HALCON
a product of MVTec

HDevelop User's Guide



HALCON 21.05 *Progress*

MVTec Software GmbH

Building Vision for Business

HDevelop, the interactive development environment of HALCON, Version 21.05.0.0

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without prior written permission of the publisher.

Copyright © 1997-2021 by MVTec Software GmbH, München, Germany



Protected by the following patents: US 7,062,093, US 7,239,929, US 7,751,625, US 7,953,290, US 7,953,291, US 8,260,059, US 8,379,014, US 8,830,229. Further patents pending.

Microsoft, Windows, Windows Server 2008/2012/2012 R2/2016, Windows 7/8/8.1/10, Microsoft .NET, Visual C++, and Visual Basic are either trademarks or registered trademarks of Microsoft Corporation.
OpenGL is a trademark of Silicon Graphics, Inc.
macOS and OpenCL are trademarks of Apple Inc.

All other nationally and internationally recognized trademarks and tradenames are hereby recognized.

More information about HALCON can be found at: http://www.halcon.com/

About This Manual

This manual is a guide to HDevelop, the interactive development environment for the HALCON machine vision library. It is intended for users who want to use HDevelop as a convenient gateway to the HALCON library or who want to deploy and test machine vision applications with it. It is not an introduction to the HALCON machine vision library. A working knowledge of the concepts of HALCON is assumed. Please refer to the Quick Guide to become acquainted with HALCON. In-depth knowledge of image processing is not required to start working with HDevelop.

The manual is divided into the following chapters:

- **Introducing HDevelop**
  This chapter explains the basic concepts of HDevelop.
- **Getting Started**
  This chapter explains how to start HDevelop. It provides a quick overview of the graphical user interface, and shows you how to run the supplied example programs.
- **Acquiring Images with HDevelop**
  This chapter explains the fundamental part of machine vision applications – how to acquire images.
- **Programming HDevelop**
  This chapter explains how to develop applications in HDevelop.
- **HDevelop Procedures**
  This chapter introduces the concepts of breaking a large program into small maintainable and reusable units.
- **Graphical User Interface**
  This chapter explains the graphical user interface of HDevelop and how to interact with it.
- **HDevelop Assistants**
  This chapter describes how to use the machine vision assistants of HDevelop.
- **HDevelop Language**
  This chapter explains the syntax and semantics of the language used in HDevelop expressions.
- **Remote Debugging**
  This chapter explains how remote debugging is supported.
- **Code Export**
  This chapter explains the export of an HDevelop program to C, C++, Visual Basic .NET, or C#.
- **Appendix**
  The appendix contains a glossary, a list of predefined color names, a list for command line usage, and a list of keyboard shortcuts.

Contents

1 Introducing HDevelop **11**
- 1.1 Facts about HDevelop . . . 11
- 1.2 HDevelop XL . . . 12
- 1.3 Terminology & Usage . . . 12

2 Getting Started **15**
- 2.1 Running HDevelop . . . 15
- 2.2 Start Dialog . . . 16
- 2.3 User Interface . . . 16
- 2.4 Organizing HDevelop's Workspace . . . 17
- 2.5 Running Example Programs . . . 18

3 Acquiring Images with HDevelop **23**
- 3.1 Reading Images From Files . . . 23
- 3.2 Viewing Images . . . 24
- 3.3 Image Acquisition Assistant . . . 24
  - 3.3.1 Acquiring Images From Files or Directories . . . 24
  - 3.3.2 Acquiring Images Through Image Acquisition Interfaces . . . 26
  - 3.3.3 Modifying the Generated Code . . . 29

4 Programming HDevelop **31**
- 4.1 Start a New Program . . . 31
- 4.2 Enter an Operator . . . 32
- 4.3 Specify Parameters . . . 32
- 4.4 Getting Help . . . 33
- 4.5 Add Additional Program Lines . . . 33
- 4.6 Understanding the Image Display . . . 34
- 4.7 Inspecting Variables . . . 35
- 4.8 Improving the Threshold Using the Gray Histogram . . . 35
- 4.9 Edit Lines . . . 36
- 4.10 Re-Execute the Program . . . 37
- 4.11 Save the Program . . . 37
- 4.12 Selecting Regions Based on Features . . . 37
- 4.13 Open Graphics Window . . . 38
- 4.14 Looping Over the Results . . . 39
- 4.15 Summary . . . 40

5 HDevelop Procedures **41**
- 5.1 Procedure Types . . . 41
- 5.2 File Types . . . 42
  - 5.2.1 HDevelop Programs . . . 42
  - 5.2.2 Procedure Files . . . 42
  - 5.2.3 Libraries . . . 42
- 5.3 Procedure Scope . . . 42
- 5.4 Procedure Locations . . . 43
- 5.5 Procedure Resolution . . . 43
- 5.6 Protected Procedures . . . 45
- 5.7 Procedure Documentation . . . 45

5.8 Legacy Procedures . . . 45
5.9 Just-In-Time Compilation . . . 45

6 Graphical User Interface **49**
6.1 Main Window . . . 49
6.2 Menu . . . 51
6.2.1 Menu `File` . . . 51
6.2.2 Menu `Edit` . . . 59
6.2.3 Menu `Execute` . . . 82
6.2.4 Menu `Visualization` . . . 88
6.2.5 Menu `Procedures` . . . 99
6.2.6 Menu `Operators` . . . 101
6.2.7 Menu `Suggestions` . . . 105
6.2.8 Menu `Assistants` . . . 106
6.2.9 Menu `Window` . . . 106
6.2.10 Menu `Help` . . . 110
6.3 Tool Bar . . . 113
6.4 Program Window . . . 113
6.4.1 Program Window Actions . . . 114
6.4.2 Editing Programs . . . 115
6.4.3 Program Counter, Insert Cursor, and Breakpoints . . . 119
6.4.4 Context Menu . . . 119
6.4.5 Creating Procedures . . . 121
6.4.6 Editing Procedures . . . 126
6.4.7 Side Effects of Procedure Changes . . . 126
6.4.8 Providing Procedure Documentation . . . 128
6.4.9 Protecting a Procedure . . . 132
6.4.10 Profiler . . . 134
6.5 Operator Window . . . 138
6.5.1 Operator Name Field . . . 139
6.5.2 Parameter Display . . . 139
6.5.3 Control Buttons . . . 142
6.6 Variable Window . . . 142
6.6.1 Iconic Variables . . . 144
6.6.2 Control Variables . . . 147
6.7 Inspecting Variables . . . 148
6.7.1 Inspecting Tuples . . . 148
6.7.2 Inspecting Vectors . . . 150
6.7.3 Inspecting Handles . . . 151
6.7.4 Inspecting Matrices . . . 151
6.7.5 Inspecting Poses . . . 152
6.7.6 Inspecting Image Acquisition Device Handles . . . 152
6.7.7 Inspecting Functions . . . 152
6.7.8 Inspecting 3D Object Models . . . 157
6.8 Graphics Window . . . 158
6.9 Canvas Window . . . 161
6.10 ROI Window . . . 162
6.11 Help Window . . . 165
6.12 Zoom Window . . . 168
6.13 Interacting with Plot Windows . . . 169
6.13.1 Mouse-based Operations . . . 169
6.13.2 Setting the Plot Bounds Parametrically . . . 170
6.13.3 Plot Window Context Menus . . . 170
6.14 Gray Histogram Window . . . 173
6.14.1 Context Menus . . . 175
6.14.2 Interactive Visual Operations . . . 176
6.15 Feature Histogram Window . . . 178
6.16 Feature Inspection Window . . . 179
6.17 Line Profile Window . . . 180
6.17.1 ROI Menu of the Line Profile Window . . . 181

6.17.2 Line Profile Display . . . 182
6.17.3 Context Menus . . . 182
6.17.4 Line Profile Options . . . 183
6.17.5 Input and Output . . . 183
6.17.6 Statistics . . . 184
6.17.7 Focusing Your Camera: How to Test Images for Sharpness . . . 184
6.18 OCR Training File Browser . . . 185
6.18.1 Windows of the Training File Browser . . . 185
6.18.2 Steps for working with the OCR Training File Browser . . . 186
6.18.3 Actions within the Training File Browser . . . 187
6.19 Dialogs . . . 190
6.19.1 File Selection Dialog . . . 190
6.19.2 Unsaved Changes . . . 191

7 HDevelop Assistants 193
7.1 Image Acquisition Assistant . . . 194
7.1.1 Tab `Source` . . . 194
7.1.2 Tab `Connection` . . . 195
7.1.3 Tab `Parameters` . . . 196
7.1.4 Tab `Inspect` . . . 197
7.1.5 Tab `Code Generation` . . . 197
7.1.6 Menu Bar . . . 198
7.2 Calibration Assistant . . . 199
7.2.1 Introducing the Calibration Assistant of HDevelop . . . 199
7.2.2 How to Calibrate with the Calibration Assistant . . . 200
7.2.3 Results of the Calibration . . . 210
7.2.4 Generating Code . . . 211
7.2.5 Calibration Assistant Reference . . . 214
7.3 Matching Assistant . . . 217
7.3.1 Introducing the Matching Assistant of HDevelop . . . 217
7.3.2 How to Use the Matching Assistant of HDevelop . . . 218
7.3.3 Matching Assistant Reference . . . 223
7.4 Measure Assistant . . . 249
7.4.1 Introducing the Measure Assistant of HDevelop . . . 249
7.4.2 How to Use the Measure Assistant of HDevelop . . . 250
7.4.3 Results . . . 252
7.4.4 Code Generation . . . 253
7.4.5 Advanced Measuring Tasks . . . 254
7.4.6 Measure Assistant Reference . . . 258
7.5 OCR Assistant . . . 262
7.5.1 Introducing the OCR Assistant of HDevelop . . . 262
7.5.2 Setup . . . 264
7.5.3 Segmentation . . . 265
7.5.4 OCR Classifier . . . 268
7.5.5 Results . . . 272
7.5.6 Code Generation . . . 274
7.5.7 OCR Assistant Reference . . . 275

8 HDevelop Language 281
8.1 Basic Types of Parameters . . . 281
8.2 Control Types and Constants . . . 282
8.3 Variables . . . 284
8.3.1 Variable Types . . . 284
8.3.2 Scope of Variables (local or global) . . . 285
8.4 Operations on Iconic Objects . . . 286
8.5 Expressions for Input Control Parameters . . . 286
8.5.1 General Features of Tuple Operations . . . 286
8.5.2 Assignment . . . 287
8.5.3 Basic Tuple Operations . . . 289
8.5.4 Tuple Creation . . . 290

8.5.5 Type Operations . . . 291
8.5.6 Basic Arithmetic Operations . . . 291
8.5.7 Bit Operations . . . 292
8.5.8 String Operations . . . 292
8.5.9 Set Operations . . . 297
8.5.10 Comparison Operations . . . 297
8.5.11 Elementwise Comparison Operations . . . 298
8.5.12 Boolean Operations . . . 298
8.5.13 Trigonometric Functions . . . 299
8.5.14 Exponential Functions . . . 299
8.5.15 Numerical Functions . . . 299
8.5.16 Miscellaneous Functions . . . 301
8.5.17 Operation Precedence . . . 302
8.6 Vectors . . . 303
8.7 Reserved Words . . . 305
8.8 Control Flow Operators . . . 305
8.9 Error Handling . . . 311
8.9.1 Tracking the Return Value of Operator Calls . . . 311
8.9.2 Exception Handling . . . 312
8.10 Parallel Execution . . . 312
8.10.1 Starting a Subthread . . . 313
8.10.2 Waiting for Subthreads to Finish . . . 314
8.10.3 Execution of Threads in HDevelop . . . 315
8.10.4 Inspecting Threads . . . 316
8.10.5 Suspending and Resuming Threads . . . 317
8.11 Summary of HDevelop Tuple Operations . . . 317
8.12 HDevelop Error Codes . . . 321
8.13 Emergency Backup . . . 326

9 Remote Debugging **327**
9.1 Requirements . . . 327
9.2 Attaching to an External Application . . . 327
9.3 Debugging . . . 329
9.4 Handling of Procedures . . . 329
9.5 Handling of Protected Procedures . . . 329
9.6 Error Handling . . . 329
9.7 Threading . . . 330
9.8 Terminating a Remote Debug Session . . . 330
9.9 Limitations . . . 330

10 Code Export **331**
10.1 Exporting Library Projects . . . 331
10.1.1 Requirements . . . 331
10.1.2 Project Preparation . . . 332
10.1.3 Exporting a Library Project . . . 332
10.1.4 Using the Exported Library Project . . . 333
10.2 Exporting entire HDevelop Programs . . . 335
10.2.1 Code Generation for C++ . . . 335
10.2.2 Code Generation for C# (HALCON/.NET) . . . 338
10.2.3 Code Generation for Visual Basic .NET (HALCON/.NET) . . . 339
10.2.4 Code Generation for C . . . 341
10.2.5 General Aspects of Code Generation . . . 342

A Glossary **347**

B Color names **349**

C Command Line Usage **351**
C.1 HDevelop . . . 351
C.2 Hrun . . . 354

D Keyboard Shortcuts **355**
D.1 HDevelop 355
D.2 Program Window 358
D.3 HDevelop Help Window 358
D.4 Graphics Window 359
D.5 Plot Area 360
D.6 Variable Inspect 360
D.7 Handle Inspect 360
D.8 OCR Training File Browser 361

Index **363**

Chapter 1

Introducing HDevelop

HDevelop is a tool box for building machine vision applications. It facilitates rapid prototyping by offering a highly interactive programming environment for developing and testing machine vision applications. Based on the HALCON library, it is a versatile machine vision package suitable for product development, research, and education.

There are four basic ways to develop image analysis applications using HDevelop:

- *Rapid prototyping in the interactive environment HDevelop.*
  You can use HDevelop to find the optimal operators or parameters to solve your image analysis task, and then build the application using various programming languages, e.g., C, C++, C#, or Visual Basic .NET.
- *Development of an application that runs within HDevelop.*
  Using HDevelop, you can also develop a complete image analysis application and run it within the HDevelop environment. The example programs supplied with HDevelop can be used as building blocks for your own applications.
- *Execution of HDevelop programs or procedures using HDevEngine.*
  You can directly execute HDevelop programs or procedures from an application written in C++ or any language that can integrate .NET objects using HDevEngine. This is described in detail in the Programmer's Guide, part VI on page 129.
- *Export of an application as C, C++, Visual Basic .NET, or C# source code.*
  Finally, you can export an application developed in HDevelop as C, C++, Visual Basic .NET, or C# source code. This program can then be compiled and linked with the HALCON library so that it runs as a standalone (console) application. Of course, you can also extend the generated code or integrate it into existing software.

Let's start with some facts describing the main characteristics of HDevelop.

1.1 Facts about HDevelop

HDevelop actively supports your application development in many ways:

- ☐ With the graphical user interface of HDevelop, operators and iconic objects can be directly selected, analyzed, and changed within a single environment.
- ☐ HDevelop suggests operators for specific tasks. In addition, a thematically structured operator list helps you to find an appropriate operator quickly.
- ☐ An integrated online help contains information about each HALCON operator, such as a detailed description of the functionality, typical successor and predecessor operators, complexity of the operator, error handling, and examples of application. In addition, the online help provides a search facility that allows you to search the complete documentation of HALCON.
- ☐ HDevelop comprises a program interpreter with edit and debug functions. It supports standard programming features, such as procedures, loops, or conditional statements. Parameters can be changed even while the program is running.

- HDevelop immediately displays the results of operations. You can try different operators and/or parameters, and immediately see the effect on the screen. Moreover, you can preview the results of an operator without changing the program.
- Several graphical tools allow you to examine iconic and control data online. For example, you can extract shape and gray value features by simply clicking onto the objects in the graphics window, or inspect the histogram of an image interactively and apply real-time segmentation to select parameters.
- Built-in graphical assistants provide interactive interfaces to more complex machine vision tasks. The assistants can also generate HDevelop code in the current program.
- Variables with an automatic garbage collection are used to manage iconic objects or control values.
- HDevelop supports just-in-time compilation of procedures for optimized performance as well as calling procedures as separate sub-threads.

1.2 HDevelop XL

In addition to the standard HDevelop, there is also a variant called HDevelop XL, which is based on HALCON XL. The user interface is identical, but underneath HALCON XL is optimized for large images. In the remainder of this document, when we refer to HDevelop you can substitute HDevelop XL if that is the variant you will be using.

1.3 Terminology & Usage

HDevelop adheres to well-established conventions and usage patterns regarding its graphical user interface. Most of the terminology explained here will have become second nature to most users and may most likely be skimmed over.

Mouse Usage

click A single click with the left mouse button, e.g., to mark and select items or to activate buttons. To select multiple items, hold down the Ctrl key and click the desired items. To select many items from a list, click the first item, hold down the Shift key and click the last item. All intermediate items are then also selected.

double-click Two quick successive clicks with the left mouse button, e.g., to open dialogs of selected items. Double-clicks are mostly shortcuts for single clicks followed by an additional action.

right-click A single click with the right mouse button to access additional functionality of the user interface, e.g., context-sensitive menus. Clicking the right mouse button also ends interactive drawing functions in HDevelop.

drag Keeping the left mouse button pressed while moving the mouse and finally releasing the mouse button. Typically used to move items, resize windows, select multiple items at once, e.g., program lines, or to draw shapes.

drag-and-drop HDevelop supports drag-and-drop of image files and HDevelop programs from other applications. You can, e.g., drag an HDevelop program icon from a file browser and drop it on the HDevelop window to load it.

middle mouse button With three-button mice, the middle mouse button is used under Linux to paste text from the clipboard into text fields.

mouse wheel Most recent three-button mice combine the middle mouse button with a scrolling wheel. HDevelop supports the mouse wheel in many places. The mouse wheel operates the GUI element under the mouse cursor. Using the mouse wheel you can, for instance, quickly scroll large program listings, select values from lists or perform continuous zooming of displayed images. In general, windows that provide a scroll bar can be quickly scrolled with the mouse wheel. Furthermore, the values of spinner boxes (text fields that expect numerical data) can be decremented and incremented with the mouse wheel.

Keyboard Usage

HDevelop is very keyboard-friendly. Most functions of the graphical user interface that can be operated using the mouse can be accessed from the keyboard as well. Many of the most important functions are available through keyboard shortcuts, which are worthwhile memorizing. When programming with HDevelop, keeping both hands on the keyboard can increase the productivity. Therefore, many navigational tasks like selecting parameter fields or selecting values from lists can easily be done using just the keyboard. The most common keyboard functions are listed in the appendix D on page 355.

To make it easier for you to memorize the keyboard shortcuts, many of them are introduced by a common combination to indicate the context. For example, many shortcuts related to the graphics window are introduced by pressing Ctrl+Shift+G followed by another key, e.g., Ctrl+Shift+G,Del clears the graphics window. Because it is often easier to keep Ctrl+Shift pressed when hitting the second key the alternative Ctrl+Shift+G,Ctrl+Shift+Del is also allowed.

On macOS, Cmd is used instead of Ctrl.

Certain key combinations may conflict with keyboard functions of the operating system or the window manager. For example, using Ctrl+Alt+Cursor Keys in the graphics window pans the displayed image while displaying pixel information. Under Windows it may also change the screen orientation. See your system documentation on how to disable or change the conflicting key bindings in this case.

Abbreviations

BP breakpoint

IC insert cursor

GUI graphical user interface

PC program counter

XLD extended line description (see also chapter A on page 347)

Chapter 2

Getting Started

2.1 Running HDevelop

In the following it is assumed that HALCON has already been installed as described in the Installation Guide.

Windows

Under Windows, HDevelop is usually started from the "Apps" screen or from the "Start" menu:

`Start` ▷ `Programs` ▷ `MVTec HALCON 21.05` ▷ HDevelop

You can also start HDevelop from the Windows Command Prompt or from the `Start` ▷ `Run...` menu, making it easy to pass optional command line switches:

```
hdevelop
```

Linux

Under Linux, HDevelop is started from the shell:

```
hdevelop &
```

macOS

Under macOS, HDevelop is started from the Applications folder of the Finder.

2.2 Start Dialog

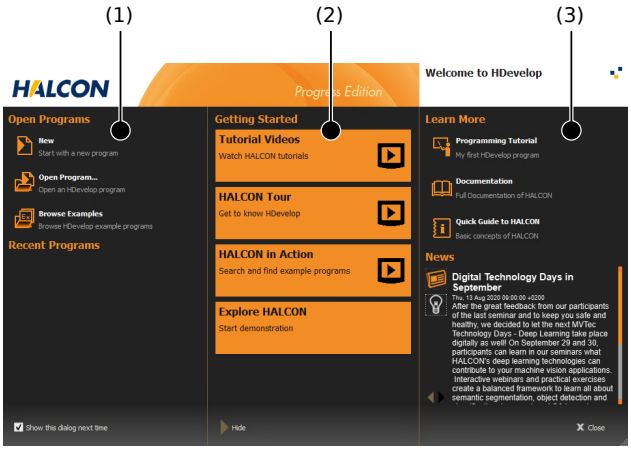


Figure 2.1: Start dialog.

The start dialog provides quick access to HDevelop programs (1), introductory material (2), and documentation (3). For a first introduction to HDevelop, try the links listed under “Getting Started”. To close the start dialog without any further action, press Esc. If you have accidentally closed the start dialog, it can be re-opened from HDevelop’s `Help` menu.

2.3 User Interface

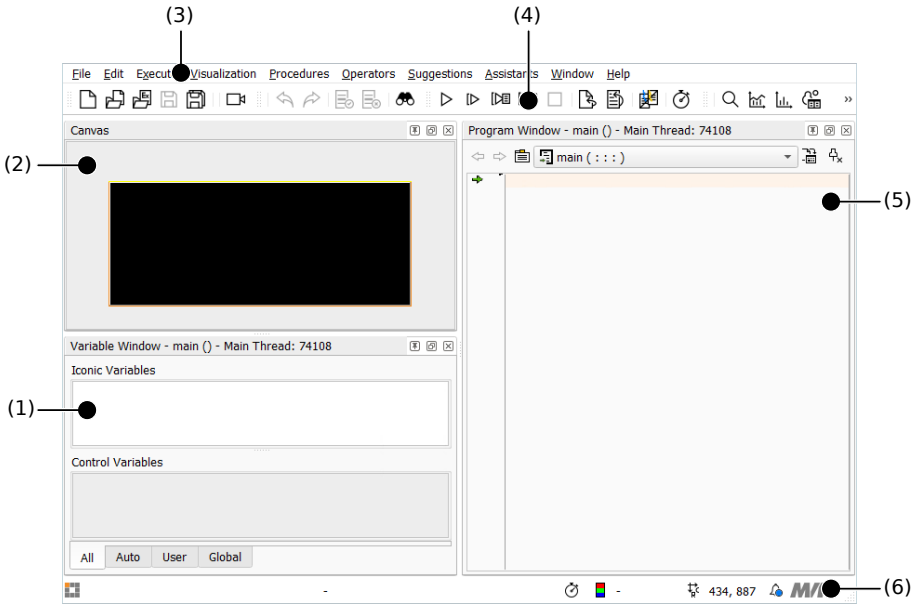


Figure 2.2: User interface: (1) variable window, (2) canvas window, (3) menu, (4) tool bar, (5) program window, (6) status bar.

When HDevelop is started for the first time it looks similar to figure 2.2. The main window offers a menu (3) and a tool bar (4) for quick access to frequently used functions. The status bar (6) at the bottom of the window displays messages and image properties. In addition, the following windows are available by default:

(1) Variable window

Program variables can be watched in this window. It displays all variables of the current procedure and their current values. Iconic variables are displayed as thumbnails, whereas control variables are displayed as text. The layout of this window can be switched between horizontal and vertical splitting by double-clicking the separator. See also Variable Window (page 142).

You can double-click iconic variables to display them in the active graphics window. Double-clicking control variables opens an inspection window with a nicely formatted list of the current values and statistical data. See also Inspecting Variables (page 148).

(2) Canvas window with graphics window

The canvas window is a container for graphics windows. At first startup of HDevelop, it contains one empty graphics window. Graphics windows are used to display iconic data like images, regions, and XLDs. See also Graphics Window (page 158). Each graphics window provides its own toolbar to adapt the image display to your needs, i.e., to choose between different zoom steps, for example. By default, the toolbar of graphics windows on the canvas is hidden. To display it, right-click in the canvas window and activate `Show Graphics Window Toolbar`.

The canvas window helps you to organize multiple graphics windows. You can arrange and resize the graphics windows on the canvas freely. For this, right-click in the canvas and activate `Show Graphics Window Frame`. Now you can drag the title bar of the graphics window to move it or drag the bottom right corner to resize the window. You can also detach the graphics windows from the canvas to display them floating. See also section 6.9 on page 161 and section 6.2.9.8 on page 109.

(5) Program window

This window displays the current program. It provides syntax highlighting with user-definable colors. The left column displays the program line numbers. The small black triangle is the insert cursor, which is where new program lines will be added. In the following, it is referred to as IC. The green arrow is the program counter which marks the next line to be executed. In the following, the program counter is referred to as PC. You can also add or remove breakpoints in the current program in this column. These will halt the program execution at user-defined places so that intermediate results may be examined. See also Program Window (page 113).

When adding new lines or modifying existing lines, advanced autocompletion (page 116) features speed up typing and help keeping the program consistent. Program lines can also be modified by double-clicking them and editing them in the operator window.

2.4 Organizing HDevelop's Workspace

HDevelop offers various possibilities to organize and personalize your workspace. The following paragraphs show which changes to HDevelop's layout are possible.

Docking Floating Windows

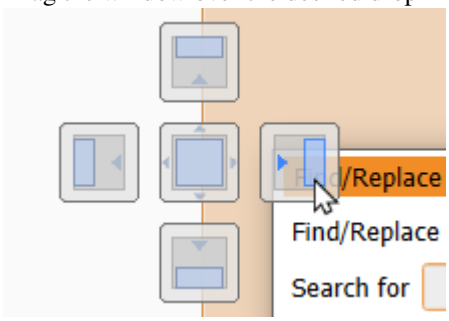
HDevelop supports docking of windows. Docked windows are bound to the window area of the main window (page 49). In the default layout, the program window, the graphics window and the variable window are already docked. You can always restore this default layout by clicking `Window` ▷ `Restore Default Layout`.

To dock a floating window, do the following:

1. Click and drag the title bar of the window.
   → Drop indicators appear, showing possible docking positions. (The middle one tabs [1] the window.)

[1] The window joins the already docked window as tab card. Note that graphics windows can only be tabbed to other graphics windows. Thus, it is not possible to tab a graphics window to any other kind of window and the other way round.

2. Drag the window over the desired drop indicator and release the mouse button.



Alternatively, click Dock window in the top right corner of the window. The window docks or tabs at its prior position.

To undock a window, do one of the following:

- Drag the window at the title bar/tab card and drop it apart from the drop indicators. The window will keep its current size.
- Click Undock window in the top right corner of the window or double-click on the title bar of the window. The window will resize to its prior size as floating window.

The position and size of docked windows is restored after closing and reopening HDevelop. There are only a few exceptions to this rule: HDevelop Assistants (Calibration, Measure...), Call Stack, Find Replace Dialog, Output Console Preferences, ROI Window, and Inspection Windows (ROI Inspect, Handle Inspect, Variable Inspect...). Note that the main HDevelop windows, namely the graphics window, the program window, and the variable window are always restored, even if they have been closed before.

Hiding Windows in the Side Bar

To keep your workspace clear, you can hide docked windows in the sidebar.

Auto-hide: Hide the docked window in one of the 4 side bars (top, left, right, bottom). Click the docked window's entry on the side bar to show it again. It will be displayed overlaid (on top of the current layout).

Disable auto-hide: Restore the docked window into the layout.

Changing the Window Size

Docked windows occupy a fixed position within the window area of the main window. To change their size, drag the border between two docked windows. Double-clicking on the border between two docked windows distributes them equally.

Floating windows can be positioned independently from other windows, even outside the main window. They can be resized by dragging their window border.

Organizing Graphics Windows on the Canvas

You can use HDevelop's canvas window to organize multiple graphics windows. To do this, add graphics windows to the canvas and arrange them as you wish. For more information, please refer to section 6.9 on page 161.

2.5 Running Example Programs

HALCON comes with a large number of HDevelop example programs from a variety of application areas. These range from simple programs that demonstrate a single aspect of HALCON or HDevelop to complete machine vision solutions. As an introduction to HDevelop, we recommend trying some of these programs to quickly get accustomed to the way HDevelop works.

The example program "Explore the Power of HALCON" demonstrates many different capabilities of HALCON in one program. It can be started from the start dialog, see figure 2.1 on page 16 (2). Running this program is highly recommended to get a good overview of the many application areas of HALCON.

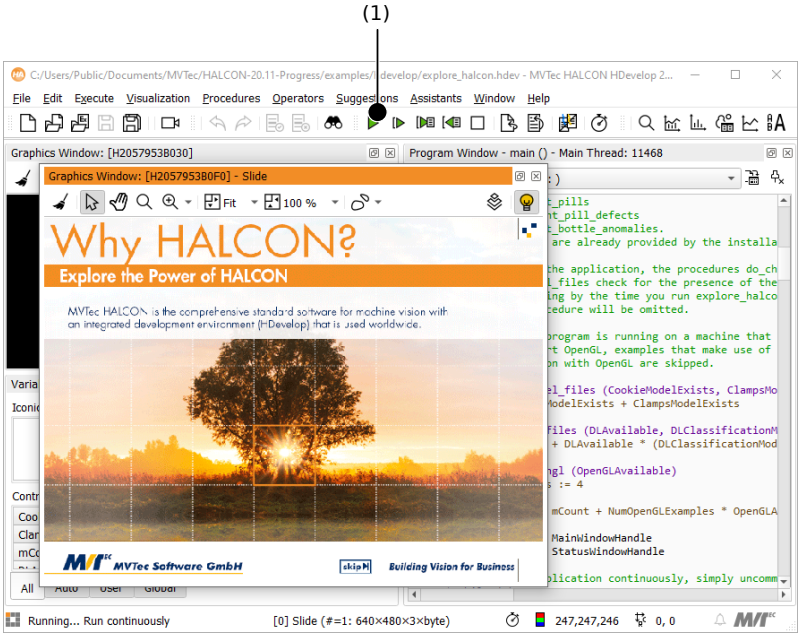


Figure 2.3: Explore the power of HALCON: Click (1) to run the program.

"Explore the Power of HALCON" starts up automatically when loaded from the start dialog. After loading it manually or loading one of the other example programs click ▷ or press F5 to start it, see figure 2.3 (1).

The example programs have been categorized by application area, industry, method, and operator usage. A special category "New in version" groups examples by their appearance in specific HALCON releases. Browsing these categories, you can quickly find example programs that cover image processing problems that you may wish to solve with HALCON. These programs may serve as a foundation for your own development projects.

Browse and Load Example Programs

- Click `File` ▷ `Browse HDevelop Example Programs...`

  This will open the example program browser (see figure 2.4). Similar to a file browser, it shows a tree of topics on the left and a list of example programs from the selected topics on the right. Subtopics can be toggled on or off by clicking the corresponding icon (1) or double-clicking the category name.

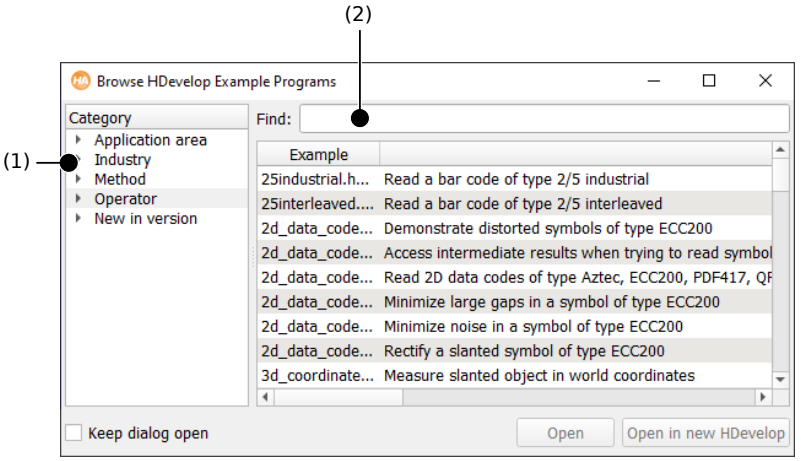


Figure 2.4: HDevelop example programs.

Browse the categories: Click on a topic to select it and display its example programs. You can select multiple topics at once by holding the Ctrl key while clicking on the categories.

Filter the example programs: To reduce the amount of listed example programs, enter a word or substring into the `Find` text field (2). Subsequently, only example programs matching this substring in the file name or short description will be displayed.

We pretend that you are looking for a measuring example from the semiconductor industry:

- Double-click on `Industry`.
- Click on the subtopic `Semiconductors`. The examples belonging to the semiconductor industry are listed on the right.
- Enter the word `measure` into the `Find` text field.

  Note how the list is updated as you type. Now, you have a short list of example programs to select from. You may need to resize the example browser to fully read the short descriptions of the listed programs.
- Select `measure_ic_leads.hdev` by clicking on it.
- Click `Open`. The selected example program is then loaded. Alternatively, you can load an example program by double-clicking on it. The example browser is closed unless `Keep dialog open` is checked.

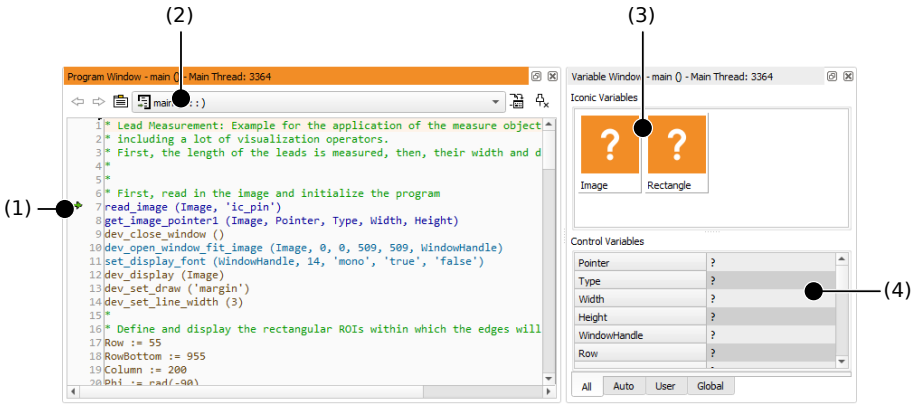


Figure 2.5: The variable and program window after loading the example program: (1) PC (program counter), (2) current procedure, (3) iconic variables, (4) control variables.

The program lines of the loaded example program are now displayed in the program window. The PC is set to the first executable line of the program (leading comments are ignored). The variable window is also updated: It lists the variables that are used in the main procedure, which is initially the current procedure. The variables are currently uninstantiated, i.e., their current value is undefined. This is indicated by the question mark (?). Both windows are displayed in figure 2.5.

Run Example Program

- Click `Execute` ▷ `Run` or click the corresponding button (1) from the tool bar (see figure 2.6).

  The program line next to the PC is executed, the PC is moved to the following line and so forth until the execution stops. There are four reasons for the program execution to stop: 1) the last program line has been executed, 2) a breakpoint has been reached, 3) the HDevelop instruction `stop` has been encountered as in this example, or 4) an error has occurred.

  During execution, the graphics window is used for visualization. Changes to the variables are reflected in the variable window. When the program execution stops, the status bar displays the number of executed lines and the processing time.

  To continue with the program execution, click `Execute` ▷ `Run` again until the end of the program is reached.
- Click `Reset Program Execution` (4) to reset the program to its initial state. (see figure 2.6).
- Using the button `Step Over` (2) you can execute the program line by line and inspect the immediate effect of each instruction.

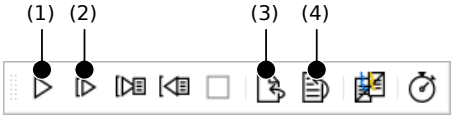


Figure 2.6: The basic execution buttons: (1) run continuously, (2) run step-by-step, (3) stop, (4) reset.

Command Line Switches

HDevelop supports several command line switches to modify its startup behavior. You can also add the path and file name of an HDevelop program on the command line to load it directly. This is identical to an invocation of HDevelop without any parameters and a subsequent loading of the program. The program name may contain environment variables in Windows syntax as in:

```
hdevelop %HALCONEXAMPLES%/hdevelop/explore_halcon.hdev
```

Or, you can convert HDevelop programs to other programming languages without opening the graphical user interface at all. A full list of the supported command line switches is available with the following command:

```
hdevelop --help
```

Under macOS command line switches are passed by calling HDevelop from a terminal window in the following way:

```
/Applications/hdevelop.app/Contents/MacOS/hdevelop OPTIONS
```

See appendix C.1 on page 351 for a listing of the available switches, and some example uses of the command line.

Chapter 3

Acquiring Images with HDevelop

Image acquisition is crucial for machine vision applications. It will usually be an early if not the first step in your programming projects. This chapter explores the different ways of image acquisition in HDevelop.

3.1 Reading Images From Files

Especially in the prototyping phase you often have a set of sample image files to work from. HDevelop (or rather the underlying HALCON library) supports a wealth of image formats that can be loaded directly (see `read_image` in the Reference Manual).

Drag-and-Drop

The easiest way to read an image is to simply drag it from a file browser to the HDevelop window and drop it there. When the file is dropped, HDevelop opens the dialog `Read Image` (see figure 3.1).

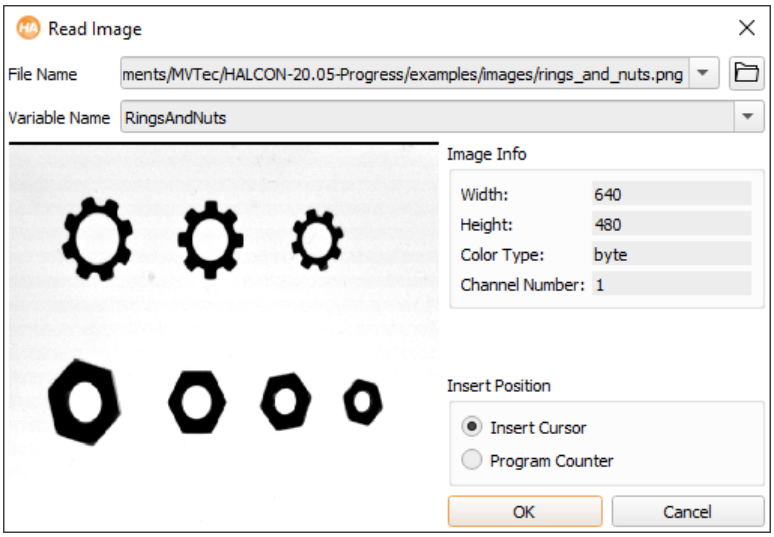


Figure 3.1: After dragging an image file onto the HDevelop window.

This dialog displays the full path of the image and automatically proposes a variable name derived from the file name. This name can be edited, or another iconic variable name from the current program may be selected from the drop-down list.

Furthermore, a preview of the image and basic image properties are displayed in the dialog (width, height, color type, and number of channels). If you picked the wrong image, you can select another one from the same directory by pressing the button next to the file name. This will open a file browser native to the operating system, i.e., on

Windows you may be able to switch to thumbnail view in this dialog. When another image is selected, the dialog is updated accordingly.

When you click the button OK, the instruction read_image is added to the current program. With the setting of Insert Position you determine where the instruction will be put: At the IC or the PC. If you changed your mind about reading the selected image at all, click Cancel.

Drag-and-Drop of Multiple Images

You can also drag multiple images or directories containing multiple images to HDevelop. HDevelop will then open an image acquisition assistant with the images preselected. See section 3.3 for further information.

Images from Selected Directories

Apart from dragging and dropping images, there is an equivalent method from within HDevelop: Select File ▷ Read Image... to get the dialog described above. Browse to and select the desired image from this dialog, and click OK to add the selected image to your program.

3.2 Viewing Images

When images are read as described above, they are automatically displayed in the active graphics window. This is the default behavior in HDevelop, but the automatic display of images can be suppressed if desired, e.g., to speed up computationally intensive programs.

Initially, new images are fitted into the graphics window at a 1:1 aspect ratio with unused bars on the left and right side. The aspect ratio will be preserved when resizing the graphics window. Using the tool bar of the graphics window (page 159) you can easily zoom the image or change the resize mode from the default Keep Aspect Ratio behavior. Furthermore, you can change the default resize behavior under Edit ▷ Preferences ▷ General Options. This default resize behavior applies to graphics windows that are opened via menu, i.e, Visialization ▷ Open Graphics Window... or Window ▷ Open Graphics Window.

An iconic view of the loaded image is also displayed in the variable window. When the image is cleared in the graphics window, the image can always be restored by double-clicking the corresponding icon in the variable window. See also section Displaying Iconic Variables (page 144).

3.3 Image Acquisition Assistant

The image acquisition assistant is a powerful tool to acquire images from files (including AVI files), directories or image acquisition devices supported by HALCON through image acquisition interfaces. To use this assistant, select Assistants ▷ Open New Image Acquisition. The window is displayed in figure 3.2. It features several tab cards that can be stepped through one after another. Ultimately, the assistant generates HDevelop code that can be inserted into the current program. Select the entry Help in the menu of the image acquisition assistant to open its online help.

The tab card Source determines the acquisition method and the image source. In the default setting images are acquired from files. This is described in the following section. Alternatively, images are acquired from an image acquisition device, e.g., a camera. This is described in section 3.3.2 on page 26.

3.3.1 Acquiring Images From Files or Directories

You can specify a selection of image files or a directory to load images from. Make sure the radio button Image File(s) is selected in the tab card Source. You can directly enter image names or the name of a directory into the text field. Multiple image names are separated by a semicolon. Usually, it is more convenient to use one of the following buttons:

Select File(s) ...

HDevelop opens a file selection dialog in the current working directory, displaying the image files supported by HALCON. Multiple image files can be selected by holding down the Ctrl key while clicking additional image files. Click Open to confirm the selection. The first selected image is displayed in the active graphics window.

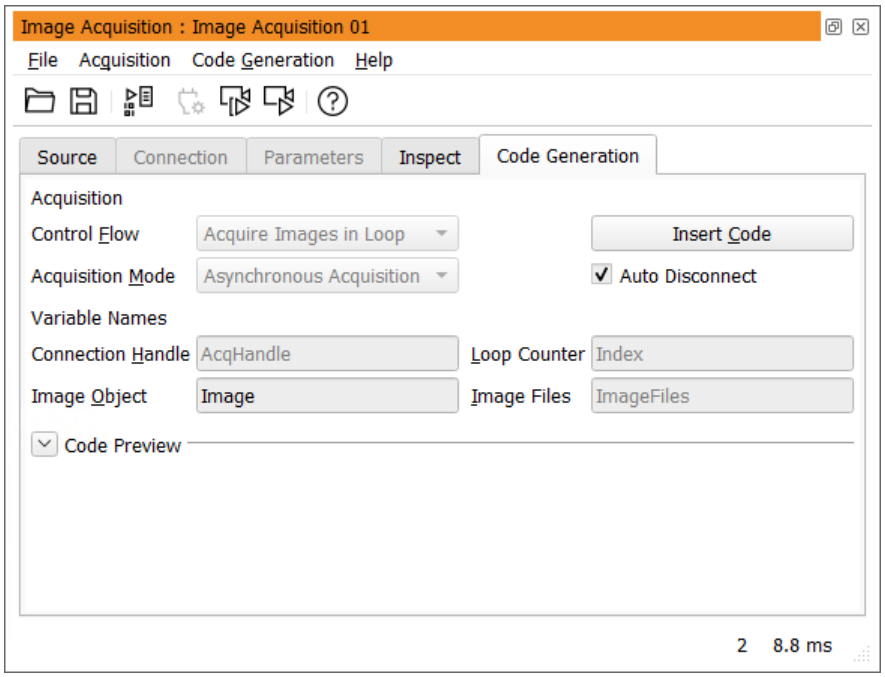


Figure 3.2: Image acquisition assistant.

`Select Directory ...`

HDevelop opens a directory browser. It is not possible to select multiple directories. Confirm your selection by clicking `Open` or `OK`. The first image from the selected directory is displayed in the active graphics window. If the check box `Recursive` is ticked, all subdirectories of the specified directory are scanned for images as well.

View Images

You can single-step through the selected images by clicking the `Snap` button (see figure 3.3). Each time you click the button, the next image is displayed in the active graphics window. You can also loop through the images by clicking the button `Live`. This is especially useful for animations. Both functions are also available from the menu `Calibration`.

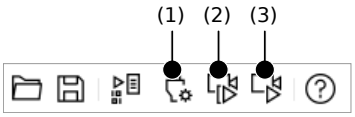


Figure 3.3: Browsing the selected images: (1) connect, (2) snap (single-step images), (3) live (continuous display).

Generate Code

Switch to the tab card `Code Generation`, and specify a variable name in the text field `Image Object`. You can later access the image in the program by this name. If multiple images or a directory were selected in the tab card `Source`, the image acquisition assistant will read the images in a loop. In this case the following additional variable names need to be specified:

`Loop Counter:` The name of the loop index variable. While looping over the images in the program, this variable will contain the object number of the current image.

`Image Files:` The name of the variable that will contain the names of the selected images.

Click `Code Preview` to inspect the code that would be generated from the currently specified parameters.

Click `Insert Code` to generate the code and insert it at the position of the IC in the current program.

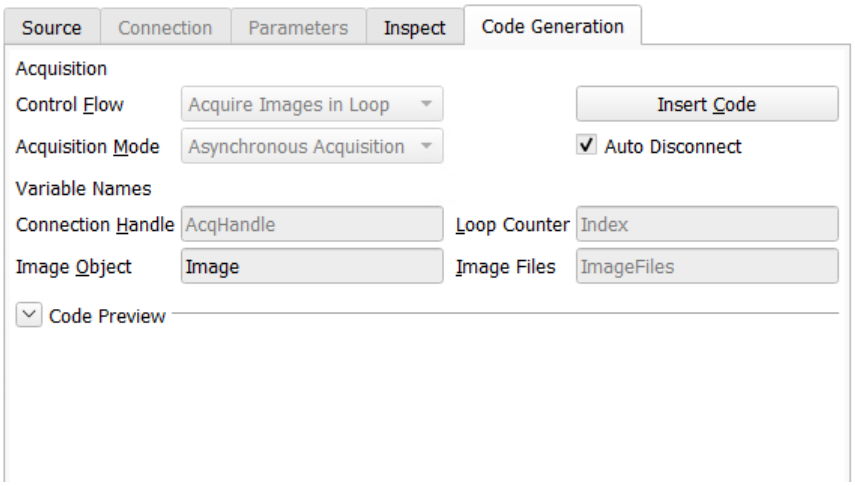


Figure 3.4: Specifying variable names for code generation.

The following piece of code is an example generated from three selected images. It is a self-contained HDevelop program that runs without alteration.

```
* Image Acquisition 01: Code generated by Image Acquisition 01
ImageFiles := []
ImageFiles[0] := 'W:/images/fin1.png'
ImageFiles[1] := 'W:/images/fin2.png'
ImageFiles[2] := 'W:/images/fin3.png'
for Index := 0 to |ImageFiles| - 1 by 1
    read_image (Image, ImageFiles[Index])
    * Image Acquisition 01: Do something
endfor
```

3.3.2 Acquiring Images Through Image Acquisition Interfaces

Select `Image Acquisition Interface` in the `Source` tab. The drop-down list below the radio button becomes active. Initially, it lists all image acquisition interfaces supported by HALCON. You can tidy this list by clicking the button `Auto-detect Interfaces` (1). HDevelop will then probe all the image acquisition interfaces and remove those that do not respond. Probing the interfaces might cause the system to hang due to erroneously installed drivers or hardware failures. If there are unsaved changes in the current program, HDevelop will display a warning dialog. You are advised to save the changes before you proceed. You can also ignore the warning and proceed, or abort the operation. After the interfaces have been probed, you can select the desired image acquisition interface from the list (2).

Selecting the entry `Help` from the menu of the image acquisition assistant will open the online help for the selected image acquisition interface.

Connect to the Device

Once an image acquisition interface is selected, its connection parameters are detected and updated in the tab card `Connection` (see figure 3.6). Here you can specify the device that is connected to the selected image acquisition interface. If, for example, the interface of a frame grabber board with multiple cameras has been selected as the source, the actual device can be selected here. The parameters of this tab card are described in general in the reference section of the operator `open_framegrabber`; please refer to the HTML page of the selected interface for detailed information (menu `Help`).

If the acquisition interface `File` is selected, two buttons become available to select an image file or an image directory, respectively. The `File` interface also supports AVI files, or sequence files (.seq). The latter are special to HALCON; they contain a list of image file names that will be loaded in succession.

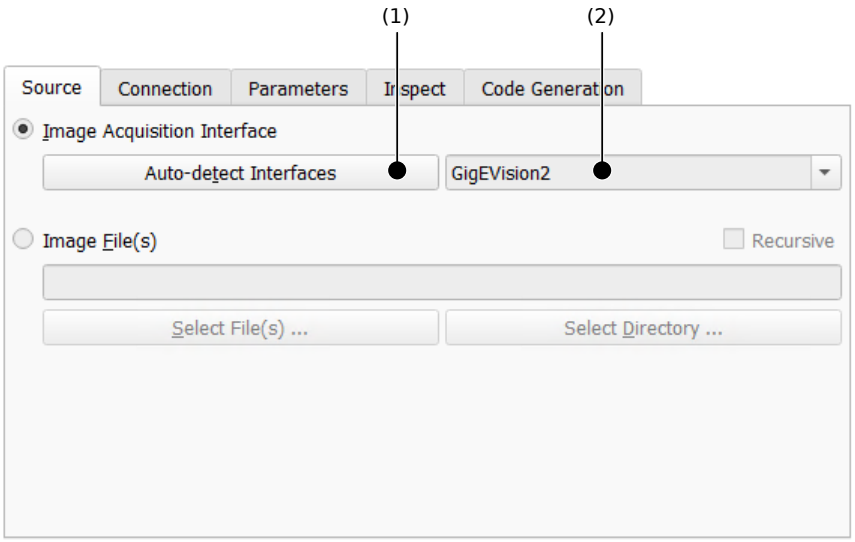


Figure 3.5: Source selection (example).

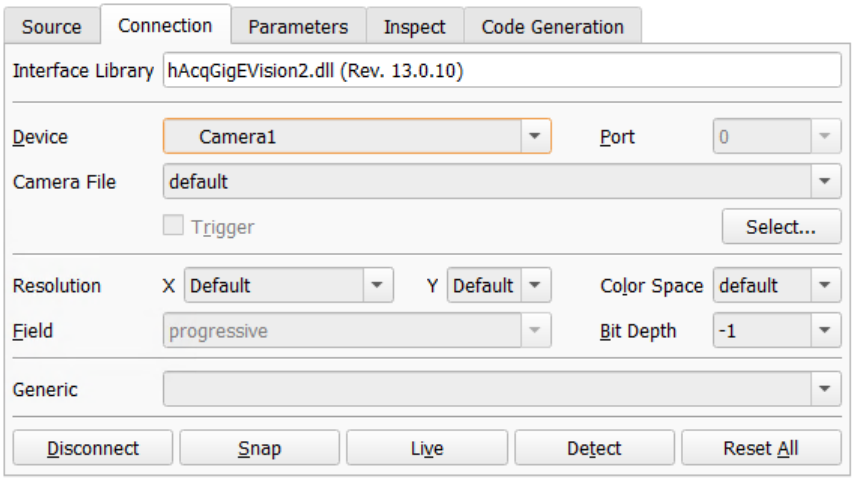


Figure 3.6: Connection parameters (example).

Specify the desired connection parameters and click `Connect` to establish or update the connection to the actual device. The connection status can also be toggled in the tool bar (see figure 3.3 on page 25).

You can grab single images with the button `Snap`, or switch to continuous grabbing mode using the button `Live`. Live mode can be stopped by clicking the same button again which is now labeled `Stop`.

Clicking the button `Detect` attempts to re-detect valid parameters for the currently selected image acquisition interface. Usually, this is done automatically, when the interface is selected from the list on the tab card `Source`.

The button `Reset All` sets all connection parameters back to their default values.

Set Device Parameters

The tab card `Parameters` contains a list of parameters specific to the selected device. It becomes available once the connection to the device has been activated. See figure 3.7 for an example parameter list. The range of parameters can be limited by category and visibility (1). Please refer to the HTML page of the selected interface for detailed information. You can click the help button of the assistant to get to the corresponding page automatically.

Depending on the parameter type, different selection methods are enabled. As an example, parameters with a defined range of values can be specified by dragging a slider (4) or entering the value parametrically. If a value is changed, a reset button to the right is activated (3). Some parameters provide a check box which attempts to set the parameter automatically if clicked.

If `Update Image` is checked (2), parameter changes are immediately reflected in the graphics window by acquiring a new image. The button `Refresh` updates the list of parameters, which is useful if parameters have side effects. You can reset all parameters to their default values at once by clicking `Reset All`.

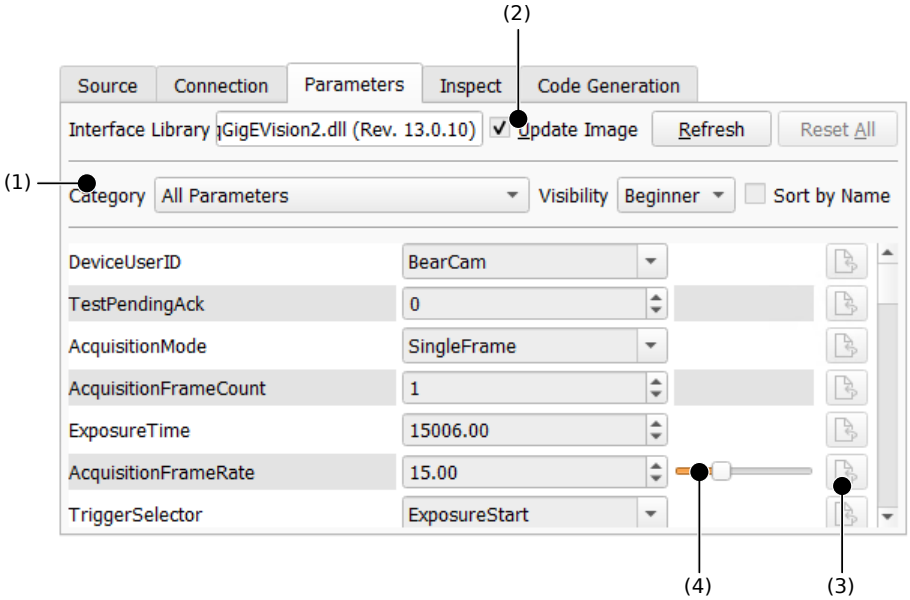


Figure 3.7: Device-specific parameters (example).

Generate Code

On the tab card `Code Generation` the settings made in the other tab cards are turned into executable code. The basic structure of the code and the corresponding variable names can be specified.

`Control Flow`

`Initialization Only`: Generate only code to initialize the image acquisition interface with the parameters specified in the other tab cards and to close it down properly. Additional code for image acquisition and processing can be added later.

`Acquire Single Image`: Also generate code to acquire an image.

`Acquire Images in Loop`: Also add a loop around the image acquisition code. Further image processing can be added inside this loop.

The image acquisition interface is addressed by a so-called handle. The variable name of this handle can be specified in the text field `Connection Handle`. The variable name of the acquired image(s) can be set in `Image Object`.

Click `Code Preview` to inspect the code. Click `Insert Code` to generate the code in the program window at the IC.

Here is a code example:

```
* Image Acquisition 01: Code generated by Image Acquisition 01
open_framegrabber ('GigEVision', 0, 0, 0, 0, 0, 0, 'default', -1, 'default',
-1, 'false', 'default', 'KickerCam', 0, -1, AcqHandle)
grab_image_start (AcqHandle, -1)
while (true)
    grab_image_async (Image, AcqHandle, -1)
    * Image Acquisition 01: Do something
endwhile
close_framegrabber (AcqHandle)
```

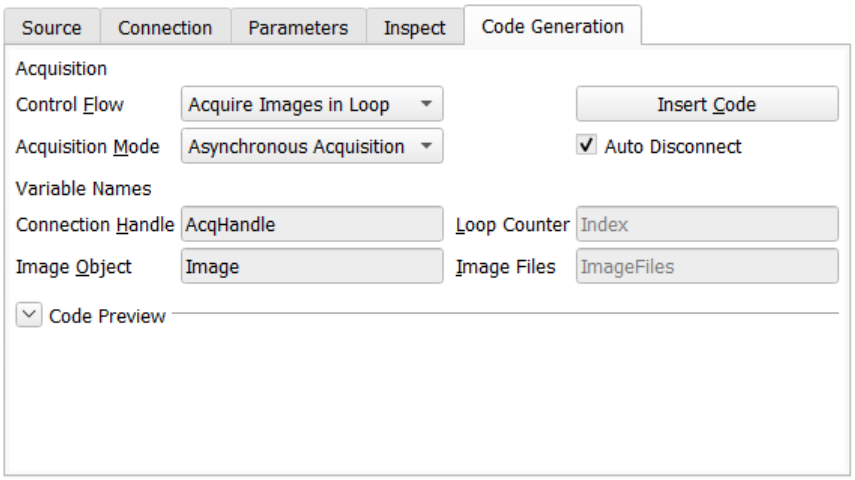


Figure 3.8: Code generation.

3.3.3 Modifying the Generated Code

After the generated code has been inserted into the program window, HDevelop internally keeps the code linked to the corresponding assistant. This link is kept until the assistant is quit using the menu entry `File` ▷ `Exit Assistant`. If you close the assistant using the menu entry `File` ▷ `Close Dialog` or using the close icon of the window, the assistant can be restored from the top of the menu `Assistants`.

You can change the settings inside the assistant and update the generated code accordingly. The code preview will show you exactly how the generated code lines will be updated. Furthermore, you can delete the generated code lines, or release them. When code lines are released, the internal links between the assistant and those lines is cut off. Afterwards, the same assistant can generate additional code at a different place in the current program.

Chapter 4

Programming HDevelop

This chapter explains how to use HDevelop to develop your own machine vision applications. It is meant to be actively followed in a running instance of HDevelop. In the following, it is assumed that the preferences of HDevelop are set to the default values. This is always the case after a fresh installation of HALCON. If you are uncertain about the current settings, you can always start HDevelop with the default settings by invoking it from the command line in the following way (see also chapter 2 on page 15):

```
hdevelop -reset_preferences
```

This chapter deals with a simple example. Given is the image displayed in figure 4.1. The objective is to count the clips and determine their orientation.

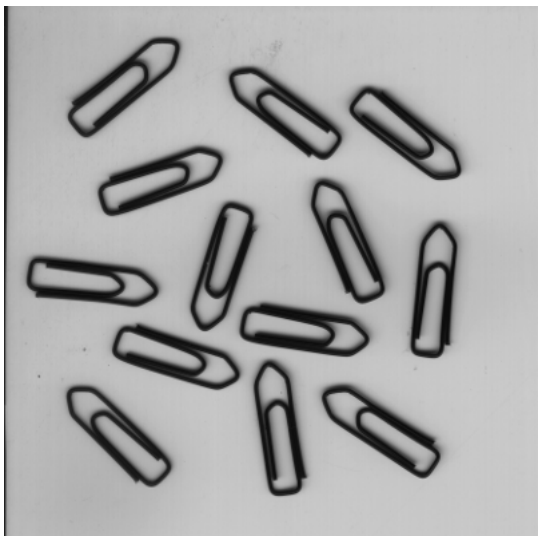


Figure 4.1: Paper clips.

4.1 Start a New Program

Start HDevelop or, if it is still running, click `File` ▷ `New Program` to start a new program. HDevelop will notify you if there are unsaved changes in the current program. If it does, click `Discard` to throw away the changes and start anew.

The first thing to do is read the image and store it in an iconic variable. From the last chapter we know that we can simply drag an image to the HDevelop window. We also know that this inserts the operator `read_image` into the program. Therefore, we can just as well insert the operator directly.

4.2 Enter an Operator

Click into the text box of the operator window, type `read_image` and press Return. You can also type any partial operator name and press Return. HDevelop will then open a list of operators matching that partial name. This way, you can easily select operators without having to type or even know the exact name. Selection is done with the mouse or using the arrow keys to highlight the desired operator and pressing Return. If you selected the wrong operator by accident, you can reopen the list by clicking on the drop-down arrow next to the operator name. When entering a partial name, operators commencing with that name appear at the top of the list.

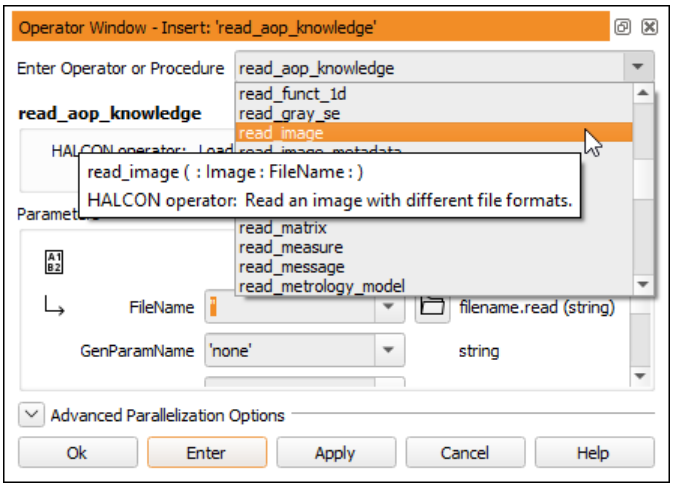


Figure 4.2: Matching operators after typing `read_` and pressing Return.

4.3 Specify Parameters

After selecting an operator, its parameters are displayed in the operator window. They are grouped by iconic and control parameters. The icons next to the parameter names denote the parameter type: Input (2) vs. output (1) (see figure 4.3). The semantic type is displayed to the right of the parameters. Parameters are specified in the text fields. The first parameter gets the input focus.

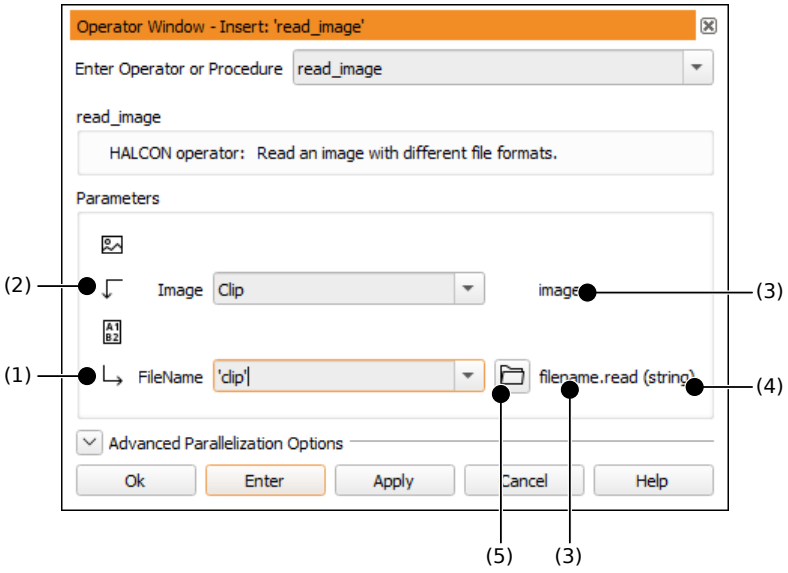


Figure 4.3: Specifying parameters: (1) iconic output parameter, (2) control input parameter, (3) semantic type, (4) data type, (5) file selection dialog.

Enter Clip into the text field Image. The image will be stored in this variable. Next, enter 'clip' into the text field FileName. You can press Tab to go to the next input field. Pressing Shift+Tab takes you back to the previous field. This way you can enter all parameters without using the mouse.

Click OK to add the operator to the current program and execute it. This will do the following:

- An operator call is added as the first line of the current program.
- The IC is advanced, so that additional lines will be added after the inserted line.
- The character * is added to the window title to indicate unsaved changes in the current program. The current procedure (main) is also marked with * in the program window.
- The program line is executed and the PC is advanced. To be more precise: All the lines from the PC to the IC are executed which makes a difference when adding program lines in larger programs.
- The image is displayed in the graphics window.
- The status bar is updated, i.e., the execution time of the operator read_image is displayed and the format of the loaded image is reported.
- The output variable Clip is created and displayed in the variable window.
- The operator window is cleared and ready for the insertion of the next operator.

4.4 Getting Help

You may be wondering where the clip image was loaded from since we did not specify any path or even a file extension. This is a detail that is related to the way the HALCON operator read_image works. HDevelop does not know anything about it. It just executes the operator with the parameters you supply. Accessing the operator documentation from within HDevelop is very easy.

Double-click the first program line in the program window. The operator is displayed in the operator window for editing. Now click Help to open the HDevelop online help window. It will automatically jump to the documentation of the displayed operator. The reference manual is completely cross-linked. The navigation at the left part of the window provides quick access to the documentation. The tab card Contents presents the hierarchical structure of the documentation. The tab card Operators lists all operators for direct access. Enter any desired substring into Find to quickly find an operator.

In the remainder of this chapter, try to use the online help as much as possible to get information about the used operators. The online help window is described in section 6.11 on page 165.

4.5 Add Additional Program Lines

Select the clips by thresholding

Now, we want to separate the clips from the background, i.e., select them. They clearly stand out from the background, thus a selection based on the gray value is appropriate. This operation is known as thresholding.

Enter threshold into the operator window. This is both the full name of an operator and part of other operator names. Thus, you get a list of matching operators with threshold pre-selected when you press Return. Press Return once more to confirm the selected operator and show its parameters.

In figure 4.4 you can see that the input parameter Image is set to Clip automatically. For input variables with no default value, reasonable suggestions are inferred automatically by collecting previous output variables of the same type. Therefore, the name of the most recent matching output parameter will be suggested (most recent being the closest predecessor of the current program line). In this example, only Clip is available.

Set MinGray and MaxGray to 0 and 30, respectively. This will select the dark pixels in the image.

Click Apply. This button executes the operator without adding it to the program. Additionally, it keeps the current parameters open for editing. This way, you can easily try different settings and immediately see the result. The selected pixels (the so-called *region*) are stored in the output variable Region, which is displayed in the variable window. The region is an image mask: White pixels are selected while black pixels are not.

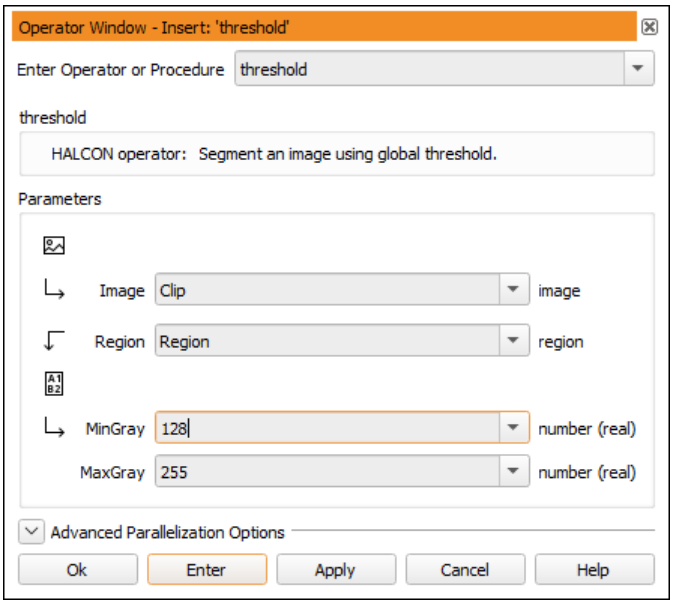


Figure 4.4: Parameter suggestions.

The region is also displayed as an overlay in the graphics window. The selected pixels are displayed in red (unless you changed the default settings).

The selected threshold values are not perfect, but we will correct this later. For now, click `Enter` to add the operator to the program window. Contrary to clicking `OK`, this does not execute the operator. Note that the variable `Region` keeps its value but is no longer displayed in the graphics window. Also, the PC is not advanced, indicating that the second line of the program is yet to be executed.

Adding program lines with `Enter` is especially useful if some of the input parameters use variable names that will be added to the program at a later time.

Successor

Click on the just inserted program line to select it. You can let HDevelop suggest operators based on the selected line. Open the menu `Suggestions` ▷ `Successors`. This menu is filled dynamically to show typical successors of the currently selected operator. We want to split the selected pixels into contiguous regions. Move the mouse pointer over the menu entries. The status bar displays a short description of the highlighted operator. Looking through the menu entries, the operator `connection` looks promising, so we click on it. Any operator selected through this menu is transferred to the operator window.

Again, the variable names suggested by HDevelop look reasonable, so click `OK`. This time, two program lines are executed: The `threshold` operation and the `connection` operation. As noted above: Clicking `OK` executes from the PC to the IC.

In the graphics window, the contiguous regions calculated by the operator `connection` are now displayed in alternating colors.

4.6 Understanding the Image Display

After having executed the three lines of our program, the graphics window actually displays three layers of iconic variables: the image `Clip`, the region `Region`, and the tuple of regions `ConnectedRegions` (from bottom to top). Place the mouse pointer over the icons in the variable window to obtain basic information about the variables.

The display properties of images and the topmost region can be adjusted from the context menu of the graphics window. For images, the look-up table (henceforth called LUT) and the display mode (referred to as “paint”) can be set. The LUT specifies gray value mappings. Experiment with different settings: Right-click in the graphics window and select some values from the menus `Lut` and `Paint`. Make sure, the menu entry `Apply Changes Immediately` is checked. Notice how the display of the image changes while the regions remain unchanged.

The menu entries Colored, Color, Draw, Line Width, and Shape change the display properties of the topmost region. Set Draw to 'margin', Color to 'cyan', and Shape to 'ellipse'. The display of ConnectedRegions (which is the topmost layer) changes accordingly. The region Region is still displayed in filled red.

A more convenient way to set many display properties at once is available through the menu entry Set Parameters. It opens the settings window displayed in figure 4.5.

After trying some settings, click the button Reset to restore the default visualization settings.

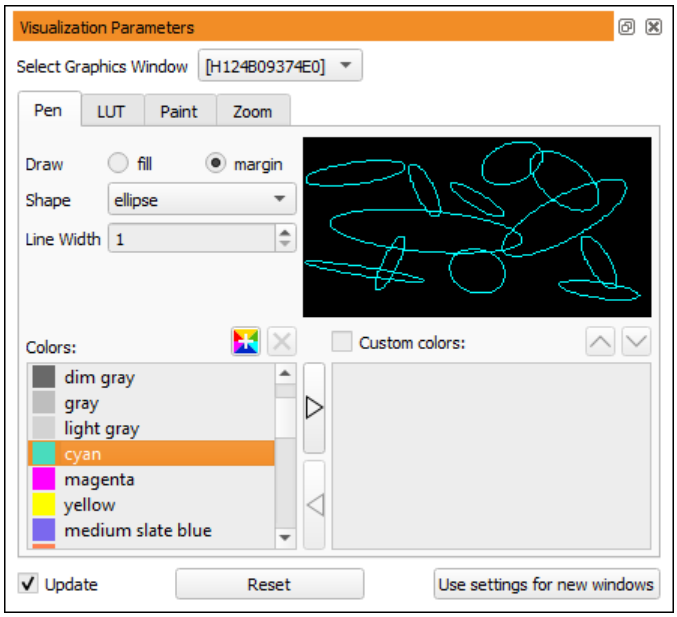


Figure 4.5: Changing the display parameters.

You cannot change the display properties of regions (or XLDs) other than the topmost. What you can do is rebuild the image stack in the graphics window manually by double-clicking iconic variables in the variable window and changing the properties each time another layer is added. The stack is cleared whenever an image is added that uses the full domain. To clear the stack (and thus the graphics window) manually, click (see figure 6.77 on page 158).

4.7 Inspecting Variables

When you move the mouse cursor over the variable ConnectedRegions you see that it contains 98 regions.

Right-click on the icon ConnectedRegions and select Clear / Display to display only the connected regions in the graphics window. Right-click again and select Display Content ▷ Select.... This menu entry opens a variable inspection window which lists the contents of the variable ConnectedRegions. The selected region of this inspection window is displayed in the graphics window using the current visualization settings. Set Draw to 'margin' and Shape to 'ellipse' and select some regions from the list. An example is illustrated in figure 4.6.

For now, close the variable inspection window. The large number of regions is due to the coarse setting of the bounds of the threshold operator. In the following we will use one of HDevelop's visual tools to find more appropriate settings interactively.

4.8 Improving the Threshold Using the Gray Histogram

Click Visualization/Tools ▷ Gray Histogram to open a tool for the inspection of gray value histograms. One of its uses is to determine threshold bounds visually. Because the graphics window currently displays only regions, the gray histogram is initially empty. Double-click the Clip icon in the variable window to re-display the original image and watch its gray histogram appear.

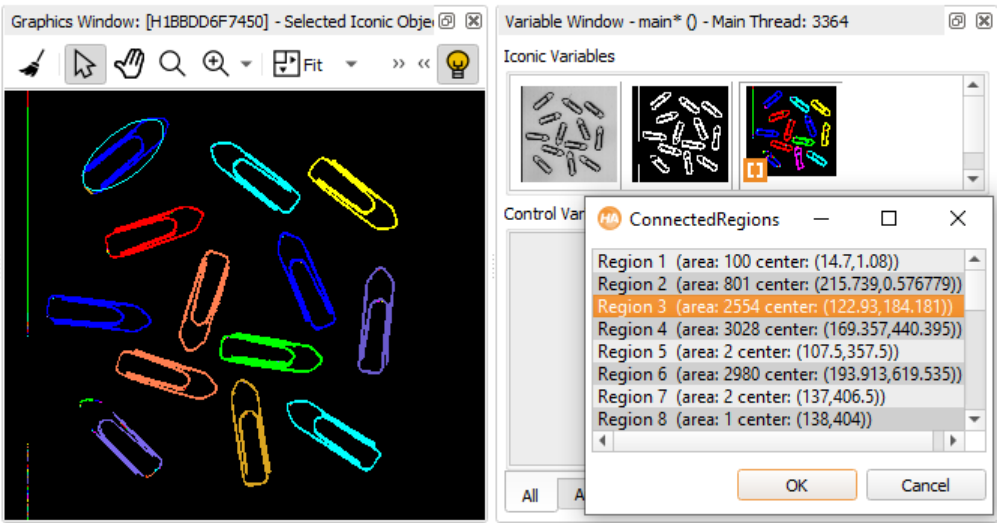


Figure 4.6: Interactive inspection of an iconic variable containing regions.

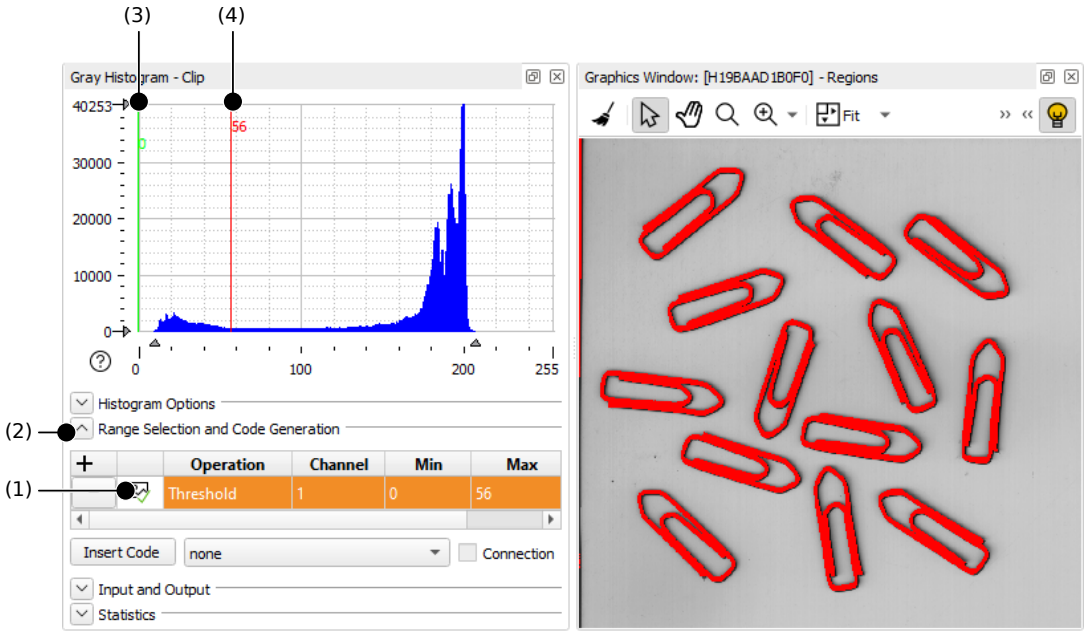


Figure 4.7: Determining threshold bounds interactively using the gray histogram.

Make sure `Range Selection and Code Generation` is visible in the histogram window (2), see figure 4.7. Select `Threshold` in the column `Operation` of the gray histogram window, and click the icon next to `Threshold` (1) to visualize the operation. Now, you can try different threshold bounds by altering the values in `Min` and `Max` or by dragging the green (3) and the red line (4) in the histogram area. Any changes to these values are immediately visualized in the active graphics window. The values 0 and 56 seem suitable for the lower and upper bounds, respectively.

4.9 Edit Lines

To edit a line in the operator window, double-click it in the program window. If you make changes to the parameters and click `OK` or `Replace`, the original line in the program is updated. You can also edit the program directly in the program window (see section 6.4.2 on page 115).

Double-click the second line of the program to adjust the threshold operation. Replace the value 30 with 56 and

click `Replace`. The program line is updated in the program window.

4.10 Re-Execute the Program

The last editing step was just a tiny modification of the program. Often, after editing many lines in your program with perhaps many changes to the variables you want to reset your program to its initial state and run it again to see the changes.

Click `Execute` ▷ `Reset Program Execution` to reset the program. Now, you can select `Execute` ▷ `Run` to run the complete program, or click `Execute` ▷ `Step Over` repeatedly to execute the program line by line.

4.11 Save the Program

Perhaps now is a good time to save your program. Select `File` ▷ `Save` and specify a target directory and a file name for your program.

4.12 Selecting Regions Based on Features

Inspecting the variable `ConnectedRegions` after the changed threshold operation yields a much better result. Still, a contiguous area at the left edge of the image is returned (4), see figure 4.8. To obtain only the regions that coincide with the clips, we need to further reduce the found regions based on a common criterion. Analogous to the gray histogram tool, which helps to select regions based on common gray values, HDevelop provides a feature histogram tool, which helps to select regions based on common properties or features.

Click `Visualization/Tools` ▷ `Feature Histogram` to open the tool. Make sure `Feature Selection and Code Generation` is visible in the histogram window (3). The column `Feature` allows you to select the feature that the region selection will be based on. The default feature is “area”, which is adequate in this case: The actual clips are all the same size, thus the area of the regions is a common feature. In the feature histogram the horizontal axis corresponds to the values of the selected feature. The vertical axis corresponds to the frequency of certain feature values.

