.

If the restore operation fails and FlexLoader Vision cannot be restarted at all, you need to do a manual backup, where the contents in the backup directory are copied directly to the FlexLoader Vision directory. For more information on this, see *Manually restoring a backup on page 304*.

Robot backups are restored via the robot operator panel.

This page is intentionally left blank

14 Robot integration

14.1 Introduction

This chapter describes how the ABB robot communicates and works together with FlexLoader Vision and how to integrate the ABB robot with FlexLoader Vision.

In order for FlexLoader Vision to send the correct coordinates to the robot, the systems must be calibrated together, so that for example the coordinate systems of FlexLoader Vision and the robot are made to correspond. Use the supplied calibration plate and calibration tool during the calibration.

See *Camera on page 109* for information on camera calibration.

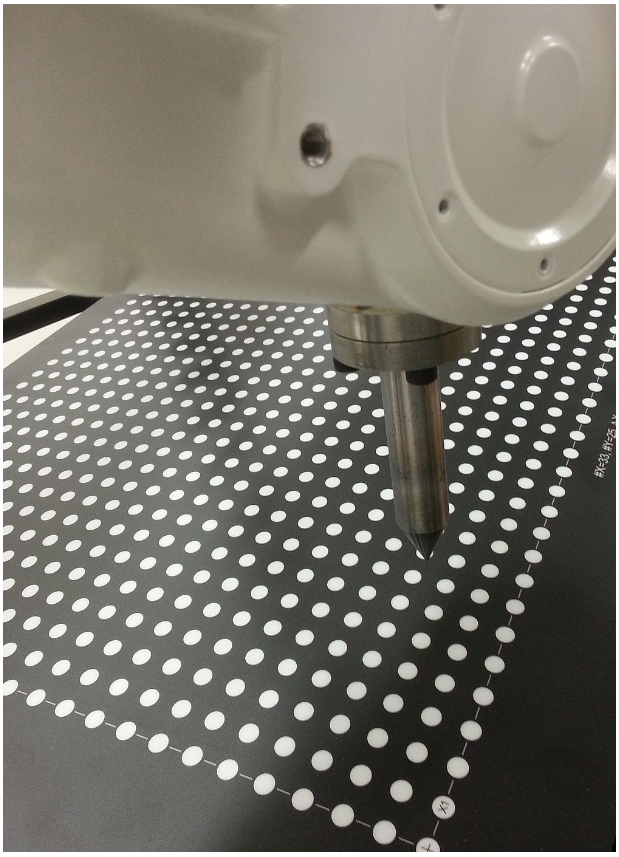
14.2 Calibrating the robot

General

When the calibration of FlexLoader Vision is complete, you will need to calibrate the robot. This is done from the FlexPendant.

Calibrating for use with a 2D camera

When calibrating the robot for use with a 2D camera, a calibration plate and a calibration tool is needed. Note that the calibration plate must not be moved until the robot calibration is complete. To ensure that the calibration plate is not moved, it can be taped to the camera belt.



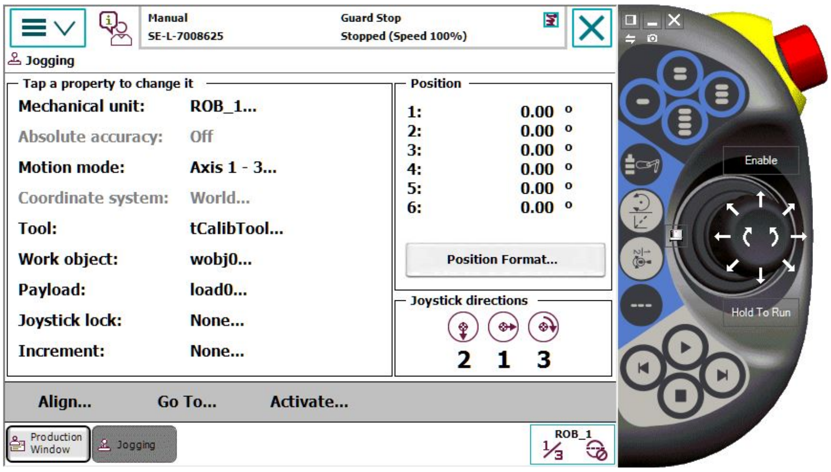
xx1800000260

Figure 14.1: The calibration tool, positioned over the calibration plate

Continues on next page

Defining the calibration tool

1. Attach the supplied calibration tool to the robot. Note that the calibration tool may look different from that shown in image above, depending on which type of robot and which other gripper equipment is used in the specific application.



Tap a property to change it

Mechanical unit:	ROB_1...
Absolute accuracy:	Off
Motion mode:	Axis 1 - 3...
Coordinate system:	World...
Tool:	tCalibTool...
Work object:	wobj0...
Payload:	load0...
Joystick lock:	None...
Increment:	None...

Position

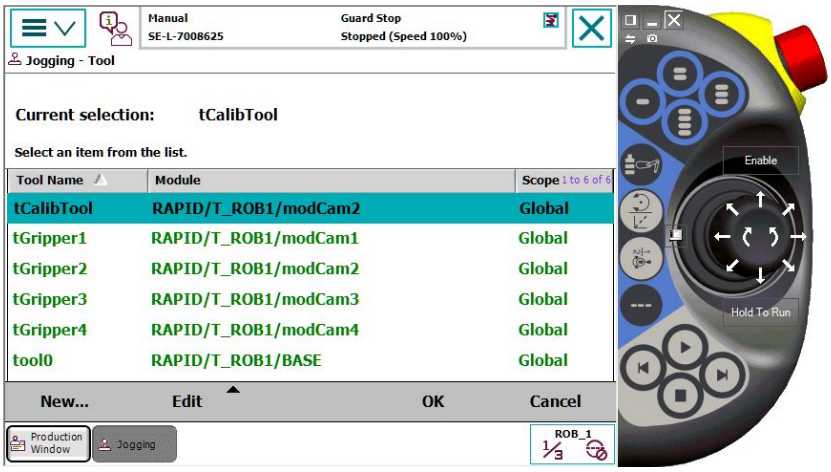
1:	0.00 °
2:	0.00 °
3:	0.00 °
4:	0.00 °
5:	0.00 °
6:	0.00 °

Position Format...

Joystick directions
2 1 3

xx1800000262

2. Open the main menu on the FlexPendant and select **Jogging** > **tCalibTool**.



Current selection: tCalibTool

Select an item from the list.

Tool Name	Module	Scope 1 to 6 of 6
tCalibTool	RAPID/T_ROB1/modCam2	Global
tGripper1	RAPID/T_ROB1/modCam1	Global
tGripper2	RAPID/T_ROB1/modCam2	Global
tGripper3	RAPID/T_ROB1/modCam3	Global
tGripper4	RAPID/T_ROB1/modCam4	Global
tool0	RAPID/T_ROB1/BASE	Global

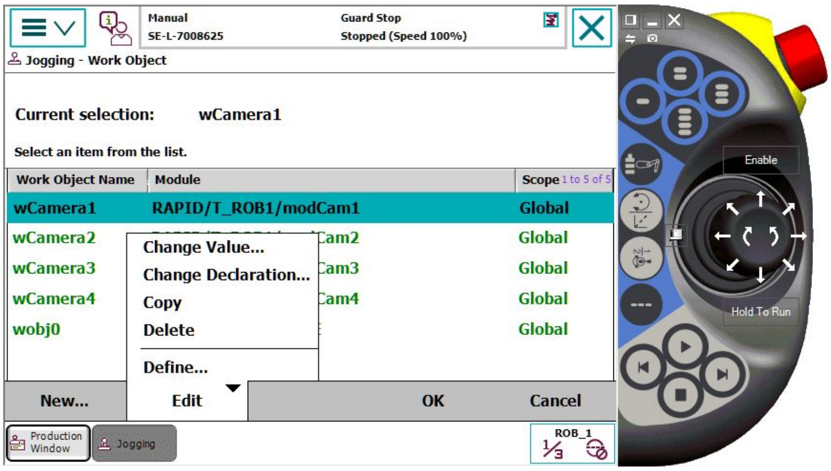
xx1800000263

Defining the work object

1. Open the main menu and select **Programdata** > **wobjdata** > **Show data**.
2. Select the work object to be calibrated. **wCamera1** for camera 1, **wCamera2** for camera 2, etc. Then tap **Edit** > **Define**.

> **Note**
>
> wCamera is only used temporarily in the robot and should not be calibrated.

Continues on next page



Manual SE-L-7008625 Guard Stop Stopped (Speed 100%)

Jogging - Work Object

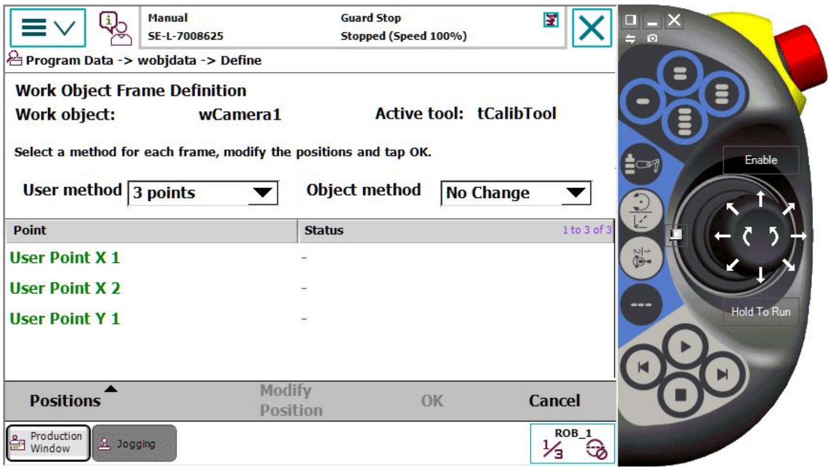
Current selection: wCamera1

Select an item from the list.

Work Object Name	Module	Scope 1 to 5 of 5
wCamera1	RAPID/T_ROB1/modCam1	Global
wCamera2	Cam2	Global
wCamera3	Cam3	Global
wCamera4	Cam4	Global
wobj0		Global

xx1800000264

3 For **User method**, select **3 points**. For **Object method**, select **No change.**



Manual SE-L-7008625 Guard Stop Stopped (Speed 100%)

Program Data -> wobjdata -> Define

Work Object Frame Definition

Work object: wCamera1 Active tool: tCalibTool

Select a method for each frame, modify the positions and tap OK.

User method: 3 points Object method: No Change

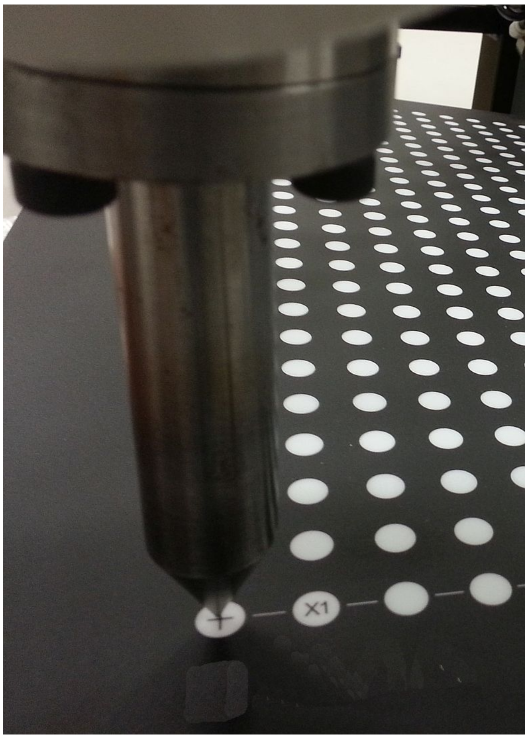
Point	Status 1 to 3 of 3
User Point X 1	-
User Point X 2	-
User Point Y 1	-

xx1800000265

Continues on next page

Performing the calibration

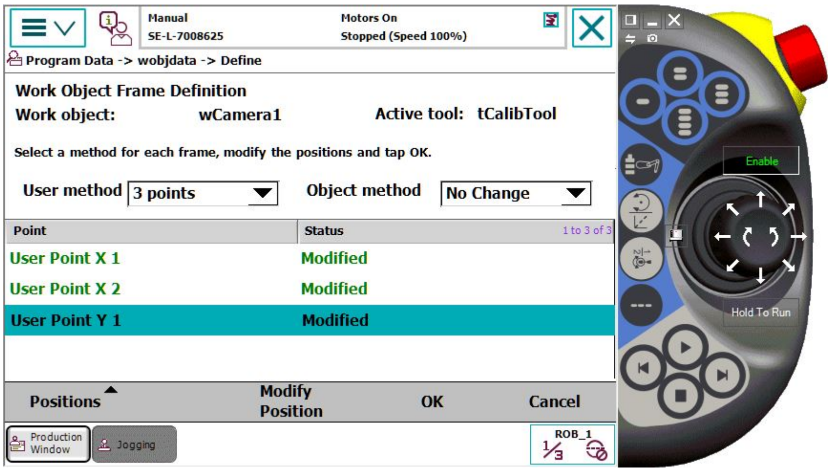
1 Jog the robot so that the tip of the calibration tool is positioned correctly, preferably at point X1 on the calibration plate (see image below). Optionally, a point on the X-axis near X1 can be selected.



xx1800000261

Figure 14.2: Positioning the calibration tool.

2 Select **User point X1** > **Modify position.** Repeat the jogging and position modifying process for positions X2 and Y1. Then tap **OK.**


Manual
SE-L-7008625

Motors On
Stopped (Speed 100%)

Program Data -> wobjdata -> Define

Work Object Frame Definition

Work object: wCamera1 Active tool: tCalibTool

Select a method for each frame, modify the positions and tap OK.

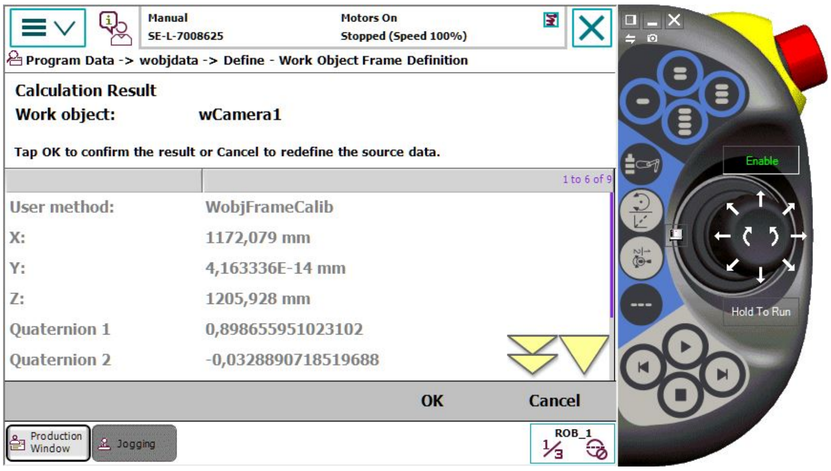
User method 3 points Object method No Change

Point	Status
User Point X 1	Modified
User Point X 2	Modified
User Point Y 1	Modified

xx1800000266

Continues on next page

3 A summary of the calibration results is displayed. The system is now calibrated and ready, but the calibration must be saved to a file to avoid the risk of losing it when the system is started up the next time.



Program Data -> wobjdata -> Define - Work Object Frame Definition

Calculation Result

Work object: wCamera1

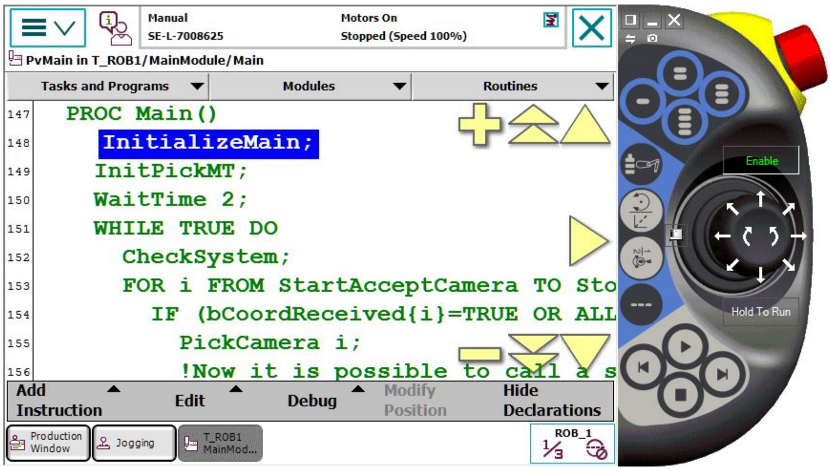
Tap OK to confirm the result or Cancel to redefine the source data.

User method:	WobjFrameCalib
X:	1172,079 mm
Y:	4,163336E-14 mm
Z:	1205,928 mm
Quaternion 1	0,898655951023102
Quaternion 2	-0,0328890718519688

xx1800000267

Saving the calibration file

1 Open the **ProgramEditor**.



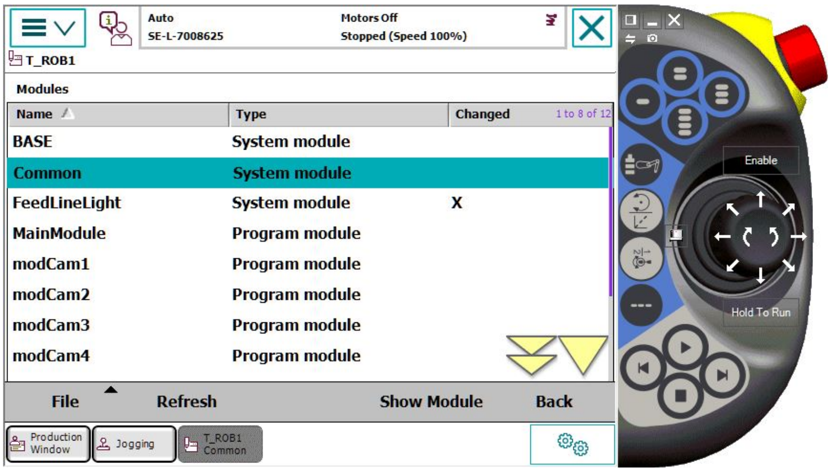
PvMain in T_ROB1/MainModule/Main

```
PROC Main()
  InitializeMain;
  InitPickMT;
  WaitTime 2;
  WHILE TRUE DO
    CheckSystem;
    FOR i FROM StartAcceptCamera TO Sto
      IF (bCoordReceived{i}=TRUE OR ALL
        PickCamera i;
        !Now it is possible to call a s
```

xx1800000315

Continues on next page

2 Select **Modules**. Then select the module **Common**.



Auto
SE-L-7008625

Motors Off
Stopped (Speed 100%)

T_ROB1

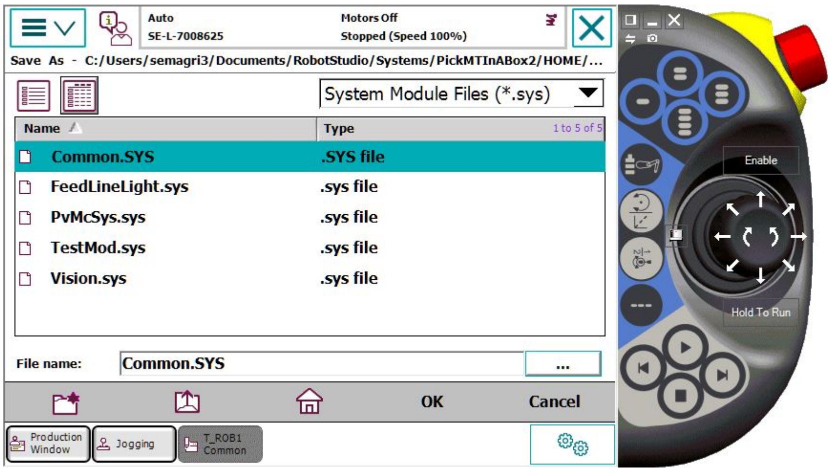
Modules

Name	Type	Changed
BASE	System module	
Common	System module	
FeedLineLight	System module	X
MainModule	Program module	
modCam1	Program module	
modCam2	Program module	
modCam3	Program module	
modCam4	Program module	

xx1800000268

3 Tap **File** and select **Save module as....**

4 Browse to the folder **HOME:/System**. Normally, there already is a file called **Common.sys** in this folder. Tap **OK** to confirm this is where you want save the calibration file.



Auto
SE-L-7008625

Motors Off
Stopped (Speed 100%)

Save As - C:/Users/semagri3/Documents/RobotStudio/Systems/PickMTInABox2/HOME/...

System Module Files (*.sys)

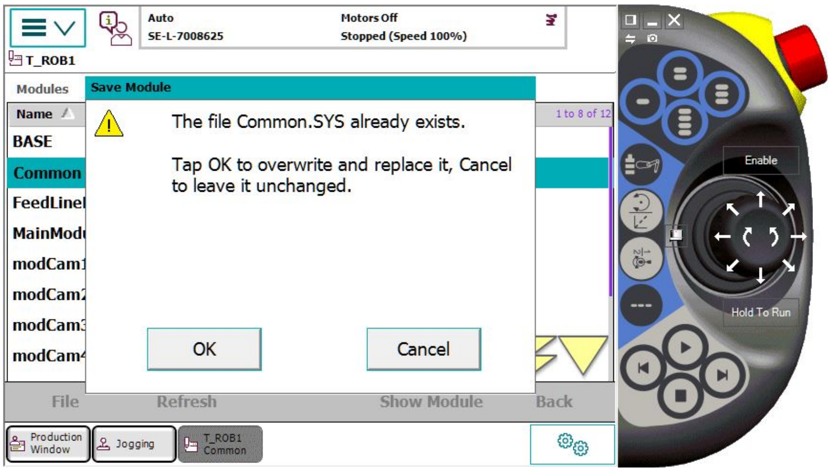
Name	Type
Common.SYS	.SYS file
FeedLineLight.sys	.sys file
PvMcSys.sys	.sys file
TestMod.sys	.sys file
Vision.sys	.sys file

File name: Common.SYS

xx1800000270

Continues on next page

5 Tap **OK** to confirm the next message about replacing the file.



Save Module

The file Common.SYS already exists.

Tap OK to overwrite and replace it, Cancel to leave it unchanged.

xx1800000271

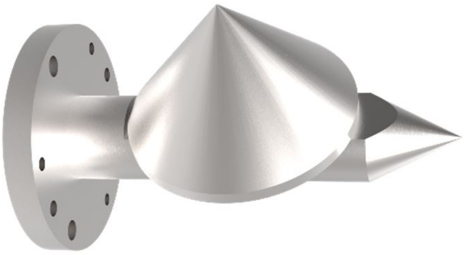
6 The calibration file has now been saved and the system is ready to start working with the current camera.

Calibrating for use with a 3D camera

Note

Before starting the calibration, ensure that **wCamera1** (if the 3D camera is camera 1) is suitably calibrated in the robot.

When calibrating the robot for use with a 3D camera you need the calibration tool and a calibration cone. An example of such tool is shown in the image below.



xx1800002010

Figure 14.3:

The robot operator must first calibrate the coordinate system **wCamera1** (if the 3D camera is camera 1). A suitable positioning is for example on the pallet guides (if a pallet is used). The z axis must be pointing upwards and the x axis must be aligned with the long side of the pallet. We recommend to select calibration points so that the Z=0 for wCamera is below the surface of the pallet bottom. See the previous section for help with defining and saving the work object.

The RAPID module **Calibration3D.sys** includes robot demo code that supports the calibration process. There are 15 recommended positions defined. Fit and activate

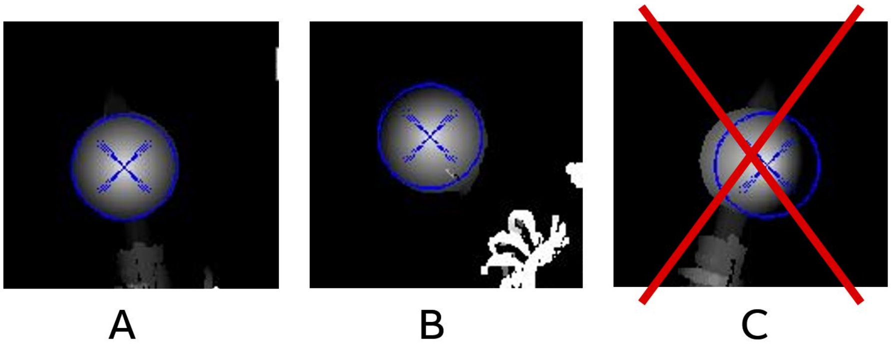
Continues on next page

the calibration cone as a tool with the tool center point (TCP) at the top of the cone. Check the predefined positions and modify them if necessary. They can be placed in many different ways.

For best results, it is recommended to include as much as possible of the 3D camera measurement volume used for robot picking. Note that it is possible to add and delete positions. Change **Calibration3DRoutine** accordingly so that the robot moves to the positions to be used.

The orientation of the cone with respect to the sensors internal coordinate system influences the accuracy of the calibration. For best results, the cone base shall be placed horizontally with respect to the sensor coordinate system. This is of special importance if the sensor is mounted inclined with respect to wCamera1.

A warning is issued during calibration if the angle deviation is more than 4°, an alarm is issued if the angle deviation is more than 8°. The calibration will proceed, however, a degradation in accuracy can be expected. The figures below show visual examples of both good, acceptable and unacceptable alignment.



xx1800002614

A	Well aligned calibration cone.
B	Slightly misaligned calibration cone, slight increase in calibration error. Avoid if possible.
C	Unacceptably misaligned calibration coned, relevant increase in calibration error. Mandatory to adjust cone orientation.

When FlexLoader Vision requests to start the robot at the time of calibration, the routine **Calibration3DRoutine** must be called from the FlexPendant. If all movements are already tested, the robot can be switched to automatic mode to simplify the calibration process. The RAPID module contains code that communicates with FlexLoader Vision and sends all the information needed to FlexLoader Vision.

Note

The module name **Calibration3D** must not be changed. Neither may the names of the communication variables **nConeCalibrationAnswer**, **n3DCalibrateX**, **n3DCalibrateY**, and **n3DCalibrateZ**.

14.3 RAPID program

14.3.1 Overview

⚠ **WARNING**

Applicable training is required in order to use the product. Incorrect use of the product can lead to personal injury and material damage.
Before programming the FlexLoader Vision, it is your responsibility to carefully read the chapter on safety, to become familiar with the FlexLoader Vision and its options, and to become familiar with the safety devices.

This chapter is used for commissioning and adapting the code skeleton to the actual application. The information covers the basic structure of the robot program. More or less extensive additions and adjustments are required depending on the design and the functionality of the robot cell.

For more information on robot programming see the programming manual of the robot.

The FlexLoader RAPID code consists of several main blocks that supports the functionality of FlexLoader systems:

- FlexLoader application functionality
- FlexLoader Vision interface
- FlexLoader Vision Lite functionality
- FlexLoader RW Add-in
- FlexLoader assistance and utility functionality
- FlexLoader machine tool interface functionality

14.3.2 FlexLoader application functionality

Overview

The FlexLoader Vision application code handles the complete workflow of parts in the cell (e.g. unloading and loading of parts, option handling, picking from inconveyor and leaving on outconveyor).

The application code consists of a basic executable skeleton, which must be modified by the integrator in order to match the actual application. However, the basic structure should be followed to ensure smooth operation.

Main structure

Main.pgf and MainModule.mod are automatically loaded into the controller when a part is started from FlexLoader Vision. During the following initialization, project modules with part specific information are loaded. This happens upon every start (if not configured otherwise, see FlexLoader Vision ConfigurationEditor).

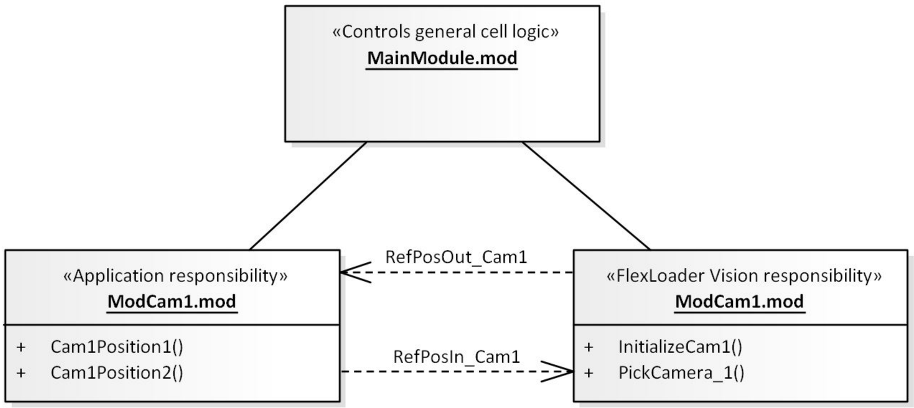
MainModule.mod is always the same for all parts and all operating modes of the robot cell. MainModule.mod is responsible for loading the modules that are linked to the parts to be run. Below, these are called PartCamX for a general reference to any PartCam module, or specifically PartCam1 for parts to be run using camera 1, PartCam2 for parts to be run using camera 2, and so on. These files can be renamed according to project needs.

Cell definitions like tool data, work objects and so on along with some event routines and help functions are placed in Common.sys and CommonCell / CommonSC3000 / CommonSC6000.

Application code vs FlexLoader Vision code

FlexLoader RAPID code has RAPID system code responsible for picking parts from the feeder. The interface between application code and system code for picking are so called reference positions.

RefPosIn is a robot position from which picking of parts is possible. After picking a part, the system code returns the robot to RefPosOut. The application part is responsible to handle parts, starting from RefPosOut, and handling all RAPID code until the robot can be returned to RefPosIn.



xx1800002289

Continues on next page

WARNING

After changing MainModule.mod or any PartCamX module, these modules MUST be saved, otherwise the changes will be lost upon next start due to the automatic program loading.

Example files

The following example programs show the principle structure of the robot code and how the robot program should communicate with FlexLoader Vision.

There is no guarantee that any of the examples work unless the program has been adjusted and tested for the specific robot cell.

RAPID example MainModule.mod

MainModule.mod contains the main routine which is the standard entry point. Some typical example application skeleton for MainModule.mod are shown below. All examples use the same initialization code.

```
PROC Main()
  ! set system specific control parameters
  InitializeMain;
  ! set vision specific control parameters
  InitVision;
  WaitTime 2;
  WHILE TRUE DO
    ! control end-of-cyle or entry requests
    CheckSystem;
    ! ----------------------------
    ! see example codes below
    ! ----------------------------
  WaitTime 0.05;
  ENDWHILE
ENDPROC
```

Generic solution

Many standard feeding solutions can be handled by the following schematic code. This is the most simple code example.

It can be used for picking from up to four feeders, e.g. FP 100, FP 300 and FP 400 or any FlexLoader function package.

```
! loop through desired cameras to perform picks and leave parts.
FOR i FROM nStartAcceptCamera TO nStopAcceptCamera DO
  ! This call will make the robot pick a part from camera i.
  ! When using desired position as in example, only parts found
  ! in position 1 is picked.
  PickPartAtCamera i,\DesiredPosition:=1;
  ! call part specific routine for the found position
  IF bDetailInGripper=TRUE THEN
    %"Cam"+ValToStr(i)+"Position_"+ValToStr(nPosition)%;
  ENDIF
ENDFOR
```

Continues on next page

For ease of use, the PickPartAtCamera routine combines some operations like grab image, wait for result, do retry if nothing found and evaluate which action vision proposed for the robot. The image grabbing could also be handled outside of the routine, either with autograb or with manual calls. Below is part of the code used in this routine.

Note the handling of flags for CHANGE_BIN, PICK_INTERLAYER and PICK_INTERLAYER_AND_CHANGE_BIN relevant for picking with 3D camera. Also note that there is another routine for handling picking several parts from same image. Read routine comments for PickMultiPartsAtCamera for information about this.

```
VisionResult{CameraNumber}:=GetNewVisionResult(CameraNumber
     \DesiredPosition?DesiredPosition,\MaxImageRetries?MaxImageRetries,
     \TimeoutTimeVision?TimeoutTimeVision);
TEST VisionResult{CameraNumber}.Action
CASE ABORTED:
  ! No valid coordinate from vision
  MT_SetAlarm "flv_ImageAborted",\WildCardTexts:=
     [VisionResult{CameraNumber}.AbortMsg];
CASE NO_DETAILS:
  ! Allow manual mode again if there is a feeder of that type
  IF nFeederType{CameraNumber}=FEEDER_NO_RETURN SetDOSignal
     "DOF_AllowManualModeFeeder"+ValToStr(CameraNumber),1;
  ! Nothing found in image, even after potential retries
  MT_SetAlarm "flv_NoPartFoundError";
  Stop;
  EXIT;
  ! TODO -------- ONLY FOR CELLS WITH NO BELT, EXAMPLE PALLET
     PICKER. REMOVE DOWN TO NEXT DASHED LINE IF NOT NEEDED!
     --------------
CASE CHANGE_BIN:
  ! No valid Pick position in pTarget
  ! TODO - Make sure pallet is switched
  Stop;
  ! Call ConfirmPick or RefPosOut to move out of camera area and
     confirm pick to vision.
  ! In cells with auto grab, this will also trigg a new image. Use
     RefPosOut_ to move out of camera area at the same time as
     confirming.
  CallByVar "ConfirmPick",CameraNumber;
CASE PICK_INTERLAYER:
  ! pTarget holds coordinates for an interlayer
  ! TODO - Pick interlayer with robot or ask operator to remove it
  Stop;
  ! Call ConfirmPick or RefPosOut to move out of camera area and
     confirm pick to vision.
  ! In cells with auto grab, this will also trig a new image. Use
     RefPosOut_ to move out of camera area at the same time as
     confirming.
  CallByVar "ConfirmPick",CameraNumber;
CASE PICK_INTERLAYER_AND_CHANGE_BIN:
  ! Action only in cells with no feeder
```

Continues on next page

```
    ! VisionResult{CAMERA_NO_1}.Target holds coordinates for an
       interlayer
    ! TODO - Pick interlayer with robot or ask operator to remove it
    ! TODO - Pallet is also empty, so make sure it's switched out
    Stop;
    ! Call ConfirmPick or RefPosOut to move out of camera area and
       confirm pick to vision.
    ! In cells with auto grab, this will also trig a new image. Use
       RefPosOut_ to move out of camera area at the same time as
       confirming.
    CallByVar "ConfirmPick",CameraNumber;
    ! ----------------------------------------------------------
  DEFAULT:
    ! Move robot to camera area
    CallByVar "RefPosInCam_",CameraNumber;
    ! Call routine to pick the part
    CallByVar "PickCam_",CameraNumber;
    ! Move robot out of camera area and confirm pick
    CallByVar "RefPosOutCam_",CameraNumber;
  ENDTEST
```

Single camera feeder with post-pick adjustments

This example is valid for a feeder with a single camera, where the same camera is used for the normal pick and for post-pick adjustments. This can be used for more exact positioning of the part in the gripper fingers.

```
! Belt setting: setup how belt should be controlled
nBeltAction:=[RUN_ONE_DETAIL,RUN_NEVER,RUN_NEVER,RUN_NEVER];
! Camera setting: grab images automatically or only from user
     program
bAllowAutoGrab:=[TRUE,FALSE,FALSE,FALSE];
```

In order to take the post-pick image, use the prepared routines for this. Note that the RAPID modules for this are optional and might not be included in the RAPID package if not asked for. See comments in RAPID code for more information about the functions in the example code.

```
VAR visionres vrResult;
! Call to Control image function to prepare for second image, grab
     that image and then reset camera to normal image settings.
vrResult:=ControlImage(CAMERA_NO_1,\ControlImagePositions:=[3,4],
     \NormalImagePositions:=[1,2],\CameraSettingsControl:=[80,70,5],
     \CameraSettingsNormal:=[60,60,10]);
```

The ControlImage routine will depend on the user having created or modified a couple of movement routines to help robot get to correct position for the post-pick image. Routines in example below must exist and have movements according to the comments.

```
PROC MoveToControlImage1()
! TODO - if control image is used add code here to move robot in
     position for control image
ENDPROC
```

Continues on next page

```
PROC MoveFromControlImage1()
! TODO - if control image is used add code here to move robot away
    and out of camera field of view
ENDPROC
```

Single camera feeder with additional camera for post-pick adjustments

This example is valid for a feeder, e.g. a FP 800 with an additional camera for post-pick adjustments. All camera image acquisitions are controlled manually due to process requirements.

```
! Belt setting: setup how belt should be controlled
nBeltAction:=[RUN_NEVER,RUN_NEVER,RUN_NEVER,RUN_NEVER];
! Camera setting: grab images automatically or only from user
    program
bAllowAutoGrab:=[FALSE,FALSE,FALSE,FALSE];
```

The main loop is handled according to code shown below. This example shows an example of how to handle that parts occasionally are not found.

```
! do initial image acquisition and pick of part
GrabImage CAMERA_NO_1;
RefPosInCam_1
WaitUntil bCoordReceived{CAMERA_NO_1}=TRUE;
SetNextTarget CAMERA_NO_1;

! handle that parts occasionally are not found.
WHILE nAction{CAMERA_NO_1} <= NO_DETAIL DO
  ! change camera settings if necessary
  DiscardAndTakeNewImage CAMERA_NO_1;
  WaitUntil bCoordReceived{CAMERA_NO_1}=TRUE;
  SetNextTarget CAMERA_NO_1;
  Incr nImageRetries{CAMERA_NO_1};
  IF nImageRetries{CAMERA_NO_1} > MAX_IMAGE_RETRIES{CAMERA_NO_1}
    THEN
    ! Create alarm or inform user in some other way
  EXIT;
  ENDIF
ENDWHILE

IF nAction{nCameraNumber}=CHANGE_BIN THEN
  ! TODO - Do something to take care of pallet change, can be
    removed if not a pallet picker cell
ELSEIF nAction{nCameraNumber}=PICK_INTERLAYER THEN
  ! TODO - Do something to take care of interlayer, can be removed
    if not a pallet picker cell
ELSEIF nAction{nCameraNumber}=PICK_INTERLAYER_AND_CHANGE_BIN THEN
  ! TODO - Do something to take care of interlayer AND pallet
    change, can be removed if not a pallet picker cell
ELSE
  PickCam_1;
  RefPosOutCam_1;
```

Continues on next page

```
    ! start process of post-pick processing
    MoveRobotToCamera2;
    RefPosInCam_2;
    GrabImage CAMERA_NO_2;
    WaitUntil bCoordReceived{CAMERA_NO_2}=TRUE;
    SetNextTarget CAMERA_NO_2;

    ! handle that parts occasionally are not found.
    WHILE nAction{CAMERA_NO_2} <= NO_DETAIL DO
      ! change camera settings if necessary
      DiscardAndTakeNewImage CAMERA_NO_2;
      WaitUntil bCoordReceived{CAMERA_NO_2}=TRUE;
      SetNextTarget CAMERA_NO_2;
      Incr nImageRetries{CAMERA_NO_2};
     IF nImageRetries{CAMERA_NO_1} > MAX_IMAGE_RETRIES{CAMERA_NO_1}
       THEN
        ! Create alarm or inform user in some other way
        EXIT;
      ENDIF
    ENDWHILE

    ! handle parts according to results from vision data
    RefPosOutCam_2;
    %"Cam"+ValToStr(CAMERA_NO_1)+"Position_"+ValToStr(nPosition)%;
  ENDIF
```

Multiple robot with camera handling

If the robot controller has several attached robots which should interact with FlexLoader Vision there are some additional settings and variables that needs to be handled. First, FlexLoader Vision will load Main.pgf to T_ROB1 just as normal. If there is a second robot task and there is a MainRob2.pgf placed in robot HOME folder, then it is loaded to T_ROB2. In the same way MainRob3.pgf is loaded to T_ROB3 if it exists and MainRob4.pgf to T_ROB4 if that task exists.