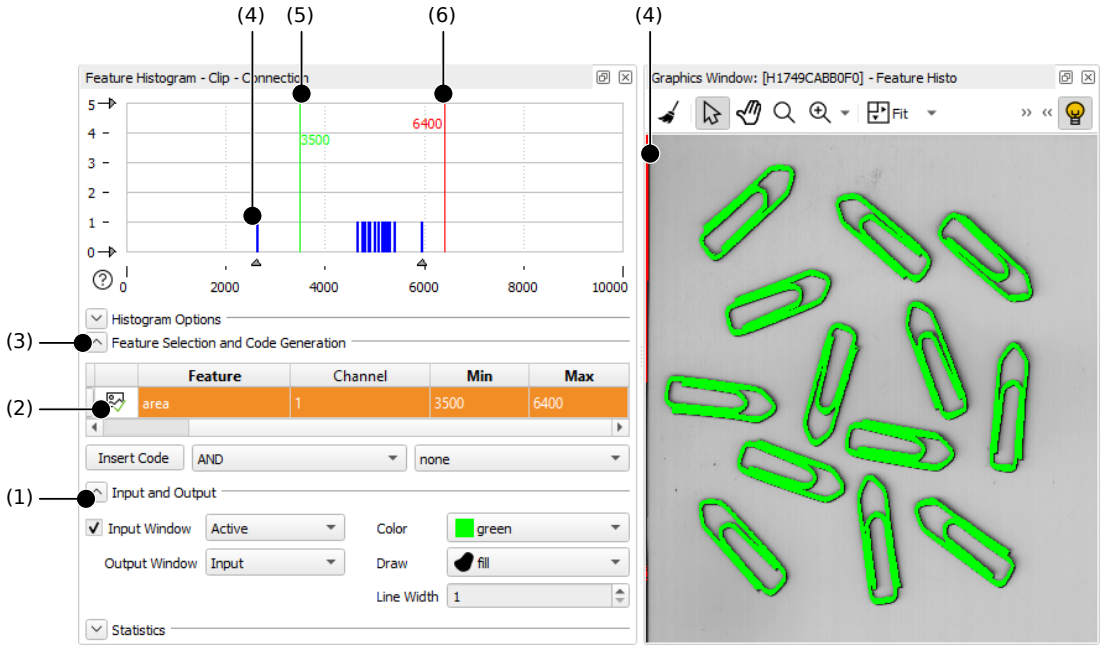


Figure 4.8: Selecting regions with a similar area in the feature histogram.

Similar to the gray histogram window, you can visualize the selected regions, i.e., the regions whose area falls between the values `Min` and `Max`, which are represented by the green and red vertical lines, respectively. Click the icon next to the selected feature (`area`) (2) to enable the visualization.

Specify the parameters in the `Input and Output` section of the feature histogram window (1). Drag the green (5) and red line (6) to see how this affects the selected regions. In the histogram we can see that in order to cover all the clips, we can safely select regions whose area is between, say, 4100 and the maximum value in the histogram. When you are satisfied with the selection, click the button `Insert Code`. The following line (with similar numeric values) will be added to your program at the position of the IC:

```
select_shape (ConnectedRegions, SelectedRegions, 'area', 'and', 4100, 5964)
```

Run the program, and inspect the output variable `SelectedRegions`. The regions corresponding to the clips are now determined correctly. To obtain the orientation and the center of gravity of the clips, add the following operator calls to the program:

```
orientation_region (SelectedRegions, Phi)
area_center (SelectedRegions, Area, Row, Column)
```

The operator `orientation_region` returns a tuple of values: For each region in `SelectedRegions` a corresponding orientation value in `Phi` is returned. The operator `area_center` in the same way returns the area, row and column of each input region as tuples. Again, run the program and inspect the calculated control variables. You can inspect multiple control variables in one inspection window. This is especially useful if the control variables all relate to each other as in this example. In the variable window select all control variables (hold down the Ctrl key), and right-click `Inspect` (see figure 4.9).

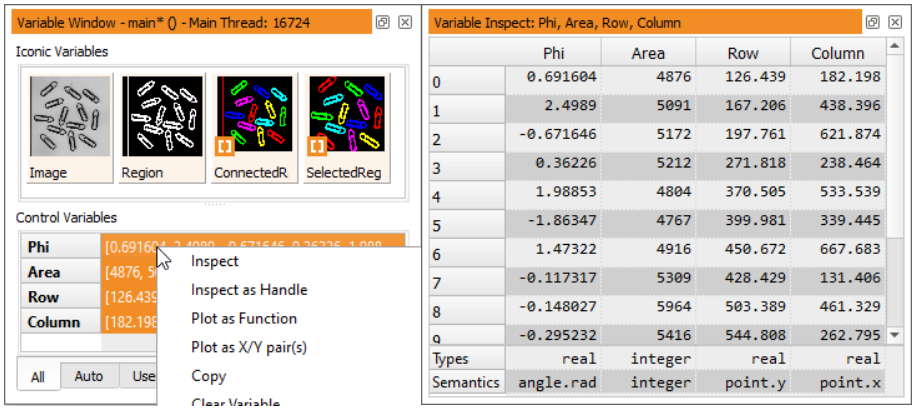


Figure 4.9: Inspecting control variables.

4.13 Open Graphics Window

Up until now, the visualization of iconic results relied on the fact that a graphics window is always opened by default when HDevelop starts. If you need control over the size and position of graphics windows you can explicitly open a distinct number of floating graphics windows.

Add the following lines before the first line of your program:

```
dev_close_window ()
dev_open_window (0, 0, 512, 512, 'black', WindowHandle)
```

The first line closes a floating graphics window potentially left over from a previous run to ensure a consistent state at the start of the program. The second line opens a single graphics window that can be referenced by the output variable `WindowHandle`. The window handle is a magic value that is also displayed in the title bar of the corresponding graphics window.

Graphics windows that are opened via operator `dev_open_window` will be opened as floating windows, displaying images in `Full Stretch` mode for maximum backwards compatibility.

4.14 Looping Over the Results

Being an integrated development environment, HDevelop provides features found in other programming languages as well: Variable assignment, expressions, and control flow. Variable assignment and control flow are implemented in terms of specific HDevelop operators. These operators can be selected from the menu `Operators` ▷ `Control`. Expressions are implemented in terms of a specific HDevelop language which can be used in input control parameters of operator calls.

To iterate over the elements in `Phi`, we use a `for` loop which counts from zero (the index of the first element of a tuple) to the number of elements minus one. The `for` loop is entered just like a common HALCON operator: Enter `for` into the operator window and specify the parameters as in figure 4.10. Note that the closing `endfor` is entered automatically if the corresponding check box is ticked. Also note that the IC is placed between the added lines so that the body of the loop can be entered.

The notation `|Phi| - 1` is part of the HDevelop language. This operation calculates the number of elements in `Phi` minus one. When inserted in the program window, the operator `for` is displayed in a different format to make it more readable.

Add the following instruction to the program. It is automatically indented in the program window to highlight the nesting inside the `for` loop. The backslash at the end of the first line allows the program line to continue at the next line in the program window for readability. You can either enter the instruction as displayed or in a single long line without the backslash.

```
dev_disp_text (deg(Phi[Index]) + ' degrees', 'image', Row[Index], \
               Column[Index], 'black', [], [])
```

The operator `dev_disp_text` provides a convenient way to output text at specific positions in the activated graphics window. Press F1 to get more information about the meaning of the parameters. The notation `Phi[Index]` is another operation of the HDevelop language. It provides access to a single value of a tuple. The function `deg` is part of the HDevelop language. It converts its argument from radians to degrees. In this example the operation `+` performs a string concatenation because the argument `' degrees'` is a `string` value. Before the two operands of `+` are concatenated, an automatic type conversion (`double` to `string`) of the numeric argument takes place. The details of the HDevelop language are explained in chapter 8 on page 281.

Please note that the loop around `dev_disp_text` was chosen to illustrate how to access single elements of a tuple. In this specific example it is not really required because `dev_disp_text` is smart enough to directly operate on tuples. Instead of the loop, you could simply use the following call:

```
dev_disp_text (deg(Phi) + ' degrees', 'image', Row, Column, 'black', [], [])
```

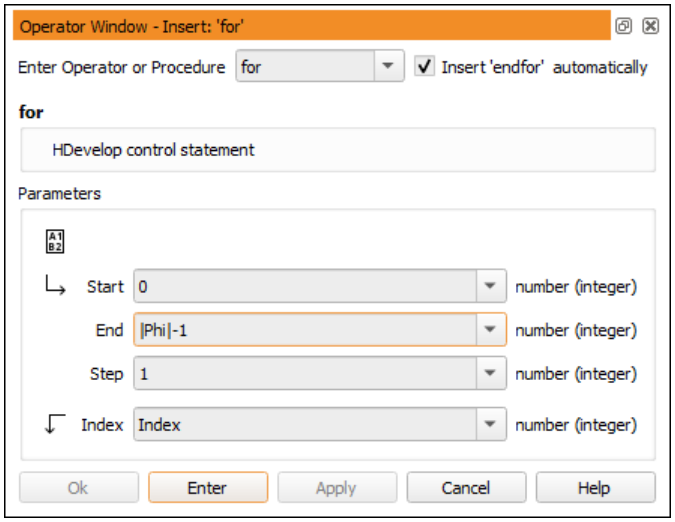


Figure 4.10: Entering a loop in HDevelop.

4.15 Summary

This is basically the way to create programs in HDevelop. Select an operator, specify its parameters, try different settings using the button `Apply`, add a new program line using `Enter` or `OK`, and edit it later by double-clicking it in the program window. Use the interactive tools provided by HDevelop to assist you, e.g., to find appropriate values for the operators.

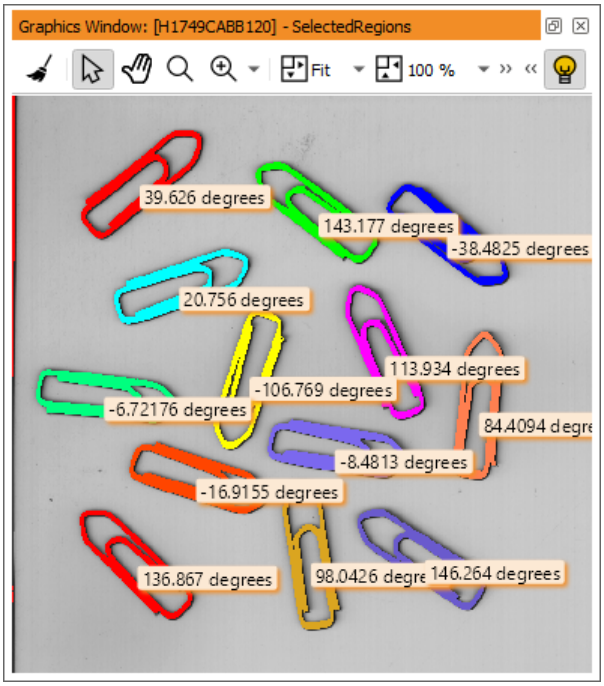


Figure 4.11: Final result of the example program.

Chapter 5

HDevelop Procedures

HDevelop offers a mechanism for the creation and execution of procedures. Procedures are meant to increase the readability and modularity of HDevelop programs by encapsulating functionality of multiple operator calls in one or more procedure calls. It also makes it easier to reuse program code in other HDevelop programs by storing repeatedly used functionality in external procedures.

An HDevelop procedure consists of an interface and a program body. Procedure interfaces resemble the interfaces of HALCON operators, i.e., they contain parameter lists for iconic and control input and output parameters. The procedure body contains a list of operator and procedure calls. Furthermore, HDevelop provides extensive support to supplement procedures with structured documentation. The documentation is automatically integrated into the online help system.

Every HDevelop program is made up of one or more procedures. It always contains the main procedure, which has a special status inside the program, because it is always the top-most procedure in the calling hierarchy and cannot be deleted from the program.

HDevelop offers all necessary mechanisms for creating, loading, deleting, copying, modifying, saving, and exporting procedures. Once a procedure is created, it can basically be used like an operator: Calls to the procedure can be added to any program body and be executed with the appropriate calling parameters. Generally, the concept of using procedures inside HDevelop is an extension to the concept of calling HALCON operators since procedure and operator interfaces have the same parameter categories, and the same rules apply for passing calling parameters.

5.1 Procedure Types

HDevelop supports different types of procedures. The type is specified when a procedure is created, and determines the location of the procedure in the file system.

- Local procedures

  Local procedures are stored inside the HDevelop program. Each program contains at least the “main” procedure. Local procedures cannot be called from other programs or external procedures.

- External procedures

  External procedures are stored as separate files, and can be shared between different HDevelop programs. The modification of an external procedure immediately affects all HDevelop programs using it. There are two types of external procedures:

  - Procedure files

    Each procedure file contains a single external procedure. The file name determines the name of the procedure. Thus, if the file name of an external procedure is changed, programs and other procedures using it will have to be adapted.

  - Libraries

    Libraries contain a collection of (typically related) procedures in a single file. They share the same properties as a collection of external procedures in a single directory. The idea of creating a library is to keep related procedures as a unit.

If the type of an existing procedure is changed, the procedure resolution might be affected (see section 5.5 on page 43).

5.2 File Types

HDevelop programs, procedures, and libraries are stored in files with different formats and extensions.

5.2.1 HDevelop Programs

.hdev (HALCON 10 or higher) This is the default file format for HDevelop programs, i.e., the "main" procedure and all local procedures. It stores programs in XML format and is suitable for revision control software.

Please note that programs in this format are not supported in HDevelop versions up to and including HALCON 9.

.dev This is the default file format for HDevelop programs in HDevelop versions up to and including HALCON 9. It is not suitable for managing programs using revision control software. This format is required if you want to be able to load HDevelop programs in older versions of HDevelop. This format is now marked as legacy and its use is no longer recommended.

5.2.2 Procedure Files

.hdvp (HALCON 10 or higher) This is the default file format for external procedure files. It stores external procedures in XML format and is suitable for revision control software. Procedures with the extension *.hdvp* always override procedures with the same name but the extension *.dvp* in the same directory.

.dvp This is the default file format for external procedure files in HDevelop versions up to and including HALCON 9. It is not suitable for managing external procedures using revision control software. This format is required if you want to be able to use external procedures in older versions of HDevelop. This format is now marked as legacy and its use is no longer recommended.

The file name (without the extension) determines the name of the contained external procedure (see also section 5.1 on page 41).

5.2.3 Libraries

.hdpl (HALCON 11 or higher) This is the file format for procedure libraries. It stores libraries in XML format and is suitable for revision control software. Libraries were introduced in HALCON 11 and cannot be used in older versions of HDevelop. Library procedures have to be converted to external procedure files to be accessible in older versions of HDevelop.

5.3 Procedure Scope

The scope of a procedure defines its visibility to other procedures. If a procedure is visible, it can be called and thus executed. The scope can be either private or public.

- Private scope

  The procedure is only visible from procedures in the same directory or the same file (program or library). The scope of local procedures is always private to the current program, i.e., external procedures can never see them.

- Public scope

  The procedure is visible to all other procedures.

5.4 Procedure Locations

HDevelop looks for procedures in a set of locations in the order specified in this section. Locations can be either directories or library files.

1. Import locations

   Import locations are specified in the program code using the `import` statement. Program lines following the `import` statement may call procedures from that location.

   For example, if you want to call the procedure *config.hdvp* from the directory *C:/Users/Public/procedures/common*, here is how this can be achieved:

   ```
   ...
   import C:/Users/Public/procedures/common
   config()
   ...
   ```

   An import location is valid from its `import` call to the end of the procedure.

2. Session locations

   HDevelop can be started from the command line with a set of locations that are searched for procedures in the current session only.

   Calling

   ```
   hdevelop -external_proc_path:<path name(s)>
   ```

   adds the locations given in *path name(s)* to the list of searched procedure locations. Multiple locations may be specified by using the system-dependent separator, i.e., ";" on Windows systems and ":" on Linux/macOS systems.

   Subdirectories of the specified locations are searched recursively.

3. Static user-defined locations

   Arbitrary locations can be added or removed in the preferences of HDevelop. Just like the standard procedure path below, each user-defined directory can be enabled or disabled independently. The user-defined directories and their settings are persistent, i.e., they survive a restart of HDevelop. It is good practice to put commonly used procedures into these locations.

   Subdirectories of the specified locations are searched recursively.

4. Standard procedure path

   By default, HDevelop is set up to look for procedures in a predefined directory which contains several useful procedures. The standard procedure path is `%HALCONROOT%\procedures` under Windows and `$HALCONROOT/procedures` under Linux/macOS, Because many example programs shipped with HALCON rely on these procedures, the standard procedure path cannot be removed, but it can be disabled to make the corresponding procedures unavailable.

   Subdirectories of the specified location are searched recursively.

5.5 Procedure Resolution

A procedure can only be called if it can be resolved from the point of insertion of the procedure call. Because duplicate procedure names are allowed in HDevelop, understanding the order of resolution is very important. The resolution order determines which procedure is used if several procedures of the same name exist. For example, the imported procedure *xyz.hdvp* takes priority over the local procedure *xyz.hdvp*.

Procedures are resolved based on their name only and not on the matching of the signature. If the resolved procedure has a different signature (i.e., different number and/or types of parameters), the call will be invalid.

A procedure call looks for a procedure of the given name in the following order:

1. Imported procedures

   First, HDevelop looks for imported procedures (section 5.4). Note that for several import statements that contain procedures of the same name, the last import call has priority.

2. Context of the calling procedure

   If no imported procedure is found the context of the calling procedure is significant for the procedure resolution. The context depends on the type of the calling procedure:

   - If the calling procedure is a local procedure, it looks for local procedures.
   - If the calling procedure is a procedure file, it looks for procedure files in the same directory.
   - If the calling procedure is a library procedure, it looks for procedures in the same library.

3. External procedures

   Next, HDevelop looks for external procedures (section 5.4) in the following order:

   (a) Session locations, i.e.:

   ```
   hdevelop -external_proc_path:<path name(s)>
   ```

   (b) Static user-defined locations

   (c) Standard procedure path

The tool tip of a procedure call reveals the location of the resolved procedure.

Resolution Example

As an example we look at a program with three local procedures. Two procedure directories are defined (disregarding the standard procedure path).

```
+ C:/Users/Public/
|
+--+ example.hdev
|  |
|  +--- main
|  +--- init
|  +--- compute_results
|
+--+ procedures/project/
|  |
|  +--- init.hdvp                public
|  +--- local.hdvp               private
|  +--- setup.hdvp               public
|  |
|  +--+ visualization.hdpl       library
|     |
|     +--- init                  private
|     +--- process               private
|     +--- setup                 public
|
+--+ procedures/common/
   |
   +--- config.hdvp              public
```

In procedure	a call to	resolves to
main	init	init (local procedure in example.hdev)
project/visualization.hdpl/setup	init	project/visualization.hdpl/init
common/config	init	project/init
main	process	-
project/visualization.hdpl/init	process	project/visualization.hdpl/process
compute_results	local	-
project/setup	local	project/local

5.6 Protected Procedures

Procedures can be protected by a password. Protected procedures may be executed by all users. However, the interface, documentation, and program code can only be accessed if the correct password is supplied.

Instead of protecting single procedures individually, all procedures of an HDevelop program or a library, respectively, can also be protected as a whole.

Protected procedures alter between two states:

- locked

  A protected procedure is locked if the password has not been entered in the current session. Locked procedures cannot be modified, and the program code is not visible in the program window.

- unlocked

  A protected procedure is unlocked after the correct password has been entered. Unlocked procedures may be modified, and the program code is visible in the program window.

To protect individual procedures, see section 6.4.9 on page 132.

To manage the protection state of multiple procedures, see section 6.2.2.16 on page 74.

5.7 Procedure Documentation

HDevelop procedures can be documented like operators. The documentation may include a detailed description of the functionality of the procedure, example code, links to other procedures or operators, and concise documentation of each parameter.

To manage a large collection of procedures, the procedures can be ordered in a hierarchical way, i.e., procedures can be ordered by chapters and sections just like operators.

The description of procedures can be formatted using Markdown syntax.

See section 6.4.8 on page 128 for details.

5.8 Legacy Procedures

Certain HDevelop procedures are obsolete. These procedures are only provided for backward compatibility. They are listed in the corresponding Legacy chapter and, just as for operators, a warning icon is displayed as a reminder in the operator window as well as in the program window. If you move the mouser cursor over the latter one you get a tool tip with corresponding warning message.

You can also add warning messages with corresponding icons yourself. For more information, see section 6.4.8.1 on page 128.

5.9 Just-In-Time Compilation

HDevelop supports just-in-time (JIT) compilation of procedures for optimized performance of HDevelop programs. The JIT compiler is enabled in the "experienced user" settings of the preferences dialog (see section "`General Options` ▷ `Experienced User`" on page 79). Once enabled, JIT compilation works without any user intervention. Before executing a procedure for the first time, HDevelop automatically compiles it to optimized bytecode. The additional compilation time is only relevant when executing a procedure for the first time in the current session. A re-compilation is triggered by changes to the corresponding procedure.

Messages of the JIT compiler are logged to the output console (see section "`Open Output Console`" on page 107). This includes informative events to indicate that a procedure has been (re-)compiled as well as warning events to indicate problems related to JIT compilation.

The speed-up that can be achieved by JIT compilation depends on the structure of the procedure code. If the procedure body contains only consecutive operator calls, the performance gain will be negligible. However, if the execution of a procedure includes a considerable amount of program line switching (e.g., loops), the performance gain can be significant.

When using F7 during program execution to step into a procedure, the procedure is executed uncompiled by the HDevelop interpreter.

If an error occurs inside a compiled procedure, the whole procedure call is aborted, i.e., the program counter will not indicate the actual program line that caused the error. To debug the corresponding procedure you can step into the procedure call with F7 (thereby interpreting its code) or disable JIT compilation altogether in the preferences.

Error message dialogs within `try-catch` blocks are always suppressed in JIT-compiled procedures regardless of the corresponding setting in the "experienced user" preferences (see section "`General Options` ▷ `Experienced User`" on page 79).

In HDevelop, the HALCON system parameter "use_window_thread" (see `set_system`) is activated by default to ensure that on windows systems all top-level HALCON graphics and text windows are opened in a special window thread. This setting must not be deactivated. Otherwise, HDevelop may hang, e.g., when displaying objects in a HALCON graphics window during regular execution and the window was previously opened during JIT-compiled execution.

JIT Compilation and Semantic Types

In JIT-compiled code, control variables always have the semantic type "any". In most cases this does not cause any problems since the semantic type is used mainly for visualization purposes (e.g., the inspection of variables). However, for thread IDs returned by the qualifier `par_start` (see section "Starting a Subthread" on page 313), the loss of the semantic type "thread_id" can lead to a different runtime behavior of JIT-compiled code. This is caused by the fact that the lifetime of a thread depends on any variables that are still referencing its thread ID. Now, when a control variable with the original semantic type "thread_id" is copied to another control variable inside a JIT-compiled procedure, HDevelop loses track of the additional reference because of the missing semantic type.

In general, procedures that manipulate variables with the semantic type "thread_id" should be excluded from JIT compilation, which is best done by classifying the semantic types of its parameters accordingly as described in section "`Parameter Documentation`" on page 131.

Limitations

JIT compilation is not supported for procedures that use any of the following features:

- procedure parameters with semantic type "thread_id" (see above)
- invalid program lines
- `stop` or active breakpoint
- `dev_inspect_ctrl`
- `dev_close_inspect_ctrl`
- `dev_error_var`
- `drag_region_*` operators
- `get_mbutton_*` operators
- `draw_*` operators
- procedure calls using the `par_start` qualifier
- `par_join`
- `dev_display` with a vector expression as variable
- `for` loop with a vector expression as index variable
- call of any procedure that cannot be JIT compiled due to the above reasons.

If one of these features is found, the corresponding procedure is called uncompiled as before by the HDevelop interpreter.

Note that when an exception is thrown in compiled procedures, the data slot “call_stack_depth” is always returned as `-1` (either when inspecting the exception in the variable window or using `dev_get_exception_data`).

Chapter 6

Graphical User Interface

This chapter is the reference to the graphical user interface of HDevelop.

6.1 Main Window

The main window handles HDevelop programs. It comprises the following components:

Window Title

The window title shows the name of the current program (or `unnamed` if no file name has been specified yet). Unsaved changes in the current program are indicated with a trailing asterisk (*) in the window title.

Menu

The menu at the top provides access to the functionality of HDevelop. The menus and their entries are described in the section “Menu” on page 51.

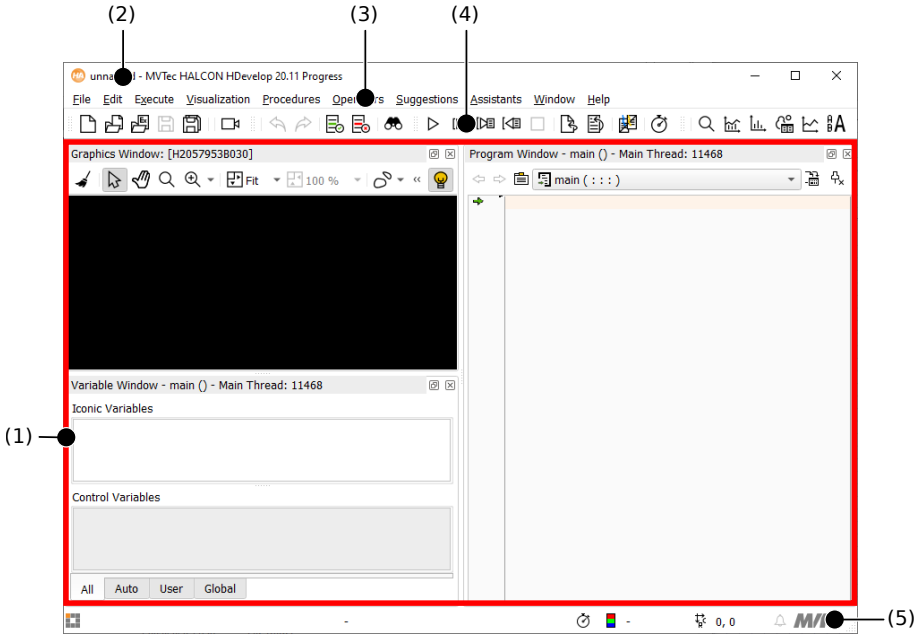


Figure 6.1: HDevelop main window: (1) window area, (2) window title, (3) menu, (4) tool bar, (5) status bar.

Tool Bar

The tool bar icons provide convenient shortcuts for frequently used functions. It is described in the section "Tool Bar" on page 113.

Window Area

The main part of the window is reserved for the dockable windows and dialogs of HDevelop. The most important windows are the following:

- Program window (see page 113)
- Graphics window (see page 158)
- Canvas window (see page 161)
- Variable window (see page 142)
- Operator window (see page 138)
- Online help window (see page 165)

Status Bar

The status bar at the bottom of the main window displays status information, e.g., context-sensitive information about a specific user action or the runtime of operator or procedure calls (unless time measurement has been deactivated in the preferences, see section "`Runtime Settings` ▷ `Runtime Settings`" on page 80).

For some very time-consuming operators, like `trainf_ocr_class_mlp` or `train_class_svm`, the status bar also displays a progress bar.

The status bar is divided into the following areas (from left to right, see figure 6.2):

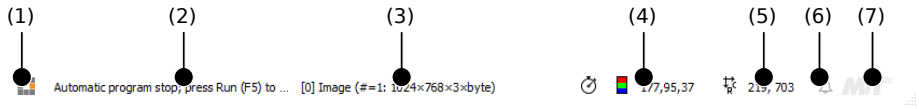


Figure 6.2: The status bar.

(1) Status icon

Shows the current run status of the program. : Program stopped, : Program executing.

(2) Messages and runtime information.

For example, if you select an operator from the menu, the corresponding short description is displayed here. The runtime information depends on the run mode:

- When single-stepping through the program, the runtime of the last operator or procedure call is displayed. If the option `Show memory usage` is activated in the preferences (see "`General Options` ▷ `Experienced User`" on page 79), the operator's required temporary HALCON memory will be displayed additionally. The displayed memory does not include objects created by the operator, like images or tuples.
- In continuous run mode, a runtime summary of the executed program lines is displayed when the program stops.

Right-clicking in the message area opens the context menu. It provides the following entries:

- `Show Processing Time`: Toggles whether execution messages are displayed in the status bar.
- `Copy History to Clipboard`: A history of the latest execution messages is displayed as a tool tip when placing the mouse pointer over the message area of the status bar. The history can be copied to the clipboard by selecting the entry `Copy History to Clipboard` in the context menu of the status bar.

- `Open Output Console`: The output console displays the message history in a separate window. See section "`Open Output Console`" on page 107.

(3) Information about the image in the active graphics window.

The display format is [*index*] *variable name* (#=*number of objects*: *height* x *width* x *channels* x *type*)

(4) Gray value of the image in the active graphics window at the mouse cursor position.

For multi-channel images, the gray values of all channels are displayed separated by commas.

(5) Image coordinates in the graphics window (row, column).

(6) Notification icon

Indicates new downloadable HALCON updates. : No new update available; : New HALCON update available. Click to open a version check in your web browser. See `https://www.mvtec.com/imprint` for information about MVTec's privacy policy.

(7) MVTec logo

Click on the logo to open our website `https://www.mvtec.com/products/halcon`.

6.2 Menu

The menu of the main window provides access to the complete functionality of HDevelop. Here, you may choose HALCON or HDevelop operators or procedures, or manipulate the graphical output. Every menu item opens a *pull-down* menu (henceforth abbreviated as menu) with optional submenus. You open a menu by clicking a menu item or via the keyboard (by pressing the Alt key in combination with the underlined letter of the menu item). In the following sections the menu entries are described in the order in which they appear.

6.2.1 Menu `File`

This menu provides functions to load existing programs and to save recently created or modified programs and procedures, respectively. See section "File Types" on page 42 for the supported file types. Furthermore, you may export HDevelop programs to C++, C, Visual Basic .NET, C#, or plain text and also print them. The menu also provides access to the supplied example programs and allows you to read arbitrary images.

Please note that the file type of programs and external procedures is persistent: If you load a program in the older *.dev* format and save it again, it will *not* be converted to the newer *.hdev* format unless explicitly specified in the dialog `Save Program As...` (or in the dialog `Save Procedure As...` for external procedures). Additionally, it is possible to convert HDevelop programs and procedures between the old and the new format by calling `hdevelop -convert` from the command line.

For new programs, the default file format (*.hdev*) will be used. When you save a program for the first time, you can also select the older file format in the corresponding dialog. If you want to use the older format all the time, you can make it the default by modifying the preferences, see `General Options` (page 76).

6.2.1.1 `New Program`

Synopsis: Initialize a new HDevelop program.

Checks for: Unsaved changes (page 191)

Shortcut: Ctrl+N

This menu item deletes the current program including all local procedures. The contents of variables are deleted before removing them. In addition, all graphics windows except one are closed. The last graphics window is cleared. The display parameters for the remaining graphics window are reset to the default values stored in the preferences (see section "`Visualization Settings` ▷ `Pen / LUT / Paint`" on page 80). The runtime settings of the preferences are reset to their default values (see section "`Runtime Settings` ▷ `Runtime Settings`" on page 80).

6.2.1.2 `Open Program...`

Synopsis: Load an existing HDevelop program.

Checks for: Unsaved changes (page 191)

Shortcut: Ctrl+O

A file selection dialog (page 190) pops up to select an HDevelop program. Please note that only native HDevelop programs (*.hdev* or *.dev*) can be loaded. Thus, text, C, C++, Visual Basic .NET, and C# versions of a file are rejected.

After you have loaded a program, the corresponding file name is added to the top of the menu `Recent Programs`. This allows you to quickly switch between recently loaded programs.

6.2.1.3 `Browse HDevelop Example Programs...`

Synopsis: Select an HDevelop example program from a categorized list.

Checks for: Unsaved changes (page 191)

Shortcut: Ctrl+E

Selecting this menu item opens a dialog that allows you to load HDevelop example programs grouped by categories. The dialog is displayed in figure 6.3.

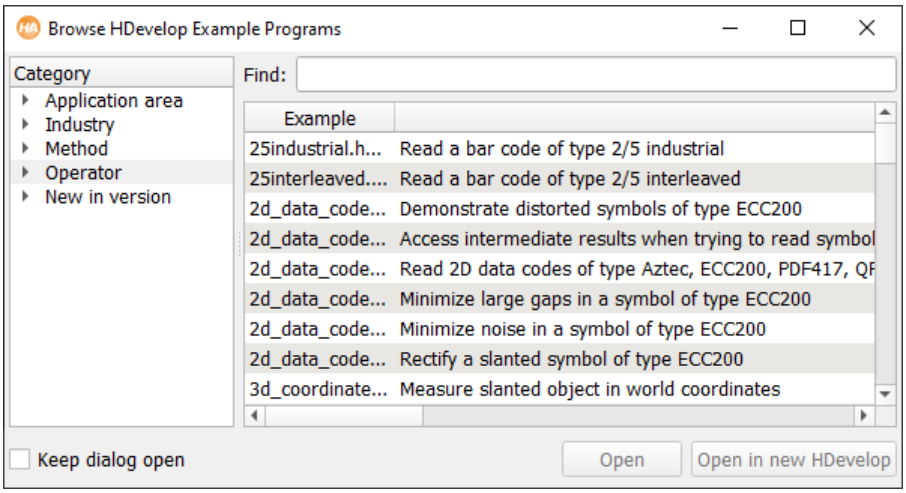


Figure 6.3: `Browse HDevelop Example Programs...`.

Browsing the Categories

The tree on the left contains a structured list of categories. Clicking the icon in front of a category toggles the display of its children. Alternatively, double-clicking any category label shows and hides the subcategories while also selecting the node of the tree. There can be multiple levels of categories. If you select a category, all its matching example programs are listed in the area on the right. You can select multiple categories by holding down the Ctrl key while clicking additional categories. An HDevelop example program may appear multiple times under different topics and categories.

Filtering the Matched Example Programs

Both the file name and the short description of the matched example programs are displayed. You can reduce the number of listed programs by entering a search string into the `Filter` text box. As you type, the list is updated to contain only example programs matching the string in either the file name or the short description field. The filtering is case-insensitive.

Loading an Example Program

Double-click an example program in the list, or select it and click the button Open. Clicking Open in new HDevelop opens the program in a new instance of HDevelop, which is useful if there are unsaved changes in the current program.

Either way, you can keep the dialog open by checking the corresponding box beforehand. This can be useful if you wish to scan through several example programs quickly.

6.2.1.4 Recent Programs

Synopsis: Load recently used HDevelop programs.

Checks for: Unsaved changes (page 191)

This submenu contains a list of the most recently used HDevelop programs. Simply click on a program name to load it. The number of entries in this menu may be customized in the preferences (see General Options ▷ General Options).

6.2.1.5 Open Procedure For Editing...

Synopsis: Load an existing external procedure or library.

From the file selection dialog (page 190) an arbitrary external procedure or library can be selected for editing. If a library is selected, all included procedures are loaded for editing. Procedures opened through this dialog need not be part of the configured procedure locations (section "Procedure Locations" on page 43).

6.2.1.6 Close Procedure

Synopsis: Close an HDevelop procedure.

This menu item closes an HDevelop procedure that is currently visible in the program window. If a procedure was opened explicitly, it will be kept open even if its path is not configured in the preferences. Closing the procedure will uncheck this option. Thus, if the procedures is not reachable via the procedure path, it will be closed.

6.2.1.7 Close All Procedures

Synopsis: Close all HDevelop procedures.

This menu item closes all HDevelop procedures that are currently opened. If a procedure was opened explicitly, it will be kept open even if its path is not configured in the preferences. Closing all procedures will uncheck this option for all open procedures. All procedures that are not reachable via the procedure paths will be closed.

6.2.1.8 Save

Synopsis: Save changes of the current HDevelop program or the currently selected external procedure.

Shortcut: Ctrl+S

The actual functionality of this menu entry depends on the selected procedure in the program window:

- *Main or local procedure selected in program window:*

  Save changes of the current HDevelop program. If no file name has been specified yet, a file selection dialog (page 190) will be opened where you can specify the program name and optionally the file format (see section "File Types" on page 42). Local procedures are saved within the HDevelop program.

  The file name of the program you save is added to the menu Recent Programs.

  Please note that modified external procedures are not saved automatically. To save them as well, select the menu entry Menu File ▷ Save All, or select the corresponding external procedure in the program window and click Menu File ▷ Save again (see below).

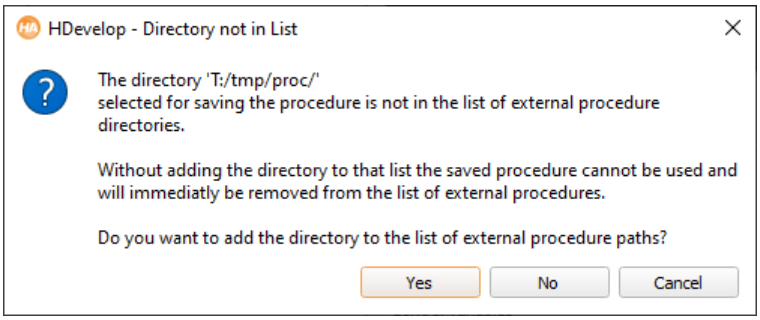


Figure 6.4: Adding a new directory to the list of external procedure directories.

- *External procedure selected in program window:*

  Save changes to the currently selected external procedure back to the originating file. The operation is done quietly. A modified external procedure is marked with an asterisk (*) in the program window.

If you try to save a file that has been modified outside of your running instance of HDevelop (possibly by another user), a warning message is displayed asking whether you want to overwrite the file. If you are uncertain about the external changes to the file, we recommend clicking `No`, and then saving your program under a different name using `Save Program As...`.

6.2.1.9 `Save Program As...`

Synopsis: Save the current HDevelop program to a new file.

Shortcut: Ctrl+Shift+S

A file selection dialog (page 190) is opened. You can specify a new file name and optionally a file format (see section "File Types" on page 42), and save the current program under that name. The new file name becomes the default name of the current program so that subsequent `Save` operations will use that name instead of the old.

The file name of the program you save is added to the menu `Recent Programs`.

6.2.1.10 `Save Procedure As...`

Synopsis: Save the current procedure as an external procedure or as a stand-alone HDevelop program.

Shortcut: Ctrl+Shift+P,S or Ctrl+Shift+P,Ctrl+Shift+S

Using this menu entry you can save the currently selected procedure as an external procedure or an HDevelop program. A file selection dialog (page 190) is opened where you can select the file type (see also section "File Types" on page 42):

- HDevelop procedures (**.hdvp* or **.dvp*)

  The procedure is saved as an external procedure. If the target directory is not already configured in the external procedure directories (see Menu `Edit` ▷ `Preferences`, `Directories` (page 71)), HDevelop will suggest adding the directory to the list. An example dialog is displayed in figure 6.4. If you click `No`, HDevelop will not be able to access the saved procedure unless the directory is later added to the external procedure locations manually.

  This is one method to make an internal procedure external. If you do not change the name of the procedure, the internal procedure will conceal the external procedure.

- HDevelop local procedure (**.hdev* or **.dev*)

  If this file type is selected, an empty main procedure is added to the target file, and the procedure is added to the program as a local procedure.

This menu item is disabled if the main procedure is selected in the program window.

6.2.1.11 Save All

Synopsis: Save the current program and all modified external procedures.

Shortcut: Ctrl+Alt+S

If no name has been specified for the current program yet, the behavior is similar to that of Save Program As.... In addition, all modified external procedures marked with an asterisk (*) in the program window's combo box are saved.

6.2.1.12 Export

This menu contains export functionality.

☐ Export...

Synopsis: Export program code to a programming language or as a text file.

Shortcut: Ctrl+Shift+O,X or Ctrl+Shift+O,Ctrl+Shift+X

See also: hdevelop -convert (command line switch (page 351)).

Using this dialog, you can select an export format and write (parts of) the current program to a file in that format. The dialog is displayed in figure 6.5.

The button next to the export file name opens a file selection dialog (page 190) to select a file name and an export format. The following formats are supported:

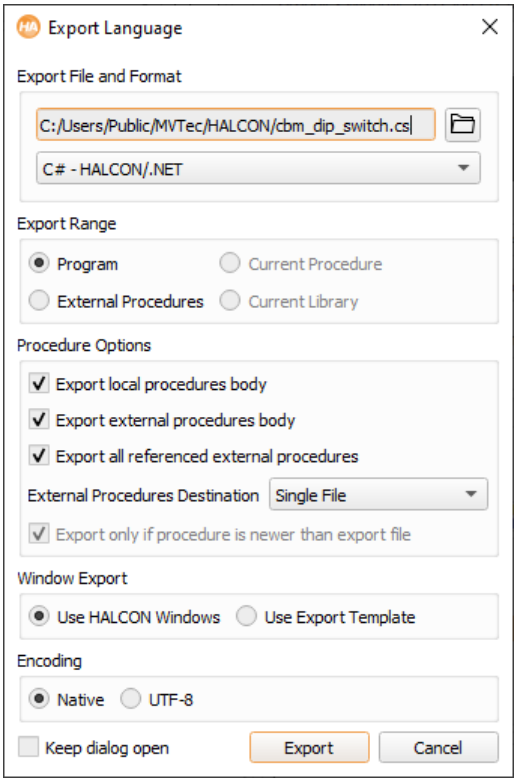


Figure 6.5: Export.

Format	Syntax	Extension	More information
Text File	HALCON/HDevelop	`.txt`	Plain text export
C	HALCON/C	`.c`	section 10.2.4 on page 341
C++	HALCON/C++	`.cpp`	section 10.2.1 on page 335
Visual Basic .NET	HALCON/.NET	`.vb`	section 10.2.3 on page 339
C#	HALCON/.NET	`.cs`	section 10.2.2 on page 338

The file name extension corresponding to the selected export format is appended to the specified file name.

`Export Range`: The export range specifies which parts of the current program are to be exported. The following options are available:

- `Program`: The entire program is exported (main procedure and all local procedures). All used external procedures are exported depending on the setting of the external procedure options (see below).
- `Current Procedure`: The current procedure and all used local procedures are exported. All used external procedures are exported depending on the setting of the external procedure options (see below).
- `External Procedures`: All external procedures are exported depending on the setting of the external procedure options (see below).
- `Current Library`: The current library is exported (all procedures that are part of the current library). All used external procedures (that are not part of the current library) are exported depending on the setting of the external procedure options (see below).

The short description and chapter information of procedures are exported as comments. Arbitrary code can be embedded with special comment lines (see section 10.2.5 on page 342).

`Procedure Options`: Defines the export behavior for procedures.

- `Export local procedures body`: If checked, both the declaration and the body of local procedures is exported. Otherwise, only the declaration is exported.
- `Export external procedures body`: If checked, both the declaration and the body of external procedures is exported. Otherwise, only the declaration is exported.
- `Export all referenced external procedures`: Determines, if all referenced (i.e., used) external procedures are also exported with the current program.
- `External Procedures Destination`: By default, external procedures are exported to a single file. Select `Separate Files` to export external procedures to separate files. The file name corresponds to the procedure name while the file extension is derived from the export format. Select `Separate or Library Files` to export external procedures to separate files but keep the files of libraries together in a single file.
- `Export only if procedure is newer than export file`: Export only those procedures to separate files whose time stamp is newer than the time stamp of the destination file.

`Window Export`: Specifies the export behavior of HALCON windows:

- `Use HALCON Windows`: Export as a stand-alone project.
- `Use Export Template` (HALCON/.NET only): Export as a project using the supplied project template.

`Encoding`: Specifies the encoding of exported programs. The following options are available:

- `Native`: Export in the encoding defined by the operating system.
- `UTF-8`: Force export in UTF-8 encoding (Unicode).

For C/C++, the encoding must match the interface encoding used by the application. For C#, exporting to UTF-8 should always work fine.

`Keep dialog open` Checking this box keeps the dialog open for subsequent exports.

☐ `Export Library Project...`

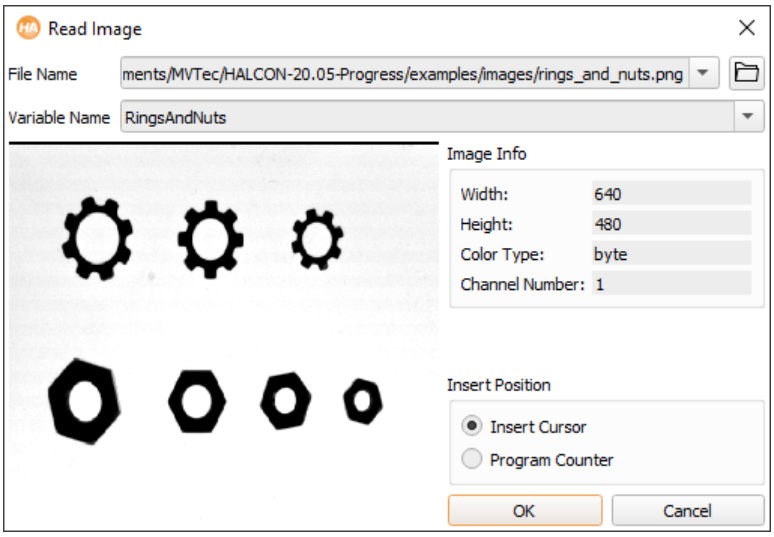


Figure 6.6: `Read Image...`.

Synopsis: Export library/local procedures as a C++ or C# project

See also: `hdevelop -export_project` (command line switch (page 351)).

Exports a selected procedure library or the local procedures of the current program as a C++ or C# project, which can be integrated into your own projects. See the full description at Library Project Export (page 331).

6.2.1.13 `Read Image...`

Synopsis: Add an image file to the current program.

Shortcut: Ctrl+R

See also: `read_image`.

This menu entry provides a quick way to open an image file and access it in the current program. The associated dialog `Read Image` is displayed in figure 6.6.

`File Name` You can enter the name of an image file into this field. A relative file name is looked up in several directories in that order:

1. The current working directory (`.`, a single dot), i.e., the directory HDevelop was started from.
2. The subdirectory `images` of the directory specified by the environment variable `HALCONROOT`.
3. The directories specified by the environment variable `HALCONIMAGES`.
4. The directories used in previous invocations of `Read Image...`.

The first matching image file is displayed as a thumbnail preview along with its width, height, color type and number of channels.

Alternatively, open a file selection dialog (page 190) by selecting a predefined directory from the `File Name` combo box or clicking the browse button. The latter will start browsing in the current working directory, or in the last used directory. Depending on the operating system you may be able to switch to a thumbnail view in the file selection dialog.

`Variable Name` HDevelop suggests a variable name derived from the selected file name. You may adopt or edit this name. If you want to use a name of an already created iconic variable, this combo box offers you all known iconic variable names. Simply click the arrow on the right side of the combo box to select a variable name. Note that the reuse of a variable deletes the old content and replaces it with the new image.

After selecting a file name, click `OK` to load the image into HDevelop. The operator `read_image` is inserted at the specified insert position (IC or PC). The specified iconic variable is updated in the variable window and the image is displayed in the active graphics window. Clicking `Cancel` aborts the operation.

By default, an absolute path to the selected image is inserted. You can instruct HDevelop to use relative path names (see `General Options` ▷ `General Options` in the preferences).

6.2.1.14 `Properties`

Synopsis: Display various properties of the current program.

The tab card `General` displays file properties of the current program, such as file name, path, creation and modification date, and write permission. It also shows the file size, the number of lines of code, used and unused local procedures, used external procedures and used protected procedures. This is displayed in figure 6.7.

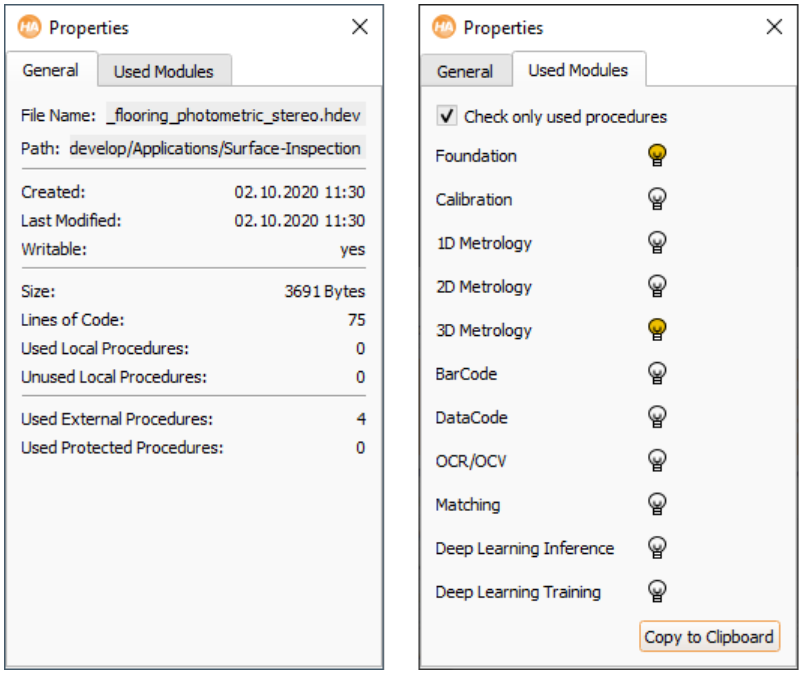


Figure 6.7: `Properties`: `General` (left), `Used Modules` (right).

The tab card `Used Modules` lists the HALCON modules used by the current program. Modules marked with 💡 are used. This window allows you to get an estimate of how many modules your application will need in a runtime license. Please refer to the Installation Guide for more information about modules and runtime licenses. See figure 6.7 for the corresponding dialog of an OCR example.

`Check only used procedures` If checked, only used procedures are considered for the evaluation of the used modules. Otherwise, all procedures are considered.

`Copy to Clipboard` Copy the names of the used modules to the system clipboard. This way the list can be easily pasted into other applications.

6.2.1.15 `Print...`

Synopsis: Print the current program or selected procedures.

Shortcut: Ctrl+P 🖶

The print dialog is displayed in figure 6.8.

Print Range

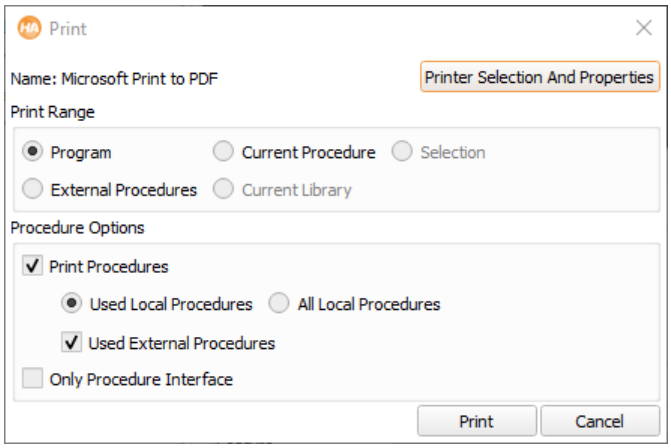


Figure 6.8: Print.

Program Complete program including all procedures.

Current Procedure Current procedure and its used procedures.

Selection Highlighted program lines and their used procedures.

External Procedures All external procedures.

Procedure Options

Print Procedures Define whether procedures are printed or not.

- Used Local Procedures: print only used local procedures.
- All Local Procedures: print all local procedures.
- Used External Procedures: also print used external procedures.

Only Procedure Interface If this box is checked, the procedure body is not printed. Instead, only the interface of the procedure is printed.

The bodies of locked procedures (see section "Protected Procedures" on page 45) are not printed.

6.2.1.16 Quit

Synopsis: Quit HDevelop.

Checks for: Unsaved changes (page 191)

Shortcut: Ctrl+Q

See also: exit.

This menu item terminates HDevelop.

6.2.2 Menu Edit

In this menu you find all necessary functions to modify the current HDevelop procedure body displayed in the program window. Furthermore, a comprehensive find and replace functionality is offered. You can also access the preferences of HDevelop from this menu.

6.2.2.1 Undo

Synopsis: Undo your previous editing activities.

Shortcut: Ctrl+Z

You may undo your previous editing activities via this menu item. For example, by selecting it three times you cancel the last three editing actions. The menu entry always states the last editing action that will be undone.

The undo functionality purely applies to *editing* activities. No file operations will be undone. Thus, if you create an external procedure from some selected lines and undo the operation, the external procedure will not be removed from the file system.

The undo item does not work for the password assignment for procedures (see section "Protected Procedures" on page 45). To undo the password assignment you either have to remove the password as long as you can edit the procedure, or you quit HDevelop without saving the corresponding procedure.

6.2.2.2 Redo

Synopsis: Revoke undo activities.

Shortcut: Ctrl+Y

This is a quick way to restore the state before the last undo operation. The menu action explicitly states the last Undo action that will be revoked.

6.2.2.3 Cut

Synopsis: Cut the highlighted text in the program window to an internal buffer.

Shortcut: Ctrl+X or Shift+Del

The highlighted text is deleted from the selected procedure and placed into an internal buffer and the system clipboard for later use.

Additionally, for every procedure call the corresponding procedure and all procedures that can be reached from it are copied to the buffer. This is necessary to obtain a consistent program when pasting procedure call lines to a program in which the corresponding procedures might not exist.

6.2.2.4 Copy

Synopsis: Copy the highlighted text from the program window to an internal buffer.

Shortcut: Ctrl+C

The highlighted text is copied to an internal buffer and the system clipboard.

Additionally, for every procedure call the corresponding procedure and all procedures that can be reached from it are copied to the buffer. This is necessary to obtain a consistent program when pasting procedure call lines to a program in which the corresponding procedures might not exist.

6.2.2.5 Paste

Synopsis: Insert text into the currently selected procedure at the IC.

Shortcut: Ctrl+V or Shift+Ins

You can insert code lines from previous Cut or Copy operations or text placed in the system clipboard into the current procedure. The text is inserted at the text cursor position.

If the paste buffer contains local procedures that do not exist, they are copied depending on the preferences (see section "General Options ▷ General Options" on page 76). By default, HDevelop will ask whether the corresponding local procedures shall be added. If, however, the paste buffer contains local procedures that do already exist, no further copies of such procedures are added. This is the case when the same content of the clipboard is pasted repeatedly.

If the paste buffer contains calls to external procedures, the paths to those procedures are copied, too. However, before an external procedure path is added during a paste action, you are asked whether or not you want to add that particular path to the external procedure paths.

The mechanism of copying and pasting procedure calls together with the corresponding procedures is an easy way to transfer procedures between different HDevelop programs. It also works between multiple instances of HDevelop. The contents of the internal buffer are kept, allowing this command to be repeated.

6.2.2.6 Delete

Synopsis: Delete the highlighted text from the program window.

Shortcut: Del 

This menu item deletes the highlighted text without storing it in an internal buffer. The only way to restore the deleted text is to use the menu item Undo.

6.2.2.7 Activate

Synopsis: Enable the highlighted program lines.

Shortcut: F3 

All of the highlighted program lines that were previously disabled using the Deactivate command are re-enabled. If the program is currently running, it is stopped before the program lines are activated. Comment lines created with the operator comment are unaffected by this command.

6.2.2.8 Deactivate

Synopsis: Disable the highlighted program lines.

Shortcut: F4 

The highlighted program lines are disabled. This is a quick way to suppress the execution of portions of the program for testing purposes. An asterisk is placed at the beginning of the selected lines, i.e., the lines appear as comments in the program window and have no influence on the program during runtime. If the program is currently running, it is stopped before the program lines are deactivated. Comment lines created with the operator comment are unaffected by this command.

The deactivated lines are still part of the program, i.e., they are stored like all other lines and their variables are still needed like all other variables. To reverse this action, select Activate.

Note that you can insert a *real* comment by starting a program line with an asterisk (*) as the first non-whitespace character, or using the operator comment.

6.2.2.9 `Find/Replace...`

Synopsis: Find and replace text in the current program.

Shortcut: Ctrl+F

This dialog provides comprehensive facilities for searching the program code. You can perform a full text search or search for variable names as well as operator (or procedure) calls. In addition, you can replace variable names and substitute operator or procedure calls. The dialog is displayed in figure 6.9.

The search context can be set to one of the following entities:

- `Variables`
  Find program lines with variable names that match the search text.
- `Operators`
  Find program lines with operator or procedure calls that match the search text.
- `Texts`
  Full text search. Find program lines that match the search text anywhere.

The search scope can be specified as follows:

- `All`
  Search the main procedure, all local and all external procedures.
- `Program`
  Search the main procedure and all used procedures.
- `Current Procedure`
  Search the current procedure only.

Please note that locked procedures are not searched (see section “Protected Procedures” on page 45).

The following parameters specify how the search is performed:

- `Case Sensitive`
  By default, the case of the search text is ignored, thus searching for `image` will find `Image` or `IMAGE` as well. Check this box to make the search case-sensitive.
- `Whole Words`
  By default, program lines are matched even if the search text is only part of a word, thus an operator search for `grab_image` also matches operator calls to `grab_image_async` or `grab_image_start`. Check this box to find only exact matches.
- `Backward`
  Check this box to reverse the search direction.

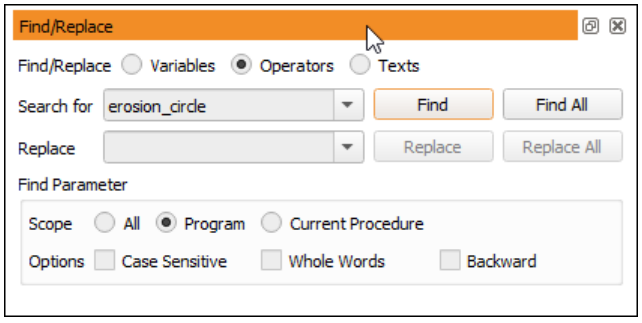


Figure 6.9: Looking for operator calls containing “grab_image”.

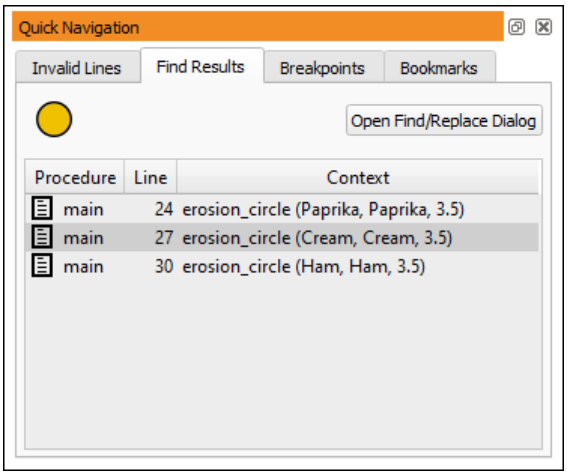


Figure 6.10: Finding all occurrences of the search text.

Finding Single Occurrences of the Search Text

Enter the search text and click `Find`. If there is no match, the text field will blink shortly. Otherwise, the first matching program line in the current procedure is highlighted. Each subsequent click of `Find` highlights the next matching program line. If the last matching line of the current scope has been reached, the text field blinks shortly. The next click on `Find` starts over at the beginning.

Finding All Occurrences of the Search Text

Enter the search text and click `Find All`. The search result will be presented in a separate window, i.e., the tab card `Find Results` of the quick navigation window (see section "`Open Quick Navigation`" on page 108).

Clicking an entry focuses the corresponding program line in the active program window. If the selected procedure is already displayed in a program window tab, the corresponding tab is activated. Otherwise, the current view switches to the selected procedure.

You can even select multiple lines from the search result by holding the Ctrl key. The following actions may be performed for all selected lines (either from the context menu of the search result or the corresponding menu entries): `Cut` (page 60), `Copy` (page 60), `Delete` (page 61), `Activate` (page 61), and `Deactivate` (page 61).

- Add bookmarks to the selected program lines. This action will also add the selected program lines to the tab `Bookmarks`. This way you can "remember" a search result for later use; just open the quick navigation window again (Menu `Window` ▷ `Open Quick Navigation`) and go to the tab card `Bookmarks`.

The "find all" operation is recommended before doing a global replace to preview which program lines will be affected. An example is displayed in figure 6.10.

Replacing Variable Names

Click `Variables` to specify the search context. Enter the search text and the replace text. You can replace parts of variable names by keeping `Whole Words` unchecked.

Click `Find` until the desired line is found. Afterwards, click `Replace` to replace *all* occurrences of the search text in the matched line. The next matching line is highlighted automatically.

Click `Replace All` to replace all occurrences of the search text in the specified scope. We recommend doing a `Find All` beforehand to estimate the extent of this operation.

Replacing Operator Calls

You can replace one operator or procedure call with another. Because different operators very likely have different parameters, the source parameters have to be mapped to the target parameters beforehand. See figure 6.11 for an example.

Click `Operators` to specify the search context. Enter the source operator or procedure name and the target operator or procedure name. When both names are specified, the parameters of the target operator/procedure are listed at the bottom of the dialog. For every target parameter you have to select or enter a corresponding source parameter.

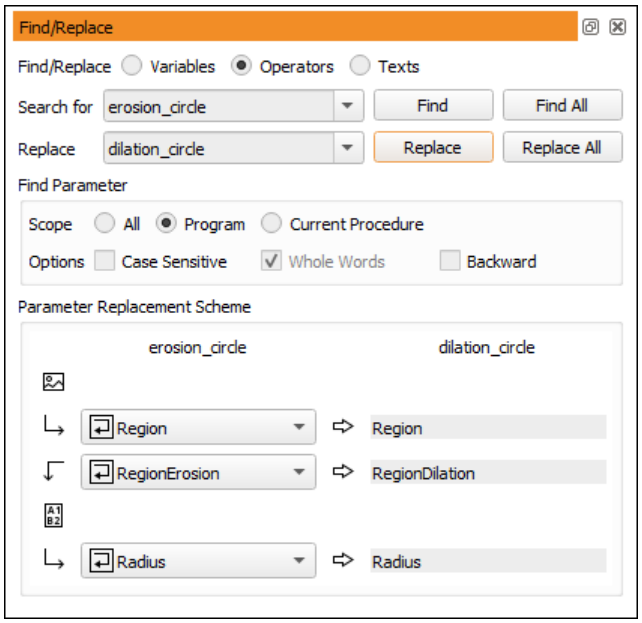


Figure 6.11: Replacing operator calls.

Please note that the option `Whole Words` is always enabled in this mode because only exact matches are valid when replacing operator calls.

6.2.2.10 `Find Again`

Synopsis: Find the next match of the last entered search string.

Shortcut: Ctrl+G

This menu item repeats the search specified via the menu item `Find/Replace...`.

6.2.2.11 `Set/Clear Bookmark`

Synopsis: Toggle bookmarking of selected program lines.

Shortcut: Ctrl+F11

Bookmarks help you remain focused when editing large programs or programs using many procedures. You can set bookmarks to selected lines, jump between bookmarks, or manage bookmarks from the quick navigation window (see section "`Open Quick Navigation`" on page 108). Bookmarked program lines are highlighted by in the left column of the program window. Bookmarks are transient, i.e., they are not saved with the program.

6.2.2.12 `Next Bookmark`

Synopsis: Jump to next bookmarked program line.

Shortcut: F11

The cursor jumps to the next bookmark, i.e., the next bookmarked program line in the current procedure, or the first bookmarked program line in the first following procedure that contains bookmarks. The procedures are traversed in alphabetical order with the exception of the main procedure which always comes first in the list.

6.2.2.13 Previous Bookmark

Synopsis: Jump to previous bookmarked program line.

Shortcut: Shift+F11

The cursor jumps to the previous bookmark, i.e., the previous bookmarked program line in the current procedure, or the last bookmarked program line in the first preceding procedure that contains bookmarks. The procedures are traversed backwards in alphabetical order with the exception of the main procedure which always comes first in the list.

6.2.2.14 Manage Bookmarks

Synopsis: Manage program line bookmarks.

Shortcut: Ctrl+Shift+O,F11 or Ctrl+Shift+O,Ctrl+Shift+F11

The quick navigation window (see section "Open Quick Navigation" on page 108) contains a tab card to manage your bookmarks and navigate to the bookmarked program lines.

The tab card lists all program lines that are currently bookmarked. Clicking an entry focuses the corresponding program line in the active program window. If your program contains many bookmarks, this will be more transparent and convenient then repeatedly jumping through your selection of bookmarks with F11 and Shift+F11. If the selected procedure is already displayed in a program window tab, the corresponding tab is activated. Otherwise, the current view switches to the selected procedure.

- Clear the bookmarks of the selected program lines and immediately remove the corresponding entries from the list. This action can also be triggered from the context menu, or by pressing Del.
- Clear all bookmarks and remove all entries from the list, leaving you with an empty window.

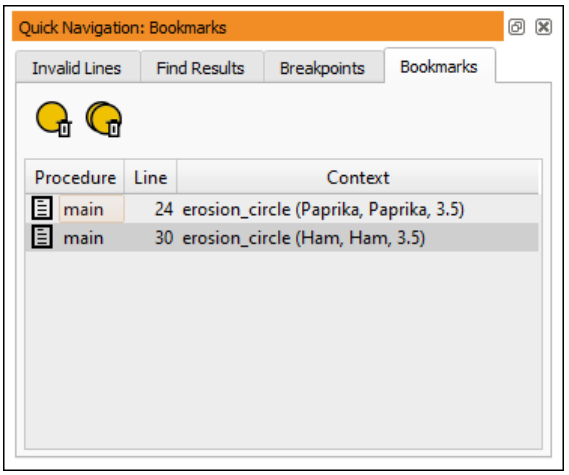


Figure 6.12: Managing bookmarks in the quick navigation window.

6.2.2.15 Invalid Lines

Synopsis: Manage invalid program lines.

Shortcut: Ctrl+Shift+O,F12 or Ctrl+Shift+O,Ctrl+Shift+F12

The quick navigation window (see section "Open Quick Navigation" on page 108) contains a tab card to manage invalid lines in your program. Examples of invalid lines are unresolved procedure calls, operator or procedure calls with a wrong number or type of parameters, or syntax errors.

The tab card lists all invalid program lines within the currently selected scope (All refers to all procedures, Program refers to the main procedure and all procedures it uses, and Current Procedure refers to procedure displayed in the active program window). Hopefully, you can set the scope to All without cluttering your display.

Clicking an entry focuses the corresponding program line in the active program window so you fix the error. If the selected procedure is already displayed in a program window tab, the corresponding tab is activated. Otherwise, the current view switches to the selected procedure.

If you make changes to the program while the display of invalid program lines is still open, you will need to click the button Refresh to update the entries.

Using the context menu, you can perform the following actions on the selected program lines:

- Copy Values (page 60)
- Cut (page 60)
- Delete (page 61) (also by pressing Del)
- Activate (page 61)
- Deactivate (page 61)

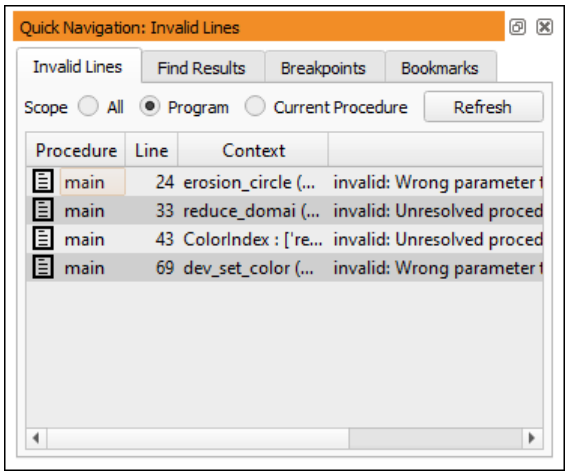


Figure 6.13: Managing invalid program lines in the quick navigation window.

6.2.2.16 Preferences

Synopsis: Set global preferences of HDevelop.

Shortcut: Ctrl+Shift+O,S or Ctrl+Shift+O,Ctrl+Shift+S ⚙

HDevelop maintains a set of preferences that are persistent between sessions. You can customize the appearance of HDevelop's user interface (syntax highlighting, fonts, and language) as well as its behavior, configure the settings of procedures, and change the default visualization settings of the graphics windows.

The following settings are provided:

User Interface:

- Program Window
- Fonts (page 70)
- Language (page 70)
- Themes (page 70)

Procedures:

- Directories (page 71)
- External Procedures (page 72)
- Manage Procedure Libraries (page 73)
- Manage Passwords (page 74)
- Procedure Use (page 75)
- Unresolved Procedure Calls (page 75)

General Options:

- General Options (page 76)
- Experienced User (page 79)

Visualization Settings:

- Pen (page 80)
- LUT (page 80)
- Paint (page 80)

Runtime Settings:

- Runtime Settings (page 80)
- Override Operator Behavior (page 81)

Telemetry (page 82)

Changes to the settings in this dialog are saved automatically without any user intervention. The location of the generated file depends on the operating system:

Windows: `%APPDATA%\MVTec\HDevelop.ini`

Linux: `$HOME/.hdevelop/MVTec/HDevelop.ini`

macOS: `$HOME/Library/Preferences/com.mvtec.HDevelop.plist`

HDevelop can read additional preferences from another startup file specified with the command line switch `-add_preferences <file>`. To load the preferences only from another startup file, start HDevelop with the command line switch `-load_preferences <file>`. In both cases the settings will be stored back to `HDevelop.ini` when HDevelop is quit. To use a different startup file altogether, start HDevelop with the command line switch `-use_preferences <file>`.

The preferences dialog provides its own menu with the following entries:

`Import` Using this menu entry you can import a selection of preferences which were previously saved using the menu entry `Export` (see below). The dialog is displayed in figure 6.14.

In the import dialog you can select a file with saved HDevelop preferences (default file extension: `.hdp`). The check boxes allow you to import groups of settings selectively. They correspond to the categories of the dialog. The runtime settings are not persistent and can neither be exported nor imported.

`Export` The export dialog is identical to the import dialog. Using the check boxes you can specify which settings will be saved to the selected file.

`Reset` Selecting this menu entry resets all preferences (except the window geometry and layout) to the default settings. If you want to reset the window geometry as well you can start HDevelop with the following command line switch:

```
hdevelop -reset_preferences
```

The preferences dialog contains a list of categories on the left and several related tab cards on the right. The size of these elements is controlled by a splitter. The available categories of preferences are described in the following sections.

☐ User Interface ▷ Program Window

`Font:` Specifies the font that is used in the program window.

`Indent Size:` Specifies the number of spaces an indenting level in the program window accounts for. In HDevelop the bodies of loops and conditionals are indented automatically.

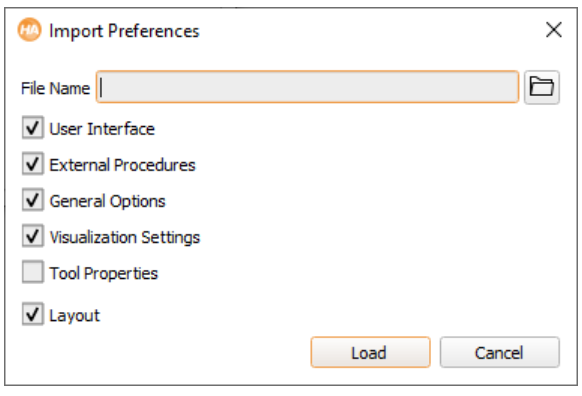


Figure 6.14: `Import...`

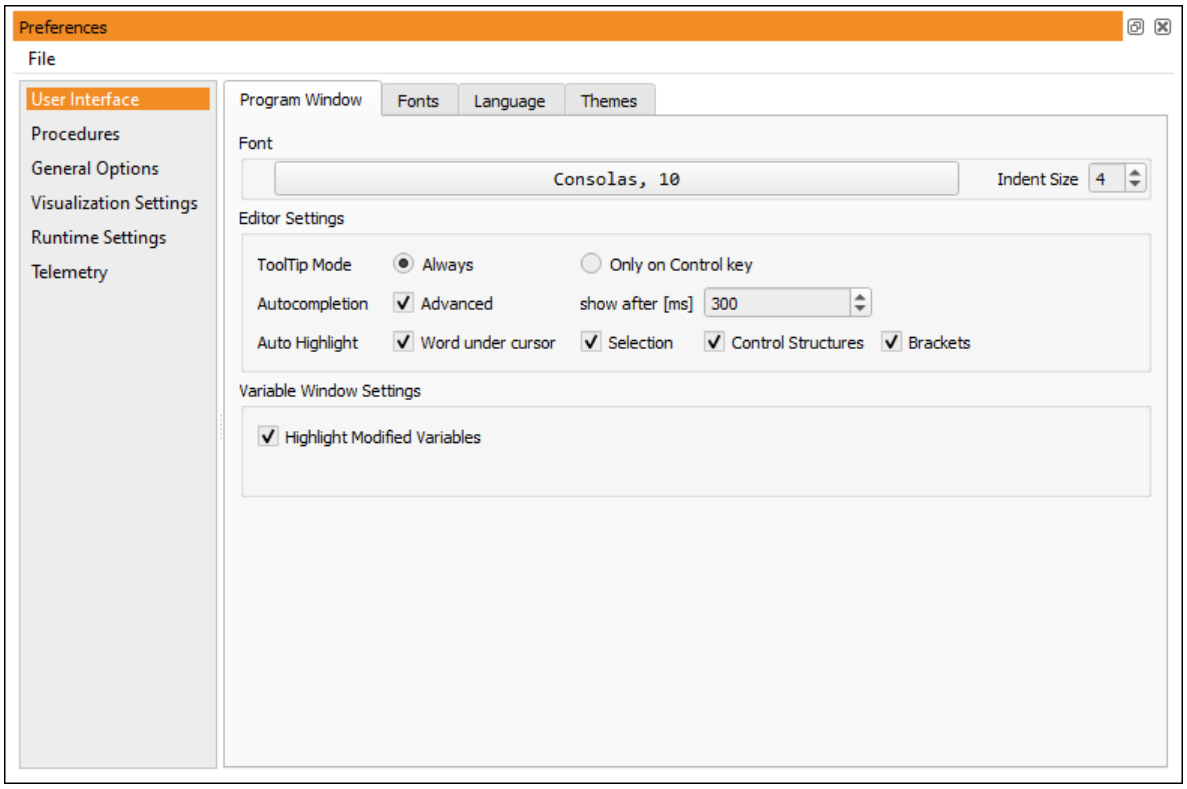


Figure 6.15: `User Interface` ▷ `Program Window`.

`Editor Settings`**:** These settings specify the editing behavior in the program window.

`ToolTip Mode`:

When the mouse cursor rests on a program line, a tool tip is displayed by default. When placed over an operator name, the interface of the corresponding operator and its short description is displayed. When placed over an instantiated iconic variable, an icon is displayed. When placed over the program line column, warning messages may be displayed, e.g., if the used operator is deprecated. This behavior may be turned off if it disturbs your editing activities.

- `Always`

  Always display a tool tip when the mouse cursor rests on a program line.

- `Only on Control key`

  Display the tool tip only when the Ctrl key is pressed.

`Autocompletion`:

- `Advanced`

  Enables advanced autocompletion in the program window (see section “Editing Programs” on page 115). This option can also be set in the program window (see section “Program Window Actions” on page 114).

- `show after [ms]`:

  This value defines the delay before the autocompletion is displayed.

`Auto Highlight`:

Enables the auto highlight feature which can highlight

- occurrences of the word under the cursor,
- occurrences of the currently selected string,
- matching brackets, and
- matching control structures.

If a matching control structure is highlighted, other instances of the word are not highlighted to avoid visual clutter.

`Variable Window Settings:` Enables the highlighting of modified variables in the variable window (page 142) and of modified entries in handle inspect windows. Note that a variable is marked as modified if it was written into during the last step or run operation, even if its value did not change because the same value was written into the variable again.

The color set for `Background of Current Line` is used to highlight modified variables.

☐ `User Interface` ▷ `Fonts`

In this tab card, the font settings of HDevelop may be modified.

- `General`

  The font used throughout the user interface (menu entries, labels etc.)

- `Help Window`

  The body font used in the help window (menu `Help` (page 110) ▷ `Help`).

- `Program Window`

  The font used in the program window. This is the same font setting as on the tab card `Program Window` (see above).

- `Advanced Autocompletion`

  The font used in the advanced autocompletion overlays.

- `Values and Parameters`

  The font used for displaying values in the variable window and associated inspection windows as well as parameters in the operator window.

- `Printing`

  The font used when printing program listings.

☐ `User Interface` ▷ `Language`

In this tab card you can change the language of the user interface. Please note that HDevelop needs to be restarted if a different language is selected. By default, HDevelop uses the language that is specified in the operating system locale. For this, the environment variables `LANG`, `LC_ALL`, `LC_CTYPE`, and `LC_COLLATE` are tested in that order. Once the language is changed in this dialog, the operating system locale is disregarded.

☐ `User Interface` ▷ `Themes`

- `Current theme`

  Specifies the appearance of HDevelop, for example the background color and the icon set. You can choose between a dark and a light theme.

- `Colors`

  Specifies the colors of the current theme. Click the color field to select a color. To reset a color, click . To reset all colors, click `Reset all colors`.

☐ `Procedures` ▷ `Directories`

Use this tab card to manage the list of directories that contain external procedures. The directories are recursively scanned for external procedures in their listing order (see section 5.4 on page 43).

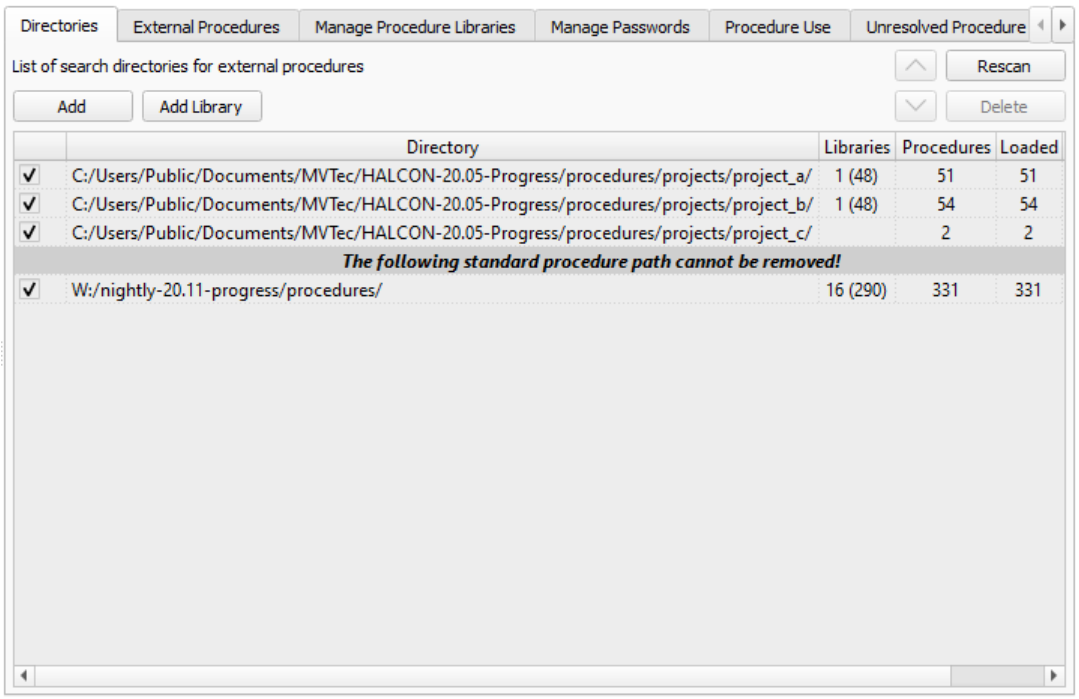


Figure 6.16: `Procedures` ▷ `Directories`.

Libraries: For each directory, the number of libraries is displayed. The number in parentheses sums up the number of procedures contained in libraries.

Procedures: This column displays the total number of procedures, i.e., the number of procedures found in libraries and all other external procedures.

Loaded: The number of loaded procedures is usually equal to the total number unless the directory contains corrupted procedure files, or files with the wrong permissions.

Procedures with the extension *.hdvp* always override procedures with the same name but the extension *.dvp* in the same directory.

Please note that HALCON comes supplied with a set of standard procedures. These are general-purpose procedures used by many of the supplied example programs. The path cannot be altered or deleted. It is, however, possible to override the supplied external procedures by placing external procedures with the same name in one of the user-defined directories.

The documentation of the supplied procedures is available in the online help of HDevelop under Procedure Reference Manual.

`Add`: Select an additional directory from the file selection dialog (page 190). This directory will be added to the list. All subdirectories of the selected directory will be scanned as well.

`Add Library`: Explicitly add a library (*.hdpl*) file to the list of directories. This will make the library procedures available without looking at other procedures in the same directory.

`Delete`: Delete the selected entry from the list. Programs using any external procedure from that directory will no longer run.

`Rescan`: Rescan all listed entries to reflect any changes in the file system.

Using the check boxes you can disable the corresponding directories temporarily without removing them from the list.

The list entries can be reordered using the up/down arrow buttons.

☐ `Procedures` ▷ `External Procedures`

This tab card lists all external procedures in the order they are loaded from the configured directories (page 71). For each procedure, the following information is displayed:

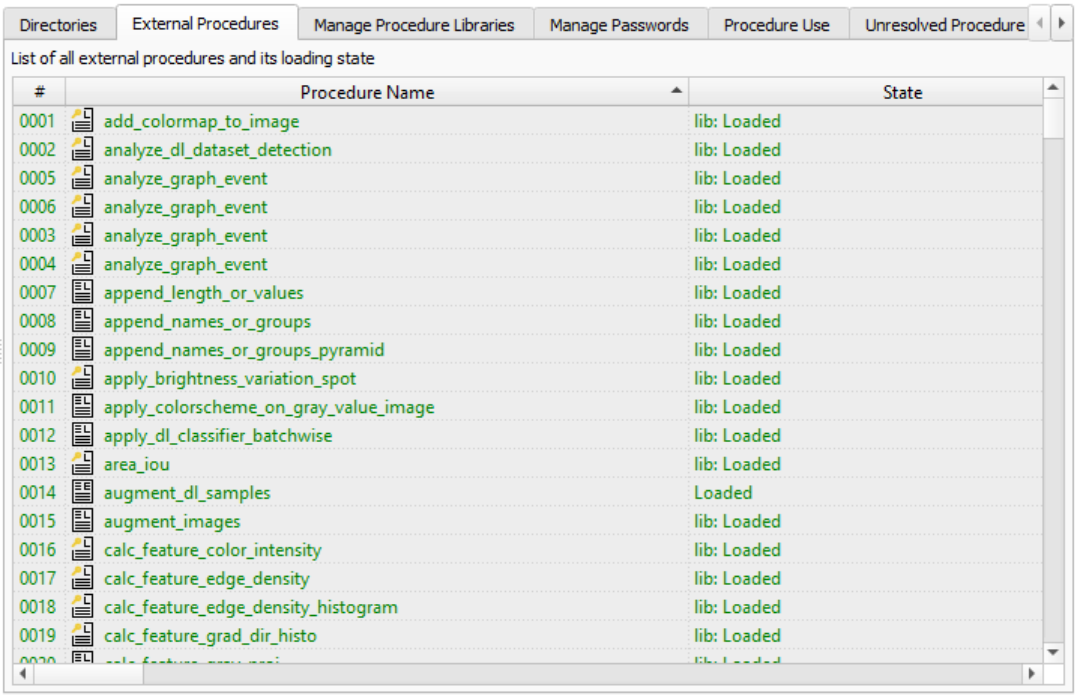


Figure 6.17: `Procedures` ▷ `External Procedures`.

Column	Meaning
`#`	Number of the external procedure.
`Activated`	The check box toggles the activation status of the corresponding procedure, i.e., whether the procedure can be resolved. It can be used to temporarily disable procedures, e.g., to make concealed procedures of the same name available.
`Procedure Name`	Name of the external procedure.
`State`	*Loaded*: The external procedure has been loaded successfully. It can be used in any HDevelop program. *Unloaded*: An error occurred while trying to load the external procedure, e.g., the file permissions are wrong or the external procedure file is corrupted.
`Search Directory`	Directory name from the tab card `Directories` where this procedure is found.
`Relative Path`	Path name of the external procedure relative to the search directory.
`Used by`	Usage counter and the names of the callers of this procedure.
`Modifications`	The number of modifications to the external procedure after it has been loaded.

Using the context menu, basic file system operations can be executed. You can copy, delete, or rename the selected procedure file in the file system.

☐ Procedures ▷ Manage Procedure Libraries

Use this tab card to

- create new libraries,
- delete existing libraries,
- add procedures to libraries,
- remove procedures from libraries.

The table on the left lists all libraries, the number of contained procedures, and the path name to the library file. The special entry “Procedures” contains all local procedures and all external procedures that are not contained in a library.

- Select a list entry to display the contained procedures.
- Double-click a library from the list to edit its properties (see below).

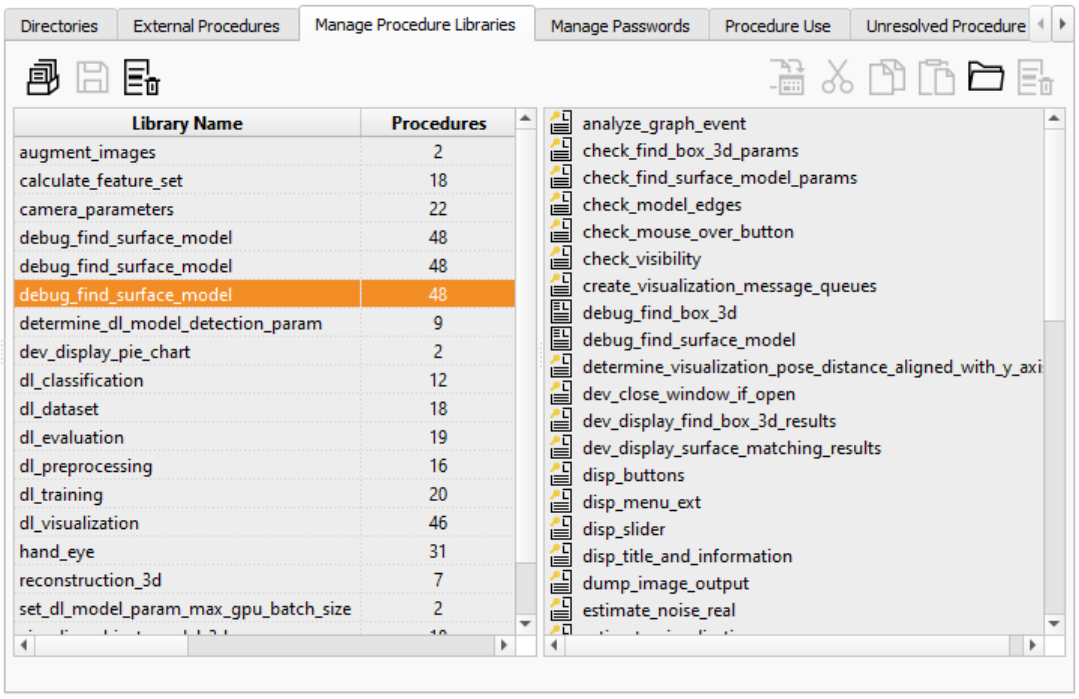


Figure 6.18: Procedures ▷ Manage Procedure Libraries.

The following buttons are available:

Create a new procedure library. See below.

Save changes in the selected procedure library. Modified libraries are marked with an asterisk (∗).

Delete the selected procedure library. After a confirmation, the procedure library file is irreversibly lost.

Cut the selected procedures, i.e., mark them for moving.

Copy the selected procedures, i.e., mark them for duplication.

Paste the marked procedures into the selected entry.

Add procedures from the file system to the selected entry.

Delete the selected procedures.

Drag-and-drop support:

In practice, it is much easier to drag procedures from one location to another than using the corresponding buttons. A simple drag-and-drop moves the selected procedures from one place to another. To copy the selected procedures, start to drag, then hold down Ctrl, and drop at the destination (this is visualized by a + icon).

Create a new procedure library (or edit existing libraries)

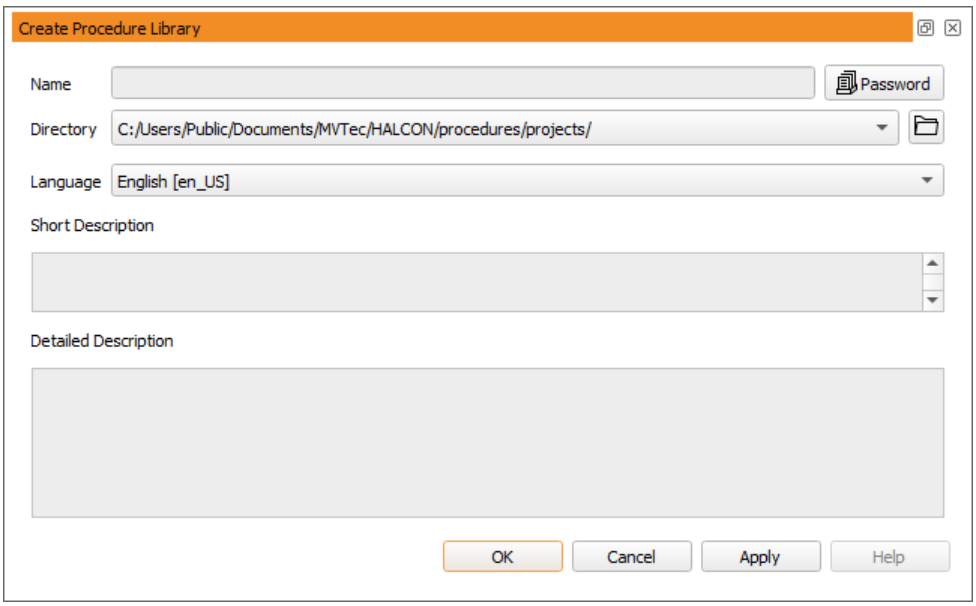


Figure 6.19: Create a new procedure library.

First, you have to specify the name of the new library and select a target directory. The configured directories can be selected from the drop-down list, or you can select an arbitrary directory using the "browse" button.

Click the `Password` button to protect the entire library with a password (see also section 6.4.9 on page 132).

The fields `Short Description` and `Detailed Description` allow you to describe the new library in multiple languages (select the desired language first, and then enter the description).

☐ `Procedures` ▷ `Manage Passwords`

Using this tab card, you can conveniently manage the status and passwords of all procedures (local and external). The procedures are divided into three categories (from left to right): Procedures without a password (unprotected), procedures for which the password has already been entered in this session (unlocked protected), and procedures that are locked with a password (locked protected). For an explanation of the different states, see section 5.6 on page 45.

Using the arrow buttons between the columns or the left and right cursor key, you can move the selected procedures to a different status. If you move procedures from the first to the second column, a password dialog is displayed which is described in section 6.4.9 on page 132. The same password is applied to all selected procedures.

If you move procedures from the second to the third column, the corresponding procedures will be locked, i.e., the procedure interfaces can no longer be edited and the procedure bodies will no longer be displayed. They can only be accessed if the correct password is supplied. This can either be done from this dialog by simply moving the corresponding procedures back to the middle column and entering the password. Or, you can unlock procedures individually from the program window as described in section 6.4.9 on page 133.

If you select multiple procedures in the third column and move them to the left, a password dialog appears to unlock the procedures. Only those procedures are moved (and thus unlocked) that match the supplied password. This way, you can conveniently edit a group of procedures that share the same password.

If all local procedures of the current program have been protected at once (by protecting the *main* procedure and enabling the corresponding option, see figure 6.51 on page 133, the local procedures will always move as a single group, even if only one local procedure is selected. The icon next to `Program Protection State:` (at the

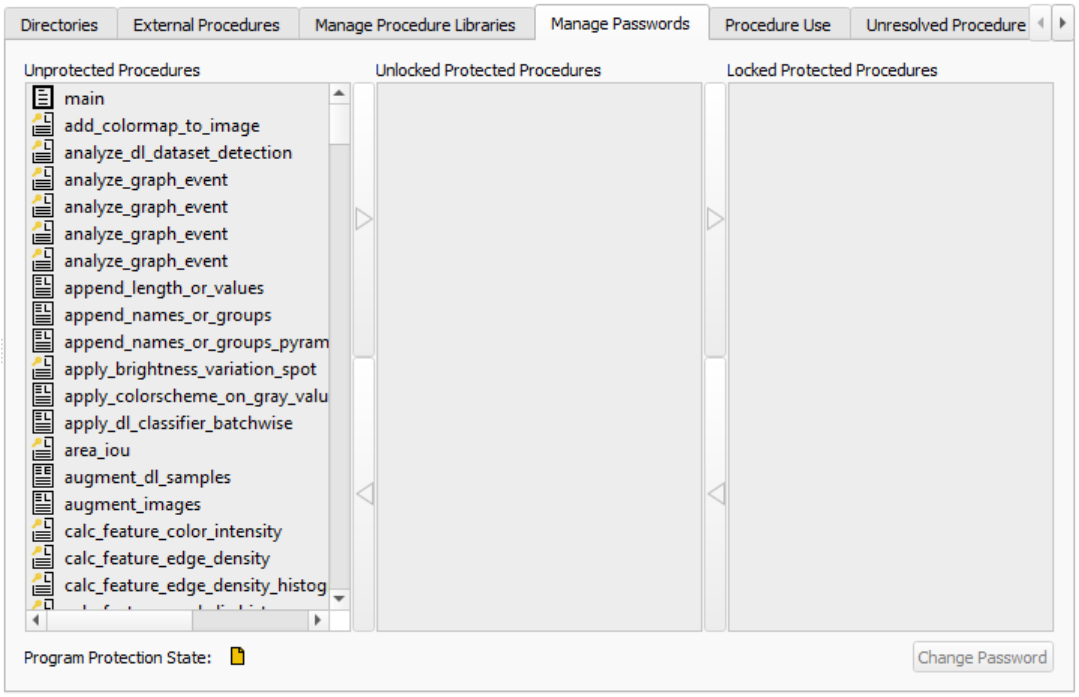


Figure 6.20: `Procedures` ▷ `Manage Passwords`.

bottom of the dialog) is marked with a lock if the entire program (main procedure and all local procedures) is protected.

The button `Change Password` is available if one or more procedures are selected in the middle column. It assigns a new password to the selected procedures, regardless if the previous passwords were different.

Please note, that password changes or moving procedures from or to the first column require the corresponding procedures to be saved. See `Save` (page 53) and `Save All` (page 55).

☐ `Procedures` ▷ `Procedure Use`

This tab card lists the usage of procedures grouped by their calling procedures. You can select a procedure and the type of used procedures (either local or external). For the main procedure you can also list the unused procedures. The tab card is displayed in figure 6.21.

☐ `Procedures` ▷ `Unresolved Procedure Calls`

This tab card helps you to find unresolved procedures in your current program. If the current program or the loaded procedures contain unresolved procedure calls, they are listed here along with the calling procedure name.

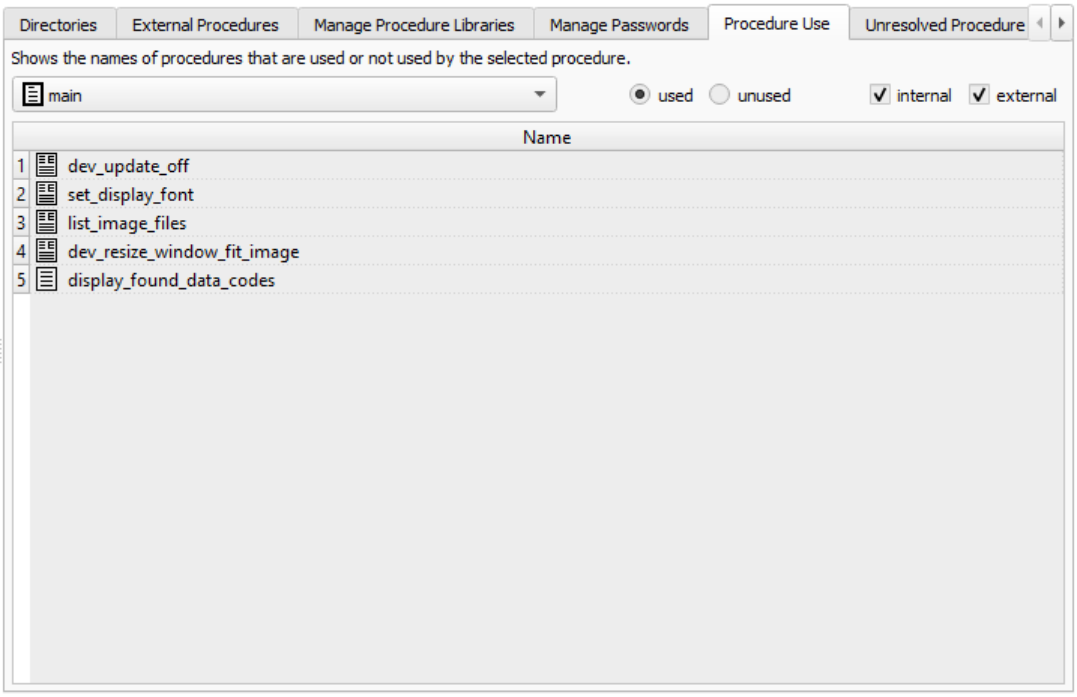


Figure 6.21: `Procedures` ▷ `Procedure Use`.

☐ `General Options` ▷ `General Options`

- `Show Start Dialog when starting HDevelop`

  If this option is checked, the Start Dialog is displayed automatically when opening HDevelop.

- `Select the behavior of pressing the [Return] key...`
  - `OK (Enter and execute)`

    Enter the operator in the program window and execute it.
  - `Enter`

    Enter the operator without executing it (the default behavior).

- `Display string values with special characters quoted`

  Special characters (like `\n` for a newline character) in string values are usually interpreted in the variable window and the variable inspect window. If this option is ticked, special characters are displayed verbatim, i.e., as they are entered. See table 8.1 on page 283 for a list of special characters.

- `Precision for displaying real values`

  This option sets the number of significant digits that are displayed in the variable window (page 142) and variable inspection windows, see “Inspecting Variables” on page 148.

- `Precision for displaying mouse position values`

  If set to a value greater than `0`, subpixel mouse positions are enabled. See `Position Precision` (page 97) for more information.

- `Default resize mode for new Graphics window`

  This option controls the default resize mode for newly opened graphics windows. See section 6.2.4.6 on page 90 for more information about the resize modes. The default resize mode applies only to graphics windows that are opened via menu, i.e., `Visualization` ▷ `Open Graphics Window...` or `Windows` ▷ `Open Graphics Window`.

- `Action when spinning the mouse wheel down`

  The mouse wheel is used for zooming in and out in windows with graphical content. This works consistently in the graphics window (page 158), the zoom window (page 168), the gray histogram window (page 173), the

feature histogram window (page 178), and the line profile window (page 180). The default setting is to zoom out when moving the mouse wheel down. Depending on this setting, the zoom direction can be reversed.

- `Default sort order in the Variable window`

  Variables can be sorted by name or by occurrence. See also variable window (page 145).

- `Return relative file paths when appropriate`

  If enabled, HDevelop will try to convert file names selected from file selection dialogs (page 190) to relative path names.

  HALCON uses several distinguished directories that act as the preferred search directories for certain operators. For example, `read_image` will at first search for image files in `%HALCONROOT%/images` when given a relative file path. For such operators HDevelop tries to turn the selected path into a path relative to the preferred search directory.

  If this fails, HDevelop tries to turn the selected path into a path relative to the current working directory. As a minimum requirement, the selected path and the current working directory must have the same device and toplevel directory. This will be useful in cases where, e.g., an image directory and a directory for HDevelop scripts are placed side by side within a common project directory.

  If no relative path can be established, HDevelop will use the absolute path.

- `Replace backslashes by slashes in pasted paths` (Windows only)

  This option determines the behavior when Windows path names are copied from an external program and pasted into HDevelop. If set to `Always`, backslashes will always be replaced by slashes. If set to `Never`, the path will be pasted unaltered. If set to `Ask`, HDevelop lets you choose the behavior each time a Windows path is pasted from the clipboard. For relative path names the replacement will only be performed if the path is pasted to a position where a path name is expected, e.g., an operator parameter with the corresponding semantic type.

- `Copy local procedures in Full Text Editor`

  This option determines the behavior when program lines containing calls to local procedures are copied and pasted. If set to `Always`, the corresponding local procedures will be copied as well unless they are already available. If set to `Never`, the corresponding local procedures will not be copied. Note that this will result in invalid calls if the copied lines are pasted into a program where the corresponding local procedures are not available. If set to `Ask`, HDevelop lets you choose the behavior each time you paste the lines into a program where the local procedures are not available.

- `Number of recent program files`

  The maximum number of recent program files that are displayed in the menu `File` ▷ `Recent Programs` (page 53) can be adjusted by altering this value.

- `Show recent program files`

  If you select the option `Only available`, the list of recent programs contains only programs that are currently available. This option is useful, if the list is likely to contain files from network drives that might be disconnected at times.

- `Show full path in main window title`

  If enabled, the full path of the current program is displayed in the title bar of the HDevelop window. Otherwise the file name is displayed.

- `Load the latest program automatically while starting HDevelop`

  If enabled, the most recently used program, i.e., the top entry under `File` ▷ `Recent Programs` (page 53), is loaded automatically when HDevelop starts. This behavior can be overridden from the command line using the switches `load_recent_prog`/`load_no_prog` (see page 351).

- `Save program and external procedures automatically before execution`

  If this option is enabled and you click any of the execution buttons (like `Run` or `Step Over`) and there are unsaved changes in the current program, the program will be saved before being executed. Use this option with care: You usually do not want to select this option if you are experimenting with a program, e.g., when trying out different parameter settings.

- `Encoding for saving HDevelop programs and procedures when the legacy HDevelop 5.0 - 9.0 file format (dev, dvp) is used`

HDevelop supports two different file types for programs and procedures (see section “File Types” on page 42). Programs and procedures in the default file format (.hdev and .hdvp, respectively) are always saved in `UTF-8` encoding. When saving programs and procedures in the legacy format (.dev and .dvp, respectively), the encoding depends on this setting. In order to exchange data with older versions of HDevelop (before HALCON 8), the encoding must be set to `Native`.

- `Default file format for saving HDevelop programs`

  Sets the default file format for new programs, see section “File Types” on page 42.

- `Default file format for saving HDevelop procedures`

  Sets the default file format for new external procedures, see section “File Types” on page 42.

- `Update Windows in Single Step Mode`

  Specifies the window update policy when stepping through the program. The different options are described in section 6.2.4.20 on page 96.

- `Maximum number of subthreads`

  Specifies the maximum number of subthreads that be started using the `par_start` qualifier (see section “Parallel Execution” on page 312).

- `Window open offset`

  Specifies an offset for the origin of floating windows. These offset values are added to the values for `Row` and `Column`, which are input parameters for some HALCON operators, like `dev_set_window_extents` or `dev_open_tool`. For example, `dev_open_tool('bookmarks_dialog', 0, 50, ...)` with offset values (50, 50) opens at position (50, 100). See figure 6.22 for an illustration of the window offset.

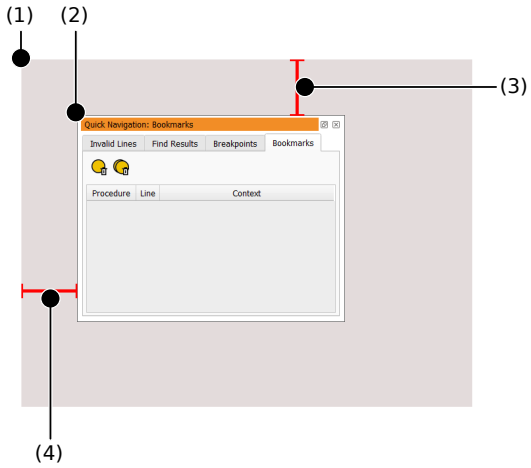


Figure 6.22: Window offset. By default, the reference point for the window position is the upper left corner of the desktop (1). (You can change this reference point with the setting `Origin of coordinates`, see below.) The origin of the HDevelop window is its upper left corner (2), its value is defined by the operator’s parameter values for `Row` and `Column`, here (0, 0). Specified `Window open offset` for row (3) and column (4).

- `Origin of coordinates`

  Sets the reference point for the origin of floating windows, opened via operators like `dev_open_tool` and `dev_open_window`.

  - `Main Display (Primary Screen)`

    Your primary screen, usually defined in the display settings of your operating system.

  - `Screen of HDevelop`

    The screen which displays the HDevelop main window (page 49).

  - `HDevelop`

    HDevelop’s main window (page 49).

  Please also note the setting `Window open offset`.

- Floating Windows behaviour

  Specifies the accessibility of floating windows. To apply the new settings, a restart of HDevelop is required.

  - Non minimizable and remains in foreground

    Floating windows stay in the foreground of HDevelop. They cannot be minimized, nor can they be accessed separately via taskbar.

  - Minimizable, does not remain in foreground

    Floating windows move to the background as soon as an other window gets the focus. They can be minimized and get a taskbar entry. Thus, you can access them via Alt+Tab under Windows. (Depending on your operating system, the shortcut for quickly switching between windows might differ.)

- Default layout uses Canvas

  In the default layout, the variable window is docked bottom left and the program window on the right. Depending on the setting of Use Canvas, either the canvas window or the graphics window is docked top left.

  - If Use Canvas is **enabled**, the **canvas window** is docked top left.
  - If Use Canvas is **disabled**, the **graphics window** is docked top left.

  You can restore the default layout any time via the menu entry Window ▷ Restore Default Layout.

☐ General Options ▷ Experienced User

- Show HALCON Low Level Error Message Boxes

  Low-level errors are normally invisible for the user because they are transformed into more comprehensive error messages or simply ignored. Activating this item generates a message box each time a low-level error occurs.

- Suppress error message dialogs within try-catch blocks

  HDevelop normally displays a dialog when a run-time error occurs (unless this has been changed in the tab Runtime Settings ▷ Runtime Settings). Per default, these dialogs are suppressed when the error occurs in a watched block of program lines (surrounded by try ... catch). In this case, the exception is caught by the program, i.e., the program continues at the corresponding catch ... endtry block.

  Deactivating this option will bring up a dialog even if the error occurs within a try-catch block.

  Error message dialogs within try-catch blocks are always suppressed in JIT-compiled procedures (see option below) regardless of this setting.

- Stop execution at invalid program lines

  If enabled, a dialog will be displayed if an invalid program line is reached, allowing to stop the program and edit the corresponding line, or to ignore it and continue execution. If disabled, invalid program lines will always be ignored.

- Suppress warnings for HALCON operators that are replaced by dev-operators

  Some operators are deprecated in HDevelop, and issue a warning message when selected in the operator window. Activating this option suppresses the warning message.

- Disable parameter detection for acquisition operators

  In the operator window, the parameter suggestions for the operators open_framegrabber, set_framegrabber_param, and get_framegrabber_param depend on the selected image acquisition interface. This behavior can be disabled by ticking the check box. See also Parameter Display on page 139.

- Ignore semantic type

  By default, the parameter suggestions in the operator window and the program window (with advanced autocompletion enabled) include only variable names that match the semantic type of the corresponding parameter. If Ignore semantic type is checked, these suggestions are extended so that they also include suggestions of variables with a different semantic type.

- Show memory usage

If this option is activated, the internal temporary memory usage of the last operator or procedure call is displayed in the status bar.

- `Execute procedures JIT-compiled`

  HDevelop supports just-in-time (JIT) compilation of procedures for optimized performance of HDevelop programs (see section "Just-In-Time Compilation" on page 45). The JIT compilation is globally enabled or disabled using this setting.

☐ `Visualization Settings` ▷ `Pen / LUT / Paint`

The tab cards in this category define the default visualization settings for graphics windows when HDevelop is started. See the description of "`Set Parameters`" on page 92. The dialog is displayed in figure 6.23.

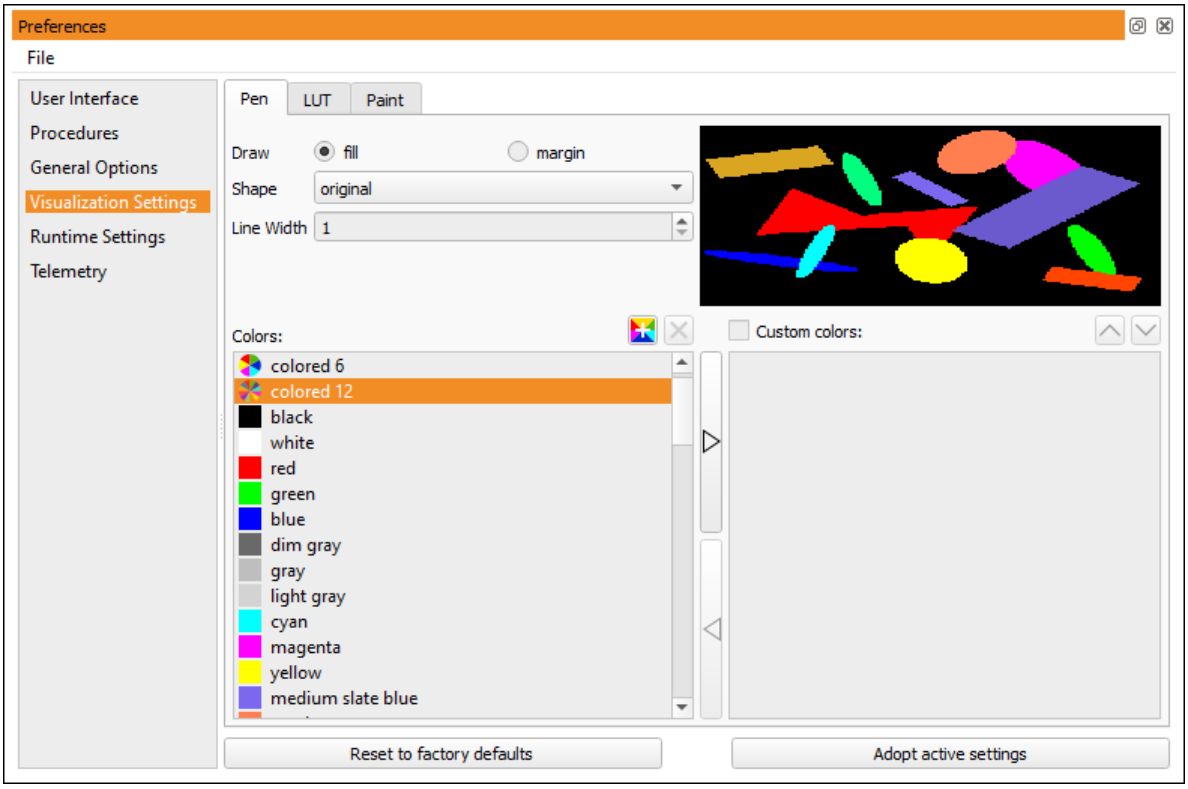


Figure 6.23: `Visualization Settings` ▷ `Visualization Parameters`.

`Reset to factory defaults` Remove the user-defined colors and custom color sets.

`Adopt active settings` Adopt the user-defined colors and custom color sets permanently. To define your own colors and color sets, use the `Visualization Parameters` dialog (see section "`Pen` settings" on page 93).

☐ `Runtime Settings` ▷ `Runtime Settings`

The settings in this category affect the runtime behavior of HDevelop. Please note that the runtime settings are not persistent between sessions. The runtime settings are reset to their default values, when a new HDevelop program is started with Menu `File` ▷ `New Program`.

- `Give Error`

  This check box specifies the behavior of HDevelop if an error occurs. If it is checked, HDevelop stops the program execution and displays an error message. Otherwise the error is ignored. See also: `dev_set_check`.

- `Show Processing Time`

  This check box indicates whether the required runtime of the last operator or procedure call should be displayed after the execution has stopped. It is a measurement of the needed time for the current operator or

procedure call (without output and other management tasks of HDevelop). Along with the required runtime, the name of the operator or procedure is displayed in the status bar at the bottom of the main window. Please note that the displayed runtime can vary considerably. This is caused by the inaccuracy of the operating system's time measurement procedure. See also: dev_update_time.

This option can also be toggled from the context menu of the status bar (see page 50).

- Update Program Counter

  This option concerns the display of the current position while running the program. The PC always indicates the line of the currently executing operator or procedure call or the line before the next operator or procedure call to execute. Using the PC in this way is time consuming. Therefore, you may suppress this option after your test phase or while running a program with a lot of "small" operators inside a loop. See also: dev_update_pc.

- Update Variables

  This check box concerns the execution of a program: Every variable (iconic and control) is updated by default in the variable window. This is very useful in the test phase, primarily to examine the values of control data, since iconic data is also displayed in the graphics window. If you want to save time while executing a program with many operator calls, you may suppress this output. Independent of the selected mode, the display of all variables will be updated after the program has stopped. See also: dev_update_var.

- Update Graphics Window

  This item concerns the output of iconic data in the graphics window after the execution of a HALCON operator. With the default settings, all iconic data computed in the run mode is displayed in the current graphics window. You may want to suppress this automatic output, e.g., because it slows down the performance or because the program handles the visualization itself. If the output is suppressed, you have the same behavior as exported C, C++, Visual Basic .NET, or C# code, where automatic output of data is not supported. See also: dev_update_window.

- Enable the context menu in the Graphics window

  If this option is activated, the context menu is available when clicking in a graphics window with the right mouse button. This behavior may be undesirable if a program provides user interaction with the mouse. See also: dev_set_preferences.

- Enable the mouse wheel in the Graphics window

  By default the mouse wheel is used to zoom in and out in the graphics window. If this interferes with a custom mouse handling, the mouse wheel may be disabled. This is desirable, e.g., if 3D objects are displayed in the graphics window and the zooming functionality is implemented with the help of 3D display operators. See also: dev_set_preferences.

- Enable tooltip showing coordinates and gray value at the mouse cursor position...

  If enabled, a tooltip will be displayed if the mouse cursor is in the graphics window and the Ctrl key is held down. The tooltip shows the pixel coordinates (row and column), and the gray value(s) at the mouse cursor position.

- Record Interactions

  If enabled, changes to the visualization settings will be recorded by adding the corresponding operator calls to the current program. See section "Record Interactions" on page 96.

☐ Runtime Settings ▷ Override Operator Behavior

This tab card allows you to modify the default behavior of stop and wait_seconds. Click the check box next to the corresponding operator and specify the modification.

stop: By default, the stop operator halts the program execution. It is mainly used to highlight or evaluate processing steps in a larger program. If you want to run such a program uninterrupted without altering the actual program code, you can make the stop operator behave like wait_seconds, i.e., perform a defined delay. The delay is specified in seconds.

wait_seconds: The operator wait_seconds is often used in situations where an intermediate program result is presented that would otherwise pass by too fast. Sometimes, you want to run such a program without any

delays, e.g., for performance measuring purposes. At other times, you would like to stretch the delays, e.g., for evaluation or presentation purposes. Therefore, you can redefine the actual delay of `wait_seconds`. The delay is specified in seconds.

Selecting `exactly` causes the specified delay for each `wait_seconds` instruction.

Selecting `with minimum` causes delays up to the specified duration. Calls to `wait_seconds` with a shorter duration will not be affected.

Selecting `with maximum` causes delays of at least the specified duration. Calls to `wait_seconds` with a longer duration will not be affected.

The following table shows the actual delay caused by different override settings:

Actual program call	with minimum(10s)	with maximum(10s)
`wait_seconds`(1)	1s	10s
`wait_seconds`(20)	10s	20s

☐ `Telemetry`

You can help us improve your HALCON user experience by anonymously sharing usage data with MVTec. To enable the telemetry feature, select `Share the following anonymous data with MVTec` and decide which information to send:

`System information` The sent data comprises geographical data (i.e., country and region), operating system and platform, monitor resolution, graphic card models, and the number of CPU cores. Note that this option cannot be excluded separately.

`HDevelop usage statistics` The sent data comprises system information about HDevelop (i.e., selected language, HALCON version and edition), workflow information (i.e., export and tool usage).

`HDevelop Help usage statistics` The sent data comprises workflow information inside HDevelop Help (i.e., used search terms and viewed chapters and pages).

`HALCON Library statistics` The sent data comprises the list of used HALCON modules.

You can inspect the raw data being sent by clicking `Get session data`. To export the data as a CSV file, click `Export session data`.

6.2.3 Menu `Execute`

In this menu item you find all necessary functions to execute an HDevelop program. In HDevelop, program execution is always continued at the top-most procedure call, which in most cases corresponds to the current procedure call. The procedure body displayed in the program window belongs to the current procedure.

6.2.3.1 `Run`

Synopsis: Execute the current program from the PC.

Shortcut: F5 ▷

The program line marked by the PC is the first line that is executed. All following program lines are going to be performed until the end of the current program. After the execution is finished, the main procedure becomes the current procedure. Note that a breakpoint, `stop` instruction, or runtime error may interrupt the execution of your program. If the HDevelop program waits for the user to draw something in the graphics window, a notification message is printed in the status bar. The program halts until the user finishes the draw operation and confirms this by clicking the right mouse button.

During the execution of operator or procedure calls the following special behavior occurs:

- You may initiate limited activities. For example, if you double-click variables in the variable window (page 142) they will be visualized; you may modify parameters for the graphics windows as described in the Menu Visualization; you may even modify the current procedure body.

  Note that all user interaction except Stop is disabled during program execution in case the latter was not started in the main procedure. HDevelop may be slow to react to your actions while the program is running. This is caused by the fact that HALCON reacts to user input only *between* calls to operators.

- A variable window update during runtime will only be performed if it has not been suppressed (see section "Runtime Settings ▷ Runtime Settings" on page 80). In any case, the values of all variables are shown in the variable window after the execution's termination.

While the program is running, the menu items Run, Step Over, Step Into, and Step Out (and the corresponding tool bar buttons) are grayed out, i.e., you cannot execute them.

You have the following possibilities to stop your HDevelop program:

- The program runs until the last operator or procedure call in the current program (i.e., the main procedure body) has been called. The PC is positioned behind this operator. This is the usual way a program terminates.
- The menu Menu Execute ▷ Stop (or the corresponding tool bar button) is pressed.
- A breakpoint is reached (see section "Program Window" on page 113). In this case, the last operator or procedure call that will be executed is the one *before* the breakpoint. In case of a breakpoint on a variable, the program stops *after* the program line that modified the corresponding variable (see also section "Breakpoints on Variables" on page 143).
- The entry Menu File ▷ Quit has been executed (see "Quit" on page 59).
- A runtime error has occurred. An input variable without a value or values outside a valid range might be typical reasons. In this case the PC remains in the line of the erroneous operator or procedure call.
- A stop instruction is executed. The PC remains on the line containing the stop instruction. Note that stop instructions inside locked procedures (see "Protected Procedures" on page 45) are obeyed. However, the code of the locked procedure will only be visible if the correct password is entered in the program window.

The procedure and procedure call in which program execution was stopped automatically become the current procedure and procedure call.

Clicking Run continues the program immediately if the current operator is wait_seconds.

6.2.3.2 Run To Insert Cursor

Synopsis: Run from PC to IC.

Shortcut: Shift+F5

The menu entry starts executing the program at the line next to the PC. The execution continues until the line before the IC is executed. Any breakpoints or stop instructions in between cause the program execution to be stopped.

Clicking Run To Insert Cursor continues the program immediately if the current operator is wait_seconds.

6.2.3.3 Step Over

Synopsis: Execute the next operator in the current program.

Shortcut: F6 ⧐

This entry enables you to run a program (even if it is not complete) step by step. HDevelop executes the operator or procedure call directly to the right of the PC.

After the operator or procedure call has terminated, all computed values are assigned to their respective variables that are named in the output parameter positions. Their graphical or textual representation in the variable window is also updated. If iconic data has been computed, you will see its presentation in the active graphics window.

In the status bar the runtime of the operator or procedure call is indicated (unless the time measurement has been deactivated).

The PC is then moved to the next operator or procedure call to execute. If the operators or procedure calls are specified in a sequential order, this is the textual successor. In case of control statements (e.g., `if` ... `endif` or `for` ... `endfor`), the PC is set *on* the end marker (e.g., `endif` or `endfor`) after the execution of the last operator or procedure call inside the statement's body. After `endfor` and `endwhile`, the PC is always set on the beginning of the loop. If a condition (like `if` or `while`) evaluates to `false`, the PC is set *behind* the end marker.

Suggestions in the menu Menu `Suggestions` are determined for the recently executed operator. Finally, HDevelop is available for further transactions. Any user input which has been made during execution is handled now.

Clicking `Step Over` continues the program immediately if the current operator is `wait_seconds`.

6.2.3.4 `Step Forward`

Synopsis: Execute the next line in the current program.

Shortcut: Shift+F6

This entry always steps forward in the current program. The difference to `Step Over` is apparent in loops: Only the first run of the loop is single-stepped. When the closing statement of the loop is reached, the remaining runs of the loop are executed without interruption, and the line following the loop is executed stepwise again.

Clicking `Step Forward` continues the program immediately if the current operator is `wait_seconds`.

6.2.3.5 `Step Into`

Synopsis: Step into HDevelop procedure.

Shortcut: F7

This entry allows you to step into procedure calls. Executing `Step Into` with the PC on a procedure call line causes the corresponding procedure and procedure call to become the current procedure and procedure call, respectively. The PC is set on the first executable program line in the new current procedure. `Step Into` has the same effect as `Step Over` if the program line to be executed is not a procedure call.

6.2.3.6 `Step Out`

Synopsis: Step out of HDevelop procedure.

Shortcut: F8

This entry steps out of the current procedure call. Program execution is continued until the first executable program line after the previous procedure call in the calling procedure is reached. The previous calling procedure becomes the current procedure. If the current procedure is the main procedure, the behavior is the same as `Run`.

Clicking `Step Out` continues the program immediately if the current operator is `wait_seconds`.

6.2.3.7 `Stop`

Synopsis: Stop program execution.

Shortcut: F9

HDevelop continues processing until the current operator has completed its computations. This may take a long time if the operator is taking a lot of time to execute. Some operators can be interrupted, in which case their intermediate results are discarded. In particular, operators that support timeouts and operators that display a progress bar can be interrupted.

The procedure and procedure call in which the program execution was stopped becomes the current procedure and procedure call, respectively. If the currently executed procedure was JIT-compiled, its processing is aborted.

After interrupting a program you may continue it by selecting Run or Step Over, or Step Into if the next program line is a procedure call.

You may also edit the program before continuing it (e.g., by parameter modification, by exchanging operators with alternatives, or by inserting additional operators).

Clicking Stop stops the program immediately if the current operator is wait_seconds.

6.2.3.8 Stop After Procedure

Synopsis: Stop program execution after current procedure.

Shortcut: Shift+F9 ☐

Same as Stop (page 84) but finishes the current procedure (if any) first.

6.2.3.9 Attach To Process...

Synopsis: Debug an external HDevEngine application.

Checks for: Unsaved changes (page 191)

See chapter "Remote Debugging" on page 327 for details.

6.2.3.10 Stop Debugging

Synopsis: Stop debugging an external HDevEngine application.

6.2.3.11 Thread View / Call Stack

Synopsis: Visualize the calling hierarchy.

Shortcut: Ctrl+Shift+O,C or Ctrl+Shift+O,Ctrl+Shift+C 

This window displays information about the current execution status of the program.

The Thread View in the upper half displays information about all the threads that have been started (see section "Inspecting Threads" on page 316).

The Call Stack in the lower half contains a list of the names of all procedures that are currently called on HDevelop's internal call stack. The top-most procedure call belongs to the most recently called procedure, the bottom-most procedure call always belongs to the main procedure. Clicking on a procedure call in the dialog makes the selected procedure call the current procedure call and thus the procedure belonging to the selected procedure call the current procedure.

When you click on a procedure call that belongs to a locked procedure (see "Protected Procedures" on page 45), you can only see the procedure body if you enter the correct password into the program window.

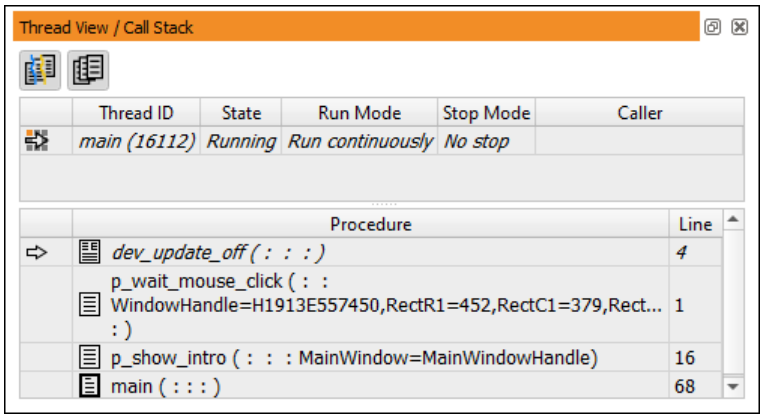


Figure 6.24: `Thread View / Call Stack`.

6.2.3.12 `Set/Clear Breakpoint`

Synopsis: Set or clear breakpoint(s) at the selected program lines.

Shortcut: F10

This menu item sets or clears breakpoints on the lines that are currently selected in the program. In most cases, however, it is more convenient to set individual breakpoints by holding the Ctrl key and clicking in the left column of the program window as described in "Program Counter, Insert Cursor, and Breakpoints" on page 119.

6.2.3.13 `Activate/Deactivate Breakpoint`

Synopsis: Deactivate or activate breakpoints at the selected program lines.

Shortcut: Shift+F10

This menu item toggles the state of the breakpoints in the selected program lines. It does not remove the breakpoints but rather disables them temporarily and re-enables them later. This might be useful to switch the run mode of the current program between continuous mode and debug mode. If you want to deactivate/activate the breakpoints in many procedures at once, consider using the tab `Breakpoints` of the quick navigation window which is discussed in section 6.2.3.15.

6.2.3.14 `Clear All Breakpoints`

Synopsis: Clear all breakpoints in the current program.

Both line-based breakpoints as well as breakpoints on variables are cleared.

6.2.3.15 `Manage Breakpoints`

Synopsis: Manage breakpoints

Shortcut: Ctrl+Shift+O,F10 or Ctrl+Shift+O,Ctrl+Shift+F10

The quick navigation window (see section "`Open Quick Navigation`" on page 108) contains a tab card to manage breakpoints. See section "Breakpoints on Variables" on page 143 for information about breakpoints on variables and section "Program Window" on page 113 for line-based breakpoints.

The tab card lists all variables or program lines marked with a breakpoint (see figure 6.25). Information about the breakpoints is given in the column `Context`: Breakpoints on variables (1) list the variable name, while line-based breakpoints (2) list the program line (its line number is given in the column `Line`).

Clicking the entry of a line-based breakpoint focuses the corresponding program line in the active program window. If the selected procedure is already displayed in a different program window tab, the corresponding tab is activated. Otherwise, the current view switches to the selected procedure.

- Clear the breakpoints of the selected program lines and immediately remove the corresponding entries from the list. This action can also be triggered from the context menu, or by pressing Del.
- Clear all breakpoints and remove all entries from the list, leaving you with an empty window.
- Activate all breakpoints at once. This action can also be triggered from the context menu.
- Deactivate all breakpoints at once. This action can also be triggered from the context menu.

Individual breakpoints may be activated/deactivated by clicking the check boxes in front of the entries.

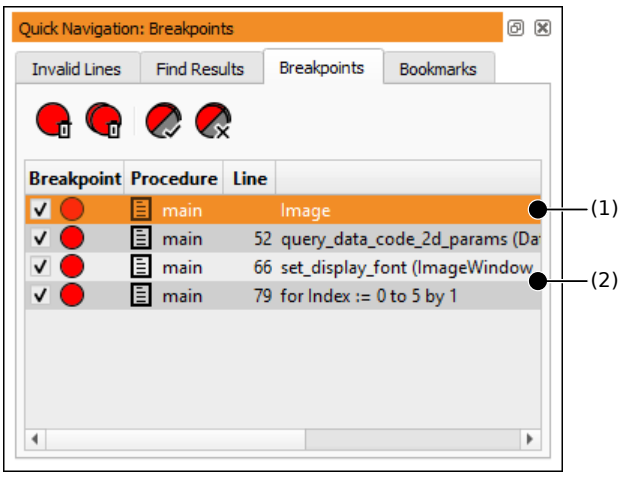


Figure 6.25: Managing breakpoints in the quick navigation window: (1) breakpoint on variable, (2) line-based breakpoints.

6.2.3.16 Reset Program Execution

Synopsis: Reset program to its initial state.

Shortcut: F2

The main procedure becomes the current procedure and the call stack is cleared of all procedure calls except the main procedure call. The latter is reset, i.e., all variables have undefined values and the PC is set to the first executable line of the main procedure. The breakpoints, however, are not cleared. All graphics windows except one are closed, and the remaining graphics window is cleared. This menu item is useful for testing and debugging programs.

6.2.3.17 Reset Procedure Execution

Synopsis: Reset procedure execution.

Shortcut: Shift+F2

The variables of the current procedure are reset, i.e., all variables have undefined values, and the PC is set to the first executable line of the current procedure. This menu item is useful for debugging procedures without affecting the calling procedures.

6.2.3.18 Abort Procedure Execution

Synopsis: Abort execution of current procedure.

Shortcut: Shift+F8

All variables of the current procedure are reset. The PC is set back to the line in the calling procedure from which the current procedure was called. The calling procedure becomes the current procedure.

6.2.3.19 Activate Profiler

Synopsis: Activate or deactivate the profiler.

Shortcut: Ctrl+Shift+F,F or Ctrl+Shift+F,Ctrl+Shift+F

The profiler is described in section "Profiler" on page 134.

6.2.3.20 Profiler Display

The entries in this submenu select what is being displayed in the profiler section of the program window.

The profiler is described in section "Profiler" on page 134.

6.2.3.21 Reset Profiler

Synopsis: Reset profiler data.

Shortcut: Ctrl+Shift+F,R or Ctrl+Shift+F,Ctrl+Shift+R

The profiler is described in section "Profiler" on page 134.

6.2.3.22 Show Runtime Statistics

Synopsis: Evaluate runtime statistics.

Shortcut: Ctrl+Shift+F,S or Ctrl+Shift+F,Ctrl+Shift+S

The profiler is described in section "Profiler" on page 134.

6.2.4 Menu Visualization

Via this menu, you can open or close graphics windows and clear their displays. Furthermore, you may specify their output behavior during runtime. Most functions are also available from the context menu of the graphics window.

6.2.4.1 Open Graphics Window...

Synopsis: Open a new graphics window.

Shortcut: Ctrl+Shift+G,O or Ctrl+Shift+G,Ctrl+Shift+O

See also: dev_open_window.

When selecting this menu entry, a dialog window pops up. Here, you may specify some graphics window attributes. The dialog is displayed in figure 6.26. The position, size and background color of the new graphics window can be specified. For example, it is more convenient to have a white background while building graphics for slides or reports (see the HALCON operator dump_window). If the window height and width are set to -1, the window size is set by HDevelop. It is taken from the persistent preferences of HDevelop (usually the size of the last graphics window in the previous HDevelop session). The handling of graphics windows is described in more detail in the section "Graphics Window" on page 158.

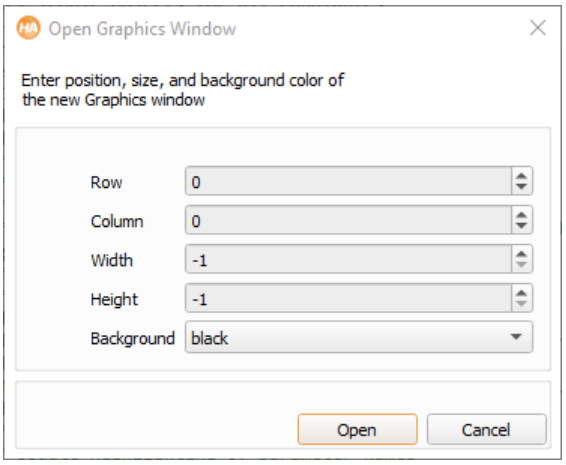


Figure 6.26: Specifying the parameters of the new graphics window.

6.2.4.2 Clear Graphics Window

Synopsis: Clear active graphics window.

Shortcut: Ctrl+Shift+G,Del or Ctrl+Shift+G,Ctrl+Shift+Del

See also: dev_clear_window.

The history (previously displayed objects) of the window is also cleared.

6.2.4.3 Close Graphics Window

Synopsis: Close active graphics window.

Shortcut: Ctrl+Shift+G,Q or Ctrl+Shift+G,Ctrl+Shift+Q

See also: dev_close_window.

6.2.4.4 Display

Synopsis: Select iconic variable to be displayed in active graphics window.

See also: dev_display.

This submenu lists all instantiated iconic variables for quick selection. The submenu is split in three parts (from top to bottom): images, regions, and XLDs.

6.2.4.5 Window Size

Synopsis: Set window size of active graphics window. (Only available for floating windows.)

See also: dev_set_window_extents.

This submenu offers a list of fixed percentages to resize the graphics window with respect to the size of the most recently displayed image.

The entries Double and Half change the size of the graphics window to half and double its current window size, respectively, independent of the size of the displayed image.

The entry Aspect Ratio 1:1 scales down the current window size, so that the aspect ratio of the displayed image is maintained.

6.2.4.6 Image Size

Synopsis: Adapt the image display in the active graphics window.

This submenu offers a list of zoom options to scale the image and to control its resizing behavior.

The following values for image scaling are available:

`Fit`
: Scales the image to completely fill the graphics window.

`400%, 200%, ...`
: A list of fixed percentages scales the image with respect to its natural size.

`Double`
: Double the current image size.

`Half`
: Half the current image size.

`Aspect Ratio 1:1`
: Scales down the image size, so that its aspect ratio is maintained.

You may also change the size of the graphics window, e.g., by "gripping" the window border with the mouse. Then you can resize the window by dragging its border. After this size modification the window content is redisplayed, depending on the chosen resize mode. The following modes to control the resizing behavior of the image are available:

`Full Stretch`
: The visible image part remains constant. The zoom levels are adapted to fully fit the previous view into the new window size, i.e., the aspect ratio of that image part might be skewed.

  This is the classic mode. Graphics windows that are opened via operator `dev_open_window` will be opened as floating windows using the "Full Stretch" mode for maximum backwards compatibility.

`Keep Aspect Ratio`
: The zoom level is changed to preserve the previous aspect ratio. The visible image part may change in one axis as a result of the resize operation.

  This is the default mode. You can change the default mode under `Edit` ▷ `Preferences` ▷ `General Options`.

`No Stretch`
: The zoom level remains fixed. Thus, resizing the window will show a correspondingly larger or smaller portion of the image.

6.2.4.7 Colored

Synopsis: Disambiguate the display of regions and XLDs by using multiple colors.

See also: `dev_set_colored`.

This is an easy way to display multiple regions or XLDs. Each region is displayed in a different color, where the number of different colors is specified in the submenu. You can choose between 3, 6 and 12 colors. If all regions are still displayed with one color, you have to use the operator `connection` beforehand. You can check this also with the operator `count_obj`. The default setting is to use 12 colors.

6.2.4.8 Color

Synopsis: Display regions, XLDs, and text in a specific color.

See also: dev_set_color.

This item allows you to choose a color for displaying segmentation results (regions and XLDs), text created with write_string, and general line drawings (e.g., 3D plots, contour lines, and bar charts). The number of colors that are available in the submenu depends on the graphics display (i.e., the number of bits used for displaying). After selecting a color, the previously displayed region or XLD object will be redisplayed with this color if the menu entry Apply Changes Immediately is checked.

The default color is red.

6.2.4.9 Draw

Synopsis: Draw type of regions.

See also: dev_set_draw.

Here, you can select a visualization mode to display regions. It can either be *filled* (menu entry fill) or outlined (menu entry margin). If set to margin, the line thickness of the displayed regions is specified using the menu item Line Width.

6.2.4.10 Line Width

Synopsis: Line width used for the display of lines in active graphics window.

See also: dev_set_line_width.

Here, you determine the line width for painting XLDs, borders of regions, or other types of lines. You can select between a wide range of widths using the submenu.

6.2.4.11 Shape

Synopsis: Specify representation shape for regions.

See also: dev_set_shape.

Here, you specify the representation shape for regions. You can display not only the region's original shape but also its enclosing rectangle or its enclosing circle.

6.2.4.12 Lut

Synopsis: Specify look-up table for gray value mapping.

See also: dev_set_lut.

This menu item activates different look-up tables, which can be used to display gray value images and color images in different intensities and colors. In the case of a true color display the image has to be redisplayed due to the missing support of a look-up table in the graphics hardware. For color images only the gray look-up tables can be used, which change each channel (separately) with the same table.

6.2.4.13 Paint

Synopsis: Specify image visualization.

See also: dev_set_paint.

This menu item defines the mode to display gray value images. For more information see the menu item Set Parameters below.

6.2.4.14 `Add To Canvas/Float From Canvas`

- Add To Canvas
  Add the selected graphics window to the canvas (page 109).
- Float From Canvas
  Remove the selected graphics window from the canvas (page 109) again.

6.2.4.15 `Apply Changes Immediately`

Synopsis: Update behavior of visualization changes in active graphics window.

If this menu entry is checked, any changes to the visualization settings are applied immediately to the active graphics window. Otherwise, the changes are deferred until the next object is displayed in the active graphics window.

6.2.4.16 `Set Parameters`

Synopsis: Set visualization parameters of the active graphics window with interactive preview.

Shortcut: Ctrl+Shift+G,P or Ctrl+Shift+G,Ctrl+Shift+P

This menu entry opens the window `Visualization Parameters` which allows convenient access to the visualization settings of the active graphics window. Most of the settings are also available as individual menu entries in the menu `Visualization`, but some more advanced settings are only provided in this window. Furthermore, an interactive preview is provided, which visualizes the current settings.

`Select Graphics Window` **(only with multiple graphics windows)** Each graphics window keeps its own private set of visualization settings. When multiple graphics windows are opened in the current session, you can switch between the settings of the different graphics windows by selecting the corresponding window handle magic value.

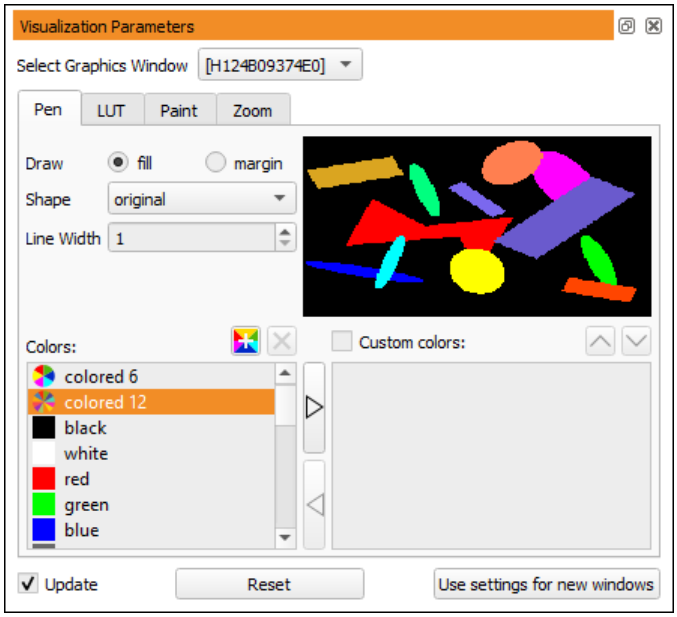


Figure 6.27: `Visualization Parameters` with multiple graphics windows.

`Update` This check box corresponds to the setting of Menu `Visualization` ▷ `Apply Changes Immediately`. If it is checked, every change of a parameter will immediately lead to a redisplay of the image, regions, or XLD in the graphics window. Otherwise, the parameters become active for the next display of an object (double-click on an icon or execution of an operator).

`Reset` Reset to the visualization settings defined in the `Preferences` (page 67).

`Use settings for new windows` Make the current settings also the default settings for new graphics windows.

☐ Pen settings

The pen settings specify the drawing mode (filled or outlined), the shape, the line width, and the color(s) of regions and XLDs.

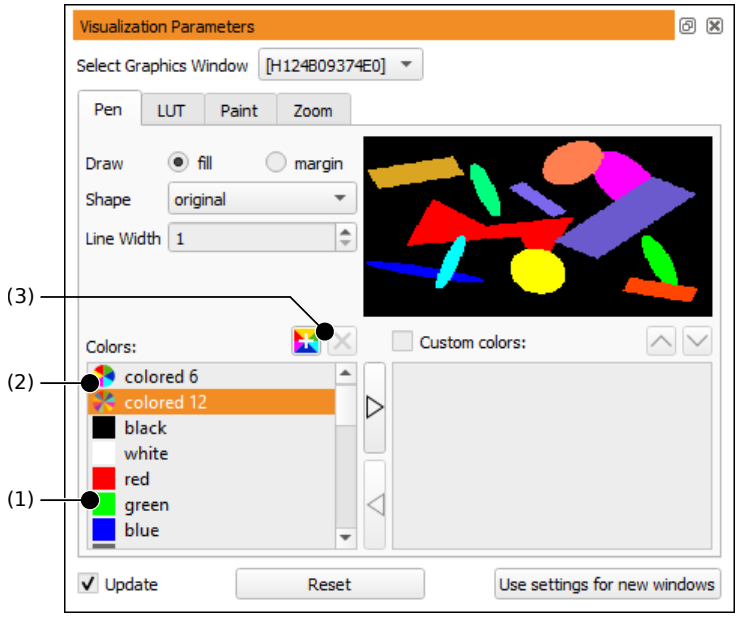


Figure 6.28: `Pen` settings: (1) predefined colors, (2) color sets, (3) add/remove user-defined color .

For regions the draw mode can set to *filled* (item `fill`) or *outlined* (item `margin`).

The parameter `Shape` (default is `original`) specifies the presentation shape for regions. You can display not only the region's original shape but also its enclosing rectangle or its enclosing circle, etc.

The setting of `Line Width` applies to XLDs and regions in draw mode `margin`.

The color of the regions/XLDs can be set to one of the named colors. If you select one of the color sets, each region or XLD will be displayed in an alternating color from a set of 3, 6, or 12 colors. This visualizes the connectivity of different regions in the graphics window.

Click the "add" button to define additional colors. Colors are specified by hue, saturation, value, or by giving the red, green and blue values (see figure 6.29). User-defined colors are added to the list of named colors. They can be removed by selecting them and clicking the "remove" button.

You can also define your own custom color sets: Add selected colors to the list of custom colors by clicking the "right" button, or remove them by clicking the "left" button. The check box next to `Custom colors:` specifies if the custom color set is used.

User-defined colors and custom color sets are discarded when you quit HDevelop. If you want them to be permanent, you will have to define them in the preferences dialog (see section "`Visualization Settings` ▷ `Pen` / `LUT` / `Paint`" on page 80).

The pen settings are also available from the corresponding menu entries in the menu `Visualization`. A description of the functionality is provided there. The preview shows the current settings, which is helpful if the active graphics window does not contain any regions or XLDs.

- "`Draw`", see also page 91
- "`Colored`", see also page 90
- "`Color`", see also page 91

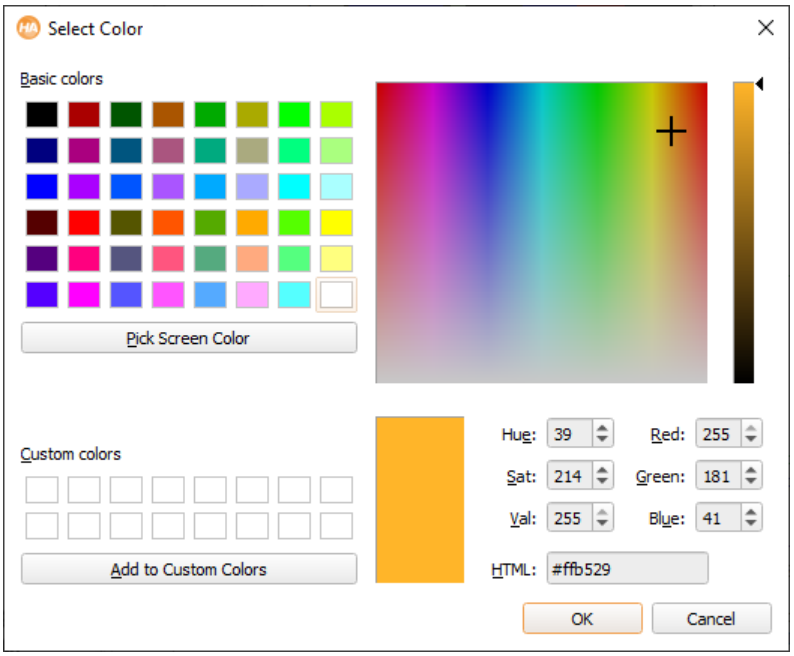


Figure 6.29: Defining a new color.

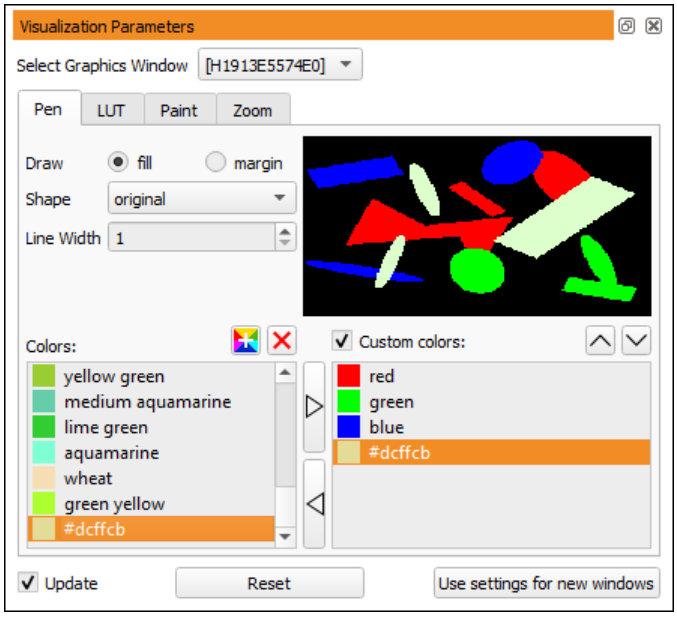


Figure 6.30: Custom color set.

- “Shape”, see also page 91
- “Line Width”, see also page 91

☐ LUT settings

Using LUT you are able to load different look-up tables for visualization. With the help of a false color presentation you often get a better impression of the gray values of an image. In the case of a true color display, the image has to be redisplayed due to the missing support of a look-up table in the graphics hardware. For color images only the gray look-up tables can be used, which change each channel (separately) with the same table.

See the description of the menu entry “Lut” on page 91.

☐ Paint settings

Here, you can select between two graphical image presentations. In the default mode the image is displayed unmodified. In the 3d_plot mode, the gray values of the image are taken as height information: The greater the gray value, the higher the resulting image point. See figure 6.32 on page 96 for an illustration of the different modes. Further information can be found at the description of the operators dev_set_paint and set_paint.

default Display the image unmodified.

3d_plot Display a 3D plot using OpenGL which can interactively be modified in the graphics window. This mode can also be enabled from the tool bar of the graphics window. See page 158.

☐ Zoom settings

See also: dev_set_part.

As opposed to the mouse-based zoom functionality that is available in the tool bar of the graphics window, the tab card Zoom is parameterized. You can specify the bounding box of the visible area of an image, or set the center position.

This tab card specifies which part of an image, region, XLD, or other graphic item is going to be displayed. The four text fields of Set part specify the coordinate system. Upper Left Corner defines the pixel which will be displayed at the upper left corner of the window. Lower Right Corner defines the pixel which will be displayed at the lower right side of the window.

Below the coordinates of the rectangle, you can specify its center.

The buttons Zoom Out and Zoom In activate a zooming with factor 0.5 or 2, respectively.

To get the image's full view back on your graphics window, you simply click the button Reset.

The button Aspect adjusts the parameters so that the aspect ratio of the image is maintained.

6.2.4.17 Reset Parameters

Synopsis: Reset the visualization parameters of the active graphics window to the default settings.

Here, the display parameters of the active graphics window are set to their initial state (as defined in the preferences, see page 67). The only exception is the size of the window. To clear the history of previously displayed objects, use Clear Graphics Window. To set the size, use Window Size.

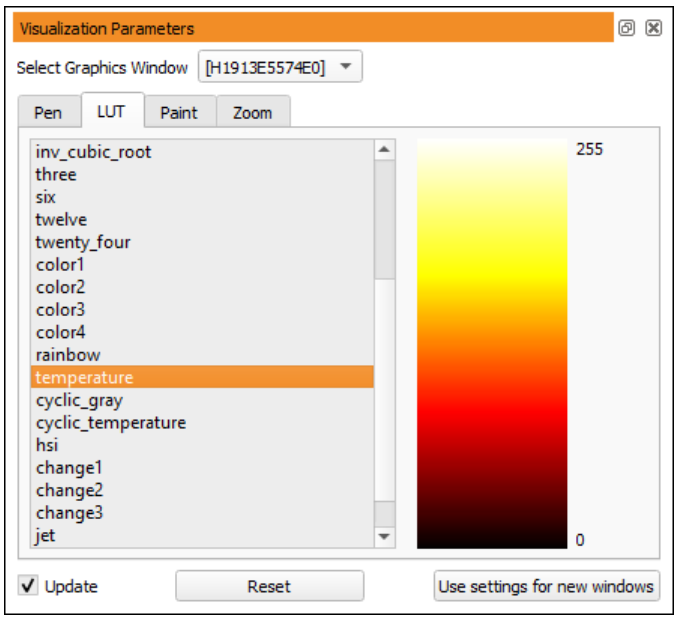


Figure 6.31: Visualization Parameters: LUT settings.

6.2.4.18 `Record Interactions`

Synopsis: Record visualization changes in the current program at the IC.

Shortcut: Ctrl+I

If this entry is enabled, any subsequent changes to the visualization settings will insert the corresponding operator call(s) into the current program. Unlike the entry `Insert Code...` which dumps the current settings in one block, using `Record Interactions` you can selectively adjust the settings you want to replay in your program.

This includes opening, activating, and closing graphics windows, displaying or clearing iconic objects, adjusting the image or window size from the menu or the tool bar, as well as adjusting the draw colors, draw mode, line width, region shape, LUT, and paint mode.

6.2.4.19 `Insert Code...`

Synopsis: Insert current visualization settings into the current program at the IC.

Shortcut: Ctrl+Shift+G,I or Ctrl+Shift+G,Ctrl+Shift+I

Using this entry you can insert code into the current program that will restore the current state of the graphics window. The parts to be restored are selected from the dialog shown in figure 6.34.

`Iconic objects from the graphics stack`**:** Insert operator calls to display all iconic objects that are currently visible in the active graphics window.

`Visualization parameters`**:** Insert operator calls to restore the draw colors, draw mode, line width, region shape, LUT, and paint mode.

`Window geometry`**:** Insert operator calls to restore the window size and the visible part.

6.2.4.20 `Update Window`

Synopsis: Specify the update behavior of the active graphics window.

If the menu entry `In Run Mode` is checked, every object (image, region, or XLD) is displayed in the active graphics window during program execution. Otherwise, the active graphics window is not updated.

The submenu `In Single Step Mode` specifies the update behavior when stepping through the program:

Setting	Behavior
`Always`	display iconic results
`Never`	do not display iconic results
`Clear And Display`	clear graphics window before displaying iconic results
`Last Image And Display`	keep displaying the latest image and additional iconic results
`As in Run Mode`	same behavior as run mode setting

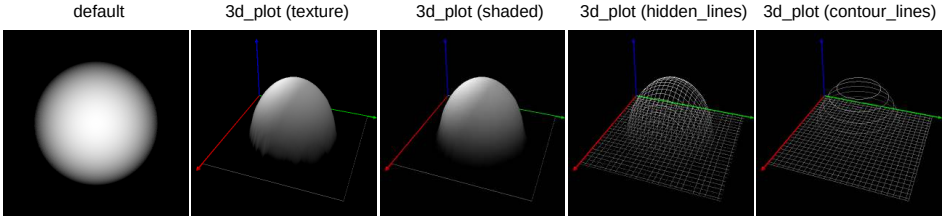


Figure 6.32: Comparison of the different paint settings.

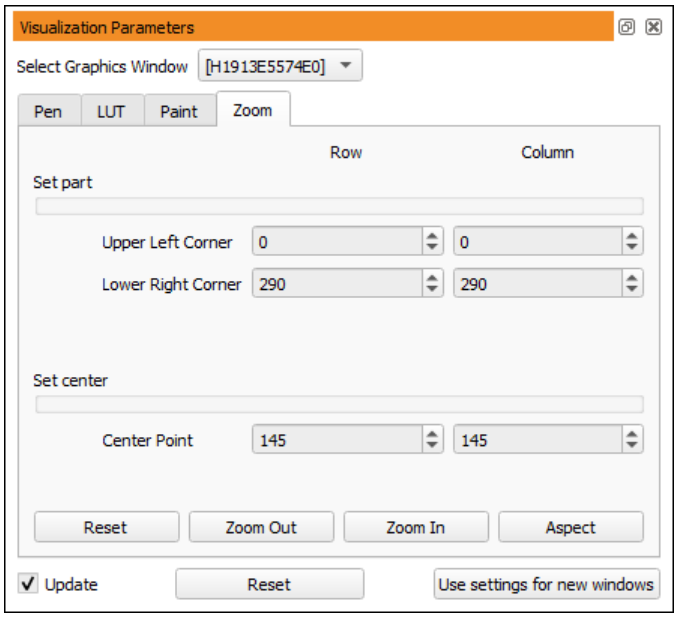


Figure 6.33: `Visualization Parameters`: Zoom settings.

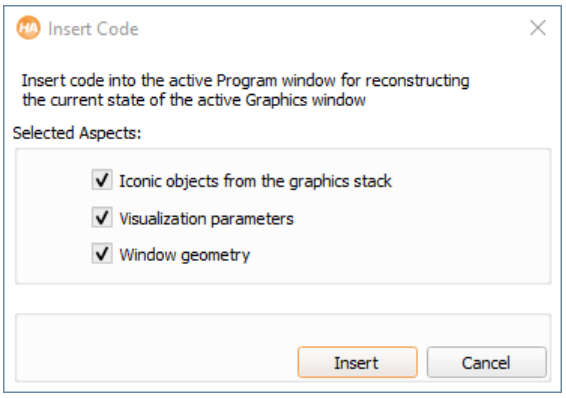


Figure 6.34: `Insert Code...`.

6.2.4.21 `Position Precision`

Synopsis: Select the precision of subpixel mouse positions.

By default, mouse positions are displayed as integers (precision 0), where the upper left image pixel is displayed as `0, 0`. Increasing the precision results in mouse positions being reported as subpixel-precise positions. Please note that when subpixel mouse positions are enabled, the position `0.0, 0.0` refers to the *center* of the upper left pixel, i.e., the upper left *edge* of the image is displayed as `-0.5, -0.5`.

6.2.4.22 `Tools`

This menu contains several visualization tools.

☐ `Zoom Window`

Synopsis: Open zoom window for image details and pixel inspection.

Shortcut: Ctrl+Shift+O,Z or Ctrl+Shift+O,Ctrl+Shift+Z

The zoom window is described in section “Zoom Window” on page 168.

☐ `New Zoom Window`

Synopsis: Open additional zoom window.

See section “Zoom Window” on page 168.

☐ `Gray Histogram`

Synopsis: Display gray value histogram of active graphics window.

Shortcut: Ctrl+Shift+O,H or Ctrl+Shift+O,Ctrl+Shift+H 

Selecting this entry opens a sophisticated tool for the inspection of gray value histograms, which can also be used to select thresholds interactively and to set the range of displayed gray values dynamically. It is described in section “Gray Histogram Window” on page 173.

☐ `Feature Histogram`

Synopsis: Interactive inspection of feature histograms.

Shortcut: Ctrl+Shift+O,F or Ctrl+Shift+O,Ctrl+Shift+F 

This menu item opens a sophisticated tool for the inspection of feature histograms. It is described in section “Feature Histogram Window” on page 178.

☐ `Feature Inspection`

Synopsis: Inspection of shape and gray value features of individual regions.

Shortcut: Ctrl+Shift+O,I or Ctrl+Shift+O,Ctrl+Shift+I 

This window provides a tool for the convenient inspection of shape and gray value features of individual regions. It is described in section “Feature Inspection Window” on page 179.

☐ `Line Profile`

Synopsis: Display line profile of active graphics window.

Shortcut: Ctrl+Shift+O,L or Ctrl+Shift+O,Ctrl+Shift+L 

Selecting this entry opens a useful tool for the detailed inspection of the gray-value profile of a line or circular arc. Using the line profile is helpful in particular for optimizing edge detection in the Measure Assistant (page 249) or when checking the focus of your camera. The line profile is described in section “Line Profile Window” on page 180.

☐ `OCR Training File Browser`

Synopsis: Browse OCR training files.

Shortcut: Ctrl+Shift+O,T or Ctrl+Shift+O,Ctrl+Shift+T 

Selecting this entry opens a useful tool for inspecting and modifying training files. Using the OCR Training File Browser is especially useful to eliminate errors made during the teaching process by checking if the samples are assigned the correct classes. This tool can be used within any OCR application and also in combination with the OCR Assistant (page 262). The OCR Training File Browser is described in section “OCR Training File Browser” on page 185.

6.2.4.23 Save Window...

Synopsis: Save the contents of the active graphics window to an image file.

Shortcut: Ctrl+Shift+G,S or Ctrl+Shift+G,Ctrl+Shift+S

See also: dump_window.

The graphics window is saved 'as is' (including displayed regions and XLDs). A file dialog pops up. Select the destination directory, enter a file name, and select the output format (TIFF, BMP, JPEG, PNG, or PostScript). Afterwards, click Save to actually save the image file, or Cancel to abort.

6.2.5 Menu Procedures

The menu Procedures contains all functionality that is needed to create, modify, copy, or delete HDevelop procedures. To save procedures, refer to the Menu File menu (page 51).

6.2.5.1 Create New Procedure

Synopsis: Create a new internal or external procedure.

Shortcut: Ctrl+Shift+P,C or Ctrl+Shift+P,Ctrl+Shift+C

Selecting this item opens the procedure interface dialog (page 121) window. The procedure interface dialog and the mechanism of creating procedures are described in section "Creating Procedures" on page 121.

6.2.5.2 Duplicate...

Synopsis: Copy a procedure under a different name.

Selecting this menu item opens a dialog with which it is possible to copy existing procedures. The dialog is displayed in figure 6.35. The combo box Source contains all local procedures in the current program and all external procedures. In the Target text field the name of the copied procedure can be entered. Clicking the OK button creates a copy of the source procedure, Cancel dismisses the dialog. The copy retains the status (local or external) of the source procedure. The copy of an external procedure is placed in the same directory as the source procedure.

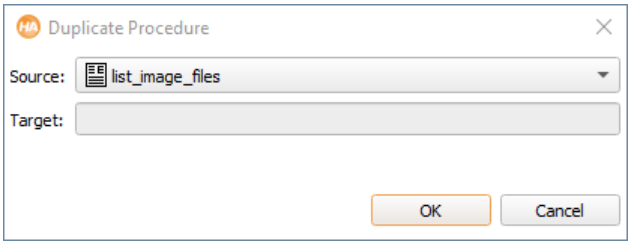


Figure 6.35: Duplicate Procedure.

Duplicating procedures that are protected with a password (see "Protected Procedures" on page 45) is also possible. The associated password is also used for the duplicated procedure.

6.2.5.3 Edit Procedure Interface

Synopsis: Edit procedure interface.

Shortcut: Ctrl+Shift+P,I or Ctrl+Shift+P,Ctrl+Shift+I 

This menu item opens the procedure interface window and displays the interface of the current procedure. The menu item has the same effect as the button  in the program window (page 113).

The interface of protected procedures can only be edited after the corresponding password has been entered (see section “Protected Procedures” on page 45).

6.2.5.4 Delete Current

Synopsis: Delete the current procedure.

Shortcut: Ctrl+Shift+P,Del or Ctrl+Shift+P,Ctrl+Shift+Del

If the current procedure is a local procedure, it is deleted from the program and the main procedure becomes the current procedure. All calls to the local procedure in the current program are marked as invalid code. This item is disabled if the current procedure is the main procedure, or if it is an external procedure.

6.2.5.5 Delete All Unused Local

Synopsis: Delete all local procedures that are not used in the current program.

All local procedures that cannot be reached by any procedure call from the main procedure are deleted from the program. If the current procedure is among the deleted procedures, the main procedure becomes the current procedure.

6.2.5.6 Insert Used As Local

Synopsis: Insert all used external procedures into the current program as local procedures.

The external procedures used in the current program are copied as local procedures. The external procedure files are left untouched.

6.2.5.7 Insert All As Local

Synopsis: Insert all external procedures into the current program as local procedures.

All external procedures are copied to the current program as local procedures, regardless if they are used or not. The external procedure files are left untouched.

With this menu item, you can change all of your procedures to become local. If your program contains protected external procedures, HDevelop issues a warning and inserts only the procedures that are not locked. Individual procedures can be made local (or external) via the combo box in the procedures interface (see section 6.4.5.1 on page 121). For changing the status of a procedure see section 5.6 on page 45.

6.2.5.8 Make All External

Synopsis: Convert all local procedures into external procedures.