Some communication between robot and FlexLoader Vision is general and should be communicated once for one robot controller. Most of the communication is done per camera and is communicated to the robot task that is in control of the specific camera. To handle all this there are a couple of variables that must be set correctly. Normally those are set in InitializeMain.

```
! Communication to FlexLoader Vision, the other robots are setup
    with TRUE for this value.
bIsSlaveVisionRobot:=FALSE;

! TODO - Set cameras to use in this robot task. Risk for reference
    errors if accepting cameras that is not there in reality.
nStartAcceptCamera:=1;
nStopAcceptCamera:=1;
```

In the example above, this robot task has set TASK PERS bIsSlaveVisionRobot to FALSE. This means that all general communication that is needed is handled by

Continues on next page

this task. It's not so important which robot task is Master as long as it is only one task. The other robot tasks set this value to TRUE to make them slaves.

The other variables define which camera or cameras this robot should handle. Note that it must be in order, robot 1 could handle camera 1-2 but not camera 1 and 3. Be careful when deciding camera numbers!

Normally there is one PartCamX.mod module for each camera. This module is selected during teach-in in the mechanical tab of FlexLoader Vision and later on loaded into the robot task that controls that camera. See example below.

```
! Code in InitializeMain in robot task T_ROB2
bIsSlaveVisionRobot:=TRUE;
! Assign camera 3-4 to robot 2.
nStartAcceptCamera:=3;
nStopAcceptCamera:=4;

! Code in main program of T_ROB2
GrabImage CAMERA_NO_3;
! ...normal handling to of receiving coordinates
! Then call to pick part, code in the PartCam3.mod in T_ROB2.
PickCam_3;
```

At start up FlexLoader Vision and RAPID code will co-work to make the PartCam3.mod and PartCam4.mod load into T_ROB2 task according to the variable setting.

Single camera feeder with multiple coordinate transfer for multiple picking

In certain applications, the robot is expected to pick more than one part in one pick cycle. This can reduce cycle times.

FlexLoader Vision needs to be configured for multiple coordinate transfer (see FlexLoader Vision ConfigurationEditor).

The main code uses following instructions to handle this case.

```
VAR num nPartsToPick;
! This call will make the robot pick 3 parts from the same image,
     first and second pick must be a part found in position 1 and
     third pick must be found in position 2.
! If this combination is not found vision system will try a new
     image twice and then give up.
nPartsToPick:=PickMultiPartsAtCamera(CAMERA_NO_1,
     \NumberOfPartsToPick:=3,
     \DesiredPositions:=[1,1,2],\MaxImageRetries:=2);
```

In the part module file there must be a special pick routine named as in example below. It should handle how to pick the different parts.

```
PROC PickMultiPartsCam_1(num PartNo,num Position)
  VAR dnum dnGripper;
  ! Switch off configuration control to be sure to reach the
     position, those will switch on in MoveFromCamera1.
  ConfJ\Off;
  ConfL\Off;
  ! TODO - set correct gripper to use depending on which part to
     pick
```

Continues on next page

```
 TEST PartNo
 CASE 1:
  ! TODO - Set correct gripper to grab first part from vision
   result
  tPickingGripper:=tGripper1;
  dnGripper:=GRIPPER_1;
 CASE 2:
  ! TODO - Set correct gripper to grab second part from vision
   result
  tPickingGripper:=tGripper2;
  dnGripper:=GRIPPER_2;
 CASE 3:
  ! TODO - Set correct gripper to grab third part from vision
   result
  tPickingGripper:=tGripper3;
  dnGripper:=GRIPPER_3;
 ENDTEST
 ! Update grip load based on currently used gripper
 GripLoadUpdate dnGripper;
 ! Set correct data in part tracker for feeder in
 MT_SetPartInfo\Station:=sdFeederIn,\ProgCode:=dnProgCodePartCam1,
   \State:=psRAW,\UserDef:="PickNo: " +ValToStr(PartNo)+" Pos:
   "+ ValToStr(Position);
 MoveJ RelTool(VisionResult{CAMERA_NO_1}.Target,0,0,-nPZ),v1000,
   z10,tPickingGripper\WObj:=wCamera1;
 MoveL VisionResult{CAMERA_NO_1}.Target,v200,fine,
   tPickingGripper\WObj:=wCamera1;
 ! TODO - Close gripper to pick part
 ! TODO - If needed use your own solution to keep track of what's
   held in gripper
 PickPartAt sdFeederIn,dnGripper;
 MT_Log MT_LVL_INFO,"PickMultiPartsCam_1",["Picked part no: ",
   ValToStr(PartNo)," pos: ",ValToStr(Position)];
 ! TODO - If several parts is picked, the VisionResult will be
   overwritten between each one.
 ! If the result for each part is needed later, store it somewhere!
 MoveL Offs(VisionResult{CAMERA_NO_1}.Target,0,0,nPZ),v1000,z10,
   tPickingGripper\WObj:=wCamera1;
ENDPROC
```

RAPID example PartCam1.mod

The provided detail specific application code offers various possibilities.

Generic teachin – Everything is part specific

Application code for a general teachin.

Robot tasks as picking, machine tending, or leaving on the outbelt, have to be programmed manually. A skeleton is provided, at location HOME:\PartModules\PartCam1.mod.

However, no general robot program can be provided for unknown parts, so the skeleton has to be adapted to each new part. Positions, movements and option

Continues on next page

handling have to be modified or generated for each new part. This is the most general and open approach to machine tending.

Note

These routines are provided as example code. Always ensure that they are working with the actual parts and processes.

Programming of e.g. Cam1Position_1, Cam1Position_2, Cam2Position_1... for all used cameras and positions must be done if those routines should be used.

RAPID example for Initialization

General initialization that is not part specific is placed in the InitializeMain in the MainModule. This includes the following variables which will affect the functionality of how and when images are grabbed.

```
! Belt setting: setup how belt should be controlled
nBeltAction{CAMERA_NO_1}:=RUN_ONE_DETAIL;

! Camera setting: grab images automatically or only from user
     program
bAllowAutoGrab{CAMERA_NO_1}:=TRUE;

! Set image grab delay (after stop at sensor)
nImageGrabDelay{CAMERA_NO_1}:=0.2;
```

Initialization that is part specific is placed in the PartCam1 module (if part is run at camera 1) in the InitializeCam1. This could include part dimensions, weight and which options should be used for the part.

```
! Options - Set value TRUE for the options that should be used for
     this part
bUseAirClean:=FALSE;
bUseMarking:=FALSE;
bUseWash:=FALSE;
bUseSamplePart:=FALSE;
bUseDeburr:=FALSE;
bUseReGrip:=FALSE;
bUseTurnStation:=FALSE;

! Initiate options (those which need initialization)
IF (bHasMarking=TRUE AND bUseMarking=TRUE) SetMarker
     sMarkingFontSize,sMarkingText;

! --------------------------------------------
! Weight of part [kg]
! ------------\ /----------------------------
nWeight_PART:=1;

! --------------------------------------------
! Distance [mm] from TCP to center of gravity in Z direction of
     tool for a part
! -----------------\ /------------------------
nCOGFromTCP_PART:=30;
```

Continues on next page

RAPID example MT_MoveRobotTo

The principal motion in the FlexLoader Vision is controlled by the MT_MoveRobotTo routine, which moves the robot from known zone positions to a target zone position. From these zone positions (via positions), movement within certain areas of the FlexLoader Vision can be initiated.

For detailed reference on zone and via positions, see *MoveRobotVia.mod on page 317*

```
! order robot to another zone
MT_MoveRobotTo ZONE_OUTBELT;
```

CAUTION

The MT_MoveRobotTo routine must be verified for all parts and grippers.

When zones, via positions, parts, grippers or any other conditions are changed, the integrator has to ensure that the MT_MoveRobotTo routine can execute without risk for collisions. The MT_MoveRobotTo routine is by default general. But due to variations it might need to be made part specific which is possible to handle within a project.

When starting the robot from unknown positions, collisions can occur.

14.3.3 FlexLoader Vision interface

Overview

The FlexLoader Vision interface code handles the interface and the communication to and from FlexLoader Vision. No changes should be made to this code.

The basic application related use of the FlexLoader Vision interface is described above, see *FlexLoader application functionality on page 231*. This section describes additional functionality that can be used for a large variety of purposes.

For further reference, see *FlexLoader Vision interface on page 306*.

RAPID examples using Vision.sys

Camera exposure time, contrast and gain can be controlled from the robot.

```
! set contrast for camera 1 to 50%
CameraContrast CAMERA_NO_1, 50;

! set exposure time for camera 2 to 10%
CameraExposureTime CAMERA_NO_2, 10;

! set gain for camera 3 to 30%
CameraContrast CAMERA_NO_3, 30;
```

Positions for the currently used part can be switched on and off, if required by the application.

```
! enable all positions for camera 1
EnablePosition CAMERA_NO_1, 1000;
! disable position 1 for camera 1
DisablePosition CAMERA_NO_1, 1;

! disable all positions for camera 1
DisablePosition CAMERA_NO_1, 1000;
! enable position 1 for camera 1
EnablePosition CAMERA_NO_1, 1;
```

Set alarm, warning and information in FlexLoader Vision alarm list.

```
! Set an alarm
SetAlarm "This is an alarm text";
! Set a warning
SetWarning "This is a warning text";
!Set an information message
SetInformation "This is an information";

!Reset the alarm
ResetAlarm "This is an alarm text";
! Reset the warning
ResetWarning "This is a warning text";
!Reset the information message
ResetInformation "This is an information";
```

Continues on next page

There is a lot of more functions, look into the Vision.sys module and the routine comments for more information.

Limitations

Note

Please note the following system limitation regarding robot system events.

It is not possible to use elements of the FlexLoader Vision interface in system event routines (e.g. STOP, QUICKSTOP, START).

Do not use of e.g. master/slave functionality, alarms handling, initiating image acquisition or setting other parameters in such system event RAPID code. Non-compliance may result in undesired system behavior.

Note

Please note that some code relies on predefined I/O signal names. If any of those signal names is renamed the code can't execute as desired. If a signal name should change always search the code and make sure it is possible first.

14.3.4 FlexLoader Vision Lite functionality

Overview

The application code for FlexLoader Vision Lite provides an extension to the general teachin described previously.

This extension is used for parameter-based simplified teachin for upright standing cylindrical details, the so called FlexLoader Vision Lite. Normally, this code needs no changes.

For further reference, see *FlexLoader Vision Lite functionality on page 320*

The RAPID code follows the same structure as described in *FlexLoader application functionality on page 231*. It contains a preconfigured MainModule.mod, preconfigured machine tool sequences, and a parameter interface to FlexLoader Vision.

RAPID example LitePartCam1.mod

A specialized LitePartCam1.mod contains the application code for parameter-based simplified teachin for upright standing cylindrical parts (i.e. FlexLoader Vision Lite). A generic robot program based on geometrical part data is executed. No robot programming is necessary for different parts, and teachin is reduced to entering a few geometrical parameters in FlexLoader Vision. But note that some work objects and configurations needs to be handled.

The module contains parametric movement instructions, option handling and machine tool handling.

In this case, the LitePartCam1.mod is used for all parts, and only parametrized code can be executed. It is located in HOME\FlexLoaderVisionLite\.

Note

The handling routines are called for each part. Make sure your code works with all parts. If not, activate the special robot program option in FlexLoader Vision.

Special robot program option for part specific tasks

A part specific PartCam1.mod can be used and stored at a different location. Option handling, e.g. deburring, air cleaning or marking, can be handled on part level.

RAPID example for Initialization

Belt handling and grab setting is done in InitializeMain. Note that this is an example, other settings for those are also possible.

```
! Belt setting: setup how belt should be controlled
nBeltAction{CAMERA_NO_1}:=RUN_ONE_DETAIL;

! Camera setting: grab images automatically or only from user
    program
bAllowAutoGrab{CAMERA_NO_1}:=TRUE;

! Set image grab delay (after stop at sensor)
nImageGrabDelay{CAMERA_NO_1}:=0.2;
```

Continues on next page

Other part specific initialization could include option activation, outbelt setup, part data and load and unload positions. Those type of initialization is placed in InitializeCam1.

```
! Set up outbelt, see manual for documentation about all arguments.
SetUpFeederOutBelt FEEDER_WIDTH,nFeederOutSpeed,SPACE_BETWEEN_PARTS,
     SPACE_TO_EDGE,Y_DISTANCE_TO_LEAVE;
SetUpFeederOutDetail maxNumValue(nFinishedDetailDiameter,
     nFinishedDetailOuterDiameter),maxNumValue(nFinishedDetailDiameter,
     nFinishedDetailOuterDiameter),nGripHeightFinishedDetail,
     \allowedDetailsInHeight:=getAllowedNumberToStack(),\detailHeight:=
     nFinishedDetailHeight,\hasLeftOverPart:=bIsThereLeftOverPart,
     \leftOverPartWidth:=nLeftOverPartDiameter,\leftOverPartLength:=
     nLeftOverPartDiameter,\leftOverPartHeightToTCP:=nGripHeightLeftOverPart;

! Setup possible robot part loads for this detail program
lPartLite:=[nRawDetailWeight,[0,0,nGripHeightRawDetail-
     (nRawDetailHeight/2)],[1,0,0,0],0,0,0];

! Calculations for positions
pLoadMachine1_MAIN.trans:=calcPosLoadMachine();
pLoadMachine1_SUB.trans:=calcPosLoadMachine(\nSpindleOffset:=
     nUnloadSubSpindleOffset);
pUnLoadMachine1_MAIN.trans:=calcPosUnloadMachine();
pUnLoadMachine1_SUB.trans:=calcPosUnloadMachine(\nSpindleOffset:=
     nUnloadSubSpindleOffset);
pUnLoadLeftOverPart_MAIN.trans:=calcPosUnloadLeftOverPart();
pUnLoadLeftOverPart_SUB.trans:=calcPosUnloadLeftOverPart
     (\nSpindleOffset:=nUnloadSubSpindleOffset);
```

Cell emptying

In some cases of machine tending, the cell must be emptied from parts, i.e. the machine shall be unloaded without loading the next part.

For FlexLoader Vision Lite, this behavior is pre-programmed. By setting the output DOF_EmptyCell to “1”, the robot will finish a cycle by unloading a finished part without loading the next raw part.

The mechanism for changing the signal state (external signal, button, FlexPendant, ...) must be decided and implemented by the integrator.

14.3.5 FlexLoader Library Add-in

Overview

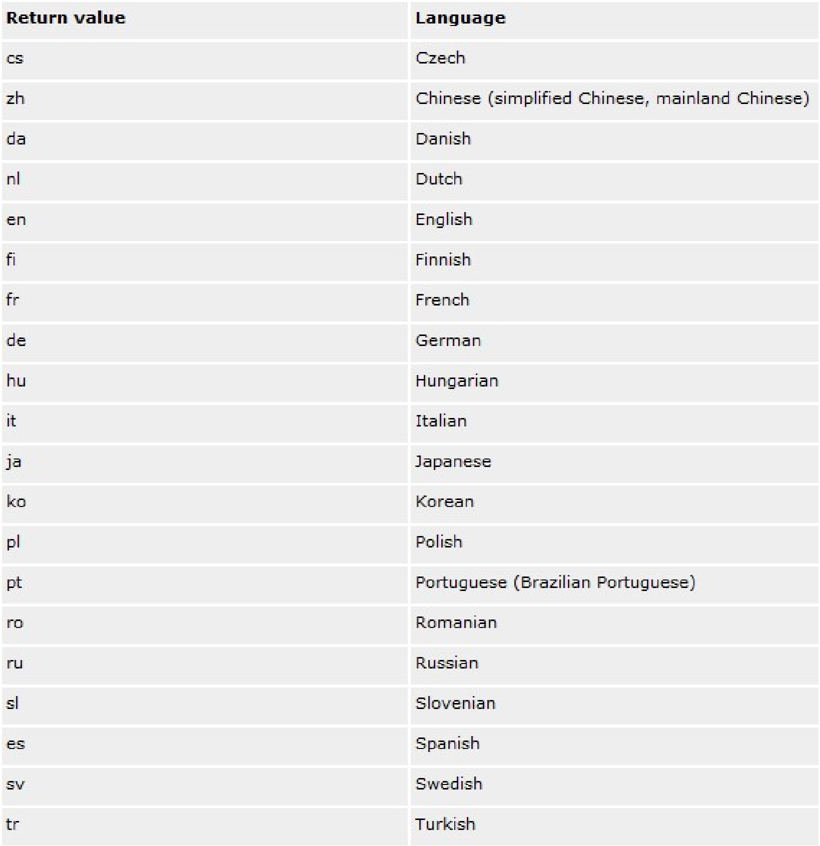
The FlexLoader Library Add-in package includes code that is installed into the RobotWare and hidden to end user. The following sections describe the different FlexLoader Library Add-in code blocks in short. For more information of all included instructions, functions and data types, see appendix.

Alarms & messages

Alarms & Messages helps user to create alarms, warnings, informations and other messages shown to the user in the local language. At robot start up, all texts will be read into the system from the source files holding texts of the currently used language in the Flexpendant.

The source files for English are found under HOME\Language\en. If the FlexPendant language is changed to Swedish, the files are expected to be in the folder HOME\Language\sv. If the standard text files don't exist for the selected language, a translation must be done and new files must be created.

Possible languages are the same as the languages that can be used for the FlexPendant itself. The table below shows selectable languages and return values that will be used in the robot code for each language. These return values should also be the name on the language folder to allow RAPID code to use that language.



Return value	Language
cs	Czech
zh	Chinese (simplified Chinese, mainland Chinese)
da	Danish
nl	Dutch
en	English
fi	Finnish
fr	French
de	German
hu	Hungarian
it	Italian
ja	Japanese
ko	Korean
pl	Polish
pt	Portuguese (Brazilian Portuguese)
ro	Romanian
ru	Russian
sl	Slovenian
es	Spanish
sv	Swedish
tr	Turkish

xx1900001037

Continues on next page

To create a new text in the cell, add a row to an existing file or create a new csv file. There is also a possibility to add project specific alarms directly from code using RAPID instruction MT_CreateAlarm. All csv files in the selected language folder will be read into the robot. Inside the csv file there are 7 columns to be filled to define the alarm or message.

Element	Description
Identifier	Unique string through the system. This is later used in routine calls to point to this message.
Category	“Error” = this is an alarm, “Warning” = this is a warning, “Information” = this is an information. Leave this blank for other messages used in input boxes and so on. If defined as any of the three this message will show in event log and alarm lists.
Station	If wanted, name the station to which this message is connected. Leave blank if not needed.
Number	If wanted, give an alarm number. This will be included in the alarm number showing in alarm lists.
Domain	If wanted, group texts and alarms in domains. In that case, give the alarm a domain number here.
Header	Header for the message. Depending on where you see this message, the header might be the only line you see.
Text	Body text for the message. Possible to give a more detailed description of the message.

In the Header and Text wild cards can be used. A wild card is a place holder that later can be changed to wanted text through RAPID code. For example Header: “Emergency stop {1} machine number {2}”.

The placeholder numbers can be replaced by customer defined text when RAPID calls are made in the program. If the example has the identifier "EMStop" then a call in the program might look like this:

```
! Creating alarm: Emergency stop from machine number 3
MT_SetAlarm "EMStop",\WildCardTexts:=["from",ValToStr(3)];
```

The text shown in alarm list will be “Emergency stop from machine number 3”.

A maximum number of 5 wild cards per message is possible, which can be used freely in both Header and Text.

Example of tasks that are handled by the alarms & messages are setting and resetting of alarms, keeping track of active alarms and language handled UI instructions. Most of the RAPID standard UI instructions exists here with the MT_ prefix. They handle the same arguments as the standard instructions, only difference is they fetch the texts to show from the localized text handling. Read more in the documentation for each specific instruction.

Predefined data

There is some predefined data in the alarms & messages module that could be good to know about.

```
MT_nMsgReadFromCSV
```

This variable will be set to 1 during the time that csv file information is read and stored into the system. This could take some time and the alarms & messages system will not work as intended during this time. If needed, make sure this variable

Continues on next page

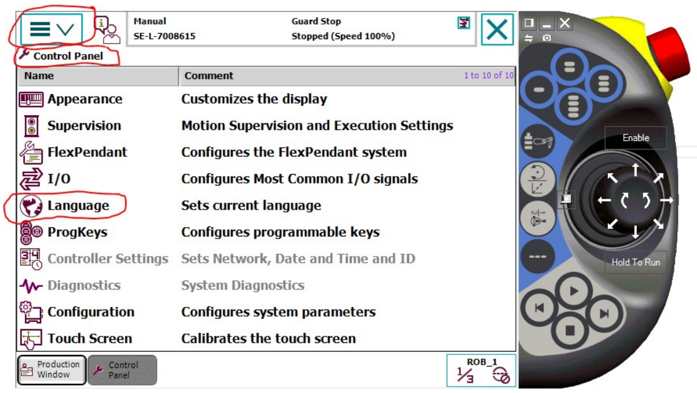
is set to 0 before starting normal execution and messaging usage in your user program.

```
MT_bUIDialogActive
```

This variable will be TRUE as long as there is any type of user question/input box active on the FlexPendant (any active UI instruction). This means that as long as this is TRUE robot is waiting for some kind of user input through the Flexpendant. This could be used to indicate for the user that input is needed.

Language selection

To select a new language select the main menu in top left corner of the FlexPendant screen. Then select **Control Panel** and then **Language**, see image below.



Name	Comment
Appearance	Customizes the display
Supervision	Motion Supervision and Execution Settings
FlexPendant	Configures the FlexPendant system
I/O	Configures Most Common I/O signals
Language	Sets current language
ProgKeys	Configures programmable keys
Controller Settings	Sets Network, Date and Time and ID
Diagnostics	System Diagnostics
Configuration	Configures system parameters
Touch Screen	Calibrates the touch screen

xx1900001039

Find your desired language in the list and press **Ok**. The change will become active after a restart and you will be asked to restart directly when you choose a new language.

When system is restarted the FlexPendant language has changed. The new text file is also read upon restart and your new application text will also be updated now.

Note that if the correct language file for the application texts was not found the system will search for and use the English file instead. There will be a log in the FlexPendant of that in the case this happens.

Stations

Stations enable a good object-oriented structure of the cell. They open possibilities to visualize station level information in a user-friendly way on a station level in the user interface.

Most modules or objects in a cell can be defined as a station. The station concept is especially useful for objects that perform a certain task or function or could hold a part. Typical examples are machine tools, buffer stations, in- and out-feeders and testing units.

Continues on next page

The part tracker concept, described in another section, depends on stations. Each station can include up to a maximum of 20 part trackers. Everywhere the robot can store parts or does work with a part could be a good candidate for a station.

> **Tip**
>
> If a buffer could hold more than 20 parts it is probably a good idea to create two different stations to be able to create part trackers for all the positions at the buffer.

Part trackers

Part trackers are used to keep track of where parts are located in the cell. Part trackers can also carry some specific data for each part, for example a part state like raw, machined, or deburred. It's also possible to assign user data for a part that could hold an index number or some other kind of data specific for to part. In addition to this it is also possible to create part property data that connects to a part of a specific state. These properties will then be used by the tracker routines to present part property data like loaddata, dimensions and so on.

Each station could have part trackers connected to it. User can decide to define 1 to 20 separate trackers, or part positions, for each station. A big fixture table for example could hold 20 parts and need all the available trackers. If there is need of even more, then it is of course possible to create a second station for this to handle an additional 20 trackers.

The part tracker concept has routines that helps user to move the part tracker data between the stations. Current data held by a tracker could be useful in the robot program to perform correct action. It could also be a good idea to show some of the information on a operator interface to give the operator a better understanding of what is going on in the cell.

To get things started, the parts need to be coming into the cell somewhere and the code needs to create a new part tracker data which is connected to this in-feeder station. After this, the part data could be moved around in the cell. It's a good idea to also view the robot as a station with potentially several tracker positions depending on the number of robot grippers.

If using the part property data, there is some rules that must be followed. When a part is added to a tracker it's name is passed as a string to the tracker. If using the properties then a new array of type mtpartproperties should be defined somewhere in the user program. This must be named exactly as the part name with the additional ending "_Prop". The length of this array should be the number of possible states possible for the part. And each array element describes the properties which are valid for a specific state.

For more detailed information about the part tracker instructions, see the RAPID reference.

Continues on next page

MoveVia

The MoveVia functionality helps to move the robot between different stations in the cell. Normally, in every cell there is one robot position in front of each station where robot is doing some kind of work.

Those points in front of the different stations in the cell are used when navigating between the different stations and are defined by the user and packaged into a mttargetdata. This data is defined in the user program and gives the MoveVia more input to find which station is the closest when the robot is started in an undefined position.

In the user main program the normal instruction calls will be to MT_MoveRobotTo to which the mttargetdata of the target position is passed, no matter where the robot is positioned right know. But to have a safe operation robot should already be positioned in one of the via positions (in front of a station) when calling the MT_MoveRobotTo. If robot is in zone but not at it's via position there is an optional argument called CheckStart that could be used which will also call the routine MT_MoveToVia first. Note that this routine needs to be filled by user, if there is any special movement to be done to navigate out to the via position, it's up to the user to create that. In fact all real movements have to be created by the user. But MoveVia concept will out to reuse code and get a good interface to the main program.

Find the MoveRobotVia example module to get a good picture of how to use this functionality. One key is to define all mvXXXXX routines where XXXXX is the name of each defined mttargetdata. These mv-instructions should move the robot to that corresponding target or zone.

Example

A small project might define these zones: HOME, MACHINE and FEEDER. Within the MoveVia concept you define in the user program a position and a mttargetdata:

```
PERS robtarget pHome:=[[400,0,600], .....
PERS robtarget pViaMachine:=[[-1500,300,1000], .....
PERS robtarget pViaFeeder:=[[1500,300,1000], .....

CONST mttargetdata ZONE_HOME:=[1,"pHome",[-80,70], .....
CONST mttargetdata ZONE_MACHINE:=[2,"pViaMachine",[-100,-80], .....
CONST mttargetdata ZONE_FEEDER:=[5,"pViaFeeder",[-100,170], .....
```

Note that in each mttargetdata, there is an indirect pointer to the actual robtarget defined by a string value. Furthermore, a couple of routines should be created:

```
PROC MT_MoveToVia(mttargetdata Target)
  ! In this routine add code to move to via position (position
     connected to the targetdata) of the zone. This will be called
     if robot is somewhere in the zone but not at a defined
     position and needs to come back. The easiest way could be to
     always go straight to that position:
  MoveJ MT_GetViaPosition(target),v500,fine,tTool0;

  ! But in some cases that might not be possible and there it could
     be needed to add some additional smart code to navigate to a
     free area before moving directly to the via position. This
```

Continues on next page

```
     added code must be created by the integrator who have
     knowledge of the specific cell.
ENDPROC
```

In the mv-instructions define how to navigate from any other position to this position/zone. Note that the routine name must be mv followed by the exact name of the corresponding mttargetdata. mvZONE_HOME could in this example look like this. Note that there also must be similar routines for mvZONE_MACHINE and mvZONE_FEEDER too.

```
PROC mvZONE_HOME()
  VAR mttargetdata FromZone;

  FromZone:=MT_GetCurrentZone();
  TEST FromZone
  CASE ZONE_MACHINE:
    !TODO - add movement instructions here
  CASE ZONE_FEEDER:
    !TODO - add movement instructions here
  DEFAULT:
    !No path available
    !TODO - error handling in a way that fits your project
    TPWrite "Movement to ZONE_FEEDLINE from this position is not
     defined";
    Stop;
    EXIT;
  ENDTEST
  MoveJ pHome,v7000,z200,tTool0;
ENDPROC
```

Predefined data

The MT_SAFE_START_DISTANCE is default 100 and is used internally as the safe start distance. This means that the robot is free to start without any extra user warning or anything if it is within this distance to a via position.

```
CONST num MT_SAFE_START_DISTANCE:=100;
```

For more detailed information about the instructions, see RAPID references.

Logging

Sometimes it’s a good idea to create some logs to keep track of what is happening in the robot code. This module gives the possibility to keep some order by having different log levels that can be enabled and disabled independently.

The log messages are written to a csv file on a USB stick that has to be inserted to the robot controller computer before starting to log. If no USB memory is inserted, user will be notified of this.

Note

The USB port in the FlexPendant is not usable for this logging purpose.

Use the USB ports on the main computer unit or robot cabinet instead.

Continues on next page

Note

Alarm will be created when USB is unplugged during logging. But there is any problems with the USB stick when reinserting it, user might not get any indication of this!

A log message is created with a call to MT_Log and will include date and time, robot task, log level, source and message. Each log will occupy one row in the log file. When the log file is over 7.6MB the system will create a new file and continue in that file. And if more than 100 files have been used the system starts to remove the oldest when a new files is to be created. This is to make sure the latest log history is always there even if the log area is full. Note that calls to MT_Log could left in the program, if logging is disable those calls will have no effect at all. Logs will only be written if the log level used in the call to MT_Log has been enabled.

For more detailed information about the instructions, see RAPID references.

14.3.6 FlexLoader assistance and utility functionality

Overview

Assistance and utility functionality enables efficient and user friendly programming of the robot cell. It provides means for tool handling, feeder handling, entry control, indication lights and several global utility functions. Some of these function packages like options and feeder handling will be included in the RAPID code package if those kind of equipment exist in the cell.

For further reference, see *FlexLoader assistance and utility functionality on page 322*.

RAPID example tool handling

The following example is typical use of the tool handling system.

```
! Call to define all robot tools in cell. This call is normally
    done in the PowerOn event.
SetupTools;

! Example of how SetupTool could look like. This is placed in
    CommonXXXX.sys
PROC SetupTools()
  MT_Log MT_LVL_INFO,"CommonCell",["Performing Tool setup"];
  ! General tool data setup
  tGripper0:=tool0;
  !----- Define gripper data -----------
  !!TODO - set correct values for you grippers tool data, then
      remove alarm row below
  !tGripper1:=
  !tGripper2:=
  MT_SetAlarm "tool_NoToolData";
  !---------------------------------------
  !Modify cog and mass for tGripper0, default to use data for
      tGripper1
  !!TODO - If necessary modify those cog and mass data to something
      that fits even better for tGripper0
  tGripper0.tload.cog:=tGripper1.tload.cog;
  tGripper0.tload.mass:=tGripper1.tload.mass;
  !!TODO - modify gripper setups to fit the current cell. Read
      comments in ToolSetup for more information about this.
  ClearToolData;
  ToolSetup GRIPPER_0,"Gripper 0",tGripper0,sdRobot,\StnIndex:=10;
  ToolSetup GRIPPER_1,MT_GetText("msgGripper1"),tGripper1,sdRobot,
      \StnIndex:=1,\OpenOrOnSignal:="DOF_OpenGripper1",
      \WaitTimeOpenOrOn:=0.5,\LabelOpenOrOn:=MT_GetText("msgOpened"),
      \WaitTimeCloseOrOff:=0.5,\LabelCloseOrOff:=MT_GetText("msgClosed");
  ToolSetup GRIPPER_2,MT_GetText("msgGripper2"),tGripper2,sdRobot,
      \StnIndex:=2,\OpenOrOnSignal:="DOF_OpenGripper2",\WaitTimeOpenOrOn:=0.5,
      \LabelOpenOrOn:=MT_GetText("msgOpened"),\WaitTimeCloseOrOff:=0.5,
      \LabelCloseOrOff:=MT_GetText("msgClosed");
  ToolSetup COMPRESSED_AIR_1,MT_GetText("msgCompressedAir1"),
      tGripper0,sdRobot,\StnIndex:=3,\OpenOrOnSignal:="doValve5Gripper",
      \WaitTimeOpenOrOn:=0.5,\LabelOpenOrOn:=MT_GetText("msgOn"),
      \WaitTimeCloseOrOff:=0.5,\LabelCloseOrOff:=MT_GetText("msgOff");
```

Continues on next page

```
ENDPROC

!Sets correct gripload depending on all currently held parts (in
     all grippers). This should normally be called every time the
     robot is about to start working with another gripper (ie.
     gripper 1, gripper 2, tool0 and so on)
GripLoadUpdate GRIPPER_1;

!Makes sure gripper is open before going in to pick
ToolAction\Release,GRIPPER_1,\InsideGrip:=bInsideGrip;

!Pick part by using standard function that handles both picking
     and moving part tracker information in one
!Picks the part with gripper 1 from feeder in (includes moving of
     tracker data from feeder in to robot gripper). Uses inside
     grip if boolean value is set.
PickPartAt sdFeederIn,GRIPPER_1,\InsideGrip:=bInsideGrip;

!How to check what is in a gripper and act on that information
IF getToolPartState(GRIPPER_1)=psRAW THEN
  !use regrip if needed
  IF (bHasRegrip=TRUE AND bUseRegrip=TRUE) RegripDetail;
  !Load machine
  LoadMachine;
ENDIF
```

All tools that are defined through the tool handling system can be access through the tool control menu. This menu is used in manual mode to open, close, turn on or off grippers for example if a part should be released.

RAPID example feeder handling

Feeder handling includes in-feeder control in cases no PLC is taking care of it and some help functions to handle leaving and running the out belt. It also includes additional functions to monitor if camera belt has been in manual mode since this might indicate someone entered the cell and might have touched parts which was already found and prepared for robot picking.

RAPID example entry control

Entry requests are handled in a background task. The main task uses the CheckSystem routine to test and react to entry and stop requests.

RAPID example indication lights

The indication light is controlled automatically through the user messaging system. The indication lamps are switched on and off depending on the current alarm, warning, question and information status. Indication light control is executed in a background task.

Conditions and indication light colors are defined during program initialization. A color is assigned to all conditions, and the integrator selects flashing or static lights. In most cases RGB lighting is used.

```
! predefined colors: RED, GREEN, BLUE, WHITE, YELLOW
```

Continues on next page

```
IndicationLightsSetupRGBLights \AlarmFlashingColor:=RED
     \CycleOnColor:=GREEN \InformationFlashingColor:=WHITE
     \QuestionFlashingColor:=BLUE \WarningFlashingColor:=YELLOW;

! static light instead
IndicationLightsSetupRGBLights \AlarmStaticColor=RED
     \CycleOnColor:=GREEN \InformationStaticColor=WHITE
     \QuestionStaticColor=BLUE \WarningStaticColor=YELLOW;
```

Sometimes a traditional light tower with discrete signals for every color is used. In this case, a different setup can be used:

```
! predefined lamps: BLUE_LAMP, GREEN_LAMP, ORANGE_LAMP,
! RED_LAMP, WHITE_LAMP, YELLOW_LAMP
IndicationLightsSetupLightTower \AlarmStaticColor:=RED
     \CycleOnColor:=GREEN \QuestionStaticColor:=BLUE
     \WarningStaticColor:=YELLOW;
```

If needed, the lighting can be switched on by integrator RAPID code by using dedicated signals.

```
! force alarm indication on
Set DOF_Alarm;
! switch back to automatic control
Reset DOF_Alarm;

! force warning indication on
Set DOF_Warning;
! switch back to automatic control
Reset DOF_Warning;

! force information indication on
Set DOF_Information;
! switch back to automatic control
Reset DOF_Information;

! force question indication on
Set DOF_Question;
! switch back to automatic control
Reset DOF_Question;
```

Light indications should always be accompanied with user messages for explanation.