The formerly local procedures are now stored as external procedures in a selectable directory of the list of external procedure directories (see section 6.2.2.16 on page 71). If no directories are configured, you can select a target directory from a dialog. HDevelop will add the selected target directory to the list if you tell it to. Otherwise, the operation will be canceled. For changing the status of a procedure see section 5.6 on page 45.

6.2.5.9 Manage Procedures

Synopsis: Configure procedure settings. ☼

This menu entry opens the dialog Preferences ▷ Procedures.

See section 6.2.2.16 on page 71 (ff) for a complete description.

6.2.5.10 Edit Procedure

Synopsis: Select a procedure for editing in the program window.

This submenu lists all procedures in submenus grouped by chapter and section title (see section 6.4.8.1 on page 128). Procedures without a chapter title are listed directly in the menu Edit Procedure. If you click on a procedure name, it will become the current procedure in the program window. You can also select procedures in the combo box of the program window (page 113).

6.2.6 Menu Operators

Synopsis: Select HALCON operators and procedures.

This menu item comprises all HALCON and HDevelop operators including the HDevelop control constructs as well as all internal and external HALCON procedures.

The following descriptions provide an overview of the operators and procedures specific to HDevelop programs. For detailed information about all operators and procedures, we strongly recommend reading the corresponding sections of the reference manual. To get there quickly, select an operator or procedure from the menu, and click the button Help in the operator window.

6.2.6.1 Control

Synopsis: Select control flow operators.

Here, you may select control structures for the program. This involves the execution of a program segment (henceforth named body) depending on a test (if, elseif) or comparison (switch, case, and endswitch) and the repetition of a program segment (for, while, and repeat) with controlled loop execution (break, continue). Exception handling is implemented with try, catch, and endtry along with throw for user-defined exceptions. Furthermore, you may stop the program's execution at any position (stop) or terminate HDevelop (exit). The operators assign and insert do not influence the execution, but serve to specify values for control data (assignment). The operator comment is used to add a comment, that means any sequence of characters, to the program. Using the operator export_def, you can enter special comments that will get uncommented when you export the HDevelop program to another programming language. This way, arbitrary code may be embedded into HDevelop programs. See page 342 for more information. The operator return terminates the current procedure call and returns to the calling procedure (see section "Creating Procedures" on page 121 for more information about HDevelop procedures).

Selecting a menu item displays the corresponding control construct in the operator window, where you can set the necessary parameters. After specifying all parameters you may transfer the construct into your program. A direct execution for loops and conditions is not possible, in contrast to other HDevelop and HALCON operators, because you have to specify the loop's and condition's body first to obtain useful semantics. If necessary, you may execute the program after the input with Step Over or Run. The IC is positioned after the construct head to ensure the input of the construct's body occurs in the correct place. This body is indented to make the nesting level of the control constructs visible, and thus to help you in understanding the program structure. The semantics for loops and conditions are shown in section "Control Flow Operators" on page 305.

Assignment

The operator assign serves as an assignment operator for control variables (numbers, strings and handles). Analogously to "normal" operators the input is made in the operator window by specifying both "parameters" Input and Result (i.e., right and left side of the assignment). An instruction in C, e.g.,

```
x = y + z;
```

is declared inside the operator window as

```
assign(y + z,x)
```

and displayed in the program window by

```
x := y + z
```

The operator `insert` implements the assignment of a single value (tuple of length 1) at a specified index position of a tuple. Thus, an array assignment (here in C syntax)

```
a[i] = v;
```

is entered as

```
insert(a,v,i,a)
```

in the operator window, and is displayed as

```
a[i] := v
```

in the HDevelop program window.

Program termination

The operators `stop` and `exit` are used to terminate the program. More precisely, `stop` *interrupts* an execution and `exit` *terminates* HDevelop. Having interrupted the execution you may continue the program by pressing `Step Over` or `Run`. This is useful, e.g., in demo programs to install defined positions for program interruption. Under Linux, you can use `exit` in combination with a startup file and the command line switch `-run`. Thus, HDevelop will not only load and run your application automatically, but also terminate when reaching `exit`.

Comments

The operator `comment` allows you to add a line of text to the program. This text has no effect on the execution of the program. A comment may contain any sequence of characters.

6.2.6.2 `Develop`

Synopsis: Select operators specific to HDevelop.

This menu contains several operators that help to adapt the user interface. These operators offer the same functionality that you have using mouse interaction otherwise. They are used to configure the environment from within a program. Using these operators, the program performs actions similar to the setting of a color in the parameter window, opening a window in the menu, or iconifying the program window with the help of the window manager.

All operators in this menu start with the prefix `dev_`. It has been introduced to have a distinction to the underlying basic HALCON operators (e.g., `dev_set_color` and `set_color`).

The effects of each operator are described as follows:

`dev_open_window`, `dev_close_window`, `dev_clear_window` The operators `dev_open_window` and `dev_close_window` are used to open and to close a floating graphics window, respectively. During opening, the parameterization allows you to specify the window's size and position. The operator `dev_clear_window` clears the active window's content and its history. This corresponds to the usage of the button `Clear` in the graphics window. Please note that `dev_open_window` and `dev_close_window` are not exported to Visual Basic .NET, and C# because here one `HWindowXCtrl` is used.

dev_set_window_extents With this operator, you can set the size and position of the active floating graphics window.

dev_set_window This operator activates the graphics window containing the given ID. This ID is an output parameter of dev_open_window. After the execution, the output is redirected to this window. This operator is not needed for exported code in C++ or C, because here every window operation uses the ID as a parameter. The operator has no effect for exported code in Visual Basic .NET, and C#.

dev_set_color, dev_set_colored dev_set_color has the same effects as the menu item Menu Visualization ▷ Color (page 91). dev_set_colored is equal to the menu item Menu Visualization ▷ Colored (page 90).

dev_set_draw This operator has the same effects as the menu item Menu Visualization ▷ Draw (page 91).

dev_set_line_width For an explanation see the menu item Menu Visualization ▷ Line Width (page 91).

dev_set_lut For an explanation see the menu item Menu Visualization ▷ Lut (page 91).

dev_set_paint For an explanation see the menu item Menu Visualization ▷ Paint (page 91). If you want to specify all possible parameters of a given paint mode, you have to specify them as a tuple, analogously to the HALCON operator set_paint.

dev_set_shape For an explanation see the menu item Menu Visualization ▷ Shape (page 91).

dev_set_part This operator adjusts the coordinate system for image, region, XLD and other graphic output. This is done by specifying the upper left and the lower right corner coordinates. This specified part is shown in the entire graphics window. If the width or height of the specified rectangle has a negative value (e.g., Row1 > Row2), the result is equivalent to the menu Menu Visualization ▷ Reset Parameters: the zoom mode is switched off, i.e., the *most recently* displayed image fills the whole graphics window. This feature of dev_set_part is *not* supported for exported C, C++, Visual Basic .NET, and C# code.

dev_display Iconic variables are displayed in the active graphics window by this operator. It is reasonable to do this when the automatic output is suppressed (see dev_update_window below and Menu Edit ▷ Preferences ▷ Runtime Settings ▷ Runtime Settings (page 80).

dev_clear_obj This operator deletes the iconic object stored in the HDevelop variable that is passed as the input parameter. In the variable window, the object is displayed as undefined (with a ? as its icon).

dev_inspect_ctrl This operator opens an inspection window displaying the values of the variable passed to the operator. To inspect multiple variables at once, you can pass a tuple of variable names. In most cases a list dialog is opened, which shows *all* values of the variable (see also section "Inspecting Variables" on page 148). In the case of an image acquisition device handle, a description of this image acquisition device is opened. In addition, this dialog allows online grabbing of images (see page 147). This operator is not supported for exported code.

dev_close_inspect_ctrl This is the opposite operator to dev_inspect_ctrl, and closes the floating inspect window. This operator is not supported for exported code.

dev_map_par, dev_unmap_par These operators open and close the parameter dialog, which can also be opened using the menu Menu Visualization ▷ Set Parameters. This operator is not supported for exported code.

dev_map_var, dev_unmap_var These operators iconify the variable window (dev_unmap_var), and retransform the iconified window to the normal visualization size, respectively (dev_map_var). This means that the variable window always remains visible on the display in one of the two ways of visualization. These operators can be executed with the help of the window manager. These operators are not supported for exported code.

dev_map_prog, dev_unmap_prog Analogously to dev_map_var and dev_unmap_var, these operators iconify or deiconify the program window. These operators are not supported for exported code.

dev_update_window, dev_update_var, dev_update_time, dev_update_pc Using these operators, you may configure the output at runtime. It corresponds to the settings in menu Menu Edit ▷ Preferences ▷ Runtime Settings ▷ Runtime Settings (page 80). These operators are not supported for exported code.

dev_set_check This operator is equivalent to set_check of the HALCON library. It is used to handle runtime errors caused by HALCON operators that are executed inside HDevelop. The parameter value 'give_error', which is the default, leads to a stop of the program together with an error dialog if a value not equal to H_MSG_TRUE is returned. Using the value '~give_error', errors or other messages are ignored and the program can continue. This mode is useful in connection with operators like get_mposition, file_exists, read_image, or test_region_point, which can return H_MSG_FAIL.

dev_error_var This operator specifies a variable that contains the return value (error code) of an operator after execution. This value can be used to continue, depending on the given value. dev_error_var is normally used in connection with dev_set_check. Note that, as the procedure concept in HDevelop only allows for local variables, the variable set by dev_error_var will only be valid in calls to the relevant procedure. Furthermore, every corresponding procedure call will have an own instance of the variable, i.e. the variable might have different values in different procedure calls. For an example how to use dev_error_var in connection with dev_set_check see %HALCONEXAMPLES%\hdevelop\Graphics\Mouse\get_mposition.hdev.

dev_open_dialog, dev_open_tool, dev_close_tool, dev_show_tool,

dev_set_tool_geometry, dev_open_file_dialog These operators perform GUI operations from an HDevelop program.

dev_get_window This operator returns the handle of the active graphics window.

Please note that operations concerning graphics windows and their corresponding operators have additional functionality in comparison to HALCON operators with corresponding names (without dev_): graphics windows in HDevelop are based on HALCON windows (see open_window in the HALCON reference manual), but in fact, they have an enhanced functionality (e.g., history of displayed objects, interactive modification of size, and control buttons). This is also true for operators that modify visualization parameters (dev_set_color, dev_set_draw, etc.). For example, the new visualization parameter is registered in the parameter window when the operator has been executed. You can easily check this by opening the dialog Menu Visualization ▷ Set Parameters ▷ Pen and apply the operator dev_set_color. Here you will see the change of the visualization parameters in the dialog box. You have to be aware of this difference if you export dev_* to C, C++, Visual Basic .NET, and C# code.

In contrast to the parameter dialog for changing display parameters like color, the corresponding operators (like dev_set_color) do not change the contents of the graphics window (i.e., they don't cause a redisplay). They are used to prepare the parameters for the *next* display action.

6.2.6.3 1D Measuring, 2D Metrology, 3D Matching, ...

Synopsis: Select HALCON operators and procedures.

In these menu entries, you can find all HALCON operators as well as all internal and external procedures, arranged in chapters and sections. This set of image analysis operators and procedures forms the most important part of HALCON: the HALCON library. HALCON operators implement the different image analysis tasks such as preprocessing, filtering, or measurement.

You may look for a detailed description of each operator in the HALCON reference manual. Operators in the menus Control and Develop are special operators of HDevelop. Thus, you will not find them in the reference manuals for HALCON/C, or HALCON/C++.

The menu has a cascade structure, according to the chapter structure of the HALCON reference manual. As this menu is built up dynamically when HDevelop starts, it might take some time until it is available. During the build-up time the menu is "grayed out". Selecting a chapter of the menu opens a pulldown menu with the corresponding sections or operators, respectively.

This operator hierarchy is especially useful for novices because it offers all operators sorted by thematic aspects. This might be interesting for an experienced user, too, if he wants to compare, e.g., different smoothing filters, because they reside in the same subchapter. To get additional information, a short description of an operator (while activating its name in the menu) is displayed in the status bar.

Note, that some operators are visible in the menus but should not be used, e.g., open_window (in Menu Operators ▷ Graphics ▷ Window) or reset_obj_db (in Menu Operators ▷ System ▷ Database). If you select one of these operators, a warning text is displayed in the operator window. This warning will usually refer to a legal substitute. In the case of most of these operators, you should use the corresponding Develop operator (e.g., dev_open_window instead of open_window) *within* HDevelop.

6.2.6.4 Legacy

Synopsis: Select obsolete HALCON operators.

This menu lists all HALCON obsolete operators, arranged in chapters and sections. These operators are only provided for backward compatibility. For new applications the corresponding operators listed in "1D Measuring, 2D Metrology, 3D Matching, ..." on page 104 should be used. Selecting an operator or a procedure inserts it into the operator window for editing.

6.2.6.5 Unclassified

Synopsis: Select unclassified HALCON operators and procedures

This menu lists all unclassified HALCON operators and procedures. These operators and procedures are not listed in any of the chapters listed in "1D Measuring, 2D Metrology, 3D Matching, ..." on page 104 as there is no sectioning information associated with these operators and procedures.

6.2.7 Menu Suggestions

Synopsis: Let HDevelop suggest operators based on the current operator.

This menu shows you another possibility how to select HALCON operators. But here they are proposed to you in a different manner. It is assumed that you have already selected an operator in a previous step. Depending on this operator, five different suggestions are offered.

Suggestions are separated into groups as described below.

6.2.7.1 Alternatives

Since HALCON includes a large library, this menu item suggests alternative operators. Thus, you may, for example, replace mean_image with operators such as binomial_filter, gauss_filter, or smooth_image.

6.2.7.2 See also

Contrary to Alternatives, operators are offered here which have some *connection* to the current operator. Thus, the median filter (median_image) is not a direct alternative to the mean filter (mean_image). Similarly, the regiongrowing operator (regiongrowing) is no alternative for a thresholding. In any case, they offer a different approach to solve a task. References might consist of pure informative nature, too: the operator gen_lowpass, which is used to create a lowpass filter in the frequency domain, is a reasonable reference to a Gaussian filter.

6.2.7.3 Predecessors

Many operators require a reasonable or necessary predecessor operator. For example, before computing junction points in a skeleton (junctions_skeleton), you have to compute this skeleton itself (skeleton). To obtain a threshold image you usually use a lowpass filter before executing a dynamic threshold (dyn_threshold). Using the watershed algorithms (watersheds), it is reasonable to apply a smoothing filter on an image first, because this reduces runtime considerably.

6.2.7.4 Successors

In many cases the task results in a "natural" sequence of operators. Thus, as a rule you use a thresholding after executing an edge filter or you execute a region processing (e.g., morphological operators) after a segmentation. To facilitate a reasonable processing, all the possible operators are offered in this menu item.

6.2.7.5 Keywords

This menu item gives access to HALCON operators through keywords which are associated with each operator. The tab card Index of the online help window is opened. It is described in "Index" on page 167.

6.2.8 Menu Assistants

This menu assembles assistants for specific machine vision tasks. The general concept of the assistants is described in the chapter "HDevelop Assistants" on page 193.

The following assistants are available:

- Image Acquisition Assistant (page 194)
- Calibration Assistant (page 199)
- Matching Assistant (page 217)
- Measure Assistant (page 249)
- OCR Assistant (page 262)

6.2.9 Menu Window

This menu offers support to manage the sub-windows of the main window, i.e., the program, operator, variable, graphics window(s), and possibly other dialogs. At the bottom of the menu all open windows are listed. Clicking an entry here brings the corresponding window to the front.

6.2.9.1 Open Graphics Window

Synopsis: Open an additional graphics window.

Shortcut: Ctrl+Shift+O,G or Ctrl+Shift+O,Ctrl+Shift+G 

See also: dev_open_window, and section "Open Graphics Window..." on page 88..

If no graphics window is open, double-clicking an iconic variable will also open a new graphics window.

6.2.9.2 Open Program Window

Synopsis: Open a program window.

Shortcut: Ctrl+Shift+O,P or Ctrl+Shift+O,Ctrl+Shift+P 

See also: dev_map_prog.

This menu item opens a new program window. See section "Program Window" on page 113.

6.2.9.3 Open Variable Window

Synopsis: Open the variable window.

Shortcut: Ctrl+Shift+O,V or Ctrl+Shift+O,Ctrl+Shift+V 

See also: dev_map_var.

This menu item is grayed out if the variable window is already open.

6.2.9.4 Open Operator Window

Synopsis: Open the operator window.

Shortcut: Ctrl+Shift+O,O or Ctrl+Shift+O,Ctrl+Shift+O

This menu item is grayed out if the operator window is already open. You can also open the operator window by holding Ctrl and double-clicking a line in the program window.

6.2.9.5 Open Output Console

Synopsis: Open the output console window.

Shortcut: Ctrl+Shift+O,E or Ctrl+Shift+O,Ctrl+Shift+E

The output console window contains a log of the most recent messages. This includes all messages displayed on the status bar as well as HALCON low-level errors. Therefore, HALCON low-level errors can now be logged without an interruption of the program execution.

The window can also be opened by double-clicking the message area of the status bar. See figure 6.36 for an example.

The logged messages are grouped into the following categories:

(Info) Info events related to the program execution, the just-in-time (JIT (page 45)) compiler, and file operations such as loading and saving programs.

(Warning) Warning events related to the program execution, the just-in-time (JIT (page 45)) compiler, file operations, HALCON low-level errors, and invalid program lines.

(Error) Program execution errors, HALCON errors, and internal errors.

(Exception) Exception events.

The tool bar includes the following action buttons:

Clear the output console.

Save the log messages as a plain text file (*.txt* or *.csv*). Each line contains a log message, and the columns are separated by tabs.

Copy the selected log messages to the system clipboard.

Open the preferences window of the output console. You can specify the maximum number of log messages that are kept in the console. The check boxes toggle the visibility of the corresponding messages.

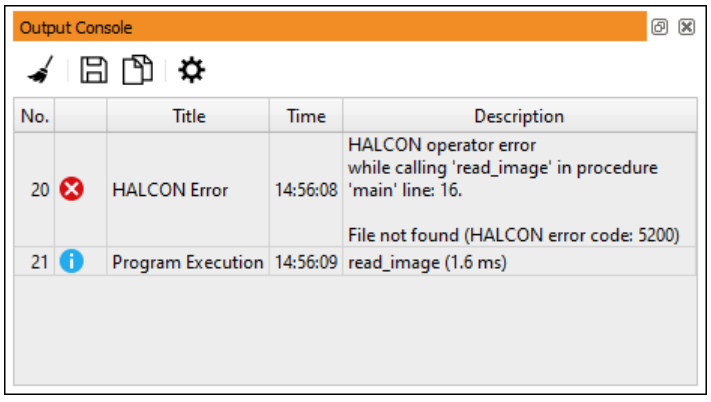
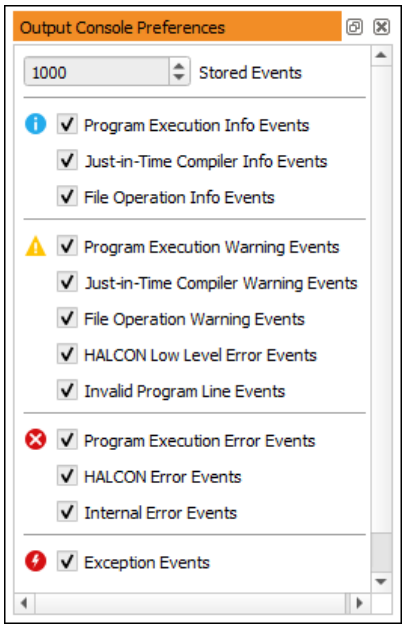


Figure 6.36: Open Output Console.



6.2.9.6 `Open Quick Navigation`

Synopsis: Manage program lines based on certain criteria.

Shortcut: Ctrl+Shift+O,R or Ctrl+Shift+O,Ctrl+Shift+R

The quick navigation window contains a listing of program lines that share a common property. Clicking an entry focuses the corresponding program line in the active program window. If the selected procedure is already displayed in a program window tab, the corresponding tab is activated. Otherwise, the current view switches to the selected procedure.

Furthermore, it is possible to activate, deactivate, copy, cut, or delete one or more of the selected program lines directly from the quick navigation.

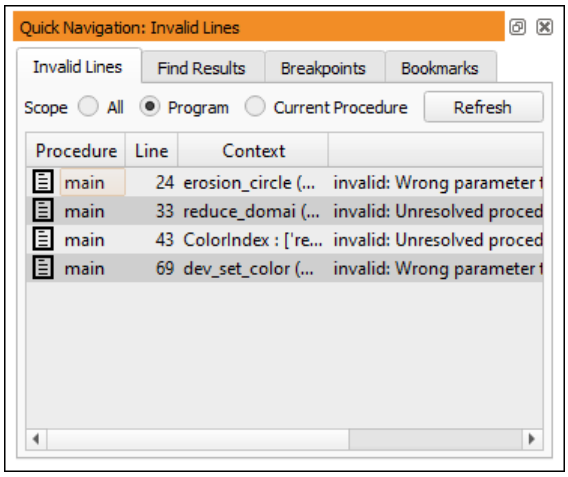


Figure 6.37: Managing program lines in the quick navigation window.

The “common properties” are available as tab cards:

- `Invalid Lines`

  Provides a listing of invalid program lines (see page 65).

- `Find Results`

  Provides a listing of program lines that match a previous “find all” operation (see page 63). If the `Find/Replace` dialog is not open, click the button `Open Find/Replace Dialog`.

- Breakpoints

  Provides a listing of set breakpoints (see page 86).

- Bookmarks

  Provides a listing of bookmarked program lines (see page 65).

6.2.9.7 Open Canvas Window

Synopsis: Arrange your graphics windows on the canvas.

Shortcut: Ctrl+Shift+O,A or Ctrl+Shift+O,Ctrl+Shift+A

The canvas window is a container for graphics windows. It allows you to organize multiple graphics windows side-by-side. For more information, please refer to section 6.9 on page 161.

For configuration of the canvas, open the context menu by right-clicking on the canvas. Alternatively, choose Canvas Options from the menu Window. Learn more about canvas options in the next section.

6.2.9.8 Canvas Options

Synopsis: Configure the canvas window.

- Show Grid

  The grid may help you arranging and resizing your graphics windows.

- Show Graphics Window Toolbar

  You can hide the toolbar, for example to maximize the image display. Most graphics windows features are available via context menu, too. For example the feature Float window to remove the graphics window from the canvas again.

- Show Graphics Window Frame

  The frame is necessary to make changes to the size or position of the graphics window. Disable this option, if you are happy with its current position and size or if you want to focus on the image.

6.2.9.9 Restore Default Layout

Synopsis: Reset the window layout to defaults.

Shortcut: Ctrl+Shift+W,R or Ctrl+Shift+W,Ctrl+Shift+R

In the default layout, the variable window is docked bottom left and the program window on the right. Depending on the setting of Use Canvas, either the canvas window or the graphics window is docked top left (see Preferences ▷ General Options ▷ General Options ▷ Default layout uses Canvas).

- If Use Canvas is **enabled**, the **canvas window** is docked top left.
- If Use Canvas is **disabled**, the **graphics window** is docked top left.

6.2.9.10 Cascade Floating Windows

Synopsis: Arrange floating windows in a cascade to quickly switch between windows.

Shortcut: Ctrl+Shift+W,C or Ctrl+Shift+W,Ctrl+Shift+C

Arranges all floating windows so that they overlap each other while keeping their title bars visible. The origin of the first cascading window is set by Origin of coordinates (page 78).

6.2.9.11 Bring Main Window To Front

Synopsis: Raises the main window to the top.

Shortcut: Ctrl+Shift+W,W or Ctrl+Shift+W,Ctrl+Shift+W

6.2.9.12 Bring Others To Front

Synopsis: Raises all floating windows to the top.

Shortcut: Ctrl+Shift+W,B or Ctrl+Shift+W,Ctrl+Shift+B

6.2.9.13 Full Screen

Synopsis: Toggle full screen mode.

Shortcut: Ctrl+Shift+W,F or Ctrl+Shift+W,Ctrl+Shift+F

In full screen mode the main window is maximized, and the window decorations are removed to make full use of the screen estate.

6.2.10 Menu Help

Here, you may query information about HALCON itself and all HALCON and HDevelop operators.

6.2.10.1 Help

Synopsis: Open the online help window.

Shortcut: F1

The help window provides access to the documentation of HDevelop and HALCON. In particular, the complete HALCON Reference Manual is available with extensive documentation of each operator. While in the program window, pressing F1 opens the documentation of the operator in the current line. Another possibility of requesting information about the current operator is pressing the button Help inside the operator window (see section "Operator Window" on page 138).

The help window is described in section "Help Window" on page 165.

6.2.10.2 Context Help

Synopsis: Provide context-sensitive online help.

Shortcut: Shift+F1

Based on the currently focused window or tab card, the corresponding page in the online help is opened.

6.2.10.3 Start Dialog

Synopsis: Display the HDevelop Start Dialog.

Shortcut: Ctrl+Shift+H,S or Ctrl+Shift+H,Ctrl+Shift+S

6.2.10.4 HALCON Reference

Synopsis: Display the HALCON Reference Manual in the online help window.

Shortcut: Ctrl+Shift+H,R or Ctrl+Shift+H,Ctrl+Shift+R

6.2.10.5 HDevelop User's Guide

Synopsis: Display the HDevelop User's Guide in the online help window.

Shortcut: Ctrl+Shift+H,U or Ctrl+Shift+H,Ctrl+Shift+U

6.2.10.6 HDevelop Language

Synopsis: Display the HDevelop Language chapter in the online help window.

Shortcut: Ctrl+Shift+H,L or Ctrl+Shift+H,Ctrl+Shift+L

6.2.10.7 Search Documentation

Synopsis: Open the online help window and show the search tab to enter search queries.

Shortcut: Ctrl+Shift+H,F or Ctrl+Shift+H,Ctrl+Shift+F

The online help provides an integrated search engine. You can enter search queries there and search the HALCON documentation suite.

The search syntax is described in section "Help Window" on page 165.

6.2.10.8 HALCON News (WWW)

Synopsis: Visit the HALCON home page.

Shortcut: Ctrl+Shift+H,W or Ctrl+Shift+H,Ctrl+Shift+W

This menu item lets you check for the latest news about HALCON on MVTec's WWW server, e.g., whether new extension packages, image acquisition interfaces, or HALCON versions are available.

6.2.10.9 Check For Updates

Synopsis: Check for newer versions of HALCON.

Shortcut: Ctrl+Shift+H, C

This menu item opens a version check in your web browser. See https://www.mvtec.com/imprint for information about MVTec's privacy policy.

6.2.10.10 About

Synopsis: Display HDevelop version and licensing host IDs.

Shortcut: Ctrl+Shift+H,A or Ctrl+Shift+H,Ctrl+Shift+A

This window provides the following information:

- HALCON edition (Progress, Steady, or Steady Deep Learning)

- HDevelop version (and build date)
- HALCON version (and build date)
- HALCON license type
- license expiration date
- platform version (HALCONARCH)
- host and dongle IDs

The host and dongle IDs are detected by the license manager (see the Installation Guide for more information).

If the used CPU supports AVX2, this is displayed after the platform version.

6.3 Tool Bar

The following table lists all tool bar actions and their shortcuts.

	Action	Shortcut	Info
	New Program	Ctrl+N	51
	Open Program...	Ctrl+O	52
	Browse HDevelop Example Programs...	Ctrl+E	52
	Save	Ctrl+S	53
	Save All	Ctrl+Alt+S	55
	Image Acquisition Assistant		194
	Undo	Ctrl+Z	60
	Redo	Ctrl+Y	60
	Activate	F3	61
	Deactivate	F4	61
	Find/Replace...	Ctrl+F	62
	Run	F5	82
	Step Over	F6	83
	Step Into	F7	84
	Step Out	F8	84
	Stop	F9	84
	Reset Program Execution	F2	87
	Reset Procedure Execution	Shift+F2	87
	Thread View / Call Stack	Ctrl+Shift+O,C or Ctrl+Shift+O,Ctrl+Shift+C	85
	Activate Profiler	Ctrl+Shift+F,F or Ctrl+Shift+F,Ctrl+Shift+F	88
	Zoom Window	Ctrl+Shift+O,Z or Ctrl+Shift+O,Ctrl+Shift+Z	97
	Gray Histogram	Ctrl+Shift+O,H or Ctrl+Shift+O,Ctrl+Shift+H	98
	Feature Histogram	Ctrl+Shift+O,F or Ctrl+Shift+O,Ctrl+Shift+F	98
	Feature Inspection	Ctrl+Shift+O,I or Ctrl+Shift+O,Ctrl+Shift+I	98
	Line Profile	Ctrl+Shift+O,L or Ctrl+Shift+O,Ctrl+Shift+L	98
	OCR Training File Browser	Ctrl+Shift+O,T or Ctrl+Shift+O,Ctrl+Shift+T	98

6.4 Program Window

A program window displays the HDevelop program code of a single procedure. You can open multiple program windows to view different procedures or different parts of the same procedure at the same time (see Menu `Window` ▷ `Open Program Window` (page 106)). To switch between selected procedures rapidly, program windows support multiple tabs.

The program window (see figure 6.38) is divided into three areas:

- The main part of the program window contains the program code of the current HDevelop procedure. See section 6.4.2.
- The column at the left side displays line numbers. It also contains the PC, the IC, and optionally, one or more breakpoints. Furthermore, bookmarked lines and warning indicators are displayed. See section 6.4.3 on page 119.
- At the top, the displayed procedure can be selected from the drop-down list. If you hold down Ctrl while selecting a procedure, it will be displayed in a new tab card.

  The arrow buttons provide convenient access to previously displayed procedures. For example, if the current procedure is the main procedure, and you select another procedure from the drop-down list, the left arrow button takes you back to the main procedure. When you get there, the right arrow button moves forward, and displays the previously selected procedure again.

  Using the tool bar button to the right of the procedure list, the interface of the current procedure and its documentation can be edited, i.e., the number of parameters as well as their names and types, can be modified. See section 6.4.5 on page 121 for a detailed description.

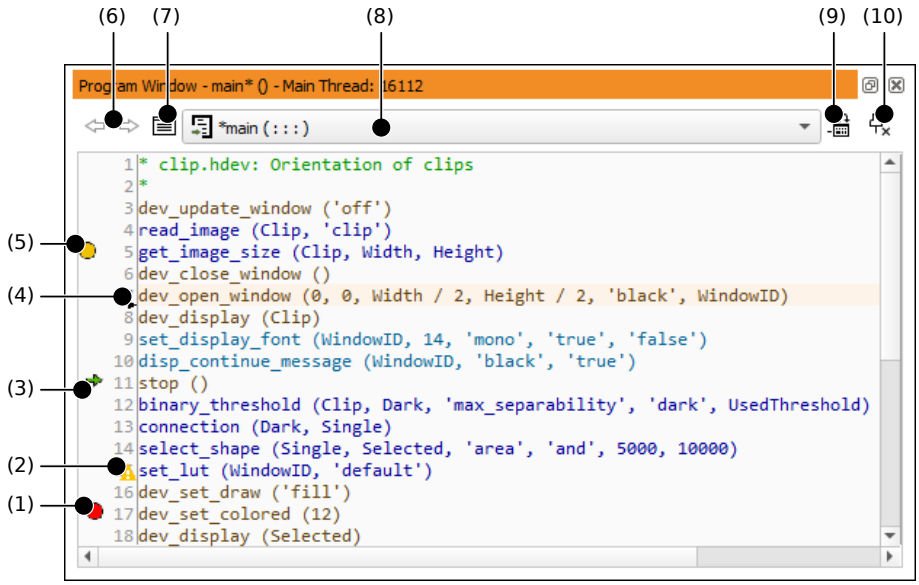


Figure 6.38: Program window: (1) BP, (2) warning, (3) PC, (4) IC, (5) bookmark, (6) browse history, (7) add new tab, (8) select procedure, (9) edit procedure interface, (10) pin program listing.

6.4.1 Program Window Actions

	Action	Shortcut	Description
⇦	Back in History	Alt+Shift+Left	Back in history.
⇨	Forward in History	Alt+Shift+Right	Forward in history.
	Show Previous Tab Card	Alt+Left	Select the next tab card to the left.
	Show Next Tab Card	Alt+Right	Select the next tab card to the right.
	New Tab		Open a new tab in the current program window.
	Edit Procedure Interface	Ctrl+Shift+P,I or Ctrl+Shift+P,Ctrl+Shift+I	Edit interface and documentation of current procedure.
	Pin the program listing		If enabled, automatic scrolling is turned off during program execution.
	List Open Tabs		Lists all open tabs (if multiple tabs are displayed). Click on a list item to activate the corresponding tab.
	Set Program Counter	Ctrl+,	Set program counter to current program line.
	Set Insert Cursor	Ctrl+.	Set insert cursor to current program line.
	Set/Clear Breakpoint	F10	Set or clear breakpoint in current line.
	Help	F1	Open the reference documentation of the operator or procedure call of the current line.
	Select a Procedure	Alt+Up	Display a selected procedure from the list.
	Move tab to the left	Ctrl+Alt+Shift+Left	Move the current tab to the left.
	Move tab to the right	Ctrl+Alt+Shift+Right	Move the current tab to the right.
	Go to Line	Alt+G	Move cursor to the specified line.

Action	Shortcut		Description
Go to Program Counter	Alt+,		Move cursor to program counter.
PC Down	Ctrl+Down		Move program counter down.
PC Up	Ctrl+Up		Move program counter up.
Show main	Alt+Home		Display main procedure.
Execute Current Line	Ctrl+Return Ctrl+Enter	or	Execute current line
Zoom in	Ctrl+mouse wheel up		Increase font size in program window.
Zoom out	Ctrl+mouse wheel down		Decrease font size in program window.
Scroll up	mouse wheel up		Scroll up through the lines of program code.
Scroll down	mouse wheel down		Scroll down through the lines program code.
Scroll up page-wise	Shift+mouse wheel up		Scroll up through the pages of program code.
Scroll down page-wise	Shift+mouse wheel down		Scroll down through the pages of program code.

6.4.2 Editing Programs

The program window supports full text editing, i.e., text can be freely edited by placing the text cursor with the left mouse button and typing away. Continuous portions of text may be selected by dragging with the mouse. An existing selection may be extended by holding down Shift and clicking the left mouse button. Double-clicking selects the word under the mouse pointer while triple-clicking selects the entire line.

To edit a program line in the operator window, double-click it with the left mouse button. If the operator window is not currently open, hold down Ctrl while double-clicking.

- The window title of the operator window clearly indicates that you are *editing an existing program line*. It also displays the procedure name and the line number.
- Clicking `OK` or `Replace` in the operator window **will replace** the original program line. This is even the case if the corresponding program line is no longer in view, e.g., if a different procedure is selected in the program window.

  If the program line is deleted before the changes are committed in the operator window, the edited line will be inserted as a new program line at the IC. If you are in doubt about the current status, check the window title of the operator window.

Comments

Comments start with an asterisk (*) as the first (non-whitespace) character. You may also add comments at the end of regular program lines by introducing them with //.

```
* this comment is ignored during program execution
read_image (Image, 'clip')              // instantiate iconic variable
threshold (Image, Region, 0, 63)         // select dark pixels
```

Special input formats

Certain operator calls may be entered in different formats. Syntactically, they are equivalent but the first variant is more readable and thus recommended. It is also the one used when entering the corresponding calls via the operator window.

- variable assignments

  ```
  FileName := 'clip'
  assign('clip', FileName)                  // deprecated
  ```

- setting individual tuple elements

```
Line[12] := 'text'
assign_at(12, 'text', Line)              // deprecated
```

Note that `for` loops always have to be entered in the following format:

```
for Index := Start to End by Step
 ...
endfor
```

Line continuation

Operator calls may span several lines for readability. A line may be continued by entering a backslash character as the last character of that line.

For example, you can enter

```
disp_arrow (WindowID, \
            Row[i], \
            Column[i], \
            Row[i]-Length*sin(Phi[i]), \
            Column[i]+Length*cos(Phi[i]), \
            4)
```

instead of

```
disp_arrow (WindowID, Row[i], Column[i], Row[i]-Length*sin(Phi[i]), Column[i]+...
```

Auto Indenting

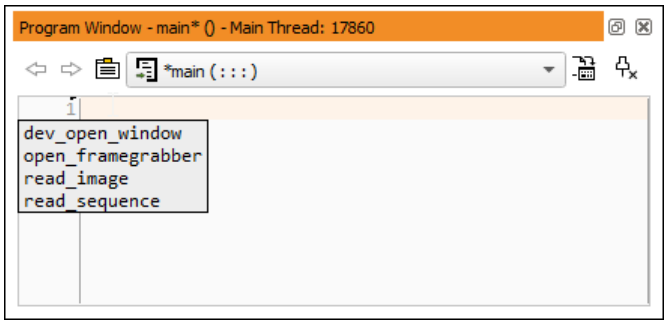
The indent of new lines is adjusted automatically. Usually, the indent of the previous line is maintained. If a line is continued inside the parentheses of an operator call, the new line is indented up to the opening parenthesis. If the previous line opens a control structure (e.g., `if` or `while`), the indent is increased by the indent size. The indent size is specified in the preferences (see `Program Window` (page 68)). It defaults to four spaces. If a control structure is closed (e.g., by entering `endif` or `endwhile`), the indent of the current line is decreased by the indent size.

Advanced Autocompletion

The program window provides advanced autocompletion to support you in several ways.

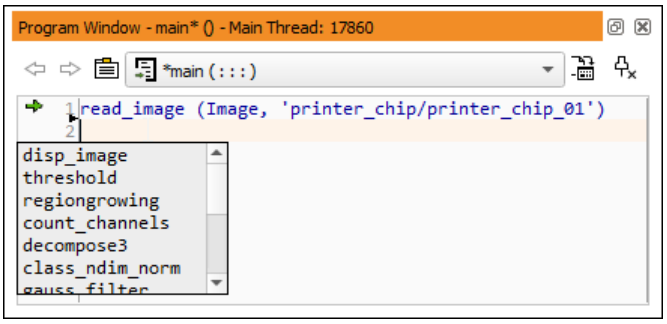
- **Getting started**

  HDevelop's autocompletion can assist you getting started with a completely empty program. Simply press Tab to get a list of typical first operators like `dev_open_window` and `read_image`.

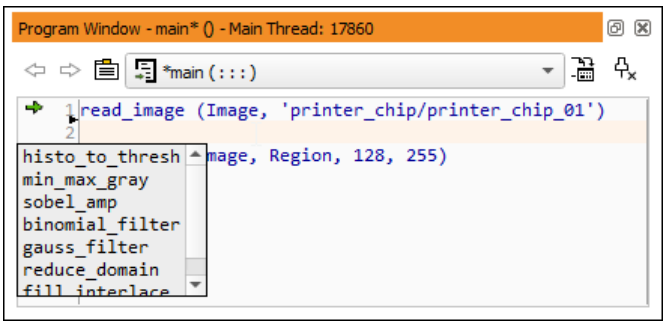


- **Suggestions for empty code lines**

  Possible successors of the previous HALCON operator are suggested after pressing Tab.

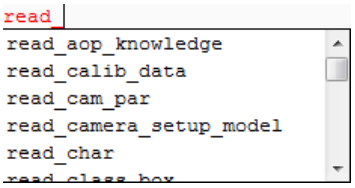


Press Shift+Tab to get suggestions for possible predecessors.



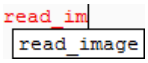
- **Typing support**

  Enter programs quickly and correctly without restricting your typing in any way. When you start typing a new line, HDevelop will suggest a list of matching operator names:



  Note that the line is highlighted as invalid (red in the default color scheme) because it is still incomplete.

  The list is updated immediately as you continue typing:



  Press Tab to complete to the longest common string. In this example, only one operator name remains in the list. Thus, it is fully completed, including the opening parenthesis of the operator call:



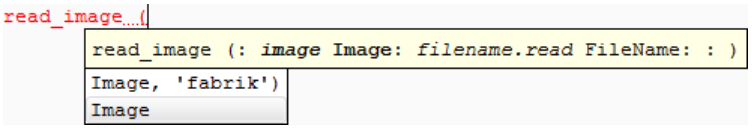
  Once the cursor moves inside the parentheses, the suggestion list changes from operator mode to parameter mode. Furthermore, the signature of the selected operator is displayed, and the parameter corresponding to the cursor position is highlighted in bold.

  The first entry in this list is a suggestion that completes the full operator call up to the closing parentheses. Again, typing ahead updates the list of suggestions accordingly. The remaining entries are suggestions for the first parameter of the operator call.

  At this point, press Tab to select the first suggestion,



  or press Up or Down to step through the list entries,



and press Tab or Return to select the highlighted entry. Then, enter a comma or press Tab again to get suggestions for the second parameter:

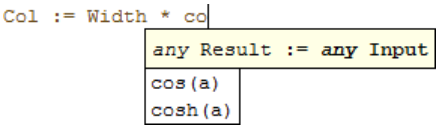


Note the browse button in the suggestion list. Double-clicking it opens up a file selection dialog (page 190) to specify the file name parameter. These browse buttons are included in the suggestions lists of all parameters that specify a file name.

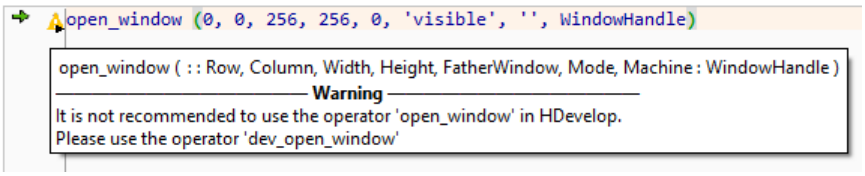
In this example, we want to load the image `clip`, so none of the suggestions fits. Just type the file name in single quotes (`'clip'`) and press Tab to complete the parameter list. The closing parenthesis is inserted automatically:

```
read_image (Image, 'clip')
```

When entering expressions, the advanced autocompletion also suggests operations (functions) supported by the HDevelop language (see section "HDevelop Language" on page 281).



Please note that the advanced autocompletion does not suggest legacy or deprecated operators. If you enter such an operator manually, a warning icon is displayed in the left column. Move the mouse cursor over this icon to get a tool tip with a corresponding warning message:



The program window also supports autocompletion of block statements. When entering a control statement, e.g., the control statement of a `for` loop, the corresponding end statement, e.g., `endfor` of the block is automatically inserted by pressing Return.

Special Keyboard Shortcuts in the Program Window

General:

Tab	*cursor at the beginning of line*: adjust indentation of current line *selected text:* indent corresponding code lines one level
Shift+Tab	*selected text*: outdent corresponding code lines one level
Shift+Return	reverse action of Return key (see Program Window Actions)
Ctrl+Return	execute current line (same as clicking `Apply` in operator window, see section "Control Buttons" on page 142)
Ctrl+F	open find/replace dialog with selected text (section "`Find/Replace...`" on page 62).

Advanced autocompletion:

Escape	hide suggestion list
Ctrl+Space	re-display suggestion list based on cursor position or selection
Up	highlight previous entry in suggestion list
Down	highlight next entry in suggestion list
Tab	*operator suggestions:* complete to highlighted suggestion or to longest common string from suggestion list *parameter suggestions:* complete to highlighted suggestion or first suggestion if no suggestion is highlighted
Return	*cursor at the end of a block statement*: insert the corresponding end statement of the block

Special Operator Handling

Some operators provide additional functionality when being edited. This functionality is available via an action button next to the parameter field in the operator window, or as an action button in the advanced autocompletion suggestion list.

If an operator contains a parameter that specifies a file name, the parameter value can be specified in a file selection dialog (page 190). See the previous section for an example (`read_image`).

The operators `info_framegrabber` and `open_framegrabber` provides a button to detect the available image acquisition interfaces automatically (see also Image Acquisition Assistant (page 194)).

6.4.3 Program Counter, Insert Cursor, and Breakpoints

The column to the left of the displayed program body contains

- the program counter (PC) ➪,
- the insert cursor (IC) ▶,
- and optionally one or more breakpoints (BPs) ●.

The PC resides in the line of the next operator or procedure call to execute. The IC indicates the position to insert a new program line. A breakpoint (BP) shows the program line on which the program is stopped.

You may position or activate these three labels by clicking in the left column of the program window. That column itself is divided into three areas: Depending on the horizontal position of the mouse cursor, all three label types are available. The actual type is indicated through a change of the mouse cursor. At the leftmost position, BPs can be placed. In the middle position, the PC can be placed. And finally, in the rightmost position, the IC can be placed. If this seems confusing, you can force a specific label by holding down the following keys regardless of the horizontal position:

- Hold Shift to place the IC.
- Hold Ctrl to place or delete a BP.
- Hold Ctrl+Shift to place the PC.

The PC can only be placed on program lines of procedures on the callstack. For example, in figure 6.39 on page 122, the PC may be placed anywhere in the procedures *main*, *first*, or *second*. It may not be placed within the procedure *third*. If the PC is placed in *first*, the first element of the callstack is popped. If the PC is placed in *main*, only *main* remains on the callstack. Please note the outlined green arrows in *main* and *first*: They indicate the return positions.

6.4.4 Context Menu

By clicking into the program window with the right mouse button you can open a context menu, which contains shortcuts to some of the actions of the menus Menu `Edit`, e.g., copy and paste lines, and Menu `Execute`, e.g., activate and deactivate lines or set and clear breakpoints. Please note that these actions behave slightly differently than their counterparts in the main menus: When called via the main menus, the actions are performed only on the selected part of the program; if nothing is selected, no action is performed. In contrast, when an action is called via the context menu and no line is selected in the program, the action is performed for the line that you right-clicked.

Note that any actions that modify the position of the PC will cause the call stack to pop all procedure calls until the current procedure call remains on top. This is relevant in case the current procedure call is not the top-most procedure call and is necessary to secure the consistency of the call stack. Modification of the PC can happen as well directly as described above or indirectly by, e.g., inserting a program line above the PC in the current procedure body.

Action	Shortcut	Description
`Run Until Here`	Shift+F5	Execute the lines from the PC to the line under the mouse cursor.
`Open Operator Window`	Ctrl+Shift+Space	Open the current operator or procedure call in the operator window for editing.

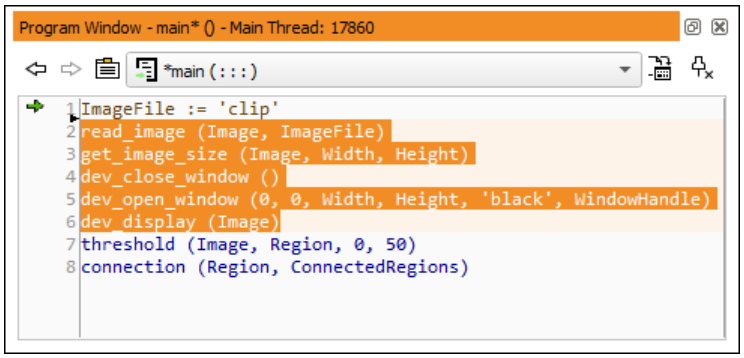
	Action	Shortcut		Description
	Help			If the line under the mouse cursor contains an operator call, the corresponding page will be opened in the online help window. This is a shortcut to double-clicking the program line and clicking Help in the operator window.
	Show Procedure	Alt+Return Alt+Enter	or	If the line under the mouse cursor contains a procedure call, the corresponding procedure will become the current procedure, i.e., it is displayed for editing.
	Show Procedure in New Tab	Alt+Ctrl+Return Alt+Ctrl+Enter	or	If the line under the mouse cursor contains a procedure call, the corresponding procedure will be displayed as a new tab in the current program window. If the corresponding procedure tab exists already, it will be activated.
	Show Procedure in New Window	Alt+Shift+Return Alt+Shift+Enter	or	If the line under the mouse cursor contains a procedure call, the corresponding procedure will be displayed in a new program window.
	Show Caller			This menu item lists all the places in the current program where the currently selected procedure is called. Clicking on an entry takes you to the corresponding program line.
	Cut	Ctrl+X	or Shift+Del	Cut selected text.
	Copy	Ctrl+C		Copy selected text.
	Paste	Ctrl+V	or Shift+Ins	Paste clipboard content at cursor position or replace selected text with clipboard content.
	Delete	Del		Delete selected text.
	Activate	F3		Activate selected lines.
	Deactivate	F4		Deactivate selected lines.
	Auto Indent	Ctrl+Shift+I		The indenting level of all selected program lines is updated, i.e., nested program blocks are indented by the amount of spaces set in the preferences.
	Create New Procedure	Ctrl+Shift+P,C Ctrl+Shift+P,Ctrl+Shift+C	or	See Create New Procedure (page 99).
	Set Program Counter	Ctrl+,		Set the PC to the selected line.
	Set Insert Cursor	Ctrl+.		Set the IC to the selected line.
	Update Program Counter			See preferences (page 80).
	Set/Clear Bookmark	Ctrl+F11		See Set/Clear Bookmark (page 64).
	Set/Clear Breakpoint	F10		See Set/Clear Breakpoint (page 86).
	Activate/Deactivate Breakpoint	Shift+F10		See Activate/Deactivate Breakpoint (page 86).
	Set/Clear Breakpoint on Variable			Toggle breakpoint on variable under mouse cursor. See section “Breakpoints on Variables” on page 143.
	Activate/Deactivate Breakpoint on Variable			Toggle activation of breakpoint on variable under mouse cursor.

	Action	Shortcut	Description
	Manage Breakpoints	Ctrl+Shift+O,F10 or Ctrl+Shift+O,Ctrl+Shift+F10	See Manage Breakpoints (page 86).
	Add Watch		Add the variable under the mouse cursor to the user tab of the variable window.
	Print...	Ctrl+P	See Print... (page 58).

6.4.5 Creating Procedures

Procedures may be created from scratch or from selected program lines in the currently displayed procedure. When you start a new HDevelop program, there is only the *main* procedure. As the program grows, you often find that chunks of code have to be reused or they constitute a functional unit. In these cases, it is good practice to relocate the corresponding lines to a new procedure.

As an example, consider the following example program:



In this example, you want to reuse the selected program lines. To create a new procedure from these program lines, click the menu entry Procedures ▷ Create New Procedure (page 99) or select the corresponding entry from the context menu of the program window. The new procedure can now be setup in the procedure interface dialog.

6.4.5.1 Setting Up the General Settings of a Procedure

See also: Procedures ▷ Create New Procedure (page 99) / Edit Procedure Interface (page 100).

Name The procedure name must start with a letter and may consist of alphanumeric characters and underscores. If you enter an invalid name, i.e., an operator name, a reserved word, or a name that contains invalid characters, the text field will be highlighted. You will not be able to close the dialog and apply the changes until a valid name is provided.

Operator names cannot be used because operators and procedures share the same namespace. However, procedures with the same name (but different locations) are allowed in HDevelop (see section 5.5 on page 43).

Password You can optionally protect procedures by a password. Protected procedures may be used in HDevelop programs without restrictions. However, to view and modify them the correct password needs to be provided. See section 5.6 on page 45 for more information.

Type This check box determines the procedure type (see section 5.1 on page 41). By default, a local procedure is created. Local procedures are saved within the HDevelop program. External procedures are saved as stand-alone files. Libraries may contain multiple procedures in a single file.

The file type of external procedures may be specified explicitly (.hdvp or .dvp, see section 5.2 on page 42).

External and library procedures can be reused in other HDevelop programs. You can change the procedure type at any time.

Directory For external procedures a target directory has to be specified.

The first directory specified in the procedure preferences (see section 6.2.2.16 on page 71) is suggested as the target directory. You can select an appropriate directory from the list, or click the browse button to select an arbitrary directory.

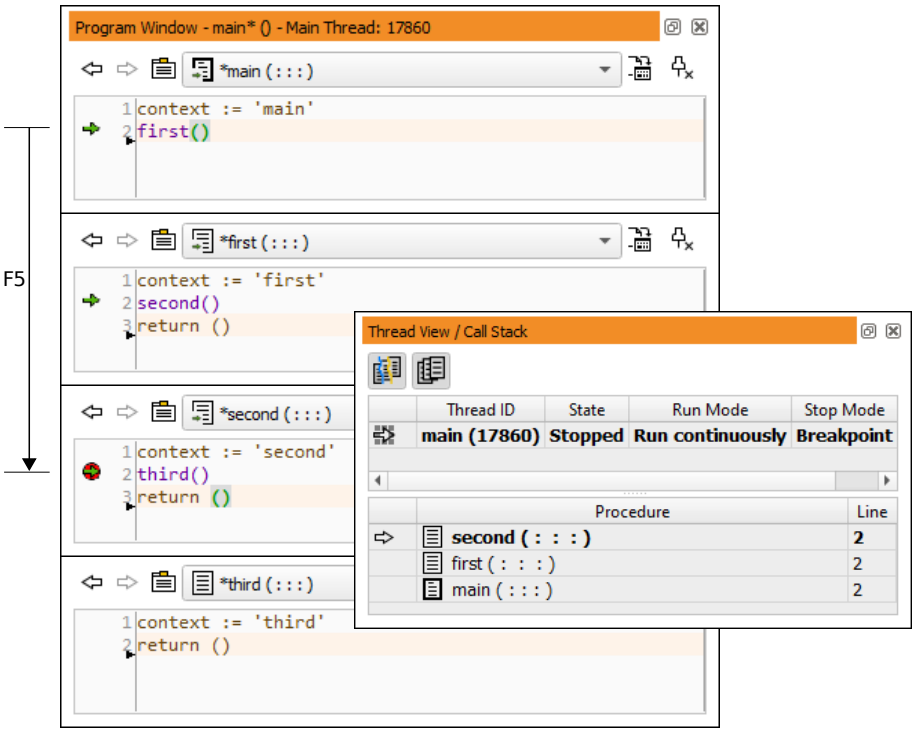


Figure 6.39: PC and call stack.

If the selected directory is not currently contained in the configured directories, HDevelop will ask you if you want to add it to the list when you commit the dialog.

`Library` For library procedures a target library has to be specified. The list contains all currently available libraries. The buttons next to the list allow you to create a new library or browse for an existing library that is

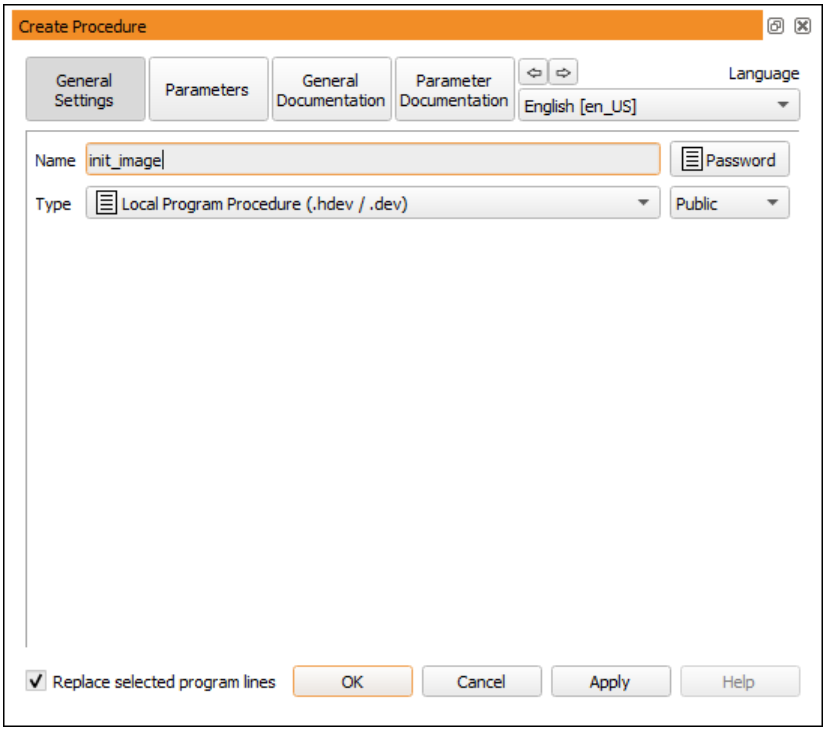


Figure 6.40: Dialog for creating a new procedure.

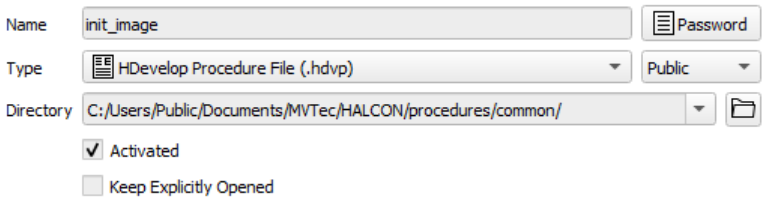


Figure 6.41: Settings for an external procedure.

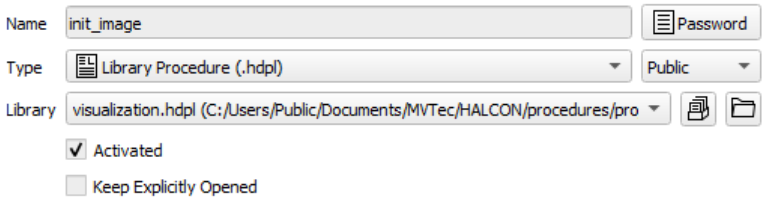


Figure 6.42: Settings for a library procedure..

currently not configured.

Scope (`Public` **or** `Private`) Specifies whether procedures can be called by any procedure (public), or only by procedures in the same directory or file (private) as described in section 5.3 on page 42. The icons of private procedures are decorated with a green dot.

`Activated` This check box determines whether or not the selected external procedure can be resolved. This option can also be toggled in the preferences (see section 6.2.2.16 on page 72).

`Keep Explicitly Opened` **(existing external procedures only)** If checked, the selected procedure will be kept open for editing even if its path is not configured in the preferences. Procedures marked this way can always be selected from the drop-down list of the program window.

`Add Path` **(existing external procedures only)** Add the directory of the selected procedure to the list of procedure directories (see section 6.2.2.16 on page 71).

The addition may either be permanent or for the current session only. This is further explained in section 5.4 on page 43.

6.4.5.2 Setting Up the Procedure Parameters

This part of the dialog is used for the definition of procedure parameters. HDevelop procedure interfaces have the same structure as HALCON operator interfaces, that is, they may contain parameters of the four categories iconic input, iconic output, control input, and control output in this order. The procedure interface dialog contains four separate areas where the different parameters types may be edited. Each area contains a button for appending new parameters to the parameter list.

When creating a new procedure from selected program lines, HDevelop automatically determines suitable interface parameters for the procedure from the usage of the variables in the selected code. The combo box `Selection Scheme` determines the suggestion of the procedure parameters. The meaning of this selection is as follows:

`First In` If the *first use* of a variable *inside the selected lines* is as an input variable, it will be suggested as an input parameter of the procedure.

`Last Out` If the *last use* of a variable *inside the selected lines* is as an output variable, it will be suggested as an output parameter of the procedure.

`All In` All input variables inside the selected lines are suggested as input parameters in the procedure.

`All Out` All output variables inside the selected lines are suggested as output parameters of the procedure.

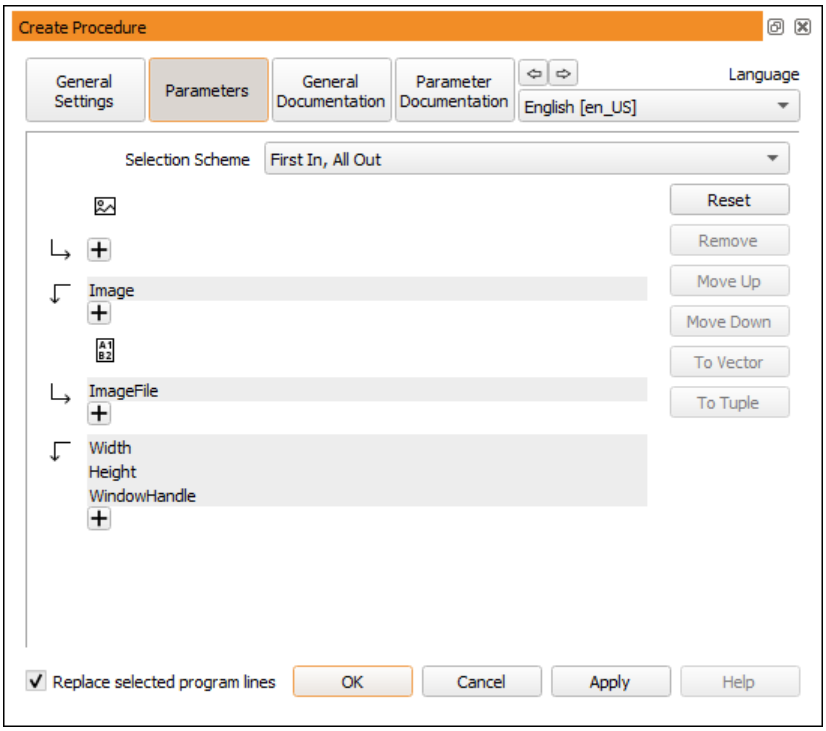


Figure 6.43: Procedure parameters..

The classification of variables in the selected program lines is performed separately for iconic and control variables.

If a variable is an input as well as an output variable, it is assigned to the first category, i.e., the corresponding procedure parameter becomes an input parameter.

If, according to the above rules, a variable name would be suggested as an input as well as an output parameter, it becomes an input parameter of the procedure. In addition, an output parameter with the variable name extended by "`Out`" is created.

As an illustration, the following program lines are selected for a new procedure:

```
threshold (Image, Region, 128, 255)
connection (Region, ConnectedRegions)
```

Then, based on the selection scheme `All In All Out`, the procedure body will read

```
copy_obj (Region, RegionOut, 1, -1)
threshold (Image, RegionOut, 128, 255)
connection (RegionOut, ConnectedRegions)
```

To the right of the parameter list, the following buttons are provided:

`Reset` If you are creating a new procedure, clicking this button removes all entered parameters. If you are editing an existing procedure, the original interface is restored, i.e., any changes to the parameters are undone.

`Remove` Using this button you can remove single parameters from the list. Before clicking this button, focus the corresponding parameter by clicking its text field.

`Move Up`, `Move Down` Using these buttons you can alter the order of the parameters. Select a parameter by clicking its text field and use the buttons to change its position.

To move a parameter from one parameter group to another parameter group, follow these steps:

- Remove the parameter from the current group.
- Click `Apply` to save the current state.
- Add the parameter to the desired group.

Vector Parameters

HDevelop 12.0 and later versions support vector variables. See section "Vectors" on page 303 for a detailed description. Vector variables may be used in procedure calls. The following buttons handle the conversion of parameters to vector and back again. Please note that the dimension of a vector in a procedure call must match the dimension specified in the procedure interface, i.e., if the procedure expects a vector of tuples, it cannot be called by passing a vector of vectors of tuples.

Vector parameters are distinguished from other parameters by showing the contained type and the dimension next to the parameter name, e.g., `object {1}` for a vector of iconic objects, or `tuple {2}` for a vector of vectors of tuples.

`To Vector` Set the type of the selected parameter to vector. If the selected variable is of type vector already, its dimension may be increased with this button (label changes to `Dim +`).

`To Object` / `To Tuple` Set the type of the selected parameter to object (iconic parameters), or to tuple (control parameters). If the selected variable is a multi-dimensional vector, its dimension may be decreased with this button (label changes to `Dim -`).

6.4.5.3 Committing the Procedure Interface

`Replace selected program lines` If this option is enabled, the selected program lines are replaced by an appropriate call to the newly created procedure see figure 6.44 for an illustration. Otherwise, the original program lines are kept and no procedure call is added. In any case, the selected program lines are copied to the body of the new procedure.

`Adapt program` **(existing procedures only)** This setting is not relevant for new procedures, but very useful when modifying the interface of existing procedures. When enabled, all program lines calling the procedure in question will be adapted according to the interface changes, i.e., changes to the procedure name or its parameters.

For example, if you decide that a certain parameter is no longer necessary, the corresponding expressions or variable names will be removed from all procedure calls in the program when you close the dialog and apply the changes. If this is an input parameter, the program will continue to run without further modifications. If it is an output parameter, subsequent program lines relying on the value of that parameter will have to be adjusted manually.

As another example, if a new parameter has become necessary, a variable of the same name will be added to all procedure calls. If this is an input parameter, the corresponding variable will most likely not be initialized at the time of the procedure call and has to be assigned to manually. If it is an output parameter, the program will continue to run without further modifications.

Leaving this feature enabled is highly recommended to keep the program consistent.

`OK` Activating the button `OK` at the bottom of the dialog either creates a new procedure or commits the changes made in the procedure interface, depending on whether the interface dialog was invoked to create a new procedure or to modify an existing procedure. In the latter case not only the interface itself might be changed but also the procedure's program body and variable lists, as new variables might have been added or existing variables might have been removed or renamed.

If you change the interface of an external procedure, be aware of the fact that other programs containing it do not update the procedure calls. When loading these programs, the procedure calls are disabled. If the changes were applied to a procedure that is called from inside a protected external procedure, that procedure call is not even updated in the current program.

`Cancel` This button dismisses the dialog. Any changes to the interface or the documentation of the edited procedure are lost (with the exception of the editing status, see section "Protected Procedures" on page 45).

`Apply` Applies the changes in the dialog (just like pressing `OK`) without closing it.

`Help` Displays the documentation of the selected procedure. If the documentation is empty, the button will be grayed out.

The newly defined procedure is now available for selection in the operator window. The variables that were used to determine the procedure interface parameters are now being offered as input parameters for the procedure call.

Please note that a `return` call has been added at the end of the procedure body. If you create a procedure from scratch, the newly created procedure body will contain only the `return` operator initially.

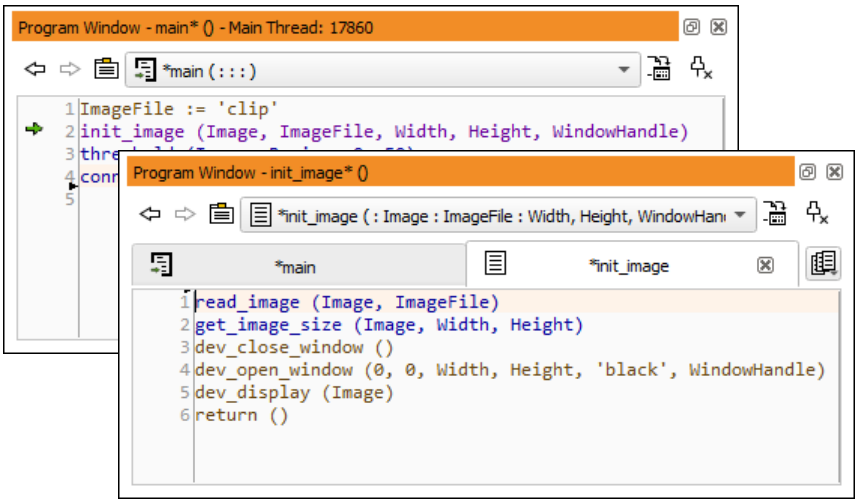


Figure 6.44: Resulting new procedure.

6.4.6 Editing Procedures

The combo box on top of the program window displays the name of the current procedure. You can select all available procedures from this box. The first element of the list will always be the main procedure, followed by the local procedures of the current program, followed by the available external and library procedures. The procedure groups are sorted alphabetically.

Procedures may be protected with a password. Those procedure can still be selected from the list, but unless the correct password has been entered, they will remain in a locked state. If the procedure is locked, a password button is displayed instead of the procedure body. For more information about protected procedures, see section 5.6 on page 45.

To view and modify the interface of the current procedure

- select `Procedures` ▷ `Edit Procedure Interface` (page 100)
- or click the button

Using the upper buttons of this dialog, you can select the data associated with the current procedure: The button `General Settings` provides access to the procedure name, its type and its parameters.

- `General Settings`: See section 6.4.5.1 on page 121
- `Parameters`: section 6.4.5.2 on page 123

The remaining buttons provide access to the documentation of the current procedure. This is described in section 6.4.8 on page 128.

You can step through the individual tab cards of the dialog using the arrow buttons at the bottom of the dialog.

6.4.7 Side Effects of Procedure Changes

Certain changes to procedure properties may have side affects that have be taken care of. For example, if you change the type of an external procedure to “local”, what happens with the original procedure file? Is it kept, or will it be deleted? The procedure dialog lets you choose the appropriate action yourself.

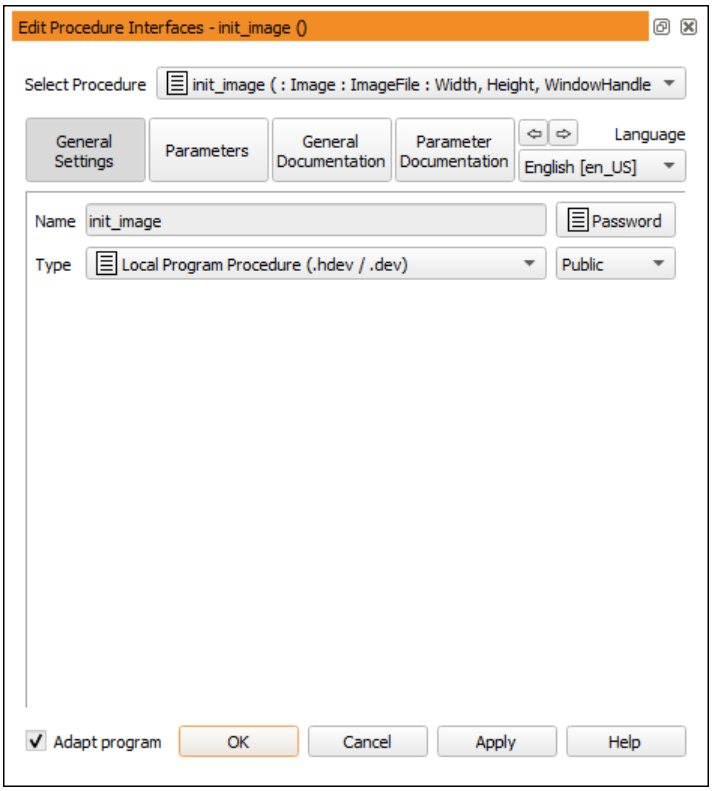


Figure 6.45: Editing a procedure.

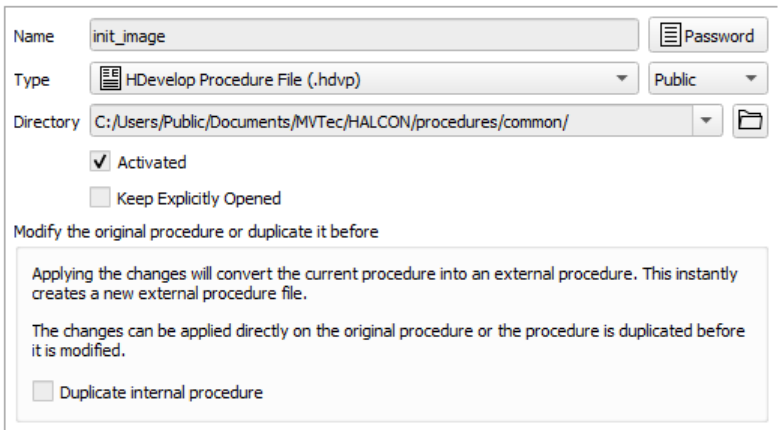


Figure 6.46: Procedure type changed from local to external.

Changing the type or name of a local or library procedure

If the procedure type is changed to “external”, a new external procedure file will created in the specified directory. Changing the type “local” to “library” modifies the library file of the selected target library. Conversely, setting the type of a library procedure to “local” modifies the current program. In all these cases the option `Duplicate internal procedure` determines what happens with the original procedure.

If `Duplicate internal procedure` is unchecked, the local procedure is moved to the new location. Otherwise, the original local procedure is kept unmodified, and the external procedure gets created (along with other changes made in the dialog). In this case the newly created external procedure cannot be called from the program because the internal procedure has a higher priority.

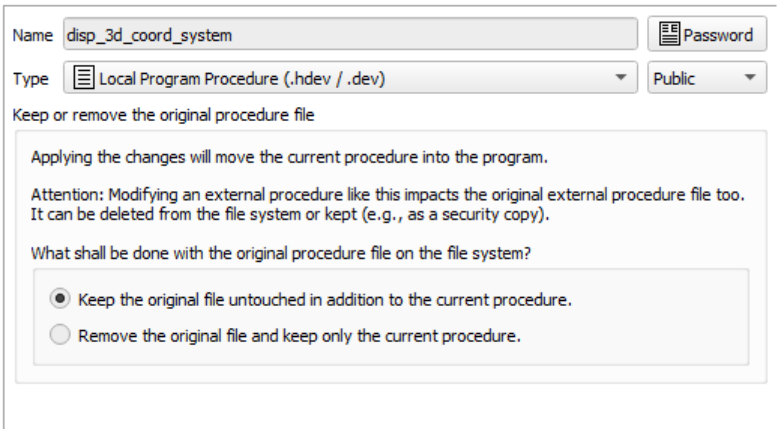


Figure 6.47: Procedure type changed from external to local.

Changing the procedure type or name of external procedures

If the type or name of an external procedure is changed, you will have to decide whether the original file is kept. The following options are available:

- `Keep the original file untouched in addition to the current procedure.`
- `Remove the original file and keep only the current procedure.`

6.4.8 Providing Procedure Documentation

HDevelop supports the preparation of procedure documentation in the procedure interface dialog. Since procedures are treated like operators, the same documentation slots are available. The procedure documentation is seamlessly integrated into the online help system. For example, if you select a procedure in the operator window, clicking the help button will take you to the corresponding page in the help window.

The documentation of the procedure may be entered in multiple languages. The language used for displaying the procedure documentation in the online help depends on the language set in the preferences of HDevelop. To edit the procedure documentation in a specific language, select the corresponding entry from the drop-down list `Language`.

6.4.8.1 `General Documentation`

`Basics`

Procedures can be grouped by `Group` and `Chapters` (chapter and section).

`Group` This is the top level element of the content hierarchy in the procedure online help. It can be used to apply a vendor-specific tag to a group of procedures. The external procedures supplied with HALCON use the group tag “MVTec Standard Procedures”. If no group is specified, the corresponding procedures are listed under “No group assigned (local procedures)” or “No group assigned (external procedures)” depending on their type.

`Chapters` The text fields next to `Chapters` can be used to specify chapter and section, so that your procedures can be grouped thematically in the list at the bottom of the menus Menu `Procedures` and Menu `Operators`.

Note that the logical structure created by the chapter and section information does not correspond to the automatically created directory structure. At least for the external procedures, you can create the corresponding directory structure afterwards, outside of HDevelop. The recognition of the procedures in HDevelop is still ensured, as all subdirectories of the external procedure paths are scanned as well. When editing already existing external procedures, the changed procedures are stored in the paths they were originally found in.

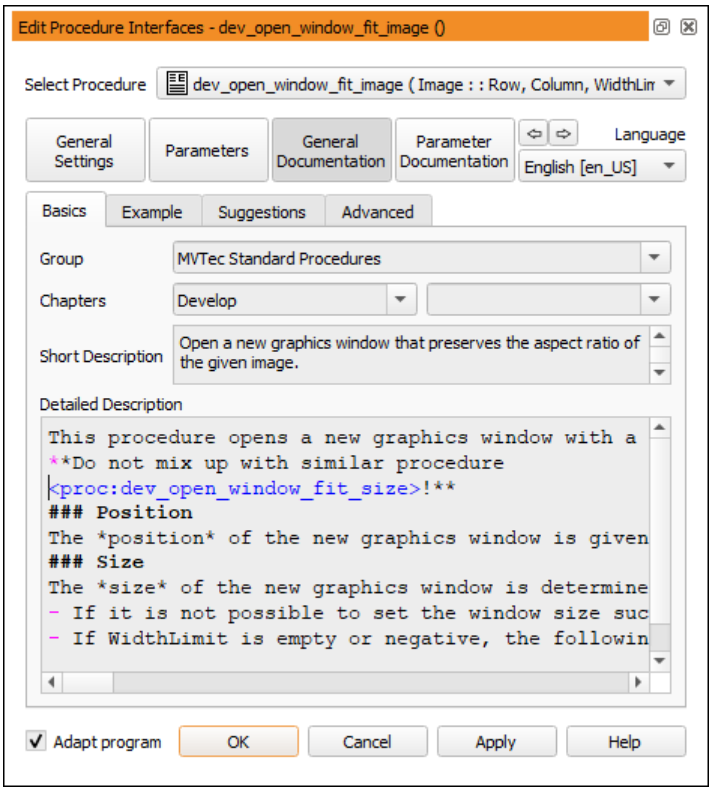


Figure 6.48: Editing the general documentation of a procedure.

`Short Description` Enter a short description. Usually, this should be a single sentence that describes the purpose of the procedure. It appears in the overview sections of the online help of the procedures. Additionally, the short description is displayed in HDevelop's status bar when the procedure is selected from the menu.

`Detailed Description` Enter a detailed description of the procedure using Markdown syntax. For more information about the syntax, see section 6.4.8.1 on page 130.

`Example`

This section of the documentation is intended for code examples. This could be a working program or some code fragments that illustrate the usage of the procedure.

`Suggestions`

The first field on this tab card allows you to associate keywords with the procedure. Enter a comma-separated list of keywords into this field. The tab card `Index` of the online help may be used as a reference for keyword suggestions.

Furthermore, you can specify suggested successors, predecessors, and alternatives to the current procedure. Enter comma-separated lists of operator or procedure names into the fields. See Menu `Suggestions` for the meaning of these fields.

`Advanced`

The text boxes in this tab card are for advanced usage only. We recommend searching the online reference manual for usage examples.

`Attention` Notes about special observances when using the procedure. Supports Markdown syntax (see section 6.4.8.1).

`Complexity` Notes about intricate details about the procedure usage. Supports Markdown syntax (see section 6.4.8.1).

`Warning` Usually used to indicate obsolete or deprecated procedures that are kept for backward compatibility. The warning text should indicate the recommended alternative.

A warning icon is displayed in the left column of the program window. Additionally, if the procedure is selected, the warning text will be displayed in the operator window as a reminder.

Supports Markdown syntax (see section 6.4.8.1).

`References` Bibliographic references with recommended reading about certain aspects of the procedure. Supports Markdown syntax (see section 6.4.8.1).

Using Markdown to Format Text

Text entered in the following fields is formatted as *Github Flavored Markdown:*
`Detailed Description`, `Attention`, `Complexity`, `Warning`, `References`

For detailed information about this syntax, see `https://github.github.com/gfm`. See the list below for examples of common formatting cases.

Inline text formats Emphasize text with `*italic*` (*italic*) or make it strong with `**bold**` (**bold**). `~strikethrough~` is also possible.

Code blocks Insert a code block (i.e., a block of monospaced text) by preceding the lines with four blanks. Alternatively, use three backticks:

```
```
my code
```
```

Lists Insert an ordered list like this:

```
1. first item
2. second item
```

Insert an unordered list like this:

```
* an item
* another item
  * a nested item
```

Tables Insert a simple table like this:

```
| header col 1 | header col 2 |
|--------------|--------------|
| col 1 row 2  | col 2 row 2  |
| col 1 row 3  | col 2 row 3  |
```

Links Insert a link like this: `[link text](URL)`
E.g., `[This is a link](https://example.org)`. If you omit the protocol, the URL is interpreted relative to the file containing the procedure.

Images Insert an image like this: `![alt text](path/to/image)`
Images are referenced relative to the file containing the procedure. They cannot be stored within the procedure file.

References To reference other procedures, operators, or table of contents, use the autolink syntax:

- Procedures: `<proc:procedure_name>` (e.g., `<proc:train_dl_model>`)
- Operators: `<op:operator_name>` (e.g., `<op:train_dl_model_batch>`)
- Table of contents: `<toc:chapter_name>` (e.g., `<toc:deeplearning_anomalydetection>`)

In the `Detailed Description` field, syntax-highlighting shows whether a reference can be resolved.

The list of available procedures is loaded during the start of the Procedure Interface dialog and only updated when the dialog is started again. Therefore, features like syntax highlighting will not work for other procedures if they are created or changed while the dialog is open.

6.4.8.2 `Parameter Documentation`

This section of the dialog provides tab cards for all parameters of the current procedure. The documentation consists of a fine-grained specification of the parameters, and a short description. The specification fields depend on the parameter type (iconic or control parameter), and on the selected semantics. In the following, the most common fields of both iconic and control parameters are listed.

Please refer to the Extension Package Programmer's Manual (Chapter 2.3) for additional information about the documentation fields (especially, the semantic types).

Iconic Parameter Documentation

Field	Meaning
`Semantics`	Specifies the semantic type of the parameter.
`Pixel Types`	Only available for `Semantics` *image*. Lists the accepted pixel types. The buttons `Select All` and `None` toggle the selection of all parameters.
`Multi Channel`	Only available if `Semantics` = *image*. *False*: Only the first channel of the image is processed, *True*: Only a multi-channel image is accepted, *Optional*: Images with an arbitrary number of channels are accepted.
`Multi Value`	*False*: Only a single object (no object tuple) is accepted, *True*: Only object tuples are accepted, *Optional*: A single object as well as an object tuple is accepted.
`Description`	Short description of the iconic parameter.

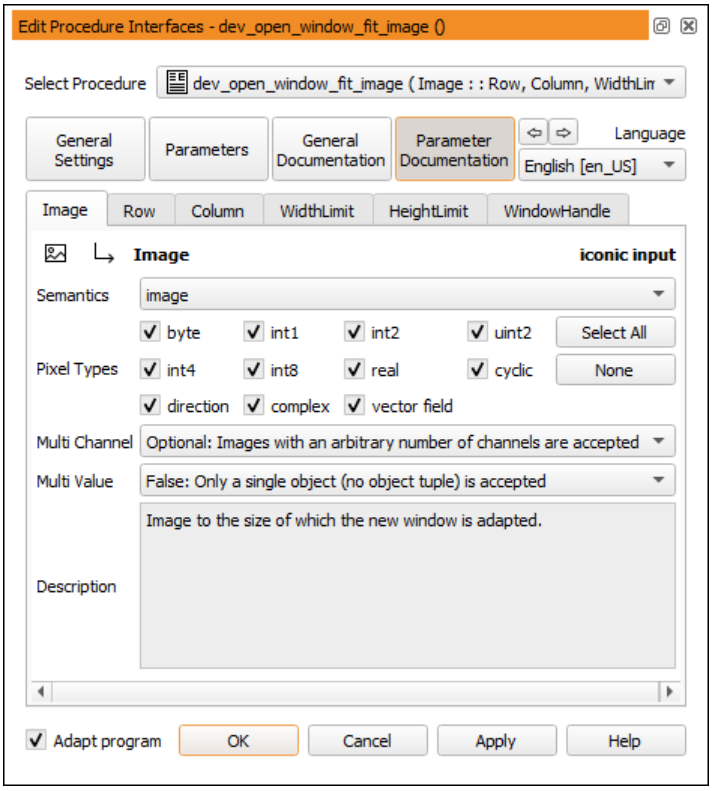


Figure 6.49: Editing the iconic parameter documentation of a procedure.

Control Parameter Documentation

Field	Meaning
`Semantics`	Specifies the semantic type of the parameter. For some semantic types, additional subtypes may be selected. The semantic types are split into three groups which are each sorted alphabetically. The first group contains the basic types, the second group contains complex types with additional semantics, and the third group contains handles. The groups are separated by a dividing line.
`Type List`	Specifies the accepted data types.
`Default Type`	Specifies the default data type.
`Mixed Types`	*False*: All values of a tuple have the same type, *True*: Values of different types can be mixed in one tuple.
`Default Value`	The entered value is suggested as the default value by HDevelop.
`Values`	Comma-separated list of suggested values. Check `Exclusively` to restrict the selection to the specified values.
`Value Min`	Minimum value for numeric control data. Check `Enabled` to enforce this setting.
`Value Max`	Maximum value for numeric control data. Check `Enabled` to enforce this setting.
`Multi Value`	*False*: The parameter accepts only a single value, *True*: The parameter always expects a tuple of values, *Optional*: Single values as well as tuple values are accepted.
`Description`	Short description of the control parameter.

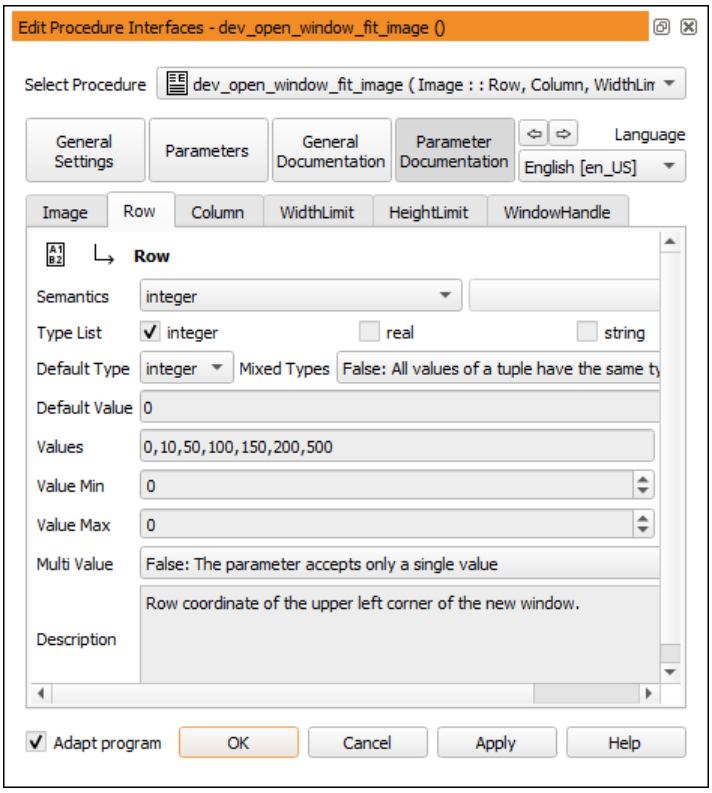


Figure 6.50: Editing the control parameter documentation of a procedure.

6.4.9 Protecting a Procedure

The concept of protected procedures is explained in section 5.6 on page 45. The status of a procedure can be changed via the procedure interface dialog (page 121). To manage the status of multiple procedures at once, click Menu `Procedures` ▷ `Manage Procedures` and select the tab card `Manage Passwords` (page 74).

If you want to protect a procedure with a password, do the following:

- Select the corresponding procedure in the program window.
- Click to edit the interface of the selected procedure.
- Click the button `Password` to assign a password to the procedure.

Then, a separate window appears and the new password must be entered twice (see figure 6.51). If both times the same password is used, clicking `OK` assigns the password. Otherwise, an error message is displayed and you have to repeat the password assignment. When a protected procedure is finally saved, it is stored in a binary format.

If you set up a password for the *main* procedure, you may optionally lock the entire program, i.e., protect all local procedures with the same password. The same mechanism works for library procedures: You may either protect the library procedures individually, or protect the entire library at once (see page 74). If individual local procedures have been password-protected before, this option will only work, if the same password is selected. Otherwise you will have to remove all other passwords from local procedures before locking the entire program.

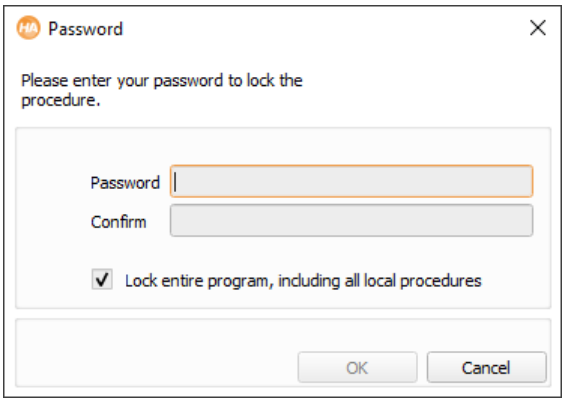


Figure 6.51: Entering a password to protect a procedure. The option “Lock entire...” is only available for the main() procedure.

When you start HDevelop the next time, the protected procedure is locked, i.e., when trying to edit the procedure, e.g., by selecting it from the combo box in the program window, a corresponding message is displayed in the program window see figure 6.52). Additionally, a password button is displayed in the program window. Upon entering the correct password, the procedure remains unlocked for this session, i.e., until you close HDevelop or lock the procedure again manually.

Changing the Status of a Protected Procedure

To change the status of a protected procedure, you must first unlock it by entering the password. Then, you can use the procedure interface dialog (page 121) to change the password or remove the password entirely. Click the button `Password` to change the status.

The following options are available:

`Lock` You can lock the protected procedure so its body cannot be accessed in the current session without supplying the password again.

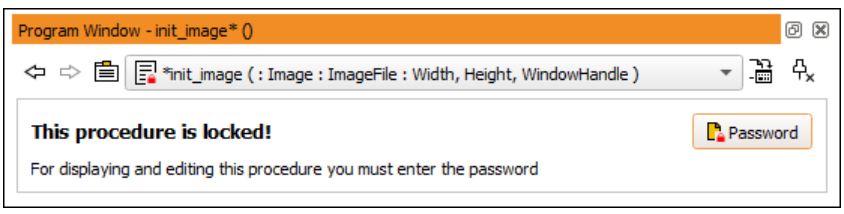


Figure 6.52: A locked procedure.

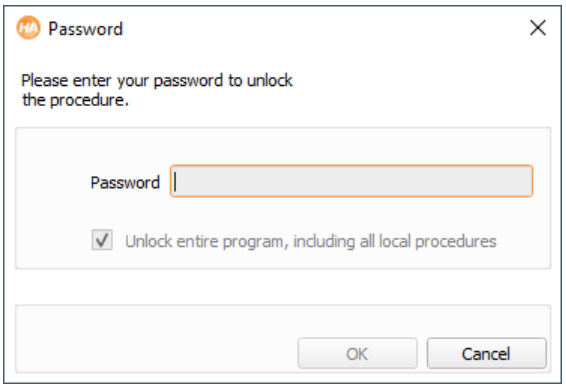


Figure 6.53: Changing the status of a protected procedure. The option “Unlock entire...” is only available if local procedures have been locked at once.

`Remove` Selecting this option removes the password. When the procedure is saved, it is no longer protected.

`New password` The password window is displayed and you can assign the new password by the same process you used for the old one.

`Cancel` The operation is canceled without altering the status.

Warning

When working with protected procedures, be aware that the password cannot be reconstructed, so be very careful not to forget it and not to repeat a typing error when assigning it! Further, in some situations protected procedures behave differently from common procedures. In particular, as they cannot be viewed and modified by unauthorized users, they also cannot be copied, printed, or exported to any programming language (however, they can be duplicated using the menu entry Menu `Procedures` ▷ `Duplicate...`). Additionally, if a protected procedure contains a call to another procedure for which the interface was changed, the procedure call is not adapted to the changes but is disabled for the current program.

6.4.10 Profiler

The built-in profiler analyzes the runtime behavior of HDevelop programs. It counts operator and procedure calls, and measures the processing times of operator calls (referred to as *operator time* in the following). The operator time plus the additional overhead of each operator call inside HDevelop is measured as well (referred to as *execution time* in the following). The operator time is the appropriate measure if you plan to export your HDevelop program to a programming language. On the other hand, the execution time is the appropriate measure if you run the program inside HDevelop or HDevEngine.

The profiling data may be used to evaluate the overall program execution, to optimize its performance, and to find bottlenecks.

Keep the following in mind when profiling your code:

- Disable GUI updates by calling the convenience procedure *dev_update_off()* at the beginning of your program.
- Run the code to be profiled a couple of times and use the average profiling data to get more accurate results.
- Reduce the number of concurrent (background) processes.
- Avoid `stop` and `wait_seconds` instructions, or override these two operators (see `Override Operator Behavior` (page 81)).

To activate the profiler, click `Execute` ▷ `Activate Profiler` (page 88).

Once the profiler is activated, each program execution collects profiling data.

The profiler data can be reset by clicking `Execute` ▷ `Reset Profiler` (page 88),

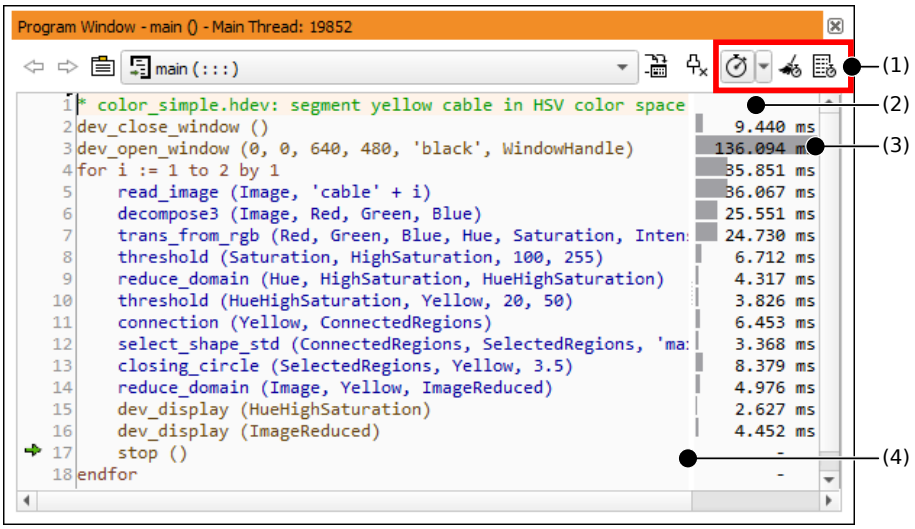


Figure 6.54: Profiling results of the first program execution.

To deactivate the profiler, click `Execute` ▷ `Deactivate Profiler` (page 88).

The profiling results are displayed in the program window. In addition, a summarized view of the runtime statistics is also available.

6.4.10.1 Profiler Display

To illustrate the way the profiler works, the HDevelop program *solution_guide/basics/color_simple.hdev* is loaded and executed (see figure 6.54). The profiling data is displayed in a separate column (2) of the program window. The display area can be resized by dragging its left edge (4).

By default, the total execution time of each program line is displayed. The gray bars illustrate the execution times in relation to the largest value (3). If the profiler is enabled, the tool bar of the program window contains some additional profiler-related buttons (1).

The options described below are available in the drop-down menu of the toolbar icon or the context menu of the profiler area.

`Activate/Deactivate Display` / `Activate Only selected` See section “Selective Profiler Display” on page 137.

`Number of Calls` Displays how many times each program line has been executed in total. This value is accumulative until the profiler is reset.

`Total Execution Time` Displays the total processing time of each program line. This value is accumulative until the profiler is reset.

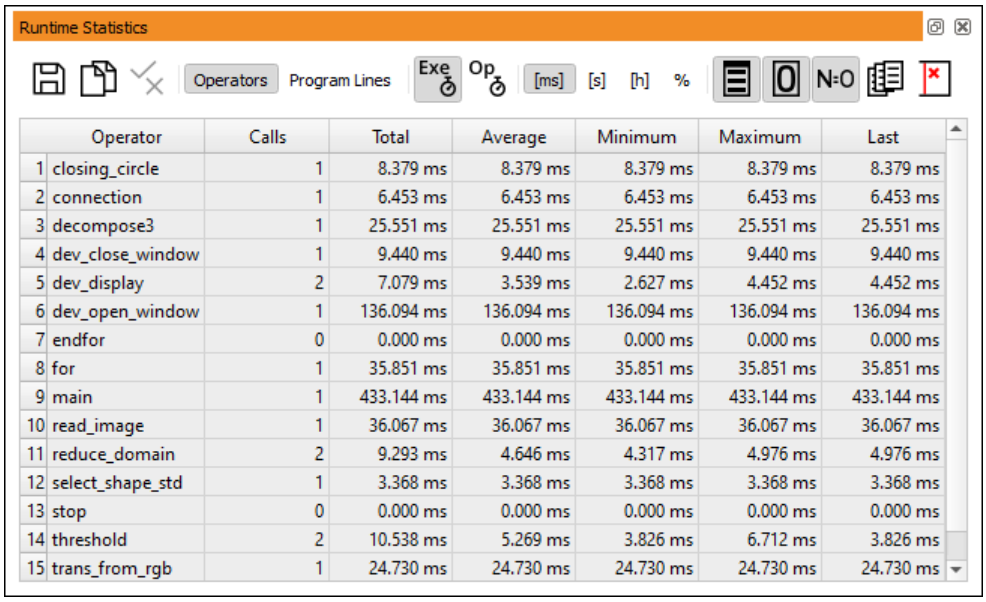
`Average Execution Time` Displays the average processing time of each program line. This value differs from the total processing time if the corresponding program line has been executed more than once, e.g., in a loop. The average processing time is more meaningful if the program is reset and run multiple times.

`Minimum Execution Time` Displays the minimum processing time of each program line. This value is only meaningful if the corresponding program line has been executed more than once, e.g., in a loop.

`Maximum Execution Time` Displays the maximum processing time of each program line. This value is only meaningful if the corresponding program line has been executed more than once, e.g., in a loop.

`Last Execution Time` Displays the processing time of the last execution of each program line.

Note that the processing time is displayed in one of two modes as explained in section 6.4.10 on page 134, depending on the setting in the context menu:



	Operator	Calls	Total	Average	Minimum	Maximum	Last
1	closing_circle	1	8.379 ms	8.379 ms	8.379 ms	8.379 ms	8.379 ms
2	connection	1	6.453 ms	6.453 ms	6.453 ms	6.453 ms	6.453 ms
3	decompose3	1	25.551 ms	25.551 ms	25.551 ms	25.551 ms	25.551 ms
4	dev_close_window	1	9.440 ms	9.440 ms	9.440 ms	9.440 ms	9.440 ms
5	dev_display	2	7.079 ms	3.539 ms	2.627 ms	4.452 ms	4.452 ms
6	dev_open_window	1	136.094 ms	136.094 ms	136.094 ms	136.094 ms	136.094 ms
7	endfor	0	0.000 ms	0.000 ms	0.000 ms	0.000 ms	0.000 ms
8	for	1	35.851 ms	35.851 ms	35.851 ms	35.851 ms	35.851 ms
9	main	1	433.144 ms	433.144 ms	433.144 ms	433.144 ms	433.144 ms
10	read_image	1	36.067 ms	36.067 ms	36.067 ms	36.067 ms	36.067 ms
11	reduce_domain	2	9.293 ms	4.646 ms	4.317 ms	4.976 ms	4.976 ms
12	select_shape_std	1	3.368 ms	3.368 ms	3.368 ms	3.368 ms	3.368 ms
13	stop	0	0.000 ms	0.000 ms	0.000 ms	0.000 ms	0.000 ms
14	threshold	2	10.538 ms	5.269 ms	3.826 ms	6.712 ms	3.826 ms
15	trans_from_rgb	1	24.730 ms	24.730 ms	24.730 ms	24.730 ms	24.730 ms

Figure 6.55: Runtime statistics.

`Execution Time` Display processing times as "execution time", i.e., the raw time of an operator call the additional overhead inside HDevelop.

`Operator Time` Display processing times as "operator time".

The display can be toggled between duration values and percentages:

`Time` Display the processing times as absolute values. The unit of measure defaults to ms and is adjusted appropriately, e.g., it switches to seconds once the value exceeds 1000ms. To specify the unit of measure explicitly, switch to the runtime statistics window described in section 6.4.10.2.

`Percentage` Display the processing times as percentages. 100% refers to the accumulated times *inside* main, or the currently displayed procedure, respectively.

Reset profiler values.

Open runtime statistics (see section 6.4.10.2).

6.4.10.2 Runtime Statistics

This window displays the accumulated profiling data of the procedure that is currently displayed in the active program window (or all procedures, see below). The table rows can be sorted by clicking the corresponding headers. Two different display modes are available:

`Operators` In this mode, the profiling data is displayed per operator/procedure call, i.e., multiple calls of the same operator/procedure are summarized.

`Program Lines` In this mode, the profiling data is displayed per program line.

Save the runtime statistics as a plain text file (*.txt* or *.csv*). Each line contains a table row, and the columns are separated by tabs.

Copy the selected entries to the clipboard. If no entries are selected, the entire table is copied.

Toggle the display status of the selected profiler lines (see section "Selective Profiler Display" on page 137). This option is only available if the display is set to "`Program Lines`".

Exe Show execution time.

Op Show operator time.

ms Display times in milliseconds.

s Display times in seconds.

h Display times in hours.

% Display values as percentages. If percentages are selected, the values refer to the accumulated times of the entire program, i.e., the run-time of *main* corresponds to 100%.

Include runtime statistics of procedures in display.

Include runtime statistics of operators in display.

N=0 Include program lines with zero calls.

Usually, the runtime statistics window displays the profiler data of the procedure that is currently displayed in the active program window. Turn this button on to show the profiling data of all procedures.

In general, deactivated profiler lines are not listed in the runtime statistics window (see section "Selective Profiler Display" on page 137). If this button is on, deactivated profiler lines will be displayed in light gray so that they can be activated again.

6.4.10.3 Selective Profiler Display

The profiler collects data for *all executed* program lines. In general, the collected data is also displayed in the program window and the runtime statistics window. This behavior is not always desirable. In many cases, the runtime statistics of only a small portion of the program code is relevant when evaluating its performance. Accordingly, the display of profiler data in both the program window and the runtime statistics window can be restricted to a selection of profiler lines.

Profiler lines can be selected in three different ways:

- Drag over a range of lines in the profiler area (see figure 6.56).
- Click a single line in the profiler area, and Shift click another line to select the range between those lines.
- Click the first line in the profiler area, and then Ctrl click additional lines to select a non-contiguous selection of lines.

In all cases, the selection is also highlighted in the left part of the program window until the mouse button is released. This visualization supports you in selecting the desired program lines, which can be difficult if the program window is very wide.

The values of the selected profiler lines are summed up and displayed in the status bar as shown in figure 6.57.

To display the profiler data of only the selected lines, click `Activate Only selected` in the context menu of the profiler area. The result is shown in figure 6.58. Note that the gray bars are automatically adjusted to the selected data.

To toggle the status of specific profiler lines, select the corresponding lines as described above and click `Activate/Deactivate Display` in the context menu of the profiler area.

The runtime statistics window is updated to list only activated program lines as shown in figure 6.59 on page 139. To remove additional lines from this window, select the corresponding lines and click the button. Deactivated lines can be re-displayed in light gray if the button is turned on.

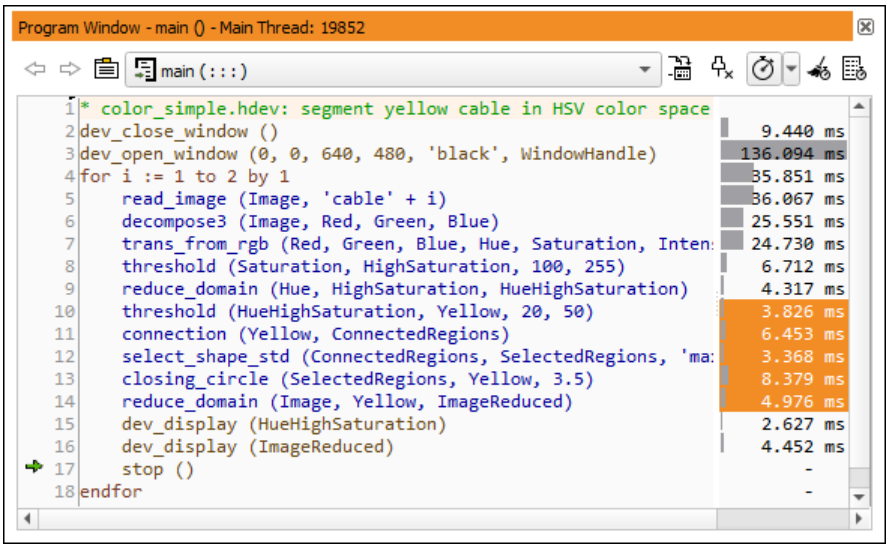


Figure 6.56: Selecting program lines in the profiler.

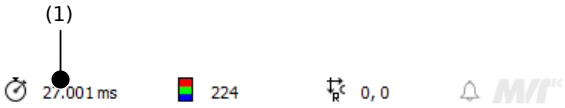


Figure 6.57: Selected values are summed up (1) in the status bar.

6.5 Operator Window

This window is used to edit and display an operator or procedure call with all its parameters. Here you will obtain information about the number of the parameters of the operator or procedure, the parameter types, and parameter values. You can modify the parameter values according to your image processing tasks. For this you may use the values proposed by HDevelop or specify your own values.

The operator window consists of the following four parts:

- At the top you find the operator name field, with which you can select operators or procedures.

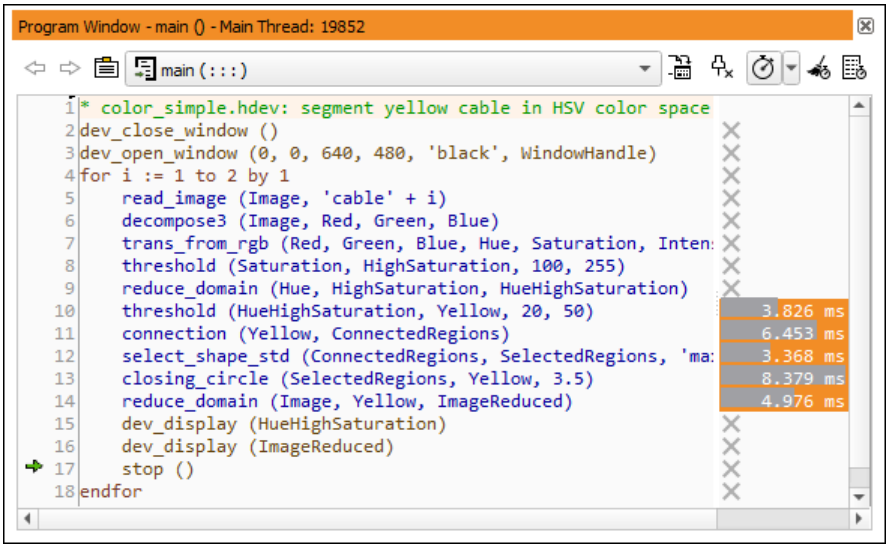
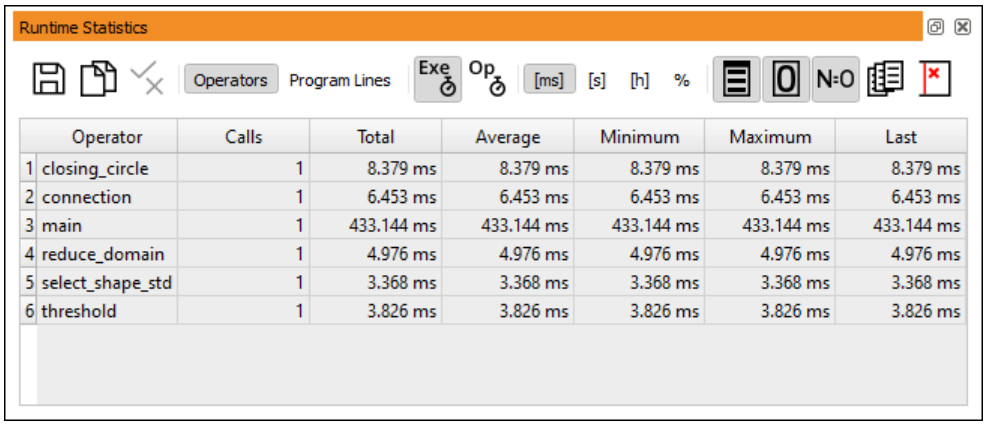


Figure 6.58: Activate selected program lines in the profiler.



	Operator	Calls	Total	Average	Minimum	Maximum	Last
1	closing_circle	1	8.379 ms	8.379 ms	8.379 ms	8.379 ms	8.379 ms
2	connection	1	6.453 ms	6.453 ms	6.453 ms	6.453 ms	6.453 ms
3	main	1	433.144 ms	433.144 ms	433.144 ms	433.144 ms	433.144 ms
4	reduce_domain	1	4.976 ms	4.976 ms	4.976 ms	4.976 ms	4.976 ms
5	select_shape_std	1	3.368 ms	3.368 ms	3.368 ms	3.368 ms	3.368 ms
6	threshold	1	3.826 ms	3.826 ms	3.826 ms	3.826 ms	3.826 ms

Figure 6.59: Runtime statistics of selected program lines.

- The large area below the operator name field is called the parameter display; it is used to edit the parameters of an operator or procedure.
- The section labeled `Advanced Parallelization Options` allows you to call operators or procedures as a subthread (see section “Starting a Subthread” on page 313).
- The row of buttons at the bottom allows you to control the parameter display.

6.5.1 Operator Name Field

The operator name field allows you to select operators or procedures by entering (part of) their name. After pressing Return or pressing the button of the combo box, the system is looking for all operators or procedures that contain the entered name. The order of the listed result is as follows: Operators and procedures whose names begin with the given substring are listed first, followed by all operators and procedures that contain the substring elsewhere. Both parts of the list are sorted in alphabetical order.

If there is an unambiguous search result, the parameters are displayed immediately in the operator window. If there are several matching results, a combo box opens and displays all operators or procedures containing the specified substring. By clicking the left mouse button you select one operator and the combo box disappears. Now, the operator’s parameters are shown in the operator window.

The short description of the selected operator is displayed in the status bar. The operator name is displayed in the window title of the operator window.

6.5.2 Parameter Display

The parameter display is the main part of the operator window. If you have selected an operator or procedure call, HDevelop displays its interface, i.e., the name, value, and semantic type of each parameter.

- In the first column of the parameter display the parameter types are indicated by icons. Note that icons are not repeated if a parameter is of the same type as its predecessor. Hover the mouse cursor over the icons to get a tool tip with the short description of the parameter.
- In the second column of the operator window you find the parameter names.
- The third column consists of the text fields, which contain variable names in case of iconic and control output parameters and expressions in case of control input parameters. If you want to change the suggestions offered by the system (variable names or default values), you may do so either manually or by clicking the arrow button connected with the respective text field. This opens a list containing a selection of already defined variables and other reasonable values from the operator knowledge base. By clicking the appropriate item, you set the text field and the list disappears.

  For the operators `open_framegrabber`, `set_framegrabber_param`, and `get_framegrabber_param`, the value list of certain parameters is dynamic: It depends on the selected image acquisition interface. An

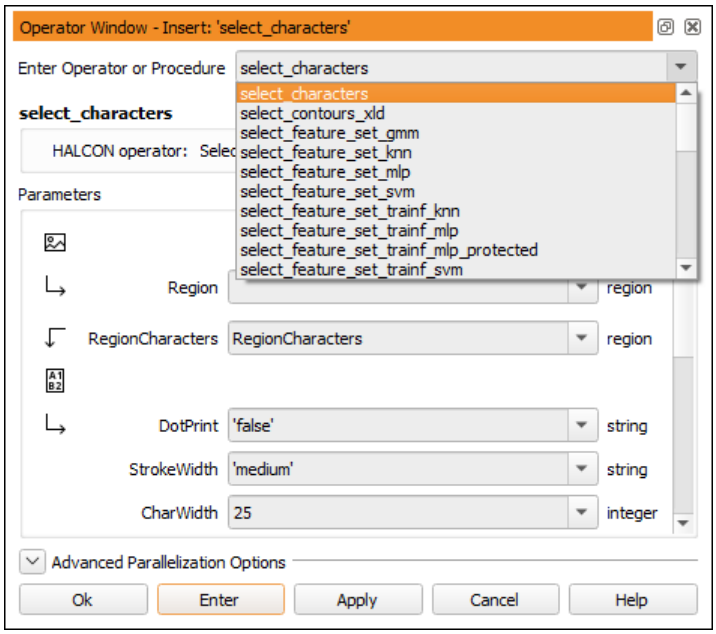


Figure 6.60: Selecting an operator after typing `select_`.

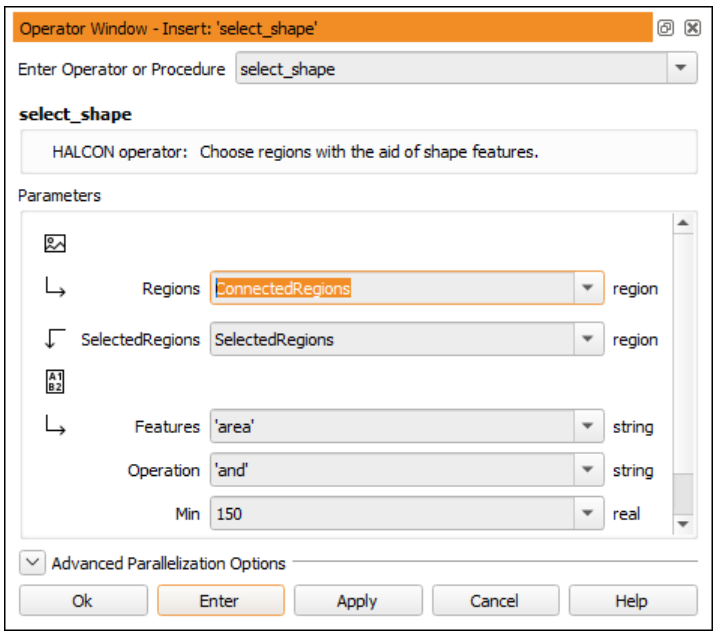


Figure 6.61: Specifying parameters for the operator `select_shape`.

even more reasonable parameter suggestion is given if the corresponding handle is opened. If this dynamic behavior is undesired, it can be disabled in the preferences, see `General Options` ▷ `Experienced User` settings on page 79.

This column may also contain action buttons for special semantic types, e.g., a button to browse the file system for the parameters that expect a file name.

- The fourth column indicates the parameter's default semantic type and, optionally, its data type in parentheses.

Hover the mouse cursor over the second to fourth column to get a short description for the corresponding parameter as a tool tip.

Please refer to the following rules on how parameters obtain their values and how you may specify them:

Iconic input parameters Possible inputs for these parameters are iconic variables of the corresponding type. If there is no need to execute the operator or procedure call immediately, you may specify new variable names, i.e., names, that do not already exist in the variable window, but will be instantiated later by adding further operators or procedure calls to the program body. In any case, you have to specify iconic parameters exclusively with variable names. It is not possible to use expressions.

Iconic output parameters These parameters contain default variables, which have the same names as the parameters themselves. If a variable with the same name as the output parameter is already instantiated, a number is added to the name to make it unique. Because the parameter names characterize the computed result very well, you may adopt these default names in many cases. Besides this, you are free to choose arbitrary names either by yourself or by opening the list (see above). If you use a variable that already has a value, this value is overwritten with the new results. It is possible to specify a variable both in an input and output position.

Control input parameters These parameters normally possess a default value. As an alternative, you may use the text field's button to open a combo box and to select a suggested value. In addition, this combo box contains a list of variables that contain values of the required type. A restriction of proposed variables is especially used for parameters that contain data like file, image acquisition, or OCR handles.

Input control parameters may contain constants, variables, and expressions. Common types are integer numbers (`integer`), floating-point numbers (`real`), boolean values (`true` and `false`), character strings (`string`), and handles (`handle`).

You can also specify multiple values of these types at once by using *tuples*. This is an array of values, separated by commas and enclosed in square brackets. Furthermore, you may build up expressions with these values. You may use expressions in HDevelop similar to the use of expressions in C or in Pascal. You will find a detailed description in section "Expressions for Input Control Parameters" on page 286.

Control output parameters: These parameters are handled in the same way as iconic output parameters. Their defaults are named as their parameter names. Other possibilities to obtain a control output variable name are either using the combo box or specifying variable names manually. You cannot use any expressions for these parameters.

After discussing what can be input for different parameters, it is explained *how* this is done. Nevertheless, you have to keep in mind that you need to modify a parameter only if it contains no values or if you are not satisfied with the suggested default values.

Text input: Give the input focus to a parameter field by clicking into it. Now, you may input numbers, strings, expressions, or variables. There are some editing functions to help you doing input: Backspace deletes the character to the left and Delete deletes the one to the right. You may also select a sequence of characters in the text field using the mouse or holding Shift and using the cursor keys. If there is a succeeding input, the marked region is going to be deleted first and afterwards the characters are going to be written in the text field. See page 355 for a summary of the keyboard mappings.

Combo box selection: Using this input method, you can obtain rapid settings of variables and constants. To do so, you have to click the button on the text field's right side. A combo box is opened, in which you may select an item. Thus, you are able to choose a certain variable or value without risking erroneous typing. Previous entries are deleted. Afterwards, the combo box is closed. If there are no variables or appropriate values, the combo box remains closed.

6.5.3 Control Buttons

Below the parameter display, you find five buttons that comprise the following functions:

`OK` By clicking `OK` you execute the operator or procedure call with the specified parameters. When doing so, the execution mode depends on the position of the PC: If the PC is placed above the insertion position, the system executes the program from the PC until the insertion position first. *Then*, the operator or procedure call that has been edited in the operator window is executed. The reason for this is that the parameter values that are used as input values for the currently edited operator or procedure call have to be calculated. If the PC is placed at or after the insertion position, only the currently edited operator or procedure call is executed.

The operator or procedure call is entered into the program window before it is executed. After the execution, the PC is positioned on the next executable program line after the edited operator or procedure call.

The computed output parameter values are displayed in the variable window. Iconic variables are shown in the current graphics window if you haven't suppressed this option (compare section "`Runtime Settings` ▷ `Runtime Settings`" on page 80). Afterwards, the operator window is cleared. If you did not specify all parameters or if you used wrong values, an error dialog is raised and execution is canceled. In this case, the operator window remains open to allow appropriate changes.

`Enter` / `Replace` By clicking the button `Enter`, the currently edited operator or procedure call is transferred into the program window without being executed. When editing existing program lines (through double-clicking in the program window, see page 115), the button label changes to `Replace`. When clicked, the original program line is replaced.

`Apply` If you click `Apply`, the operator is executed with the specified parameters, but not entered into or changed in the program. This enables you to determine the optimum parameters rapidly since the operator dialog remains open, and hence you can change parameters quickly. Note that this functionality is not available for procedure calls, and thus the button is grayed out in this case.

Unlike the button `OK`, only the single line you edit or enter is executed, no matter where the PC is located. Thus, you have to ensure that all the input variables contain meaningful values. By clicking `Apply`, the corresponding output variables are changed or created, if necessary, to allow you to inspect their values.

`Cancel` Clicking `Cancel` clears the contents of the operator window. Thus, there are neither changes in the program nor in any variables.

`Help` Clicking `Help` invokes the online help for the selected operator or procedure. For this the system activates the online help window (see Help Window).

6.6 Variable Window

There are two kinds of variables in HALCON, corresponding to the two parameter types of HALCON: iconic objects (images, regions, and XLDs) and control data (numbers, strings, handles). The corresponding variables are called iconic and control variables. These variables may possess a value or may be undefined. An undefined variable is created, for example, when loading a program or after inserting an operator with a new variable that is not executed immediately into a program. You may access these undefined variables only by writing to them. If you try to read such a variable, a runtime error occurs. If a variable obtains a value, the variable type is specified more precisely. A control variable that contains, for example, an integer is of type `integer`. This type might change to `real` or a tuple of `integer` after specifying new values for this variable. But it always remains a control variable. Similarly, this is the case for iconic variables, which may contain regions, images, or XLDs. You may assign new values to an iconic variable as often as you want to, but you cannot change its type so that it becomes a control variable.

New variables are created in the operator dialog area during specification of operator or procedure call parameters. Here, every sequence of characters without single quotation marks is interpreted as a variable name. If this name did not exist before, the variable is created in the operator dialog area by pressing `OK` or `Enter`. The variable type is specified through the type of the parameter where it was used for the first time: Variables that correspond to an iconic object parameter create an iconic variable; variables for a control parameter create a control variable. Every time an operator or procedure call is executed, the results are stored in variables connected to its output parameters. This is achieved by first deleting the contents of the variable and then assigning the new value to it.

The variable window is similar to a watch window used in window-oriented debuggers. Inside this window you are able to keep track of variable values. Corresponding to the two variable types, there are two areas in the variable window. One for iconic data (above or left) and the other for control data (below or right).

All computed variables are displayed showing their iconic or control values (unless the automatic update has been switched off, see section "`Runtime Settings` ▷ `Runtime Settings`" on page 80). In case of a tuple result which is too long, the tuple presentation is shortened, indicated by three dots. In this case the full content of a variable can be displayed in an inspection window by double-clicking the value list. See also the following sections.

Breakpoints on Variables

In addition to line-based breakpoints (see section "Program Window" on page 113), breakpoints can also be set on variables. Simply right-click on a variable name and select `Set/Clear Breakpoint on Variable`. Variables with breakpoints are marked with ● next to their name.

Once a breakpoint has been set on a variable, the program execution stops whenever its value changes. A change of value is detected in the same way the HDevelop operation `!=` works (see section "Comparison Operations" on page 297):

- Iconic objects: a value change is triggered if the variable refers to a different object in the database (see also `test_equal_obj`).
- Tuples: A change of the data type (e.g., integer to real) does not necessarily trigger a value change. In the following example the second assignment would not trigger a variable breakpoint on `tuple`:

```
tuple := [1,2,3]
tuple[1] := 2.0     // 2 == 2.0
```

Using `clear_obj` does not trigger a variable breakpoint. Use `dev_clear_obj` instead.

When a breakpoint on a variable triggers, the program execution stops and HDevelop highlights the corresponding variable in the variable window. If the variable is not currently displayed in the selected tab of the variable window, HDevelop switches to the tab `Auto` (see below).

Information about the event is displayed in the status bar, e.g., HDevelop displays both the old and the new value of a tuple.

Switching Between Horizontal and Vertical Layout

You can toggle the orientation of the two parts of the variable window. To do this, double-click the dividing line between both parts. You can also drag that line to resize the parts.

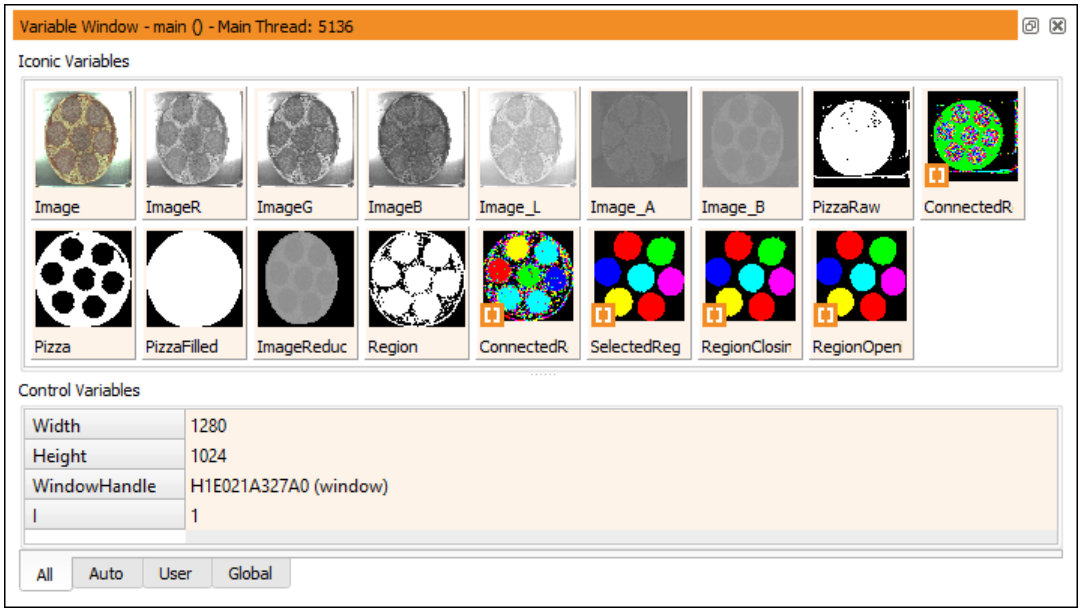
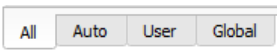


Figure 6.62: Variable window with instantiated iconic and control variables.

Managing Variables

In large programs the variable window can become quite cluttered, which makes watching selected variables difficult. Therefore, you can customize the selection of displayed variables. At the bottom of the variable window, the following tabs are available:



`All` All variables used by the current procedure are displayed at once. Global variables (see section 8.3.2) are marked with 🌐.

`Auto` The variables of the current and the previous operator call are displayed. This is useful when single-stepping through the program, because only the variables relevant to the current context are displayed.

`User` A user-defined selection of variables is displayed. Variables may be added and removed using the context menu of the variable window (see below). If the tab `User` is active, variables may be added from a list in the context menu. In the other tabs variables are added by selecting them first and clicking `Add to User Tab` in the context menu.

It is also possible to select a variable name in the program window and drag it to the variable window. The corresponding variable will then be added to the watched variables, and the tab `User` will be activated.

`Global` All global variables (see section 8.3.2) of the current program are displayed. This includes global variables in external procedures, even if they are not used in the current program.

The context menu of the global variables includes an additional entry `Declared in`. It lists the names and line numbers of the procedures that declare (and thus use) the corresponding global variable. Click on an entry to jump to the corresponding location in the program window.

6.6.1 Iconic Variables

The iconic variables are represented by icons, which contain an image, a region or an XLD, depending on the current value. The icons are created depending on the type of data according to the following rules:

- For images the icon contains a zoomed version of them, filling the icon completely. Due to the zooming onto the square shape of the icon, the aspect ratio of the small image might be wrong. If there is more than one image in the variable, only the *first* image is used for the icon. Similarly, for multi-channel images only the *first* channel is displayed. An exception is made for images with three channels: These are displayed as color icons (RGB).

  The domain of the image is not reflected in the displayed icon. Information about the domain can be obtained from the tool tip which appears when the mouse cursor points to the icon. See figure 6.63 for an illustration.

- Regions are displayed by first calculating the smallest surrounding rectangle and then zooming it so that it fills the icon using a border of one pixel. In contrast to images, the aspect ratio is always correct. This can lead to black bars at the borders. The color used to draw the region is always white without further modifications (except zooming).

- XLD data is displayed using the coordinate system of the largest image used so far. The color used for XLD objects is white on black background.

Because of the different ways of displaying objects, you have to be aware that the coordinates cannot be compared. The variable name is positioned below each icon. They are displayed in the variable window in the order of occurrence or name from left to right. If there is not enough space, a scrollbar is created, which you can use to scroll the icons.

Displaying Iconic Variables

Double-clicking an icon with the left mouse button displays the data in the active graphics window. If you use images of different sizes in a program, the system uses the following output strategy for an automatic part reset, depending on the chosen resize mode (page 90): Every window keeps track of the size of the most recently displayed image. If you display an image with a different size, the system modifies the graphics window coordinate system in a way that the image is visible completely in the graphics window. In `Full Stretch` mode, the part

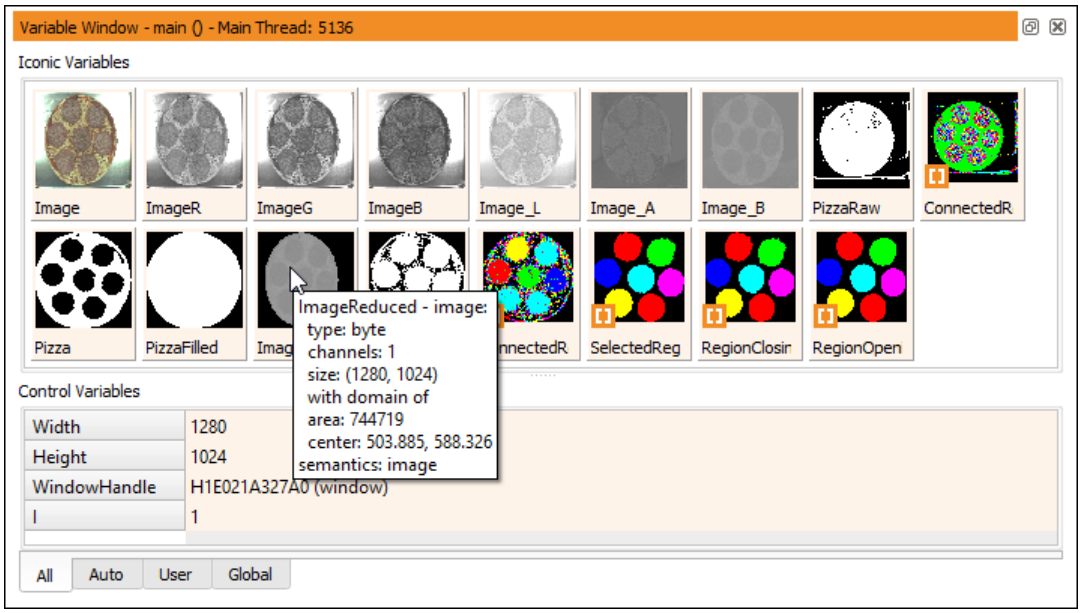


Figure 6.63: Displaying information about an iconic variable with a reduced domain.

will be set to the full image size. In `No Stretch` mode or `Keep Aspect Ratio` mode, the part is extended to fit the full image in a 1:1 aspect ratio with black bars on one side.

If a partial zooming has been activated before (see section "Graphics Window" on page 158), it is going to be suppressed.

Displaying Information about Iconic Variables

You can get information about an instantiated variable by placing the mouse pointer over the corresponding icon in the variable window. See also figure 6.63 for an illustration. The information depends on the contents of the corresponding variable:

- Images: The image type and size and the number of channels is displayed. If the iconic variable contains multiple images, the properties of the first image are reported.
- Regions: The area and the center of the region is displayed. If the iconic variable contains multiple regions, the properties of the first region are reported.
- XLDs: The number of contour points and the length is displayed. If the iconic variable contains multiple XLDs, the properties of the first XLD are reported.

Context Menu

Clicking on an icon with the right mouse button opens a context menu with several options:

`Display`**:** Display the selected iconic variable in the active graphics window. Regions and XLDs are displayed on top of the previous contents of the active graphics window.

`Display Channel`**:** Display a single channel of the selected iconic variable in the active graphics window. This menu lists up to 15 channels. If the iconic variable contains more than 15 channels, you can access the remaining channels by clicking "`Select...`", which opens an auxiliary window listing all channels.

This entry is only available if the selected iconic variable contains a mult-channel image.

`Display Content`**:** Display a single object of the selected iconic variable in the active graphics window. This menu lists up to 15 objects. If the iconic variable contains more than 15 objects, you can access the remaining objects by clicking "`Select...`", which opens an auxiliary window listing all objects.

This entry is only available if the selected iconic variable contains multiple objects, e.g., multiple images, regions, or XLDs.

`Clear / Display`**:** Clear the active graphics window before displaying the selected iconic variable.

`Clear Variable`**:** Clear the selected iconic variable. The contents of the variable become undefined.

`Save`**:** Save the contents of the selected iconic variable to a file. Depending on the content type of the variable (image, region, XLD, ...), different file formats are offered. See also: `write_image`, `write_region`, `write_contour_xld_dxf`, `write_polygon_xld_dxf`, `fwrite_serialized_item`.

`Set/Clear Breakpoint on Variable`**:** Toggle a breakpoint on the variable under the mouse cursor (see section "Breakpoints on Variables" on page 143).

`Activate/Deactivate Breakpoint on Variable`**:** Toggle activation of breakpoint on the variable under the mouse cursor.

`Add to User Tab`**:** The selected variable is added to the tab `User`.

`Find Variable`**:** Open the `Find/Replace...` (page 62) dialog with the name of the selected variable preselected.

`Insert dev_display() into program`**:** Insert the operator `dev_display` into the program window at the IC. The parameter of the inserted instruction is the name of the selected iconic variable.

Shortcut: Ctrl + double click on variable icon.

`Declared in` **(global variables only)** List the names and line numbers of the procedures that declare (and thus use) the selected global variable. Click on an entry to jump to the corresponding location in the program window.

`Sort by Name`**:** Sort all variables in alphabetical order.

`Sort by Occurence`**:** Sort the variables in the same order as they are defined in the program.

`Update Variables`**:** Toggle whether variables will be updated during program execution. This is the same setting as in the runtime preferences (see page 80).

`Add Variable` **(tab** `User` **only):** This submenu contains a list of all variables that are currently *not* displayed in the tab `User`. Click a variable name to add the corresponding variable to the tab.

`Remove from User Tab` **(tab** `User` **only):** The selected variables are removed from the tab `User`.

You can display the corresponding iconic variable in the active graphics window (with or without clearing the window first), and you can clear the iconic variable. If an iconic variable contains multiple items, you can also select a specific item from a submenu. Up to 15 items are listed in this menu. If an iconic variable contains more than 15 items, the remaining items can be accessed by clicking `Select...`. If you click `Select...` in this submenu, you can quickly browse the items of the iconic variable from a dialog. This also works for multi-channel images. See figure 6.64 for an example.

Visualization of Iconic Variables

Normally, regions, images, and XLDs are represented in variable icons. Besides this there are three exceptions, which are shown by special icons:

- Undefined variables are displayed as a question mark icon ?. You may *write to* but not read them, because they do not have any value.
- Brackets [] are used if a variable is instantiated but does not contain an iconic object (empty tuple). This may be the case using operators like `select_shape` with "wrong" specified thresholds or using the operator `gen_empty_obj`. Such a value might be reasonable if you want to collect iconic objects in a variable gradually in a loop using `concat_obj`. Here, an empty tuple is used as starting value for the loop.
- A last exception is an *empty region*. This is *one* region that does not contain any pixels (points), i.e., the area (number of points) is 0. You must not confuse this case with the empty tuple, because there the area is not defined. The empty region is symbolized by an empty set icon ∅.

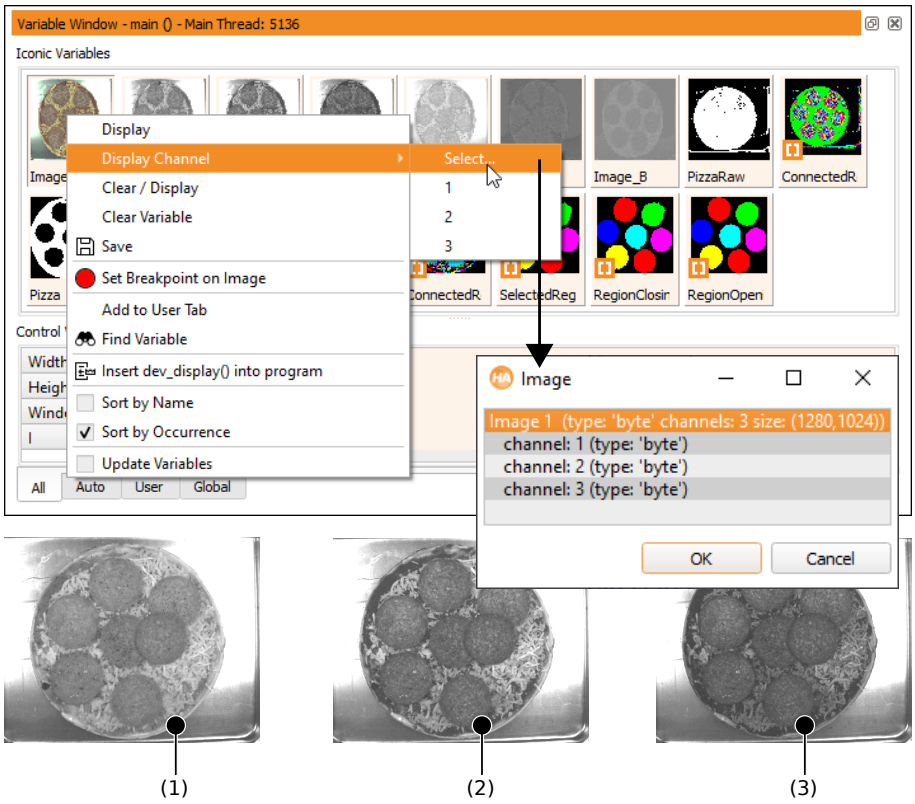


Figure 6.64: Interactive channel selection from an RGB image. (1) channel 1: red, (2) channel 2: green, (3) channel 3: blue.

6.6.2 Control Variables

To the right of the variable name you find its values in the default representation. If you specify more than one value for one variable (tuple), they are separated by commas and enclosed by brackets. If the number of values exceeds an upper limit, the output is clipped. This is indicated by three dots at the end of the tuple. For undefined variables, their name and a `?` are shown in the variable field. An empty tuple is represented by `[]`. Both exceptions use the same symbols as the corresponding cases for the iconic variables.

Context Menu

`Inspect:` Inspect the values of the selected control variables in an auxiliary window (see section 6.7). The display depends on the semantic type of the selected control variable:

- Tuples
- Vectors (page 150)
- Handles (page 151)
- Matrices (page 151)
- Poses (page 152)
- Image acquisition devices (page 152)
- Functions (page 152)
- 3D object models (page 157)

`Inspect as Handle` Force the generic handle inspect window for all types (see section 6.7.3 on page 151).

`Inspect as Tuple` Force the tuple inspect window for all types (see section 6.7.1).

`Plot as Function` Plot the tuple values (see section 6.7.7 on page 152). The variable is assumed to contain y values for equidistant x values. This entry is only available for numeric tuples.

`Plot as X/Y pair(s)` Generate a scatter plot of two variables selected using Ctrl. This entry is only available if two variables containing numeric tuples of equal length are selected.

`Copy`: Copy the values of the selected variables to the system clipboard. If the variable window has the keyboard focus, Ctrl-C can be used as a shortcut. Tuples with zero or more than one values are returned in tuple notation: `[.., ..]`. If multiple variables are selected, the tuples of the different variables are separated by a new line.

`Clear Variable`: Clear the selected control variables. The contents of the variables become undefined.

`Save`: Save the contents of the selected control variable to a file. Depending on the content type of the variable (tuple, handle, ...), different file formats are offered. See also: `write_tuple`, `write_bar_code_model`, `write_calib_data`, `write_data_code_2d_model`, `write_matrix`, `write_object_model_3d`.

`Set/Clear Breakpoint on Variable`: Toggle a breakpoint on the variable under the mouse cursor (see section "Breakpoints on Variables" on page 143).

`Activate/Deactivate Breakpoint on Variable`: Toggle activation of breakpoint on the variable under the mouse cursor.

`Add to User Tab`: The selected variables are added to the tab `User`.

`Find Variable`: Open the `Find/Replace...` (page 62) dialog with the name of the selected variable preselected.

`Declared in` **(global variables only)** List the names and line numbers of the procedures that declare (and thus use) the selected global variable. Click on an entry to jump to the corresponding location in the program window.

`Sort by Name`: Sort all variables in alphabetical order.

`Sort by Occurence`: Sort the variables in the same order as they are defined in the program.

`Update Variables`: Toggle whether variables will be updated during program execution. This is the same setting as in the runtime preferences (see page 80).

`Add Variable` **(tab** `User` **only):** This submenu contains a list of all variables that are currently *not* displayed in the tab `User`. Click a variable name to add the corresponding variable to the tab.

`Remove from User Tab` **(tab** `User` **only):** The selected variables are removed from the tab `User`.

6.7 Inspecting Variables

See also: `dev_inspect_ctrl`.

Double-clicking a control variable opens an inspection window. The type and functionality of the inspection window depends on the semantic type of the variable.

You can also select multiple control variables at once in the variable window by holding down the Ctrl key. To inspect these variables, right-click on the selected variables and select `Inspect`. Selected variables of the same semantic type are grouped in a single inspection window.

6.7.1 Inspecting Tuples

Control variables containing tuples are displayed in a tabular format (see figure 6.65). This is especially useful for the inspection of tuples with a large number of values.

In addition to the tuple values, some statistical data may be displayed:

- minimum value
- maximum value
- sum of values

- mean value
- deviation
- value types
- number of values
- semantics (if appropriate).

You can select, which statistical data is displayed by right-clicking on the statistics table and selecting the corresponding entries.

The selected values of an inspection window can be copied to the system clipboard using the context menu or pressing Ctrl+C. The columns of the copied values are separated by tabs and the rows are separated by newlines, or presented as an HTML table (depending on the application the values are pasted into).

Editing Variable Values

To change a variable value, click into the corresponding cell and type the new value followed by Return.

You enter...	**it becomes**
text	'text' (string)
'text'	'text' (string)
42	42 (integer)
'42	'42' (string)
0.7	0.7 (real)
2e2	200.0 (real)

Depending on the variable type, only certain value types may be allowed, e.g., matrix variables (see below) contain only real values.

You can also add new elements to a tuple: Instead of entering a single value, simply enter a list of values enclosed in square brackets (tuple notation).

For example:

```
Variable Inspect
  1
  2       -> [2,22,222]
  3
```

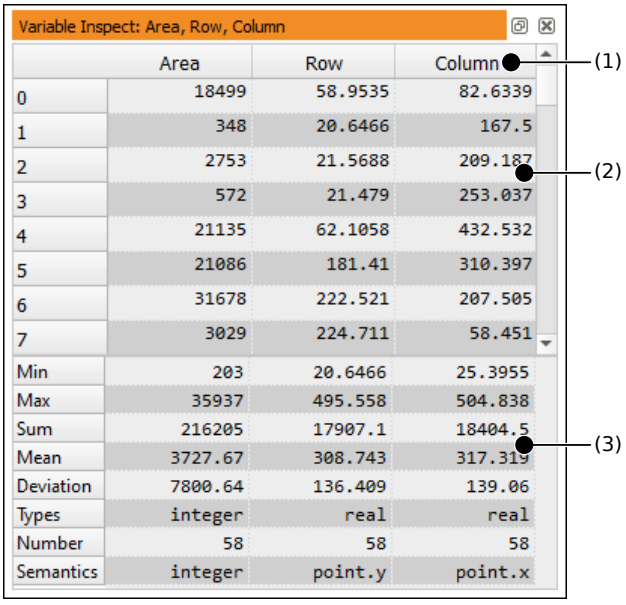


Figure 6.65: Control variable inspection: (1) variable names, (2) list of tuple values, (3) statistics - select from context menu.

results in the following tuple:

```
1
2
22
222
3
```

Do not forget to include the original value (2) if you want to extend the tuple.

Additional values can be appended at the end of the tuple by entering them into the column labeled +.

To delete a variable value, press Delete or enter an empty tuple:

```
Variable Inspect
  1
  2       -> []
  3
```

results in the following tuple:

```
1
3
```

6.7.2 Inspecting Vectors

Vectors are container variables for tuples, iconic objects, or vectors. See section "Vectors" on page 303 for a detailed description. Because vector variables can be multi-dimensional, the inspection window presents their values in a tree-like fashion.

```
vector := {{[1],[2]}, {[3],[4]}, {[5],[6]}}
```

The example variable `vector` is a two-dimensional vector. It contains three vector variables which in turn contain two single-element tuples each. Figure 6.66 shows the inspection window of this variable. Depending on the dimensionality of the inspected vector variable, the sub-nodes can be further inspected by clicking the icon in front of the corresponding element. If the inspected vector variable contains control values as in the example, the individual values at the leafs of the tree can be edited like tuple variables (see section "Editing Variable Values" on page 149).

In addition to the tuple values, some statistical data may be displayed:

- minimum value
- maximum value
- sum of values
- mean value
- deviation
- value types
- number of values
- semantics (if appropriate)
- vector dimension.

You can select, which statistical data is displayed by right-clicking on the statistics table and selecting the corresponding entries.

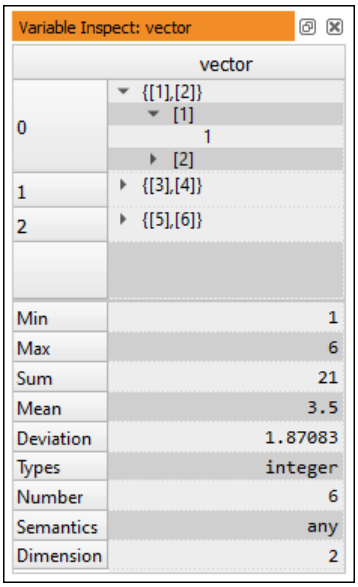


Figure 6.66: Vector variable inspection.

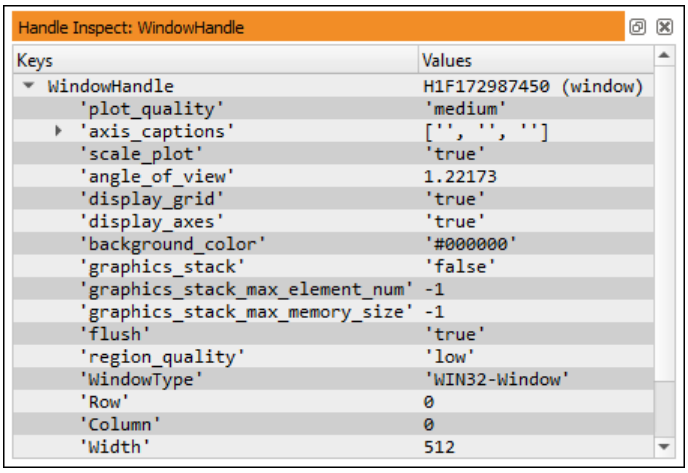


Figure 6.67: Inspection of a window handle variable.

6.7.3 Inspecting Handles

Handles are references to complex data structures. The handle inspect window displays the attributes and values of the selected handle variable. The values are read-only, i.e., they cannot be modified in the dialog. They are updated automatically if they change in the background.

Depending on the semantic type of the referenced data, specific inspection windows are available which are described in the following sections. Double-clicking a handle variable with no specific inspection window always opens the handle inspect window.

When inspecting handles that contain 3D object models, those 3D object models can be visualized directly from within the handle inspect window by checking the checkbox next to them. This is also possible for 3D object models that are contained in, for example, dictionaries. Note that when visualizing multiple 3D object models from within a handle inspect window, all those models will be shown in a single visualization window.

6.7.4 Inspecting Matrices

Control variables that reference a matrix are displayed in a tabular format as displayed in figure 6.68.

The selected values of an inspection window can be copied to the system clipboard using the context menu or pressing Ctrl+C. The columns of the copied values are separated by tabs and the rows are separated by newlines, or presented as an HTML table (depending on the application the values are pasted into).

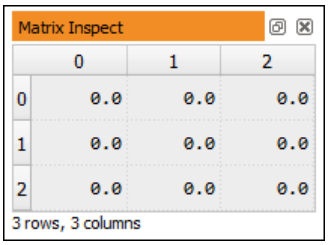


Figure 6.68: Inspection of a matrix control variable.

6.7.5 Inspecting Poses

Control variables that reference a pose are displayed in a tabular format as displayed in figure 6.69.

The selected values of an inspection window can be copied to the system clipboard using the context menu or pressing Ctrl+C. The columns of the copied values are separated by tabs and the rows are separated by newlines, or presented as an HTML table (depending on the application the values are pasted into).

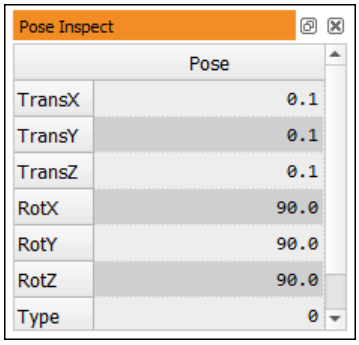


Figure 6.69: Inspection of a pose control variable.

6.7.6 Inspecting Image Acquisition Device Handles

For an image acquisition device handle, a dialog with basic image acquisition device parameters is opened (see figure 6.70). It displays the name, image size, device, port, and other features of the image acquisition device.

The toggle button `Online` allows you to grab images continuously and to display them in the active graphics window. Multiple online inspections from different image acquisition devices at the same time are also supported by opening additional graphics windows before clicking the corresponding button `Online`. If an error occurs during grabbing, it is displayed in the status bar of the dialog.

6.7.7 Inspecting Functions

Control variables with the semantic type `function_1d` are displayed as a function plot by default upon inspection. It is also possible to plot the data of variables containing arbitrary numeric tuples (length > 1) by selecting `Plot as Function` from the context menu of the variable window. It is assumed that these variables contain y values for equidistant x values. Finally, you can select two variables containing numeric tuples of equal length to generate a scatter plot (see below).

Example:

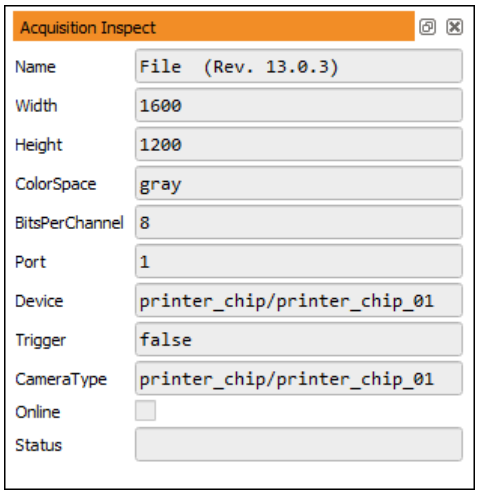


Figure 6.70: Inspecting an image acquisition device handle.

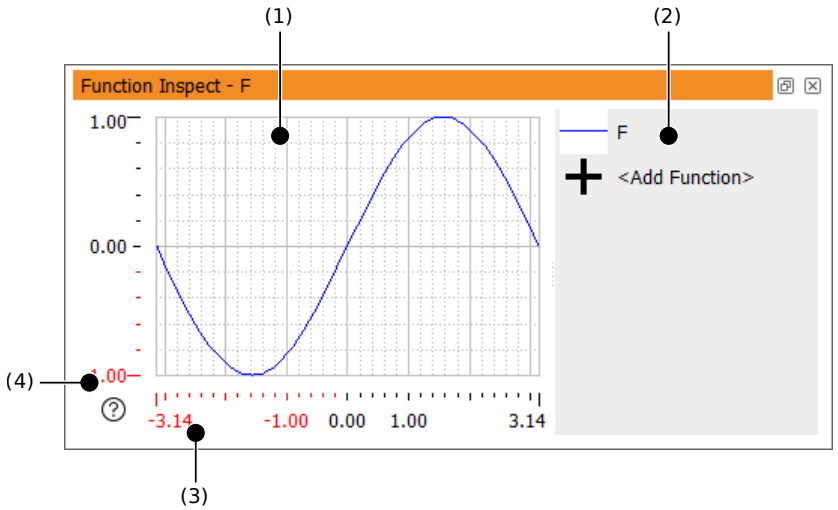


Figure 6.71: Inspecting a variable of semantic type `function_1d`: (1) plot area, (2) plot list, (3) x-axis, (4) y-axis.

```
X := [rad(-180) : rad(10) : rad(180)]
SinX := sin(X)
create_funct_1d_pairs (X, SinX, F)
```

The variables `X` and `SinX` store discrete input values for the sine function in the range $[-\pi, \pi]$.These values are fed into the operator `create_funct_1d_pairs` to create the function variable `F` with the semantic type `function_1d`.

Double-click `F` in the variable window to generate a function plot (see figure 6.71).

Right-click `SinX` in the variable window and select `Plot as Function` to generate a function plot of the y values (see figure 6.72). Note that the tuple index (0 .. 36) is used to set the x range. The plot style of numeric tuples defaults to steps. This can be changed from the context menu of the plot list (see below).

Adjusting the Function Plot

The zoom level of the function plot may be adjusted with the mouse wheel over the plot area or one of the axes. The displayed part may be panned by dragging the plot area or one of the axes, and adjusted by dragging the axis limits. Additional display settings are available in the context menus (see below). You can add additional variables to the current function plot window by clicking `<Add Function>` in the plot list.

Context Menu (x-axis)

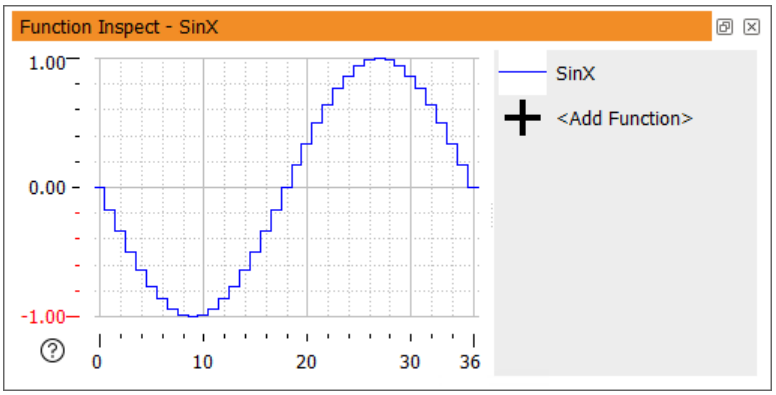


Figure 6.72: Inspecting a tuple with y values for equidistant x values.

	Action	Shortcut	Description
	`Reset Bounds`	w or Ctrl+Shift+W	Set width to default.
	`Set 1:1 Aspect Ratio`		Set x range to same scale as y range.
	`Linear Scale`		Set x-axis to linear scale.
	`Logarithmic Scale`		Set x-axis to logarithmic scale.
	`User-defined Range`		Freeze x range.
	`Increasing Range`		Let x range grow on demand.
	`Adaptive Range`		Let x range grow or shrink on demand.

Context Menu (y-axis)

	Action	Shortcut	Description
	`Reset Bounds`	h or Ctrl+Shift+H	Set height to default.
	`Set 1:1 Aspect Ratio`		Set y range to same scale as x range.
	`Linear Scale`		Set y-axis to linear scale.
	`Logarithmic Scale`		Set y-axis to logarithmic scale.
	`User-defined Range`		Freeze y range.
	`Increasing Range`		Let y range grow on demand.
	`Adaptive Range`		Let y range grow or shrink on demand.

Context Menu (plot area)

	Action	Shortcut	Description
	`Reset Bounds`	r or Ctrl+Shift+R	Reset width and height to default.
	`Zoom In Mode`	+ or Ctrl+Shift++	Zoom in (both axes).
	`Zoom Out Mode`	- or Ctrl+Shift+-	Zoom out (both axes).
	`Enter Bounds...`	b or Ctrl+Shift+B	Enter vertical and horizontal bounds parametrically.
	`Show Mouse Position`	p or Ctrl+Shift+P	Visualize the mouse position with a cross hair.
	`Show Function Value At X`	x or Ctrl+Shift+X	Display the function value at horizontal mouse position.
	`Show Function Value At Y`	y or Ctrl+Shift+Y	Display the function value at vertical mouse position.
	`Show Background Grid`	g or Ctrl+Shift+G	Display grid lines in the background of the plot area.
	`Insert Plot Code for Graphics Window`	Ctrl+Shift+V	Generate code to plot the current view in a graphics window.

Context Menu (plot list)

Action	Shortcut	Description
`Color`		Set the plot color of the corresponding variable.
`Thickness`		Set the plot thickness of the corresponding variable.
`Style`		Set the plot style of the corresponding variable (see below).
`Inspect as Tuple`		Display the values of the corresponding variable.
`Fit Graph`		Adjust bounds to view all values of the corresponding variable.
`Remove`		Remove the corresponding variable from the function plot.

Scatter Plot

A scatter plot plots values of one variable against values of another. Both variables must have the same length. There are two ways to generate a scatter plot:

- Select two variables containing numeric tuples of equal length in the variable window using Ctrl. Select `Plot as X/Y pair(s)` from the context menu. The first variable in the current sorting order is used as the x variable.
- Select the first variable in the variable window. Select `Plot as Function` from the context menu. In the plot list of the function plot add the other variable. Open the context menu of the variable containing the y values, select `Plot against X values`, and select the other variable.

The context menu of the plot list contains additional entries for the variables used for a scatter plot:

Action	Shortcut	Description
`Swap (X, Y)`		Swap the variables used for the scatter plot.
`Ungroup`		Ungroup the variables used for the scatter plot and plot each on its own.

As an illustration we add the following assignment to the above code lines, and select the variables `SinX` (sine function) and `CosX` (cosine function) to generate the scatter plot from figure 6.73.

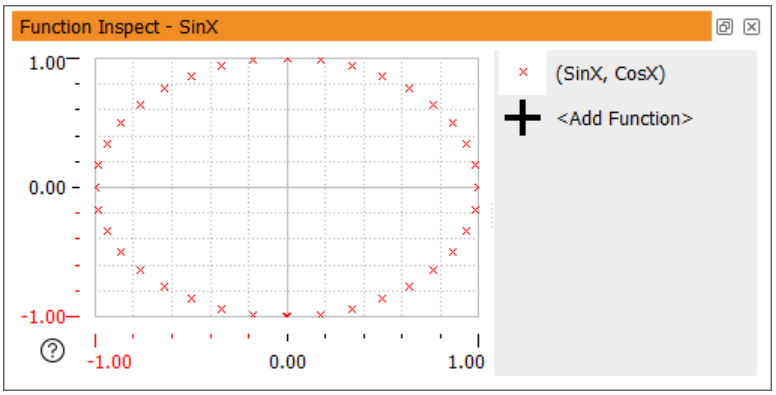


Figure 6.73: Example for a scatter plot using the variables `SinX` and `CosX`.

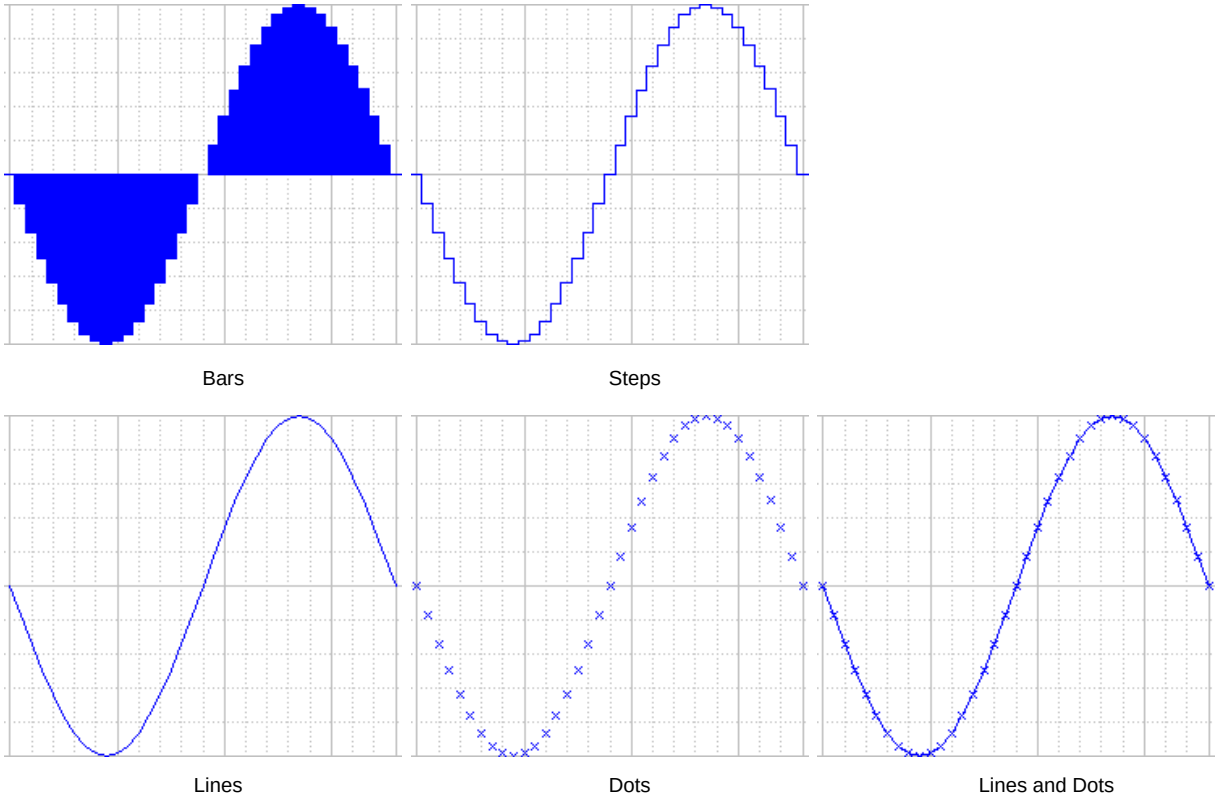


Figure 6.74: Plot styles.

```
CosX := cos(X)
```

Note that the aspect ratio of the plot is chosen to match the window size. To get a correct aspect ratio (in this case: a round plot), select `Set 1:1 Aspect Ratio` from the context menu of either the x- or the y-axis.

Plot Style

The available plot styles are illustrated in figure 6.74.

The first two styles (bars and steps) are only available for plots of numeric tuples against their index. The other styles (lines, dots, lines and dots) are available for all function plots.