RAPID example global utility functions

Look into the module GlobalCodeAndConfig and read routine header comments to see some of the utility routines that could be good to have. This module is loaded as shared which makes those routines accessible through all tasks.

14.3.7 FlexLoader machine tool interface functionality

Overview

The FlexLoader Vision standard modules communicate with the machine tool by means of the interface module TemplateMachine_Lathe.sys or TemplateMachine_CNC.sys, which must be modified and adapted to the actual machine tool by the integrator. Those modules are template modules that describes how the machine communication should be handled.

Read the module and routine comments, in order to add machine tool specific code where needed.

For further reference, see *FlexLoader machine tool interface functionality on page 329*

States and actions

This module contains a number of predefined states and actions that are consistently used throughout the FlexLoader Vision.

For further reference, see *FlexLoader machine tool interface functionality on page 329*

RAPID example

Depending on the type of the machine tool and the type of application, a variety of communication flows are possible.

A typical machine tool communication flow for a load-unload sequence as realized in the standard setup could be as follows. Note that no movement is included in this schematic code.

```
! Prepare for machine tool interaction
Machine1Action INIT_MACHINE;
WaitUntil Machine1Check(LOAD_MAIN_OK)=TRUE;
Machine1Action OPEN_DOOR;
WaitUntil Machine1Wait(DOOR_OPENED,10);

! Enter machine tool and load part in main chuck
Machine1Action CLOSE_CHUCK_MAIN;
! grip detail in sub chuck
Machine1Action OPEN_CHUCK_SUB;

! leave machine
Machine1Action CLOSE_DOOR;
Machine1Action CYCLE_START;
```

Different machine tools have different behavior. As described in the previous section, FlexLoader Vision supplies a comprehensive set of tools and adjustment possibilities to handle these behaviors.

CAUTION

All predefined states and actions that are used in the template module shall be implemented or deleted. This minimizes the risk of unexpected machine behavior.

Continues on next page

An example implementation of the routine Machine1Action is shown below.

```
PROC Machine1Action(string sAction)
  TEST sAction
    CASE CLOSE_CHUCK_MAIN:
      ! TODO - Send signal to machine
      Set doMachineSpare01;
      WaitDI diMachineSpare06,1;
      Reset doMachineSpare01;
    CASE CLOSE_CHUCK_SUB:
      ! TODO - Send signal to machine
      Set doMachineSpare05;
      WaitDI diMachineSpare08,1;
      Reset doMachineSpare05;
    CASE OPEN_CHUCK_MAIN:
      ! TODO - Send signal to machine
      Set doMachineSpare02;
      WaitDI diMachineSpare05,1;
      Reset doMachineSpare02;
    CASE OPEN_CHUCK_SUB:
      ! TODO - Send signal to machine
      Set doMachineSpare03;
      WaitDI diMachineSpare07,1;
      Reset doMachineSpare03;
    CASE CLOSE_DOOR:
      ! TODO - Send signal to machine
    CASE PREPARE_LOAD_MAIN:
      ! TODO - Send signal to machine
      WaitUntil Machine1Wait(MAIN_CHUCK_OPENED,30)=TRUE;
    CASE PREPARE_UNLOAD_MAIN:
      ! TODO - Send signal to machine
    CASE PREPARE_UNLOAD_SUB:
      ! TODO - Send signal to machine
    CASE OPEN_DOOR:
      ! TODO - Send signal to machine
    CASE CYCLE_START:
      ! TODO - Send signal to machine
      PulseDO\High\PLength:=0.5,doMachineSpare04;
    CASE INIT_MACHINE:
      ! TODO - Send signal to machine
    CASE STOP_MACHINE:
      ! TODO - Send signal to machine
    DEFAULT:
      ! TODO - No valid action to perform,
      ! create alarm or do something if needed
  ENDTEST
ENDPROC
```

14.3.8 WorldZones

Some integrators prefer the use of WorldZones to verify the robots presence in certain areas/volumes, and/or to block certain movements. FlexLoader Vision offers predefined WorldZone handling in the module Common.sys.

The FlexLoader Vision defined WorldZones are active in both jogging mode and automatic mode.

The following zone is predefined and must be configured during commissioning.

- forbiddenVolume: A zone that e.g. lies above the volume that the robot needs for movements. Prevents the robot from moving through the forbidden volume.

The following zones are predefined for integrator convenience and can be configured during commissioning.

- safeVolume: The robot could be started from here directly. Connected to DOF_SafeZone.
- machineVolume: Robot is outside of machine tool. Connected to DOF_LoaderOut.

The definition of these WorldZones can be changed by geometrical data defined in Common.sys. After changing these values, the robot controller must be restarted in order to activate the new definitions.

For further details, see *Common.sys on page 316*.

14.4 Communication information

14.4.1 Communication between FlexLoader Vision and the robot

FlexLoader Vision and the robot communicates via PC SDK and Robot WebServices. The information is transmitted between the robot and FlexLoader Vision through direct reading/writing to/from variables and I/O signals.

Operation in a standard system

From FlexLoader Vision	From the Robot	Description
	bGrab bGrab1..4	Take new image, with camera 1..4
x		X-value
y		Y-value
z		Z-value
pZ		Z-value over picking position
rotX		Rotation around x-shaft
rotY		Rotation around y-shaft
rotZ		Rotation around z-shaft
position		Detail position
action		Result information
nAmountOfDetails		Detail quantity (option)
nAmountOfDetails1		Detail amount 1 (option)
nAmountOfDetails2		Detail amount 2 (option)
nAmountOfDetails3		Detail amount 3 (option)
DOF_Coord		Set to coordinates printing
	bCoord	Coordinator received
cam		Camera number

The result information reports how many pickable details there are in the image field: No details (**action**=1), precisely one detail (**action**=2), or more than one detail (**action**=4).

When the option for multi-coordinate gripping is used, additional result information is available: More than one detail, but no new images must be taken after picking (**action**=256).

When a 3D camera is used, the following additional result information is available: Replace pallet (**action**=8), pick interlayers (**action**=16), pick interlayers and replace pallet immediately afterwards (**action**=32).

Continues on next page

Other commands

From FlexLoader Vision	From the robot	Description/result
	bClearSend	Clear the coordinate queue. This should be used in all machine types at each start.
DOF_EndCycle		Stop the program at a suitable point.
	bStop	Stop FlexLoader Vision.
	bPVStatus	Request for operating status of FlexLoader Vision.

nPVStatus		PV_IDLE=1
		PV_OPERATION=2
		PV_STARTING=4
		PV_STOPPING=8

	bConfPickCam1 .. 4	Confirm picking for camera 1, 2, 3 or 4.
	bConfIntPickCam1 .. 4	Confirm picking of interlayers for camera 1, 2, 3 or 4.
	sMessageBoxOk sMessageBoxYesNo	Show message on FlexLoader Vision interface.
sMsgResponse		RESPONSE_OK RESPONSE_YES RESPONSE_NO
	sMsgAnswer sInputBox	Show message (**sInputBox**) on FlexLoader Vision interface and retrieve input from the operator (**sMsgAnswer** as suggested answer).
sMsgResponse		Text entered by the operator.
	nDisablePosCam1 .. 4	Disables the selected position for camera 1, 2, 3 or 4. Disables all positions if the value is > 9999.
	nEnablePosCam1 .. 4	Enables the selected position for camera 1, 2, 3 or 4. Enables all positions if the value is > 9999.
	sBlackRegion	Colors some area of the image field black. Format: **A-BBB-CCC-DDD-EEE-F**. **A** camera, **BBB** StartX (%), **CCC** StopX (%), **DDD** StartY (%), **EEE** StopY (%), **F** black on inside (1) or outside (0).
sMsgResponse		Text entered by the operator.
	nSlaveAutoStart	Starts FlexLoader Vision.
	nSlaveCycleStop	Stops FlexLoader Vision.

Continues on next page

From FlexLoader Vision	From the robot	Description/result
	nSlaveSelectIdCam1..4	Replaces the detail in FlexLoader Vision for the specified camera based on the selected ID number, i.e. the value of **nSlaveSelectIdCam1..4**.
	nExposureTimeCam1 .. 4	Sets the exposure time for camera 1..4 within the area 0%–100%.
	nContrastCam1..4	Sets the contrast for camera 1..4 within the area 0%–100%.
	nGainCam1..4	Sets the gain for camera 1..4 within the area 0%–100%.
	bResetPickedCam1..4	Resets the previous information on already picked details.

The following status signals are used in the communication. The signals are digital outputs and must be defined in the robot.

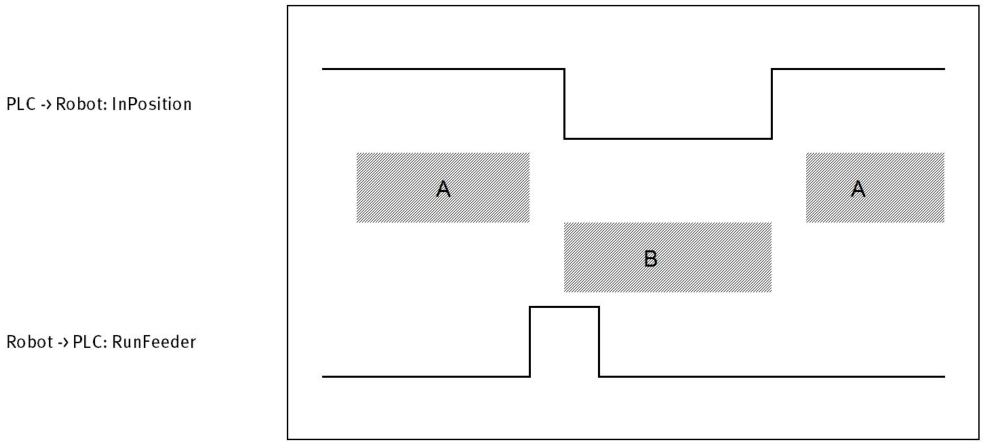
Signal	Description
DOF_Coord	Used to signal to the robot program that new coordinates are available.
DOF_MotSupTrigg	Cross-connected to the system output **MotSupTrigg**. Used so that FlexLoader Vision can receive the collision information.
DOF_Error	Cross-connected to the system output **Error**. Used so that FlexLoader Vision can receive information about fault incidents.
DOF_EndCycle	Used so that FlexLoader Vision can stop the robot when a correct cycle is completed and not in the middle of a cycle.
DOF_CycleEnded	Response signal from robot on **DOF_EndCycle**.
DOF_CycleOn	Cross-connected to the system output **CycleOn**. This signal is checked before the coordinates are sent. If the robot is not running, an alarm is displayed and FlexLoader Vision waits until the robot program starts again before coordinates are sent.
DOF_AutoOn	Cross-connected to the system output **AutoOn**. This signal is checked upon startup and stop through FlexLoader Vision.
DOF_ResetEStop	Cross-connected to the system input **ResetEStop**. Used so that FlexLoader Vision can automatically start the robot motors after an emergency.
DOF_ MotorsOn	Cross-connected to the system input **MotorOn**. Used so that FlexLoader Vision at the selected option can start the robot via system inputs.
DOF_ StartAtMain	Cross-connected to the system input **MotorOn**. Used so that FlexLoader Vision at the selected option can start the robot via system inputs.
DOF_ StopEndInstr	Cross-connected to the system input **MotorOn**. Used so that FlexLoader Vision at the selected option can stop the robot via system inputs.
DOF_ MotOnStart	Cross-connected to the system input **MotorOn**. Used so that FlexLoader Vision at the selected option can start the robot via system inputs.
DOF_RunFeeder1...4	Cross-connected to **RunFeeder1...4**. Used by **VisionCom** to run supply equipment under camera 1-4.

Continues on next page

Signal	Description
DOF_InPosition1...4	Cross-connected to **Feeder1(...4)InPosition**. Used by **VisionCom** to check when a new detail has reached camera 1-4.

14.4.2 Communication between the robot and the PLC

The robot and the PLC communicates via digital signals. The typical communication between part feeder and robot is shown below.



xx1800001066

Pos.	Description
A	Taking images and picking.
B	The conveyor is running.

14.4.3 Communication between Master and Slave

Signal definitions

The following commands can be used:

Master		FlexLoader Vision	Description
AutomaticStart byte{0;1}	>		Starts the system. Reply with either **AutomaticStrtOk** or **AutomaticStrtNok**. For handshaking, see diagram below.
	<	AutomaticStrtOk byte{0;1}	System is started.
	<	AutomaticStrtFail byte{0;1}	Error when starting system.
CycleStop byte{0;1}	>		Stops the system. Reply is either **CycleStopOk** or **CycleStopNok**. For handshaking, see diagram below.
	<	CycleStopOk byte{0;1}	The system is stopped.
	<	CycleStopFail byte{0;1}	Error when stopping the system.
SelectIdCam1..4 byte{0;1}	>		Used for selecting a new ID number. Reply is either **SelectIdOk** or **SelectIdFail**. For handshaking, see diagram below.
ID-No. double word	>		ID-Value is locked by the **SelectIdCamX** command and must therefore be set beforehand.
	<	SelectIdOk byte{0;1}	Choice of ID successful.
	<	SelectIdNok byte{0;1}	Choice of ID unsuccessful.

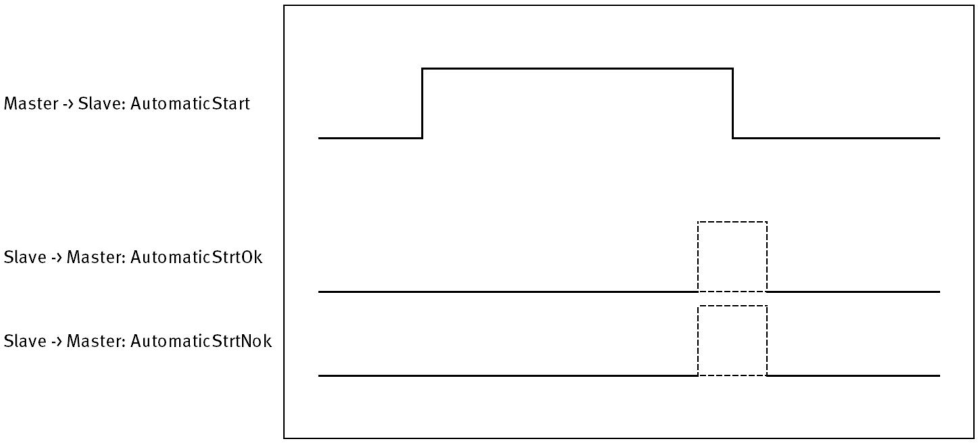
The default handshaking process is performed as follows:

The command must be 0 before the sequence begins. The command is then set to 1. The machine begins to carry out the desired operation, after which the reply is set to 1. To confirm the receipt of the reply, the command must be set back to 0, after which the reply is set to 0. Note that in most cases there are two alternative replies.

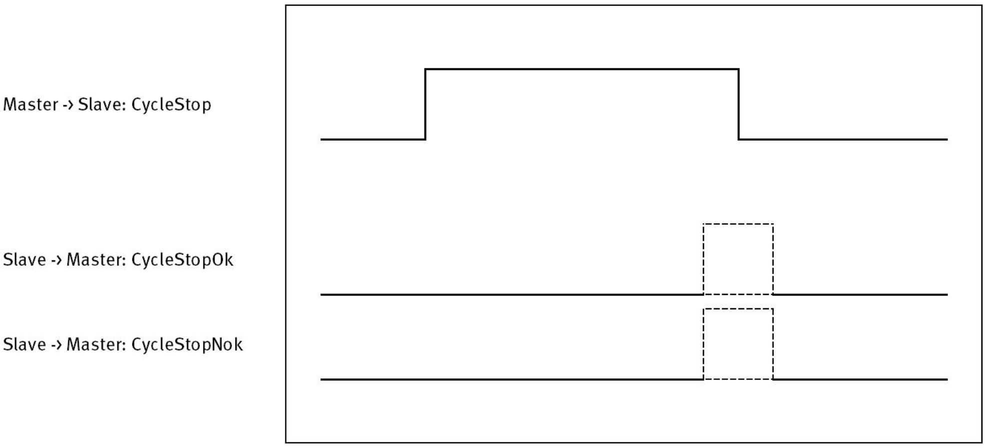
When the program is started all signals are reset to 0.

Continues on next page

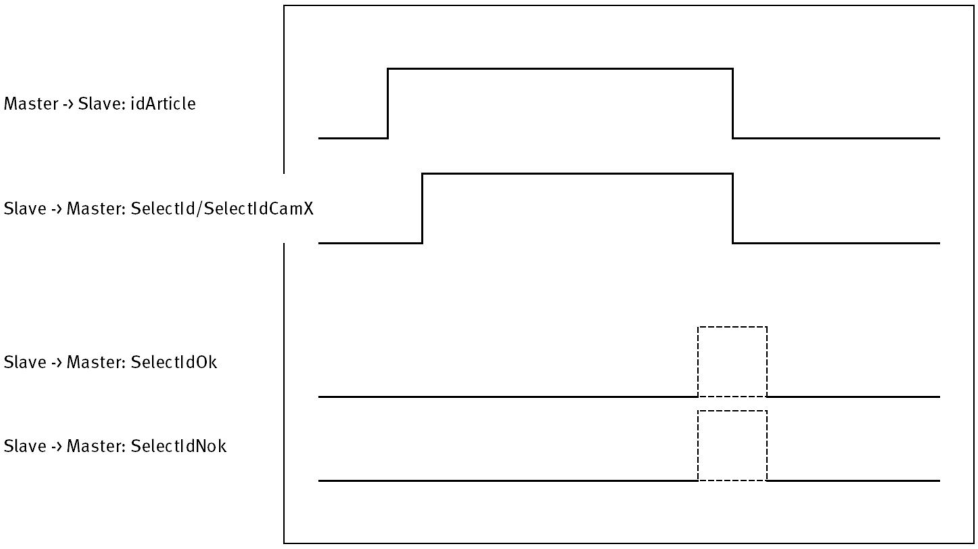
The handshaking sequences for **AutomaticStart**, **CycleStop** and **SelectIdCamX** are shown in the diagrams below.


Master -> Slave: AutomaticStart
Slave -> Master: AutomaticStrtOk
Slave -> Master: AutomaticStrtNok

xx1800001015


Master -> Slave: CycleStop
Slave -> Master: CycleStopOk
Slave -> Master: CycleStopNok

xx1800001016


Master -> Slave: idArticle
Slave -> Master: SelectId/SelectIdCamX
Slave -> Master: SelectIdOk
Slave -> Master: SelectIdNok

xx1800001017

Continues on next page

PLC

FlexLoader Vision and Master communicates via a network to special memory locations in the PLC. The communication consists of commands (such as start/stop/selection of detail), and some alarm monitoring. The information is located in the PLC memory through definition in the user interface of FlexLoader Vision. For information about memory cell types in the PLC memory (byte, word, double word), see the table above.

Robot

It is possible to allow an external PLC to be Master for FlexLoader Vision via the robot. This means that the external PLC can stop, start and change detail in FlexLoader Vision. The robot acts as an intermediary to get it to work. The following signals can be exchanged between the robot and the PLC.

- **AutomaticStart**, **CycleStop**, **SelectIdCamX**
- **AutomaticStrtOk**, **AutomaticStrtNok**, **CycleStopOk**, **CycleStopNok**, **SelectIdOk**, **SelectIdNok**
- **IdArticle** (group input 16 bits)

Note that both background modules for the Slave function through the robot and program module must be called **VisionSlave** for FlexLoader Vision to communicate correctly. **VisionSlave** is available as a standard module.

This page is intentionally left blank

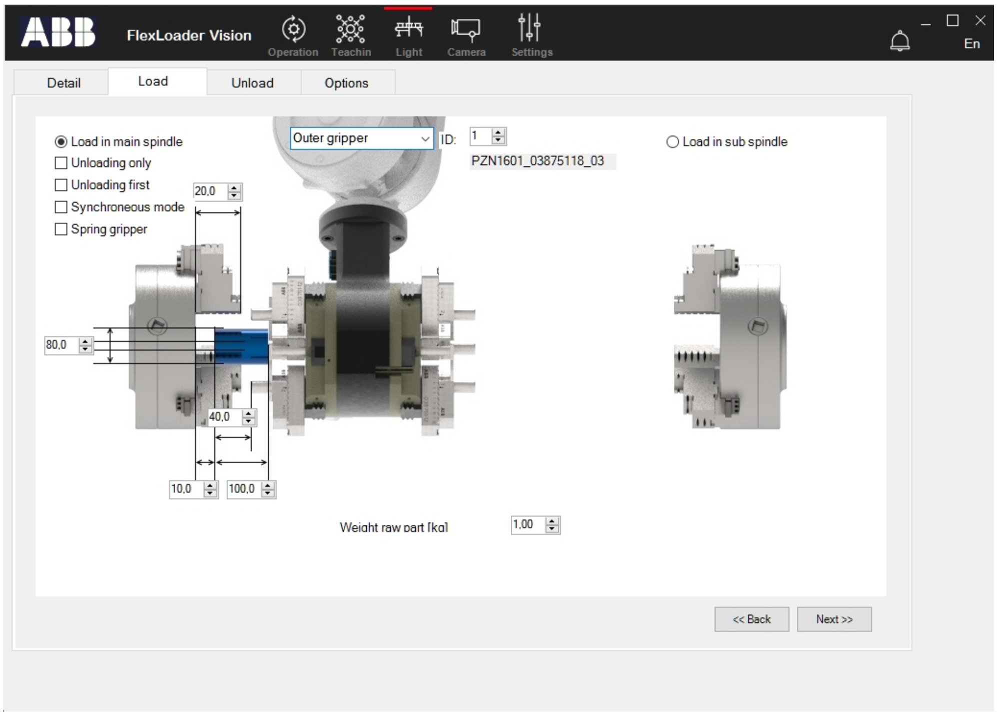
15 FlexLoader Vision Lite

15.1 Introduction

General information

FlexLoader Vision Lite is an automation concept especially designed for lathe applications. FlexLoader Vision Lite makes it possible to load and unload blanks and pipes with 6-axis robots for a CNC controlled lathe without the need of programming. Automation cells are fully managed from a specially designed interface where key data such as the blank dimensions, gripper data and the lathe's chuck data is provided in an intuitive and simple way via a graphic interface.

The interface uses data and designations that are well-known and established terms when it comes to CNC programming. This makes it possible to control loading and unloading of new blanks and pipes without knowledge of robot programming. The teachin is carried out in a way that feels intuitive and obvious to operators who are used to programming CNC machines.



● Load in main spindle
□ Unloading only
□ Unloading first
□ Synchroneous mode
□ Spring gripper
Outer gripper ID: 1
PZN1601_03875118_03
○ Load in sub spindle
20,0
80,0
40,0
10,0
100,0
Weight raw part [kg] 1,00

xx1800000275

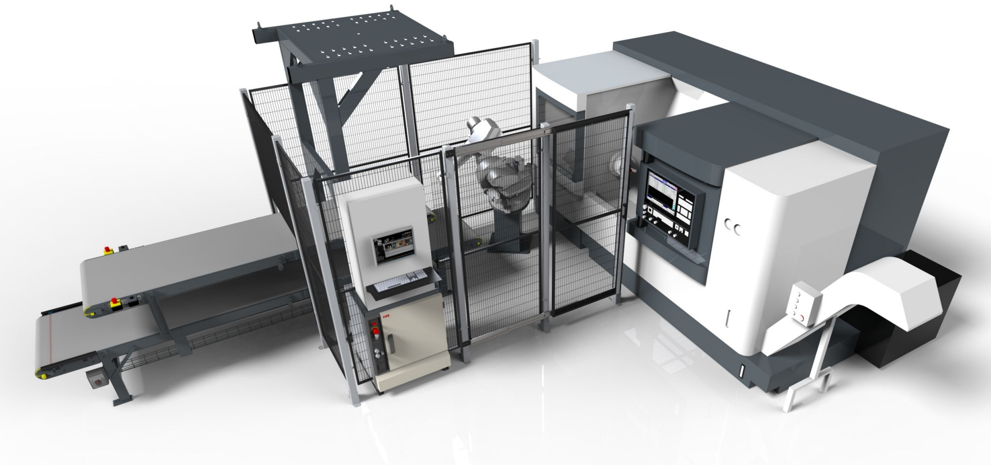
How it works

The raw material is placed on a conveyor belt in the robot cell. The positions of the blanks or pipes are identified using ABB's proven proprietary vision system, FlexLoader Vision, which guides the robot to pick the parts directly from the buffer belt. The robot is equipped with a double gripper with three finger gripping and adjustable fingers.

One gripper is used to pick and load the material while the other is used to pick out finished machined parts and place them on a buffer belt leading out of the cell. The position of the fingers can be adjusted to manage the raw material and finished

Continues on next page

parts with varying diameters. When a new diameter is specified at teachin, the system states the position that the gripper fingers should be in.



xx1800000276

Easy to use and upgrade

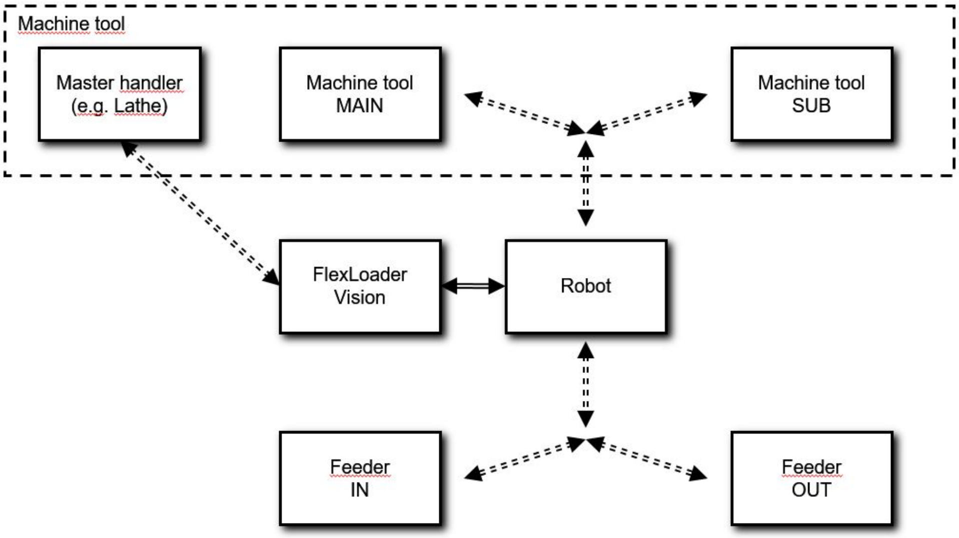
FlexLoader Vision Lite is an easy and economical step into automation with 6-axis robots. Even customers without any previous experience of 6-axis robots and who lack the skills to program robots themselves can make use of the flexible automation solution. FlexLoader Vision Lite even makes automation of short series with different types of blanks and pipes possible. In addition, CNC operators can easily and quickly operate loading and unloading.

One limitation in a FlexLoader Vision Lite system is that it can handle axially symmetrical parts only. For customers who, in the future, come across other types of parts that need loading with the correct rotation or require different types of post processing (e.g. deburring, marking or air cleaning), the software for the system can easily be upgraded to a standard FlexLoader Vision system, offering complete functionality and flexibility.

Continues on next page

System structure

A FlexLoader Vision Lite system is usually made up as follows:



xx1800000277

The operator interface of FlexLoader Vision Lite can be displayed in two ways:

- Integrated in FlexLoader Vision (and then run on the FlexLoader Vision computer).
- On the operating panel of the lathe, as a Master to FlexLoader Vision. There it is possible to choose details, start and stop the system, and perform teachins of new parts.

Operating modes

The following standardized operating modes are supported:

- First loading in main-spindle, then unloading from main- or sub-spindle.
- First unloading from main or sub-spindle, then loading in main-spindle.
- Unloading from sub-spindle. No loading is performed. (Used with bar feeders.)
- Handling of remains.
- After unloading, the selected actions are carried out according to the actions chosen on the **Options** tab.
- Both one and two spindle lathes are supported.

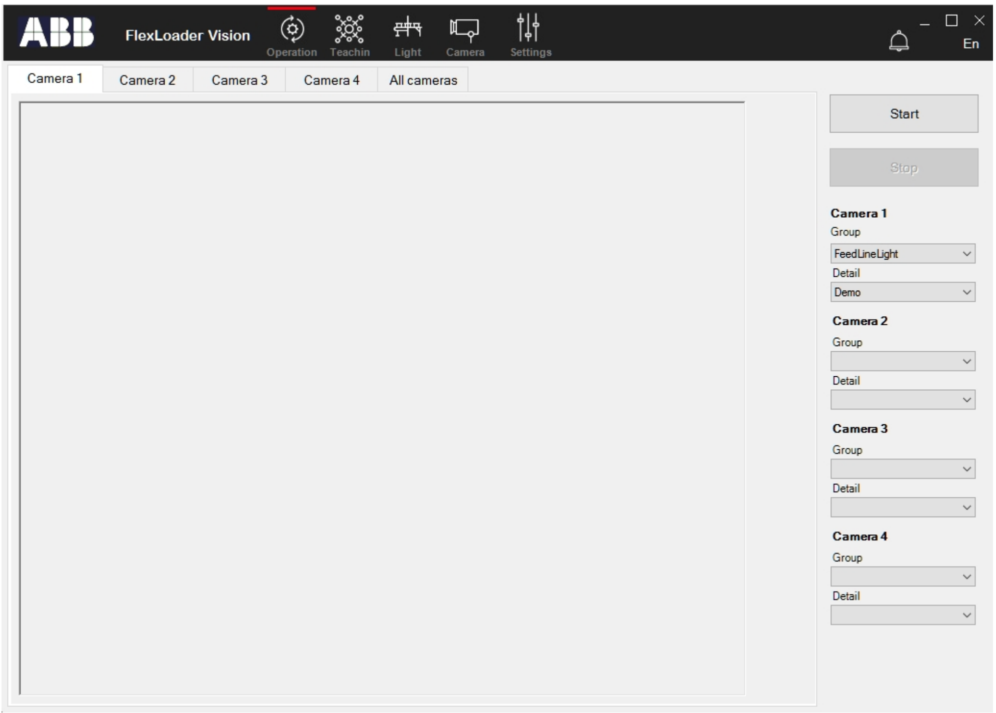
General operation

The general operation of FlexLoader Vision, as well as installation and configuration, is described in the first part of this manual.

15.2 Operation

Introduction

The following **Operation** view is shown when the system is integrated in FlexLoader Vision:



Camera 1
Group: FeedLineLight
Detail: Demo

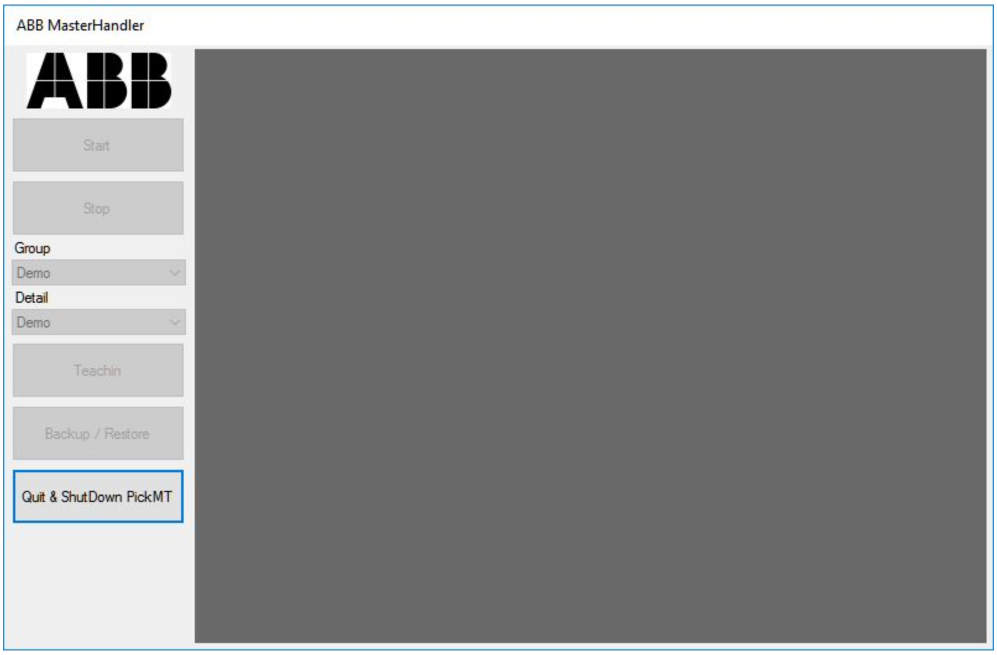
Camera 2
Group:
Detail:

Camera 3
Group:
Detail:

Camera 4
Group:
Detail:

xx1800000278

The following **Operation** view is shown when the system is integrated in the lathe operators panel:



Group: Demo
Detail: Demo

xx1800000279

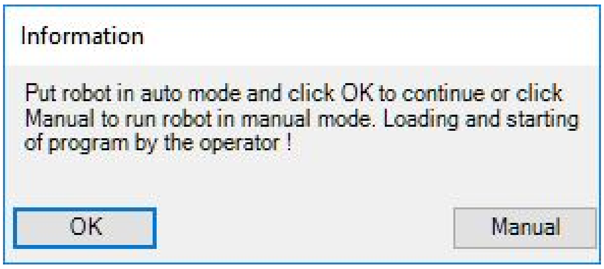
Continues on next page

The **Group** drop-down list is used to select the current group. The **Detail** drop-down list is used to select the detail to be run.

Starting the system

To start the system, click the **Operation** icon. Then in the drop-down lists under **Camera,** select the **FlexLoaderVisionLite** group, select the correct detail and click **Start.** The specified program is now loaded and the robot will start automatically if set to auto mode.

If the robot is not in auto mode, the following dialog box will appear when you click **Start.**



Information

Put robot in auto mode and click OK to continue or click Manual to run robot in manual mode. Loading and starting of program by the operator !

xx1800000274

If you want to continue with the robot in auto mode, put the robot in auto mode and click **OK.**

If you want to continue with the robot running at reduced speed, click **Manual.** Note that FlexLoader Vision does not load the program that is specified for the detail if you select **Manual**, instead the correct program must be loaded and started manually.

We recommend to start the system at least once with the robot in auto mode in order to load all modules correctly.

Note that the program must be started from the beginning.

Stopping the system

To stop the system, click **Stop** in FlexLoader Vision. If the robot is in **Auto** mode, it will also be stopped. Note that clicking **Stop** does not make the robot stop immediately. Instead it will stop at the end of its cycle or after a certain time. To stop the robot immediately, you must use the stop button on the robot or, if an emergency situation has arisen, use the emergency stop.

If FlexLoader Vision is configured so that the robot should control the stop situation, the stop button behaves differently. In this case FlexLoader Vision does not actively stop the robot when the operator clicks **Stop.** Instead, only **DOF_EndCycle** is sent to the robot, whereupon FlexLoader Vision awaits the robot to call the procedure **StopFlexLoader Vision** which in turn will stop everything as usual (as described above).

In case the user clicks **Stop** in FlexLoader Vision again, while FlexLoader Vision is awaiting the call to **StopFlexLoader Vision**, FlexLoader Vision will enforce a stop. This means that FlexLoader Vision will no longer be waiting for the robot, but will instead stop as usual (as in the case where FlexLoader Vision is configured to control the robot stop).

Continues on next page

Letting the robot control the stop situation is, for example, useful when running special cycles that cannot be stopped in the middle of operation. In this case, the robot normally knows when a stop is appropriate.

15.3 FlexLoader Vision Lite teachin

The FlexLoader Vision Lite teachin can be performed in two ways:

To perform a teachin in the operator panel of the lathe, click **Teachin**.

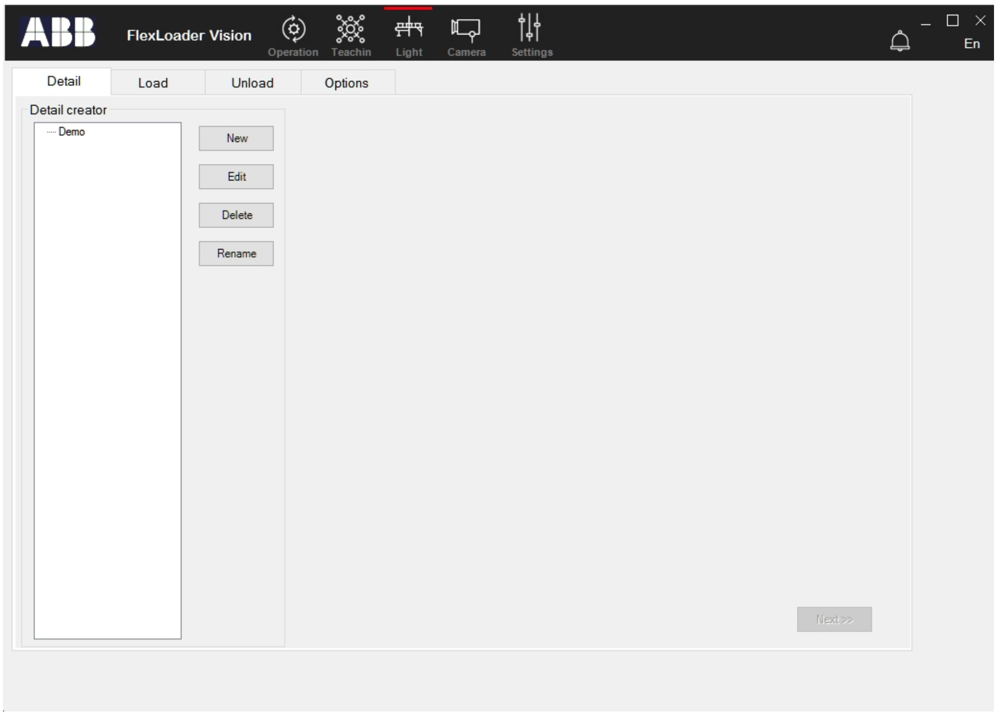
To perform a Lite teachin on the FlexLoader Vision PC, click the **Lite** icon.