6.7.8 Inspecting 3D Object Models

See also: `disp_object_model_3d`.

3D object models are referenced by control variable handles with the semantic type *object_model_3d*. Double-clicking such handles opens a special inspection window that shows the parametric properties of the contained 3D object model(s).

As an example, the following line of code loads one of the supplied 3D object models from disk and stores it in `ObjectModel3D`:

```
read_object_model_3d ('bmc_mini', 'mm', [], [], ObjectModel3D, Status)
```

After executing this line, double-clicking the control variable `ObjectModel3D` opens the inspection window from figure 6.75. See `get_object_model_3d_params` for a description of the displayed properties.

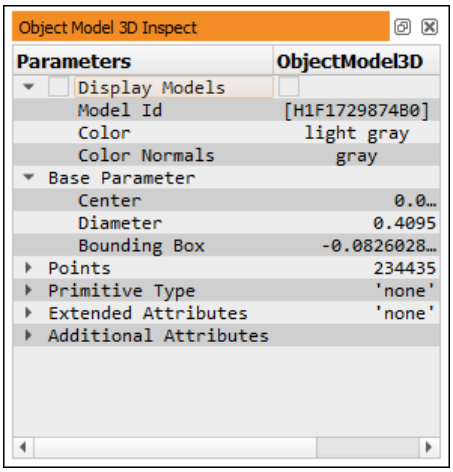


Figure 6.75: Parametric inspection of a 3D object model.

To visualize a specific 3D object model or all 3D object models (in case the handle contains multiple models), select the checkbox of the corresponding model ID or the checkbox next to `Display Models`, respectively. The selected models are then visualized in a special visualization window shown in figure 6.76.

The slots `Color` and `Color Normals` of the parametric inspection window define the color of the displayed model(s) or the optionally displayed normals, respectively. These colors can be changed by double-clicking the

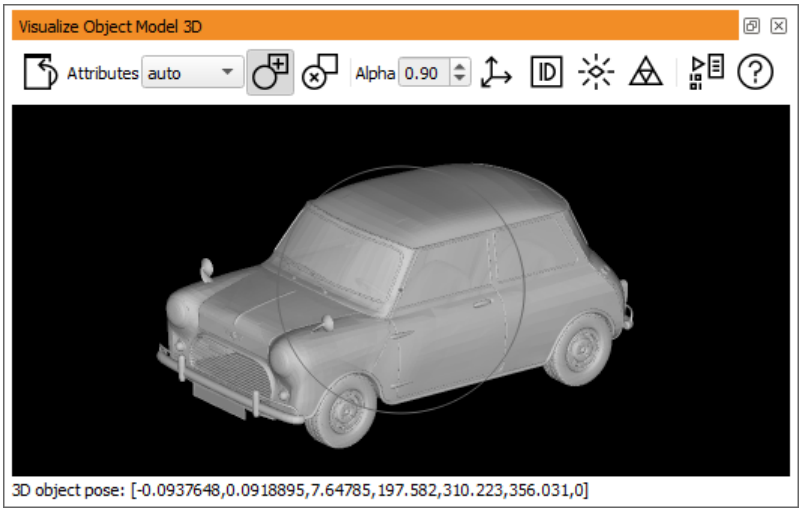


Figure 6.76: Visual inspection of a 3D object model.

corresponding slot and selecting a color name from the drop-down list. You can also enter a valid color name (page 349) or an arbitrary hex triplet into the color slots.

The 3D object model visualization window supports the following interactions:

drag inside circle	rotate around X/Y axis
drag outside circle	rotate around Z axis
mouse wheel	zoom in and out
Shift +drag	zoom in and out
Ctrl +drag	pan 3D object model(s)

`Attributes`: Select the display mode of the 3D object model(s). Possible values are: `auto` (chooses the most appropriate mode), `faces`, `primitive`, `points`, and `line`.

`Alpha`: Set the translucency of the displayed 3D object model(s). The alpha value ranges from 0 (full transparency) to 1 (opaque).

The tool bar buttons control the way the 3D object models are displayed:

- Reset to start pose of the displayed 3D object model(s).
- Rotate around the center of the 3D object model(s).
- Rotate around the selected surface point of the 3D object model(s).
- Display coordinate system of 3D object model(s).
- Show ID(s) of 3D object model(s) (useful if multiple models are displayed).
- Show the normals of the 3D object model(s).
- Display the polygons of the 3D object model(s) (if applicable).
- Generate code lines that will display the current view in the graphics window.
- Display a short usage information in the status bar.

6.8 Graphics Window

This window displays iconic data: images, regions, and XLDs. You may open several graphics windows. The active graphics window is shown by in the window's tool bar.

Figure 6.77 (1) shows an example graphics window which is displaying a color image of a pizza overlaid with region data (the segmented salami slices). One of the displayed regions is selected (illustrated by the dashed border). The variable name and index of the selected region is displayed in the title bar.

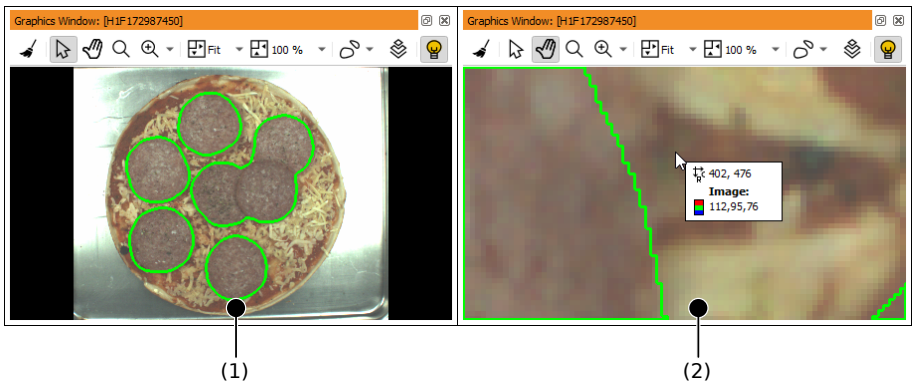


Figure 6.77: Graphics window displaying a tasty pizza: (1) full view, (2) zoomed in with pixel information.

The origin of the image coordinate system is the upper left image corner with the coordinates (0,0). The x values (column) increase from left to right, the y values (row) increase from top to bottom. When the mouse cursor is placed inside a graphics window, the coordinates *(row, column)* and the gray value (or in this case: the RGB values) at that position are displayed in the status bar (see page 50). Sometimes, it is desirable to display this information close to the mouse cursor. This can be achieved by holding down the Ctrl key (note: this does not work when the zoom in and out tool is selected since pressing Ctrl inverts the corresponding zoom action). Figure 6.77 (2) shows the coordinate/color value display after zooming in with the mouse wheel.

Normally, the visible part in the graphics window is based on the most recently displayed image, which is automatically zoomed so that every pixel of the image is visible. The coordinate system can be changed interactively using the menu `Visualization` ▷ `Set Parameters` ▷ `Zoom` (see section "Menu `Visualization`" on page 88) or the operator `dev_set_part` (see section "`Develop`" on page 102). Every time an image with another size is displayed, the coordinate system is adapted automatically.

Every HDevelop graphics window has its own visualization parameters. Thus, modifying the parameters (see section "Menu `Visualization`" on page 88) applies to the currently active graphics window *only*, i.e., the parameter settings of all other open graphics windows remain unchanged.

Each window has a history that contains all

- objects and
- display parameters

that have been displayed or changed since the most recent clearing or display of an image. This history is used for redrawing the contents of the window. The history is limited to a maximum number of 30 "redraw actions", where one redraw action contains all objects of one displayed variable.

Other output like text or general graphics like `disp_line` or `disp_circle` or iconic data that is displayed using HALCON operators like `disp_image` or `disp_region` are *not* part of the history, and are *not* redrawn. Only the object classes image, region, and XLD that are displayed with the HDevelop operator `dev_display` or by double-clicking on an icon are part of the history.

Tool Bar

Clear the graphics window and its history.

Switch to *select* mode. In this mode, you can select regions or XLDs that are displayed in the graphics window. A selected item is highlighted with a dashed border. If multiple layers of region/XLD data are displayed in the graphics window, the first click selects the uppermost region/XLD under the mouse cursor. Each subsequent click at the same position selects the region/XLD below the currently selected item. The variable name of the selected item is displayed in the title bar of the graphics window for reference.

You can use the *select* mode to inspect gray value histograms and features of individual regions or XLDs.

In the example image illustrated in figure 6.77 on page 158, the displayed image of a pizza is overlaid with region data. A single region is selected.

Combined move/zoom tool. Drag the displayed image with the left mouse button to alter the displayed portion.

Magnifying glass. Click into the graphics window to magnify the area at the mouse cursor.

Zoom in. Click the small arrow next to the icon to switch to zoom out.

Zoom out. Click the small arrow next to the icon to switch to zoom in.

Set image size. Clicking this icon sets the image size to the shown value. The value can be selected from the menu attached to the small arrow. Additionally, the resizing behavior can be selected. See section "`Image Size`" on page 90 for additional information.

Set window size. (Only available for floating windows.) Clicking this icon sets the window size to the shown value. The value can be selected from the menu attached to the small arrow. See section "`Window Size`" on page 89 for additional information.

Draw ROIs and XLDs interactively. See section "ROI Window" on page 162.

Toggle 3D plot mode (see below).

Active graphics window.

Non-active graphics window. Click the icon to activate the corresponding graphics window. Only one graphics window may be active any given time.

If you want to specify display parameters for a window, you may select the menu item `Visualization` in the menu. Here you can set the appropriate parameters by clicking the desired item (see section "Menu `Visualization`" on page 88). The parameters you have set this way are used for the active window. The effects of the new parameters will be applied directly to the *last* object of the window history and alter its parameters only.

Context Menu

The context menu may be enabled/disabled from the preferences (page 80).

The entries of the context menu are a subset of the menu `Visualization` (page 88).

3D Plot Mode

Clicking activates an interactive 3D plot mode. It displays meaningful information for height field images, i.e., images that encode height information as gray values. The greater the gray value, the higher the corresponding image point. Figure 6.78 shows a height field image of a solder ball (1) and the corresponding 3D plot (2).

The 3D plot mode uses OpenGL and benefits from hardware acceleration.

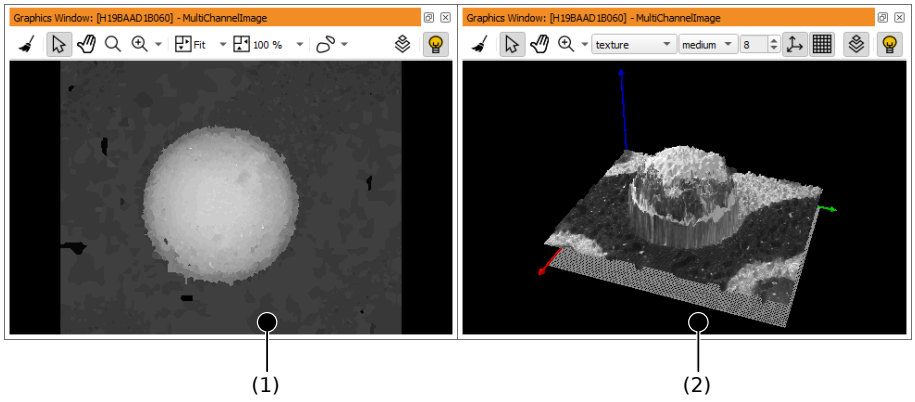


Figure 6.78: (1) Default image display, (2) 3D plot mode.

Using the mouse you can alter the view of the 3D image (*select* mode must be active for this to work, click in the tool bar):

- Drag the image to rotate the view.
- Hold Shift and drag the image up and down to zoom out and in, respectively. Alternatively, use the mouse wheel.
- Hold Ctrl and drag the image to translate the view.

There are four different rendering methods (texture, shaded, hidden_lines, and contour_lines) which can be selected from the drop-down menu in the tool bar. See `set_paint` for detailed information about the different methods. The display quality may be fine-tuned in the tool bar of the graphics window or in the visualization parameters of the graphics window (right-click into the graphics window, select `Set Parameters`, and open the tab card `Paint`).

`Mode` sets the rendering mode just like the drop-down menu in the graphics window.

`Plot Quality` allows setting the rendering quality in four steps. On systems without proper display hardware acceleration a lower quality should be selected to speed up the display.

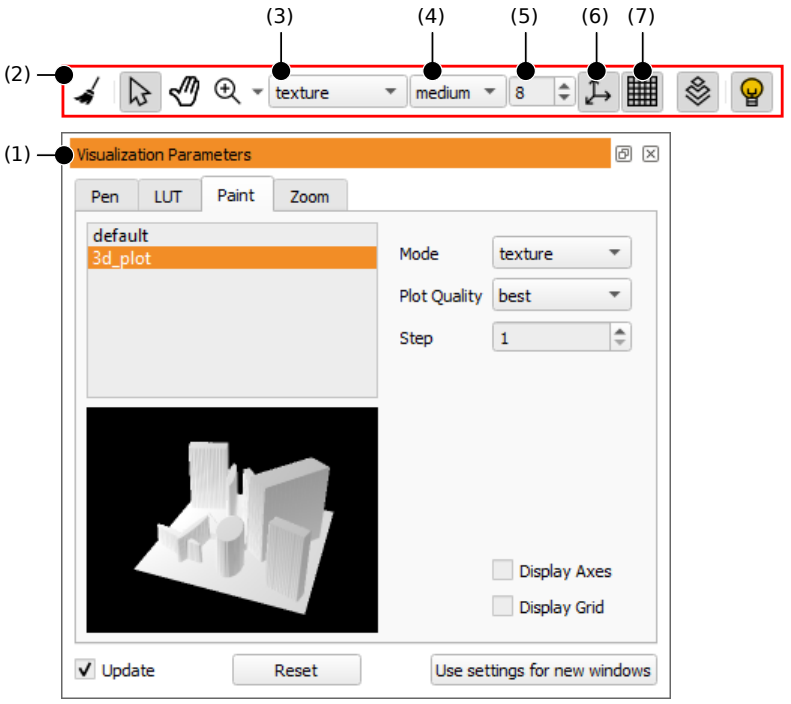


Figure 6.79: 3D plot mode settings: (1) visualization parameters, (2) tool bar, (3) mode, (4) quality, (5) step, (6) axes, (7) grid.

`Step` sets the level of detail. In general, the lower the step value, the higher the level of detail. However, if the rendering mode is set to contour_lines, increasing the step value increases the level of detail.

`Display Axes` If this is enabled, the axes of the 3D coordinate system are displayed in the 3D view.

`Display Grid` If this is enabled, the "floor" of the 3D plot is painted as a grid.

See also section "`Paint` settings" on page 95 for the other paint modes that may be selected in this window.

Special Keyboard Shortcuts in the Graphics Window

Left, Right, Up, Down	move mouse cursor 1 pixel
Alt+Left, Right, Up, Down	move mouse cursor 10 pixels
Ctrl+Left, Right, Up, Down	pan image 1 pixel
Ctrl+Alt+Left, Right, Up, Down	pan image 10 pixels

If there is at least one docked graphics window, all floating graphics windows are closed with F2. This shortcut also works in the main window.

More keyboard functions of the graphics window are listed in appendix D.4 on page 359.

6.9 Canvas Window

The canvas window is a container for graphics windows. It allows you to organize multiple graphics windows side-by-side.

You can open the canvas via `dev_open_tool` or via menu `Window` ▷ `Open Canvas Window`. As soon as the canvas window is open, all new graphics windows opened with `dev_open_window` will automatically show up on the canvas. Regardless of this, you can add any graphics window to the canvas manually by clicking its context menu entry `Add To Canvas`. Once on the canvas, drag the graphics window by its title bar to change its position. To adapt its size, drag the bottom right window corner. To remove a graphics window from the canvas again, open its context menu and choose `Float From Canvas`.

The canvas displays graphics window states as follows:

Active: A yellow stroke highlights the active graphics window, see figure 6.80.

Selected: An orange colored frame indicates the selected graphics window.

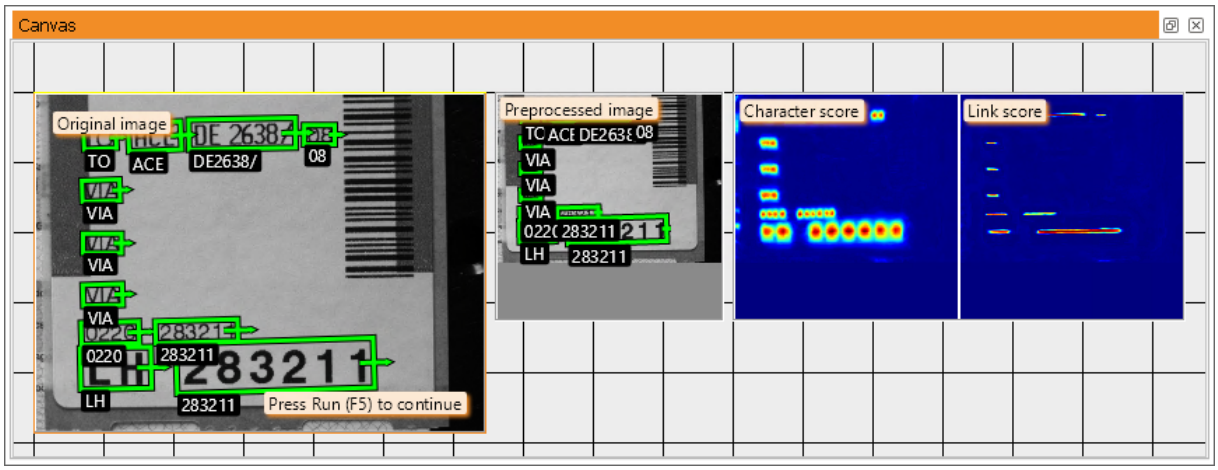


Figure 6.80: The canvas window.

For configuration of the canvas, open the context menu by right-clicking on the canvas. Alternatively, choose `Canvas Options` from the menu `Window`. See section 6.2.9.8 on page 109 for information about your configuration possibilities.

Note that the canvas window cannot be tabbed with other windows.

6.10 ROI Window

Using the ROI window you can draw and manage multiple figures interactively. These figures consist of an arbitrary collection of geometric elements or free-form drawings. Ultimately, these figures are interpreted as ROIs or XLDs that can be used in your current program by generating the appropriate program lines. You can also perform simple interactive measurements in that you specify a known dimension and thus translate pixels to real-world units.

As an introductory example, the image shown in figure 6.81 (1) shall be used as the base image for a completeness check application. To determine the location and the alignment of the chip in subsequent images, parts of its imprint will be used as a landmark. Therefore, an ROI has to be created. It would be cumbersome to generate an ROI based on written-down pixel coordinates. Instead, it is much easier to draw a figure right into the image and let HDevelop generate the corresponding instructions.

Generating an ROI Interactively

- Click the tool bar icon to create a new figure.

  This will open the dialog shown in figure 6.81 (2) which will act as a tool box for your new figure.

  By default, the figure type is set to ROI (cf. in the tool bar), which is just what we want for this example.

- Click to create a rectangle.

  Click and hold inside the graphics window to draw an initial rectangle.

  You can modify the rectangle by dragging its sides or corners. Make the rectangle match the big label on the chip (see figure 6.82 (1)).

  Click the right mouse button to confirm the rectangle and it will be created in the toolbox.

- Repeat the last step to create another rectangle and make it match the lower part of the label (see figure 6.82 (2)).

- Click to generate program code in the current program.

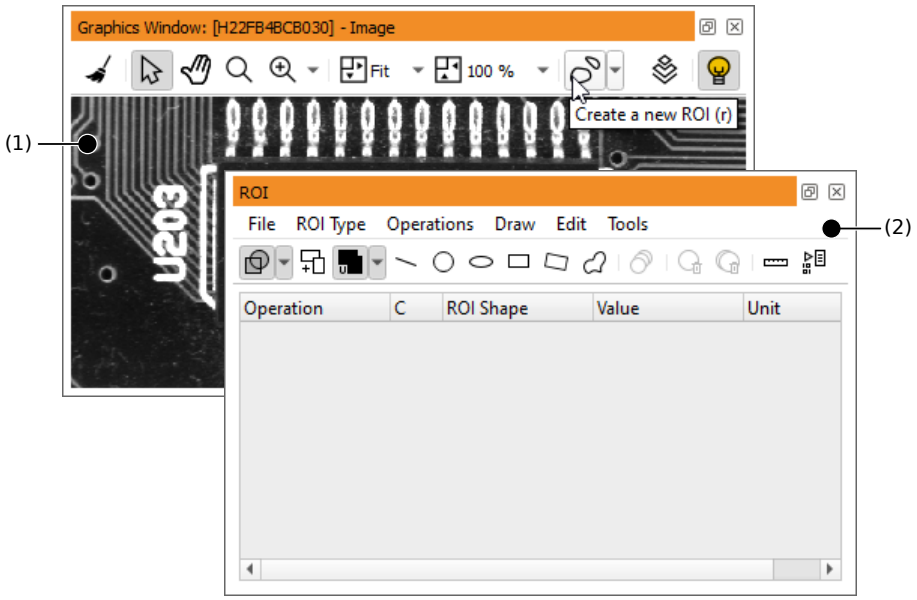


Figure 6.81: Base image for the creation of an ROI.

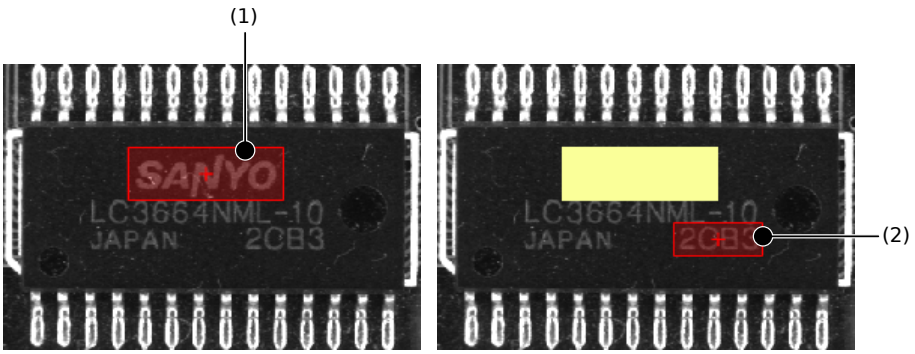


Figure 6.82: Drawing an ROI of rectangles.

```
gen_rectangle1 (ROI_0, 197.724, 233.807, 235.947, 360.162)
gen_rectangle1 (TMP_Region, 260.088, 326.066, 284.229, 398.269)
union2 (ROI_0, TMP_Region, ROI_0)
```

In this example, the iconic variable ROI_0 corresponds to the region defined by the two rectangles.

Editing the Figure

Please note that figure changes are not recorded in the undo buffer and cannot be reverted. However, you can save all figures in the tool box and load them later using the menu `File`.

You can make changes to a figure by clicking on its individual constituent parts to select them. In figure 6.83, the upper rectangle is selected and can be modified by dragging its handles. You can also edit the selected element in the mode it was created in by clicking (right-click to confirm the changes).

The corresponding values in the tool box will be updated accordingly. You can also edit the values in the tool box parametrically. The parameters *Perimeter* and *Area* are calculated depending on the other parameters of the rectangle. They can not be modified directly and are therefore grayed out.

If you click on the upper rectangle twice, the entire figure will be selected and can be dragged with the mouse.

In general, if the figure consists of overlapping elements, the first click will select the topmost element. Each following click will select the element below the selected one. When the bottommost element is reached, the next click will select the entire figure. In case of a very complex figure with many stacked elements it might be easier to select a distinct element by clicking the corresponding data block in the tool box.

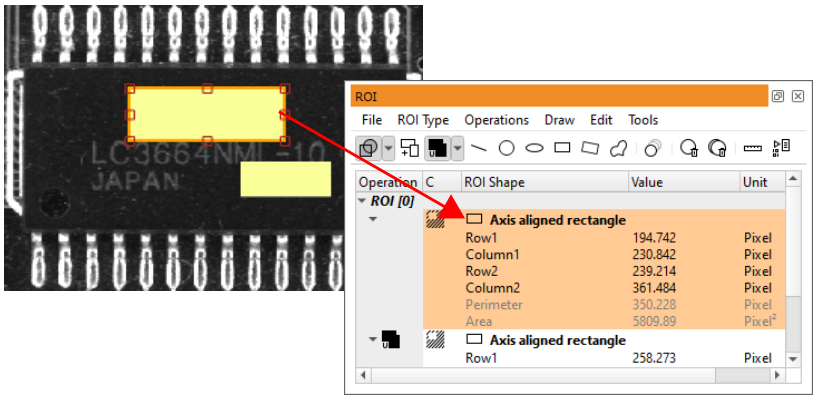


Figure 6.83: An ROI made of two rectangles.

Delete Elements From the Figure

To delete the selected element, press Del, or click . All figures can be deleted by clicking .

You can also add additional geometric elements to the figure by clicking the corresponding icons on the tool bar.

Add Additional Figures to the Tool Box

As already mentioned, the tool box supports the creation of multiple figures. Click to add a new (empty) figure to the tool box. When you subsequently draw new elements, they will be added to the new figure. Multiple figures are converted to an object tuple when generating code.

Mode Selection

The tool box supports three modes of operation. The modes can be selected from the drop-down button of the tool bar or from the menu `ROI Type`:

(ROI) Select this mode if you want to generate a pixel-based region of interest. The ROI consists of geometric elements. The faces defined by these elements are connected by set operations (see below).

Furthermore, for each element a masking mode may be toggled independently by clicking the corresponding icon in the tool box:

The default masking mode ( ) selects the *inside* of the element, e.g., the inner area of a circle.

The complement masking mode ( ) selects the *outside* of the element, e.g., everything outside of a circle.

(XLD) Select this mode if you want to generate a vector-based closed-contour XLD (or a line XLD). The XLD consists of geometric elements. The contours defined be these elements are connected by set operations (see below).

(Path) Select this mode if you want to generate a vector-based path XLD. The path consists of geometric elements. The individual elements of the path are connected with straight lines. This connection is done automatically. Of the two endpoints of each element the one that is closest to an endpoint of another element is connected to that endpoint.

Depending on the selected mode, different geometric elements are available in the tool box:

Tool	Type	ROI	XLD	Path
	line	X	X	X
	circle	X	X	-
	circular arc	-	X	X
	ellipse	X	X	-
	elliptic arc	-	X	X
	axis aligned rectangle	X	X	-
	rotated rectangle	X	X	-
	arbitrary region	X	X	-

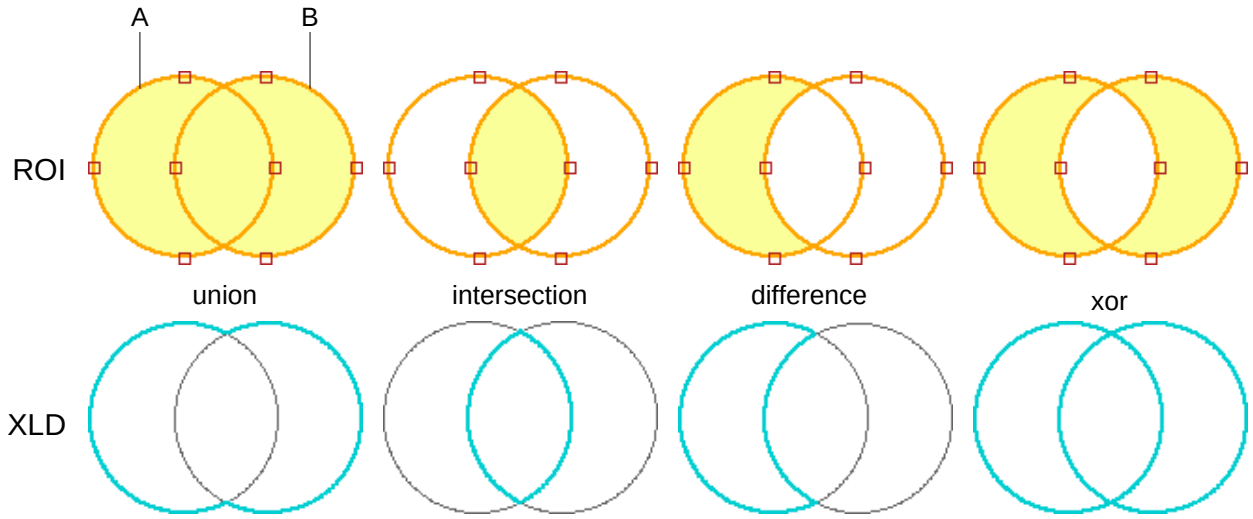


Figure 6.84: Set Operations.

Set Operations

The elements of a figure are connected by set operations. In the case of ROIs, applying the set operations determines the final face of the compound region. For closed-contour XLDs, applying the set operations determines the final contour of the compound XLD. Set operations are not meaningful and therefore not available in path mode.

The set operation of the next new element can be selected from the drop-down button of the tool bar or the menu `Operations`. The set operation of an existing element can be changed by clicking the corresponding icon in the element's data block.

The following set operations are available (see figure 6.84 for an illustration):

- (union of A and B)
- (intersection of A and B)
- (difference, i.e., A minus B)
- (xor, i.e., A or B exclusively)

Interactive Measurements

Usually, all dimensions in the tool box are given in pixels. To convert pixel values to real-world dimensions, a simple calibration can be performed, i.e., a known dimension has to be specified.

See figure 6.85 for an example. The length of the line is known, so it can be used for the calibration.

- Click to open the 2D calibration panel.
- Select the desired unit from the drop-down combo box `Unit`.
- Select `Set 2D Calibration` to start 2D calibration mode.
- Click on the *Length* value in the data block of the line, and specify the known dimension, e.g., 10 for 10mm.
- Click `Modify Regions` to leave 2D calibration mode. Now, any changes to the parameters of the data block will again modify the figure itself, leaving the calibration untouched.

Now, all dimensions are given in the selected unit.

6.11 Help Window

The help window provides access to HALCON's online documentation. The window is split into two areas: On the left, navigational panels are available as tab cards. They are described below. Please note that you can press the

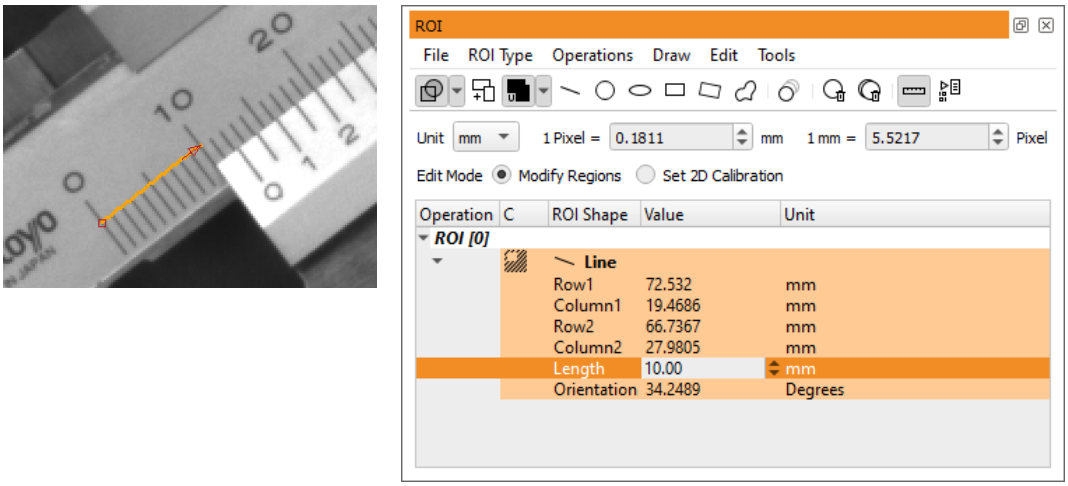


Figure 6.85: Performing 2D calibration.

Up and Down keys in the navigational panels to select the previous and the next entry, respectively. This allows you to examine, e.g., search results quickly. The main area displays the actual hypertext content. The size of the two parts of the help window can be adjusted by dragging the dividing line.

Parts of the documentation are available in PDF format. The help window does not display these files itself but launches the default viewer when a PDF link is being followed. If a link to a PDF file appears in the navigation (either as the result of a full-text search or from the selection of index keywords), it will be marked by a PDF icon. A single click will select the entry and display a link to the file in the contents area. A double-click will open the corresponding file in the external viewer. This way, you can quickly browse the search or index results in the navigation without accidentally running the PDF viewer.

Contents

This tab card presents the chapters and sections of the online documentation as a hierarchical tree. Click on a node of the tree to display the associated document. See figure 6.86 for an example.

Operators

This tab card lists all operators in alphabetical order. Click on an operator name to display the corresponding page from the Reference Manual. Enter any name into the text field Find to show only operators matching that name.

Search

Enter a search query into the text field, and click Find to start a full-text search. The check boxes below the text field indicate the search scope. For example, to make sure only HTML matches are displayed, deselect the check box PDF. The search result is displayed below the query. The rank (in percent) indicates how well each found document satisfies the query.

The query may consist of one or multiple words. HDevelop will find all documents that contain any of the specified words.

To search for a phrase, enclose it in double quotes:

```
"radiometric calibration"
```

Precede all mandatory words with a plus sign (+). For example, to find all documents that contain *filter* and *gauss*, enter:

```
+filter +gauss
```

You can exclude specific words from your search result. To find all documents that say anything about filters except Gaussian filters, enter:

```
filter -gauss
```

`Index`

This tab card provides access to HALCON operators and relevant sections of the documentation through index entries just like an index in a book. The list of index entries can be filtered by entering any word into the text field `Find`. If you enter multiple words, only index entries matching *all* the words are displayed.

When you select index entries from the list, the related operator names and links to the corresponding parts of the documentation are displayed below the index entries. The subtopics can be navigated using Ctrl+Up and Ctrl+Down to select the previous and the next match, respectively. Please note that the text of PDF links also includes the page number so you can find the desired information very quickly.

`Bookmarks`

This tab card lists all user-defined bookmarks. You can add the currently displayed document to the list by clicking the button `Add`. To remove a bookmark from the list, select it and click the button `Delete`.

Help Window Actions

	Action	Shortcut		Description
	`Locate page`	Ctrl+L		Display the location of the current page in the tree of the contents tab.
	`Back`	Alt+Left		Go back in the browse history.
	`Forward`	Alt+Right		Go forward in the browse history.
	`Home`	Alt+Home		Go to the starting page of the HALCON Reference Manual.
	`Bookmark`	Ctrl+D		Add the currently displayed document to the tab card `Bookmarks`.
	`Insert operator into program`	Alt+Return Alt+Enter	or	If the currently displayed document is the reference page of a HALCON operator, enter this operator into the operator window.

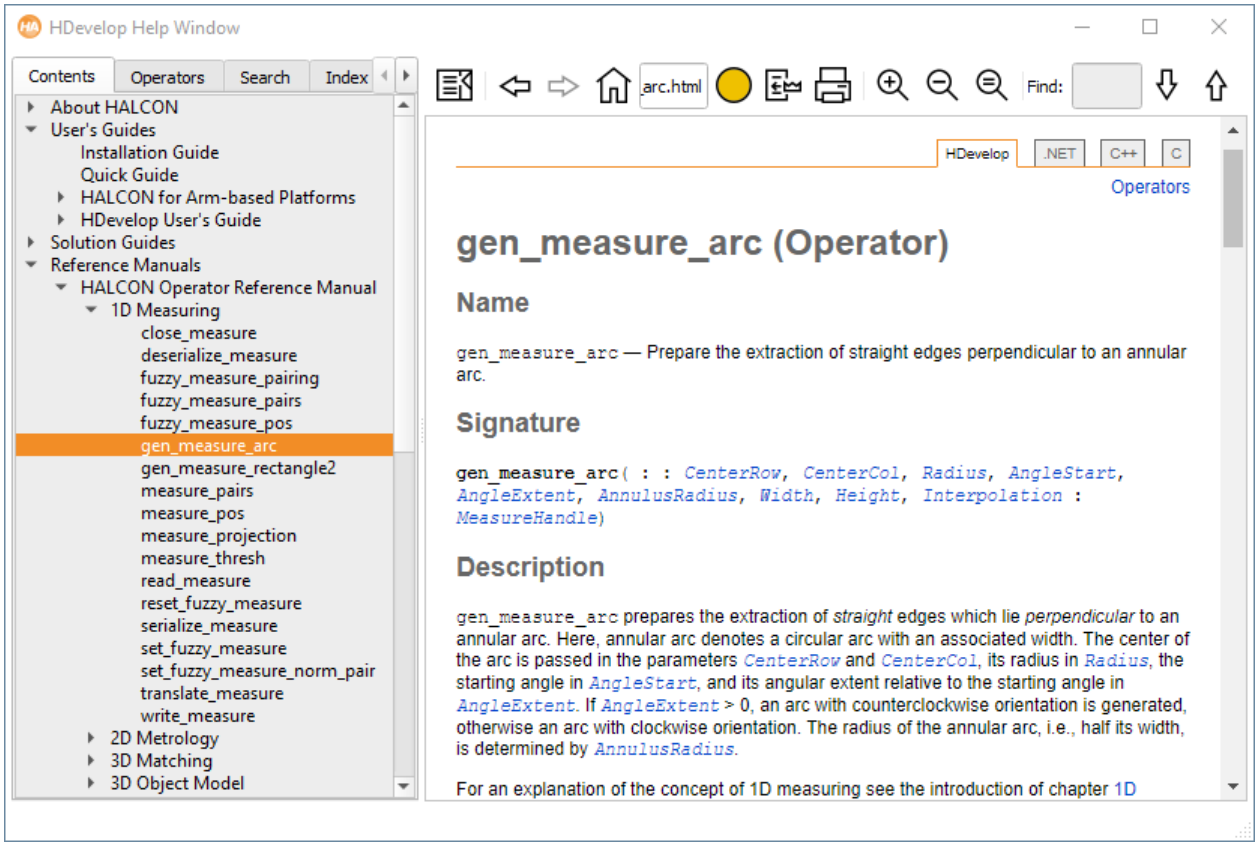


Figure 6.86: Help window with the contents tab card.

	Action	Shortcut	Description
	`Print...`	Ctrl+P	Open the operating system dependent printer selection dialog to print the currently displayed page.
	`Increase Zoom Factor`	Ctrl++	Increase the font size of the help window.
	`Decrease Zoom Factor`	Ctrl+-	Decrease the font size of the help window.
	`Reset to normal size`	Ctrl+0	Reset to normal size
	`Find`	Ctrl+F	Enter a word or substring to find it in the currently displayed document. The first match is highlighted as you type. If no match is found, the text field blinks shortly. You can use the cursor keys (down and up) to highlight the next match or the previous match, respectively. Alternatively, you can use the following two buttons.
	`Next`	Down	Highlight the next match.
	`Previous`	Up	Highlight the previous match.
	`Next Link`	Tab	Highlight the next link on current page.
	`Next Link`	Shift+Tab	Highlight the previous link on current page.
	`Jump to Link`	Return	Jump to the highlighted link.

6.12 Zoom Window

Synopsis: Zoom window for image details and pixel inspection.

See also: Menu `Visualization` ▷ `Zoom Window`.

The zoom window enables the interactive inspection of image details. You can open up any number of zoom windows with different zoom levels (see Menu `Visualization` ▷ `New Zoom Window`). The window also displays the gray values of each image channel at the mouse cursor position. Apart from this, the pixel type, the number of channels, and the current position of the mouse cursor are displayed. The percental scale can be selected from the combo box. It is related to the original size of the image.

There are multiple methods to navigate the zoom window:

Check `Follow Mouse` and move the mouse pointer over the image to select the zoomed area. Click once to keep the currently displayed area in the zoom window, when the mouse cursor moves out of the image window. Or, uncheck `Follow Mouse` and click (or drag) inside the image to select the zoomed area. The red square in the center of the zoom window indicates the position of the mouse cursor. The corresponding coordinates are also displayed at the bottom of the window.

You can select a particular pixel by single-clicking on it with the left mouse button. The zooming tool stores this position internally, and will redisplay the thus selected part of the image object when you leave the graphics window. This enables you to have a meaningful display in the zooming tool whenever you want to do actions outside of the graphics window.

For finer control of the zoomed area, click inside the zoom window to give it the focus and use the cursor keys to move pixel-wise. Press and hold the Alt key and use the cursor keys to move ten pixels at a time. Click inside the zoom window to move relative to the center position. For example, clicking ten pixels above the center will move the view up ten pixels.

The lower part of the window contains a gauge to display the gray value of the center pixel graphically. The range goes from 0 (left) to 255 (right). Normally, the gray value of the first channel is displayed with a black bar. For images with multiple channels the gauge is split accordingly to show individual bars for each channel. Thus, for color images in RGB-space (three channels with red, green, and blue values) three colored bars are used. If the gray value is below 1, the gauge is light gray (background). If the value is above 255, the gauge is dark gray or colored for RGB images.

Above the gauge, the gray values are displayed as numbers. Up to five channels are displayed. If more than five channels are present, the remaining channel values are truncated.

Next to the gauge, the coordinates of the mouse position are displayed. Below these, the image size, pixel type, and the number of channels are shown.

The button next to the scale combo box enlarges the zoom window so that partially visible pixels at the border become fully visible.

6.13 Interacting with Plot Windows

HDevelop uses plot windows for the visualization of data in several areas. These plot windows are all handled in a similar way, which is described in this section. Differences specific to a given task are described in the corresponding sections listed below:

- Inspection of 1D functions and tuple data (section 6.7.7 on page 152).
- Visualization of gray value histograms (section 6.14 on page 173).
- Visualization of line profiles (section 6.17 on page 180).
- Visualization of feature histograms (section 6.15 on page 178).
- Visualization of fuzzy measure graphs (section 7.4.5.2 on page 255).

An example plot window is illustrated in figure 6.90 on page 174.

6.13.1 Mouse-based Operations

The following mouse-based operations are supported to manipulate the plot display:

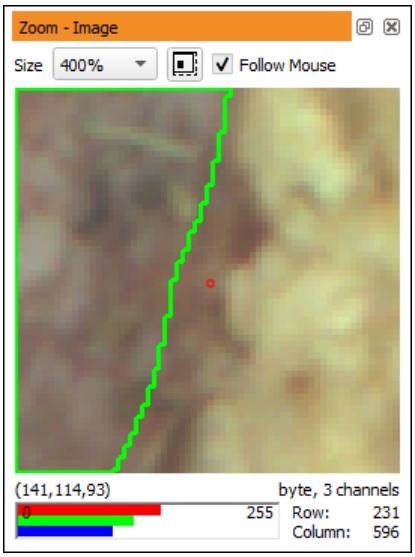


Figure 6.87: Zoom.

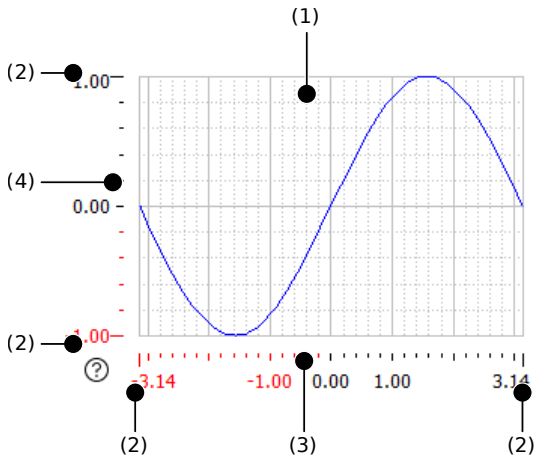


Figure 6.88: Example plot window: (1) plot area, (2) axis limits, (3) x-axis, (4) y-axis.

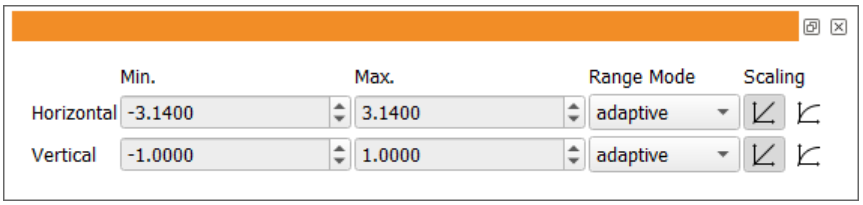


Figure 6.89: Plot bounds.

Operation	Description
Mouse wheel on plot area	Zoom in and out.
Shift-drag on plot area	Zoom into a rectangular region.
Ctrl+mouse wheel	Zoom x-axis.
Shift+mouse wheel	Zoom y-axis.
Drag plot area	Select the visible part of the graph.
Drag on an axis	Restrict the movement to the corresponding axis.
Mouse wheel on an axis	Zoom in and out on the corresponding axis only.
Drag min/max (4) on an axis	Modify the displayed limits of an axis.
Ctrl	Hide values temporarily while held.

6.13.2 Setting the Plot Bounds Parametrically

The displayed range of the plot window can also be specified parametrically by pressing b or selecting from the context menu.

The `Range Mode` determines if and how the displayed data range is updated when new data becomes available.

adaptive The displayed data range is automatically adjusted depending on the displayed data.

increasing The displayed data range is allowed to grow if new data becomes available.

user-defined The displayed data range is never adjusted automatically, even when the data changes drastically.

The horizontal and vertical scaling may be set to linear or logarithmic mode independently.

6.13.3 Plot Window Context Menus

The following context menu entries are valid for all plot windows. Depending on the plot window type, additional entries may be available which are described in the corresponding sections.

X-Axis

	Action	Shortcut	Description
	`Reset Bounds`	w or Ctrl+Shift+W	Set width to default.
	`Linear Scale`		Set x-axis to linear scale.
	`Logarithmic Scale`		Set x-axis to logarithmic scale.
	`User-defined Range`		Freeze x range
	`Increasing Range`		Let x range grow on demand.
	`Adaptive Range`		Let x range grow or shrink on demand.

Y-Axis

	Action	Shortcut	Description
	`Reset Bounds`	h or Ctrl+Shift+H	Set height to default.
	`Linear Scale`		Set y-axis to linear scale.
	`Logarithmic Scale`		Set y-axis to logarithmic scale.
	`User-defined Range`		Freeze y range.
	`Increasing Range`		Let y range grow on demand.
	`Adaptive Range`		Let y range grow or shrink on demand.

Plot Area

	Action	Shortcut	Description
	`Reset Bounds`	r or Ctrl+Shift+R	Reset width and height to default.
	`Zoom In Mode`	+ or Ctrl+Shift++	Zoom in (both axes).
	`Zoom Out Mode`	- or Ctrl+Shift+-	Zoom out (both axes).
	`Enter Bounds...`	b or Ctrl+Shift+B	Enter vertical and horizontal bounds parametrically (see above (page 170)).
	`Show Mouse Position`	p or Ctrl+Shift+P	Visualize the mouse position with a cross hair.
	`Show Function Value At X`	x or Ctrl+Shift+X	Display the function value at horizontal mouse position.
	`Show Function Value At Y`	y or Ctrl+Shift+Y	Display the function value at vertical mouse position.
	`Show Background Grid`	g or Ctrl+Shift+G	Display grid lines in the background of the plot area.
	`Insert Plot Code for Graphics Window`	Ctrl+Shift+V	Generate code to plot the current view in a graphics window.

6.14 Gray Histogram Window

Synopsis: Display gray value histograms.

See also: Menu `Visualization` ▷ `Gray Histogram`.

The gray histogram window is a sophisticated tool for the inspection of gray value histograms, which can also be used to select thresholds interactively and to set the range of displayed gray values dynamically.

When opening the tool, the histogram of the image shown in the active graphics window is displayed. When the tool is already open, the following means of sending new image data to the tool are available:

- Make another graphics window active or display another image in the active graphics window. Whenever you do so, the histogram of this image is computed and drawn, and the tool records the graphics window from which the image was sent (the originating window).
- Select a graphics window number in `Input and Output` (see below).
- Whenever image data is displayed overlaid with region data in a graphics window, you can click into any of the segmented regions, and the histogram of the image within that region will be computed and shown. If you click into a part of the image that is not contained in any of the overlaid regions, the histogram of the entire image will be displayed.
- The same mechanism is used for regions that have gray value information, e.g., image objects created by `reduce_domain` or `add_channels`. Here, the histogram of the image object you click into will be displayed.

Gray Histogram Display

The main part of the tool is the plot area, in which the histogram of the image is displayed in blue (2). Images with three channels are displayed in RGB mode by default. The vertical green and red lines denote the minimum and maximum selected gray value of the histogram, respectively. They can be dragged with the mouse. The gray values on which the two vertical lines lie are displayed next to the lines in the same color.

The x-axis below the histogram (3) represents the gray values in the image. For byte images, the range is always 0...255. For all other image types, e.g., real images, the x-axis runs from the minimum to the maximum gray value of the image, and the labeling of the axis is changed accordingly.

The y-axis (1) represents the frequency of the gray values.

The two upward pointing arrows on the x-axis denote the maximum and minimum gray value of the image. The two rightward pointing arrows on the y-axis denote the maximum and minimum frequency of the displayed histogram. This data is also displayed in textual form within the `Statistics` area of the display. The peak of the histogram,

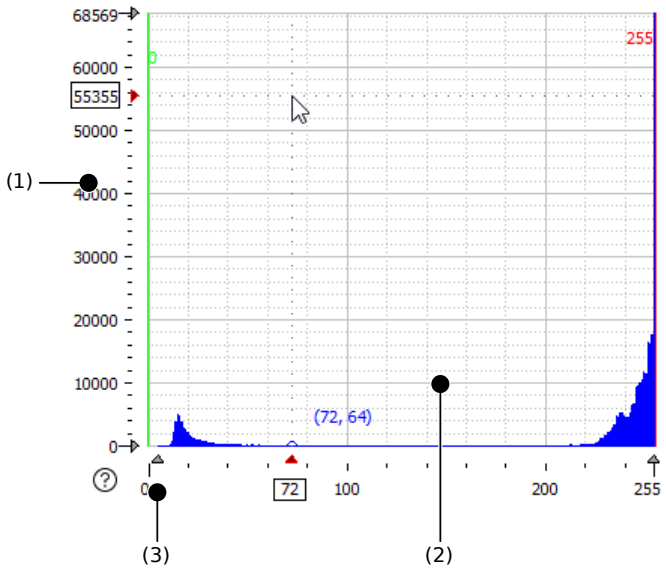


Figure 6.90: Gray histogram: (1) y-axis, (2) plot area, (3) x-axis.

i.e., the gray value that occurs most frequently is also displayed in the statistics (see below). For int4, int8, or real images, the peak value is displayed as a value range in the `Statistics`. That is, the range of input values is divided in quantization steps to obtain a meaningful histogram, and, as a consequence, the histogram's "peak value" may actually represent a whole range of input values.

Whenever new image data is evaluated in the gray histogram window, the adaptation of these values depends on the selected adaptation mode, which can be set independently for horizontal and vertical ranges:

- `adaptive`

  In this mode, the upper and lower boundary of the displayed gray values will always be adapted when a new image is displayed. The maximum and minimum value for the threshold bars (green and red) are also fixed to the maximum gray value of the type of image currently displayed.

  Note that if you are using 8-bit and 16-bit images in a mixed mode, the histogram will constantly be reset. Thus, it is not possible to display a 16-bit image, set thresholds, then display an 8-bit image and keep the threshold values of the 16-bit image.

  In adaptive mode, the displayed data range depends on the image type:

  - int1, byte, direction, and cyclic images automatically use their maximum data range (e.g., byte images use the range 0 to 255).
  - int2, uint2, int4, and int8 images use limits that are based on the actual data range but rounded to powers of two.
  - int2 and uint2 images can be forced to an explicit bit range using the `set_system` parameter "int2_bits".
  - real images use limits that are based on the actual data range, but rounded to values that are round in base 10 (e.g., 0.1, 0.2, 0.5, or 100, 200, 500).

- `increasing`

  In this mode, only the upper boundary of the displayed gray values will be adapted and it will only increase, but never decrease. This for instance is useful when first inspecting 8-bit images, but then switching to 16-bit images. In this situation, the histogram will simply display the 16-bit gray value range after displaying the first 16-bit image.

  In this mode, the minimum and maximum value of the threshold bars are not limited to the currently displayed image type. The reason is simple: This mode allows you to inspect images of a different data type with the same threshold values. If the values were always limited, the histogram would "forget" the values like in the adaptive mode.

- `user-defined`

In this mode, the boundaries are not adapted automatically (but can be changed manually). This mode is also suitable for scenarios with images of mixed data types.

Like in the mode `increasing`, the minimum and maximum value of the threshold bars are not limited to the currently displayed image type.

`Histogram Options`

These controls define the visible area of the histogram and the way it is displayed.



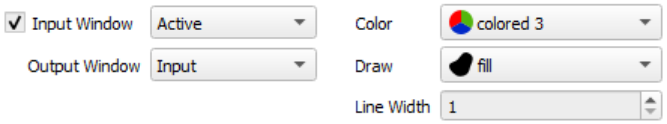
- `Quantization`: Display the histogram quantized. The bucket size can be specified with the slider or entered into the spinner box. Click `Auto Select` to let HDevelop select a suitable bucket size automatically.
- `Smoothing`: Display the histogram smoothed. The smoothing factor can be specified with the slider or entered into the spinner box. The check box specifies whether smoothing is applied or not.

`Range Selection and Code Generation`



See section 6.14.2.

`Input and Output`



`Input Window` specifies the graphics window of which the gray value histogram is displayed.

`Output Window` specified the graphics window that is used for the visualization of threshold or scale operations (see below). The visualization style is specified with the following settings:

- `Color` (page 91)
- `Draw` (page 91)
- `Line Width` (page 91)

Sometimes, it is desirable to suppress the updating of the histogram when new image data is available, e.g., if you want to select thresholds for a gradient image, but want to visualize the original image along with the segmentation (see below). In that case you can freeze the histogram by unchecking `Input Window`. The currently displayed histogram is preserved until `Input Window` is checked again in which case the histogram will be re-calculated from the selected graphics window.

6.14.1 Context Menus

The common context menu entries are described in section “Plot Window Context Menus” on page 170. The context menu entries specific to the gray histogram window are listed below:

	Action	Shortcut	Description

Plot Area

	Action	Shortcut	Description
	`Fit Data Range`	d or Ctrl+Shift+D	Set the lower and upper bounds so that the histogram is displayed in its entirety.
	`Zoom To Selection`	s or Ctrl+Shift+S	Zoom to the range between the green and red line.
	`No output`		Do not process the selected gray values.
	`Highlight Selection`		Highlight the selected gray values.
	`Scale Selection`		Scale the selected gray values.
	`InsertCode`		Generate code for the selected operation.

X-Axis

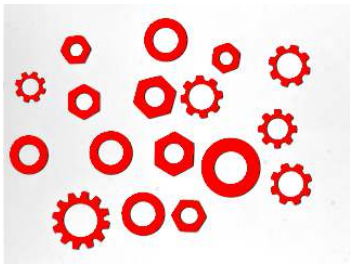
Note that histogram quantization is still performed using equal-sized linear bins if the x-axis is set to logarithmic scale.

Y-Axis

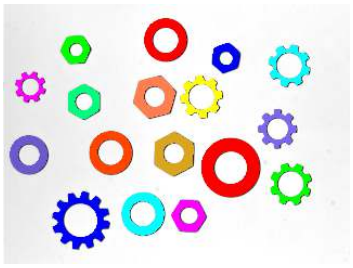
In logarithmic mode no negative values are permitted when dragging the plot area using the mouse.

6.14.2 Interactive Visual Operations

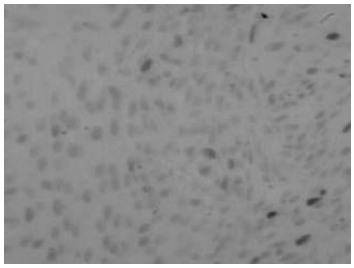
The selected range of gray values can be used for two major purposes: Thresholding (segmentation) and scaling the gray values. This is illustrated using the images *rings/mixed_03.png*, and *meningg5.png*.



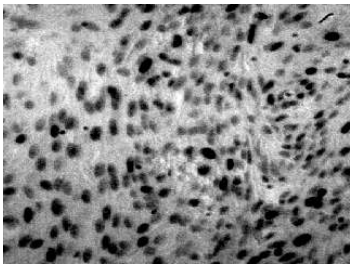
threshold



threshold + connection



no scale



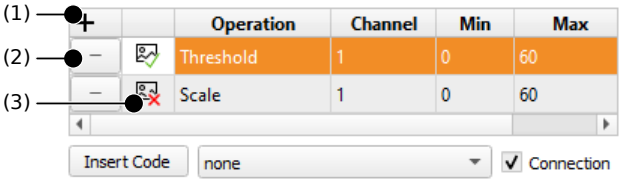
scale

The setting of `Output Window` specifies the graphics window that is used to visualize the gray values between the green line and the red line: You can select the originating window (`Input`), the active graphics window (`Active`), or an arbitrary window ID from the list.

The type of visualization is specified in the table below the histogram. Click the + button (1) to add a new operation to the table. Click the - button (2) to remove an operation from the table. The column `Operation` specifies the operation that is applied to a selected range of gray values (threshold or scale, see below). To visualize a specific operation, click the corresponding icon next to the operation (3).

When a multi-channel image, e.g., a RGB color image, is sent to the tool, by default the histogram of the first channel is displayed. The column `Channel` lets you select the channel from which to compute the histogram. For RGB images, `Channel` may also be set to the special mode RGB which shows a combined histogram of all three channels.

The columns `Min` and `Max` correspond to the position of the green and the red line, respectively. Each operation may specify its own range of gray values.



6.14.2.1 Threshold Operation

The image from which the histogram was computed is segmented with a `threshold` operation with the selected minimum and maximum gray value.

With the three combo boxes `Color`, `Draw`, and `Line Width` in the `Input and Output` section of the window you can specify how the segmentation results are displayed (see also `Colored` (page 90), `Draw` (page 91), and `Line Width` (page 91)).

If you want to select threshold parameters for a single image, display the image in the active graphics window and open the histogram tool. For optimum visualization of the segmentation results, it is best to set the visualization color to a color different from black or white. Now, set `Operation` to `Threshold` and interactively drag the two vertical bars until you achieve the desired segmentation result. The parameters of the threshold operation can now be read off the two vertical lines.

If you want to select threshold parameters for an image that is derived from another image, but want to display the segmentation on the original image, e.g., if you want to select thresholds for a gradient image, two different possibilities exist. First, you can display the derived image, open the histogram tool, deselect `Input Window`, display the original image, and then select the appropriate thresholds. This way, only one window is needed for the visualization.

Alternatively, you can display the derived image in one graphics window and the original image in another. Activate the first graphics window image, and make sure `Input Window` is checked so that the corresponding gray value histogram is calculated. Afterwards, `Input Window` can be turned off to prevent the histogram from being updated. In the gray histogram window set `Output Window` to the window ID of the second graphics window and select your thresholds.

Multiple Threshold Operations You can combine as many threshold operations as you like. If multiple operations are visualized at the same time, the display depends on the combo box below the table of operations: If `none` is selected, the results of the different threshold operations are displayed independently. If `union` is selected, the results are combined to a single region. If `intersection` is selected, only the common pixels from all results are visualized.

Connected Regions

Clicking `Connection` displays the connected regions of the selected gray values in the style specified with `Color`, `Draw`, and `Line Width`.

This display mode is similar to a plain threshold operation. Additionally, it performs a `connection` operation. The separate regions can only be distinguished if `Color` is set to `colored 3`, `colored 6`, or `colored 12`.

Click the button `Insert Code` to generate HDevelop code that performs the currently visualized threshold operation(s) in your program. The code is inserted at the IC.

The resulting regions of the threshold (and connection) operation can be used as input to the feature histogram window or the feature inspection window if the gray histogram window is kept open. These windows are described in the next sections.

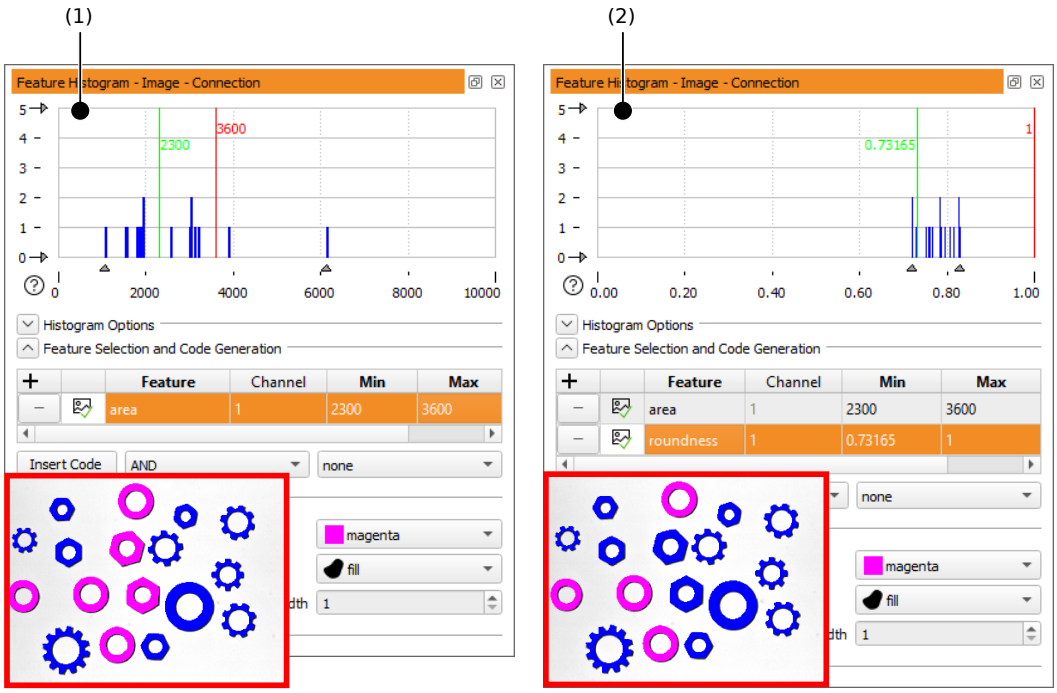


Figure 6.91: Combining different features selections: (1) first, select regions of similar size, (2) then, restrict the selection to round regions.

6.14.2.2 Scale Operation

The scale operation maps the gray values between the green line and the red line to the full range (usually 0...255). See also `scale_image`.

The gray values of the image are scaled such that the gray value 0 of the scaled image corresponds to the selected minimum gray value and the gray value 255 to the selected maximum gray value. Again, the combo box `Output Window` determines the graphics window, in which the result is displayed. This mode is useful to interactively set a "window" of gray values that should be displayed with a large dynamic range.

You can define as many scale operations as you like, but only one of them may be visualized in the graphics window at the same time.

Click the button `Insert Code` to generate HDevelop code that performs the currently visualized scale operation in your program. The code is inserted at the IC.

6.15 Feature Histogram Window

Synopsis: Interactive inspection of feature histograms.

See also: Menu `Visualization` ▷ `Feature Histogram`.

This window provides a sophisticated tool for the inspection of feature histograms. In contrast to the gray value histogram described in the previous section, this tool does not inspect individual pixels, but regions or XLDs; for these iconic objects, it displays the distribution of values of a selected *feature*, e.g., the area of an XLD or the mean gray value of the pixels within a region. The feature histogram can also be used to select suitable thresholds for the operators `select_shape` and `select_shape_xld` interactively. Similar to the gray histogram tool, the interactive selection can be translated into generated HDevelop program code.

Upon opening, the tool displays the histogram of the area (default feature selection) of the regions or XLDs that were displayed most recently in the currently active graphics window. You can select various features in the combo box `Feature`; Further information about region features can be found in section "Feature Inspection Window" on page 179.

See figure 6.91 for an illustration. First, all objects (regions) of a certain size (`area`) are selected. Then, the selection is refined by adding further restrictions. In this example, the final selection should only include round objects, i.e., regions with a high `roundness` feature. The following code would be generated if you clicked the button "`Insert Code`" in this example:

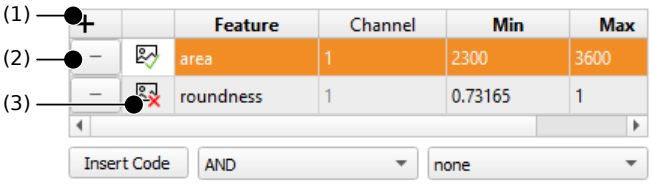
```
select_shape (Connection, SelectedRegions, ['area','roundness'], 'and',
              [2900,0.72], [3900,0.79462])
```

Most parts of the tool are built up similarly to the gray value histogram, which is described in detail in section "Gray Histogram Window" on page 173 (Menu `Visualization` ▷ `Gray Histogram`). Reading this description beforehand is highly recommended. In the following, we concentrate on points specific to the feature histogram. An important point regards the "source" of the regions or XLDs: The feature histogram is calculated for the regions or XLDs that were displayed most recently in the graphics window. Thus, if you display an image, and there are no regions or XLDs, the histogram remains "empty". As soon as you display regions or XLDs on top of an image, the histogram is calculated. If you display regions or XLDs without an image, you can still calculate feature histograms, but only for shape features. Please keep in mind that only the most recently displayed regions or XLDs are the source of the histogram, not all objects currently displayed in the graphics window!

The histogram itself is displayed with the horizontal axis corresponding to the feature values and the vertical axis corresponding to the frequency of the values, i.e., to the number of regions or XLDs with a certain feature value.

When comparing feature histograms to gray value histograms, you will note a typical difference: Because in most cases the overall number of regions or XLDs is much smaller than the overall number of pixels, feature histograms often consist of individual lines, most of them having the height 1. Of course, this effect depends on the selected feature: For features with floating-point values, e.g., the orientation, the probability that two regions or XLDs have the same feature value is very small, in contrast to features with integer values, e.g., the number of holes.

You can influence the calculation of the histogram with the slider `Quantization`. The selected value is used to discretize the horizontal axis: Instead of determining the frequency of an "exact" feature value, regions with feature values falling within discrete intervals are summed. Graphically speaking, the horizontal axis is subdivided into "bins" with a width equal to the value selected with the slider `Quantization`.



You can add additional features using the + button (1), or remove features using the − button (2). As with the gray histogram operations, each selected feature has to be enabled (3) to visualize the selection in the graphics window.

6.16 Feature Inspection Window

Synopsis: Inspection of shape and gray value features of individual regions.

See also: Menu `Visualization` ▷ `Feature Inspection`.

This window provides a tool for the convenient inspection of shape and gray value features of individual regions and XLDs. It can, for instance, be used to determine thresholds for operators that select regions based on these features, e.g., `select_shape` or `select_gray`.

The strategy to determine the data from which to compute the features is very similar to that of the gray histogram inspection window (see section "Gray Histogram Window" on page 173). You can display an image or region by double-clicking on it in the variable window or you can select a region or an image which is already displayed by single-clicking it. If you display or click into an image, the gray value features of the entire image will be calculated. If you click into a region that is not underlaid with an image, only the shape features of this region will be displayed. If you click into a region that is underlaid with an image or into a region that has gray value information (e.g., from `reduce_domain` or `add_channels`), both the shape and gray value features of that region will be displayed. Finally, if you have overlaid an image with a region, but click into a part of the image that is outside the region, only the gray value features of the entire image will be calculated.

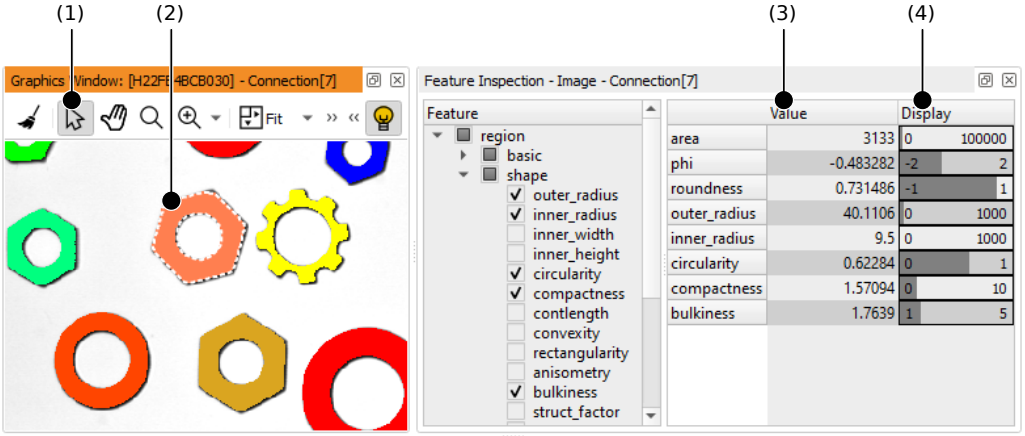


Figure 6.92: Inspection of selected features: (1) select, (2) selected region, (3) feature value of selected region, (4) range visualization.

Use the “select” tool of the graphics window to select a region or XLD. The selected region or XLD is highlighted in the graphics window. The corresponding variable name and index are displayed in the title of the feature inspection window.

The gray value features of a multi-channel image are calculated from all channels independently.

The tree on the left side of the feature inspection window groups the features into several categories. At the top-most level, the following groups of features are distinguished:

- Region features: This group contains features that describe the selected region, e.g., area, center, and orientation.
- Gray value features: The feature values of this group are calculated from the gray values of the image *under* the selected region, e.g., minimum and maximum gray value, mean gray value, anisotropy and entropy.
- XLD features: This group contains features that describe the selected XLD (e.g., its dimensions or shape properties).

You can select the features to be inspected by ticking the corresponding check boxes in the tree. The selected features are displayed on the right side of the window. For each feature the calculated value of the selected region or XLD is displayed (or the value for the entire image). The current value is also visualized as a gauge in a value range that can be set to the desired values. Simply select `Show Minimum/Maximum`, which is available in the context menu of the right side of the window.

See figure 6.92 for an illustration of a feature inspection. The range for the *area* feature has been set to [2000, 7000]. Individual rings can be inspected by selecting them in the graphics window.

Moving the mouse pointer over a feature value displays a tool tip. It shows the name and short description of the HALCON operator used for the calculation of that value. Using the context menu, you can insert the corresponding operator into the operator window.

6.17 Line Profile Window

Synopsis: Display line profile of active graphics window.

See also: Menu `Visualization` ▷ `Line Profile`.

Selecting this entry opens a tool for the detailed inspection of a gray-value profile of a linear or circular ROI (see figure 6.93). Using the line profile is helpful in particular for optimizing edge detection in the Measure Assistant (page 251) or when checking the focus of your camera (see the section ’Focusing Your Camera’ (page 184)). The displayed line profile of the ROI is described in the section ’Line Profile Display’ (page 182). Note that the line profile window is a visualization tool that cannot be used to create any output, like performing changes within the image or producing code.



Figure 6.93: The Line Profile Window.

When opening the line profile without using the Measure Assistant, nothing is displayed in the line profile until an ROI is drawn in the graphics window using the corresponding buttons in the line profile window.

There are various options for linking the line profile to different windows:

- Make another graphics window active, select another window ID in the `Input Window` drop-down menu, or display another image in the active graphics window. Whenever you do so, the line profile of the ROI in this image is computed and visualized, and the tool stores the graphics window from which the image was sent (the originating window).
- A Measure Assistant can be selected as data source by activating the checkbox `Measure Assistant` under `Input Window` within the line profile window and choosing the correct assistant from the drop-down menu if several assistants are open at the same time.
- The line profile can be opened from the Measure Assistant window by clicking on the `View Line Profile` button on the `Edges` tab.

6.17.1 ROI Menu of the Line Profile Window

If the Measure Assistant (page 249) is not selected as source of an ROI, a new ROI can be created and edited using the ROI menu buttons above the line profile display. Those buttons allow you to draw either a linear or a circular ROI, delete all ROIs or view the ROI shape. Once a ROI has been created with the `Draw Line` or `Draw Circular Arc` buttons, those ROI creation buttons are grayed out because the line profile window can only handle one ROI at a time.

6.17.2 Line Profile Display

The main part of the tool is the area in which the gray-value profile of the image along an ROI is displayed in blue.

The horizontal axis represents the length of the ROI in pixels and therefore gives the position of the gray values along the ROI.

The dynamic parts of the line profile area are the two colored lines, which can be manipulated. The vertical green and red lines denote the minimum and maximum selected position along the ROI, respectively. Those lines are also displayed in the active graphics window. Their visualization can be adapted under `Output`. Another dynamic part is the vertical coordinate axis, which displays the gray values in the image. For byte images, this ranges from 0 to 255. As it comprises only the gray-value range between darkest and the brightest pixel, these values do not usually start with 0 and end with 255. For all other image types, e.g., real images, the horizontal axis runs from the minimum to the maximum gray value of the image and the labeling of the axis is changed accordingly.

Initially, the line profile is displayed at full vertical range, i.e., up to the peak value. With the buttons to the left of the line profile display, you can modify the displayed part:

The common interaction with plot windows is described in section “Interacting with Plot Windows” on page 169.

6.17.3 Context Menus

The common context menu entries are described in section “Plot Window Context Menus” on page 170. The context menu entries specific to the line profile window are listed below:

Plot area

	Action	Shortcut	Description
	`Fit Data Range`	d or Ctrl+Shift+D	Set the lower and upper bounds so that the line profile is displayed in its entirety.
	`Zoom To Selection`	s or Ctrl+Shift+S	Zoom to the range between the green and red line.

6.17.4 Line Profile Options

`Smoothing`: Display the profile smoothed. You can smooth the profile before displaying it by specifying a smoothing factor with the slider in the spinner box and by clicking the checkbox. If smoothing has been applied with the Measure Assistant (page 249), those values are automatically adopted. The smoothing can then be performed in the line profile window as well and likewise affects the Measure Assistant's smoothing (see figure 6.94).

`Derivative`: Display the derivation of the line profile.

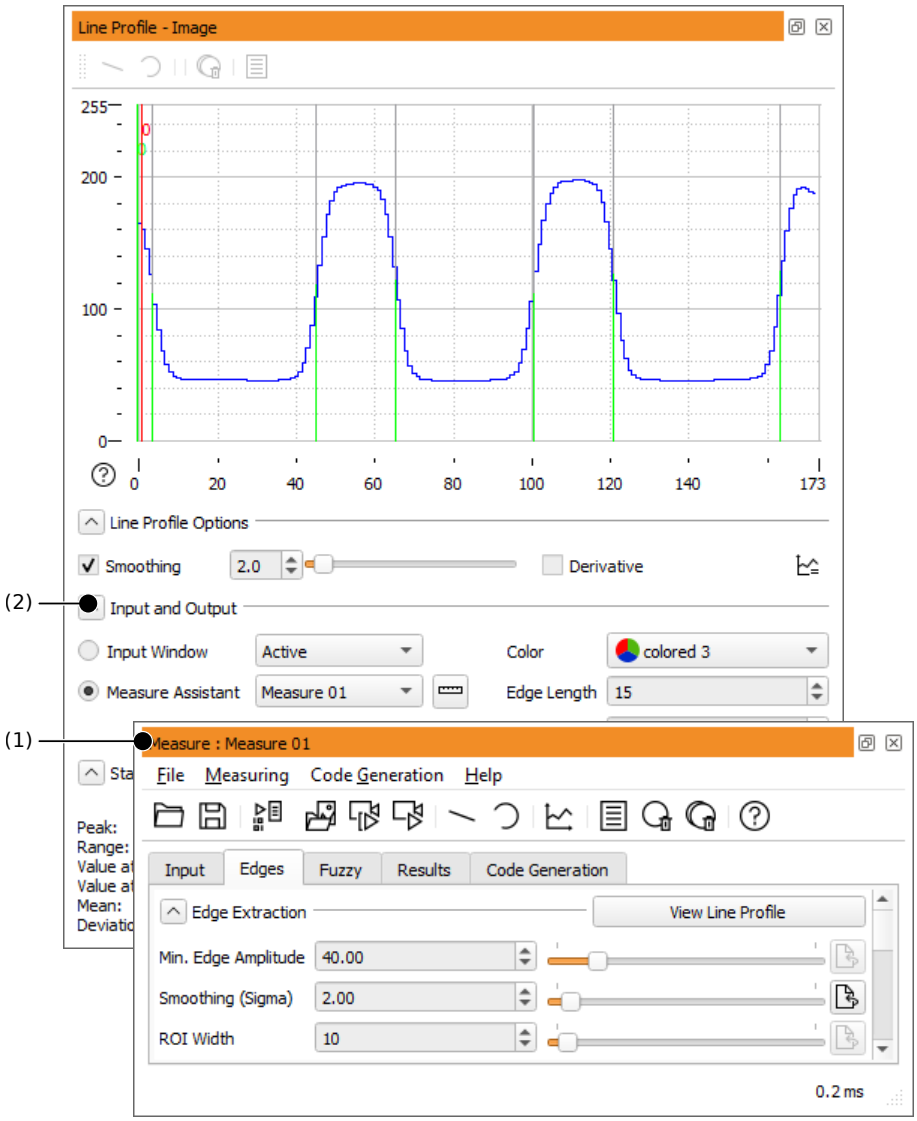


Figure 6.94: (1) Display and modification of the smoothing via the Measure Assistant, (2) Choosing a data source and options for the line profile.

`Force minimum line profile width`: Reduce the line profile visualization to a minimum width by using the button .

6.17.5 Input and Output

This section lets you choose the source of your image and therefore the source of the ROI. By default, the active graphics window is chosen into which an ROI can be drawn using the ROI menu of the line profile window. The ID of another graphics window can be selected from the drop-down menu. If a Measure Assistant (page 249) is activated, one assistant can be chosen from the drop down menu or it can be opened by activating the `Use Measure Assistant as Data Source` button.

The remaining buttons let you choose the visualization of the marker lines the line profile displays within in your active ROI. You can select the color and length of your lines as well as their width. Changing those output features can be necessary to achieve optimum visibility within an image.

6.17.6 Statistics

The `Statistics` section of the line profile window displays the values from the line profile display above. It includes values for the position (`x Value`) as well as the gray values. It therefore informs you where interesting gray values can be found. Those gray values include

- the `Peak`, which marks the position of the highest gray value,
- the `Range`, defining the length of the ROI as well as the range of gray values along the ROI.
- `Value at Min` and `Value at Max`, defining the position and gray value of the above determined selection (that was localized with the green and red line) as can also be seen in the active graphics window.
- There are two further values that only concern the gray values: `Mean` defines the mean gray value while `Deviation` is the average deviation.

6.17.7 Focusing Your Camera: How to Test Images for Sharpness

When focusing your camera, it might help to take a quick look at the gray-value transitions along a line within the image to see whether the edges are sharp or still a bit blurry. While sharp images are defined by abrupt changes between dark and bright gray values, no abrupt changes but rather gray-value transitions can be found in blurry images (see figure 6.95).