On the **FlexLoader Vision Lite** menu there are four tabs to choose between: **Detail**, **Load**, **Unload** and **Options**.

Continues on next page

15.3.1 Detail

Overview



xx1800000280

On the **Detail** tab, a list of all FlexLoader Vision Lite details that are in the system is displayed.

Detail creator

To edit a detail, select the detail and click **Edit**.

To create a new detail, click **New** and enter a new name. Click **OK**.

To rename a detail, select the detail, click **Rename** and specify a new name.

To delete a detail, select the detail and click **Delete**. Then confirm the deletion.

15.3.2 Load

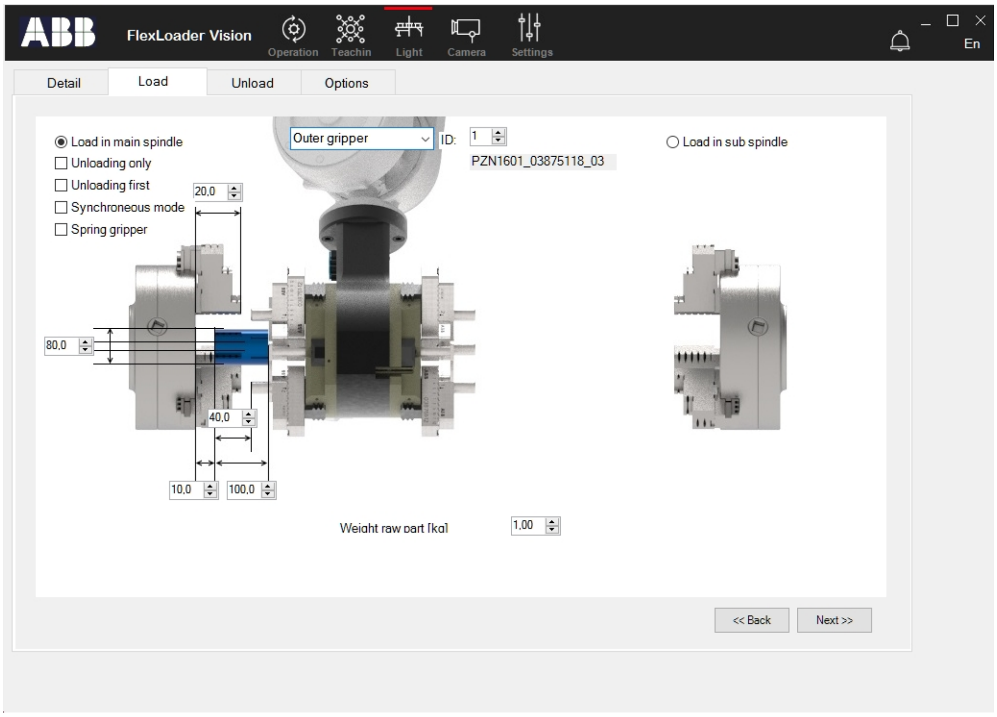
Overview

The **Load** tab specifies all the essential dimensions for loading a detail.

- Diameter and height of the raw material.
- Grip height for the raw material (TCP for the gripper fingers is assumed to be at the lower edge of the fingers).
- The protrusion of the chuck fingers and the insertion depth of the part.
- Part weight.

In addition, the gripper type is specified (**Outer gripper**/**Inner gripper**) as well as the ID number of the gripper fingers. If desired you can choose to load in the sub-spindle.

If enabled by the integrator, activation codes for user-defined sequences can be sent to the robot.



● Load in main spindle
□ Unloading only
□ Unloading first
□ Synchroneous mode
□ Spring gripper

Outer gripper ID: 1
PZN1601_03875118_03

○ Load in sub spindle

20,0
80,0
40,0
10,0
100,0

Weight raw part [kg] 1,00

xx1800000275

Normal loading procedure

Normally, the new raw material is loaded first and the finished detail is unloaded later:

1. The robot retrieves the detail on the camera belt.
2. The robot loads the new detail in the machine.
3. The robot retrieves the finished detail in the machine with its empty gripper.
4. The robot exits the machine, performs any activated options and leaves the finished detail on the outconveyor.

Continues on next page

Unloading first

If the detail weight exceeds the maximum allowed robot load, or if there is no space for a new detail in the machine when a finished detail is to be retrieved, select **Unloading first**. This means that the finished part is unloaded first. **Unloading first** should also be selected if the robot only has a single gripper.

Note that this method of loading is significantly slower than the normal procedure and should only be selected if loading cannot be carried out in the normal way.

Loading procedure when **Unloading first** is selected:

1. The robot enters the machine and retrieves the finished detail.
2. The robot exits the machine, performs any activated options and leaves the finished detail on outconveyor.
3. The robot retrieves the detail on the camera belt.
4. The robot enters the machine and loads the new detail in the machine.
5. The robot exits the machine.

Unloading only

When turning from bar material the option **Unloading only** can be activated. Note that this checkbox can be hidden through a setting in the configuration file, see *FlexLoader Vision Lite configuration on page 290*.

Synchroneous mode

If you need to operate the system in synchronous mode, select **Synchroneous mode**. In this mode, a part is picked from the inconveyor, the sub-spindle is emptied, the robot leaves the machine, the machine docks from the main-spindle to the sub-spindle, the robot loads the main-spindle, and leaves the previously retrieved part on the outconveyor.

Spring gripper

If you select **Spring gripper**, the detail will be pressed against the spring when picking and then the spring force will be used when loading the machine. The item can then be released by the robot before the machine chuck closes. This can help the chuck to self-center the detail when loading. Note that this checkbox can be hidden through a setting in the configuration file, see *FlexLoader Vision Lite configuration on page 290*.

Note

Select **Spring gripper** only if the robot is actually equipped with spring loaded grippers.

15.3.3 Unload

Overview

The **Unload** tab specifies all the essential dimensions for unloading a detail.

- Diameter and height of the finished part.
- Grip height for the finished part (TCP for the gripper fingers is assumed to be at the lower edge of the fingers).
- The protrusion of the chuck jaws and the insertion depth of the part.
- Finished part weight.

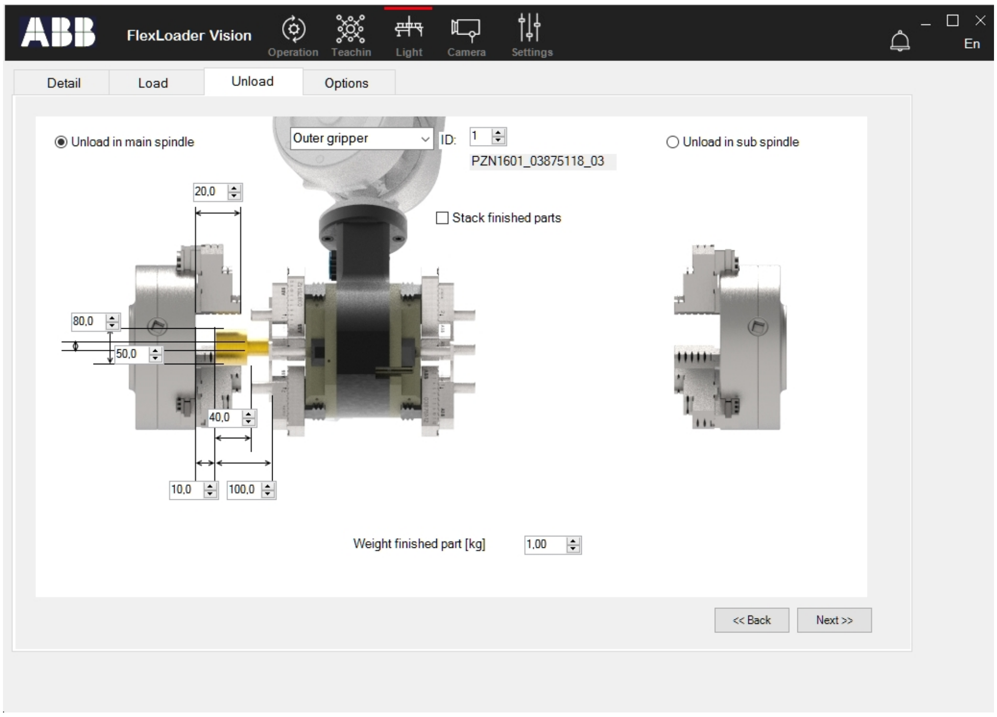
In addition, the gripper type is specified (**Outer gripper**/**Inner gripper**) as well as the ID number of the gripper fingers.

The grip diameter and outer diameter for a finished detail can differ from the dimensions of the original detail and are specified separately.

If you want the finished parts to be stacked, select **Stack finished parts** and specify the number of parts that should be stacked.

Unloading from main-spindle

The normal loading and unloading procedure is specified above, see *Normal loading procedure on page 275*.



● Unload in main spindle

Outer gripper

ID: 1

PZN1601_03875118_03

○ Unload in sub spindle

□ Stack finished parts

20,0

80,0

50,0

40,0

10,0

100,0

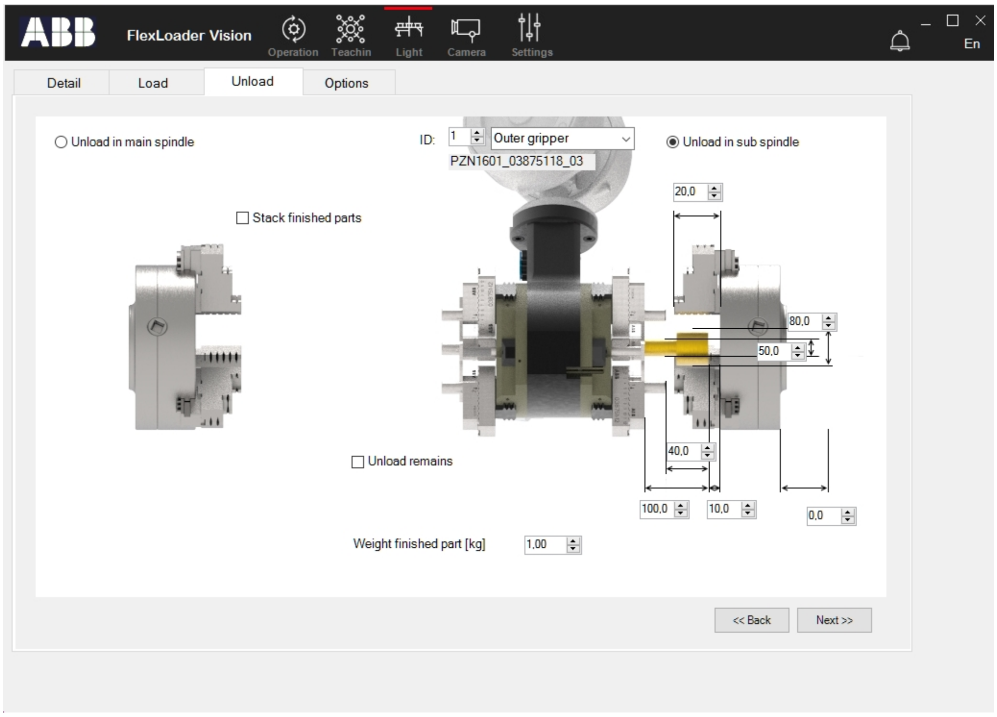
Weight finished part [kg] 1,00

xx1800000282

Continues on next page

Unloading from sub-spindle

When activating unloading from a sub spindle, a spindle offset can be specified.



○ Unload in main spindle

ID: 1 Outer gripper
PZN1601_03875118_03

● Unload in sub spindle

□ Stack finished parts

□ Unload remains

Weight finished part [kg] 1,00

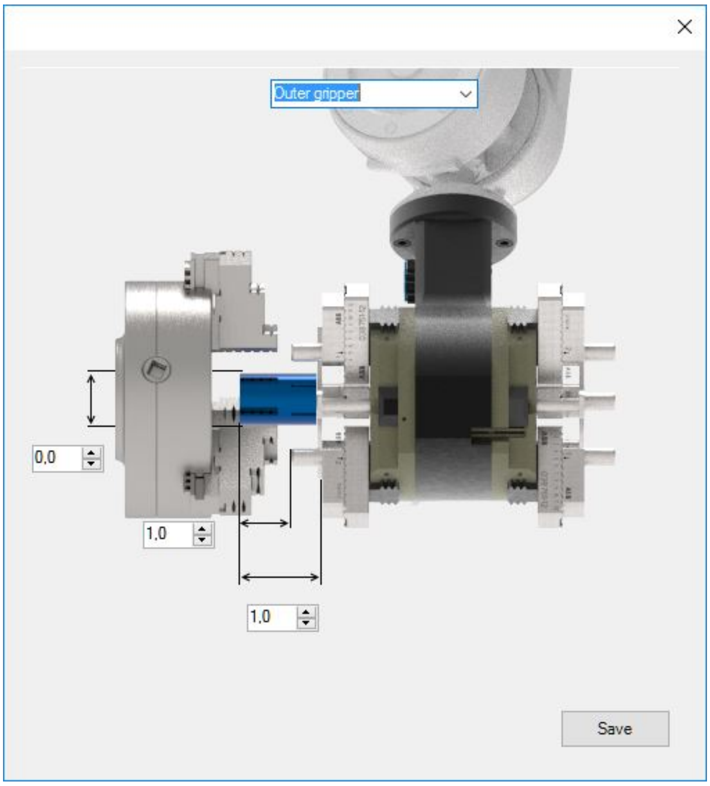
xx1800000283

Continues on next page

Unloading remains

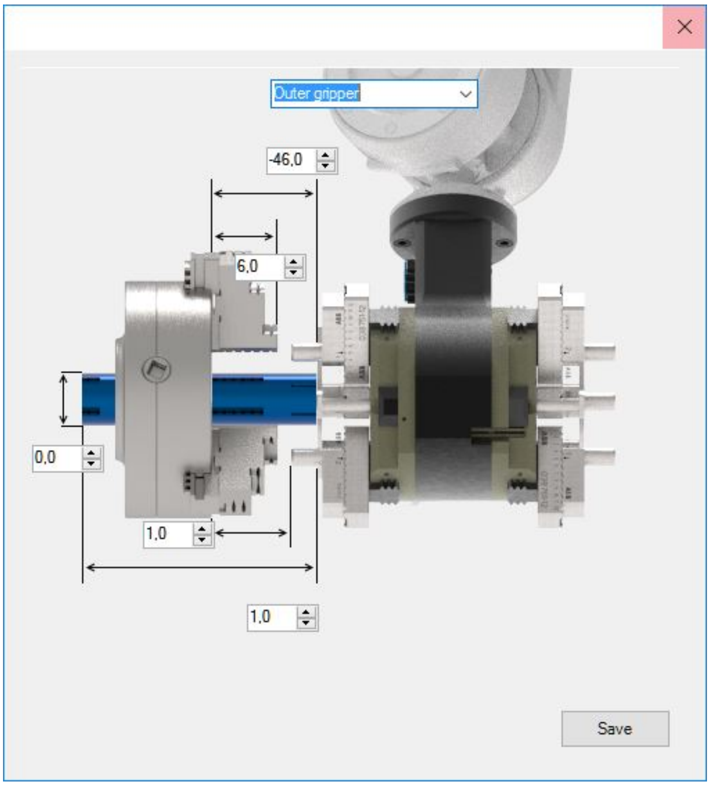
To activate handling of remains, select **Unload remains**. Click **Configure** to specify the dimensions for the detail handling. Unloading of remains is done by the loading gripper.

Note that the **Unload remains** checkbox can be hidden through a setting in the configuration file, see *FlexLoader Vision Lite configuration on page 290*.



xx1800000284

Continues on next page



Outer gripper

-46,0

6,0

0,0

1,0

1,0

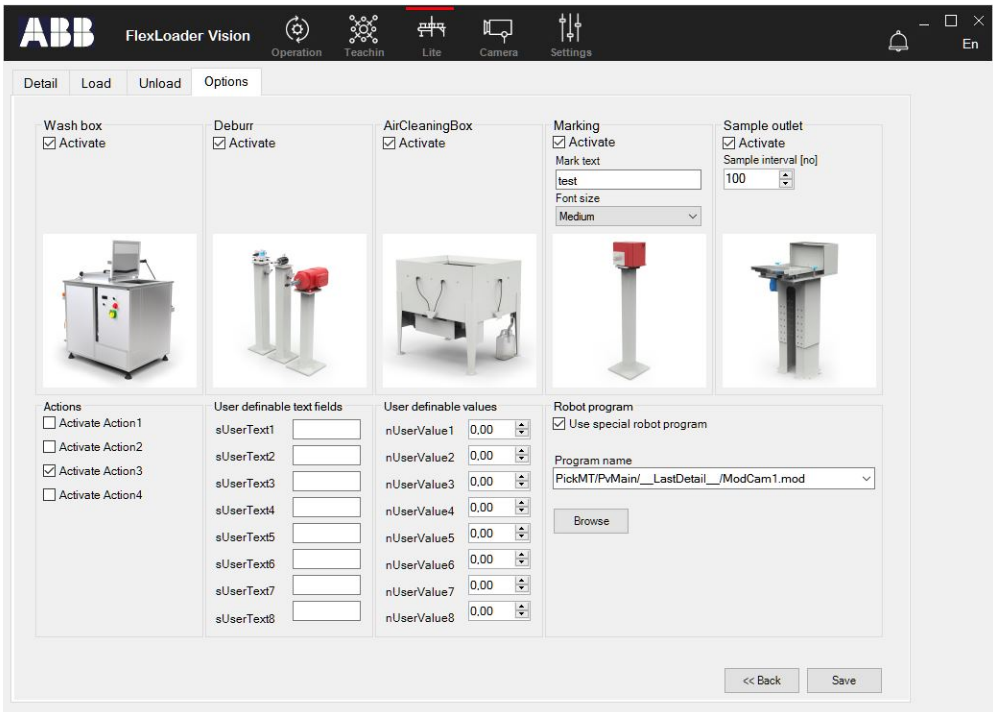
xx1800000285

15.3.4 Options

Overview

There are a number of options available for FlexLoader Vision Lite. The options selected for the current system are displayed on the **Options** tab. If there are no options added to the system, the **Options** tab does not appear at all. The following options are available:

- Wash box
- Deburr
- Air cleaning box
- Marking
- Statistical outlet
- Actions
- Special robot program
- User-definable texts
- User-definable values



Wash box
☑ Activate

Deburr
☑ Activate

AirCleaningBox
☑ Activate

Marking
☑ Activate
Mark text: test
Font size: Medium

Sample outlet
☑ Activate
Sample interval [no]: 100

Actions
□ Activate Action1
□ Activate Action2
☑ Activate Action3
□ Activate Action4

User definable text fields
sUserText1
sUserText2
sUserText3
sUserText4
sUserText5
sUserText6
sUserText7
sUserText8

User definable values
nUserValue1 0,00
nUserValue2 0,00
nUserValue3 0,00
nUserValue4 0,00
nUserValue5 0,00
nUserValue6 0,00
nUserValue7 0,00
nUserValue8 0,00

Robot program
☑ Use special robot program
Program name: PickMT/PvMain/__LastDetail__/ModCam1.mod

xx1800000286

The specified options are performed just before the detail is left on the outconveyor. If several options have been selected they are performed in the order defined in the robot program. If this order is not suitable, a simple adjustment in the robot program is required. Note that **Actions** and **Robot program** are different regarding function. For more information, see below.

Continues on next page

Wash box

When **Wash box** is activated, the robot will run washing of the detail. No extra robot programming is required for simple operations, as the detail information that is already entered determines how the robot is to perform the washing. More complex washing must be programmed for each detail and called using the **Special robot program** feature.

Deburr

When **Deburr** is activated, the robot will run deburring of the detail. Due to the complexity of the deburring operation it must be programmed for each detail and called using the **Special robot program** feature.

Air cleaning box

When **Air cleaning box** is activated, the robot will run air cleaning of the detail. No extra robot programming is required for simple operation. The detail information that is already entered determines how the robot is to perform air cleaning. More complex air cleaning must be programmed for each detail and called using the **Special robot program** feature.

Marking

When **Marking** is activated, the robot will mark the detail using the marking machine. Mark text for the detail is entered in the **Mark text** field. In addition, there are three predefined sizes for the text, **Small**, **Medium** and **Large**, which allow different sizes of mark text depending on the detail. These text sizes correspond to three setup files in the marking controller.

The maximum number of characters for the mark text will change depending on the selected font size. If too many characters are entered, the mark text will automatically be shortened to the maximum length. In addition a warning appears that the maximum length has been reached.

No extra robot programming is required for simple operation. The detail information that is already entered determines how the robot is to perform the marking. More complex marking must be programmed for each detail and called using the **Special robot program** feature.

Statistical outlet

When statistical outlet is activated, the robot leaves a sample detail in the statistical outlet at regular intervals. Sample interval indicates how many details are to be produced before the next sample detail is to be left.

Actions

This option contains four completely user definable functions that are undefined from start. If there is a further option or function that is not suitable for the previous ones, this can be especially programmed and added to the robot program in the correct place. The four activation boxes can then be connected with the new function(s). This makes it possible to activate the special functions for selected details.

Continues on next page

If these special functions are used together with any of the above, the internal order of these depends on where in the robot program the new function is inserted.

Special robot program

By default, the robot movement in FlexLoader Vision Lite is controlled by the module **ModCam1.mod**, which contains the default movements in their default workflow.

In some cases, changes are required that affect the flow in the cell and other additions that are not suitable for the aforementioned options. In these cases a special detail-specific robot program is used. To select a special robot program on the robot that is adapted for the specific detail, select **Use special program in robot** and specifiy a program file in the list below.

Note

All special programs must be programmed and tested before being selected on the **Options** tab.

As the program is completely detail-specific, other options can be placed in the robot program depending on the project requirements.

User-definable values

If required, up to 8 numeric and up to 8 text-based values can be sent to the robot. These can be used in different ways, for example custom parameterized processes.

15.4 Commissioning

15.4.1 Robot

Introduction

During commissioning, FlexLoader Vision is commissioned in the normal way, according to the first part of this manual. In addition to this, the following instructions should be followed.

FlexLoader Vision Lite assumes a specific geometry and coordinate system in the robot to work optimally. The **FlexLoaderVisionLite.sys** module must be adapted to the actual cell geometry.

The variable names in FlexLoaderVisionLite.sys must not be changed, as FlexLoader Vision transfers data to these variables.

The section below describes the setup and the necessary settings.

Coordinate systems

The base coordinate system of the robot must be horizontal (not angled), preferably with either the x or y axis parallel with the front of the lathe.

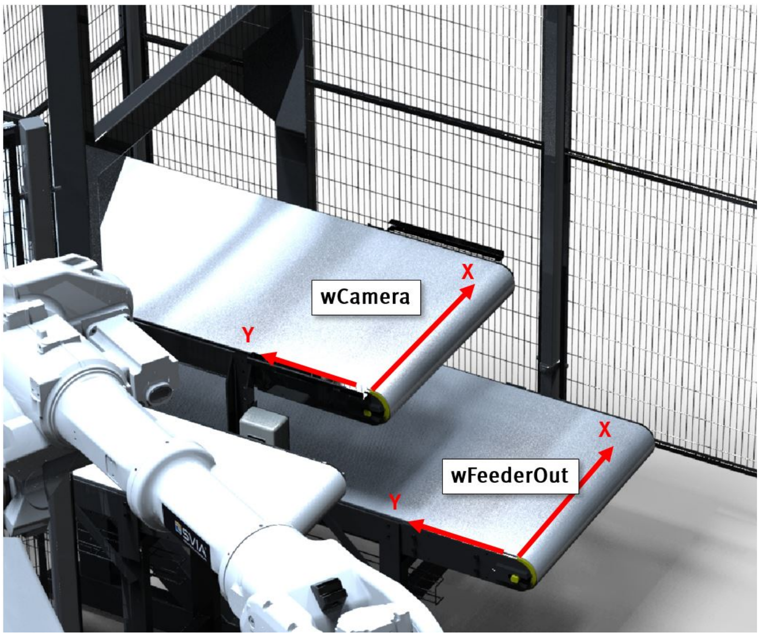
The following coordinate system is used in a FlexLoader Vision Lite system:

Robot data	Description
wCamera1	Coordinate system for picking from the feeder inconveyor, calibrated according to the calibration instructions for FlexLoader Vision, see *Camera on page 109*.
wFeederOut	Coordinate system for leaving on outconveyor. The x-axis is above the shaft of the rotation roller, the y-axis is at the left belt edge (in the movement direction of the belt) and the z-axis points upwards.

Perform a standard 3-point calibration of **wFeederOut**. Select the correct calibration tool, depending on which gripper is holding the calibration tool (Gripper 1: **tCalibTool1**; Gripper 2: **tCalibTool2**).

Continues on next page

Then, call the routine **CalibSpeedFeederOut** in the module **Calibration.sys** and follow the instructions.

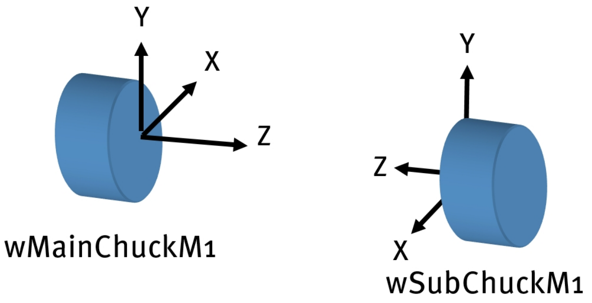


xx1800000317

Robot data	Description	Routine
wMarker	Coordinate system for the marking station. Calibration similar to **wFeederOut**.	**CalibMarker**, place zero position at the marking pins working position.
wAirClean	Coordinate system for the air cleaning box. Calibration similar to **wFeederOut**.	**CalibAirClean**, place zero position in the center of the air cleaning box, on a height where the air cleaning of the lower end of the detail is to start.
wSample	Coordinate system for the statistical outlet. Calibration similar to **wFeederOut**.	**CalibSampleOutlet**, place zero position in the center of the outlet, where the center of the details will be placed.
wDeburr	Coordinate system for the deburring units. Calibration similar to **wFeederOut**.	Manual calibration.
wWasher	Coordinate system for the washing box. Calibration similar to **wFeederOut**.	**CalibWasher**, place zero position in the center of the washing box, on a height where washing of the lower end of a detail is to start.
wMainChuckM1	Coordinate system for the left chuck of the first lathe (main spindle).	**CalibMainChuck**
wSubChuckM1	Coordinate system for right chuck of the first lathe (sub spindle).	**CalibSubChuck**

Continues on next page

The x-y plane of the coordinate system for a chuck must be at the front edge of the chuck, with x=0,y=0 at the rotation shaft of the chuck. See diagram.



xx1800000318

Perform a standard 3-point calibration of **wMainChuckM1**. The calibration tool must be placed in gripper 1 (tCalibTool1). Gripper 2 should be used for gripping the detail.

Then call the routine **CalibMainChuck** in module **Calibration.sys** and follow the instructions. On request, a detail must be gripped with the main chuck. Position the robot so that this detail can be gripped by the robot gripper, precisely centered. Confirm to the calibration routine. From the current robot position, **wMainChuckM1** is moved to be located exactly in the main spindle center.

Perform a standard 3-point calibration of **wSubChuckM1**. The calibration tool must be placed in gripper 2 (**tCalibTool2**). Gripper 2 should be used for gripping the detail.

Then call the routine **CalibSubChuck** in module **Calibration.sys** and follow the instructions. On request, a detail must be gripped with the sub-chuck. Position the robot so that this detail can be gripped by the robot gripper, precisely centered. Confirm to the calibration routine. From the current robot position, **wSubChuckM1** is moved to be located exactly in the sub spindle center.

If the sub-spindle is movable, the calibration must be checked, so that the z-axis in **wMainChuckM1** is in line with the robot z-coordinate, even for long movements.

Robot data	Description
wRegrip	Coordinate system for regrip station. Standard 3-point calibration.
wTurnStation	Coordinate system for turning station. Standard 3-point calibration.

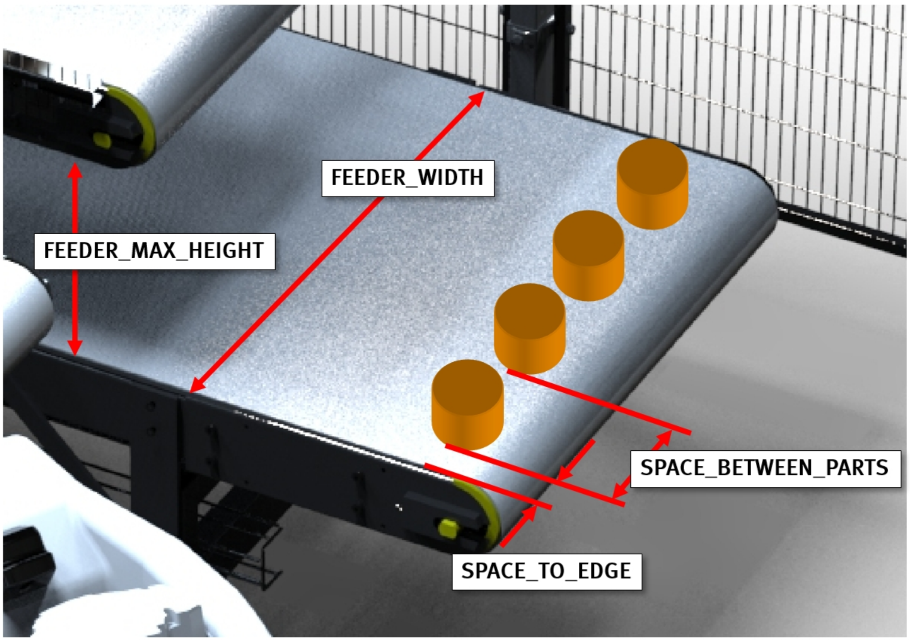
System constants

The **FlexLoaderVisionLite.sys** module contains some constants that are used to add details to the outconveyor. All dimensions are in mm.

Robot data	Description
FEEDER_WIDTH	Width of outconveyor.
SPACE_BETWEEN_PARTS	Space to be left between details on the outconveyor.

Continues on next page

Robot data	Description
SPACE_TO_EDGE	Minimum distance between belt edge and detail.
FEEDER_MAX_HEIGHT	Maximum height of details that are placed or stacked on the outconveyor.
MIN_RAW_DETAIL_DIAMETER	Minimum permitted detail diameter.
Y_DISTANCE_TO_LEAVE	Distance from outconveyor motor shaft to actual leave position.
MAX_LEAVE_ANGLE_OFFSET	The angle the robot leaves the part on the detail on the outconveyors right side. At the left side the angle offset is always zero. Set this value to 0 for no angle change.



xx1800000319

Compensation of load-induced deviations

Close to loading position, put a reasonable additional weight on the gripper and measure the load induced lowering of the robot arm.

Robot data	Description
nLoadCompensateDistance	Load induced lowering of the robot arm, in mm.
nLoadCompensate	Load that was used for measuring nLoadCompensateDistance

Gripper

The following gripper is used in a FlexLoader Vision Lite system:

Robot data	Description
tCalibTool1	Calibration gripper that attaches to gripper 1.
tCalibTool2	Calibration gripper that attaches to gripper 2.

Continues on next page

Robot data	Description
tGripper1	Used when loading and handling raw materials.
tGripper2	Used when unloading and handling finished details.

Zones and zone points

The robot's working range is divided into logical zones. These are positioned as wedges around robot axis 1 and should be contiguous.

These zones are described in detail in the RAPID reference, see appendix.

Further points

Robot data	Description
pHome	Home position for the robot in standby mode.
jpSoftwareSyncPos	This point is used for synchronization of the **SafeMove** system. It can usually be added to the same place as **pHome**.

Sequences

Standardized sequences are used to load and unload details in the machine, using the detail data from FlexLoader Vision. These sequences must be checked prior to each commissioning.

The sequences are based on the machine being loaded from the front. If the machine is loaded from above or if anything else is unusual, some adjustments are required.

For details see *FlexLoader Vision Lite functionality on page 320*.

Further sequences take care of communication, system checks, logic distribution, and similar. This might also need changing.

For details see *FlexLoader machine tool interface functionality on page 329*

WorldZones

Some WorldZones can be used to verify the presence of the robot in certain areas and to block certain movements.

Predefined world zones must be configured during commissioning, see *WorldZones on page 257*.

15.4.2 FlexLoader Vision Lite master detail

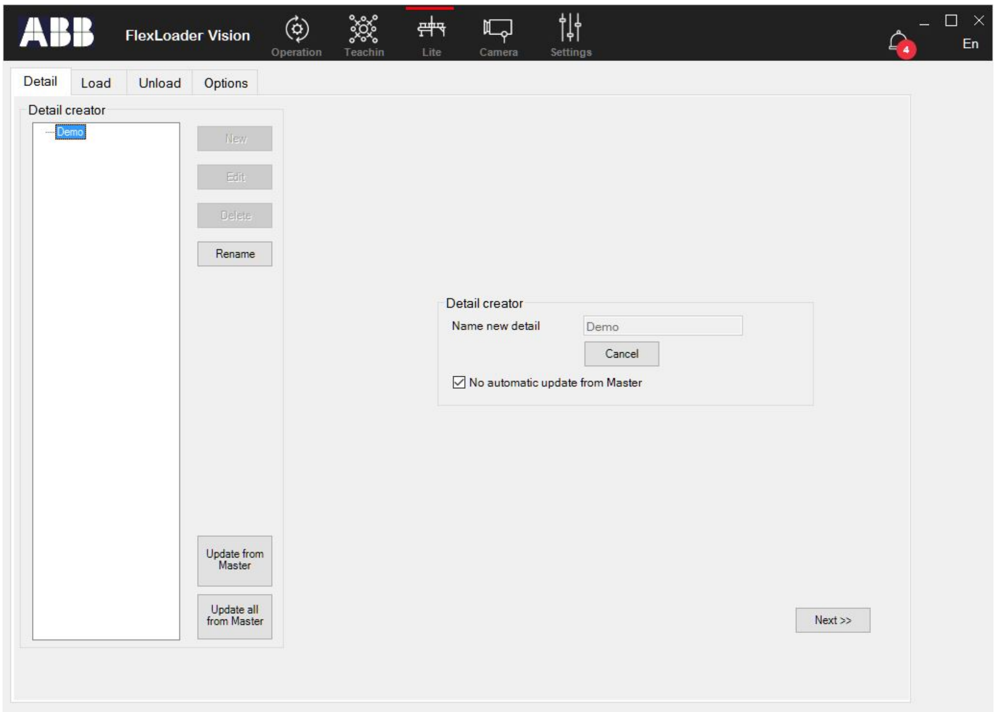
In addition to the basic configuration of FlexLoader Vision, the following settings must be made:

The detail FlexLoaderVisionLite must be taught in the **Masters** group. Each time a detail is saved in FlexLoader Vision Lite, the base values are copied from this master detail and adapted for that specific teachin detail.

This applies, for example, to settings for exposure times, image preparations, advanced position settings, accuracy requirements, settings for collision monitoring etc. It does, however, not apply to settings for the appearance of the detail and grip data.

An update of all details must be made after a change to the master detail. This update is made on the **Detail** tab on the FlexLoader Vision Lite menu. Press the Ctrl key, so that the **Update all from Master** button becomes visible. Click on it to update all details.

The automatic update of details upon saving can be disabled by checking **No automatic update from master** during teachin.



Detail creator

Name new detail: Demo

Cancel

☑ No automatic update from Master

xx1800000287

Changes to the master detail require FlexLoader Vision to be configured for FlexLoader Vision Lite with full access.

15.5 FlexLoader Vision Lite configuration

Configuration file

Introduction

The instructions below are for integrators, ABB's own technicians and for information. The customer usually does not need to edit this configuration.

The instructions below assume knowledge of the general setup of ABB's system and handling of XML files.

For operation of FlexLoader Vision Lite, the file **FlexLoaderVisionLite.xml** file must be configured. If an external operator panel is used, this file must be available for both FlexLoader Vision and MasterHandler.

Basic data

Specify whether the overlap between the gripper fingers is to be checked or not.

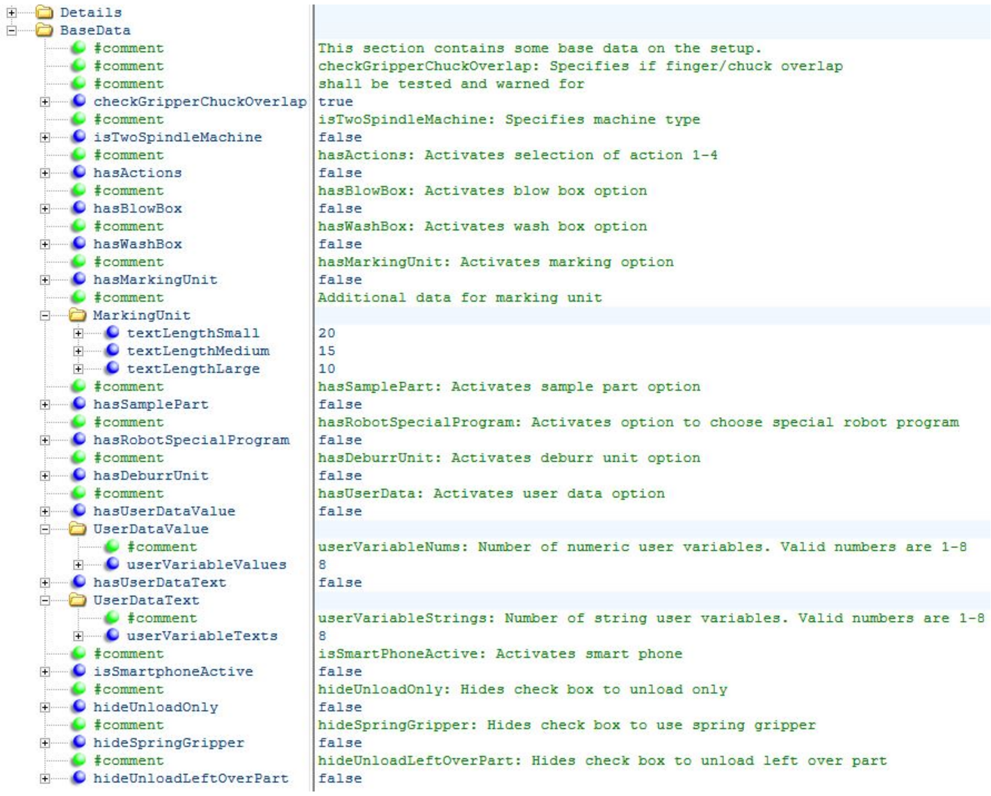
Also specify whether the lathe has a sub-spindle or not.

Indicate for each option if it is used or not. The options are: air cleaning box, wash box, marking unit, statistical outlet, deburr, actions and special program for robot. If marking is to be used, the maximum mark text lengths for small, medium and large must also be specified. These are connected to the mark data files that are used in the marker machine.

There is also the option to activate up to eight numeric values, which can be used freely for anything and will be stored with the detail. Similarly, there are up to eight

Continues on next page

string values. In addition, it is possible to to simplify for the operator by hiding certain choices on the loading and unloading page.



Details	
BaseData	
#comment	This section contains some base data on the setup.
#comment	checkGripperChuckOverlap: Specifies if finger/chuck overlap
#comment	shall be tested and warned for
checkGripperChuckOverlap	true
#comment	isTwoSpindleMachine: Specifies machine type
isTwoSpindleMachine	false
#comment	hasActions: Activates selection of action 1-4
hasActions	false
#comment	hasBlowBox: Activates blow box option
hasBlowBox	false
#comment	hasWashBox: Activates wash box option
hasWashBox	false
#comment	hasMarkingUnit: Activates marking option
hasMarkingUnit	false
#comment	Additional data for marking unit
MarkingUnit	
textLengthSmall	20
textLengthMedium	15
textLengthLarge	10
#comment	hasSamplePart: Activates sample part option
hasSamplePart	false
#comment	hasRobotSpecialProgram: Activates option to choose special robot program
hasRobotSpecialProgram	false
#comment	hasDeburrUnit: Activates deburr unit option
hasDeburrUnit	false
#comment	hasUserData: Activates user data option
hasUserDataValue	false
UserDataValue	
#comment	userVariableNums: Number of numeric user variables. Valid numbers are 1-8
userVariableValues	8
hasUserDataText	false
UserDataText	
#comment	userVariableStrings: Number of string user variables. Valid numbers are 1-8
userVariableTexts	8
#comment	isSmartPhoneActive: Activates smart phone
isSmartphoneActive	false
#comment	hideUnloadOnly: Hides check box to unload only
hideUnloadOnly	false
#comment	hideSpringGripper: Hides check box to use spring gripper
hideSpringGripper	false
#comment	hideUnloadLeftOverPart: Hides check box to unload left over part
hideUnloadLeftOverPart	false

xx1800001018

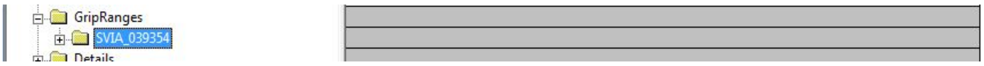
Gripper data

In a FlexLoader Vision Lite system the gripper data must be known to be able to perform collision checks and to provide information to the operator on how the gripper fingers are to be positioned on the robot.

The accompanying **FlexLoaderVisionLite.xml** already contains gripper data for standard gripper fingers. The standard fingers that are not to be used must be removed for the relevant application.

If special fingers are used, these must be defined.

The **GripRanges** interface contains individual sections for each gripper used. If necessary, copy the existing gripper data to a new gripper. The name of the section is the name that appears in the FlexLoader Vision Lite interface.

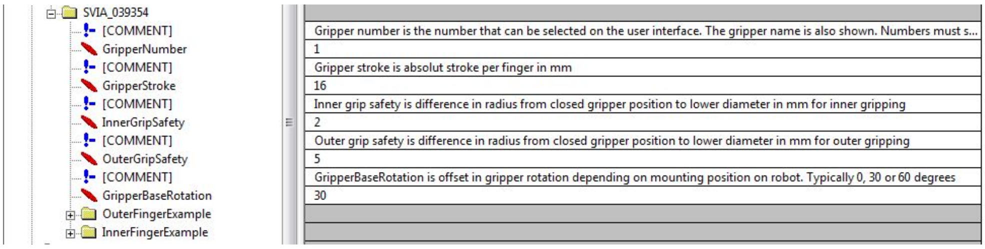


GripRanges
SVIA_039354
Details

xx1800001020

Continues on next page

Each section for a gripper contains basic data about the gripper as well as the set up of the gripper fingers. The following basic parameters are used:

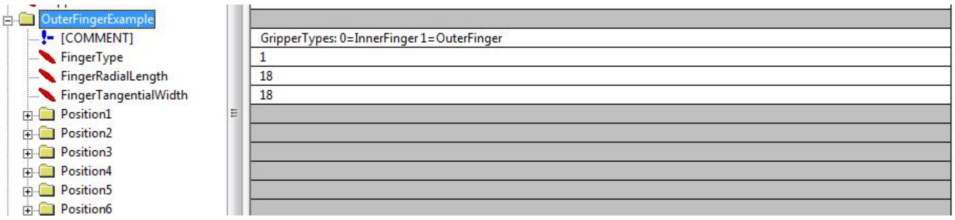


SVIA_039354	
[COMMENT]	Gripper number is the number that can be selected on the user interface. The gripper name is also shown. Numbers must s...
GripperNumber	1
[COMMENT]	Gripper stroke is absolut stroke per finger in mm
GripperStroke	16
[COMMENT]	Inner grip safety is difference in radius from closed gripper position to lower diameter in mm for inner gripping
InnerGripSafety	2
[COMMENT]	Outer grip safety is difference in radius from closed gripper position to lower diameter in mm for outer gripping
OuterGripSafety	5
[COMMENT]	GripperBaseRotation is offset in gripper rotation depending on mounting position on robot. Typically 0, 30 or 60 degrees
GripperBaseRotation	30
OuterFingerExample	
InnerFingerExample	

xx1800001021

Parameter	Description
GripperNumber	Serial number for gripper data. Must start at 1 and must be continuous for each gripper that is configured. This is the number that can be selected in FlexLoader Vision Lite user interface.
GripperStroke	Stroke per finger in mm.
InnerGripSafety	Used when gripping from the inside. Indicates the safety distance between the gripper fingers in the closed position and the minimum diameter that can be gripped. Typically 2–3 mm. See the images below.
OuterGripSafety	Used when gripping from the outside. Indicates the safety distance between the gripper fingers in the open position to the largest diameter that can be gripped. Typically 2–3 mm. See the images below.
GripperBaseRotation	Indicates the rotation of the gripper when installing the robot. Used when the gripper is drawn in FlexLoader Vision. Typically 0°, 30°, 60° or 90° (D in the image below).

Each section for a grip finger contains basic data about the grip finger as well as the set up of the positions. The following basic parameters are used:

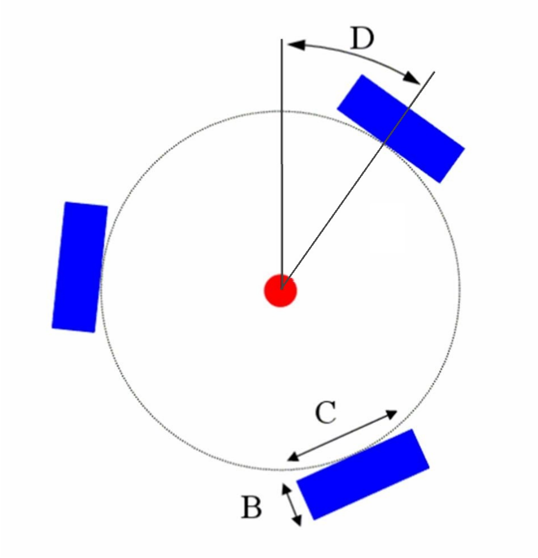


OuterFingerExample	
[COMMENT]	GripperTypes: 0=InnerFinger 1=OuterFinger
FingerType	1
FingerRadialLength	18
FingerTangentialWidth	18
Position1	
Position2	
Position3	
Position4	
Position5	
Position6	

xx1800001022

Parameter	Description
FingerType	Specify 0 for internal gripping and 1 for external gripping.
FingerRadialLength	Specify the length of the finger in radial direction in mm (B in the image below).
FingerTangentialWidth	Specify the width of the finger in mm (C in the image below).

Continues on next page

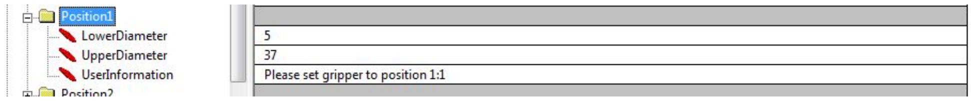


xx1800000969

Pos.	Description
B	FingerRadialLength
C	FingerTangentialWidth
D	GripperBaseRotation

Each position for a gripper finger is used for a certain diameter range.

Parameter	Description
LowerDiameter	Indicates the minimum gripper diameter for the finger position.
UpperDiameter	Indicates the maximum gripper diameter for the finger position.
UserInformation	The text that appears when the operator selects a detail. It specifies in which position the gripper finger is to be positioned to run the detail.



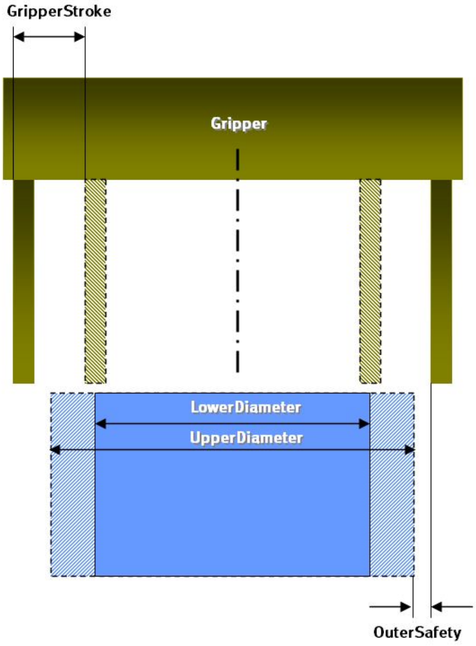
Position1	
LowerDiameter	5
UpperDiameter	37
UserInformation	Please set gripper to position 1:1
Position2	

xx1800001023

As a rule, **UpperDiameter** and **LowerDiameter** are to be placed so that the gripper ranges for the different positions are positioned adjacent to each other, but do not overlap.

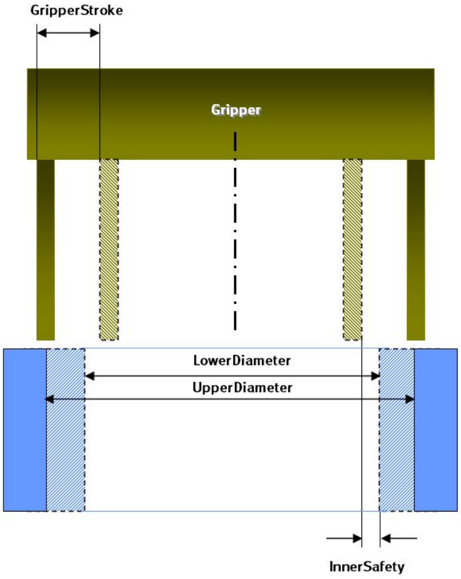
Continues on next page

The relationship between the parameters for a gripper range during external gripping is illustrated in the image below:



xx1800000320

The relationship between the parameters for a gripper range during internal gripping is illustrated in the image below:



xx1800000321

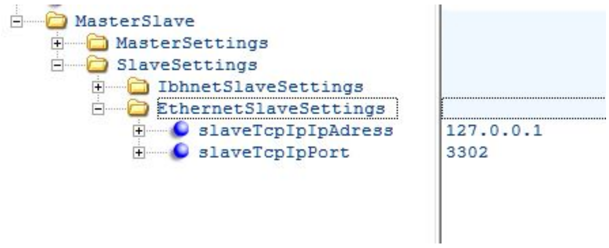
Continues on next page

Operation with external operator panel

MasterSlave.xml

If a FlexLoader Vision Lite is controlled from the external operator panel, FlexLoader Vision must be configured as **Slave (through TCP(IP)**.

Then the configuration file **MasterSlave.xml** must contain valid data for the IP address of the external panel as well as the port it is to communicate on.



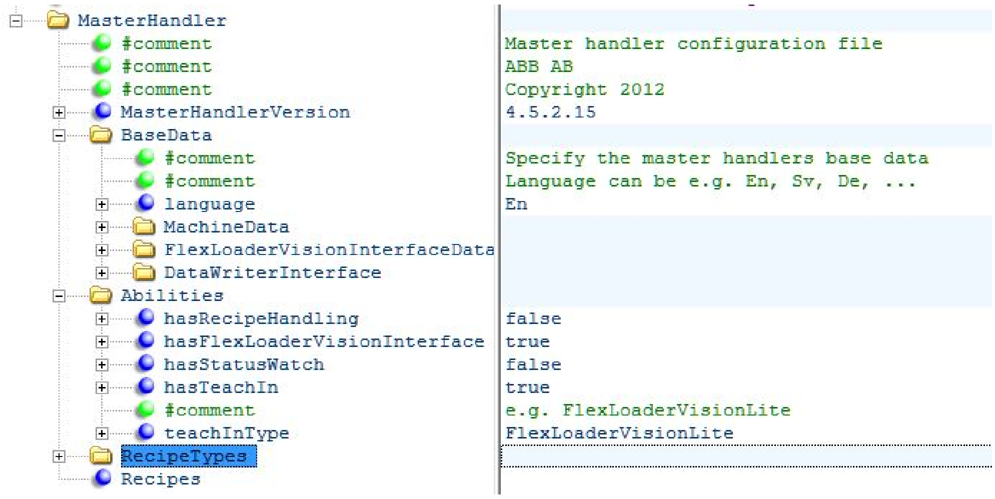
```
MasterSlave
  MasterSettings
  SlaveSettings
    IbhnetSlaveSettings
    EthernetSlaveSettings
      slaveTcpIpIpAdress          127.0.0.1
      slaveTcpIpPort              3302
```

xx1800000528

MasterHandler.xml

If a FlexLoader Vision Lite is controlled from the external operator panel, the following parameters must be changed in **MasterHandler.xml** on that external operator panel.

- Specify the language as per requirements.
- Specify the IP address of the FlexLoader Vision computer as well as the port number (same if specified in **MasterSlave.xml**).
- The values for the **hasFlexLoaderVisionInterface**, **hasTeachIn** and **teachInType** parameters are specified according to the image below.



```
MasterHandler
  #comment                          Master handler configuration file
  #comment                          ABB AB
  #comment                          Copyright 2012
  MasterHandlerVersion              4.5.2.15
  BaseData
    #comment                        Specify the master handlers base data
    #comment                        Language can be e.g. En, Sv, De, ...
    language                        En
    MachineData
    FlexLoaderVisionInterfaceData
    DataWriterInterface
  Abilities
    hasRecipeHandling               false
    hasFlexLoaderVisionInterface    true
    hasStatusWatch                  false
    hasTeachIn                      true
    #comment                        e.g. FlexLoaderVisionLite
    teachInType                     FlexLoaderVisionLite
  RecipeTypes
  Recipes
```

xx1800000970

FlexLoaderVisionLite.xml

The **FlexLoaderVisionLite.xml** file must be configured. If an external operator panel is used, this file must be available for both FlexLoader Vision and **MasterHandler**.

Continues on next page

Backup

Note

Save the backup files in a safe place regularly, to protect yourself from data losses.

FlexLoader Vision Lite

The **FlexLoaderVisionLite.xml** file is backed up in two stages each time it is saved. The backup copies are saved in the same place (**Configuration**) as the main file.

Robot

Regularly create a backup of the robot. Save the file in a safe place, to protect yourself from data losses.

Communication variables robot

Variable mapping

Writing of variables to the robot is done according to the following diagrams:

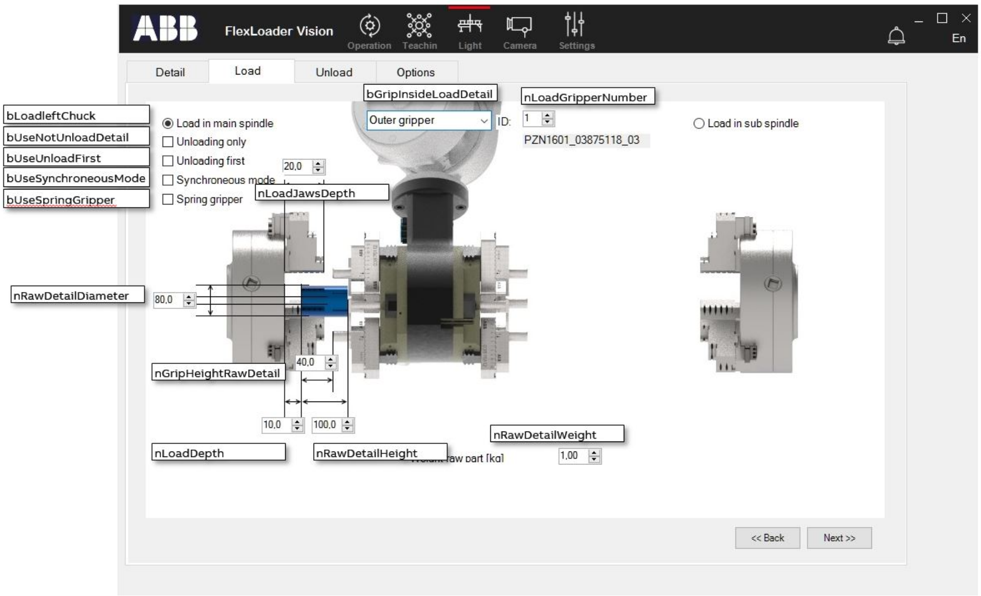


ABB FlexLoader Vision

Detail | Load | Unload | Options

● Load in main spindle
□ Unloading only
□ Unloading first
□ Synchroneous mode
□ Spring gripper

Outer gripper ID: 1
PZN1601_03875118_03

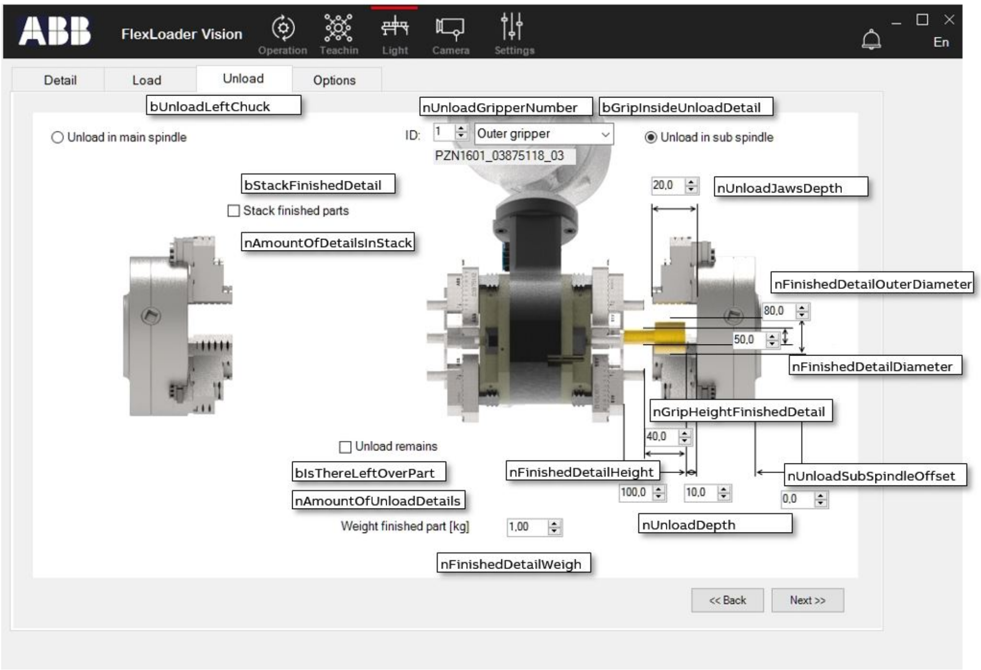
○ Load in sub spindle

20,0
80,0
40,0
10,0
100,0
Weight raw part [kg] 1,00

<< Back | Next >>

xx1800000289

Continues on next page



○ Unload in main spindle

ID: 1 Outer gripper

● Unload in sub spindle

PZN1601_03875118_03

□ Stack finished parts

20,0

80,0

50,0

40,0

□ Unload remains

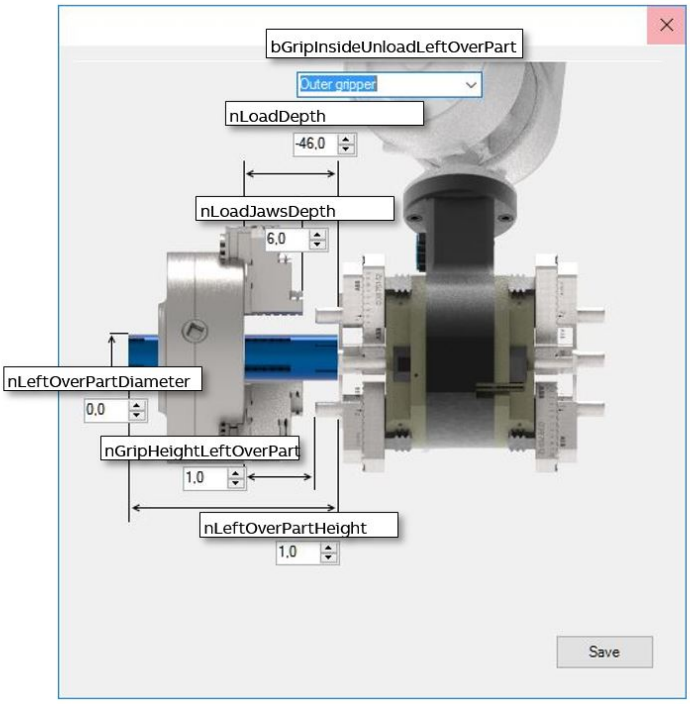
100,0

10,0

0,0

Weight finished part [kg] 1,00

xx1800000290



Outer gripper

-46,0

6,0

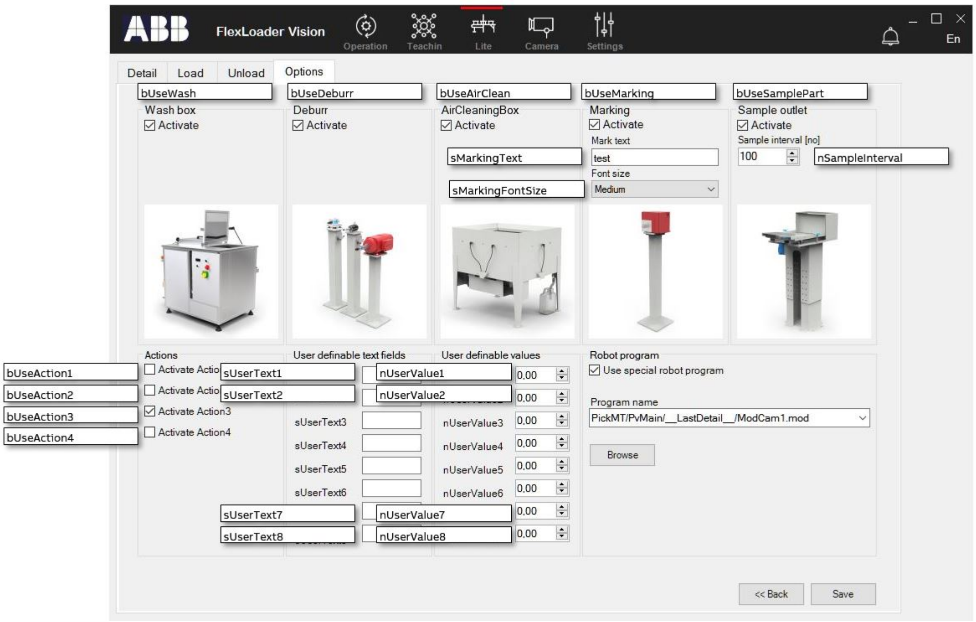
0,0

1,0

1,0

xx1800000291

Continues on next page



bUseWash — Wash box: ☑ Activate

bUseDeburr — Deburr: ☑ Activate

bUseAirClean — AirCleaningBox: ☑ Activate

bUseMarking — Marking: ☑ Activate

sMarkingText — Mark text: test

sMarkingFontSize — Font size: Medium

bUseSamplePart — Sample outlet: ☑ Activate

nSampleInterval — Sample interval [no]: 100

Actions

- bUseAction1 — □ Activate Actio
- bUseAction2 — □ Activate Actio
- bUseAction3 — ☑ Activate Action3
- bUseAction4 — □ Activate Action4

User definable text fields		User definable values	
sUserText1		nUserValue1	0,00
sUserText2		nUserValue2	0,00
sUserText3		nUserValue3	0,00
sUserText4		nUserValue4	0,00
sUserText5		nUserValue5	0,00
sUserText6		nUserValue6	0,00
sUserText7		nUserValue7	0,00
sUserText8		nUserValue8	0,00

Robot program

☑ Use special robot program

Program name: PickMT/PvMain/__LastDetail__/ModCam1.mod

xx1800000292

A Technical information

A.1 File structure in FlexLoader Vision

Introduction

FlexLoader Vision is usually installed in the following folder:

D:\Program Files\ABB Industrial IT\Robotics IT\FlexLoader Vision

The installation folder for FlexLoader Vision always contains the following folders:

Backup, **Bin**, **Configuration**, **Data**, **Database** and **Documentation.**

In this document, these subfolders are referred to as the bin-folder or the backup-folder etc. This always refers to the whole file path. For example, the bin-folder refers to **D:\Program Files\ABB Industrial IT\RoboticsIT\FlexLoader Vision\Bin.**

FlexLoader Vision\Backup

This folder is the automatically preset folder for saving backups.

Note

Avoid storing the backup file on the same hard disk as the software is installed on.

FlexLoader Vision\Bin

In this folder, all the executable files required to operate FlexLoader Vision are stored, such as FlexLoader Vision.exe and a number of dynamic libraries (***.dll**) and configuration files (***.xml**). There is also the **FlexLoader Vision log file (FlexLoaderVision_date.log)**, which contains valuable information to ABB if there are any problems.

When updating, some of the files in this folder may need to be replaced.

FlexLoader Vision\Configuration

In this folder, many of the different configuration files required for FlexLoader Vision are stored, such as the camera calibration files (***.cal**), the language file (**Language.xml**), the Master/Slave control file (**MasterSlave.xml**), the FlexLoader Vision configuration file (**FlexLoaderVision.cfg**), the camera configuration files (***.dcf**), and some system images (***.tif**). The latest applicable configuration file (**FlexLoaderVision.cfg.LastGoodKnown**) is also stored here. It can be used if a change has been made in FlexLoader Vision or if a system crash has made **FlexLoaderVision.cfg** unusable.

When updating, some of these files may need to be replaced.

For special applications, parts of the language file can be replaced by user-supplied definitions. These modifications can be placed in the file **Language.Override.xml.**

If the system has user-defined mechanical equipment, the files **UserDefinedPlcCamera_1.xml - UserDefinedPlcCamera_4.xml** and **UserDefinedPlcImageCamera_1.bmp - UserDefinedPlcImageCamera_4.bmp** are

Continues on next page

also found here. These contain the definition of the interface for the PLC of the user-defined equipment.

Extended user-defined mechanical equipment data can be defined in the files **UserDefinedPlcCamera_1_Tab2.xml - UserDefinedPlcCamera_4_Tab2.xml** and **UserDefinedPlcImageCamera_1_Tab2.bmp - UserDefinedPlcImageCamera_4_Tab2.bmp**.

FlexLoader Vision\Data

In this folder, details and tools are stored. The tools are saved in the sub-folder **Tools**, while details are saved in sub-folders corresponding to the group name. There is a ***.Detail** file for each detail, and for each position a ***.n.CollisionImage** and a ***.n.TeachinImage**, where * represents the detail name and n represents the position number.

If the secondary search has been activated for a detail, there is also a ***.Secondary.Detail-file**.

If a 3D camera is used, there is also a ***.HeightAnalysisArea**-file.

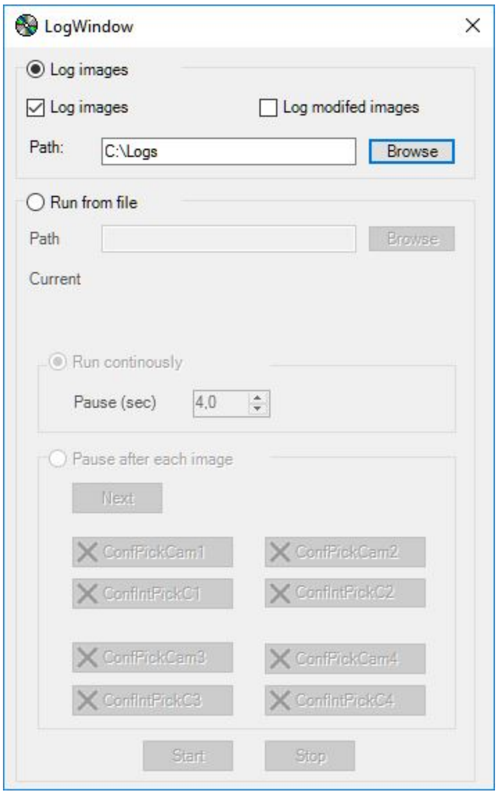
FlexLoader Vision\Database

In this folder, the FlexLoader Vision database (**FlexLoaderVision.mdb**) is stored. When FlexLoader Vision is run, there is also a **FlexLoaderVision.ldb** file.

A.2 Logging the operation

Logging of images while running

In some rare cases there may be a need to log images while running, for example in the event of very infrequent faults. A special log window must be activated to save each individual image to the hard disk during operation. This is done by first clicking the **Operation** icon and then pressing Ctrl+L. The following window opens:



xx1800001067

To activate logging, select **Log images** and enter the search path to the location where the images are to be saved. Then close the window and start operation in the normal way.

The images are saved on a rolling basis with a maximum of 19,999 images. The logging affects the cycle times, because saving to file takes time.

The image files are saved in the format **CAMX_YYYYY.tif**, where X is the camera number (0 .. 3) and YYYYY is a serial number.

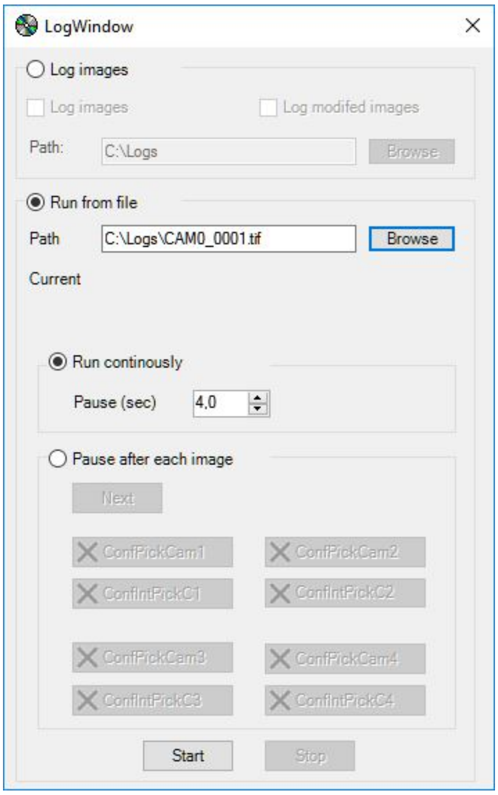
Operating with logged images (only 2D)

Images that were logged as above can then be used to run from file.

To run a detail with logged images, the detail must first be selected in the usual way in the operating window. Then the special log window must be activated. This is done by first clicking the **Operation** icon and then pressing Ctrl+L. Select **Run**

Continues on next page

from file and specify the search path for the first image. Then start and stop the operation with the **Start** and **Stop** buttons.



○ Log images
☐ Log images ☐ Log modifed images
Path: C:\Logs
● Run from file
Path C:\Logs\CAM0_0001.tif
● Run continously
Pause (sec) 4,0
○ Pause after each image

xx1800001068

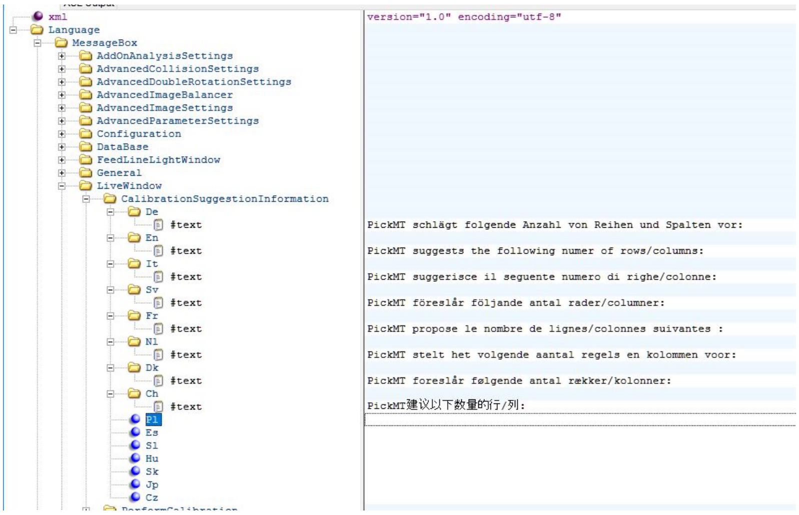
The slide show can either occur at a fixed interval (select **Run continuously**) or by pressing the **Next** button (select **Pause after each image**).

A.3 Localization

General structure

Text visible on the user interface is localized in the file **Language.xml**.

Each text block is represented and has placeholders for several languages.



```
xml                                              version="1.0" encoding="utf-8"
Language
  MessageBox
    AddOnAnalysisSettings
    AdvancedCollisionSettings
    AdvancedDoubleRotationSettings
    AdvancedImageBalancer
    AdvancedImageSettings
    AdvancedParameterSettings
    Configuration
    DataBase
    FeedLineLightWindow
    General
    LiveWindow
      CalibrationSuggestionInformation
        De
          #text                                  PickMT schlägt folgende Anzahl von Reihen und Spalten vor:
        En
          #text                                  PickMT suggests the following numer of rows/columns:
        It
          #text                                  PickMT suggerisce il seguente numero di righe/colonne:
        Sv
          #text                                  PickMT föreslår följande antal rader/columner:
        Fr
          #text                                  PickMT propose le nombre de lignes/colonnes suivantes :
        Nl
          #text                                  PickMT stelt het volgende aantal regels en kolommen voor:
        Dk
          #text                                  PickMT foreslår følgende antal rækker/kolonner:
        Ch
          #text                                  PickMT建议以下数量的行/列:
        Pl
        Es
        Sl
        Hu
        Sk
        Jp
        Cz
      PerformCalibration
```

xx1800000505

The text blocks are arranged in functional groups for the various user interface windows and messaging areas.

User defined PLC, SCADA and Safety content is localized in the respective configuration files.

Language override

For any specific project, an override text can be defined in a separate file, **Language.Override.xml**.

Elements that are to be overridden must adhere to exactly the same XML structure as used in the main language file.

Adding languages

Currently, FlexLoader Vision is prepared for the selection of a number of languages, although not all languages are actually translated.

These selectable languages are German, English, Swedish, Dutch, French, Italian, Danish, Chinese, Spanish, Finnish, Norwegian, Polish, Slowene, Hungarian, Czech, Slowak, Japanese.

By adding translated text to the corresponding language placeholders in the file **Language.xml**, these texts will be shown on the user interface. Missing translations will be replaced by their english default.

Adding additional languages that are currently not selectable can be done on request.

A.4 Manually restoring a backup

In certain cases, automatic restoring of a backup from the **Service** tab may fail, for example if the files are damaged or if the backup was created using a previous version. In these cases a backup can be restored manually.

> **Note**
>
> Errors that occur during manual management can make the system unusable and require remedial action from ABB's side.