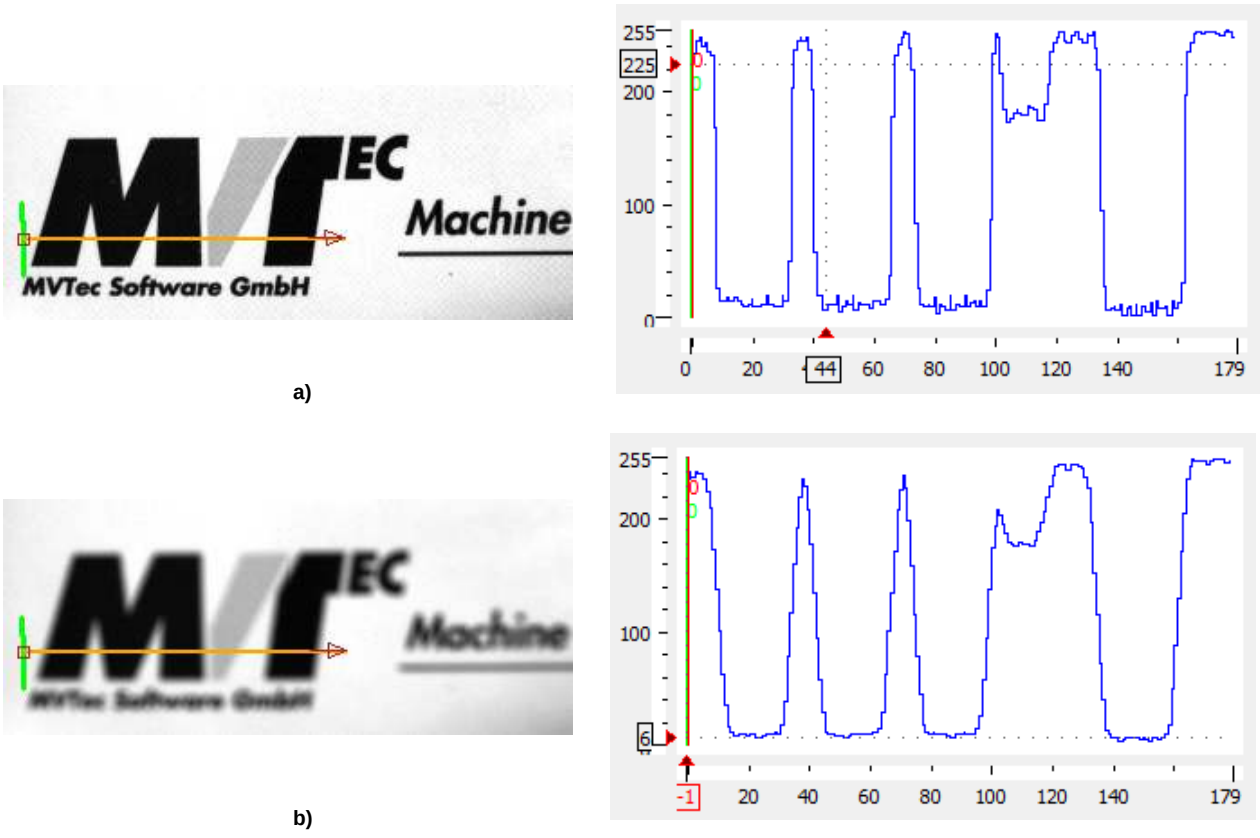


Figure 6.95: a) profile in a sharp image; b) profile in an unsharp image.

6.18 OCR Training File Browser

Synopsis: Browse and manage OCR training files.

Selecting this entry opens a tool for inspecting and modifying training files. With this tool, you can for example eliminate errors made in the teaching process, e.g. if a sample has been assigned to a wrong symbol (please follow the links for a definition of the terms *symbol* (page 263) and *sample* (page 263)).

This tool is a useful addition to the OCR Assistant (page 262) but can also be used independently for any OCR application. It can be opened either via the corresponding toolbar button, from within the OCR assistant (page 262) or via the drop-down menu Visualization ▷ Tools ▷ OCR Training File Browser.

Note that if the OCR Training File Browser is used in combination with the OCR Assistant (page 262), the samples are displayed inverted if `Light-On-Dark` has been chosen as `Symbol Appearance` within the assistant. This has an effect on the zoomed sample visualization in the lower left window as well as the thumbnail view the right window.

The following sections will introduce you to the OCR Training File Browser by

- providing an overview over the different windows within the Training File Browser,
- explaining typical steps of using the tool in an application,
- giving an overview over all possible actions (page 187) within the training file browser.

6.18.1 Windows of the Training File Browser

The design of the OCR Training File Browser

The OCR Training File Browser is composed of three windows (see figure 6.96). The upper left window lists training files and symbols and is described in the section 'Training File Window'. Underneath this window, the 'Zoomed Sample Window' allows you to view the magnified image of a selected sample. The 'Sample Inspection Window' on the right lists details for selected samples to inspect the training results.

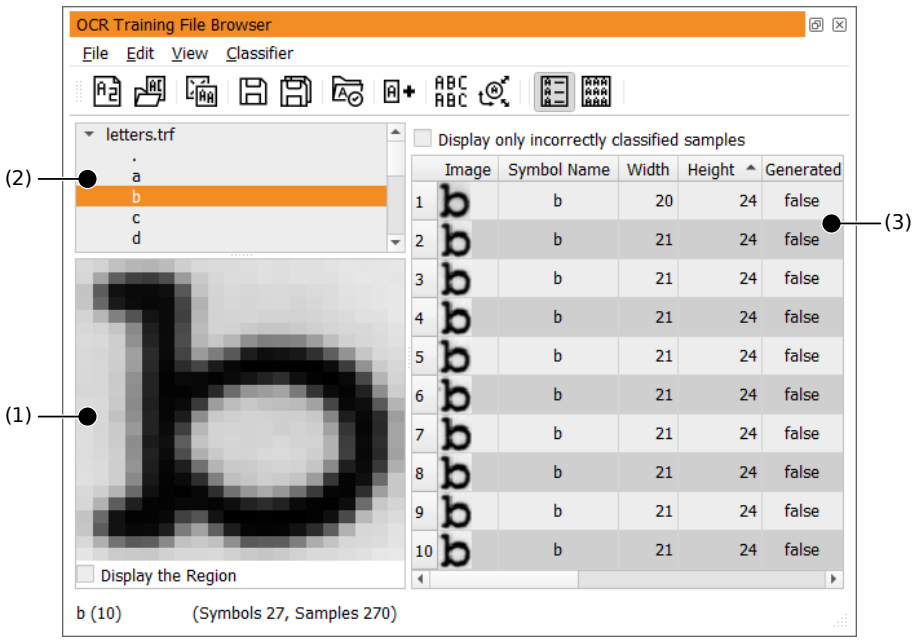


Figure 6.96: The OCR Training File Browser: (1) zoomed sample window, (2) training file window, (3) sample inspection window.

6.18.1.1 Training File Window

This window of the OCR Training File Browser enables you to view and edit training files (page 187) and symbols (page 187).

6.18.1.2 Zoomed Sample Window

This window displays the symbol that is selected in the Sample Inspection Window. If a symbol has been selected in the Training File Window, the first sample is displayed by default.

6.18.1.3 Sample Inspection Window

The Sample Inspection Window enables you to view and edit samples (page 188). It displays a table containing the following information about each sample that belongs to a symbol or a training file that is selected in the Training File Window (page 185).

The column

- `Image` displays the iconic sample,
- `Symbol Name` displays the class the sample was assigned to,
- `Width` displays the width of the iconic sample (in pixels),
- `Height` displays the height of the iconic sample (in pixels),
- `Generated` displays whether the sample is a generated variation (page 189) or not,
- `Read Symbol` displays the result of reading the sample with the selected OCR classifier,
- `Confidence` displays the confidence, i.e., a measure for the reliability of the read symbol (0 - no confidence, 1 - highest confidence), and
- `Correct` displays if a sample has been read correctly and has two states, 'true' means that the values of `Symbol` and `Read Symbol` are identical and 'false' means that those values differ. If the sample has not been classified yet, the value stays 'true'.

6.18.2 Steps for working with the OCR Training File Browser

This section quickly guides you through a possible application workflow with the OCR Training File Browser. More detailed information on the functionality of the OCR Training File Browser can be read in the sections corresponding to each of the windows within the OCR Training File Browser.

1. *Open the Training File Browser* either via the corresponding toolbar button or from the `OCR Classifier` tab within the OCR Assistant (page 262).
2. *Load an existing training file or create a new training file* via the corresponding toolbar buttons or the corresponding entries in the drop-down menu `File`. If several training files should be used or inspected, all of them can be loaded into the OCR Training File Browser.
3. *If required, it is possible to add samples to the training file with the training file functionality of the OCR Assistant (page 262)* if required.
4. *Choose a classifier* via the drop-down menu `Classifier` ▷ `Load Classifier` to classify your samples. If you have an own or previously trained classifier, you can load it. Otherwise one of the trained OCR classifiers provided by HALCON can be selected.
5. *Inspect and/or edit the contents of training files*. This may include adding new samples, deleting samples, combining existing samples for a new training file or adding sample variations (page 189) as well as checking for classification problems. Use the Training File Window (page 185) to edit the training file and the Sample Inspection Window to inspect and edit samples.
6. *Save changes*, e.g. via `File` ▷ `Save Training File`.
7. *Use the training file* within an OCR application or continue to prepare an application with the OCR assistant (page 262).

Note that all actions can be undone/redone by selecting `Undo` or `Redo` in the menu `Edit` to reverse or repeat changes, respectively.

6.18.3 Actions within the Training File Browser

The following sections describe how to manipulate different levels of training data, namely

- training files,
- symbols, and
- samples.

6.18.3.1 Training Files

The Training File Window (page 185) provides the following options for editing training files:

- *Open/Close training files*: When opening the OCR Training File Browser via the OCR Assistant (page 262), the current training file is automatically loaded. Otherwise, training files can be loaded or closed via the corresponding entries in the menu item `File` or via the corresponding toolbar buttons.
- *Create training files*: Training files can be created via the entry `New Training File` in the menu `File` or via the corresponding toolbar button.

6.18.3.2 Symbols

The Training File Window (page 185) provides the following options for editing symbols:

- *Add a symbol*

  To add a new symbol to a training file,
  - press the toolbar button `Add Symbol Name` or
  - select the menu entry `Edit` ▷ `Add Symbol Name` .
- *Delete a symbol*

  Select the symbol you want to delete and
  - press Del on your keyboard or
  - select the menu entry `Edit` ▷ `Delete`.
- *Rename a symbol*

  To assign the samples to another symbol, just edit the symbol name. This action is equal to moving the symbol (and therefore all samples that are assigned to this symbol) via drag and drop to another symbol. Note that it is also possible to assign single samples to a new symbol. To learn more about assigning samples to a new symbol class, read the information about modifying samples in the paragraph 'Improve your training file' in the section Sample Inspection Window (page 186).
- *Copy a symbol*

  A symbol and all samples that are assigned to this symbol can be copied to another training file. To copy and paste a symbol,
  - drag and drop the symbol while pressing Shift or
  - press Ctrl+C followed by Ctrl+V, or
  - select the menu entry `Edit` ▷ `Copy` followed by `Edit` ▷ `Paste`.
- *Move a symbol*

  A symbol can also be moved
  - to another symbol in the same training file,
  - to another symbol in a different training file, or
  - it can also be added to a different training file by moving it there via drag and drop.

  Moving a symbol (and therefore all samples that are assigned to this symbol) to another symbol is equal to assigning a new symbol name explicitely.

6.18.3.3 Samples

The Sample Inspection Window (page 186) provides the following options for viewing and editing samples:

Get an overview of the samples

Use the sorting mechanisms to get an overview over the samples displayed in this window.

- You can also *sort the results within the Sample Inspection Window* in ascending or descending order. To do this, simply click on the feature that should be used for sorting (e.g. `Width`, or `Symbol Name`) or choose a sorting order in the menu `View` to sort the list. Note that this feature is only available for up to 1000 displayed elements.
- Enable the checkbox `Display only incorrectly classified samples`.to *filter the information displayed for samples that were not classified correctly*.
- *Choose between two general viewing options:* `Detailed View` *or* `Thumbnail`. Select those viewing options either via the corresponding toolbar buttons or via the menu items of the menu `View`. `Detailed View` lists out all the symbol characteristics explained above, whereas `Thumbnail` provides a quick overview over the samples just showing a thumbnail image of each symbol and the symbol that was read.
- *If you click on a sample, its zoomed image is displayed* in the lower left corner of the OCR Training File Browser.
- *Directly view new samples that were saved to the training file via the OCR assistant (page 262).*

Edit the samples

- *Copy a sample*

  Select one or more samples in the Sample Inspection Window (page 186). They can then be copied to
  - another symbol in the same training file or
  - another symbol in a different training file

  in the Training File Window (page 185).

  To copy and paste a sample,
  - drag and drop the sample while pressing Shift or
  - press Ctrl+C followed by Ctrl+V, or
  - select the menu entry `Edit` ▷ `Copy` followed by `Edit` ▷ `Paste`.
- *Move a sample*

  Select one or more samples in the Sample Inspection Window (page 186). They can then be moved to
  - another symbol in the same training file or
  - another symbol in a different training file

  in the Training File Window (page 185) via drag and drop.
- *Delete a sample*

  Select one or more samples in the Sample Inspection Window (page 186). They can then be deleted
  - by pressing Del on your keyboard or
  - via the menu entry `Edit` ▷ `Delete`.

Note that you can also add variations for samples via the `Generate Sample Variations` dialog that can be opened via the corresponding toolbar button.

Check samples for correct classification

- *The following actions describe how to improve the quality of a training file*
  - Sort the samples by `Correctness` to see whether the correct symbol class was assigned. If `Correctness` is 'true', the value of `Read Symbol` is identical to the value of `Symbol` and thus the correct class was assigned. Otherwise the column is marked in red so that samples that do not belong to the symbol can easily be detected.

- Use the checkbox `Display only incorrectly classified samples` to view only samples that were classified incorrectly.
- Sort samples by `Confidence`, as this may provide a hint to classification problems.
- Edit `Symbol Name` to assign a sample to the correct symbol class if necessary. (This action is equal to moving a sample via drag and drop to another symbol in the Training File Window (page 185).)

6.18.3.4 Generating Sample Variations

The best classification results are achieved if the OCR font is trained using real data from the target application. This may however be time-consuming. To speed up the creation of a large number of different samples per symbol, it is possible to vary existing samples.

To add variations for certain samples, select either

- **one sample** in the Sample Inspection Window (page 186) to generate variations for this sample or
- **a certain symbol** in the Training File Window (page 185) to generate variations for all samples that are assigned to that symbol, or
- **the whole training file** can be selected in the tree within the Training File Window (page 185) to generate variations for all samples within this training file.

Then open the `Generate Sample Variations` dialog either via the corresponding toolbar button or via the `Edit` menu.

This dialog allows you to select several types of variations, depending on what is required for the application. In general, we recommend using as many samples as possible. So, if you are in doubt whether a variation type applies or not, you should select it.

The following types of variations can be selected:

- `Rotation` (4 variations for each selected sample)
- `Noise` (4 variations for each selected sample)
- `Stroke Width` (8 variations for each selected sample)
- `Slant` (4 variations for each selected sample)
- `Radial Deformation` (9 variations for each selected sample)
- `Local Deformation` (20 variations for each selected sample)

The `OK` button starts the generation of the new samples. Note that the generation might take some time if lots of variations are selected for a large number of samples. A progress bar shows how far the generation has proceeded. The generated samples can subsequently be viewed and edited in the Sample Inspection Window (page 186) where they are marked by '`true`' in the `Generated` column.

Note that once the training file is saved, the generated samples cannot be recognized as generated ones any more and therefore the `Generated` column will show `false` even for a sample that has been generated.

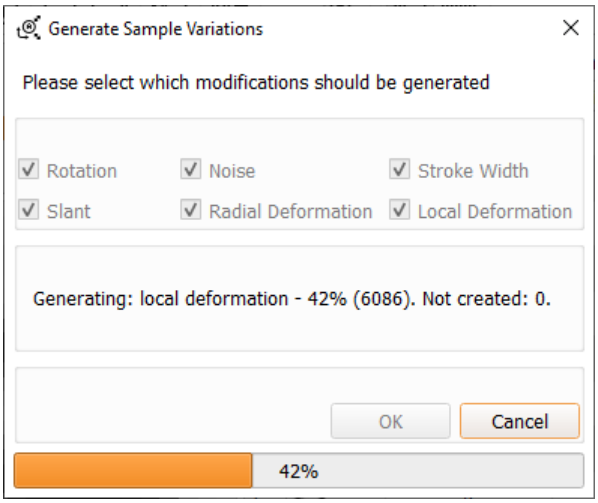


Figure 6.97: The Generate Sample Variations dialog.

6.19 Dialogs

6.19.1 File Selection Dialog

The file selection dialogs opened by actions such as `Open Program...`, `Save`, or `Read Image...` are native windows of the operating system and thus their appearance and internal functionality is beyond HDevelop's control. Their basic functionality is to browse the file system, and to select one or multiple files (or in some cases: directories). Usually, they have two buttons: The one labeled `Open` or `OK` confirms the selection and thus performs the initial action (e.g., loading a file) while the other (labeled `Cancel`) aborts the initial action.

As an example, the dialog Menu `File` ▷ `Open Program...` is shown. A detailed description of the complete functionality is beyond the scope of this manual.

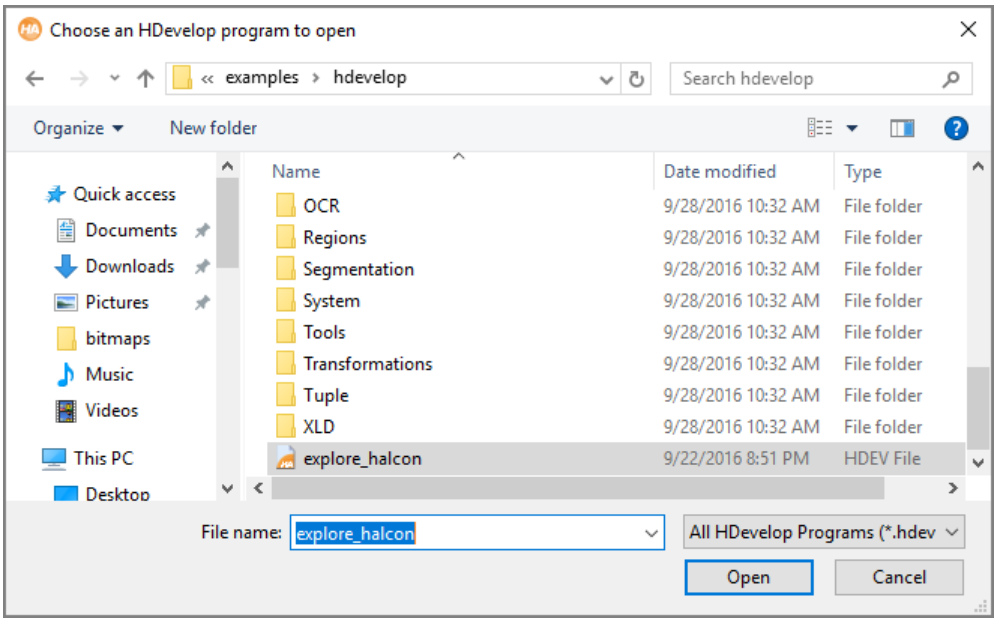


Figure 6.98: Example of a file selection dialog under Windows.

Usually, the dialog is laid out similar to the shown example. The elements at the top allow you to navigate the directory hierarchy. The area to the left provides shortcuts to special directories or system drives. The area to the right provides a file listing of the currently selected directory. At the bottom, the selected file name is displayed or

can be entered directly. The file type selection at the bottom is useful to restrict the file listing to certain files that are of interest in the current context: In this example, only subdirectories and HDevelop programs are displayed.

6.19.2 Unsaved Changes

File operations that will delete the current program (such as loading a new program) trigger a security check. This security check prevents you from deleting the current program accidentally if the program has not been saved. A dialog box appears and asks whether you want to save the HDevelop program before its dismissal:

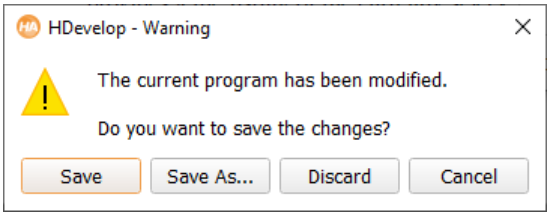


Figure 6.99: Confirmation dialog.

`Save`: Save the current program under its current name and proceed. If no name has been specified yet, a file dialog pops up to enter the name.

`Save As`: Save the current program under a different name and proceed.

`Discard`: Discard unsaved changes and proceed.

`Cancel`: Abort the current action.

Chapter 7

HDevelop Assistants

HDevelop contains assistants for specific machine vision tasks. Each assistant provides a user interface tailored to the requirements of its task. Using this interface, you can interactively set up and configure the assistant to solve a specific machine vision problem. Once the configuration is working satisfactorily, the assistant can be instructed to generate HDevelop code into the current program. You can also save an assistant's configuration for later use.

The following assistants are available:

- `Image Acquisition`: Using this assistant you can generate code to acquire images from different sources (files, directories, image acquisition interfaces).

  The assistant is described in section "Image Acquisition Assistant" on page 194. A tutorial about using this assistant is available in section 3.3 on page 24.

- `Calibration`: Using this assistant you can calibrate your camera and therefore gain information about parameters of the camera system and distortions in the image. Calibrating your system constitutes a preparation for your subsequent application as it provides the basis for you to measure with high precision in the world coordinate system.

  The assistant is described in section "Calibration Assistant" on page 199.

- `Matching`: Using this assistant you can generate code to perform shape-based, correlation-based, descriptor-based and deformable matching in your HDevelop program. You can load a reference image to train a model. Using a selection of test images containing the model you can tweak a set of parameters to find the model in all variations permitted by the application. Furthermore, the parameters can be optimized to increase the processing speed.

  The assistant is described in section "Matching Assistant" on page 217.

- `Measure`: Using this assistant you can perform a 1D measuring. By specifying one or more Regions of Interest and adapting variables, relevant edges or edge pairs between which you want to measure can be determined. Finally, code can be generated and inserted into your application.

  The assistant is described in section "Measure Assistant" on page 249.

- `OCR`: Using this assistant you can perform optical character recognition (OCR). By specifying the region of the text, adapting values, teaching samples and either classifying with an existing classifier or training an own classifier, the text can be located and read. Finally, code can be generated and inserted into your application. Note, the assistant is currently limited to OCR presented in the Solution Guide I, chapter 'OCR' and does not include Deep OCR.

  The assistant is described in section "OCR Assistant" on page 262.

Common Features of all HDevelop Assistants

Some features are common to all HDevelop assistants. First of all, you can open multiple assistants. Assistants of the same type are numbered consecutively, e.g., if you open two image acquisition assistants, they are labeled "`Image Acquisition 01`" and "`Image Acquisition 02`", respectively. When you open a new assistant, a menu entry is added to the top of the menu `Assistants`, from which the corresponding assistant can be restored if it has been closed. The current setup is lost and the menu entry disappears if the associated assistant is exited explicitly (see below). If you want to keep the setup for later sessions, you can always save it to a file.

Different assistants have different menus (usually corresponding to the available tab cards). These menus provide functionality specific to the assistant's task. There are also some menu entries that are available in every assistant. They are described in the following.

`File` ▷ `Load Assistant Settings...` Using this entry, a previous configuration can be loaded from a file which has been generated using the menu entry `Save Current Assistant Settings...`.

`File` ▷ `Save Current Assistant Settings...` You can save the configuration of an assistant to a file for later use. The default extension for these configuration files is `.das`.

`File` ▷ `Close Dialog` The assistant is closed, but the current configuration is preserved. This menu entry performs the same function as the assistant's close button. You can restore a closed assistant by clicking the numbered entry in the menu `Assistants` which is generated when a new assistant is opened.

`File` ▷ `Exit Assistant` The assistant is quit. The resources used by the assistant are released. The link to the generated code is lost, i.e., it is not possible to restore the assistant unless the setup has been saved to a file. The menu entry in the menu `Assistants` is also removed.

`Code Generation` ▷ `Insert Code for Selection` Insert HDevelop code based on the current settings of the assistant. The code is inserted at the IC. As long as the associated assistant is not quit, you can change the settings and update the code accordingly.

`Code Generation` ▷ `Release Generated Code Lines` The link to the generated code is cut off. The code remains in the program, but can no longer be updated or removed from the (formerly) associated assistant. Nevertheless, you can generate new code with the current settings of the assistant.

`Code Generation` ▷ `Delete Generated Code Lines` The generated code is deleted from the program. Please note that any manual changes to the generated lines are deleted as well.

`Code Generation` ▷ `Show Code Preview` Generate a preview of the code based on the current setup of the assistant. If the program already contains generated code which is linked to the current assistant, the changed code lines can be compared side-by-side in the preview.

7.1 Image Acquisition Assistant

The image acquisition assistant is an easy-to-use front-end to the various image acquisition methods supported by HALCON. Firstly, it lets you read images from the file system (selected files or whole directories). More importantly, it supports acquiring images from image acquisition devices that are supported by HALCON's image acquisition interfaces. When an image acquisition interface is selected, the corresponding device parameters, e.g., the image format can be set. After establishing a connection to the selected image acquisition interface, images can be grabbed and displayed in the active graphics window. Using live images, the parameters supported by the selected interface can be explored interactively.

When a suitable setup is achieved, the settings of the assistant can be saved for later reuse. The assistant can also be instructed to generate HDevelop code that will connect to the selected image acquisition interface, set the specified parameters and grab images.

7.1.1 Tab `Source`

Synopsis: Select from where to acquire images.

`Image File(s)`

Activate this radio button to load images from files. You can enter the names of image files in the text field. Multiple file names are separated by a semicolon ";". If an image with no path name or a relative path name is given, the image files are searched in the directories specified by the environment variables `HALCONROOT` and `HALCONIMAGES`.

You can also enter the full path of an image directory to specify all images of the given directory. If the check box `Recursive` is ticked, the images of all subdirectories are specified as well.

Pressing Return will display the first of the specified images in the active graphics window.

The buttons Select File(s) ... and Select Directory ... open a file browser to select multiple images or an image directory, respectively. After clicking OK in the file browser, the text field is updated with the selected items, and the first image is displayed in the active graphics window.

Use the entry Snap or Live in the menu Acquisition, or the corresponding tool bar buttons to view the selected images one after another.

Image Acquisition Interface

Activate this radio button to acquire images from an image acquisition interface. The drop-down list contains the list of all supported image acquisition interfaces.

Clicking Auto-detect Interfaces probes the image acquisition interfaces in turn, and removes those interfaces from the list that do not respond. It is recommended to save your program before probing the image acquisition interfaces.

7.1.2 Tab Connection

Synopsis: Setup connection parameters for the image acquisition interface selected in the tab Source (page 194).

This tab card is only available if the image source is set to an image acquisition interface. The connection parameters are described below. See the description of the operator open_framegrabber for additional information about the fields.

Configuration

Device Select the ID of a board, camera, or logical device if multiple devices are available for the selected image acquisition interface.

Upon building the list of devices, the assistant queries the status of each device. Depending on the image acquisition interface, devices may be reported as misconfigured. If you select such a device, the assistant may suggest a Generic parameter that potentially resolves the misconfiguration. If you confirm this suggestion, the parameter will be entered into the Generic slot (see below). If a device is labeled with a question mark icon, it is either read-only, busy, or unknown.

Port Specify the ID of the input port.

Camera Type Select a camera configuration or signal type.

Select... Select a camera configuration file (in XML format) from a file browser.

Trigger Tick the check box if the image acquisition is controlled by an external trigger.

Resolution (X / Y) Specify the factor for image width / height.

Color Space Specify the configuration for color acquisition.

Field Specify the frame selection for interlaced cameras.

Bit Depth Specify the number of bits used for one image channel.

Generic Some image acquisition interfaces support device-specific parameters to preset selected values before the camera is initialized. The parameters the interface claims to support are suggested as a drop-down list. To set a generic parameter, select it from the list, and edit the assigned value, i.e., the value after the =. Multiple generic parameters may be set by separating the entries with a comma.

If the selected image acquisition interface does not support generic parameters, this field is grayed out.

See the documentation of the individual image acquisition interfaces for more information about the supported generic parameters.

Action Buttons

Once the connection parameters are set up, the action buttons are used to connect to and acquire images from the specified device. Messages about connection errors are displayed in the status line of the image acquisition assistant window.

`Connect` Connect to the specified image acquisition device. If the connection fails, carefully check the configuration in the above fields. Not all combinations of settings are allowed for all devices. It is recommended that you enable low level error messages (see `General Options` ▷ `Experienced User`) to find out what is going wrong. Please note that an established connection is closed automatically, if the connection parameters are modified.

When the connection is established, this button can be used to disconnect the device.

`Snap` Acquire a single image from the device (first connecting to the device if needed). The image is displayed in the active graphics window unless `Display Image` is set to `Disabled` (see `Inspect`).

`Live` Start/stop live image acquisition mode. The images are displayed in the active graphics window unless `Display Image` is set to `Disabled` (see `Inspect`).

`Detect` Clicking this button will attempt to redetect valid parameters for the current device.

`Reset All` Reset all connection parameters to their default values.

7.1.3 Tab `Parameters`

Synopsis: Set parameters for the selected image acquisition device.

This tab card is available if the image source is set to an image acquisition interface and a connection to an image acquisition device has already been established. Press F1 for more information about the displayed parameters.

`Interface Library` The image acquisition interface library (DLL or shared object) used by the current connection is displayed in this field.

`Update Image` If this check box is ticked, a new image is acquired immediately after each parameter change. Disable the check box if you want to change multiple parameters at once.

`Refresh` Refreshes the list of supported parameters and their value ranges. This is useful for parameters with side affects.

`Reset All` Resets all parameters to their default values. Individual parameters can be reset by clicking the corresponding button displayed to the right of each parameter.

Parameter Grouping

The available parameters are grouped by user parameters, read-only parameters, action parameters and write-only parameters. The latter cannot be changed in the assistant and are listed only for reference. The parameters of some of the interfaces are additionally grouped by category and visibility. If grouping information is available, the amount of displayed parameters can be reduced by choosing a subject matter from the down-down list `Category`. You can further filter the parameters by selecting a skill level from the down-down list `Visibility` (beginner, expert, or guru).

By default the parameters are sorted thematically. You can also sort the parameters by name (check box `Sort by Name`).

Setting Parameters

The parameters are displayed in a tabular format. Hover the mouse pointer over a table row to get the short description of the corresponding parameter as a tool tip. The tool tip also includes the value range for numeric parameters (min.-max.).

The first column shows the parameter name. The second column depends on the parameter type:

- If the parameter is editable, its value can be entered into a text field. This field may contain value suggestions as a drop-down list. Numeric values can be incremented/decremented using the arrows next to the text field.
- If the parameter is read-only, its value is displayed, but cannot be modified.
- For action parameters, the corresponding action can be triggered by clicking the `Apply` button.

The third column is reserved for numeric parameters. It contains a slider to quickly alter the parameter value within the defined range. Please note that low level error messages are suppressed while dragging the slider. If the minimum value is below -10000, or the maximum value is above 10000, or no range is defined at all, no slider is displayed.

The fourth column contains a reset button for editable parameters. Click it to reset altered parameters to their default value.

7.1.4 Tab `Inspect`

Synopsis: Set display mode and show image acquisition statistics

Display

`Display Image` It is recommended to set the display mode to `Normal` unless you wish to make speed measurements. Other modes are `Volatile` (volatile grabbing), and `Disabled` (grabbing images without displaying them).

`Output Window` Specifies the graphics window for the image display (either the active graphics windows, or a window ID from the list).

`Statistics`

This area displays statistics for the acquisition time and the frame rate of all acquired images.

`Ignore first image of live acquisition` When the first image is acquired in live mode, a certain amount of overhead is added. Therefore, it is recommended to check box to increase the accuracy of the results.

`Status Bar`

`Show frames per second during live acquisition` Usually, the number of grabbed images and the acquisition time of the last image are displayed in the status bar of the window. Ticking this check box causes the frame rate (frames per second) to be displayed in live mode.

`Show Min/Mean/Max` If selected, the image acquisition statistics are also displayed in the status bar of the window. This way, you can watch the acquisition statistics in any tab card of the assistant.

7.1.5 Tab `Code Generation`

Synopsis: Preview / generate HDevelop program lines.

The settings made in the tab cards `Source`, `Connection`, and `Parameters` can be distilled to program lines that perform the desired image acquisition in your current program. The fields in this tab card specify the code generation details. You can preview the code lines in the panel `Code Preview`. This panel can be toggled between hidden and displayed state by clicking the button next to the panel label.

`Acquisition`

The settings of this section are available if images are acquired from an image acquisition interface.

`Control Flow`: The initialization code for the selected image acquisition interface is always generated (setting `Initialize Only`). It opens a connection to the specified image acquisition device, and sets all changed parameters. You can also generate code to acquire a single image (setting `Acquire Single Image`), or to acquire images in a loop (setting `Acquire Images in Loop`).

`Acquisition Mode`: You can switch between synchronous and asynchronous acquisition. The latter runs in the background and is recommended for continuous acquisition.

`Variable Names`

This section defines the variable names that are used in the generated code.

`Connection Handle`: Variable storing the acquisition handle. The image acquisition interface is accessed by this name. Set to `AcqHandle` in the example below.

`Image Object`: Variable used for the acquired images. Set to `Image` in the example below.

The following variables have to be specified if `Source` is set to `Image File(s)` and multiple files are specified:

`Loop Counter`: Variable storing the loop index.

`Image Files`: Variable for storing the image names as a tuple.

Generate the Code

`Insert Code`: The actual code is inserted at the IC.

Example Code

```
* Code generated by Image Acquisition 01
open_framegrabber ('GigEVision', 1, 1, 0, 0, 0, 0, 'progressive', 8, 'gray', \
                   -1, 'false', 'default', '003053095003_Basler_scA160014gc', \
                   0, -1, AcqHandle)
grab_image_start (AcqHandle, -1)
while (true)
    grab_image_async (Image, AcqHandle, -1)
*     Do something
endwhile
close_framegrabber (AcqHandle)
```

7.1.6 Menu Bar

Menus `File`, `Code Generation`, `Help`

For the description of the corresponding menu entries see section 7 on page 193.

Menu `Acquisition`

`Connect` Connect / disconnect the selected image acquisition device. See section 7.1.2 on page 195.

`Snap` Acquire a single image. See section 7.1.2 on page 195.

`Live` Acquire images in live mode. See section 7.1.2 on page 195.

7.2 Calibration Assistant

7.2.1 Introducing the Calibration Assistant of HDevelop

Most applications that need a previous calibration of the camera system belong to the area of 3D machine vision. These applications require a 3D model of the camera system. Calibration is necessary to gain information about distortions (perspective and lens distortions) in an image and about parameters of the camera system. Calibrating your camera system with the HALCON Calibration Assistant enables you to measure in the world coordinate system with a high accuracy. This task can be performed by taking images of a known object, a calibration plate.

The Calibration Assistant of HDevelop is a front-end to HALCON's operator `camera_calibration`. Using the Calibration Assistant you can

- either perform a complete calibration or
- take advantage of the user-defined mode and only calibrate chosen parameters, if the rest is already known (e.g., if you are using a special setting).

All you need is a set of suitable calibration images (the number and requirements depend on the used calibration plate, please see the reference manual chapter "Calibration"). The Calibration Assistant then returns the calibration results and enables you to generate code and insert it into a given program.

The Calibration Assistant can calibrate vision systems based on standard lenses as well as on *telecentric lenses*.

With the HALCON Calibration Assistant you can

- perform a calibration,
- view the calibration results (page 210),
- generate code (page 211) for the calibration or for using the calibration results and insert it into a program for further use in a subsequent application.

A reference to the elements of the Calibration Assistant can be found in the Calibration Assistant Reference (page 214).

For further information about camera calibration, please refer to the reference manual chapter "Calibration" or the corresponding chapter in the solution guide on 3D Vision.

ATTENTION: Keep your camera setup (aperture, focus, pose) fixed once you have chosen it! This applies to the calibration process itself as well as to the subsequent application. Any changes will result in the failure of the calibration or, even worse, in wrong output values.

In this guide, the following special terms are used:

Calibration By *calibrating* (page 203) a vision system, you extract information about it, e.g., its focal length or its position and orientation relative to the "world". However, even with such information you cannot fully reconstruct the 3D world from a single image. For example, you can determine the (3D) size of an object only if you know its distance from the vision system (when using a standard lens). Calibration is a preparation for all subsequent image processing applications. The Calibration Assistant needs to grab a set of images of a special calibration object placed in front of your vision system. You can choose between a *Full Calibration* and a *User-Defined Calibration*, where known parameters are not calibrated again.

Calibration Plate This is an object whose shape is known precisely. Two different types of standard HALCON calibration plates are available: Calibration plates with hexagonally arranged marks and calibration plates with rectangularly arranged marks. Transparent calibration plates are available for applications requiring backlight illumination. Additionally, the calibration plates are available in different sizes. Which calibration plate is suited best depends on your machine vision task: As a rule of thumb, if you grab an image of the plane of measurement, calibration plates with hexagonally arranged marks should fill the whole image and calibration plates with rectangularly arranged marks should fill a fourth of the image. The bigger calibration plates (160mm and 320 mm for calibration plates with hexagonally arranged marks and 100mm and 200mm for calibration plates with rectangularly arranged marks, made from aluminum) come together with a file containing their exact measurements (`calplate_160mm.cpd`, `calplate_320mm.cpd`, `caltab_100mm.descr`, and `caltab_200mm.descr`). Please copy this file to the subdirectory `calib` of the HALCON base directory

you chose during the installation. This is not necessary for smaller (ceramics) calibration plates as they can be manufactured very precisely and can therefore use standard description files (.cpd files for calibration plates with hexagonally arranged marks and .descr files for calibration plates with rectangularly arranged marks). If you use your own calibration plate, you have to create the description file yourself and copy it into the subdirectory `calib`.

Calibration Plate Extraction Parameters These parameters (page 209) influence the extraction of the calibration plate. You may change them to improve the extraction of the plate if necessary. We recommend, however, that you try to improve your image quality first.

Camera Parameters Internal `Camera Parameters` describe the camera itself, e.g., its `Focal Length`, `Cell Width` and `Cell Height`. These parameters are part of the calibration results, initial values for some of them are also needed for the setup of the calibration.

Camera Pose The position and orientation of the world coordinate system relative to the camera are called the external `Camera Parameters`. They are part of the calibration results.

Display Parameters On the `Calibration` (page 203) tab, you can choose the display parameters, like colors, as you prefer them. See also `Display Parameters` (page 209).

Full Calibration In a `Full Calibration`, the complete camera system is calibrated. The only information needed are approximate values for `Camera Type`, `Cell Width`, `Cell Height` and `Focal Length` as well as the question whether you are using a `Telecentric` camera (in which case the `Focal Length` is not required).

Image Rectification Based on the calibration results, you can remove image distortions. This is called image rectification. Example code is available from the `Code Generation` tab (page 211).

Pose Estimation Once the interior parameters are calibrated, it is possible to estimate the camera pose from a single image. Example code is available from the `Code Generation` tab (page 211).

Reference Image This image locates the world coordinate system, which then has its origin at the origin of the calibration plate in the reference image. The origin of the calibration plate is the center of the central mark of the first finder pattern for calibration plates with hexagonally arranged marks and the middle of the calibration plate for calibration plates with rectangularly arranged marks. By default, the first calibration image is used as the reference image. However, you can choose any other image of the calibration sequence.

Standard Lenses A standard lens is similar to the one in the human eye: It performs a *perspective projection*; hence, objects become smaller in the image the further they are away.

Telecentric Lenses Telecentric lenses perform a *parallel projection*. Therefore, objects have the same size in the image independent of their distance to a camera. This means that they can lie in different planes; only the orientation of the planes relative to the camera must be identical.

User-Defined Calibration The setup step `Calibration Task` provides a `User-Defined Calibration`, which enables you to perform calibrations with special setups or re-use parameters from previous calibrations.

World Coordinates Measurements and XLD contours can, after finishing the calibration, be transformed into (3D) world coordinates, meaning the coordinates of the world (e.g., in millimeters), as opposed to those of an image (in pixels). Example code is available from the `Code Generation` tab under `Sample Usage` (page 212).

Quality Issues A high quality of the calibration images is essential not only for the calibration itself but for the quality of the calibration results. Examples for bad image quality are overexposure of the *calibration plate*, bad mark contrast or very small mark size. These quality issues are listed under `Quality Issues` (page 206) on the `Calibration` tab. Sorting out images with too many defects improves the calibration results.

7.2.2 How to Calibrate with the Calibration Assistant

By using the Calibration Assistant, you can set up and optimize your calibration application in three steps:

- Choose the right calibration mode,
- load the calibration images (page 203),
- and respond to image quality feedback (page 206).

7.2.2.1 Choosing the correct Calibration Mode and Basic Parameters

For the calibration setup in the `Setup` tab, the basic information has to be filled in. Which information is necessary for your application will depend on the answer to the question whether you want to perform a *full calibration*, whether you have a special setup or you have calibrated before and therefore want to take advantage of the *user-defined calibration*. Furthermore, information about the *calibration plate* and the camera is required.

In short, the setup information includes

- the `Calibration Task`,
- the `Calibration Plate`,
- and the known `Camera Parameters`.

□ Choosing the task for your application

If you want to calibrate all parameters, e.g. if you are calibrating for the first time with your setup, click the radio button `Full Calibration: Pose and all Camera Parameters`.

If you are using a special setting or you have already calibrated your system before and want to re-use your resulting parameters, choose `User-defined: Select Individual Parameters for Calibration`.

After having decided on your calibration task, proceed with the details about your `Calibration Plate`.

□ Calibration Plate Parameters

First, choose the description file for your calibration plate and the calibration plate `Thickness` (in mm). The name of the description file indicates the size of the plate and the file extension indicates the type of the calibration plate. In particular, the file ending .cpd is used for calibration plates with hexagonally arranged marks and the file extension .descr is used for calibration plates with rectangularly arranged marks. Note that calibration plates with hexagonally arranged marks usually have light marks on a dark background. Nevertheless, also calibration plates with dark marks on a light background are available. Then, you have to choose a description file with the filename ending "_dark_on_light". Calibration plates with rectangularly arranged marks always have dark marks on a light background. With the parameter `Thickness`, you can modify the position of the world coordinate system and the measurement plane, which is located beneath the calibration plate surface by the value of `Thickness` in the `Reference Image` (page 205).

Then proceed to set the `Camera Parameters`.

□ Set Camera Parameters

For setting up the camera parameters

- first choose the `Camera Model`,
- then set the parameters for a full calibration or
- set the parameters for a user-defined calibration (page 203).

It is also possible to import parameters from a file. If you should decide to do this, just click the `Import Parameters` button.

Once you have finished this last part of the `Setup` tab, proceed to the `Calibration` (page 203) tab.

◇ Choose your Camera Model

First, choose the `Camera Model` you are using:

- either `Area Scan (Division)`,
- `Area Scan (Polynomial)`
- or `Line Scan`

Even though the camera model `Area Scan (Division)` typically returns good results for your application, you can improve the accuracy and lower the error rate by using the `Area Scan (Polynomial)` camera model. We therefore recommend for you to use the `Area Scan (Polynomial)` model if the `Mean Error` on the `Results` (page 210) tab under `Calibration Status` (page 210) is too high. If you decide for the `Area Scan (Polynomial)` model, it is especially important that you thoroughly cover the field of view with calibration plate images and do not leave out the edges.

Note, that a higher `Mean Error` might be caused by inappropriate calibration images.

For area scan cameras, the following additional options may be specified:

If your camera has a tilt lens, tick the checkbox `Tilt`.

Then, the `Projection Model` must be specified (either `Projective` or `Telecentric`).

Telecentric tilt lens cameras are available in different variations, which must be specified in `Projection Model`:

- `Bilateral Telecentric`
- `Object Side Telecentric`
- `Image Side Telecentric`

◇ Set Parameters for Full Calibration

If you choose a full calibration for an area scan camera, you must specify approximate values for the camera parameters. These parameters depend on the camera type and are listed in the following tables. Please have a look at the data sheet of your camera for suitable values. Information about the focal length can be found on the lens itself.

Parameters for projective area scan cameras:

Parameter	no tilt	with tilt
Cell width	X	X
Cell height	X	X
Focal length	X	X
Image plane distance		X
Tilt angle		X
Rotation angle		X

Parameters for telecentric area scan cameras:

Parameter	no tilt	bilateral	object side	image side
Cell width	X	X	X	X
Cell height	X	X	X	X
Magnification	X	X	X	
Focal length				X
Image plane distance			X	
Tilt angle		X	X	X
Rotation angle		X	X	X

Parameters for line scan cameras:

Parameter	
Cell width	X
Cell height	X
Focal length	X
Motion X	X
Motion Y	X
Motion Z	X

◇ Set Parameters for User-Defined Calibration

In the user-defined mode for the `area scan camera (division)`, you can also choose `Center Column (Cx)`, `Center Row (Cy)` and `Kappa`. The `area scan camera (polynomial)` model allows you to also choose the lens distortion parameters for radial distortion `Radial 2nd Order (K1)`, `Radial 4th Order (K2)`, `Radial 6th Order (K3)` and the two parameters for tangential distortion `Tangential 2nd Order (P1)` and `Tangential 2nd Order (P2)`.

If you want to change the parameters `Cell Width` and `Cell Hight` independently from each other, click the chain button.

7.2.2.2 Acquiring Calibration Images

The main part of the calibration process consists of acquiring images of the calibration plate in different positions and orientations relative to the vision system. Please note that the more you vary the position and orientation, the better the calibration results will be. Therefore, place the plate so that it appears in different corners (for calibration plates with rectangularly arranged marks; calibration plates with hexagonally arranged marks should cover the whole image with a single image), at different distances to the camera, and in different planes, i.e., tilt it for some images. Note that for calibration plates with rectangularly arranged marks it is necessary to not only place the calibration plate in the center of the field of view, but also move it to the corners and margins. Good calibration images will improve your calibration results significantly. Detailed instructions on how to take calibration images can be found in the section “How to take a set of suitable images?” in the reference manual chapter “Calibration”.

Obligatory steps for calibration are

- acquiring calibration plate images,
- choosing your image source,
- and calibrating.

Optional parameters, which may be changed, are

- parameters concerning `Quality Issues` (page 206),
- `Display Parameters` (page 209), and
- `Calibration Plate Extraction Parameters` (page 209)

ATTENTION: Keep your camera setup (aperture, focus, pose) fixed once you have chosen it! This applies to the calibration process itself as well as to the subsequent application. Any changes will result in the failure of the calibration or, even worse, in wrong output values.

Once the calibration images are available, you can push the `Calibrate` button and move on to the `Results` (page 210) tab.

□ Acquiring Images for a Successful Calibration

Note that the calibration assistant currently supports 8-bit and 16-bit (“byte” and “uint2”) images.

In order to achieve an accurate calibration result you should pay special attention to the acquisition of your calibration images. Therefore, see the section “How to take a set of suitable images?” in the reference manual chapter “Calibration”, which gives guidance on the acquisition process.

See figure 7.1 for an exemplary setup with the resulting calibration images shown in figure 7.2.

□ Choosing an Image Source

The images for the calibration can either be loaded from a file or acquired directly using the Image Acquisition Assistant.

When loading images from a file, just click the radio button `Image Files`.

To acquire new images, click the radio button `Image Acquisition Assistant` (page 194). The assistant will then appear in a new window and support you with acquiring new calibration images.

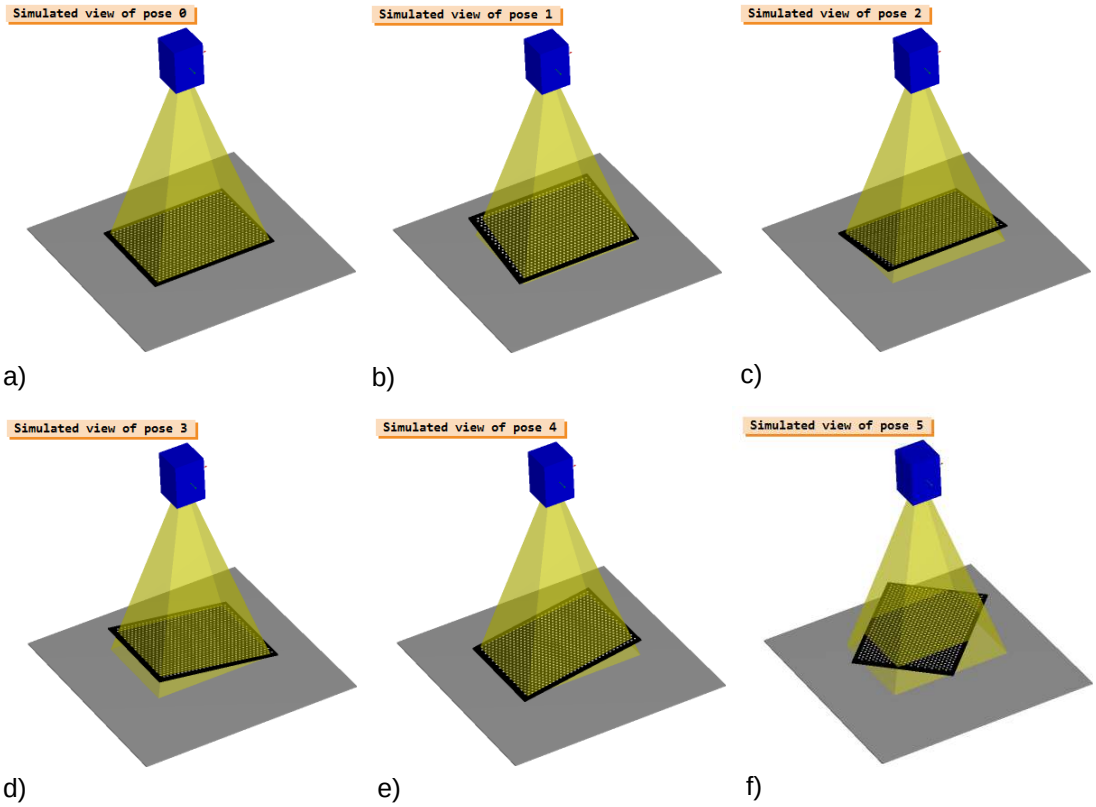


Figure 7.1: Acquisition of suitable calibration images using a calibration plate with hexagonally arranged marks. The plate is a) placed in the measurement plane, b) - e) tilted in different directions, and c) placed parallel to the measurement plane (with a rotation around the z axis).

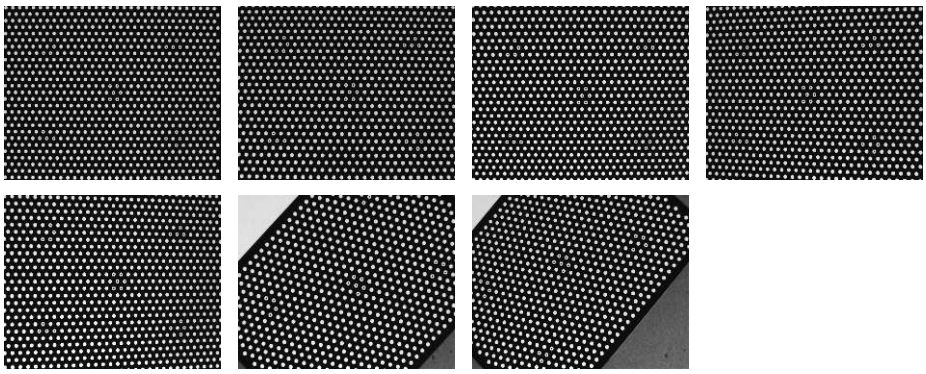


Figure 7.2: Example for suitable calibration images using a calibration plate with hexagonally arranged marks.

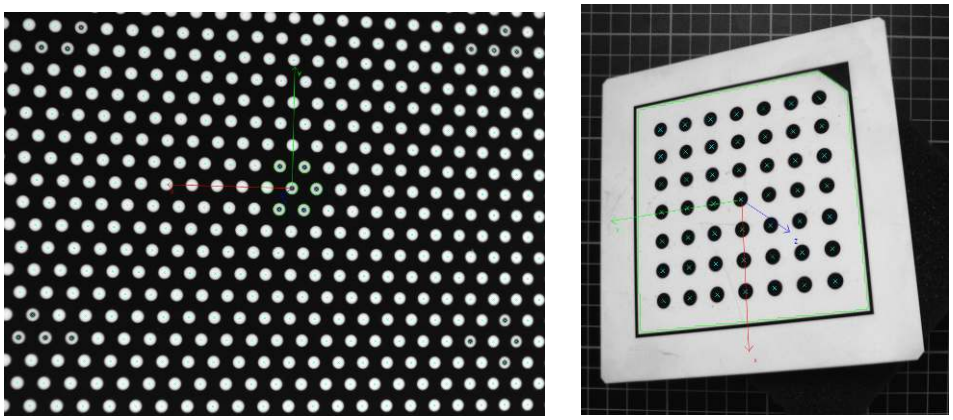
Note that the calibration works on a single channel. For color RGB images, the red channel will be used. A color transformation can be performed with the operator `trans_from_rgb`.

□ Calibration

The three basic steps of each calibration are

- acquiring calibration images,
- selecting a reference image, and
- calibrating.

◇ Calibration Images



a) b)

Figure 7.3: Images of calibration plates with their coordinate systems: a) plate with hexagonally arranged marks and b) plate with rectangularly arranged marks.

All images from files will be displayed with their path on the `Calibration` tab, whereas images acquired using the Acquisition Assistant will be displayed with their consecutive numbers. Furthermore, the image status gives information about the quality of each image. Details concerning quality can be found under `Quality Issues` (page 206). If you use the Image Acquisition Assistant and want to see a live image, you can also activate `Live Image` on the `Calibration` tab and click the `Snap` button whenever you want to keep an image for calibration. If you `Load...` images from a file into the Calibration Assistant and then decide to acquire new images with the Image Acquisition Assistant, you will be warned that the images from the file will be removed from the window. With the `Remove` and `Remove All` buttons on the left, you can remove either one or all images of the list. The `Save` and `Save All` buttons will save one or all images of the list. Click `Update` to control the time when camera parameters, segmentation parameters or quality adjustments shall be transferred for the existing images. Activate `Auto Update` to automatically update to the latest adjustments. `Quality Issues` are updated with a little delay after adapting `Calibration Plate Extraction Parameters` (page 209). Deactivating `Auto Update` enables you to change several parameters at once and speeds up the processing of bigger data sets.

◇ Select a Reference Image

With the pose of the calibration plate in the reference image, you specify the world coordinate system and the measurement plane for subsequent 3D measurements (see figure 7.3). Thus, in one calibration image (typically, the first one), you should place the calibration plate such that it lies on top of the measurement plane. If this is not possible, place the calibration plate in a position parallel to the measurement plane and “move” the coordinate system by adapting the parameter `Thickness`. The star on the left side of the `Calibration` window indicates the reference image. It is by default set on the first image. You can, however, by clicking the `Set Reference` button, pick another image as reference.

◇ Calibrating

Click the button `Calibrate` to finally perform the calibration task. You should, however, check first whether you have enough images of sufficient quality. You can check the quality under `Quality Issues`. If necessary, you can also change `Calibration Plate Extraction Parameters` (page 209) before actually calibrating. In case your calibration fails and displays the error `"Camera calibration did not converge"`, check possible error sources in the Checklist for Calibration Errors or have a look at the recommendations listed in the reference manual chapter “Calibration”, section “How to take a set of suitable images?”.

Possible Error Source	Solution
Did any camera settings (like aperture, focus or pose of the camera) change during the calibration process?	Take new calibration images and do not change any settings during calibration and later during the application. If you decide to change anything you have to start a new calibration.
Did you acquire the calibration images the way they are required?	Make sure you followed the recommendations in the section "How to take a set of suitable images?" in the reference manual chapter "Calibration" regarding the placement of the calibration plate, the setup of the cameras and the image properties.
Are you using an extreme wide angle lens?	The distortions that appear close to the image borders cause a higher `Mean Error` or can even be responsible for the failure of the calibration. You must use another lens in this case.
Is the size of your camera chip compatible with the lens?	Using a lens that is not compatible with your camera chip size (this information should be included in the instructions of the lens) will decrease the quality of your image.

Table 7.1: Checklist for Calibration Errors.

□ Handling Quality Issues

Under `Quality Issues` you find an evaluation of each image, which includes descriptions of the defective image features and a quality score percentage that tells you how severe the problem is. A result of 0% indicates a very defective image feature whereas 100% equals ideal quality. This can help you to improve your calibration result by deleting images which are not good enough and might lead to a higher error rate during the calibration process. If you need a certain quality level you can set a `Warn Level` and the defects will be listed under `Quality Issues`. The quality issues are detected by image tests and sequence tests. If you want the program to run faster or if you do not need quality feedback, you can change `Image Tests` and `Sequence Tests` either to `Quick`, which performs less tests, or `None`, which does not perform any tests at all. If the defects are too severe e.g., if the calibration marks or even the calibration plate are not found, the `Calibration` button will be grayed out, making it impossible to calibrate unless all images of such poor quality are deleted from the list.

The test results referring to the calibration plate's tilting may be ignored if later measurements are always conducted in exactly the same plane. In this case, however, the values for the `Focal Length` and `Z` are not correct each for itself but only in their combination. The reason for this is that neither of these values can be determined for itself which leads to the result that if you get, for example, a `Focal Length` that is double the value that it should be, `Z` will be half as high and vice versa. Besides, the further you place an object above the plane in which you have performed the calibration, the less precise the result will be.

Note that poor image quality leads to poor calibration results and subsequently causes bad or wrong measuring values. However, acceptable results are usually achieved even with quality score warnings in the range of 40% to 70%. If necessary check the following tables for suggestions about improving your image quality. When trying to improve your image quality, do not forget to check other error sources.

ATTENTION: If you change your camera setup (aperture, focus, pose) during the calibration process or during the subsequent application, you have to restart your calibration with the new setup. Any changes will result in the failure of the calibration or, even worse, in wrong output values.

Note: Due to special settings or unchangeable specifications of your work environment, it may be possible that you cannot fully avoid any quality reductions. If you follow these instructions, you should, however, be able to reach a feasible quality level to work with.

Quality Issue	Explanation	Possible Solution
Plate is overexposed	Parts of the calibration plate are too bright, which leads to a shifting of edges and therefore calculates a wrong center position.	Close the lens aperture or the shutter a bit more or turn down the brightness of your illumination until an adequate quality is reached.
Illumination is inhomogeneous	The image is illuminated inhomogeneously, i.e. the brightness of the calibration plate changes within one image. This condition makes it difficult to locate the calibration plate and consequently leads to a lower accuracy.	Inhomogeneity in an image is often the result of using lateral illumination. If that is the case: Can you change the setting and instead use illumination from above? Another possibility would be to use diffuse illumination.
Contrast is low	The difference between the gray values of the calibration plate and the calibration marks is not big enough.	Reasons can be either overexposure or underexposure. To improve your results, change your aperture or the brightness of your illumination.

Quality Issue	Explanation	Possible Solution
Diameter of marks are too small	The diameters of the found calibration plate marks are too small.	To fix this issue, you should either change your setup or use a calibration plate with larger marks.
Marks on plate are out of focus	The marks are not completely focused, some of them appear blurry. This leads to a lower robustness.	The depth of field has to include the whole object. To fix this error, change either your focal length or the distance of the object to the camera. Alternatively you can also make the aperture smaller and use brighter illumination.
Quality assessment failed	The image test failed, even though the plate could be found in the image.	For calibration plates with rectangularly arranged marks, check, if any part of the image is occluded and if the occlusion interrupts the black margin of the calibration plate.
Mark extraction failed for some images	It was impossible to extract the calibration plate marks in some images, which makes it also impossible to calibrate in this state.	Delete the images for which mark extraction has failed and use new images instead or adapt the extraction parameters. Make sure that you follow the recommendations in the section “How to take a set of suitable images?” in the reference manual chapter “Calibration”.

Quality Issue	Explanation	Possible Solution
Quality issues detected for some images	The quality of some images is below the warn level.	Check the quality issues of the single images by clicking on their names in the list. Handle quality issues as described in the table above.
Number of images is too low	The number of images is lower than recommended.	Check if you have taken enough images, depending on the type of calibration plate you use.
Field of view is not covered by plate images	Some part of the field of view is not covered by any image of the calibration plate, i.e. there are areas with no marks.	Press the `Show` button, which appears in a column named `Details`, to see all areas in red that are not covered by calibration plate images (compare figure 7.4). Before continuing, add the missing image(s) to your sequence. You can avoid this problem by following the recommendations in the section "How to take a set of suitable images?" in the reference manual chapter "Calibration".
Tilt angles are not covered by sequence	The calibration plate has not been tilted enough.	Add more images of your calibration plate tilted in different directions. For calibration plates with rectangularly arranged marks we recommend tilting the plate in every quadrant of the image twice and vary the tilting direction.
Not all image sizes are identical	The image list contains images of different sizes.	You have changed your setup while taking calibration images. Therefore, you should delete those images taken before the change to get useful results back.

7.2.2.3 Display Parameters

The drop-down menus under `Display Parameters` enable you to choose the colors and drawing parameters for the calibration images display that you prefer. You can either leave the default values or choose your own values for `Plate Region`, `Mark Centers` or the `Coordinate System`. The `Draw` option lets you choose whether you want to see margins or filled regions.

7.2.2.4 Calibration Plate Extraction Parameters

You should always aim for high quality images. If for some reason you should, however, have trouble with your image quality and see no other option of improving it, you can still adapt some parameters under `Calibration Plate Extraction Parameters`.

The following parameters may be changed to improve the calibration results:

For calibration plates with hexagonally arranged marks:

- `Smoothing (Sigma)` should be set to a higher value if the image is blurry or contains strong noise.

For calibration plates with rectangularly arranged marks:

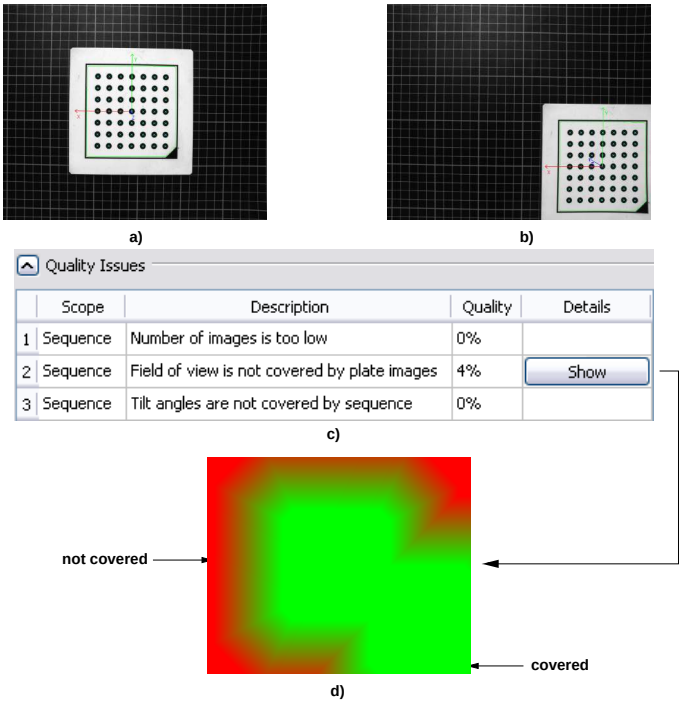


Figure 7.4: Not enough calibration images have been taken. a) and b): calibration sequence consisting of two calibration plate images c) A `Show` button appears due to the fact that the coverage is not sufficient. d) An image shows which parts of the field of view are not covered by calibration plate images.

- `Gap Tolerance` should be set to a higher value if the plate is strongly tilted and
- `Smoothing (Alpha)` should be set to a smaller value if the image is blurry.
- Furthermore the `Maximum Mark Diameter` may be changed if the checkbox is activated.

For more information about these parameters, please refer to the reference manual entry of the HALCON operator `find_calib_object`.

7.2.3 Results of the Calibration

Two types of parameters of your vision system are calculated as results: *internal* parameters, e.g., the precise focal length, the size of the camera chip, or the distortion caused by an imperfect lens, and *external* parameters, i.e., the position and orientation of the vision system.

Consequently, the calibration returns the following results:

- `Calibration Status`,
- `Camera Parameters`, and
- the `Camera Pose`.

The results displayed in `Camera Parameters` and `Camera Pose` can also be saved to a file by clicking the `Save` buttons on the right.

`Display Results` enables you to choose which results should be displayed.

Once you are finished with the results, go on to the `Code Generation` (page 215) tab.

7.2.3.1 `Calibration Status`

This box displays the `Status` of the calibration, i.e. whether the calibration was successful, and the `Mean Error` in pixels.

If you either delete calibration images (page 204) or change Calibration Plate Extraction Parameters (page 209) or Camera Parameters (page 201) after having calibrated, the former calibration data is not valid any more. Therefore, the Status will display that no calibration data is available. To continue working with your changed camera parameters, calibration parameters or images, just click Calibrate (page 205) again on the Calibration tab.

□ Mean Error

Mean Error designates the average error in pixels during the calibration process. Once the system has been calibrated, the ideal centers of the calibration marks are calculated and compared to the real mark centers. Mean Error is the deviation value between the ideal and the real mark centers. A value of 0.1 and lower can be regarded as a good result. Possible calibration errors are described in the tables about quality issues under Quality Issues (page 206); most of them can usually be solved quite easily, often just by taking better calibration images (page 203).

7.2.3.2 Camera Parameters

The internal camera parameters include Cell Width (Sx) and Cell Height (Sy) in micrometer, Focal Length in mm, Center Column (Cx) and Center Row (Cy), Image Width and Image Height in pixels. They also include Kappa in 1/ m^2 or instead of Kappa, the distortion parameters Radial 2nd Order (K1) in 1/ m^2 , Radial 4th Order (K2) in 1/ m^4 , Radial 6th Order in m^6 , Tangential 2nd Order (P1) and Tangential 2nd Order (P2) in 1/ m^2 for the polynomial area scan camera model.

If you have a line scan camera, additionally to the values of the area scan camera (division)model, values for the motion parameters Motion x (Vx), Motion y (Vy) and Motion z (Vz) in micrometer/pixel will be returned.

7.2.3.3 Camera Pose

The 3D pose of the world coordinate system relative to the camera is described by the external camera parameters X, Y and Z in mm and Rotation X, Rotation Y and Rotation Z in degrees.

7.2.3.4 Display Results

Via radio buttons you can choose Original Reference Image, to see the previously chosen reference image and Simulated Reference Image to display a synthetic reference image, which has been calculated using the internally known measures of the calibration plate and the pose of the plate in the reference image. You can also decide whether or not you want to Display Coordinate Axes of the coordinate system of the calibration plate.

7.2.4 Generating Code

This tab helps you to generate and insert code for calibrating and for using the calibration results in your HDevelop program. The tab is subdivided into four parts:

- Calibration
- Sample Usage
- Variable Names (page 213)
- Code Preview (page 213)

Once you are finished with configuring the options, *check the position of the insert cursor* and click Insert Code (page 215) under Calibration or Sample Usage to insert the code into your HDevelop program. Note that if you have already inserted code into your program and you click insert code again, the previous code will be replaced regardless of the cursor position.

7.2.4.1 Calibration

Choose your Generation Mode, either

- Calibration Procedure which exports the generated code,
- Calibration Data (Tuple) which exports the resulting calibration parameters (CameraParameters and CameraPose) as tuples,
- or Calibration Data (File) which writes the calibration results into the specified files and generates code lines for reading those files.

For the last one you can click the directory icons to browse for a stored file. Subsequently select Parameter File and Pose File.

In order to save the calibration results to files it is necessary that

- a successful calibration took place before and
- a file name exists for both files.

To generate code for initializing the image acquisition when using the Image Acquisition Assistant (page 194), enable Initialize Acquisition.

Once you are finished, *check the position of the insert cursor* and click Insert Code (page 215) to insert the code into your HDevelop program.

7.2.4.2 The Browse button

The Browse button on the Code Generation tab is similar to the Save button on the Results tab. It can be used to create file names into which the calibration results can then be written when choosing the option Calibration Results (File).

7.2.4.3 Sample Usage

Sample Usage shows you what is possible with your calibration data and provides code, which you can adapt to your own purposes. Choose the action you are interested in and the example code will be inserted into your program.

You have the choice between:

- Transform Measurements into World Coordinates,
- Transform XLD Contours into World Coordinates,
- Estimate Pose from Single Image and
- Rectify Image .

Once you are finished, *check the position of the insert cursor* and click Insert Code (page 215) to insert the code into your HDevelop program.

□ Transform Measurements into World Coordinates

In the example code, the image coordinates of the first two mark center points are transformed into world coordinates and this (3D) distance is calculated. First, the image coordinates of some points of interest lying in the reference plane are obtained. Here, simply the first two mark center points of the plate are chosen and a line is drawn between the two points for visualization. Then image coordinates are converted into world coordinates using HALCON operator image_points_to_world_plane. The Z coordinates will be 0 by definition because the measurement plane is the plane with the world coordinate Z=0 (on reference plane). The distance in world coordinates is determined using distance_pp.

To adapt this code to your application, you typically exchange the mark centers for “real” points of interest.

□ Transform XLD Contours into World Coordinates

In the example code, the XLD contours are transformed into world coordinates and this (3D) distance is calculated. The points are visualized by a line. First an XLD in image coordinates, which relates to some interesting features in the image, is obtained. Here, we simply generate a contour connecting the mark center points of the plate by using the HALCON operator `gen_contour_polygon_xld`. Then a conversion into world coordinates is performed with HALCON operator `contour_to_world_plane_xld`. Using the operator `get_contour_xld`, mark center points are extracted in world coordinates.

To adapt this code to your application, you typically exchange the mark centers for “real” points of interest and adapt or remove the visualization.

For further information about pose estimation, please refer to the section "Pose Estimation of known 3D Objects With a Single Camera" in the Solution Guide III-C.

□ Estimate Pose from Single Image

First, the position of mark centers on the calibration plate is determined. With known camera parameters, one image is enough to determine the new pose using the HALCON operator `camera_calibration`.

This sample code always determines the pose of the calibration plate. There is no further adaption possible.

□ Rectify Image

First the desired width of the visible area in world coordinates in mm is chosen and converted to m. Then `set_origin_pose` adjusts the origin so the plate is roughly centered. The HALCON operator `gen_image_to_world_plane_map` generates the rectification map. Finally, images can be rectified using the rectification map by `map_image`.

To adapt this code to your application, you typically change the scale and origin of the new image coordinate system.

7.2.4.4 Variable Names

For each calibration, default variable names are chosen. You can, however, use your own variable names and change variable names for:

- `Connection Handle`
- `Image Object`
- `Camera Parameters`
- `Start Parameters`
- `Loop Counter`
- `Image Files`
- `Camera Pose`
- `Window`

Note: These are variables which you might set before the generated code or use after the generated code. Intermediate variables have fixed names starting with `TmpCtrl` or `TmpObj`.

Once you are finished, *check the position of the insert cursor* and click `Insert Code` (page 215) to insert the code into your HDevelop program.

7.2.4.5 Code Preview

Here, you can, e.g., edit or replace individual operators of the code lines proposed by the Calibration Assistant.

For details, see also `Code Generation` (page 215) in the menu.

7.2.5 Calibration Assistant Reference

The Calibration Assistant consists of the following elements.

Pull-down menus:

- File
- Calibration
- Code Generation
- Help

Tool bar with a selection of important buttons:

- Load Assistant Settings
- Save Current Assistant Settings
- Insert Code
- Calibrate (page 203)
- Help

Tabs with the dialogs for most of the tasks that can be done with the Calibration Assistant:

- Setup (page 216)
- Calibration (page 216)
- Results (page 216)
- Code Generation (page 216)

Furthermore, it provides a status bar at the bottom in which messages are displayed. The status bar also displays the *calibration results* (page 210), i.e., if the calibration was successful. Please note that the status bar does not provide a scrolling mechanism; if the displayed message is too long, move the mouse over it, so that a tool tip displaying the full message pops up. Alternatively, if the message is only slightly larger than the status bar, you can also drag the left or right border of the Calibration Assistant window to enlarge it.

Images are displayed in the graphics window of HDevelop.

7.2.5.1 The Menu File

Via the menu File you can

- load formerly used and saved settings of the Calibration Assistant,
- save the current settings of the Calibration Assistant for later use,
- close the Calibration Assistant dialog (while retaining the current settings as long as the HDevelop session is active), and
- exit the Calibration Assistant dialog (discarding the settings).

☐ Loading Assistant Settings

If you have saved the settings of a former Calibration Assistant session, you can load them again by the menu item File ▷ Load Assistant Settings or via the corresponding button of the tool bar.

☐ Save Current Assistant Settings

You can save the current settings of a Calibration Assistant session using the menu item File ▷ Save Current Assistant Settings or the corresponding button in the tool bar. Then, you can load them again in a later session.

□ Close the Calibration Assistant Dialog

When closing the Calibration Assistant dialog with the menu item `File` ▷ `Close Dialog`, the current settings are stored for the duration of the current HDevelop session. That is, as long as you do not exit HDevelop, you can again open the Calibration Assistant with the same settings. In contrast to this, when you exit the Calibration Assistant, the settings are lost also for the current HDevelop session.

□ Exit the Calibration Assistant

When you exit the Calibration Assistant with the menu item `File` ▷ `Exit Assistant`, the assistant's dialog is closed and the current settings are lost unless you have stored them via the menu item `File` ▷ `Save Current Assistant Settings` (page 226). If you want to close the dialog but keep its settings for the current HDevelop session, you should use the menu item `Close Dialog` instead.

7.2.5.2 The Menu `Calibration`

Via the menu `Calibrate` you can run a calibration as described in the section Calibrating (page 205).

7.2.5.3 The Menu `Code Generation`

Via the menu `Code Generation` you can

- insert code to the program window of HDevelop according to the current settings of the Calibration Assistant,
- release the generated code lines in the program window,
- delete the generated code lines from the program window as long as you did not release them, and
- open the dialog for the code preview inside the tab `Code Generation`.

□ Insert the Generated Code Lines

Via the menu item `Code Generation` ▷ `Insert Code` (also accessible as tool bar button or as button inside the tab `Code Generation`), you can insert the code that is generated according to the current settings of the Calibration Assistant into the program window. Inserting code via menu or tool bar will generate code for calibration and samples.

□ Release the Generated Code Lines

Via the menu item `Code Generation` ▷ `Release Generated Code Lines` you can release the generated and inserted code lines. After releasing the code lines, all connections between the Calibration Assistant and the program window of HDevelop are lost. That is, changes, e.g., the deletion of code lines, can then only be applied directly in the program window and not from within the Calibration Assistant anymore.

□ Delete the Generated Code Lines

Via the menu item `Code Generation` ▷ `Delete Generated Code Lines` you can delete the code lines that you have previously generated and inserted into the program window of HDevelop from within the Calibration Assistant. Note that this works only as long as you have not yet released the code lines.

□ Preview of the Generated Code Lines

Via the menu item `Code Generation` ▷ `Show Code Preview` you can open the dialog for the `Code Preview` in the tab `Code Generation`.

7.2.5.4 The Menu `Help`

Via the menu `Help` you can access the online documentation.

7.2.5.5 The Tab `Setup`

The `Setup` tab consists of the following subdivisions:

- `Calibration Task` (page 201)
- `Calibration Plate` (page 201)
- `Camera Parameters` (page 201)

7.2.5.6 The Tab `Calibration`

The `Calibration` tab includes:

- `Image Source` (page 203)
- `Calibration` (page 204)
- `Quality Issues` (page 206)
- `Display Parameters` (page 209)
- `Calibration Plate Extraction Parameters` (page 209)

7.2.5.7 The Tab `Results`

The `Results` tab includes the following subdivisions:

- `Calibration Status` (page 210)
- `Camera Parameters` (page 211)
- `Camera Pose` (page 211)
- `Display Results` (page 211)

7.2.5.8 The Tab `Code Generation`

The `Code Generation` tab includes the following subdivisions:

- `Calibration` (page 212)
- `Sample Usage` (page 212)
- `Variable Names` (page 213)
- `Code Preview` (page 213)

7.3 Matching Assistant

7.3.1 Introducing the Matching Assistant of HDevelop

The Matching Assistant of HDevelop is a front-end to HALCON's powerful matching methods:

- shape-based matching,
- correlation-based matching,
- descriptor-based matching, and
- deformable matching.

For basic information on these matching methods and how to choose a suitable method for your application, please refer to the section Select a Matching Method or for more detailed information refer to the Solution Guide on Matching.

These matching methods that are available within the Matching Assistant let you locate objects with subpixel accuracy at a high speed, even when they appear rotated, partly occluded, or under changing illumination. Using the Matching Assistant you can

- configure (page 229) and test the matching process (page 219) with a few mouse clicks and
- optimize the parameters interactively to get the maximum matching speed (page 246) and recognition rate.

All you need is a single model image and a set of test images. The Matching Assistant further assists you by automatically calculating suitable parameter values based on your selections.

How to use the Matching Assistant is described here.

A reference to the elements of the Matching Assistant can be found here (page 223).

In this online help, the following special terms are used:

Matching Matching is the process of locating an object described by a *model* in an image. The results of the matching process are the position and orientation of the object and the matching *score*.

- **Shape-Based Matching** The *shape-based matching* describes the model by the shapes of contours instead of using the gray values of pixels and their neighborhood as template. In particular, *shape-based matching* should be chosen if occlusions or clutter can not be avoided or if a matching of objects with changing color is applied.
- **Deformable Matching** Like *shape-based matching*, the *local deformable matching* extracts contours and matches their shapes against the shapes of previously created models. But in contrast to *shape-based matching*, even slightly deformed objects are found and the deformations are returned as an additional result. This matching approach also allows you to rectify the image part containing the deformed object. Therefore, only the objects position is determined, whereas the orientation and scale are interpreted as part of the deformation.
- **Correlation-Based Matching** The *correlation-based matching* approach is based on gray values. It uses a normalized cross correlation to evaluate the correspondence between a *model* and a *test image*. It can compensate both additive as well as multiplicative variations in illumination. In contrast to the *shape-based matching*, also objects with slightly changing shapes, lots of texture, or objects in blurred images (contours vanish in blurred images, e.g., because of defocus) can be found.
- **Descriptor-Based Matching** In contrast to the perspective *deformable matching*, the template is not built by the shapes of contours but by a set of so-called interest points. These points are first extracted and then classified according to their location and their local gray-value neighborhood. Similar to the perspective *deformable matching*, the *descriptor-based matching* is able to find objects even if they are perspectively deformed. *Descriptor-based matching* can be applied for images from a calibrated camera as well as from an uncalibrated camera. This matching method provides a calibrated version with which also a 3D pose instead of 2D transformation parameters can be derived.

Alignment This method can be applied to transform the position of the matched object corresponding to the *reference image*. *Alignment* is useful if the following image processing step is not invariant against rotation or translation, like OCR or the variation model. Note that by *alignment* the matched object is only rotated and translated. To remove perspective or lens distortions, e.g., if the camera observes the scene under an inclined angle, you must *rectify* the image first. The matching assistant allows you to generate code for an affine transformation (page 248) which can subsequently be used for alignment with the Measure Assistant. *Alignment* is useful for all further processes that rely on fixed ROIs, like measuring and OCR.

Model In order to locate an object, you must provide the Matching Assistant with an example image of the object. From this, the Matching Assistant then creates the so-called *model*, an internal representation of the object containing only the information characterizing the object. This representation is then used when searching for the object in the *test images*. You can also provide the Matching Assistant with a *model* in DXF format (or in the HALCON formats SHM, NCM, DSM or DFM).

Model Image This is the image containing your example of the object to be searched for. This image should be a *characteristic* image of the object, i.e., the object should appear in its default position and orientation and not be occluded; furthermore, the image should not contain clutter. You can load this image via the menu item `File` ▷ `Load Model Image` (page 225).

Reference Image If a *reference image* is selected, the position of the match in this image is used as reference. This is necessary to perform *alignment*. If no reference image is chosen, the *model image* will be used as basis for *alignment*.

Model Region of Interest (ROI) This is the region in the model image which contains the object to be trained. You can select this region via the menu item `ROI` (page 226).

Rectification This method can be applied to the search image to transform it such that the found model and the model in the *reference image* appear as similar as possible. *Rectfying* an image is useful for all further processes that rely on fixed ROIs, like measuring and OCR.

Test Image You can test the performance of the matching process by providing *test images* via the menu item `Usage` ▷ `Test Images` ▷ `Load Test Images` (page 240). These images should be representative images from your matching application, i.e., the object should appear in all allowed variations of its position, orientation, occlusion, and illumination.

Score When comparing a detected match in a *test image* with the model, the Matching Assistant calculates a measure of similarity, the so-called *score*, which ranges between 0 (no similarity) and 1 (perfect similarity).

7.3.2 How to Use the Matching Assistant of HDevelop

By using the Matching Assistant, you can set up and optimize your matching application quickly and easily in four steps:

- Select a matching method,
- Create the model,
- Test the model, and
- Optimize the matching speed (page 220).

We recommend resetting all model and search parameters via the toolbar button `Reset` before starting with a new matching application. This way, the parameters are reset to their default settings and the model image, the model ROI, and the test images are deleted.

7.3.2.1 Selecting a Matching Method

Within the toolbar of the Matching Assistant, as well as on the `Parameters` tab, the matching method can be selected.