How to do a manual backup:

1. Find the folder that contains the backup files. Then search for FlexLoader Vision's program folder. The folders **Database**, **Data**, and **Configuration** are found in both the backup folder and FlexLoader Vision's program folder.
2. Make a safety copy of the whole of FlexLoader Vision's program folder.
3. Copy the whole **Database** folder from the backup folder to FlexLoader Vision's program folder. Overwrite the existing files.
4. Delete the whole **Data** folder from FlexLoader Vision's program folder.
5. Copy the whole **Data** folder from the backup folder to FlexLoader Vision's program folder.
6. Copy the following files from the **Configuration** folder to the **Configuration** folder in FlexLoader Vision's program folder:
   - All calibration files (***.cal**)
   - All configuration files (***.xml**)
   - **FlexLoaderVision.cfg**
   - All camera configuration files (***.dcf**)
   - All image files (***.tif** and ***.bmp**)
7. Copy all configuration files (***.xml**) from the **Bin** folder to the **Bin** folder in FlexLoader Vision's program folder.
8. Copy the whole **OpcUaServer** folder from the backup folder to FlexLoader Vision's program folder.

B FlexLoader RAPID reference

B.1 FlexLoader data type prefix

Variable prefixes and naming

The following prefixes are used for data types.

- b (bool), e.g. bRobotStandsStill
- btn (btnres), e.g. btnResponse
- c (clock), e.g. clkInPosition
- DOF (signaldo on virtual I/O), e.g. DOF_RunFeeder1
- i (intnum), e.g. iOutletInPosition
- io (iodev), e.g. ioSicMarker
- jt (jointtarget), e.g. jtSoftwareSyncPos
- n (num), e.g. nDetailInGripper1
- p (robtarget), e.g. pViaMachine1
- s (string), e.g. sModuleCam1
- di (signaldi), e.g. diSampleOutletInPositionSensor
- do (signaldo), e.g. doRunFeederOut
- t (tooldata), e.g. tCalibTool1
- v (speeddata), e.g. vLowSpeed
- w (wobjdata), e.g. wCamera1

Several seldom used data types do not obtain their own prefix, e.g. shapedata, errnum, wzstationary. Variables inside records do not use prefixes either.

Variables are named with prefix and a descriptive name, where start of new words is indicated with capital letters (see examples above).

Constant prefixes and naming

Constants do not have any prefix. They are named with capital letters, with words divided by “_”. Examples are LOW_AIR_PRESSURE and STATE_LOAD_MAIN.

B.2 FlexLoader Vision interface

Overview

The FlexLoader Vision interface consists of the following modules.

Module	Description
VisionCom.mod	Background task for base communication with FlexLoader Vision.
Vision.sys	Base communication with FlexLoader Vision and interface towards the application RAPID code.
Calibration3D.sys	Support module used when calibrating FlexLoader Vision with a 3D camera.
VisionSlave.mod	Background task used for external control of FlexLoader Vision by means of a master system.
VisionSlaveControl.sys	Help module loaded in all tasks if this option selected. Makes it much easier to handle the case where robot it self is the master of FlexLoader Vision.
VisionControlImage.sys	Functions for using the original camera to grab a second control image and then return settings to normal again. Loaded in all motion tasks.

VisionCom.mod

This module contains routines needed for background communication between the robot and FlexLoader Vision and must not be changed.

VisionCom synchronizes the vision system and conveyor events with robot program execution. It relieves the foreground task from handling feeding devices and coordinate reception from FlexLoader Vision.
Important routines

Routine	Description
BeltReadyTrap1..4	Handles notifications from the feeder system.
CoordTrap	Receives all coordinate and result information from FlexLoader Vision.

Continues on next page

Vision.sys

This module contains some routines needed for communication between the robot and FlexLoader Vision and must not be changed. It is loaded to all motion tasks and to VisionCom task.

Normal use

Important data

Routine	Description
BELT_ACTION{4}	Must be set according to the desired system behavior for each camera. Normally set from **InitializeCamX** in **ModCamX**. • **RUN_ONE_DETAIL** (default): Run the conveyor belt directly after the last part has been picked. • **RUN_NO_DETAIL**: Run the conveyor belt only if no parts have been found in the image. • **RUN_NEVER**: Never run the conveyor belt. • **RUN_ALWAYS**: Run the conveyor belt after each pick. **Note** When running with the setting **RUN_NEVER**, e.g. when picking from pallets: If FlexLoader Vision does not find anything after taking a new image, the robot must have some type of management for this. The user must check the action sent in the vision result to decide what to do with the result. If action value is equal to 1 (**NO_DETAIL**) the appropriate action must be taken, for example by taking a new image several times before the cell is stopped. If **SetNextTarget** is called up when the action value is 1, the robot program stops.
ALLOW_AUTO_GRAB{4}	Must be set according to the desired system behavior for each camera. Normally set from **InitializeCamX** in **ModCamX**. • **TRUE**: When the belt has been run and the **InPosition** signal set a new image is taken by the automatic device. • **FALSE**: Imaging must be explicitly initiated by the robot program by calling up **GrabImage**.

Important routines

Routine	Description
PROC ConfirmPick1..4()	If **SetNextTarget** returns valid grip information, **ConfirmPick** must be called by the user to run the conveyor and complete the started grab-pick sequence. This usually is done by calling **RefPosOut**. **Note** One **GrabImage** must always be confirmed with **SetNextTarget** and **ConfirmPick** before the next **GrabImage** is started.
PROC DiscardAndTakeNewImage(num grabCamera)	Called up by the user if a new image is to be taken in FlexLoader Vision before all sent coordinates are used. **Note** Routines such as **DiscardAndTakeNewImage()** are to be used with care. If the normal **GrabImage-SetNextTarget-ConfirmPick** chain is deviated from, it is extremely important to have full control of all image taking, to prevent that double images are taken (a new image is taken before the previous image has been processed).

Continues on next page

Routine	Description
PROC DiscardAndStartBelt(num grabCamera)	Called up by the user in order to start the feeder before all sent coordinates are used. **Note** Routines such as **DiscardAndStartBelt()** are to be used with care. If the normal **GrabImage-SetNextTarget-ConfirmPick** chain is deviated from, it is extremely important to have full control of all image taking, to prevent that double images are taken (a new image is taken before the previous image has been processed).
PROC DiscardCoordinates(num grabCamera)	Discards current coordinates which have not been used yet. **Note** Routines such as **DiscardCoordinates()** are to be used with care. If the normal **GrabImage-SetNextTarget-ConfirmPick** chain is deviated from, it is extremely important to have full control of all image taking, to prevent that double images are taken (a new image is taken before the previous image has been processed).
FUNC visionres GetNewVisionResult(num Camera,\num DesiredPosition,\num MaxImageRetries,\num TimeoutTimeVision)	This function waits for coordinates from vision and retrieve the result. Then the result is returned within a visionres data. If nothing found this function will retry a new image or run belt again if there is a belt. **Note** This routine is handling grabbing (but still possible to do it in advance too). It also calls SetNextTarget to receive results and do a first evaluation of what to do. If user wants, this routine could do image retires too.
PROC GrabImage(num grabCamera)	Called up by the user to take a new image with the selected camera. The routine waits until the accompanying signal **InPosition** is 1 and sends a command to FlexLoader Vision to take a new image with **nCamera**. **Note** Do not use **GrabImage** if the camera is set to **ALLOW_AUTO_GRAB = TRUE** (set in **InititializeMain** in **MainModule**). New images will be requested automatically after **ConfirmPick1..4()**.
PROC InitVision()	Called at the beginning of the program to initialize the communication with FlexLoader Vision.
FUNC num SendPvStatus()	Returns current operating status in FlexLoader Vision (**PV_IDLE**, **PV_OPERATION**, **PV_STARTING**, **PV_STOPPING**).
PROC SetNextTarget(nCamera,\nDesiredPosition)	Called up to retrieve the next valid grip position from FlexLoader Vision for camera **nCamera**. The function returns when the coordinates for a valid part that can be gripped are available. It is possible to specify that only one selected teachin position in FlexLoader Vision is to be returned.
PROC StopVisionSystem()	Stops FlexLoader Vision at the robot’s request.

Continues on next page

Routine	Description
FUNC string VisionInputBox()	Can be called up to write an input query on the FlexLoader Vision user interface. The function returns the reply that the operator has entered.
FUNC num VisionMessageBox()	Can be called up to write a message on the FlexLoader Vision user interface and read off the operator's reply (**Ok**, **Yes**, **No**).
PROC WriteLog()	Writes a message directly to the FlexLoader Vision log file.

Further routines

There are several other routines that can be useful. Most of them are found below. Some routines are only help routines and are only called up by other routines in **Vision.sys**. These are not mentioned here.

Routine	Description
PROC CameraContrast(nCamera, nValue)	Called up to change the camera contrast.
PROC CameraExposureTime(nCamera, nValue)	Called up to change the exposure time in the camera.
PROC CameraGain(nCamera, nValue)	Called up to change camera brightness.
FUNC bool checkCoordinates(num pickAreaWidth,num maxMovementHeightUnderCamera,\num pickAreaLength,\robtarget CheckTarget)	This routine checks that received FlexLoader Vision coordinates is within maximum picking area. Also controls that given pick offset is ok.
PROC ClearCoordinates()	For system internal usage.
PROC DisablePosition(nCamera, nPosition)	Called up to deactivate a certain teachin position in FlexLoader Vision. All positions can be locked at the same time by specifying a value **nPosition** > 999.
PROC EnablePosition(nCamera, nPosition)	Called up to activate a certain teachin position in FlexLoader Vision. All positions can be enabled at the same time by specifying a value **nPosition** > 999.
PROC ForceInPosition()	For system internal usage.
FUNC bool isPositionPresent(num cam,num desiredPosition)	Called up to check whether a certain position is among the results from FlexLoader Vision.
FUNC num numPositionPresent(num cam,num desiredPosition)	Called up to check how many times a certain position occurs in the results sent by FlexLoader Vision.
PROC ResetAlarm(sMessage)	Called up to reset a certain alarm that was set in FlexLoader Vision from the robot.
PROC ResetInformation(sMessage)	Called up to reset a specific information message that was set in FlexLoader Vision from the robot.
PROC ResetWarning(sMessage)	Called up to reset a certain warning that was set in FlexLoader Vision from the robot.

Continues on next page

Routine	Description
PROC SendBlackRegion(sBlackRegion)	Sends a reconfiguration to FlexLoader Vision to concentrate the camera's field of view on the selected areas. The user can either make a rectangular area invisible (=black image), or make the area outside a selected rectangle invisible. The areas are specified in the format A-BBB-CCC-DDD-EEE-F, as follows: A indicates camera number (1–4). BBB indicates in percent where the rectangle starts in the x-axis (000–100). CCC indicates in percent where the rectangle ends in the x-axis (000–100). DDD indicates in percent where the rectangle starts in the Y-axis (000–100). EEE indicates in percent where the rectangle ends in the Y-axis (000–100). F indicates whether the actual rectangle becomes black (1) or if the surrounding area becomes black (0).
PROC SendClearSend()	For system internal usage.
PROC SendConfirmPickCam1...4()	For system internal usage.
FUNC bool SelectDetailOnTheFly(nCamera, sGroupName, sDetailName)	For advanced integration only. Used to change part during operation. **Note** Use this functionality with care. Ensure that all other processes are finished before calling SelectDetailOnTheFly, else unpredictable behaviour or system crashes might occur.
PROC SendEdgeHeight(nCamera, nEdgeHeight)	Called up to set a new edge height in the selected camera. Usually used when running stacks on the belt.
PROC SendGrab(nCamera)	For system internal usage.
PROC SendStop()	For system internal usage.
PROC SetAlarm(sMessage)	Called up to set an alarm in FlexLoader Vision from the robot.
PROC SetInformation(sMessage)	Called up to set an information message in FlexLoader Vision from the robot.
PROC SetWarning(sMessage)	Called up to set a warning in FlexLoader Vision from the robot.
PROC StartBelt(nCamera)	Starts the belt under the selected camera. Usually only used from **VisionCom**.
FUNC bool WriteFileVisionPCFlexLoader Vision(sFileName, sMessageData)	Called up to write a text to the selected file name on the FlexLoader Vision computer.

Calibration3D.sys

This module contains routines needed for the calibration of 3D cameras.

Important routines

Routine	Description
PROC Calibration3DRoutine()	This routines performs the actual calibration in cooperation with FlexLoader Vision.
PROC Calibration3DUpdatePositions()	Support routine for convenient update and control of all calibration positions.

Continues on next page

VisionSlave.mod

This module contains routines and data that is used to control FlexLoader Visionthrough robot I/O signals.

Important routines

Routine	Description
TRAP AutoStartTrap	Handles Start requests from external master and starts FlexLoader Vision.
TRAP CycleStopTrap	Handles Stop requests from external master and stops FlexLoader Vision.
PROC main()	Execution start of the task, initiate and then just wait for the traps.
TRAP SelectIdTrap-Cam1..4	Handles part selection requests from external master and changes part in FlexLoader Vision.
PROC Init()	Initializes the various request handlers.

VisionSlaveControl.sys

This module provides some routines that makes it easier for robot to act as master directly to FlexLoader Vision. The routines in this modules connects to VisionSlave functionality to handle this.

Important routines

Routine	Description
PROC VisionControl-Start()	Starts FlexLoader Vision.
PROC VisionControl-Stop()	Stops FlexLoader Vision.
PROC VisionControl-PartSelect(\dnum Part-Camera1|dnum Part-Camera2|dnum Part-Camera3|dnum Part-Camera4)	Changes part selection for a selected camera in FlexLoader Vision. This routine only changes part in one selected camera.
PROC VisionControl-ChangeParts(\dnum PartCamera1,\dnum PartCamera2,\dnum PartCamera3,\dnum PartCamera4)	Changes parts in more than one camera. Internally uses Vision-ControlPartSelect as many times as needed.

Continues on next page

VisionControlImage.sys

This module provides some routines to prepare the standard camera for a second control image. After the control image is grabbed and analyzed, camera is prepared for normal image again. Normal operation is that robot picks from camera area after getting the normal results of where to find part. Then robot send command to prepare camera for second image, moves closer to camera and while holding the part asks to grab the second control image. After the vision result robot evaluates and do necessary actions depending on the result and then prepare the camera for normal operation again.

Important routines

Routine	Description
FUNC visionres ControlImage(num CameraNumber,\num ControlImagePositions{*},\num NormalImagePositions{*},\camerasetting CameraSettingsControl,\camerasetting CameraSettingsNormal,\string BlackRegionValuesControl,\string BlackRegionValuesNormal,\num MaxImageRetries)	This function is the only thing needed to be called from normal user program. It prepares camera for second image, call routine to get robot in position, grabs image, call routine to get the robot out of camera area, prepare for normal image and then returns the result of the second image. **Note** This routine will be depending on two movement routines that must be created inside the part module file. Make sure there is a MoveToControlImageX routine that moves robot to camera X second image position and a MoveFromControlImageX that moves away out of camera X area after image. These instructions should be purely movements. X is the number of the camera which should be used for the second image.
PROC PrepareNormalImage(num CameraNumber,\num PositionsToUse{*},\num InitBeltAction,\bool InitAllowAutoGrab,\camerasetting CameraSettingsNormal,\string BlackRegionValues)	Prepares camera for a normal image. This routine is called from within ControlImage, but could also be used from user program like in the initialization phase.
PROC PrepareControlImage(num CameraNumber,\num PositionsToUse{*},\camerasetting CameraSettingsControl,\string BlackRegionValues)	Prepares camera for the second control image. This routine is called from within ControlImage, but could also be used from user program like in the initialization phase.

B.3 FlexLoader application functionality

Overview

The FlexLoader Vision application functionality handles application and part specific aspects of the robot cell.

The integrator is responsible for adapting this functionality to the actual application.

Module	Description
Main.pgf	Main program automatically loaded by FlexLoader Vision.
MainModule.mod	This module contains the overall and part independent logic for the robot cell. Loaded to all motion tasks.
PartCam1..4.mod	This module provides part-specific application code. This module is further specified for parametrized applications (e.g. FlexLoader Vision Lite) and generic applications. In the case of FlexLoader Vision Lite only one camera is possible and the module for this is named LitePartCam1.mod. Used in all motion tasks.
Common.sys	This module contains general tool data, event routines, world zones, configuration routine and more which is general for all kinds of cell types. Could be used in all motion tasks but might need modifications for robot two and up.
MoveRobotVia.mod	This module handle robot movement between via positions (general movements in the cell). Could be used in all motion tasks but might need modifications for robot two and up.

MainModule.mod

MainModule.mod is always supplied with FlexLoader Vision. It is the main robot program of the system. MainModule.mod provides the following routines:

Routine	Description
PROC CheckSystem()	Checks whether anyone has requested entry to the cell or has been inside the cell. The routine also ensures that new images are taken when the cell is started after someone has been inside. This happens as it can no longer be guaranteed that the part remains under the camera.
PROC ControlImageHandling(num CameraNumber)	Example routine of how to use the ControlImage functionality. This routine should only be used when second control image is needed.
PROC InitializeMain()	Contains the necessary initialization for the main program.
PROC LoadCameraModules()	Loads relevant module files for each camera. The name of each module that is to be loaded is provided by FlexLoader Vision.
PROC Main()	Start of program, a base structure is prepared. Integrator could add or modify if needed.
PROC PickPartAtCamera(num CameraNumber\num DesiredPosition,\num MaxImageRetries,\num TimeoutTimeVision)	Called when the robot is to retrieve a new part by the camera. This routine calls up several sub-routines to request new coordinates and perform movements to and from the pick position.

Continues on next page

Routine	Description
FUNC num PickMultiPartsAtCamera(num CameraNumber,\num NumberOfPartsToPick,\num DesiredPositions{*},\num MaxImageRetries,\num TimeoutTimeVision)	Does the same as PickPartAtCamera with the addition that it also can handle to pick multiple parts from one image.

PartCam1.mod

PartCam1.mod is always supplied with FlexLoader Vision. It handles the part specific programming, e.g. loading, unloading, marking, air cleaning.

Two example modules are delivered, one for operation with simplified TeachIn (FlexLoader Vision Lite) which is named LitePartCam1.mod, the other for standard operation with standard TeachIn. Refer to chapter on robot program.

Standard functionality

Routine	Description
PROC Cam1Position_n()	Part specific handling routine for camera 1 depending in which position FlexLoader Vision identified the part. Note, this routine is not always used in the example routines.
PROC InitializeCam1()	Initializes the camera and part specific data.
PROC LeaveFeeder_OUT()	Leaves a part on an out belt.
PROC LoadMachine()	Loads a part into the machine, multiply routine if several machines.
PROC MainRoutine1()	Handles the main flow of the cell together with main(). This include to check status of machines and so on to figure out what to do next. All these kind of selections and checks that are part specific is placed in this routine. The rest is placed in either of the routines depending on how the integrator wants the structure.
PROC MoveToControlImage()	Moves robot to control image position. Only used if need of second control image.
PROC MoveFromControlImage()	Moves robot from control image position. Only used if need of second control image.
PROC PickCam_1()	This routine performs the actual picking of a part from the conveyor at camera position.
PROC PickMultiPartsCam_1(num PartNo,num Position)	This routine does the same as PickCam_1 but for the case of picking multiple parts from one image.
PROC RefPosInCam_1()	This routine is called to approach an intermediate position towards the conveyor before picking. Update this position to suite the current application.
PROC RefPosOutCam_1()	This is an intermediate position used when leaving the picking area. Update this position to suite the current application.
PROC StopRoutineCam1()	This procedure is executed when FlexLoader Vision is stopped.
PROC UnloadMachine()	Unloads a part from the machine, multiply routine if several machines.

Continues on next page

Additional functionality for FlexLoader Vision Lite use

LitePartCam1.mod for FlexLoader Vision Lite applications contains extended templates and lathe specific RAPID code than allows for a fully parametric teachin.

Specifically, the MainRoutine1 calls getState() to understand how the machine should be handle in the specific case and what to do at the moment. Local routines then cover all allowed combinations for loading and unloading to main and sub spindel, e.g. LoadMachine_MAIN_UnLoad_SUB, UnloadLeftOverPartMAIN, LoadMachine_SUB, UnloadMachine_MAIN_Load_MAIN.

A couple of routines is handling calling actions for selected options. Those might need changes or additions.

Routine	Description
PROC PerformFinishingOptions()	Operations that shall be done with the part after unloading from machine tool. Change according application.
PROC PerformPrepareOptions()	Operations that shall be done with the part prior to loading into the machine tool. Change according application.

Machine tool handling procedures and functions

Note that the GLOBAL procedures (like LoadMachine_MAIN) are called from the MainRoutine and perform logical work, whereas the LOCAL routines (like LoadMAIN) perform the actual movement work.

Name	Description
LOCAL PROC EnterMachine(\switch Gripper1FaceMAIN|switch Gripper1FaceSUB)	This procedure is for moving the robot into the machine. Add positions here if necessary. In this example code, pViaMachine1/pViaMachine2 is a position inside the machine where both chucks are reachable and the gripper could turn around without crashing. pViaMachine1 is placed with gripper 1 facing the main chuck. pViaMachine2 is placed with gripper 1 facing the sub chuck.
LOCAL PROC ExitMachine(\switch Gripper1FaceMAIN|switch Gripper1FaceSUB)	This procedure is for moving the robot out of the machine. Add positions here if necessary. pViaMachine1/pViaMachine1 see above. In this example code, pViaMachine is a position just outside the machine tool where the robot can request the machine to close its door.
PROC LoadMachine_MAIN()	This procedure is for loading the main chuck in the machine tool.
LOCAL PROC LoadMAIN()	This procedure loads the part in the main chuck. Make sure to use EnterMachine before all loading/unloading and ExitMachine after all loading/unloading. The load position is always calculated. X and Y is set to 0 and Z is calculated with the help from FlexLoader Vision values. To change the leave position, adjust wMainChuckM1 by running the procedure CalibMainChuck.
PROC SynchroneousMode()	This procedure unloads the sub chuck and then waits for the machine to be ready before loading the main chuck and leaving on the out belt.

Continues on next page

Name	Description
LOCAL PROC UnloadLeftOverPart()	This procedure unloads left over part in main chuck. Make sure to use EnterMachine before all loading/unloading and ExitMachine after all loading/unloading. The unload position is always calculated. X and Y is set to 0 and Z is calculated with help from FlexLoader Vision values. To change the unload position, the work object has to be adjusted until the position is correct. In this case adjust wMainChuckM1 by running the procedure CalibMainChuck.
PROC UnloadMachine_LeftOverPart()	This procedure is for unloading the LeftOverPart from the machine tool.
PROC UnLoadMachine_MAIN()	This procedure is for unloading the main chuck in the machine tool.
PROC UnloadMachine_MAIN_Load_MAIN()	This procedure is for first unloading and then loading the main chuck in machine tool.
PROC UnLoadMachine_SUB()	This procedure is for unloading the sub chuck in the machine tool.
LOCAL PROC UnloadMAIN()	This procedure unloads the part in the main chuck. Make sure to use EnterMachine before all loading/unloading and ExitMachine after all loading/unloading. The unload position is always calculated. X and Y is set to 0 and Z is calculated with the help from FlexLoader Vision values. To change the unload position, the work object has to be adjusted until the position is correct. In this case adjust wMainChuckM1 by running the procedure CalibMainChuck.
LOCAL PROC UnloadSUB()	This procedure unloads the part in the sub chuck. Make sure to use EnterMachine before all loading/unloading and ExitMachine after all loading/unloading. The unload position is always calculated. X and Y is set to 0 and Z is calculated with the help from FlexLoader Vision values. To change the unload position, the work object has to be adjusted until the position is correct. In this case adjust wSubChuckM1 by running the procedure CalibSubChuck.
PROC UnLoadMachine_SUB_Load_MAIN()	This procedure is for first unloading the sub spindle and loading the main spindle in machine tool.

Common.sys

Data definitions, world zones, event routines and other stuff general to all cell types is stored in Common.sys. This system module is accessible by all programs, and by storing the data in this module all programs always have access to the right data. Normally there is no additions to this module. When an integrator has general definitions or other general data, that is preferably stored in CommonCell, CommonSC3000 or CommonSC6000 depending on current cell type.

Routine	Description
PROC AutoConfiguration()	Reads configured IO:s and loaded modules to set cell specific configuration switches.
FUNC bool CheckInPosition(num CameraNumber)	This routine checks if inconveyor is in position, i.e. ready to take a new image. It is only to be used if the cell use a in and doesn't use AUTO_GRAB of images.
PROC ClearToolsPartInfo()	Clears parta data for all tools for current calling robot.

Continues on next page

Routine	Description
PROC DefForbiddenZone()	Activates a user-defined area where the robot may never be. This must normally be positioned directly above the robot so that it cannot make backward bending movements.
PROC DefSafeZone()	Activates a user-defined area where it is safe for the robot to start its program from the start.
PROC DefWorldZones()	Called up automatically when the robot is switched on. From this routine **DefForbiddenZone**, **DefSafeZone** and **DefMachineZone** are called up.
FUNC mttargetdata getClosestZone()	Finds the closest defined zone to where the robot is located right now.
PROC PowerOnEvent()	This routine is called on power on event. Fill with suitable code if needed in application.
PROC QStopEvent()	This routine is called on quick stop event. Fill with suitable code if needed in application.
PROC RestartEvent()	This routine is called on restart event. Fill with suitable code if needed in application.
PROC StartEvent()	This routine is called on start event. Fill with suitable code if needed in application.
PROC StopEvent()	This routine is called on stop event. Fill with suitable code if needed in application.

MoveRobotVia.mod

The principal motion in the FlexLoader Vision is controlled by the MT_MoveRobotTo routine, which moves the robot from known zone positions to a target zone position. From these zone positions (via-positions), movement within certain areas of the robot cell can be initiated. MT_MoveRobotTo is defined within the FlexLoader Library Add-in but need some help from the MoveRobotVia module to manage it's tasks. In this module there are definitions of all zones and the pure movements between them. All other logic around this when to use which movement is within the FlexLoader Library Add-in.

CAUTION

When starting the robot from unknown positions, collisions can occur.

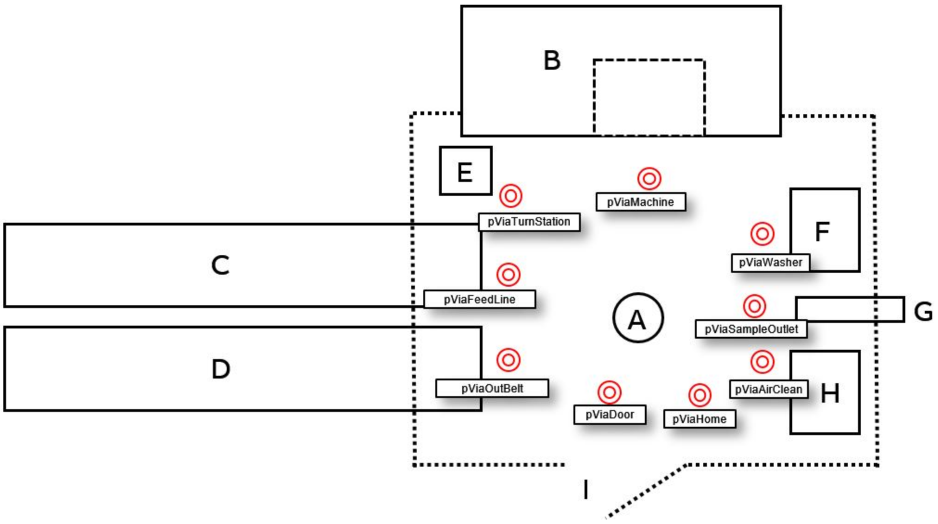
Routine	Description
PROC MoveToZoneMenu()	This will show a menu of all zones and the user has the possibility to select where robot should move.
PROC MT_MoveToVia(mttargetdata Target)	Help routine that moves directly to the via position of the zone passed as argument. User could add own code for special cases, for example if robot is inside a machine, a couple of extra positions might be needed first before moving to via position. Also called from built in FlexLoader Library Add-in routines.
PROC MT_UndefinedZoneHandler()	Help user solve problem if robot is standing in an undefined zone. Also called from built in FlexLoader Library Add-in routines.

Continues on next page

Routine	Description
PROC mvXXXXXX()	For each defined zone in the cell there must be a mv routine that moves the robot to that zone. Routine name must be exactly mvXXXXX where XXXXX is the name of the defined zone. For example; if mttargetdata ZONE_HOME is defined in the system then there must be a mvZONE_HOME routine in the MoveRobotVia module.
PROC Update_ViaPosition()	Make it easier to teach all via-positions. Only called in manual mode.

A schematic view of an example robot cell with typical via positions is shown in in the next figure.

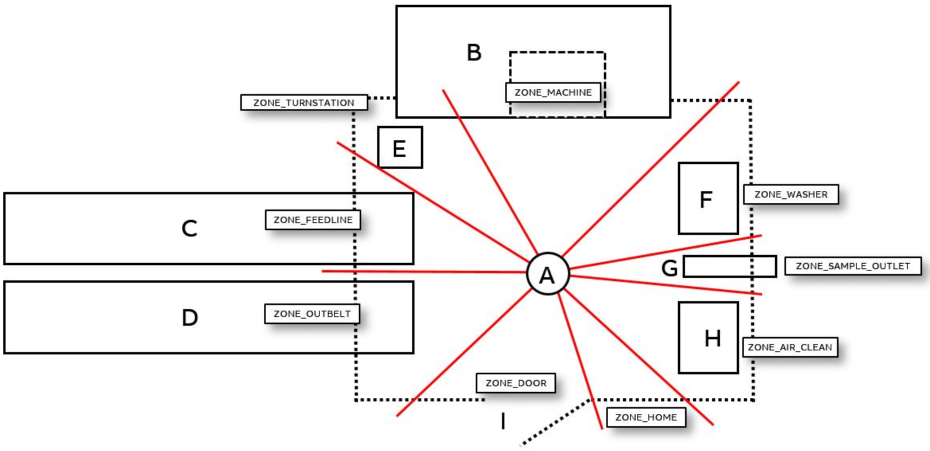
A schematic setup of the same robot cell with its typical zone setup is shown in figure thereafter.



xx1800002273

Pos.	Description
A	Robot
B	Machine tool
C	Inconveyor
D	OutConveyor
E	Turn station
F	Washer station
G	Statistical outlet
H	Air cleaning station
I	Door

Continues on next page



xx1800002272

B.4 FlexLoader Vision Lite functionality

Overview

The FlexLoader Vision Lite uses the part module LitePartCam1 as described earlier. In addition to that there is also an important system module which handles all communication variables that is needed to get correct information from FlexLoader Vision.

Module	Description
FlexLoaderVision-Lite.sys	Contains FlexLoader Vision Lite variables and calculation routines.

FlexLoaderVisionLite.sys

The module FlexLoaderVisionLite.sys contains a number of predefined procedures/functions that enables the parametrized loading and unloading of the machine tool.

The variable names in this module must not be changed, as FlexLoader Vision transfers data to these variables.

Routine	Description
FUNC num calcLoad-Compensation()	Calculate load compensation when leaving and picking from chuck.
PROC CalibLoadCom-pensation()	Calibrates load compensation constants for later use in calcLoad-Compensation.
FUNC pos calcPosLoad-Machine(\num nSpindleOffset)	Calculate load position inside machine.
FUNC pos calcPosUn-loadLeftOverPart(\num nSpindleOffset)	Calculate grip position for left over part.
FUNC pos calcPosUn-loadMachine(\num nSpindleOffset)	Calculate unload position inside machine.
FUNC bool Check-IfLiteBranchValidated()	Checks if current light branch has been tested and validated.
PROC CheckState()	Checks if current state is validated. If not, ask user how to proceed.
FUNC num getAllowed-NumberToStack()	Returns allowed number of details to stack on outbelt.
FUNC num getState()	Checks current machine and cell state. Also checks settings used in part teach in to calculate what to do next. Returns a state which describes next action for robot. This action is then called from the MainRoutine1 to handle the main flow.
PROC ValidateL-iteBranch()	Add current branch to validated array.

B.5 FlexLoader conveyor system control

Overview

The FlexLoader Vision conveyor system control is structured according to the following overviews.

These modules are part of the internal control system and must not be changed.

Module	Description
FeedLine.mod	Overall handling of conveyors and alarms connected to conveyor system. This module is loaded into a separate background task.
FeedLineInPosition.mod	Specialized task for handling start and stop of inconveyor with short response time, regular operation with InPosition-sensor. This module is loaded into a separate background task.
FeedLineToFar.mod	Specialized task for handling start and stop of inconveyor with short response time, regular operation with ToFar-sensor. This module is loaded into a separate background task.
FeedersSuperVision.mod	Specialized task for checking if operator has manually operated any conveyor. This module is loaded into a separate background task.
TpsView files	Optional files for control of conveyor system through the FlexPendant.

B.6 FlexLoader assistance and utility functionality

Overview

The FlexLoader assistance and utility functionality is structured according to the following overview.

Module	Description
AlarmsToVision.sys	Listen to events create by alarm system and sets and resets alarms to FlexLoader Vision when needed. Never touch this code! Loaded as hidden in T_ROB1.
EntryControl.mod	Module for handling entry request, entry indication lamp and door lock signals. Loaded into own background task.
FeederOut.sys	Functionality for control of outconveyor and leave pattern calculation. Could be used in all motion tasks.
GlobalCodeAndConfig.sys	Shared module containing global code for FlexLoader Vision RAPID code.
IndicationLights.mod	Module for handling indicator lights according to operational status. Loaded into own background task.
IndicationLights_shared.sys	Shared module for some task shared data and routines for indication light handling.
Tools_base.sys	Module for convenient and common tool handling. Loaded into all motion tasks.
Tools_shared.sys	Shared module for some task shared data and routines for tool handling.

EntryControl.mod

This background task monitors the entry request to the cell. As soon as an entry is requested, a signal is sent to the main program.

This task also ensures that the indication lamp for the entry request lights up/flashes correctly depending on the status of the machine. The lamp starts to flash when entry is requested and stays lit when entry is permitted. When the robot runs in normal operation, the lamp is turned off.

Routine	Description
PROC main()	Initiates entry control handling.

FeederOut.sys

The module FeederOut.sys provides routines for control of outconveyor and leave pattern calculation.

Routine	Description
FUNC pos CalcFeederOutLeaveOffset(\inout num presentLayer,\num feederNo,\bool leftOverPart)	This procedure is called from user program whenever a part should be placed on outconveyor. User should use the returned offsets for the leave position when leaving part and then call ConfirmLeaveFeederOut when finished. If no confirm, then robot will get the same leave offsets next time this function is called. If using SetUpFeederOutBelt the leave position pLeaveFeederOut is forced to x,y,z = 0,0,0 in this function! Never change that position name.

Continues on next page

Routine	Description
PROC CalibSpeedFeederOut()	Help procedure for calibrating out belt speed.
PROC ConfirmLeaveFeederOut(\num feederNo)	Call this to confirm that last part has been placed at the last coordinates that was given from CalcFeederOutLeaveOffset. Number of parts left at feeder out is updated and belt is started if current row is full.
FUNC bool hasPermissionToLeaveFeederOut(\num feederNo)	This function checks if robot has permission to leave on feeder out. If selected, it will also check if feeder out has moved and in that case re-initialize the feeder.
PROC RunFeederOut(\num feederNo,\num runLength,\num runTime)	This procedure runs feeder one part length + desired space space between parts. Also resets and initializes variables for feeder.
PROC SetUpFeederOutBasic(\num feederNo,num detailsPerRow,num leaveOffset,num feederRunTime,\bool leaveFromLeft)	Setup restrictions for out feeder belt and part. This procedure should be called once before start using RunFeederOut or CalcFeederOutLeaveOffset. This function is used for basic setup. For advanced setup, use SetUpFeederOutBelt and SetUpFeederOutDetail.
PROC SetUpFeederOutBelt(\num feederNo,num feederWidth,num feederSpeed,num spaceBetweenParts,num spaceToEdge,num yDistanceToLeave,\bool leaveFromLeft,\string RunFeederDOSignalName,\string AllowLeaveDOSignalName,\string FeederHasMovedDOSignalName,\string BlockEmptyingModeSignalName)	Setup restrictions for out feeder belt. This procedure should be called once before start using RunFeederOut or CalcFeederOutLeaveOffset.
PROC SetUpFeederOutDetail(\num feederNo,num detailWidth,num detailLength,num heightToTCP,\num allowedDetailsInHeight,\num detailHeight, \num xDistanceToTCP,\bool hasLeftOverPart,\num leftOverPartWidth,\num leftOverPartLength,\num leftOverPartHeightToTCP)	Setup restrictions for out feeder part. This procedure should be called once before start using RunFeederOut or CalcFeederOutLeaveOffset. The routine also calculates how many parts fit in each row and if a left over part space should be reserved.

Continues on next page

GlobalCodeAndConfig.sys

This is a module containing global support functions that could be used in any machine. It also includes configuration switches and global standard constant declarations.

CAUTION

Always make sure configuration switches are set to values matching the current cell. But also note that most of those switches is automatically set by AutoConfiguration().

Routine	Description
FUNC string AddTrailingBlanks(string Text,num WantedTextLength)	Returns the string Text but with added blanks to get to WantedTextLength length.
FUNC num calcInchToMM()	Calculates mm value from Inch.
FUNC num calcLbToKg()	Calculates Kg value from Lb.
FUNC tooldata calcToolData(robtarget pickPos,wobjdata pickWorkObject,\num RobotNumber)	This function returns a new tooldata updated with a TCP matching position of currently found object in FlexLoader Vision. Use this procedure first before teaching or using positions that need a TCP that is attached to the FlexLoader Vision position. Use with caution.
FUNC num getCycleTime(num nArrayIndex,\num nNumberOfCycleTimes,\num nPartsPerCycle)	Returns cycle time in seconds as a num. If no valid cycle time calculated, return value will be -1. It also adds cycle time of the last cycle to the array.
FUNC dionum getDISignalState(string SignalName)	Reads value of a digital input signal passed to function as string, returns 1 if signal is high, else 0 will returned.
FUNC dionum getDOSignalState(string SignalName)	Reads value of a digital output signal, returns 1 if signal is high, else 0 will returned.
FUNC bool getFlashingHz(num Hz)	This function returns true or false depending on a timer and a wanted frequency.
FUNC string getModuleName(string sModulePath\switch WithFileExtension)	This function returns the module name from a search path string.
FUNC num getPartCount(num nArrayIndex\num nPartsPerCount)	Help to count parts in cell.
FUNC num getRobotNumber()	Returns robot number of the robot where calling module is executing.

Continues on next page

Routine	Description
FUNC bool isDistanceOK(\num RobotNumber,\tooldata checkTool,robtarget checkPosition,wobjdata workObject,num maxDistance\switch PrintDistance)	Function that checks if a robot with its current tool is close to the specified robtarget in the specified workobject.
FUNC num maxNumValue(num Value1,num Value2)	Returns highest value of two num variables.
FUNC num minNumValue(num Value1,num Value2)	Returns lowest value of two num variables.
PROC ResetCycleTimes(num nArrayIndex)	Reset the cycle time calculation.
PROC ResetPartCount(num nArrayIndex)	Reset part counter.
PROC SetConfigurationVariable(INOUT bool ConfiguratioValue,string SignalName,\switch OutputSignal|switch InputSignal)	Sets a configuration value depending on if passed IO exists in system or not.
PROC SetDOSignal(string SignalName,dionum Value)	Set a digital output signal with signal name passed as a string.
FUNC tooldata ToolOffset(PERS tooldata ToolToAdjust,\num tOffsX,\num tOffsY,\num tOffsZ)	Returns tooldata value with offsets according to optional arguments. If no optional arguments is used Return value is the same as input value. Example: `tTempDeburr:=ToolOffset(tDeburrOriginal\tOffsX:=0\tOffsY:=0\tOffsZ:=2);`
PROC WriteActiveAlarmsToFile()	Writes all active alarms in system to a file.

IndicationLights.mod

The background task with module IndicationLights.mod, handles the indicator lights according to operational status.

Routine	Description
PROC main()	Main loop for task.
PROC BlueLight()	Sets blue light in RGB.
PROC FlashFunction()	Toggle values for flash memories.
PROC GreenLight()	Sets green light in RGB.
PROC HandleStandardTower()	Handle main logic for a standard light tower. A standard tower has 1-5 separate lamps with single color.
PROC HandleRGBTower()	Handle main logic for a RGB light tower. RGB tower has one lamp possible to handle red, green and blue and also to mix those.
PROC NoLight()	Sets no light in RGB.

Continues on next page

Routine	Description
PROC RedLight()	Sets red light in RGB.
PROC UpdateLocalAlarmStatus()	Update local bits for sum alarm, warning, information and questions
PROC VisionLightsOnControl()	Controls and checks if lights would be switched on or off.
PROC WhiteLight()	Sets white light in RGB.
PROC YellowLight()	Sets yellow light in RGB.

IndicationLights_shared.sys

The shared module IndicationLights_shared.sys handles task general data and routines for indication lights.

Routine	Description
PROC IndicationLightsSetupLightTower()	Assigns correct color to each message type.
PROC IndicationLightsSetupRGBLights()	Assign correct color to each message type.

ToolControlMenu.mod

The module ToolControlMenu.mod handles the FlexPendant menu for manual control of tool signals.

This module depends on the use of Tools_shared.sys to declare all tools in the system. As long as this is done, no further input is needed in ToolControlMenu.mod.

The tool menu is activated by the signal DOF_ToolMenuButton. For convenience, this signal can be connected to a function key.

Routine	Description
TRAP ToolControlMenuTrap	This trap displays menu for control of Tools. Normally connected to function key 1 on TeachPendant.

Tools_base.sys

The module Tools_base.sys handles tool actions and supports with some tool state functions to check state of parts held in grippers etc.

Routine	Description
FUNC bool checkAllToolsEmpty()	Returns TRUE if all robot mounted tools are empty.
FUNC mtpartstate GetToolPartState(dnum Tool)	Help function to slim down arguments when finding out part state of the part in robot tool.
PROC GripLoadUpdate(dnum ActiveTool)	Search for all robot tools and what they are holding. Calculates a total loaddata sum for all tools that holds something. Calls GripLoad with the calculated total load.
PROC PickPartAt(mtstationdata FromStation,\num TrackerIndex,dnum PickTool,\bool InsideGrip)	Modifies tool signal to pick a part from FromStation. Then move part tracker data from FromStation to PickTool.

Continues on next page

Routine	Description
PROC PlacePartAt(mtstationdata ToStation,\num TrackerIndex,dnum PlaceTool,\bool InsideGrip)	Modify tool signal to release a part from tool PlaceTool. Then move part tracker data from PlaceTool to ToStation.

Tools_shared.sys

The module Tools_shared.sys handles tool definitions and a lot of tool action and control routines. A lot of the routines is used by the Tools_base.sys module.

Routine	Description
PROC AddToSumGripLoad(INOUT loaddata pSumLoad,tooldata ActiveTool,loaddata ThisLoad,tooldata ThisTool)	Help function to calculate a total load from all loads currently held by robot.
FUNC bool CheckValidTool(\num RobotNumber,dnum ToolNumber)	Checks if a specific tool is defined in system.
PROC ClearToolData(\bool RobHold,\num RobotNumber,\dnum ToolNumber)	Clears tooldefinition data. If using optional argument, only remove that specific data, else remove all.
FUNC num getCurrentTotalLoadMass(\num RobotNumber)	Returns total load mass carried in specific robot tools.
FUNC tooldefinition getToolDefinitionData(\bool RobHold,\num RobotNumber,dnum ToolNumber)	Returns the tooldefinition data for a given tool number.
FUNC tooldata getToolData(\bool RobHold,\num RobotNumber,dnum ToolNumber)	Returns the tooldata for a given tool number.
FUNC string getToolStatus(\bool RobHold,\num RobotNumber,dnum Tool)	Returns current tool status label. Depending on tool current status return corresponding text label from the tool declaration.
PROC ToolAction(\bool RobHold,\num RobotNumber,\switch Open|switch Release|switch On|switch Close|switch Grip|switch Off,dnum Tool,\bool InsideGrip,\num WaitTimeOverride,\switch NoManualConfirmation)	Perform action with a robot tool. Actions could be Open, On, Close or Off. Uses signal data and wait times declared through ToolSetup.

Continues on next page

Routine	Description
PROC ToolSetup(\bool RobHold,\num RobotNumber,dnum ToolNumber,string Label,tooldata Tool_Data,mtstationdata Station,\num StnIndex,\string OpenOrOnSignal,\num WaitTimeOpenOrOn,\string LabelOpenOrOn,\string CloseOrOffSignal,\num WaitTimeCloseOrOff,\string LabelCloseOrOff,\string BlockManualControl)	Setup a new tool in system. Stores all values needed for the tool.
PROC ToolUpdate(\bool RobHold,\num RobotNumber,dnum ToolNumber,\string Label,\tooldata Tool_Data,\mtstationdata Station,\num StnIndex,\string OpenOrOnSignal,\num WaitTimeOpenOrOn,\string LabelOpenOrOn,\string CloseOrOffSignal,\num WaitTimeCloseOrOff,\string LabelCloseOrOff,\string BlockManualControl)	Updates tooldefinition data. Finds the robot tool (through it's ToolNumber) and updates all data for that tool which is passed in the arguments. Data which is not passed as argument will not be touched.

B.7 FlexLoader machine tool interface functionality

Overview

The FlexLoader Vision application functionality handles machine tool and part specific aspects of the FlexLoader Vision. The integrator is fully responsible for adopting this functionality to the actual application. Described below is two different template modules. Those describes which routines and data that is needed to communicate with a machine. All real machine signals is only handled within this machine module. This makes it easier to change machine or reuse an earlier made machine module. This means that in a real project, the template module should be switched out to a module that follow the same structure but are prepared for the correct machine used in the project. This could be done by rewriting the template module or replacing it by a pre-made module.

Module	Description
TemplateMachine_CNC.sys	Provides bridge between logical functions within the FlexLoader Vision application and physical interface for up to two CNC machine tools. It also includes code to handle indexing table within both machines.
TemplateMachine_Lathe.sys	Provides bridge between logical functions within the FlexLoader Vision application and physical interface to a lathe machine tool.

Note

If more machine tools is used than what is handled by the template, then the functionality can be enhanced by the integrator.

TemplateMachine_CNC.sys

This module contains a number of predefined procedures or functions for communication with the machine tool:

Name	Description
FUNC mtpartstate getMachine1PartType()	Intended use is for index table machines to keep track of parts in machine 1.
FUNC mtpartstate getMachine2PartType()	Intended use is for index table machines to keep track of parts in machine 2.
PROC InitMachine1()	Initiates what is necessary for machine 1 handling. NEEDED: Set up correct signals for interrupt handling.
PROC InitMachine2()	Initiates what is necessary for machine 2 handling. NEEDED: Set up correct signals for interrupt handling.
PROC Machine1Action(machineaction Action)	Performs desired machine 1 tool action. NEEDED: Define signals and communicate with machine tool.
FUNC bool Machine1Check(machinecheck Check,\INOUT bool Result)	Checks machine 1 tool and conditions and returns correct state for use in main flow of program. NEEDED: Define signals and communicate with machine tool.

Continues on next page

Name	Description
FUNC bool Machine1Wait(machinecheck Wait,num MaxTime)	Waits for machine 1 condition to be ok within **MaxTime**. Returns false if **MaxTime** elapses. NEEDED: Define signals and communicate with machine tool.
PROC Machine1Action(machineaction Action)	Performs desired machine 2 tool action. NEEDED: Define signals and communicate with machine tool.
FUNC bool Machine1Check(machinecheck Check,\INOUT bool Result)	Checks machine 2 tool and conditions and returns correct state for use in main flow of program. NEEDED: Define signals and communicate with machine tool.
FUNC bool Machine1Wait(machinecheck Wait,num MaxTime)	Waits for machine 2 condition to be ok within **MaxTime**. Returns false if **MaxTime** elapses. NEEDED: Define signals and communicate with machine tool.

DANGER

Ensure that the standard features in **MachineXAction**, **MachineXCheck** and **MachineXWait** are correctly implemented (where X is machine number). Malfunction and collisions might otherwise occur.

Note

We recommend the use of an additional background task if the robot cell requires time critical machine tool communication.

Note

Operation with more than two machines requires the creation of these routines with equivalent names, e.g. Machine3Action and Machine4Wait.

TemplateMachine_Lathe.sys

This module contains a number of predefined traps for information transfer with machine tool.

Name	Description
TRAP DoorClosedTrap	Used on machine tools with slow doors. Signal is caught by trap instead of foreground program so that the robot can continue working. Confirms to the machine tool that a door closed signal has been received and the whole loading sequence is finished.
TRAP DoorNotOpenedTrap	Can be used for special conditions or checks on certain machine tools.
TRAP PrePickTrap	Receives information from machine tool that a part soon needs to be loaded. If needed, add code to confirm to the machine tool that a signal has been received.

Continues on next page

This module contains a number of predefined procedures or functions for communication with the machine tool:

Name	Description
PROC CalibMainChuck()	Help procedure for calibrating chuck at main spindle.
PROC CalibSubChuck()	Help procedure for calibrating chuck at sub spindle.
PROC InitMachine1()	Initiates what is necessary for machine handling. Sets up traps. NEEDED: Set up correct signals for interrupt handling.
PROC Machine1Action(machineaction Action)	Performs desired machine tool action. NEEDED: Define signals and communicate with machine tool.
FUNC bool Machine1Check(machinecheck Check,\INOUT bool Result)	Checks machine tool and conditions and returns correct state for use in main flow of program. NEEDED: Define signals and communicate with machine tool.
FUNC bool Machine1Wait(machinecheck sWait,num nMaxTime)	Waits for machine condition to be ok within **nMaxTime**. Returns false if **nMaxTime** elapses. NEEDED: Define signals and communicate with machine tool.

DANGER

Ensure that the standard features in **Machine1Action**, **Machine1Check** and **Machine1Wait** are correctly implemented. Malfunction and collisions might otherwise occur.

Note

We recommend the use of an additional background task if the robot cell requires time critical machine tool communication.

Note

Operation with more than one machine requires the creation of these routines with equivalent names, e.g. Machine2Action and Machine3Wait.

States and actions

The machine modules uses a number of predefined states and actions that are consistently used. If needed, new values can be added in GlobalCodeAndConfig.sys. However, do not remove the existing values, as these are used in the FlexLoader core functionality.

Examples for predefined values are:

- Examples for predefined machine checks/waits: DOOR_OPENED, PREPICK_OK, LOAD_SUB_OK, LOAD_MAIN_OK, UNLOAD_SUB_OK, UNLOAD_MAIN_OK, HOME_POSITION_OK, MAIN_CHUCK_OPENED, MAIN_CHUCK_CLOSED, SUB_CHUCK_OPENED, SUB_CHUCK_CLOSED
- Examples for predefined machine actions:

Continues on next page

CLOSE_CHUCK_MAIN, CLOSE_CHUCK_SUB, CLOSE_DOOR, OPEN_CHUCK_MAIN, OPEN_CHUCK_SUB, OPEN_DOOR, PREPARE_LOAD_MAIN, PREPARE_LOAD_SUB, PREPARE_UNLOAD_MAIN, PREPARE_UNLOAD_SUB, CYCLE_START, INIT_MACHINE, STOP_MACHINE

There are further predefined values in GlobalCodeAndConfig.sys that can be implemented in machine communication modules.

C FlexLoader Library Add-in reference

C.1 Instructions

C.1.1 MT_ClearAllPartInfo - Clears all part tracker information

Usage

`MT_ClearAllPartInfo` is used to clear all part tracker information.

Basic examples

The following example illustrates the instruction `MT_ClearAllPartInfo`.

Example 1

```
MT_ClearAllPartInfo;
```

This will clear all part info in the all trackers defined in the program.

C.1.2 MT_ClearPartInfo - Resets part information of a part tracker

Usage

`MT_ClearPartInfo` is used to reset the complete part information in a specific part tracker of a station.

Basic examples

The following example illustrates the instruction .

Example 1

```
PERS mtstationdata sdRobot;
CONST num GRIPPER_1:=1;
PROC TestProc()
  MT_ClearPartInfo\Station:=sdRobot,\Index:=GRIPPER_1;
ENDPROC
```

This will clear all part info in the first part tracker of the robot station. For a better readability the GRIPPER_1 was assigned value 1.

Arguments

```
MT_ClearPartInfo [\Station] | [\StationName] [\Index]
```

`\Station`

Data type: `mtstationdata`

The station data which part tracker data should be cleared in.

`\StationName`

Data type: `string`

Name of the station data declaration as a string. Through this name the correct mtstationdata definition will be found and the corresponding part tracker where part data should be cleared.

`\Index`

Data type: `num`

Index number of the part tracker to clear. Note, each station can have up to 20 trackers. If argument not passsed, index number 1 is used.

C.1.3 MT_CopyPartInfo - Copies part info between trackers

Usage

MT_CopyPartInfo is used to copy part tracker information from one part tracker to another.

Basic examples

The following example illustrates the instruction MT_CopyPartInfo.

Example 1

```
PERS mtstationdata sdRobot;
PERS mtstationdata sdMachine;
PERS num GRIPPER_1:=1;
PROC TestProc()
MT_CopyPartInfo
     \FromStation:=sdRobot,\FromIndex:=GRIPPER_1,\ToStation:=sdMachine;
ENDPROC
```

This call will copy part information from the part tracker connected to the robot station with index 1 (corresponds to gripper 1 in this case). The data will be copied (Note, not moved!) to the part tracker of the machine, sdMachine. Since this machine only have one tracker the index was left out (means that index 1 will be used internally).

Arguments

```
MT_CopyPartInfo [\FromStation] | [\FromStationName] [\FromIndex]
     [\ToStation] | [\ToStationName] [\ToIndex]
```

\FromStation

Data type: mtstationdata

The station to copy part tracker data from.

\FromStationName

Data type: string

The station to copy part tracker data from. In this argument, the name of the mtstationdata definition is passed as a string.

\FromIndex

Data type: num

Specifies which part tracker index in FromStation to fetch part info from.

\ToStation

Data type: mtstationdata

The station to copy part tracker data to.

\ToStationName

Data type: string

The station to copy part tracker data to. In this argument, the name of the mtstationdata definition is passed as a string.

Continues on next page

`\ToIndex`

Data type: `num`

Specifies which part tracker index in ToStation to copy part info to.

C.1.4 MT_CreateAlarm - Creates new alarm or message

Usage

Normally an alarm is defined by adding a new row in a csv file, see general description of Alarms&Messages. But when integrator is adding stuff later this procedure could be used instead. This instruction adds a new alarm/message definition to the system.

Basic examples

The following example illustrates the instruction `MT_CreateAlarm`. This will define a new alarm with identifier “myAlarm” and category Error.

Example 1

```
MT_CreateAlarm "myAlarm","Header text",["body text 1","body text
     2","body text 3"]\Category:=iconError,\ErrorNumber:=123;
```

Arguments

```
MT_CreateAlarm Identifier, Header, Text{*}, [\Category],
     [\ErrorNumber], [\Domain]
```

`Identifier`

Data type: `string`

System unique string that gives the possibility to point to a specific message.

`Header`

Data type: `string`

Header text for the message.

`Text{*}`

Data type: `string array`

Up to 8 strings could be passed in this message body text.

`\Category`

Data type: `icondata`

Defines the category of an alarm. If not used this will be normal message text.

`\ErrorNumber`

Data type: `num`

Defines error number for the message if needed.

`\Domain`

Data type: `num`

Defines domain number for the message if needed.

C.1.5 MT_ErrWrite - Creates error log message in local language

Usage

`MT_ErrWrite` is used to write a message to the robot event log.

Basic examples

The following example illustrates the instruction `MT_ErrWrite`.

This will write the message which match to identifier “myAlarm” to the robot event log. Depending on defined category it will show in event log with matching category. If category is not used for the message it will appear in the event log as an error type. The text will appear with wildcard {1} replaced with “AddText”.

Example 1

```
PROC ErrWriteExample()
  VAR mtmsgdata Alarm;
  Alarm:=MT_GetMessage("myAlarm");
  MT_ErrWrite Alarm,\WildCardTexts:=["AddText"];
ENDPROC
```

Arguments

```
MT_ErrWrite Message, [\WildCardTexts{*}]
```

`Message`

Data type: `mtmsgdata`

The message data that should be used to write message to error log.

`WildCardTexts{*}`

Data type: `string array`

String array of wild card text to replace wild cards in message texts.

C.1.6 MT_GetPartInfo - Gets any kind of part tracker data from a part tracker

Usage

MT_GetPartInfo is used to read any type of part data information from a specific part tracker.

Basic examples

The following example illustrates the instruction MT_GetPartInfo.

Example 1

```
VAR mtpartstate tempState;
VAR string tempString;
MT_GetPartInfo
     \Station:=sdRobot,\Index:=GRIPPER_1,\State:=tempState,\UserDef:=tempString;
```

This code will get the current state and user defined string for the part currently in robot gripper 1 (Robot station index 1). The data will be stored in tempState and tempString.

Arguments

```
MT_GetPartInfo [\Station] | [\StationName] [\Index] [\ProgCode]
     [\TypeCode] [\AuxCode] [\ToolCode] [\MachineCode] [\Routine]
     [\Part] [\Tool] [\PLoad] [\State] [\UserDef]
```

\Station

Data type: mtstationdata

The station to fetch part tracker data from.

\StationName

Data type: string

The station to fetch part tracker data from. In this argument name of the mtstationdata definition is passed as a string.

\Index

Data type: num

Specifies which part tracker index to fetch part info from.

\PartName

Data type: string

Name of the part in the tracker.

\ProgCode

Data type: dnum

Stores ProgCode into this INOUT argument. See mtpartdata.

\TypeCode

Data type: dnum

Stores TypeCode into this INOUT argument. See mtpartdata.

Continues on next page

`\AuxCode`

Data type: `dnum`

Stores AuxCode into this INOUT argument. See mtpartdata.

`\ToolCode`

Data type: `num`

Stores ToolCode into this INOUT argument. See mtpartdata.

`\MachineCode`

Data type: `dnum`

Stores MachineCode into this INOUT argument. See mtpartdata.

`\Routine`

Data type: `string`

Stores Routine into this INOUT argument. See mtpartdata.

`\Part`

Data type: `mtpartdata`

Stores Part into this INOUT argument.

`\Tool`

Data type: `tooldata`

Stores Tool into this INOUT argument.

`\PLoad`

Data type: `loaddata`

Stores PLoad into this INOUT argument.

`\Dimension`

Data type: `mtpartdimension`

Dimensions of the part in the tracker.

`\Deviation`

Data type: `mtpartdeviation`

Deviation of the part in the tracker.

`\State`

Data type: `mtpartstate`

Stores State into this INOUT argument. See mtparttracker.

`\UserDef`

Data type: `string`

Stores UserDef into this INOUT argument. See mtparttracker.

C.1.7 MT_GoHome - Moves robot to home zone

Usage

`MT_GoHome` is used to move robot to home zone.

Basic examples

The following example illustrates the instruction `MT_GoHome`.

Example 1

```
MT_GoHome;
```

Moves robot to position in the mttargetdata ZONE_HOME.

C.1.8 MT_InitializeMessageHandling - Initiates the alarms and messaging module

Usage

`MT_InitializeMessageHandline` is used to initiate the alarms and messaging modules. The core task is to check currently used language and make sure corresponding user texts is loaded in the system.

Basic examples

The following example illustrates the instruction `MT_InitializeMessageHandling`. This call will force new loading of all texts from csv files even if the language selection didn't change since last time. All current messages will be removed first.

Example 1

```
MT_InitializeMessageHandling\ForceLoading;
```

Arguments

```
MT_InitializeMessageHandling [\ForceLoading]
```

`\ForceLoading`

Data type: `switch`

Forces loading of all messages even if there hasn’t been a language change or text file changes.

C.1.9 MT_Log - Creates a log message in log file

Usage

MT_Log is used to create a log message in log file. The log messages can be assigned to different log domains, to be enabled before logging by means of the instruction MT_Log_Enable .

Basic examples

The following example illustrates the instruction MT_Log.

Example 1

```
!Enables log for warning level
MT_LogEnable TRUE\Level:MT_LVL_WARNING;
...
!Create a new entry in the log file.
MT_Log MT_LVL_WARNING,"main",["Log message column 1","Log message
     column 2","Log message column 3"];
```

Creates a new log entry in log file if log level warning is active. The content will look as follows, depending on date and time:

```
2020-07-03;13:42:45;Warning;main;Log message column 1;Log
message column 2;Log message column 3
```

Arguments

```
MT_Log [Level] , [Source] , [Msg{*}]
```

`Level`

Data type: `mtloglevel`

Specifies the log level of this log message.

`Source`

Data type: `string`

Specifies the source of this log message, for normally routine name. Free for user to use it as something else than routine name.

`msg{*}`

Data type: `string array`

An array of log message lines.

C.1.10 MT_LogEnable - Enables or disables logging

Usage

`MT_LogEnable` is used to activate or deactivate logging in general or for a specific log level.

Basic examples

The following example illustrates the instruction .

Example 1

```
MT_LogEnable TRUE,\Level:=MT_LVL_WARNING;
```

Activates all logging to file which has the log level warning. See data type mtloglevel for more information about the different log levels.

Arguments

```
MT_LogEnable [Enabled] , [\Level]
```

`Enabled`

Data type: `bool`

Set to TRUE to activate logging. Set to deactivate logging.

`\Level`

Data type: `mtloglevel`

Specifies which kind of log level which should be turned on or off. If argument not used, turn on or off all kind of log levels.

C.1.11 MT_MovePartInfo - Moves part info between part trackers

Usage

`MT_MovePartInfo` is used to move part data information from one part tracker to another part tracker.

Basic examples

The following example illustrates the instruction `MT_MovePartInfo`.

Example 1

```
PERS mtstationdata sdRobot;
PERS mtstationdata sdMachine;
PERS num GRIPPER_1:=1;
PROC TestProc()
MT_MovePartInfo
     \FromStation:=sdRobot,\FromIndex:=GRIPPER_1,\ToStation:=sdMachine;
ENDPROC
```

This call will move part information from the part tracker connected to the robot station with index 1 (corresponds to gripper 1 in this case). The data will be moved to the part tracker of the machine, sdMachine. Since this machine only have one tracker the index was left out (means that index 1 will be used internally). Afterwards the FromStation tracker will have empty part data information.

Arguments

```
MT_MovePartInfo [\FromStation] | [\FromStationName] [\FromIndex]
     [\ToStation] | [\ToStationName] [\ToIndex]
```

`\FromStation`

Data type: `mtstationdata`

The station to move part tracker data from.

`\FromStationName`

Data type: `string`

The station to move part tracker data from. In this argument, the name of the mtstationdata definition is passed as a string.

`\FromIndex`

Data type: `num`

Specifies which part tracker index in FromStation to fetch part info from.

`\ToStation`

Data type: `mtstationdata`

The station to move part tracker data to.

`\ToStationName`

Data type: `string`

The station to move part tracker data to. In this argument, the name of the mtstationdata definition is passed as a string.

Continues on next page

`\ToIndex`

Data type: `num`

Specifies which part tracker index in ToStation to move part info to.

C.1.12 MT_MoveRobotTo - Moves robot to a specific zone

Usage

`MT_MoveRobotTo` is used to move robot to a new zone/station. Call this when robot is standing in the via position of the current zone. If robot is not in the current zones via position, add argument \CheckStart.

Basic examples

The following example illustrates the instruction `MT_MoveRobotTo`.

Example 1

```
MT_MoveRobotTo ZONE_MACHINE;
```

This code moves robot to the position that is defined within the ZONE_MACHINE definition. But it’s not a direct movement to this point. The movement will be done by calling the mvZONE_MACHINE.

Arguments

```
MT_MoveRobotTo [\CheckStart] , [TargetZone] , [\TargetZoneName]
```

`\CheckStart`

Data type: `switch`

Use switch if the robot should first move to the current zone via position. Normally the robot is in the current zone via position when calling this, and in that case don’t use this switch. If using this argument the user will first get a question if robot should start moving straight for the via position.

`TargetZone`

Data type: `mttargetdata`

The mttargetdata of the zone that robot should move to.

`TargetZoneName`

Data type: `string`

If TargetZone is passed through a variable and not by it’s real variable definition, then use this variable to pass the real name of the zone to get this working.

C.1.13 MT_MoveToStart - Moves robot to closest start position

Usage

`MT_MoveToStart` is used to move robot to closest via position, i.e. the closest safe start position.

Basic examples

The following example illustrates the instruction `MT_MoveToStart`.

Example 1

```
MT_MoveToStart;
```

Arguments

```
MT_MoveToStart [\RealMode]
```

`\RealMode`

Data type: `switch`

Only usable in virtual controller. If not used in virtual mode, robot will always go directly to closest without any checks. If used, robot will act as in auto and check distance to closest position and if too far away ask the operator if movement to that position is ok or not.

C.1.14 MT_MoveToZoneMenu - Shows manual menu to move between zones

Usage

`MT_MoveToZoneMenu` is used to move robot to any desired zone. Routine will show a list of all possible zones and ask user to select where to move. It’s a good idea to connect this to a function key on Flexpendant and allow it only in manual mode.

Basic examples

The following example illustrates the instruction `MT_MoveToZoneMenu`.

Example 1

```
MT_MoveToZoneMenu;
```

If robot is in manual mode, the selection list of possible zones to move to will show on Flexpendant.

C.1.15 MT_ResetAlarms - Resets all or a specific alarm

Usage

`MT_ResetAlarms` is used to reset one or several alarms.

Basic examples

The following examples illustrates the instruction `MT_ResetAlarms`.

Example 1

```
MT_ResetAlarms \Identifier:="myAlarm";
```

Resets all active alarms with identifier myAlarm, note it could be more than one if wild card texts differ.

Example 2

```
MT_ResetAlarms;
```

Resets all active alarms.

Example 3

```
MT_ResetAlarms
     \Identifier:="myAlarm",\WildCardTexts:=["myWildCard"];
```

Resets the alarm that have the identifier "myAlarm" and is used with the wild card number 1: "myWildCard". Note that if more than one wild card was used in the alarm creation or if more was passed to the reset above, it is not an exact wild card match and the reset would not be done.

Arguments

```
MT_ResetAlarms [\Identifier] , [\WildCardTexts{*}]
```

`\Identifier`

Data type: `string`

System unique string that gives the possibility to point to a specific message to reset.

`\WildCardTexts{*}`

Data type: `string array`

String array of wild cards text to look for. All text must match with the active alarms wild card text to reset the message.

C.1.16 MT_SetAlarm - Sets an alarm to active

Usage

MT_SetAlarm is used to set an alarm to active state and also writes the message to robot e-log. At the same time a new event MT_EV_MSG_WRITTEN is created.

Basic examples

The following example illustrates the instruction MT_SetAlarm.

Example 1

```
MT_SetAlarm "myAlarm"\WildCardTexts:=["text 1", "text 2"];
```

myAlarm is set to active with the wild card texts “text 1” and “text 2”.

Arguments

```
MT_SetAlarm Identifier , [\WildCardTexts{*}]
```

Identifier

Data type: string

System unique string that gives the possibility to point to a specific message.

\WildCardText{*}

Data type: string array

String array of wild cards text to use for this message.

C.1.17 MT_SetPartInfo - Sets part information of a part in a specific tracker

Usage

MT_SetPartInfo is used to set any type of part data information to a specific part tracker.

Basic examples

The following example illustrates the instruction MT_SetPartInfo.

Example 1

```
PERS mtstationdata sdRobot;
PERS num GRIPPER_1:=1;
MT_SetPartInfo \Station:=sdRobot,\Index:=GRIPPER_1,\State:=psRAW;
```

This code will set part state raw to robot gripper 1 (Robot station index 1).

Arguments

```
MT_SetPartInfo [\Station] | [\StationName] [\Index] [\ProgCode]
      [\State] [\UserDef]
```

\Station

Data type: mtstationdata

The station to set part tracker data to.

\StationName

Data type: string

The station to set part tracker data to. In this argument name of the mtstationdata definition is passed as a string.

\Index

Data type: num

Specifies which part tracker index to set part info to.

\ProgCode

Data type: dnum

ProgCode to set to the specified part tracker. See mtpartdata.

\PartName

Data type: string

Name of the part, use this or the ProgCode.

\State

Data type: mtpartstate

State to store in the specified part tracker. See mtparttracker.

\UserDef

Data type: string

User defined data to store in the specified part tracker. See mtparttracker.

C.1.18 MT_SetPartState - Set state of a part in a part tracker

Usage

`MT_SetPartState` is used to update a state of a part in a specific tracker.

Basic examples

The following example illustrates the instruction `MT_SetPartState`.

Example 1

```
MT_SetPartState \Station:=sdRobot,psMACHINED;
```

This example updates the robot station first tracker to state machined. Note that when the index is left out as in the example, index 1 is used.

Arguments

```
MT_SetPartState [\Station] | [\StationName] [\Index] State
```

`\Station`

Data type: `mtstationdata`

Station data of the station to update tracker for.

`\StationName`

Data type: `string`

Station data of the station to update tracker for.

`\Index`

Data type: `num`

Index number for the tracker to update.

`\State`

Data type: `mtpartstate`

New state for the part.

C.1.19 MT_UpdateMessageTextsIfNeeded - Reads messages into message array if needed

Usage

`MT_UpdateMessageTextsIfNeeded` is used to read message texts of used language into message array if file was changed since last read or if a read is asked anyway.

Basic examples

The following example illustrates the instruction `MT_UpdateMessageTextsIfNeeded`.

Example 1

```
MT_UpdateMessageTextsIfNeeded\ForceLoading;
```

This will always update message texts from files to RAPID system.

Arguments

```
MT_UpdateMessageTextsIfNeeded(\ForceLoading)
```

`ForceLoading)`

Data type: `switch`

Use if texts always should be updated independent of changes to language settings or file modifications.

C.2 Functions

C.2.1 MT_GetActiveAlarm - Get an active alarm data

Usage

Returns a complete active alarm data from the array of active alarms. If there are no alarms or nothing on the index number asked, then an empty active alarm data is returned.

Basic examples

The following example illustrates the function `MT_GetActiveAlarm`.

Example 1

```
VAR mtactivealarmdata Alarm;
Alarm:=MT_GetActiveAlarm(\Index:=1);
```

This will fetch the complete active alarm data for message with index 1 (latest alarm created) and store it into Alarm.

Return value

Data type: `mtactivealarmdata`

Arguments

```
MT_GetMessage ([\Index])
```

`\Index`

Data type: `num`

Index of the alarm that should be returned from the active alarm list. If not used, index 1 (latest created alarm) will be returned.

C.2.2 MT_GetCurrentZone - Get zone where robot is positioned now

Usage

`MT_GetCurrentZone` is used to find out in which zone robot is positioned in right now.

Basic examples

The following example illustrates the function `MT_GetCurrentZone`.

Example 1

```
VAR mttargetdata Current;
Current:=MT_GetCurrentZone();
```

This code checks where the robot is positioned right now and returns matching mttargetdata which is stored in Current.

Return value

Data type: `mttargetdata`

Target data matching the current position of the robot.

Arguments

```
MT_GetCurrentZone(\INOUT ZoneName)
```

`ZoneName)`

Data type: `string`

Name as a string of the returned zone.

C.2.3 MT_GetMessage - Get complete message data

Usage

`MT_GetMessage` is used to get the complete mtmsgdata by passing corresponding identifier.

Basic examples

The following example illustrates the function `MT_GetMessage`. This will fetch the complete message data for message with identifier “myAlarm” and store it into Alarm.

Example 1

```
VAR mtmsgdata Alarm;
Alarm:=MT_GetMessage("myAlarm");
```

Return value

Data type: `mtmsgdata`

Arguments

```
MT_GetMessage (Identifier)
```

`Identifier`

Data type: `string`

System unique string that gives the possibility to point to a specific message.

C.2.4 MT_GetNumberOfActiveAlarms - Gets number of active alarms

Usage

`MT_GetNumberOfActiveAlarms` is used to check how many alarms are currently active.

Basic examples

The following example illustrates the function `MT_GetNumberOfActiveAlarms`. This will fetch the number of active alarms in the system and store it into NumberOfAlarms.

Example 1

```
VAR num NumberOfAlarms;
NumberOfAlarms:=MT_GetNumberOfActiveAlarms();
```

Return value

Data type: `num`

C.2.5 MT_GetPartState - Gets current part state of a part in specific tracker

Usage

`MT_GetPartState` is used to get the part state of a part in a specific part tracker. This is normally a faster way to fetch the state than using MT_GetPartInfo.

Basic examples

The following example illustrates the function `MT_GetPartState`.

Example 1

```
PERS mtstationdata sdRobot;
PERS num GRIPPER_1:=1;
IF MT_GetPartState(\Station:=sdRobot,\Index:=GRIPPER_1)=psRAW THEN
!Do actions if raw part
ENDIF
```

Take some actions if the part in robot gripper 1 is a raw part.

Return value

Data type: `mtpartstate`

Arguments

```
MT_GetPartState ([\Station] | [\StationName] [\Index])
```

`\Station`

Data type: `mtstationdata`

The station data which part state data should be fetched in.

`\StationName`

Data type: `string`

Name of the station data declaration as a string. Through this name the correct mtstationdata definition will be found and the corresponding part tracker where part state data should be fetched.

`\Index`

Data type: `num`

Index number of the part tracker fetch state from. Note, each station can have up to 20 trackers.

C.2.6 MT_GetText - Gets a selected text line from a message

Usage

MT_GetText is used to get a selected text line from a specific message.

Basic examples

The following example illustrates the function MT_GetText. This will store the header text from the message corresponding to identifier “myAlarm” to the variable header. if no optional argument is used, the header text is returned.

Example 1

```
VAR string header;
header:=MT_GetText("myAlarm",\Header);
```

Return value

Data type: string

Arguments

```
MT_GetText (Identifier, [\Header] | [\Text1] | [\Text2] | [\Text3]
      | [\Text4] | [\Text5] | [\Text6] | [\Text7] | [\Text8])
```

Identifier

Data type: string

System unique string that gives the possibility to point to a specific message.

\Header

Data type: switch

Gets the header text line.

\Text1

Data type: switch

Gets the body text number 1.

\Text2

Data type: switch

Gets the body text number 2.

\Text3

Data type: switch

Gets the body text number 3.

\Text4

Data type: switch

Gets the body text number 4.

\Text5

Data type: switch

Gets the body text number 5.

Continues on next page

`\Text6`

Data type: `switch`

Gets the body text number 6.

`\Text7`

Data type: `switch`

Gets the body text number 7.

`\Text8`

Data type: `switch`

Gets the body text number 8.

C.2.7 MT_GetViaPosition - Gets corresponding via position for a zone

Usage

`MT_GetViaPosition` is used to fetch the robtarget data from a specific mttargetdata.

Basic examples

The following example illustrates the function `MT_GetViaPosition`.

Example 1

```
MoveJ MT_GetViaPosition(ZONE_MACHINE),v500,fine,tTool0;
```

Moves robot to the via position connected to ZONE_MACHINE.

Return value

Data type: `robtarget`

Arguments

```
MT_GetViaPosition([Target])
```

`Target`

Data type: `mttargetdata`

The mttargetdata for the target for which the robtarget should be fetched.

C.2.8 MT_IsAlarmActive - Check if an specific alarm is active

Usage

`MT_IsAlarmActive` is used to check if an specific alarm is currently active. Combine category and station if needed to check currently active alarm on a specific station of a specific category.

Basic examples

The following examples illustrates the function `MT_IsAlarmActive`.

Example 1

```
VAR dionum AlarmActive;
AlarmActive:=MT_IsAlarmActive();
```

AlarmActive will be 1 if any alarm is active in the system.

Example 2

```
AlarmActive:=MT_IsAlarmActive(\Identifier:="myAlarm");
```

AlarmActive will be 1 if the alarm with identifier “myAlarm” is active.

Return value

Data type: `dionum`

Arguments

```
MT_IsAlarmActive ([\Identifier] , [\WildCardTexts{*}] , [\Category]
     , [\StationName])
```

`\Identifier`

Data type: `string`

System unique string that gives the possibility to point to a specific message.

`\WildCardTexts{*}`

Data type: `string array`

String array of wild cards text to look for. All text must match to get a result 1 back. This argument only has a meaning if used at the same time as the identifier argument.

`\Category`

Data type: `icondata`

Only return result 1 if an alarm matches this category. This argument can’t be used together with Identifier argument.

`\StationName`

Data type: `string`

Only return true if there is an active alarm which is connected to this station. This argument can’t be used together with Identifier argument.

C.2.9 MT_UIBoolEntry - Show an yes no question in localized language

Usage

`MT_UIBoolEntry` is used to show a UI message box on Flexpendant. Texts are fetched from correct message, use also wild card texts if wanted. Only two buttons are possible to show, one that will give the result TRUE and one that will give result FALSE. Except for the texts, all other arguments available in UIMessageBox is available here to.

Basic examples

The following example illustrates the function `MT_UIBoolEntry`.

Example 1

```
VAR bool answer;
answer:=MT_UIBoolEntry("myMessage",\WildCardTexts:=["text 1","text
     2"],\TrueButtonText:="Yes",\FalseButtonText:="No");
```

This will show a UI message box on the Flexpendant with header and text fetched from the message corresponding to identifier “myMessage”. Wild cards {1} and {2} will be replaced with “text 1” and “text 2” correspondingly. Two buttons will show, one with text “Yes” and one with text “No”. If the "True button" is pressed, function will return TRUE.

Return value

Data type: `bool`

Arguments

```
MT_UIBoolEntry (Identifier , [\WildCardTexts{*}] , [\Wrap] ,
     [\TrueButtonText] , [\FalseButtonText] , [\DefaultBtn] ,
     [\Icon] , [\Image] , [\MaxTime] , [\DIBreak] , [\DIPassive]
     , [\DOBreak] , [\DOPassive] , [\PersBoolBreak] ,
     [\PersBoolPassive] , [\BreakFlag] , [\UIActiveSignal])
```

`Identifier`

Data type: `string`

System unique string that gives the possibility to point to a specific message.

`\WildCardTexts{*}`

Data type: `string array`

String array of wild cards text to use for this message.

`[\Wrap]`

Data type: `switch`

If selected, all the specified strings in the argument `\MsgArray` will be concatenated to one string with single spaces between each individual string and spread out on as few lines as possible.

Default, each string in the argument `\MsgArray` will be on separate line on the display.

Continues on next page

`\TrueButtonText`

Data type: `string`

Text to show on button resulting true value.

`\FalseButtonText`

Data type: `string`

Text to show on button resulting false value.

`[\DefaultBtn]`

Default Button

Data type: `btnres`

Allows to specify a value that should be returned if the message box is interrupted by `\MaxTime`, `\DIBreak`, or `\DOBreak`. It's possible to specify the predefined symbolic constant of type `btnres` or any user defined value. See *Predefined data on page 375*.

`[\Icon]`

Data type: `icondata`

Defines the icon to be displayed. Only one of the predefined icons of type `icondata` can be used. See *Predefined data on page 375*.

Default, no icon.

`[\Image]`

Data type: `string`

The name of the image that should be used. To launch own images, the images has to be placed in the `HOME:` directory in the active system or directly in the active system.

The recommendation is to place the files in the `HOME:` directory so that they are saved if a Backup and Restore is done.

A **Restart** is required and then the FlexPendant loads the images.

The image that will be shown can have the width of 185 pixels and the height of 300 pixels. If the image is bigger, only 185 * 300 pixels of the image will be shown starting at the top left of the image.

No exact value can be specified on the size that an image can have or the amount of images that can be loaded to the FlexPendant. It depends on the size of other files loaded to the FlexPendant. The program execution will just continue if an image is used that has not been loaded to the FlexPendant.

`[\MaxTime]`

Data type: `num`

The maximum amount of time in seconds that program execution waits. If no button is selected within this time, the program continues to execute in the error handler unless the `BreakFlag` is used (see below). The constant `ERR_TP_MAXTIME` can be used to test whether or not the maximum time has elapsed.

`[\DIBreak]`

Digital Input Break

Continues on next page

Data type: `signaldi`

The digital input signal that may interrupt the operator dialog. If no button is selected when the signal is set to 1 (or is already 1) then the program continues to execute in the error handler, unless the `BreakFlag` is used (see below). The constant `ERR_TP_DIBREAK` can be used to test whether or not this has occurred.

`[\DIPassive]`

Digital Input Passive

Data type: `switch`

This switch overrides the default behavior when using `DIBreak` optional argument. Instead of reacting when signal is set to 1 (or already 1), the instruction should continue in the error handler (if no `BreakFlag` is used) when the signal `DIBreak` is set to 0 (or already is 0). The constant `ERR_TP_DIBREAK` can be used to test whether or not this has occurred.

`[\DOBreak]`

Digital Output Break

Data type: `signaldo`

The digital output signal that may interrupt the operator dialog. If no button is selected when the signal is set to 1 (or is already 1) then the program continues to execute in the error handler, unless the `BreakFlag` is used (see below). The constant `ERR_TP_DOBREAK` can be used to test whether or not this has occurred.

`[\DOPassive]`

Digital Output Passive

Data type: `switch`

This switch overrides the default behavior when using `DOBreak` optional argument. Instead of reacting when signal is set to 1 (or already 1), the instruction should continue in the error handler (if no `BreakFlag` is used) when the signal `DOBreak` is set to 0 (or already is 0). The constant `ERR_TP_DOBREAK` can be used to test whether or not this has occurred.

`[\PersBoolBreak]`

Persistent Boolean Break

Data type: `bool`

The persistent boolean that may interrupt the operator dialog. If no button is selected when the persistent boolean is set to TRUE (or is already TRUE) then the program continues to execute in the error handler unless the `BreakFlag` is used (see below). The constant `ERR_TP_PERSBOOLBREAK` can be used to test whether or not this has occurred.

`[\PersBoolPassive]`

Persistent Boolean Passive

Data type: `switch`

This switch overrides the default behavior when using `PersBoolBreak` optional argument. Instead of reacting when persistent boolean is set to TRUE (or already TRUE), the instruction should continue in the error handler (if no `BreakFlag` is

Continues on next page

used) when the persistent boolean PersBoolBreak is set to FALSE (or already is FALSE). The constant ERR_TP_PERSBOOLBREAK can be used to test whether or not this has occurred.

[\BreakFlag]

Data type: errnum

A variable that will hold the error code if MaxTime, DIBreak, DOBreak, or PersBoolBreak is used. If this optional variable is omitted then the error handler will be executed. The constants ERR_TP_MAXTIME, ERR_TP_DIBREAK, ERR_TP_DOBREAK, and ERR_TP_PERSBOOLBREAK can be used to select the reason.

[\UIActiveSignal]

Data type: signaldo

The digital output signal used in optional argument UIActiveSignal is set to 1 when the message box is activated on the FlexPendant. When the user selection has been done and the execution continue, the signal is set to 0 again.

No supervision of stop or restart exist. The signal is set to 0 when the function is ready, or when PP is moved.

C.2.10 MT_UIDNumEntry - Shows a localized dnum entry

Usage

MT_UIDNumEntry is used to show a UI dnum entry box on Flexpendant. Texts are fetched from correct message, use also wild card texts if wanted. All other arguments have the exact same functionality as the corresponding standard function UIDnumEntry.

Basic examples

The following example illustrates the function MT_UIDNumEntry.

Example 1

```
VAR dnum answer;
answer:=MT_UIDNumEntry("myMessage",\WildCardTexts:=["text 1","text
     2"]);
```

This shows a UI dnum entry box with header and text fetched from the message that corresponds to the Identifier myMessage. The wild cards {1} and {2} in the texts will be replaced by “text 1” and “text 2” respectively.

Return value

Data type: dnum

Arguments

```
MT_UIDnumEntry (Identifier , [\WildCardTexts{*}] , [\Wrap] , [\Icon]
     , [\InitValue] ,[\MinValue] , [\MaxValue] , [\AsInteger] ,
     [\MaxTime] , [\DIBreak] , [\DIPassive] , [\DOBreak] ,
     [\DOPassive] , [\PersBoolBreak] , [\PersBoolPassive] ,
     [\BreakFlag] , [\UIActiveSignal])
```

Identifier

Data type: string

System unique string that gives the possibility to point to a specific message.

\WildCardTexts{*}

Data type: string array

String array of wild cards text to use for this message.

[\Wrap]

Data type: switch

If selected, all the specified strings in the argument \MsgArray will be concatenated to one string with a single space between each individual string, and spread out on as few lines as possible.

Default, each string in the argument \MsgArray will be on a separate line on the display.

[\Icon]

Data type: icondata

Defines the icon to be displayed. Only one of the predefined icons of type icondata can be used. See *Predefined data on page 371*.

Continues on next page

Default no icon.

`[\InitValue]`

Data type: `dnum`

Initial value that is displayed in the entry box.

`[\MinValue]`

Data type: `dnum`

The minimum value for the return value.

`[\MaxValue]`

Data type: `dnum`

The maximum value for the return value.

`[\AsInteger]`

Data type: `switch`

Eliminates the decimal point from the number pad to ensure that the return value is an integer.

`[\MaxTime]`

Data type: `num`

The maximum amount of time in seconds that program execution waits. If the OK button is not pressed within this time, the program continues to execute in the error handler unless the `BreakFlag` is used (see below). The constant `ERR_TP_MAXTIME` can be used to test whether or not the maximum time has elapsed.

`[\DIBreak]`

Digital Input Break

Data type: `signaldi`

The digital input signal that may interrupt the operator dialog. If the OK button is not pressed before the signal is set to 1 (or is already 1) then the program continues to execute in the error handler unless the `BreakFlag` is used (see below). The constant `ERR_TP_DIBREAK` can be used to test whether or not this has occurred.

`[\DIPassive]`

Digital Input Passive

Data type: `switch`

This switch overrides the default behavior when using `DIBreak` optional argument. Instead of reacting when signal is set to 1 (or already 1), the instruction should continue in the error handler (if no `BreakFlag` is used) when the signal `DIBreak` is set to 0 (or already is 0). The constant `ERR_TP_DIBREAK` can be used to test whether or not this has occurred.

`[\DOBreak]`

Digital Output Break

Data type: `signaldo`

Continues on next page

The digital output signal that may interrupt the operator dialog. If the OK button is not pressed before the signal is set to 1 (or is already 1) then the program continues to execute in the error handler unless the `BreakFlag` is used (see below). The constant `ERR_TP_DOBREAK` can be used to test whether or not this has occurred.

`[\DOPassive]`

Digital Output Passive

Data type: `switch`

This switch overrides the default behavior when using `DOBreak` optional argument. Instead of reacting when signal is set to 1 (or already 1), the instruction should continue in the error handler (if no `BreakFlag` is used) when the signal `DOBreak` is set to 0 (or already is 0). The constant `ERR_TP_DOBREAK` can be used to test whether or not this has occurred.

`[\PersBoolBreak]`

Persistent Boolean Break

Data type: `bool`

The persistent boolean that may interrupt the operator dialog. If no button is selected when the persistent boolean is set to TRUE (or is already TRUE) then the program continues to execute in the error handler unless the `BreakFlag` is used (see below). The constant `ERR_TP_PERSBOOLBREAK` can be used to test whether or not this has occurred.

`[\PersBoolPassive]`

Persistent Boolean Passive

Data type: `switch`

This switch overrides the default behavior when using `PersBoolBreak` optional argument. Instead of reacting when persistent boolean is set to TRUE (or already TRUE), the instruction should continue in the error handler (if no `BreakFlag` is used) when the persistent boolean `PersBoolBreak` is set to FALSE (or already is FALSE). The constant `ERR_TP_PERSBOOLBREAK` can be used to test whether or not this has occurred.

`[\BreakFlag]`

Data type: `errnum`

A variable that will hold the error code if `MaxTime`, `DIBreak`, `DOBreak`, or `PersBoolBreak` is used. If this optional variable is omitted then the error handler will be executed. The constants `ERR_TP_MAXTIME, ERR_TP_DIBREAK`, `ERR_TP_DOBREAK`, and `ERR_TP_PERSBOOLBREAK` can be used to select the reason.

`[\UIActiveSignal]`

Data type: `signaldo`

The digital output signal used in optional argument `UIActiveSignal` is set to 1 when the message box is activated on the FlexPendant. When the user selection has been done and the execution continue, the signal is set to 0 again.

No supervision of stop or restart exist. The signal is set to 0 when the function is ready, or when PP is moved.

Continues on next page

Predefined data

```
!Icons:
  CONST icondata iconNone := 0;
  CONST icondata iconInfo := 1;
  CONST icondata iconWarning := 2;
  CONST icondata iconError := 3;
```

C.2.11 MT_UIMessageBox - Show a message box in localized language

Usage

MT_UIMessageBox is used to show a UI message box on Flexpendant. Texts are fetched from correct message, use also wild card texts if wanted. Except for the header and body texts, all other arguments available in UIMessageBox is available here to.

Basic examples

The following example illustrates the function MT_UIMessageBox.

Example 1

```
VAR btnres answer;
answer:=MT_UIMessageBox("myMessage",\WildCardTexts:=["text 1","text
     2"]);
```

This will show a UI message box on the Flexpendant with header and text fetched from the message corresponding to identifier “myMessage”. Wild cards {1} and {2} will be replaced with “text 1” and “text 2” correspondingly. The rest will be according to the standard UIMessageBox function without additional arguments.

Return value

Data type: btnres

Arguments

```
UIMessageBox ( Identifier , [\WildCardTexts{*}] [\Wrap][\Buttons]
     | [\BtnArray] [\DefaultBtn] [\Icon][\Image] [\MaxTime]
     [\DIBreak] [\DIPassive] [\DOBreak] [\DOPassive]
     [\PersBoolBreak] [\PersBoolPassive] [\BreakFlag]
     [\UIActiveSignal])
```

Identifier

Data type: string

System unique string that gives the possibility to point to a specific message.

\WildCardTexts{*}

Data type: string array

String array of wild cards text to use for this message.

[\Wrap]

Data type: switch

If selected, all the specified strings in the argument \MsgArray will be concatenated to one string with single spaces between each individual string and spread out on as few lines as possible.

Default, each string in the argument \MsgArray will be on separate line on the display.

[\Buttons]

Data type: buttondata

Continues on next page

Defines the push buttons to be displayed. Only one of the predefined buttons combination of type buttondata can be used. See *Predefined data on page 375*.

Default, the system displays the OK button.

`[\BtnArray]`

Button Array

Data type: `string`

Own definition of push buttons stored in an array of strings. This function returns the array index when corresponding string is selected.

Only one of parameter `\Buttons` or `\BtnArray` can be used at the same time.

Max. 5 buttons with 42 characters each.

`[\DefaultBtn]`

Default Button

Data type: `btnres`

Allows to specify a value that should be returned if the message box is interrupted by `\MaxTime`, `\DIBreak`, or `\DOBreak`. It's possible to specify the predefined symbolic constant of type `btnres` or any user defined value. See *Predefined data on page 375*.

`[\Icon]`

Data type: `icondata`

Defines the icon to be displayed. Only one of the predefined icons of type `icondata` can be used. See *Predefined data on page 375*.

Default, no icon.

`[\Image]`

Data type: `string`

The name of the image that should be used. To launch own images, the images has to be placed in the `HOME:` directory in the active system or directly in the active system.

The recommendation is to place the files in the `HOME:` directory so that they are saved if a Backup and Restore is done.

A **Restart** is required and then the FlexPendant loads the images.

The image that will be shown can have the width of 185 pixels and the height of 300 pixels. If the image is bigger, only 185 * 300 pixels of the image will be shown starting at the top left of the image.

No exact value can be specified on the size that an image can have or the amount of images that can be loaded to the FlexPendant. It depends on the size of other files loaded to the FlexPendant. The program execution will just continue if an image is used that has not been loaded to the FlexPendant.

`[\MaxTime]`

Data type: `num`

The maximum amount of time in seconds that program execution waits. If no button is selected within this time, the program continues to execute in the error handler

Continues on next page

unless the BreakFlag is used (see below). The constant ERR_TP_MAXTIME can be used to test whether or not the maximum time has elapsed.

[\DIBreak]

Digital Input Break

Data type: signaldi

The digital input signal that may interrupt the operator dialog. If no button is selected when the signal is set to 1 (or is already 1) then the program continues to execute in the error handler, unless the BreakFlag is used (see below). The constant ERR_TP_DIBREAK can be used to test whether or not this has occurred.

[\DIPassive]

Digital Input Passive

Data type: switch

This switch overrides the default behavior when using DIBreak optional argument. Instead of reacting when signal is set to 1 (or already 1), the instruction should continue in the error handler (if no BreakFlag is used) when the signal DIBreak is set to 0 (or already is 0). The constant ERR_TP_DIBREAK can be used to test whether or not this has occurred.

[\DOBreak]

Digital Output Break

Data type: signaldo

The digital output signal that may interrupt the operator dialog. If no button is selected when the signal is set to 1 (or is already 1) then the program continues to execute in the error handler, unless the BreakFlag is used (see below). The constant ERR_TP_DOBREAK can be used to test whether or not this has occurred.

[\DOPassive]

Digital Output Passive

Data type: switch

This switch overrides the default behavior when using DOBreak optional argument. Instead of reacting when signal is set to 1 (or already 1), the instruction should continue in the error handler (if no BreakFlag is used) when the signal DOBreak is set to 0 (or already is 0). The constant ERR_TP_DOBREAK can be used to test whether or not this has occurred.

[\PersBoolBreak]

Persistent Boolean Break

Data type: bool

The persistent boolean that may interrupt the operator dialog. If no button is selected when the persistent boolean is set to TRUE (or is already TRUE) then the program continues to execute in the error handler unless the BreakFlag is used (see below). The constant ERR_TP_PERSBOOLBREAK can be used to test whether or not this has occurred.

Continues on next page

[\PersBoolPassive]

Persistent Boolean Passive

Data type: switch

This switch overrides the default behavior when using PersBoolBreak optional argument. Instead of reacting when persistent boolean is set to TRUE (or already TRUE), the instruction should continue in the error handler (if no BreakFlag is used) when the persistent boolean PersBoolBreak is set to FALSE (or already is FALSE). The constant ERR_TP_PERSBOOLBREAK can be used to test whether or not this has occurred.

[\BreakFlag]

Data type: errnum

A variable that will hold the error code if MaxTime, DIBreak, DOBreak, or PersBoolBreak is used. If this optional variable is omitted then the error handler will be executed. The constants ERR_TP_MAXTIME, ERR_TP_DIBREAK, ERR_TP_DOBREAK, and ERR_TP_PERSBOOLBREAK can be used to select the reason.

[\UIActiveSignal]

Data type: signaldo

The digital output signal used in optional argument UIActiveSignal is set to 1 when the message box is activated on the FlexPendant. When the user selection has been done and the execution continue, the signal is set to 0 again.

No supervision of stop or restart exist. The signal is set to 0 when the function is ready, or when PP is moved.

Predefined data

```
!Icons:
  CONST icondata iconNone := 0;
  CONST icondata iconInfo := 1;
  CONST icondata iconWarning := 2;
  CONST icondata iconError := 3;
!Buttons:
  CONST buttondata btnNone := -1;
  CONST buttondata btnOK := 0;
  CONST buttondata btnAbrtRtryIgn := 1;
  CONST buttondata btnOKCancel := 2;
  CONST buttondata btnRetryCancel := 3;
  CONST buttondata btnYesNo := 4;
  CONST buttondata btnYesNoCancel := 5;
!Results:
  CONST btnres resUnkwn := 0;
  CONST btnres resOK := 1;
  CONST btnres resAbort := 2;
  CONST btnres resRetry := 3;
  CONST btnres resIgnore := 4;
  CONST btnres resCancel := 5;
  CONST btnres resYes := 6;
  CONST btnres resNo := 7;
```

C.2.12 MT_UINumEntry - Shows a num entry box in localized language

Usage

`MT_UINumEntry` is used to show a UI num entry box on Flexpendant. Texts are fetched from correct message, use also wild card texts if wanted. All other arguments have the exact same functionality as the corresponding standard function UINumEntry.

Basic examples

The following example illustrates the function `MT_UINumEntry`.

Example 1

```
VAR num answer;
answer:=MT_UINumEntry("myMessage",\WildCardTexts:=["text 1","text
     2"]);
```

This shows a UI num entry box with header and text fetched from the message that corresponds to the Identifier myMessage. The wild cards {1} and {2} in the texts will be replaced by “text 1” and “text 2” respectively.

Return value

Data type: `num`

Arguments

```
MT_UINumEntry (Identifier , [\WildCardTexts{*}] , [\Wrap] , [\Icon]
     , [\InitValue] ,[\MinValue] , [\MaxValue] , [\AsInteger] ,
     [\MaxTime] , [\DIBreak] , [\DIPassive] , [\DOBreak] ,
     [\DOPassive] , [\PersBoolBreak] , [\PersBoolPassive] ,
     [\BreakFlag] , [\UIActiveSignal])
```

`Identifier`

Data type: `string`

System unique string that gives the possibility to point to a specific message.

`\WildCardTexts{*}`

Data type: `string array`

String array of wild cards text to use for this message.

`[\Wrap]`

Data type: `switch`

If selected, all the specified strings in the argument `\MsgArray` will be concatenated to one string with a single space between each individual string, and spread out on as few lines as possible.

Default, each string in the argument `\MsgArray` will be on a separate line on the display.

`[\Icon]`

Data type: `icondata`

Defines the icon to be displayed. Only one of the predefined icons of type `icondata` can be used. See *Predefined data on page 379*.

Continues on next page

Default no icon.

`[\InitValue]`

Data type: `num`

Initial value that is displayed in the entry box.

`[\MinValue]`

Data type: `num`

The minimum value for the return value.

`[\MaxValue]`

Data type: `num`

The maximum value for the return value.

`[\AsInteger]`

Data type: `switch`

Eliminates the decimal point from the number pad to ensure that the return value is an integer.

`[\MaxTime]`

Data type: `num`

The maximum amount of time in seconds that program execution waits. If the OK button is not pressed within this time, the program continues to execute in the error handler unless the `BreakFlag` is used (see below). The constant `ERR_TP_MAXTIME` can be used to test whether or not the maximum time has elapsed.

`[\DIBreak]`

Digital Input Break

Data type: `signaldi`

The digital input signal that may interrupt the operator dialog. If the OK button is not pressed before the signal is set to 1 (or is already 1) then the program continues to execute in the error handler, unless the `BreakFlag` is used (see below). The constant `ERR_TP_DIBREAK` can be used to test whether or not this has occurred.

`[\DIPassive]`

Digital Input Passive

Data type: `switch`

This switch overrides the default behavior when using `DIBreak` optional argument. Instead of reacting when signal is set to 1 (or already 1), the instruction should continue in the error handler (if no `BreakFlag` is used) when the signal `DIBreak` is set to 0 (or already is 0). The constant `ERR_TP_DIBREAK` can be used to test whether or not this has occurred.

`[\DOBreak]`

Digital Output Break

Data type: `signaldo`

Continues on next page

The digital output signal that may interrupt the operator dialog. If the OK button is not pressed before the signal is set to 1 (or is already 1) then the program continues to execute in the error handler, unless the `BreakFlag` is used (see below). The constant `ERR_TP_DOBREAK` can be used to test whether or not this has occurred.

`[\DOPassive]`

Digital Output Passive

Data type: `switch`

This switch overrides the default behavior when using `DOBreak` optional argument. Instead of reacting when signal is set to 1 (or already 1), the instruction should continue in the error handler (if no `BreakFlag` is used) when the signal `DOBreak` is set to 0 (or already is 0). The constant `ERR_TP_DOBREAK` can be used to test whether or not this has occurred.

`[\PersBoolBreak]`

Persistent Boolean Break

Data type: `bool`

The persistent boolean that may interrupt the operator dialog. If no button is selected when the persistent boolean is set to TRUE (or is already TRUE) then the program continues to execute in the error handler unless the `BreakFlag` is used (see below). The constant `ERR_TP_PERSBOOLBREAK` can be used to test whether or not this has occurred.

`[\PersBoolPassive]`

Persistent Boolean Passive

Data type: `switch`

This switch overrides the default behavior when using `PersBoolBreak` optional argument. Instead of reacting when persistent boolean is set to TRUE (or already TRUE), the instruction should continue in the error handler (if no `BreakFlag` is used) when the persistent boolean `PersBoolBreak` is set to FALSE (or already is FALSE). The constant `ERR_TP_PERSBOOLBREAK` can be used to test whether or not this has occurred.

`[\BreakFlag]`

Data type: `errnum`

A variable that will hold the error code if `MaxTime`, `DIBreak`, `DOBreak`, or `PersBoolBreak` is used. If this optional variable is omitted then the error handler will be executed. The constants `ERR_TP_MAXTIME, ERR_TP_DIBREAK`, `ERR_TP_DOBREAK`, and `ERR_TP_PERSBOOLBREAK` can be used to select the reason.

`[\UIActiveSignal]`

Data type: `signaldo`

The digital output signal used in optional argument `UIActiveSignal` is set to 1 when the message box is activated on the FlexPendant. When the user selection has been done and the execution continue, the signal is set to 0 again.

No supervision of stop or restart exist. The signal is set to 0 when the function is ready, or when PP is moved.

Continues on next page

Predefined data

```
!Icons:
  CONST icondata iconNone := 0;
  CONST icondata iconInfo := 1;
  CONST icondata iconWarning := 2;
  CONST icondata iconError := 3;
```

C.3 Data types

C.3.1 mtaxisrange - specifies an axis with an axis range

Usage

`mtaxisrange` is used to specify an axis and a range for the axis. Used to define for which axis values the robot is within a zone.

Components

The data type `mtaxisrange` has the following components:

`Axis`

Data type: `num`

Number of the robot axis to use. 1-6 normal robot axis, 7-12 external axis.

`Range`

Data type: `mtrange`

The range for this axis.

C.3.2 mtloglevel - define a log level

Usage

`mtloglevel` is used to define a log level.

Predefined data

The following symbolic constants of the data type `mtloglevel` are predefined and can be used when specifying a condition for the instructions `MT_Log` and `MT_LogEnable`.

Value	Symbolic constant	Comment
1	`MT_LVL_ERROR`	
2	`MT_LVL_WARNING`	
3	`MT_LVL_INFO`	
4	`MT_LVL_DEBUG`	
5	`MT_LVL_USER`	

Characteristics

`mtloglevel` is an alias data type for `num` and consequently inherits its characteristics.

C.3.3 mtmsgdata - alarm and message definition data

Usage

`mtmsgdata` is used to describe all messages and alarms in the system.

Components

The data type `mtmsgdata` has the following components:

`Identifier`

Data type: `string`

System unique string that gives the possibility to point to a specific message.

`Category`

Data type: `icondata`

Describes which alarm category the alarm has. Messages that’s not of any alarm type has an empty category data.

`StationName`

Data type: `string`

Name of the station to which the alarm belongs to. This could be used to sort or group alarms better, for example in an HMI.

`Number`

Data type: `num`

Message number will be shown as part of the alarm number in alarm lists. Can also be a way to keep structure of the messages, but not mandatory to use this.

`Domain`

Data type: `num`

Message domain number. This is not mandatory, could be used to group alarms in a different way (maybe a bigger type of group than station level).

`Header`

Data type: `string`

Header text for the message.

`Text1`

Data type: `string`

First text line of body text for the message.

`Text2`

Data type: `string`

Second text line of body text for the message.

`Text3`

Data type: `string`

Third text line of body text for the message.

Continues on next page

`Text4`

Data type: `string`

Forth text line of body text for the message.

`Text5`

Data type: `string`

Fifth text line of body text for the message.

`Text6`

Data type: `string`

Sixth text line of body text for the message.

`Text7`

Data type: `string`

Seventh text line of body text for the message.

`Text8`

Data type: `string`

Eighth text line of body text for the message.

C.3.4 mtpartdata - Describes a part

Usage

`mtpartdata` is used to describe a part.

Components

The data type `mtpartdata` has the following components:

`ProgCode`

Data type: `dnum`

Unique number for this part to identify it.

`TypeCode`

Data type: `dnum`

Type coding that can be used in the robot program if needed.

`AuxCode`

Data type: `dnum`

Alternative program number to address for example, a vision system which uses program numbers that are different from the robot's program numbers.

`ToolCode`

Data type: `num`

Currently not used.

`MachineCode`

Data type: `dnum`

Code for a machine, served by robot (for example, form code of an Injection Moulding Machine). Only use if it makes sense in the project.

`RoutineName`

Data type: `string`

Currently not used.

`RunInTasks`

Data type: `string`

Currently not used.

C.3.5 mtpartdeviation - Description

Usage

`mtpartdeviation` is used to describe deviations for a part of a specific state.

Components

The data type `mtpartdeviation` has the following components:

`X`

Data type: `num`

Deviation in X for this state.

`Y`

Data type: `num`

Deviation in Y for this state.

`Z`

Data type: `num`

Deviation in Z for this state.

`RotX`

Data type: `num`

Deviation in rotation X for this state.

`RotY`

Data type: `num`

Deviation in rotation Y for this state.

`RotZ`

Data type: `num`

Deviation in rotation Z for this state.

C.3.6 mtpartdimension - Description

Usage

`mtpartdimension` is used to describe the physical dimensions of a part in a specific state.

Components

The data type `mtpartdimension` has the following components:

`Height`

Data type: `num`

Height of part.

`Length`

Data type: `num`

Length of part.

`Width`

Data type: `num`

Width of part.

C.3.7 mtpartproperties - Description

Usage

`mtpartproperties` is used to describe a set of properties for a part state for a specific part. The properties that could be handled inside this are loaddata, tooldata, dimensions and deviations. Note that both loaddata and tooldata is just string references to the actual data declarations for those.

Components

The data type `mtpartproperties` has the following components:

`State`

Data type: `mtpartstate`

The state for which this property set is valid.

`PartLoadName`

Data type: `string`

Name of the loaddata declaration that fits for this part state. Optional to use.

`ToolName`

Data type: `string`

Name of the tooldata used to handle the part of the current state. Optional to use.

`Dimension`

Data type: `mtpartdimension`

Describes dimensions of the part in this state. Optional to use.

`Deviation`

Data type: `mtpartdeviation`

Describes deviations of the part in this state. Optional to use.

C.3.8 mtpartstate - Describes a part state

Usage

`mtpartstate` is used to describe current state of a part.

Predefined data

The following symbolic constants of the data type `mtpartstate` are predefined and can be used when specifying a condition for instructions like `MT_SetPartState` and `MT_GetPartState`.

Value	Symbolic constant	Comment
0	`psEMPTY`	
500	`psRAW`	
501	`psMACHINED`	
502	`psFINISHED`	
503	`psCOOLED`	

Characteristics

`mtpartstate` is an alias data type for `num` and consequently inherits its characteristics.

C.3.9 mtparttracker - describes all data for a part tracker

Usage

`mtparttracker` is used to hold all data handled in a part tracker. Each defined station must have a corresponding mtparttracker definition to make it work. The definition should be an array of wanted size if there is more than one tracker in the station. Note, maximum trackers in a station is 20.

Basic examples

The following example illustrates the data type `mtparttracker`.

Example 1

```
PERS mtstationdata sdRobot;
PERS mtparttracker sdRobot_Tracker{2};
```

First a station definition for the robot and the corresponding tracker definition if part trackers should be used.

Note! The tracker definition must start with the station name and end with “_Tracker”. In the example the robot have two grippers and therefore uses an array with 2 places which gives two trackers for the station. Also note that when the definition is made, the components of the defined data could be blank, those are filled later on when using the part tracker instructions.

Components

The data type `mtparttracker` has the following components:

`Station`

Data type: `string`

Name of the mtstationdata declaration, where the part tracker belongs to.

`Part`

Data type: `string`

Name of the mtpartdata declaration, which is currently connected to the part tracker.

`State`

Data type: `string`

Name of the mtpartstate declaration, which holds the state info of the part.

`UserDef`

Data type: `string`

String for user defined content, e.g. information, parameters, name of routines to be executed etcetera. The content is, as the name says, user-defined and so must be parsed by the application program.

C.3.10 mtrange - describes a robot axis range

Usage

`mtrange` is used to describe start (Min) and stop (Max) angles for a robot axis.

Components

The data type `mtrange` has the following components:

`Min`

Data type: `num`

Start value of the range.

`Max`

Data type: `num`

Stop value of the range.

C.3.11 mtstationdata - describes a station

Usage

`mtstationdata` is used to describe a station in the cell. The main key is to just have a station definition. The system will search for all definitions and handle those as stations. The components will not have any real usage in this release of the add-in.

Components

The data type `mtstationdata` has the following components:

`IOEnabled`

Data type: `string`

Name of the digital input or output through which the station is

selected or deselected. Here, the "high" state of the signal means "Station selected" and the "low" state means "Station deselected".

`IOInvert`

Data type: `bool`

Inversion flag, which will invert the meaning of the value in arguments IOEnabled.

`HMIEnabled`

Data type: `bool`

Station has been selected (TRUE) or deselected (FALSE) by the HMI.

Only if no signal name is defined, then the station can be selected or deselected through the HMI.

C.3.12 mttargetdata - describes a robot zone

Usage

`mttargetdata` is used to define the different zones in the cell. Normally one zone is defined for each station but it doesn’t have to be like that. To define when the robot is within a specific zone a mtrange for axis 1 is always used, i.e. a start and stop value for axis 1. In addition there is up to four additional mtaxisrange that could be optionally used where user specify which axis and the range for that axis to also consider for the zone. Note that robot must be within all ranges (between start and stop for all the specified axis) to be in that zone. If an optional axisrange shouldn’t be used, just state “0” for the axis number.

Basic examples

The following example illustrates the data type `mttargetdata`.

Example 1

```
CONST mttargetdata
      ZONE_HOME:=[1,"pHome",[-80,70],[3,[-30,30]],[4,[10,40]],[0,[0,0]],[0,[0,0]]];
```

In this example ZONE_HOME is given unique index number 1 and connects to via position “pHome”. To be considered in the zone, the robot must have axis 1 between -80 and 70 degrees. And axis 3 between -30 to 30 degrees and axis 4 between 10 to 40 degrees.

Components

The data type `mttargetdata` has the following components:

`Index`

Data type: `num`

Unique index number for this mttargetdata, the number doesn’t matter as long as it’s unique.

`PosName`

Data type: `string`

The via position connected to this target zone. This gives an indirect connection between the target zone and the actual via position. Note that spelling is very important here!

`RangeAxis1`

Data type: `mtrange`

The range used for axis 1.

`Range2`

Data type: `mtaxisrange`

The axis and range for optional axis check. Set axis number to 0 if not using this.

`Range3`

Data type: `mtaxisrange`

The axis and range for optional axis check. Set axis number to 0 if not using this.

Continues on next page

`Range4`

Data type: `mtaxisrange`

The axis and range for optional axis check. Set axis number to 0 if not using this.

`Range5`

Data type: `mtaxisrange`

The axis and range for optional axis check. Set axis number to 0 if not using this.

Predefined data

The following symbolic constants of the data type `mttargetdata` are predefined and can be used when specifying a condition for instructions like `MT_GetCurrentZone`.

Value	Symbolic constant	Comment
[9999,"", [-9E9,9E9], [0,[0,0]], [0,[0,0]], [0,[0,0]], [0,[0,0]]]	`ZONE_UNDEFINED`	

ABB AB
Robotics & Discrete Automation
S-721 68 VÄSTERÅS, Sweden
Telephone +46 (0) 21 344 400

ABB AS
Robotics & Discrete Automation
Nordlysvegen 7, N-4340 BRYNE, Norway
Box 265, N-4349 BRYNE, Norway
Telephone: +47 22 87 2000

ABB Engineering (Shanghai) Ltd.
Robotics & Discrete Automation
No. 4528 Kangxin Highway
PuDong District
SHANGHAI 201319, China
Telephone: +86 21 6105 6666

ABB Inc.
Robotics & Discrete Automation
1250 Brown Road
Auburn Hills, MI 48326
USA
Telephone: +1 248 391 9000

abb.com/robotics