The assistant currently supports the following methods:

- shape-based matching (page 217),
- correlation-based matching (page 217),
- descriptor-based matching (page 217), and
- deformable matching (page 217).

The following table provides some information which matching method is **recommended** under which circumstances. For more information on matching methods and recommendations, please refer to the Solution Guide on Matching. The abbreviations used are *SBM* for shape-based matching, *CBM* for correlation-based matching, *DBM* for descriptor-based matching and *DM* for deformable matching.

Restrictions	SBM	CBM	DBM	DM
2D object	x	x	o	x
3D object	o	o	x	o
*Camera is **perpendicular** to object*	x	x	o	x
*Camera is **not perpendicular** to object*	o	o	x	o
***Scaling** is needed*	x	o	o	o
*Object is **colored/partially occluded**; image contains **clutter***	x	o	o	o
*Object is characterized by specific **texture***	o	x	o	o
*Images are highly **defocused***	o	x	o	o
***Local deformation** of the object is expected, rectification or information about deformations needed*	o	o	o	x

7.3.2.2 Creating the Model

A *model* (page 218) is created in three steps:

- Load the so-called *model image* (page 218) via the menu item `File` ▷ `Load Model Image` (page 225) or the corresponding button and text field under `Model` and `Model Source` inside the tab `Creation`.
- Create an *ROI* (page 218) around the object either via the menu item `ROI` (page 226) or via the corresponding buttons inside the tab `Creation`.
- Specify standard and advanced model parameter values that are available for your matching method in the tab `Parameters` (page 229).

Alternatively, you can load a model (page 225) that you have saved (page 225) either with the Matching Assistant or with HALCON or that is available as DXF model. You can also acquire a new image with the Image Acquisition Assistant (page 194) as source for the model creation.

Now, you can test the model on *test images* (page 218).

7.3.2.3 Testing the Model

After you created the *model* (page 218) you test it in the following steps:

- Load one or more *test images* (page 218) via the menu item `Usage` ▷ `Test Images` ▷ `Load Test Images` (page 240) or via the button `Load` inside the dialog `Test Images` in the tab `Usage`. Alternatively, you can also choose the Image Acquisition Assistant (page 194) as source of your test images by activating the corresponding checkbox under `Test Image Source`.
- Specify standard search parameter values via the menu item `Usage` ▷ `Standard Model Use Parameters` (page 242), which opens the corresponding dialog in the tab `Usage`. Especially the number of object instances (page 243) to search for in an image should be specified. If the number of object instances varies from test

image to test image, you can specify the number of visible objects (page 241) for each test image separately; in this case the search parameter mentioned above should be set to 0 or to the maximum number of visible objects. Use the button `Detect All` (page 240) to detect all matches and automatically set the maximum number of matches if it has not been set previously (i.e. was set to 0).

- Select a reference image (page 240) with `Set Reference` if you want to perform an alignment. The position of the match in this image is then used as reference. If no reference image is chosen, the model image (page 218) will be used as basis for alignment (page 217) and rectification (page 218).
- Assure that all objects are found (page 241) in all test images by comparing the number of existing models with the number of found models or simply determine the recognition rate (page 246).

Now, you can optimize the speed of the matching process by tuning the parameters.

7.3.2.4 Optimizing the Parameters

After you configured the *matching* (page 217) process such that the search is successful in all test images, you can start to optimize the parameters on the tabs `Parameters` and `Usage` to speed up the matching as far as possible.

For the four matching methods, different parameters are especially useful to improve speed:

- To support this process for **shape-based matching** and **deformable matching**, the Matching Assistant allows you to optimize the search parameters `Minimum Score` (page 242) and `Greediness` (page 243) on the tab `Usage`.
- For **correlation-based matching** the parameter `Minimum Score` (page 242) on the tab `Usage` should be set to a value larger than 0.0 but preferably below 0.1 to reduce the number of points considered for further calculations.
- For **descriptor-based matching** the model parameters `Fern Number` (page 235) and `Fern Depth` (page 235) on the tab `Parameters` have to be adjusted. Few ferns with a large depth enable a fast online matching which, however, requires more memory.

Search Parameters can also be automatically improved via the menu item `Use Model` ▷ `Optimize Recognition Speed` (page 246), which can be accessed also via the tab `Usage`.

If the reached recognition speed is not sufficient, you can try to modify parameters manually. However, please be aware that such a modification may result in a lower accuracy of the calculated position, orientation, or scale, or even prevent the Matching Assistant from finding the object! Therefore, we recommend checking whether the matching still succeeds in all *test images* (page 218) after each modification.

How the different parameters influence the recognition speed is described below. Please note that whenever you modify a model parameter, the internally stored model must be created anew; you must start this creation (and the search) explicitly using the button `Find Model` or the button `Detect All` in the tab `Usage`. After each modification determine the resulting recognition speed using the dialog `Optimize Recognition Speed` (page 246).

There are different ways of speeding up the matching process, depending on the matching method.

Learn more about speeding up your matching application.

7.3.2.5 Speed up matching

One or more of the following modifications can speed up

- shape-based matching,
- correlation-based matching,
- deformable matching (page 222), and
- descriptor-based (page 223).

To find out if a speed up is necessary or successful, it is useful to check the Statistics on the tab Inspect (page 246).

□ Speed up shape-based matching

Parameters ▷ Standard Model Parameters (page 231)

The following Standard Model Parameters may speed up shape-based matching:

- Number of Pyramid Levels (page 233):
  Increase the value and check whether the matching still succeeds in all images.
- Allowed range of rotation (page 233): Set the parameters Starting Angle (page 233) and Angle Extent (page 233), according to the range probably needed for your images.
- Set the allowed range of scale via Starting Angle (page 233), Min. Row Scale (page 234), Max. Row Scale (page 234), Min. Column Scale (page 234), and Max. Column Scale (page 234), according to the ranges probably needed for your images.

Parameters ▷ Advanced Model Parameters (page 235)

The following Advanced Model Parameters may speed up shape-based matching:

- Min. Contrast (page 238)
  Increase the value and check whether the matching still succeeds in all images.
- Optimization (page 237) (Point Reduction)
  Select a higher reduction rate and check whether the matching still succeeds in all images.
- Angle Step (page 236) size and Scale Step (page 236) size
  Increase the values and check whether the matching still succeeds in all images. Please note that the accuracy may suffer if you increase the step size!

Usage ▷ Advanced Use Parameters (page 243)

The following Advanced Use Parameters may speed up shape-based matching:

- Subpixel (page 244)
  If your application does not require subpixel accuracy, you can speed up the matching by selecting the value 'none'.
- Last Pyramid Level (page 244)
  Increase the value and check whether the matching still succeeds in all images. Note that as a result of this modification wrong instances of the model may be found. Furthermore, the accuracy of the calculated position, orientation, and scale may decrease.
- Timeout (page 245)
  This parameter enables you to specify a certain time in milliseconds after which the detection of model within an image is aborted.

□ Speed up correlation-based matching

Parameters ▷ Standard Model Parameters (page 231)

The following Standard Model Parameters may speed up correlation-based matching:

- Number of Pyramid Levels (page 233):
  Increase the value and check whether the matching still succeeds in all images.
- Allowed range of rotation (page 233): Set the parameters Starting Angle (page 233) and Angle Extent (page 233), according to the range probably needed for your images.

Parameters ▷ Advanced Model Parameters (page 235)

Correlation-based matching may also be sped up by selecting a higher reduction rate for Angle Step (page 236).

Usage ▷ Advanced Use Parameters (page 243)

The following Advanced Use Parameters may speed up correlation-based matching:

- Subpixel (page 244)

  If your application does not require subpixel accuracy, you can speed up the matching by selecting the value 'none'.

- Last Pyramid Level (page 244)

  Increase the value and check whether the matching still succeeds in all images. Note that as a result of this modification wrong instances of the model may be found. Furthermore, the accuracy of the calculated position, orientation, and scale may decrease.

- Timeout (page 245)

  This parameter enables you to specify a certain time in milliseconds after which the detection of model within an image is aborted.

□ Speed up deformable matching

Parameters ▷ Standard Model Parameters (page 231)

The following Standard Model Parameters may speed up deformable matching:

- Number of Pyramid Levels (page 233):

  Increase the value and check whether the matching still succeeds in all images.

- Allowed range of rotation (page 233): Set the parameters Starting Angle (page 233) and Angle Extent (page 233), according to the range probably needed for your images.

- Set the allowed range of scale via Starting Angle (page 233), Min. Row Scale (page 234), Max. Row Scale (page 234), Min. Column Scale (page 234), and Max. Column Scale (page 234), according to the ranges probably needed for your images.

Parameters ▷ Advanced Model Parameters (page 235)

The following Advanced Model Parameters may speed up deformable matching:

- Min. Contrast (page 238)

  Increase the value and check whether the matching still succeeds in all images.

- Optimization (page 237) (Point Reduction)

  Select a higher reduction rate and check whether the matching still succeeds in all images.

- Angle Step (page 236) size and Scale Step (page 236) size

  Increase the values and check whether the matching still succeeds in all images. Please note that the accuracy may suffer if you increase the step size!

Usage ▷ Advanced Use Parameters (page 243)

The following Advanced Use Parameters may speed up deformable matching:

- Subpixel (page 244)

  If your application does not require subpixel accuracy, you can speed up the matching by selecting the value 'none'.

- Last Pyramid Level (page 244)

  Increase the value and check whether the matching still succeeds in all images. Note that as a result of this modification wrong instances of the model may be found. Furthermore, the accuracy of the calculated position, orientation, and scale may decrease.

□ **Speed up descriptor-based matching**

Note that for descriptor-based matching, the number of points is important for speed and robustness. Therefore it is recommended to check the point number. Many points lead to a robust matching result, whereas few points speed up the matching process.

The following modifications can speed up descriptor-based matching:

`Parameters` ▷ `Standard Model Parameters` (page 231):

- `Detector Type` (page 234)

  Use 'lepetit' for the fastest extraction of significant points.

- `Fern Depth` (page 235) and `Fern Number` (page 235)

  Few ferns with a large depth are recommendend for a fast online matching but also require more memory. If memory is an issue but runtime is not, then many ferns with a small depth are recommended.

- Set the parameters `Min. Angle` (page 235), `Max. Angle` (page 235), `Min. Scale` (page 235), and `Max. Scale` (page 235) according to the ranges probably needed for your images. Restricting these parameters to small ranges speeds up the matching process.

`Parameters` ▷ `Advanced Model Parameters` (page 235):

- `Patch Size` (page 239)

  Choose a smaller value and therefore a smaller side length of the quadratic neighborhood that describes the individual interest point for this parameter.

- `Tilt` (page 239)

  Switching this parameter - and therefore the ability to cope with perspective transformations - 'off' enhances the speed.

`Usage` ▷ `Advanced Use Parameters` (page 243):

- `Descriptor Min. Score` (page 245)

  Setting this parameter to a value larger than 0.0 (but preferably below 0.1) increases the minimal classification score that determines if the individual points are considered as potential matches. Thus, the number of points for further calculations is reduced, so that the speed of the matching can be enhanced.

- `Guided Matching` (page 246)

  Switching this parameter off, can in some cases increase the runtime of the matching by up to 10%. Thus, if robustness is less important than speed, `Guided Matching` can be switched off.

7.3.2.6 Inspect matching results for adapting parameters

After changing parameters, the `Inspect` (page 246) tab is a very useful resource for statistical data (`Statistics`) that enables you to view how a matching application would perform based on these parameter settings and gives hints as to what can still be improved. If, for example, `Angle` appears with a high value which is not necessary for the application, decreasing this value manually via `Starting Angle` and `Angle Extent` on the `Parameters` tab increases speed.

7.3.2.7 Generate code

To finish the preparation for your matching application, generate Code with the tab `Code Generation` (page 247). This code includes all of the previous adaptions and can be used in the final application.

7.3.3 Matching Assistant Reference

The Matching Assistant consists of the following elements.

Pull-down menus:

- File
- ROI (page 226)
- Parameters (page 229)
- Usage (page 239)
- Inspect (page 246)
- Code Generation (page 247)
- Help (page 249)

Tool bar with a selection of important buttons:

- Load Assistant Settings (page 226)
- Save Current Assistant Settings (page 226)
- Insert Code (page 248)
- Save Model
- Display Model (page 227)
- Reset Model (page 218)
- Optimize Recognition Speed (page 246)
- Determine Recognition Rate (page 246)
- a drop-down menu that lets you select your matching method (page 218)
- Help (page 249)

Tabs with the dialogs for most of the tasks that can be done with the Matching Assistant:

- Creation (page 227)
- Parameters (page 229)
- Usage (page 239)
- Inspect (page 246)
- Code Generation (page 247)

Furthermore, it provides a status bar at the bottom in which messages are displayed. The status bar also displays the matching results, i.e., the number of found instances, the needed time, and for each found instance the position, orientation, scale, and score. Please note that the status bar does not provide a scrolling mechanism; if the displayed message is too long, move the mouse over it, so that a tool tip displaying the full message pops up. Alternatively, if the message is only slighly larger than the status bar, you can also drag the left or right border of the Matching Assistant window to enlarge it.

Images and models are displayed in the graphics window of HDevelop.

7.3.3.1 The Menu File

Via the menu File you can

- load the model image,
- load an already existing shape, correlation, descriptor, deformable or DXF model,
- save a model,
- load camera parameters for **descriptor-based or deformable matching**,
- load a camera pose for **descriptor-based or deformable matching** (page 226),
- display image pyramid levels (for shape-based matching) (page 228),

- load formerly used and saved settings (page 226) of the Matching Assistant,
- save the current settings (page 226) of the Matching Assistant for later use,
- close (page 226) the Matching Assistant dialog (while retaining the current settings as long as the HDevelop session is active), and
- exit the Matching Assistant dialog (discarding the settings).

□ Loading the Model Image

The so-called *model image* (page 218) is used to create the model of the object you want to find later. This image should be a *characteristic* image of the object, i.e., the object should appear in its default position and orientation and not be occluded; furthermore, the image should not contain clutter.

When you select the menu item `File` ▷ `Load Model Image` or activate the corresponding radio buttons in the dialog `Model` in the tab `Creation`, a standard file selection box appears. The Matching Assistant can read the image file types TIFF, BMP, GIF, JPEG, JPEG-XR, PPM, PGM, PNG, and PBM.

The selected image is displayed automatically. Typically, the next step is to create a region of interest around the object.

As an alternative to loading a model image and creating the model (page 219) interactively, the menu item `File` ▷ `Load Model` can be used to load a model that you have saved with the Matching Assistant or HALCON or that is available as DXF model.

Note that descriptor-based matching and correlation-based matching work on a single channel. For color RGB images, the red channel will be used. A color transformation can be performed with the operator `trans_from_rgb`.

□ Loading a Model

As an alternative to loading a model image and creating (page 219) the model interactively, the menu item `File` ▷ `Load Model` or the corresponding button `Load` in the tab `Creation` can be used to load a model that you have saved with the Matching Assistant or HALCON or that is available as DXF model.

The following settings are available for DXF models:

`Scale`**:** Scale the model by the specified factor.

`Use Image`**:** Tick this option to load a reference image for the model polarity. The reference image is loaded from

- the image currently displayed in the graphics window
- an image file,
- or from an `Acquisition Assistant` (page 194).

Note that when you load the model from a file, all the menu items, buttons, and dialogs that enable you to change the model parameters or display the model image will not be selectable because a loaded model cannot be changed and contains no information about the image from which it was created. Thus, e.g., the menu items `File` ▷ `Display Image Pyramid` (page 228), which is used to inspect the model, `Parameters` ▷ `Standard Model Parameters` (page 231), and `Parameters` ▷ `Advanced Model Parameters` (page 235), are disabled.

□ Saving a Model

The menu item `File` ▷ `Save Model` enables you to save the created model (page 219) in a file for later use. For example, the *model* (page 218) can be loaded into the Matching Assistant again in a later session with `File` ▷ `Load Model`.

□ Loading Camera Parameters for **descriptor-based matching or deformable matching**

This dialog allows you to load camera parameters from file. The internal camera parameters include `Cell Width (Sx)` and `Cell Height (Sy)` in micrometer, `Focal Length` in mm, `Center Column (Cx)` and `Center Row (Cy)`, `Image Width` and `Image Height` in pixels. They also include `Kappa` in $1/m^2$ or instead of `Kappa`, the

distortion parameters `Radial 2nd Order (K1)` in 1/ m^2 , `Radial 4th Order (K2)` in 1/ m^4 , `Radial 6th Order` in m^6 , `Tangential 2nd Order (P1)` and `Tangential 2nd Order (P2)` in 1/ m^2 for the polynomial area scan camera model.

If you have a line scan camera, additionally to the values of the `area scan camera (division)` model, values for the motion parameters `Motion x (Vx)`, `Motion y (Vy)` and `Motion z (Vz)` in micrometer/pixel will be returned.

□ Loading the Camera Pose for **descriptor-based matching or deformable matching**

This dialog allows you to load the camera pose from file. The 3D pose of the world coordinate system relative to the camera is described by the external camera parameters `X`, `Y` and `Z` in mm and `Rotation X`, `Rotation Y` and `Rotation Z` in degrees.

□ Loading Assistant Settings

If you have saved the settings of a former Matching Assistant session, you can load them again by the menu item `File` ▷ `Load Assistant Settings` or via the corresponding button of the tool bar. If the settings file refers to a model image file that is no longer available because it has been moved or deleted since, you can choose to select an alternate model image. If (some of) the test images cannot be loaded, a message box with the missing image file names is displayed.

□ Save Current Assistant Settings

You can save the current settings of a Matching Assistant session using the menu item `File` ▷ `Save Current Assistant Settings` or the corresponding button in the tool bar. Then, you can load them again in a later session.

□ Close the Matching Assistant Dialog

When closing the Matching Assistant dialog with the menu item `File` ▷ `Close Dialog`, the current settings are stored for the duration of the current HDevelop session. That is, as long as you do not exit HDevelop, you can again open the Matching Assistant with the same settings. In contrast to this, when you exit the Matching Assistant, the settings are lost also for the current HDevelop session.

□ Exit the Matching Assistant

When you exit the Matching Assistant with the menu item `File` ▷ `Exit Assistant`, the assistant's dialog is closed and the current settings are lost unless you have not stored them via the menu item `File` ▷ `Save Current Assistant Settings`. If you want to close the dialog but keep its settings for the current HDevelop session, you should use the menu item `Close Dialog` instead.

7.3.3.2 The Menu `ROI` and the Corresponding Buttons on the `Creation` Tab

Via the menu `ROI` or the corresponding buttons on the `Creation` tab, you can mark the region that serves as the model by *drawing* it on the displayed model image. The Matching Assistant provides different *ROI* (page 218) shapes: axis-parallel and arbitrarily oriented rectangles, circles and ellipses, as well as free-form shapes including polygons.

You draw rectangular, circular, and elliptic ROIs as follows: Select the corresponding drawing mode and click into the image. Then, move the mouse over the object while keeping the left mouse button pressed; the selected shape appears. After releasing the mouse button you can move the ROI by dragging its center (marked with a cross) with the left mouse button. Furthermore, you can edit the shape by dragging its boundaries. You finish the creation by clicking once with the right mouse button or by clicking the `Stop` button in the tool bar of the main window.

By selecting the menu item `ROI` ▷ `Draw Arbitrary Region` or the corresponding button on the `Creation` tab, you can create polygons and free-form shapes. To create a polygon click with the left mouse button to mark each corner point; a click with the right mouse button closes the polygon and finishes the creation. To create a free-form ROI draw it directly while keeping the left mouse button pressed; a click with the right mouse button closes the shape and finishes the creation. Note that in both cases you cannot edit the ROI after its creation!

In order to create an optimal model, please assure that the region of interest contains only characteristic parts of the object and no clutter! If clutter is unavoidable it is recommended to remove it with inpainting using the `Modify Model Image` menu item on the `Creation` tab.

You can refine the ROI you have drawn by viewing the region shapes on the `Creation` tab with the button `Show List of ROI Shapes`. A table then shows you your ROI data and lets you adapt the values. Remember that for polygons and free-form ROIs, values cannot be adapted.

After creating an ROI, you can specify model parameters (page 229).

7.3.3.3 The Tab `Creation`

In addition to the two settings described before, i.e., choosing your image source (see menu `File` ▷ `Load Model Image` (page 225) and creating an ROI (see menu `ROI` (page 226), in the tab `Creation` you can modify the model image.

Furthermore, two more settings are only available some of the matching methods:

- display the image pyramid is only available for **shape-based matching**, and
- choose a calibration source (page 229) is only available for **deformable matching** and **descriptor-based matching**.

If you choose to create a model, you can display the model image via the button `Display Model` in the toolbar of the Matching Assistant. If you loaded a model (page 225) from file, the model image is not available.

Parameters are specified in the tab `Parameters` (page 229).

□ Modifying the Model Image **Available for ALL Matching Methods**

Only 8-bit (byte) model images can be modified. To modify images of another bit depth, a prior conversion is required. For example, use `scale_image_max` to convert a 12-bit image into an 8-bit image.

This menu enables you to modify your model image such that only "wanted" contours are left or existing contours are improved. Enable the menu item `Modify Model Image` by activating the checkbox. With `Modify Model Image` you can

- inpaint regions,
- smooth regions,
- remove unwanted contours, and
- repair interrupted edges.

For these modification purposes, so-called 'modification ROIs' are used also referred to as 'regions' in this context, as opposed to normal ROIs or regions, these regions are simply there to mark areas for modification to improve the model.

You can draw modification ROIs by positioning the mouse cursor as explained in the section about the menu `ROI` (page 226). Use the right mouse button to conclude the choice and view the effects of the inpainting. To place your modification ROI even more precisely, you can view the modification ROI's region primitives with the corresponding button that is located at the same height as the modification ROI buttons and also adapt the values in the list. For deleting all modification ROIs, use the corresponding button next to the modification ROI buttons.

The checkboxes `Display unmodified Model Image` and `Display modification ROIs` can be activated during the modification process to view the image as it was before it was modified and display the modification ROIs, respectively.

In order to keep the data of the modifications, you can save the modified image as well as the modification ROIs. For saving the image, click on the `Save Image` button on the right. For saving the modification ROIs, click on the corresponding button on the right of the modification ROI buttons. A star next to the `Enable` checkbox reminds you that modifications have been made that have not been saved. You can load your saved modification ROIs again with the corresponding button on the right of the modification ROI buttons.

◇ Inpaint Regions, Inpaint Region Smooth

If an model image is disturbed by clutter or unwanted structures, these can be removed by choosing `Inpaint Regions` from the drop down menu. If you just want to smooth an area within an ROI without completely removing structures, choose `Inpaint Region Smooth`.

For using either `Inpaint Regions` or `Inpaint Regions Smooth`, choose a shape for your modification ROI from the buttons on the right. The shape should not only cover the area that is to be removed but also some of the "good" area around it. You have the choice between circle, ellipse, axis-aligned rectangle and if none of these regions is suitable, it is also possible to draw an arbitrary region.

◇ Remove Contours

`Remove Contours` enables you to choose contours you want to delete from your model. Activate the function by clicking on the ROI button and start to remove edges by selection. Also choose the `Strength` of the removal, i.e., the smoothing width, either by manipulating the values in the box or using the slider. Then move your mouse cursor over the image of the model in the graphics window. Selected contours are highlighted when the mouse cursor moves over them. To remove a contour, simply click the left mouse button while it is activated. Note, that `Auto Selection` is activated as default under `Specify Standard Model Parameters` and therefore will automatically adapt to the new model characteristics which might even lead to a deleted contour that is found again after the automatical adaption.

◇ Repair Edges

If a contour is interrupted where it should be continuous, `Repair Edges` can be used to draw a linear Modification ROI that connects the two ends of the existing contour and therefore replaced the missing edges. First choose the necessary `Edge Thickness` either by directly adjusting the values in the box or by using the slider. Then draw a modification ROI into the gap within the contour.

Finally save your modifications of the model image. Note that if the default `Auto Selection` is activated, the thresholds will be adapted to the new image features. If this leads to unfavorable results - either deactivate `Auto Selection` or change the parameter once the modification is finished.

Possible ways to continue setting your matching parameters are:

- for **ALL matching methods**: specify standard model parameters (page 231),
- for **shape-based matching**: set parameters for viewing image pyramid levels,
- for **descriptor-based matching** and **deformable matching**: choose a calibration source.

□ Displaying the Image Pyramid **Available for Shape-Based Matching**

Using the dialog `Display Image Pyramid` (accessed via the menu item `Create Model` ▷ `Display Image Pyramid` or directly inside the tab `Creation`), you can display the *model image* (page 218) (see the description in the section 'The Tab `Creation`' (page 227)) and inspect the different model pyramid levels and the corresponding images by

- selecting which model level is displayed,
- selecting which image level is displayed, and
- locking or unlocking model and image level.

◇ Displaying the Model on the Different Pyramid Levels

You can select the desired pyramid level of the model by using the slider or text box for `Model` inside the dialog `Display Image Pyramid` of the tab `Creation`. The model is overlaid onto the pyramid image selected with the slider or text box `Image` within the same dialog. By default, the model and the image are displayed on the same pyramid level; you can unlock and again lock the levels using the lock/unlock button right to the sliders.

Note that the highest available pyramid level is determined automatically by the Matching Assistant based on the size of the model *ROI* (page 218); depending on the selected `Contrast` (page 232) and `Min. Component Size` (page 232), higher pyramid levels may not contain any model points.

Detailed information about the model image pyramid can be found here (page 233).

◇ Displaying the Model Image on the Different Pyramid Levels

You can select the desired pyramid level of the model image using the slider or text box for `Image` inside the dialog `Display Image Pyramid` of the tab `Creation`. Onto this image, the model on the pyramid level selected with the slider or text box for `Model` (page 228) within the same dialog is overlaid. By default, the model and the image are displayed on the same pyramid level; you can unlock and again lock the levels using the lock/unlock button right to the sliders.

Note that the highest pyramid level available for display is determined automatically by the Matching Assistant based on the size of the model *ROI* (page 218); depending on the selected `Contrast` (page 232) and `Min. Component Size` (page 232), higher pyramid levels may not contain any model points.

Detailed information about the model image pyramid can be found here (page 233).

◇ Locking the Display of Model and Image Pyramid

By default, the pyramid levels of the displayed model (page 228) and model image are locked. When pressing the unlock button right to the sliders, which are used for specifying the pyramid levels, you can select different pyramid levels for the model image and the model. When pressing the button again, both levels are locked again.

Detailed information about the model image pyramid can be found here (page 233).

Once you are finished with setting parameters in this section, continue to specify standard model parameters (page 231).

□ Choosing a Calibration Source **Available for Descriptor-Based Matching and Deformable Matching**

Using the dialog `Calibration Source` within the tab `Creation`, you can choose the source of your calibration data, if you want to use it for your matching application or start a new calibration to gain calibration data with the Calibration Assistant (page 199). In this dialog you can choose between:

- `None` (no calibration data is used),
- Loading `Calibration Files` (files that end with .cal or .dat), and
- using the Calibration Assistant (page 199) which allows you to conveniently perform a calibration by guiding you step-by-step through the calibration process.

Once you are finished with setting parameters in this section, continue to specify standard model parameters (page 231).

7.3.3.4 The Tab `Parameters` and the Corresponding Entries in the Menu `Parameters`

The Matching Assistant allows you to adapt

- standard parameters as well as
- advanced parameters

for your matching application.

Depending on the matching method, different parameters are available for adaption.

Read more about setting standard parameters (page 231).

The following table gives you an overview over the available standard model parameters for each matching method: The abbreviations used are *SBM* for shape-based matching, *CBM* for correlation-based matching, *DBM* for descriptor-based matching and *DM* for deformable matching.

Parameter Name	SBM	CBM	DBM	DM

Contrast (High/Low) (page 232)	x	o	x	x
Min. Component Size (page 232)	x	o	o	o
Pyramid Levels (page 233)	x	x	o	x
Starting Angle (page 233)	x	x	o	x
Angle Extent (page 233)	x	x	o	x
Min./Max. Angle (page 235)	o	o	x	o
Min./Max. Row Scale (page 234)	x	o	o	x
Min./Max. Column Scale (page 234)	x	o	o	x
Min./Max. Scale (page 235)	o	o	x	o
Model Type (page 232)	o	o	o	x
Detector Type (page 234)	o	o	x	o
Radius (page 234)	o	o	x	o
Min. Score (page 235)	o	o	x	o
Fern Depth (page 235)	o	o	x	o
Fern Number (page 235)	o	o	x	o
Gradient Sigma (page 234)	o	o	x	o
Gradient Mask Size (page 234)	o	o	x	o
Threshold (page 234)	o	o	x	o

Read more about setting advanced parameters (page 235).

The following table gives you an overview over the available advanced model parameters for each matching method: The abbreviations used are *SBM* for shape-based matching, *CBM* for correlation-based matching, *DBM* for descriptor-based matching and *DM* for deformable matching.

Parameter Name	**SBM**	**CBM**	**DBM**	**DM**
Angle Step (page 236)	x	x	o	x
Row/Column Scale Step (page 236)	x	o	o	x
Metric (page 237)	x	x	o	x
Optimization (page 237)	x	o	o	x
Pregenerate Shape Model (page 237)	x	o	o	o
Min. Contrast (page 238)	x	o	o	x
Check Neighbor (page 238)	o	o	x	o
Neighbor Diff. Threshold (page 238)	o	o	x	o
Subpixel (page 238)	o	o	x	o
Patch Size (page 239)	o	o	x	o
Tilt (page 239)	o	o	x	o
Smoothing Sigma (page 238)	o	o	x	o
Alpha (page 238)	o	o	x	o
Smoothing Mask Size (page 238)	o	o	x	o

□ Specifying Standard Model Parameters

Via the menu item `Parameters` ▷ `Standard Model Parameters` the tab `Parameters` is opened and you can specify basic parameters for the model, which describe the *appearance* of the object to recognize, e.g., the contrast of significant points or the allowed range of rotation.

By default, these parameters are set to values which work well for most tasks; by modifying them, you can optimize the model for your application and speed up the search process.

For **shape-based matching** the following parameters can be specified:

- the `Contrast` which points must have in order to be included in the model,
- the `Min. Component Size` of model components,
- the number of `Pyramid Levels` (page 233) on which the model is created,
- the `Starting Angle` (page 233) of the allowed range of rotation,
- the allowed range of rotation (`Angle Extent` (page 233)), and
- the scale range (page 234).

For **correlation-based matching** the following parameters can be specified:

- the number of `Pyramid Levels` (page 233) on which the model is created,
- the `Starting Angle` (page 233) of the allowed range of rotation, and
- the allowed range of rotation (`Angle Extent` (page 233)).

For **deformable matching** the following parameters can be specified:

- the `Contrast` which points must have in order to be included in the model,
- the `Model Type` which allows you to choose the type for the deformable matching model,
- the number of `Pyramid Levels` (page 233) on which the model is created,
- the `Starting Angle` (page 233) of the allowed range of rotation,
- the allowed range of rotation (`Angle Extent` (page 233)), and
- the scale range (page 234).

For **descriptor-based matching** the following parameters can be specified:

- the `Detector Type` (page 234) ,
- the `Min. Score` (page 235),
- the `Fern Depth` (page 235),
- the `Fern Number` (page 235),
- the `Min. Angle` and `Max. Angle` (page 235), and
- the `Min. Scale` and `Max. Scale` (page 235).

If you want to reset all of the model and search parameters, you can do this via the `Reset Model` button in the toolbar.

Note that - due to performance reasons - viewing the effect of your adaptions of the shape-based matching parameters `Contrast (Low)`, `Contrast (High)` and `Min. Component Size`, will only immediately be possible if the model image is small and if it is not too noisy. The reason for this is that changes in those parameters lead to a new calculation of the model. For other images, the system will wait until the changes are loaded instead of blocking the graphical user interface. This enables you to finish your adaptions of the model parameters without having to wait.

In most applications, specifying the standard parameters will already suffice. Therefore, you can directly test the model (page 219) now. Additionally, advanced model parameters can be specified via the menu item `Parameters` ▷ `Advanced Model Parameters` (page 235).

◇ The Model Parameters `Contrast (Low/High)` (**Shape-based Matching**) and `Contrast` (**Deformable Matching**)

The two parameters `Contrast (Low)` and `Contrast (High)` for shape-based matching and `Contrast` for deformable matching determine which pixels in the selected *ROI* (page 218) are included in the *model* (page 218); typically, the points corresponding to the contours of the object should be selected.

When you select a value, either by using the sliders or by entering a value in the text fields next to them, the included pixels are marked in the displayed image. In order to obtain a suitable model we recommend choosing the contrast in such a way that the *significant* pixels of the object are included, i.e., those pixels that characterize it and allow you to discriminate it clearly from other objects or from the background. Please assure that no clutter is included, i.e., pixels that do not belong to the object!

You can use the parameters `Contrast (Low)` and `Contrast (High)` in two ways:

1. **Simple threshold**:

   Set both parameters to the same value. Then, all pixels with a contrast higher than this value are included in the model.

   You can modify both parameters at the same time as follows: To increase the value, use the slider of `Contrast (Low)`; then, the value `Contrast (High)` will follow automatically. Vice versa, to decrease the value use the slider of `Contrast (High)`.

2. **Hysteresis threshold:**

   If there is no single contrast value that selects all significant object pixels without including clutter, try using different values for `Contrast (Low)` and `Contrast (High)`. Then, pixels are selected in two steps: First pixels that have a contrast higher than `Contrast (High)` are selected; then, pixels that have a contrast higher than `Contrast (Low)` and that are connected to a high-contrast pixel, either directly or via another pixel with contrast above the lower threshold, are added.

   We recommend that you proceed as follows: Increase both values (using the slider of `Contrast (Low)`), until no clutter pixels are selected anymore. Then, decrease `Contrast (Low)` to add more object pixels. If significant object parts remain unselected, decrease `Contrast (High)`.

Note that these parameters are used only to select model points in the model image. In the test images, the object may have a lower contrast.

You can also let the Matching Assistant select suitable values automatically based on the model image.

An additional method for removing clutter for shape-based matching is to specify a minimum size for the model components. If you cannot find suitable parameter values that exclude the clutter, we recommend creating a new model ROI via the menu `ROI` (page 226).

◇ Letting the Matching Assistant Select a Suitable Value for `Contrast` (**Shape-based Matching and Deformable Matching**)

When you click the button `Auto Select` that is placed right beside the sliders for the parameters `Contrast (Low/High)` and `Contrast`, the Matching Assistant selects suitable values for the contrast by trying to obtain many long and straight contour segments.

Note that you may need to set the value manually if certain model components should be included or suppressed because of application-specific reasons or if the object contains several different contrasts.

◇ The Model Parameter `Model Type` (**Deformable Matching**)

Via the combo box `Model Type` you can select two model types,

- `locally deformable` and
- `ridgid`

depending on the expectations concerning the deformation of the object.

◇ The Model Parameter `Min. Component Size` (**Shape-based Matching**)

The parameter `Min. Component Size` specifies the minimum size, i.e., number of pixels, which contour parts must have to be included in the *model* (page 218). This parameter is useful to exclude clutter.

You can also let the Matching Assistant select a suitable value automatically based on the model image.

Note that the selected value is divided by two for each successive pyramid level.

◇ Letting the Matching Assistant Select a Suitable Value for `Min. Component Size` (**Shape-based Matching**)

When you click the button `Auto Select` that is placed right beside the slider for the parameter `Min. Component Size` (page 232) the Matching Assistant selects a suitable value for the minimum component size based on the model image.

◇ The Model Parameter `Pyramid Levels` (**Shape-based Matching, Correlation-based Matching, Deformable Matching**)

To speed up the matching process, a so-called *image pyramid* is created, both for the model image and for the search images. The pyramid consists of the original, full-sized image and a set of downsampled images. For example, if the original image (first pyramid level) has the size 600x400, the second level image has the size 300x200, the third level 150x100, and so on. The object is then searched for first on the highest pyramid level, i.e., in the smallest image. The results of this fast search are then used to limit the search in the next pyramid image, whose results are used on the next lower level until the lowest level is reached. Using this iterative method the search is both fast and accurate.

You can inspect the model image pyramid together with the corresponding models via the menu item `File` ▷ `Display Image Pyramid` (page 228), which opens the corresponding dialog of the tab `Creation`. We recommend choosing the highest pyramid level at which the model contains at least ten pixels (and still resembles the original shape). You can enter the value directly in the text field or by using the slider next to it. Alternatively, you can let the Matching Assistant select a suitable value automatically.

Note that the Matching Assistant can check whether the model contains enough points on the selected number of pyramid levels only when actually creating the model. In case the model does not contain enough model points a corresponding error dialog appears.

◇ Letting the Matching Assistant Select a Suitable Value for `Pyramid Levels` (**Shape-based Matching, Correlation-based Matching, Deformable Matching**)

When you click the button `Auto Select` that is placed right beside the slider for the parameter `Pyramid Levels` the Matching Assistant selects a suitable number of pyramid levels automatically, thus relieving you of the task of examining the model image pyramid.

Please note that in rare cases the automatic selection will yield a too low value and thereby slow down the search process, or a too high value, resulting in failures to recognize the object. In such a case we recommend that you inspect the model image pyramid (page 228) and select a suitable value manually.

◇ The Model Parameter `Starting Angle` (**Shape-based Matching, Correlation-based Matching, Deformable Matching**)

With the parameter `Starting Angle` you can specify the starting angle of the allowed range of rotation (unit:°). With another parameter you can specify the extent of the allowed range. Note that the range of rotation is defined relative to the model image, i.e., a starting angle of 0° corresponds to the orientation the object has in the model image. Therefore, to allow rotations up to +/-5°, e.g., you should set the starting angle to -5° and the angle extent to 10°.

◇ The Model Parameter `Angle Extent` (**Shape-based Matching, Correlation-Based Matching, Deformable Matching**)

With the parameter `Angle Extent` you can specify how much the object is allowed to rotate (unit:°). With another parameter you can specify the starting angle of this allowed range. Note that the range of rotation is defined relative to the model image, i.e., a starting angle of 0° corresponds to the orientation the object has in the model image. Therefore, to allow rotations up to +/-5°, e.g., you should set the starting angle to -5° and the angle extent to 10°.

We recommend limiting the allowed range of rotation as much as possible in order to speed up the search process and to minimize the required memory. If the loaded test images (page 240) show the object in its extreme orientations, you can let the Matching Assistant determine the range of rotation, i.e., the Pose Bounds (page 247), by pressing the `Run` button of the tab `Inspect` and viewing the result in the `Statistics` output of the same tab.

Furthermore, you must limit the allowed range if the object is (almost) symmetrical. Otherwise the search process will find multiple, almost equally good matches on the same object at different angles; which match (at which angle) is returned as the best can therefore "jump" from image to image. The suitable range of rotation depends on the symmetry: For a cross-shaped or square object the allowed extent must be less than 90°, for a rectangular object less than 180°, and for a circular object 0°.

Note that if you use shape-based matching and have chosen a very large angle and scale range you may find it useful to switch off the complete pregeneration (page 237) of the model if it has been switched on before (default = switched off).

◇ The Model Parameters for the Scale Range (**Shape-based Matching, Deformable Matching**)

The allowed range of scale is defined separately in row and column direction. Thus, it is described by the parameters:

- `Min. Row Scale`
- `Max. Row Scale`
- `Min. Column Scale`
- `Max. Column Scale`

In the model image, the scales all have the value 1.0.

Note that if you use shape-based matching and have chosen a very large angle extent (page 233) and scale range you may find it useful to switch off the complete pregeneration (page 237) of the model.

Depending on the specified parameters, the most efficient matching method is used. This method determines how the shape model is created in the generated code.

- **Unscaled matching**:
  This method is used if all four scale factors are equal to 1.0.
- **Scale invariant matching**:
  This method is used if all four scale factors are equal (but not 1.0) or if the scale parameters are locked.
- **Anisotropic scale invariant matching**:
  This method is used if none of the above applies.

◇ The Model Parameter `Detector Type` (**Descriptor-based Matching**)

The interest points that build the model are extracted from the image by the so-called detector. The type of detector is selected via the parameter `Detector Type`. Available types are:

- `lepetit`,
- `harris`, or
- `harris_binominal`.

`lepetit` can be used for a very fast extraction of significant points, but the obtained points are not as robust as those obtained with `harris`. Especially, if a template or search image is very dark or has got a low contrast, `lepetit` is not recommendend. `harris_binomial` is a good compromise between `lepetit` and `harris`, because it is faster than `harris` and more robust than `lepetit`.

Depending on the detector type that is chosen, different values can be specified.

For the `Detector Type` **lepetit**, the following value can be specified:

- `Radius`

For the `Detector Type` **harris**, the following values can be specified:

- `Gradient Sigma` and
- `Threshold`.

For the `Detector Type` **harris_binominal**, the following values can be specified:

- `Gradient Mask Size` and
- `Threshold`.

For more information and recommended values, please refer to the reference documentation of the operator **create_uncalib_descriptor_model**.

◇ The Model Parameter `Min. Score` (**Descriptor-based Matching**)

In most cases, `Min. Score` should be set to a value of at least 0.1. To speed up the search, it should be chosen as large as possible, but of course still as small as necessary for the success of the search, as, e.g., a value of 1.0 is rather unlikely to be reached by a matching. Thus, the number of points for further calculations is reduced, so that the speed of the matching can be enhanced. But note that this speed-up is obtained at the cost of a reduced robustness, especially if the number of extracted points is low.

◇ The Model Parameter `Fern Depth`

`Fern Depth` specifies the depth of the classification fern. Interest points can be better discriminated when selecting a higher depth. On the other hand, a higher depth leads to an increasing runtime.

◇ The Model Parameter `Fern Number` (**Descriptor-based Matching**)

`Fern Number` specifies the number of used fern structures. Using many fern structures leads to a better robustness but also to an increasing runtime.

The selection of values for the depth and the number of ferns depends on your specific requirements. If a fast online matching is required, few ferns with a large depth are recommendend. Note that with these parameter settings more memory is required. If a very robust matching result is needed and neither memory consumption nor runtime are critical, many ferns and a large depth are recommended. Note that this might significantly increase the runtime of the matching. If the memory consumption is critical but runtime is not, many ferns with a small depth are recommended.

◇ The Model Parameters `Min. Angle` and `Max. Angle` (**Descriptor-based Matching**)

`Min. Angle` and `Max. Angle` define the range for the angle of rotation around the normal vector of the model. The restriction to small ranges for the angle can be used to speed up the training significantly. But note that during the later applied matching the model can be found only if its angle and scale is within the trained scope.

◇ The Model Parameters `Min. Scale` and `Max. Scale` (**Descriptor-based Matching**)

`Min. Scale` and `Max. Scale` define the scale range of the model. The restriction to small ranges for the scale can be used to speed up the training significantly. But note that during the later applied matching the model can be found only if its angle and scale is within the trained scope.

□ Specifying Advanced Model Parameters

In most applications, specifying the `Standard Model Parameters` (page 231) will already suffice. The menu item `Parameters` ▷ `Advanced Model Parameters` provides additional parameters that let you handle special cases like changing the contrast polarity or enable you to further optimize the model.

For **shape-based matching** the following parameters can be specified:

- the `Angle Step` at which the model is created,
- the scale steps, `Row Scale Step` and `Column Scale Step`, at which the model is created,

- whether to use the polarity of the contrast (`Metric` (page 237)) in the model,
- whether to optimize the model (page 237) by using a reduced number of points,
- whether to pregenerate the model completely (page 237), and
- the `Min. Contrast` (page 238) points must have in a search image to be compared with the model.

For **correlation-based matching** the following `Advanced Model Parameters` can be specified:

- the `Angle Step` at which the model is created, and
- whether to use the polarity of the contrast (`Metric`) in the model.

For **deformable matching** the following parameters can be specified:

- the `Angle Step` at which the model is created,
- the scale steps, `Row Scale Step` and `Column Scale Step`, at which the model is created,
- whether to use the polarity of the contrast (`Metric`) in the model,
- whether to optimize the model by using a reduced number of points, and
- the `Min. Contrast` (page 238) points must have in a search image to be compared with the model.

For **descriptor-based matching** the following parameters can be specified:

- parameters relating to the `Detector Type` (page 238),
- the parameter `Patch Size` (page 239), and
- the parameter `Tilt` (page 239).

If you want to reset all of the model and search parameters, you can do this via the `Reset` button in the toolbar.

Once you are finished setting the `Advanced Model Parameters`, you can continue to work with test images in the menu `Usage` (page 239).

◇ The Model Parameter `Angle Step` (**Shape-based Matching, Correlation-based Matching, Deformable Matching**)

The standard model parameters `Starting Angle` (page 233) and `Angle Extent` (page 233) specify the step size at which the angle range is sampled.

Note that each time you create a model *ROI* (page 218) or change the parameter `Contrast` (page 232), the Matching Assistant automatically selects a suitable value for `Angle Step` to obtain the highest possible accuracy. You can select a higher value manually. This may be useful to speed up the search process in special cases; please note however, that a large value may decrease the accuracy of the estimated orientation and even prevent the Matching Assistant from finding the object! You can restore the automatically selected value by clicking the button `Auto Select`.

If you already loaded test images (page 240) you can quickly test the effect of the selected parameter value via the menu item `Inspect` ▷ `Determine Recognition Rate` (page 246).

◇ Letting the Matching Assistant Select a Suitable Value for `Angle Step` (**Shape-based Matching, Correlation-based Matching, Deformable Matching**)

When you click the button `Auto Select` that is placed right beside the slider for the parameter `Angle Step` the Matching Assistant selects a suitable value for the angle step size to obtain the highest possible accuracy.

◇ The Model Parameters `Row Scale Step` and `Column Scale Step` (**Shape-based Matching, Deformable Matching**)

The standard model parameters for the scale range (page 234) specify how much the object is allowed to be scaled in row and column direction.

Note that each time you create a model *ROI* (page 218) or change the parameter `Contrast` (page 232), the Matching Assistant automatically selects a suitable value to obtain the highest possible accuracy. You can select a higher

value manually. This may be useful to speed up the search process in special cases; please note however, that a large value may decrease the accuracy of the estimated orientation and even prevent the Matching Assistant from finding the object! You can restore the automatically selected value by clicking the button `Auto Select`.

If you already loaded test images (page 240), you can quickly test the effect of the selected parameter value via the menu item `Inspect` ▷ `Determine Recognition Rate` (page 246).

◇ Letting the Matching Assistant Select a Suitable Value for `Row Scale Step` and `Column Scale Step` (**Shape-based Matching, Deformable Matching**)

When you click the button `Auto Select` that is placed right beside the sliders for the parameters `Row/Column Step Size` (page 236), the Matching Assistant selects suitable values for both scale step sizes based on the model image.

◇ The Model Parameter `Metric` (**Shape-based Matching, Correlation-based Matching, Deformable Matching**)

The parameter `Metric` lets you choose whether the *polarity of the contrast* is to be observed when comparing a test image with the model. By default, the polarity is used ('use_polarity'), i.e., the points in the test image must show the same direction of the contrast as the corresponding points in the model. This means that, for example, if a model appears bright on dark background, only bright objects on dark background can be detected.

You can choose to ignore the polarity *globally* ('ignore_global_polarity'), at the cost of a slightly lower recognition speed. In this mode, an object is recognized also if the direction of its contrast reverses, e.g., if your object can appear both as a dark shape on a light background and vice versa.

A third mode lets you ignore the polarity *locally* ('ignore_local_polarity'), i.e., objects are also recognized if the direction of the contrast changes only in some parts. This mode can be useful, e.g., if the object consists of a part with a medium gray value, within which either darker or brighter sub-objects lie. Please note, however, that the recognition speed as well as the robustness decreases dramatically in this mode, especially if you allowed a large range of rotation (page 233).

Finally, you can choose to ignore the color polarity ('ignore_color_polarity') to apply shape based matching to multi-channel images.

If you already loaded test images (page 240) you can quickly test the effect of the selected parameter value via the menu item `Inspect` ▷ `Determine Recognition Rate` (page 246).

◇ The Model Parameter `Optimization` (**Shape-based Matching, Deformable Matching**)

After you created a model *ROI* (page 218), by default all points showing the required `Contrast` (page 232) and belonging to components larger than the `Min. Size` (page 232) are selected for the *model* (page 218) and marked in the image. For particularly large models, i.e., models with a large number of model points, it might be useful to reduce the number of points using the parameter `Optimization` to speed up the *matching* (page 217) process and to reduce memory requirements. You can select a low, medium, or high point reduction; please note that regardless of your selection all points passing the contrast criterion are displayed, i.e., you cannot check which points are part of the model.

You can also let the Matching Assistant select a suitable value automatically based on the model image.

Another possibility to reduce the memory requirements of the model is to switch off the complete pregeneration of the model if it has been activated before (default = off).

If you already loaded test images (page 240) you can quickly test the effect of the selected parameter value via the menu item `Inspect` ▷ `Determine Recognition Rate` (page 246).

◇ Letting the Matching Assistant Select a Suitable Value for `Optimization` (**Shape-based Matching, Deformable Matching**)

When you click the button `Auto Select` that is placed right beside the slider for the parameter `Optimization` the Matching Assistant optimizes, i.e., reduces the number of model points based on the model image.

◇ The Model Parameter `Pregenerate Shape Model` (**Shape-based Matching**)

The parameter Pregenerate Shape Model specifies whether the internal representation of the shape model is pregenerated completely whenever the model is created.

If you select a complete pregeneration by checking the check box Pregenerate Shape Model the model generation may require a substantial amount of time and memory. In contrast, if you switch off the complete pregeneration, the model creation will be very fast and the model will consume less memory.

The advantage of selecting a complete pregeneration is that the model can possibly be found slightly faster than if the complete pregeneration is switched off. Typically, you may find it useful to switch off the complete pregeneration if your model uses a large angle and scale range.

◇ The Model Parameter Min. Contrast (**Shape-based Matching, Deformable-Matching**)

In order to select significant object points for the *model* (page 218) you have specified which Contrast (page 232) the points must show in the *model image* (page 218). With the parameter Min. Contrast you can specify a separate minimum contrast for the *matching* (page 217) process itself, i.e., when searching for the object in the *test images* (page 218). The main use of this parameter is to exclude noise from the matching process.

Note that a low value for Min. Contrast slows down the matching process because more points in the test image must be compared with the model. Therefore, we recommend choosing a value which is higher than the noise level in the test images. You can also let the Matching Assistant select a suitable value automatically based on the model image.

Note that although this parameter is only used during the search, it is already included when creating the model to speed up the matching process.

If you already loaded test images (page 240) you can quickly test the effect of the selected parameter value via the menu item Inspect ▷ Determine Recognition Rate (page 246).

◇ Letting the Matching Assistant Select a Suitable Value for Min. Contrast (**Shape-based Matching, Deformable Matching**)

When you click the button Auto Select that is placed right beside the slider for the parameter Min. Contrast the Matching Assistant selects a suitable value for the minimum contrast by estimating the noise in the model image.

Note that an automatic determination only makes sense if the image noise during the recognition is similar to the noise in the model image. For this reason, it is typically not useful when using a synthetic model image (without noise).

◇ Parameters relating to the Detector Type

The interest points that build the model are extracted from the image by the so-called detector. The type of detector is selected via the parameter Detector Type. Available types are:

- lepetit,
- harris, or
- harris_binominal.

For more information on these types, see the description of the parameter Detector Type (page 234).

Depending on the detector type that is chosen, different values can be specified.

For the Detector Type **lepetit**, the following value can be specified:

- Check Neighbor
- Neighbor Diff. Threshold
- Subpixel

For the Detector Type **harris**, the following values can be specified:

- Smoothing Sigma and
- Alpha.

For the `Detector Type` **harris_binominal**, the following values can be specified:

- `Alpha` and
- `Subpixel`.

For more information and recommended values, please refer to the reference documentation of the operator **create_uncalib_descriptor_model**.

◇ The Model Parameter `Patch Size`

`Patch Size` specifies the side length of the quadratic neighborhood that is used to describe the individual interest point. A value that is too large may increase the runtime.

◇ The Model Parameter `Tilt`

`Tilt` is used to switch 'on' or 'off' the projective transformations during the simulation phase, which leads either to an enhanced robustness of the model or to a speed-up of the training.

7.3.3.5 The Menu `Usage` and the Tab `Usage`

Via the menu `Use Model` as well as the tab `Usage` you can

- load test images,
- delete a selected test image,
- delete all test images at once,
- display the selected test image (page 241),
- access the test image settings in the tab `Usage`,
- open the dialog for the standard (page 242) and advanced search parameters (page 243),
- open the dialog for the optimization of the recognition speed (page 246), and
- directly start to optimize the recognition speed (page 246).

In the tab `Usage` you can additionally

- open the Image Acquisition Assistant (page 194) to acquire images,
- select a test image (page 241),
- specify the number of visible objects (page 241) in the image, and
- start the matching for a selected test image or
- for the whole sequence of test images.

□ Test Images

With the menu item `Usage` ▷ `Test Images` you can

- load test images,
- delete a selected test image or delete all test images,
- display an already selected test image (page 241), and
- open the dialog `Test Images` inside the tab `Usage`

The dialog `Test Images` inside the tab `Usage` you need to additionally

- select a test image (page 241) for display or deletion,
- select a test image as reference,

- specify the number of visible objects (page 241) for each image, and
- search for the model in the complete sequence of test images, in the currently selected test image, or automatically after each selection (page 241).

□ Loading Test Images

The so-called *test images* (page 218) should be representative images from your matching application, i.e., the object should appear in all allowed variations of its position, orientation, occlusion, and illumination.

When you select the menu item `Usage` ▷ `Test Images` ▷ `Load Test Images` (or click the corresponding button `Load` in the tab `Usage`), a standard file selection box appears, in which you can select one or more images to load. The Matching Assistant can read the image file types TIFF, BMP, GIF, JPEG, JPEG-XR, PPM, PGM, PNG, and PBM. Please note that the test images must have the same size as the model image!

A dialog appears in the tab `Usage` which enables you to test the matching (page 239) on the loaded images.

□ Removing a Test Image

When you select the menu item `Usage` ▷ `Test Images` ▷ `Remove Test Image` or click the button `Remove` inside the dialog `Test Images` of the tab `Usage`, the currently selected test image is removed from the list of test images. You can select a test image by clicking onto its index number or path in the text field left to the buttons.

You can also delete all test images at once.

□ Removing All Test Images

When you select the menu item `Usage` ▷ `Test Images` ▷ `Remove All Test Images` or click the button `Remove All` in the dialog `Test Images` of the tab `Usage`, all test images are removed from the list of test images.

You can also delete a selected test image.

□ Select Test Image as Reference Image

When you select an image in the tab `Usage` ▷ `Test Images` and then click the button `Set Reference`, the match in the chosen image is set as reference position for alignment. Note that this is only possible if a match is detected in the image. If no reference image is chosen, the model image (page 218) will be used used as basis for the alignment (page 217) or rectfication (page 218).

□ Searching for the Object in a Test Image

When you click the button `Find Model` in the dialog `Test Images` of the tab `Usage`, the object is searched for in the currently selected test image; the result is displayed in the graphics window.

Please note that if the button is clicked for the first time or after you changed a model parameter, the internally stored model is actually created, which takes some time. If the model creation takes a long time (i.e., if you have chosen a very large angle (page 233) and scale range (page 234)), you may find it useful to reduce angle and scale range if possible or, if you are using shape-based matching, switch off the complete pregeneration (page 237) of the model.

You can also search for the object in the whole sequence of test images at once.

□ Searching for the Object in All Test Images

When you click the button `Detect All` inside the dialog `Test Images` of the tab `Usage`, the object is searched for in the complete sequence of test images that were loaded before. The results are displayed successively in the graphics window. This button also sets the maximum number of matches if it has not been set previously (i.e. was set to 0).

Please note that if the button is clicked for the first time or after you changed a model parameter, the internally stored model is actually created, which takes some time. If the model creation takes a long time (i.e., if you have chosen a very large angle (page 233) and scale range (page 234)), you may find it useful to switch off the complete pregeneration (page 237) of the model.

You can also search for the object in a single test image.

☐ Automatically Searching for the Object in the Test Images

If you check the box `Always Find` in the dialog `Test Images` of the tab `Usage` (also accessible via the menu item `Usage` ▷ `Test Images` ▷ `Show Test Image Settings`), the object is searched for automatically whenever you select a new test image.

Please note that if the matching process is started for the first time or after you changed a model parameter, the internally stored model is created, which takes some time. If the model creation takes a long time (i.e., if you have chosen a very large angle and scale range), you may find it useful to switch off the complete pregeneration (page 237) of the model.

☐ Selecting and Displaying a Test Image

You can select a test image by clicking with the left mouse button onto its number (index) or path in the text box of the dialog `Test Images` of the tab `Usage`. The selected image is automatically displayed in the graphics window of HDevelop.

If the checkbox labeled `Always Find` is checked, the matching process is started automatically on the selected test image; its result is displayed in the graphics window.

If you want to redisplay the selected test image in a later step, e.g., after you displayed the model image (page 227) again, you can also display it via the menu item `Usage` ▷ `Test Images` ▷ `Display Selected Test Image` without newly selecting it.

☐ Specifying the Number of Objects Visible in a Test Image

In the dialog `Test Images` (page 239) in the tab `Usage`, you can specify how many objects are visible in the current test image using the corresponding text box that appears when clicking onto the currently displayed number of visible objects in the text field of the currently selected test image. The default value is 1.

If you select the corresponding recognition mode in the dialog accessed via `Usage` ▷ `Go To Optimize Recognition Speed` (page 246), the specified numbers of visible objects are used when determining the recognition rate, i.e., the recognition rate is 100% when the sum of all objects found in the test images is equal to the sum of the specified numbers.

☐ Assuring the Matching Success

After loading (page 240) the test images, you can quickly test whether all objects are found successfully via the dialog `Inspect` ▷ `Determine Recognition Rate` (page 246). If the matching succeeds in all test images, i.e., if a recognition rate of 100% is reached, you can start to optimize the speed (page 220) of the matching process.

If the matching fails in one or more test images, proceed as follows:

- Open the dialog `Test Images` (page 239) in the tab `Usage`.
- Check the box `Always Find`.
- Step through the test images to determine the images where the matching fails.
- If an object is not found, check whether one of the following situations causes your problem:
  - Is the object crossing the image border, i.e., does it lie partially outside the test image? (**Solution for shape-based matching**)

    By default the objects must lie completely within the test image in order to be found. For shape-based matching, his behavior can be changed in the dialog `Advanced Use Parameters` (page 243) in the tab `Usage` via the parameter `Shape models may cross the image border` (page 245).
  - Is the Matching Assistant too greedy ? (**Solution for shape-based matching and deformable matching**)

    By default, the Matching Assistant uses a fast search heuristic which might *overlook* an object. Therefore, try reducing the corresponding parameter `Greediness` (page 243) in the dialog `Advanced Use Parameters` (page 243) manually or automatically via the menu item `Inspect` ▷ `Optimize Recognition Speed` (page 246).

- Is the object partially occluded? (**Solution for shape-based matching, deformable matching, correlation-based matching and descriptor-based matching**)

  If the object is to be recognized in this state nevertheless, try reducing the parameter `Minimum Score` in the dialog `Standard Use Parameters` in the tab `Usage` manually or automatically via the menu item `Inspect` ▷ `Optimize Recognition Speed` (page 246).

- Has the object a low contrast? (**Solution for shape-based matching and deformable matching**) If the object is to be recognized in this state nevertheless, try reducing the parameter `Contrast Low` (page 238) or `Contrast` (page 238), respectively, in the dialog `Standard Model Parameters` (page 231) in the tab `Parameters`. That way, the model can later be detected more easily. If the values are, however, chosen to low, too many edges are detected.

- Do multiple objects overlap? (**Solution for shape-based matching, deformable matching and correlation-based matching**) If the objects are to be recognized in this state nevertheless, try decreasing the `Max. Overlap` (page 244) in the dialog `Advanced Use Parameters` in the tab `Usage`.

- If the object is found but not at the expected position or orientation check the following: (**Solution for shape-based matching, deformable matching and correlation-based matching**)
  - If multiple matches are found on one and the same object, decrease the `Maximum Overlap` (page 244) in the dialog `Advanced Use Parameters`.
  - If an almost symmetric object is found at the *wrong* orientation try reducing the parameters specifying the allowed range of rotation (page 233) in the dialog `Standard Model Parameters` (page 231) in the tab `Parameters`.

□ Specifying Standard Model Use Parameters

Via the menu item `Usage` ▷ `Standard Model Use Parameters`, you can specify

- the `Minimum Score` the object must have and
- the number of instances of the object that are searched for in an image (`Maximum Number of Matches`)

for **ALL matching methods**

Additionally, advanced search parameters can be specified via the menu item `Usage` ▷ `Advanced Model Use Parameters`.

If you are finished setting search parameters, continue to inspect (page 246) the results.

◇ The Search Parameter `Minimum Score`

When comparing a region in a test image with the *model* (page 218), the Matching Assistant calculates a measure of similarity, the so-called *score* (page 218), which ranges between 0 (no similarity) and 1 (perfect similarity). With the parameter `Minimum Score` you can specify a minimum score that a match must reach.

Graphically speaking, the parameter specifies how much of the object, i.e., how many of the model points, must be visible. A part of the object may be *invisible* not only because it is occluded, but also if its contrast is lower than the selected minimum contrast value (page 238) or has the wrong polarity (page 237). A further cause for a too low score could be a (too) large angle step size (page 236).

The larger the value is chosen, the faster the search will be, because candidate matches can be discarded earlier. Therefore, this parameter can be optimized easily: Starting from the maximum value, reduce the value until the object is found in all *test images* (page 218); in fact, this method is used by the Matching Assistant itself when you start the optimization via the menu item `Usage` ▷ `Optimize Recognition Speed` (page 246).

Choosing small values may cause the program to search for quite a while. In such a case we recommend entering a larger value.

Please note that by default the objects must lie completely within the test images in order to be found.

For **shape-based matching**, this behavior can be changed via the parameter `Shape models may cross the image border` (page 245) in the dialog accessed via the menu item `Usage` ▷ `Advanced Model Use Parameters`.

◇ The Search Parameter `Maximum Number of Matches`

The parameter `Maximum Number of Matches` specifies how many instances of the object are searched for in the image. Note that the parameter sets a maximum value, i.e., if more object instances are present in the image only the best instances of the specified number are displayed. If you specify the value 0, all found instances are displayed.

□ Specifying Advanced Model Use Parameters

Via the menu item `Usage` ▷ `Advanced Use Parameters` the tab `Usage` is opened and you can specify advanced parameters.

For **shape-based matching** the following parameters can be specified:

- the `Greediness` of the search algorithm,
- how much the objects may overlap (`Maximum Overlap`),
- the accuracy (`Subpixel`) of the calculated position, orientation, and scale,
- how far the positions of the found edges may differ (`Max Deformation`) from the model edges positions,
- the lowest pyramid level `Last Pyramid Level` to which the found matches are tracked, and
- the time (in milliseconds) after which the detection of a model within an image is aborted (`Timeout` (page 245)), and also
- whether objects that lie partially outside the image (`Shape model may cross the image border` (page 245)) should be searched for.

For **correlation-based matching** the following parameters can be specified:

- how much the objects may overlap (`Maximum Overlap`),
- the accuracy (`Subpixel`) of the calculated position, orientation, and scale,
- the lowest pyramid level `Last Pyramid Level` to which the found matches are tracked, and
- the time (in milliseconds) after which the detection of a model within an image is aborted (`Timeout` (page 245)).

For **deformable matching** the following parameters can be specified:

- the `Greediness` of the search algorithm,
- how much the objects may overlap (`Maximum Overlap`),
- the accuracy (`Subpixel`) of the calculated position, orientation, and scale, and also
- the lowest pyramid level `Last Pyramid Level` to which the found matches are tracked.

For **descriptor-based matching** the following parameters can be specified:

- the `Score Type` (page 245),
- the `Descriptor Min. Score` (page 245),
- the `Guided Matching` (page 246).

Once you are finished setting search parameters, continue to inspect (page 246) the results.

◇ The Search Parameter `Greediness` (**Shape-based Matching, Deformable Matching**)

The parameter `Greediness` influences the search algorithm used by the Matching Assistant. It ranges between 0 and 1. If you select a low value, the search is thorough but relatively slow. The higher the value, the faster the search algorithm becomes, but at the cost of thoroughness, i.e., an object might not be found even though it is visible in the image.

This parameter can be optimized easily: Starting from the value 0, increase the value until the matching fails in a test image, and then use the last value for which the object is found; in fact, this method is used by the Matching

Assistant itself when you start the optimization via the menu item Usage ▷ Optimize Recognition Speed (page 246).

◇ The Search Parameter Maximum Overlap (**Shape-based Matching, Correlation-based Matching, Deformable Matching**)

The parameter Maximum Overlap specifies how much two matches may overlap in the image; its value ranges between 0 and 1. Especially in the case of an almost symmetric object the allowed overlap should be reduced to prevent multiple matches on the same object.

◇ The Search Parameter Subpixel (**Shape-based Matching, Correlation-based Matching, Deformable Matching**)

The parameter Subpixel allows you to select the accuracy with which the position, orientation, and scale are calculated. If you select the value 'none', the position is determined only with pixel accuracy, and the accuracy of the orientation and scale is equal to the angle step size (page 236) and scale step size (page 236), respectively.

If you select the value 'interpolation', the Matching Assistant examines the matching scores at the neighboring positions, angles, and scales around the best match and determines the maximum by interpolation. Using this method, the position is therefore estimated with sub-pixel accuracy. The accuracy of the estimated orientation and scale depends on the size of the object: The larger the size, the more accurately the orientation and scale can be determined. For example, if the maximum distance between the center and the boundary is 100 pixels, the orientation is determined with an accuracy of about 0.1°.

Because the interpolation is very fast, you can select 'interpolation' in most applications.

When you choose the values 'least_squares', 'least_squares_high', or 'least_squares_very_high', a least-squares approximation is used instead of an interpolation, resulting in an even higher accuracy. However, this method requires additional computation time.

◇ The Search Parameter Max Deformation (**Shape-based Matching**)

The parameter Max Deformation enables you to allow a certain deviation (in pixels) of the found edges from the model edges. Note that higher values for Max Deformation often result in an increased runtime. Furthermore, the higher the Max Deformation value is chosen, the higher is the risk of finding wrong model instances. Both problems mainly arise when searching for small objects or for objects with fine structures. This is because such kinds of objects lose their characteristic shape which is important for a robust search if the deformations are to extreme. Also note that for higher deformations the accuracy of partially occluded objects might decrease if clutter is present close to the object. Consequently, the value for Max Deformation should be chosen as small as possible and only as high as necessary.

For more information, please refer to the reference documentation of the HALCON operator find_shape_model.

◇ The Search Parameter Last Pyramid Level (**Shape-based Matching, Correlation-based Matching, Deformable Matching**)

With the parameter Last Pyramid Level you can select the lowest pyramid level to which the found matches are tracked. For example, when selecting the value 2, the matching starts at the highest pyramid level and tracks the matches to the second lowest pyramid level (the lowest pyramid level is denoted by a value of 1).

This mechanism can be used to speed up the matching. It should be noted, however, that in general the accuracy of the extracted position, orientation, and scale is lower in this mode than in the normal mode, in which the matches are tracked to the lowest pyramid level. Hence, if a high accuracy is desired, the parameter Subpixel should be set to at least 'least_squares'.

Note that if the lowest pyramid level to use is chosen too large, it may happen that the desired accuracy cannot be achieved, or that wrong instances of the model are found because the model is not specific enough on the higher pyramid levels to facilitate a reliable selection of the correct instance of the model. In this case, the lowest pyramid level to use must be set to a smaller value.

◇ The Parameter `Timeout` (**Shape-based Matching, Correlation-based Matching**)

Setting a timeout can improve the overall speed of your application as the program will then stop searching for a model within an image after a certain time. To use the `Timeout` function, activate `Enable` on the right side of the tab. Then choose the time in milliseconds after which the search for a model shall be aborted. Activating the `Timeout` is especially useful in cases where a maximum cycle time has to be ensured. Be aware that the runtime of the search increases by up to 10 percent with activated `Timeout`.

For more information, please refer to the reference documentation of the HALCON operator `set_shape_model_param`.

◇ The Search Parameter `Increased tolerance mode` (**Shape-based Matching**)

The `Increased tolerance mode` allows you to increase the searching tolerance and, therefore, also increase speed. The matching is then less robust and accurate but faster and might find deformed and defocused objects in an image.

The first value of `NumLevels` determines the number of pyramid levels, the second level determines at which pyramid level the search for the model will stop. If the `Increased tolerance mode` is enabled, the second value of `NumLevels` will be negative. If this value is negative, the search for the model will stop on the lowest pyramid level where a model is detected.

Therefore, the increased tolerance mode is useful

- if images are sometimes defocused but still the object has to be found.
- if it is acceptable that slightly deformed objects may also be found.
- if in your application matching is not used to distinguish between two quite similar looking object types.
- if the image quality is not very high but also cannot be further improved.

◇ The Search Parameter `Shape models may cross the image border` (**Shape-based Matching**)

With the parameter `Shape models may cross the image border`, you can specify whether shape models that cross the image border, i.e., that lie partially outside the test images, should be searched for.

If you switch off the check box `Shape models may cross the image border` the shape model will only be searched for within those parts of the test images in which the shape model completely lies within the image.

If you switch on the check box `Shape models may cross the image border` the shape model will be searched for in all positions in which the model additionally lies partially outside the test images, i.e., in which the shape model extends beyond the image border. Here, points lying outside the image are regarded as being occluded, i.e., they lower the score. This should be taken into account while selecting the `Minimum Score` (page 242). Please note that the runtime of the search will increase in this mode.

◇ The Search Parameter `Score Type`

`Score Type` defines the type of score to be evaluated. You have two options to choose from:

- `inlier_ratio` and
- `num_points`

`inlier_ratio` defines the ratio of the number of point correspondences to the number of model points. Although this value can have values of from 0.0 to 1.0, it is rather unlikely that this ratio can reach 1.0. Yet, objects having inlier ratio less than 0.1, should be disregarded.

`num_points` defines the number of point correspondences per instance. An object instance should be considered good, if it has 10 or more point correspondences with the model. Fewer points are insufficient, because any random 4 point correspondences define a mathematically correct homography between two images.

◇ The Search Parameter `Descriptor Min. Score`

`Decriptor Min. Score` is the minimal classifier score for an interest point to be regarded as a potential match. The score function is between 0.0 and 1.0, but typically only values between 0.0 and 0.1 make sense (default =

0.0). Increasing the value for Min. Score can increase the detection speed significantly. Note, however, that using Descriptor Min. Score might have negative effect on the robustness of the detection process, especially when only few points can be found.

◇ The Search Parameter Guided Matching

Enable Guided Matching to use an already approximately known homography to 'guide' the matching process, which enhances the accuracy of the object recognition. Note that it increases the computational costs up to 10% in some cases.

□ Optimizing the Recognition Speed

When you select the menu item Usage ▷ Optimize Recognition Speed or click either the corresponding button in the tool bar or the button Run Optimization in the dialog Optimize Recognition Speed of the tab Usage, the Matching Assistant automatically determines values for the parameters Minimum Score (page 242) and Greediness (page 243) to optimize the recognition speed. The speed is calculated as the average recognition speed over all test images. You can interrupt this process by clicking the button labelled Stop; please note however, that this event is processed only after the current search has finished.

The two parameters are optimized as follows: At the beginning, the Greediness is set to 0 and the Minimum Score to 1. Then, the minimum score is decreased until the matching succeeds in all test images, i.e., until the recognition rate is 100%. Now, the Greediness is increased as long as the matching succeeds. This process is repeated until the optimum parameters are found. You can lower the threshold of acceptance for the recognition rate manually using the corresponding slider or text box at the bottom of the dialog.

The Matching Assistant then displays the optimal Minimum Score and Greediness and the reached recognition time. It automatically enters the parameter values in the dialogs Usage ▷ Standard Model Use Parameters (page 242) and Usage ▷ Advanced Model Use Parameters (page 243), respectively.

If a test image can contain more than one object, the term 'recognition rate' is ambiguous. Therefore, you can choose between three recognition modes:

- In each test image, at least one object is expected. The recognition rate is calculated as the percentage of test images which fulfill this condition, i.e., it is 100% if in all test images at least one object is found.
- In each test image, as many objects are expected as specified in the parameter Maximum Number of Matches (page 243) in the dialog accessed via Usage ▷ Standard Model Use Parameters (page 242). The recognition rate is calculated as the relation of found objects to the sum of expected objects over all images, i.e., it is 100% if in all test images (at least) Maximum Number of Matches objects are found.
- In each test image, as many objects are expected as specified manually (page 241) in the dialog Test Images (page 239) of the tab Usage. The recognition rate is calculated as the relation of found objects to the sum of expected objects over all images, i.e., it is 100% if in each image exactly as many objects are found as specified.

  Note that if you select Maximum Number of Matches = 0 and by mistake specify a lower number of visible objects than are actually present in a test image, a recognition rate = 100% results in a completely confused optimization algorithm. You may handle this case by selecting the condition >= 100% for the recognition rate.

7.3.3.6 The Menu and Tab Inspect

Via the menu Inspect you can determine the recognition rate and the pose bounds of the object for the used set of test images. When selecting Determine Recognition Rate via the menu Inspect, the tab Inspect is opened and the recognition rate is automatically determined. Alternatively, you can directly open the tab and select the button Run. Inside the tab, you can also specify the maximum number (page 243) of object instances the Matching Assistant should search for.

□ Determining the Recognition Rate

With the menu item Inspect ▷ Determine Recognition Rate or when you click either the corresponding button in the tool bar or the button Run in the tab Inspect, the Matching Assistant determines the recognition rate

by searching for the object in all loaded test images. You can interrupt this process by clicking the button labelled Stop; please note however, that this event is processed only after the current search has finished.

The Matching Assistant then displays at Recognition Rate the recognition rate calculated for different criteria and at Statistics the minimum and maximum *score* (page 218), as well as the minimum and maximum matching time. It also calculates the extent between minimum and maximum score and time, respectively.

Depending on what you have chosen on the Usage tab as recognition mode under Optimize Recognition Speed (page 246) and as Standard Use Parameters (page 242), three results can be viewed:

- In each test image, at least one object is expected. The recognition rate is calculated as the percentage of test images which fulfill this condition.
- In each test image, as many objects are expected as specified in the parameter Maximum Number of Matches (page 243) in the dialog accessed via Usage ▷ Standard Model Use Parameters (page 242). The recognition rate is calculated as the relation of found objects to the sum of expected objects over all images (in percent).

  Please keep in mind that if an image contains more objects than specified in the parameter Maximum Number of Matches, only the best Maximum Number of Matches instances are found! Therefore, if there are, e.g., two test images containing 1 and 3 objects, respectively, and you select Maximum Number of Matches = 2, the recognition rate will be 75%, i.e., 3 out of 4 expected objects.
- In each test image, as many objects are expected as specified manually (page 241) in the dialog accessed via Usage ▷ Standard Model Use Parameters (page 242). The recognition rate is calculated as the relation of found objects to the sum of expected objects over all images (in percent).

  Before using this mode, please check the value specified for the parameter Maximum Number of Matches (page 243): If it is not set to 0, it should not be smaller than the maximum number of objects visible in a test image; otherwise, the recognition rate will be below 100%.

  Note that if you select Maximum Number of Matches = 0 and by mistake specify a lower number of visible objects than actually present in a test image, a recognition rate = 100% causes problems with the algorithm. To further extend this line of thought: If for some reason in another test image an object is not found, the two errors cancel each other out, i.e., the recognition rate is 100%! Therefore, we recommend that you check whether the correct objects are found via the dialog Test Images (page 239) in the tab Usage.

□ Determining the Pose Bounds

When you click the button Run in the tab Inspect, besides the recognition rate (page 246) the Matching Assistant determines so-called pose bounds, i.e., the range of positions, orientations, and scales in which the object appears in the test images. You can interrupt this process by clicking the button labelled Stop; please note however, that this event is processed only after the current search has finished.

If the test images cover the whole ranges of allowed orientations and scales of the object you can use the calculated ranges to optimize the parameters Angle Extent (page 233), Start Angle (page 233), and the parameters for the scale range (page 234) in the dialog accessed via the menu item Parameters ▷ Standard Model Parameters (page 231); we recommend using slightly larger values to get accurate results at the boundaries of the ranges.

In a corresponding HALCON program you can use the calculated range of positions as a region of interest and thus further speed up the matching process.

7.3.3.7 The Menu and Tab Code Generation

Via the menu Code Generation you can

- open the dialog Options inside the tab Code Generation, where options for the code generation can be set,
- open the dialog Variable Names inside the tab Code Generation, where the names for the used variables can be specified,
- insert code to the program window of HDevelop according to the current settings of the Matching Assistant,
- release the generated code lines in the program window,

- delete the generated code lines from the program window as long as you did not released them, and
- open the dialog for the code preview (page 249) inside the tab `Code Generation`.

□ Specifying the Options for the Code Generation

Via the menu item `Code Generation` ▷ `Show Code Generation Options` you can open the dialog for determining the options for the code generation inside the tab `Code Generation`. The dialog consists of the following parts:

- radio buttons for selecting whether the model is created at run time from the *model image* (page 218) (`Create model from a model image at run time`) In this case, you can additionally select whether to use the model image and the *ROI* (page 218) that were specified inside the Matching Assistant (`Create ROI as in assistant`) or whether a new ROI has to be drawn at run time (`Generate Code for drawing an ROI at run time`). Alternatively, a radio button (`Load modified image:`) allows the selection of an already existing model,
- radio buttons that allow you to create a shape model from file or load a saved shape model file,
- a check box `Initialize Acquisition` to generate code for initializing an image acquisition device,
- a check box `Generate display code` to select whether to display the detected model instances in a loop,
- (available for shape-based matching and correlation-based matching) a check box `Generate affine transformation code` to generate code for an affine transformation that can be used for Alignment in the code of the Matching Assistant , which is useful if the following image processing step is not invariant against rotation, (e.g. OCR),
- a check box `Generate rectification code` that generates code for rectifying the image for detected models, which may be necessary if an object then appears distorted due to perspective projection,
- the button `Insert Code` to insert the code generated by the Matching Assistant into the Program window of HDevelop.

Note that for `Generate affine transformation code` as well as for `Generate rectification code` a reference image (page 240) has to be set on the `Usage` tab, otherwise the model image is used.

□ Specifying the Variables for the Code Generation

Via the menu item `Code Generation` ▷ `Show Variables for Code Generation` you can open the dialog for determining the variables used for the code generation inside the tab `Code Generation`. The dialog consists of several text fields for the individual variables needed for the code lines. The Matching Assistant automatically generates reasonable variable names, but you can change the individual names via the text fields.

□ Insert the Generated Code Lines

Via the menu item `Code Generation` ▷ `Insert Code` (also accessible as tool bar button or as button inside the tab `Code Generation`), you can insert the code that is generated according to the current settings of the Matching Assistant into the Program window.

□ Release the Generated Code Lines

Via the menu item `Code Generation` ▷ `Release Generated Code Lines` you can release the generated and inserted code lines. After releasing the code lines, all connections between the Matching Assistant and the program window of HDevelop are lost. That is, changes, e.g., the deletion of code lines, can then only be applied directly in the Program window and not from within the Matching Assistant anymore.

□ Delete the Generated Code Lines

Via the menu item `Code Generation` ▷ `Delete Generated Code Lines` you can delete the code lines that you have previously generated and inserted into the Program window of HDevelop from within the Matching Assistant. Note that this works only as long as you have not yet released the code lines.

□ Preview of the Generated Code Lines

Via the menu item `Code Generation` ▷ `Show Code Preview` you can open the dialog for the `Code Preview` in the tab `Code Generation`. Here, you have the possibility to, e.g., edit or replace individual operators of the code lines proposed by the Matching Assistant.

7.3.3.8 The Menu `Help`

Via the menu `Help` you can access the online documentation.

7.4 Measure Assistant

7.4.1 Introducing the Measure Assistant of HDevelop

The Measure Assistant of HDevelop is a front-end to HALCON's 1D measuring. Using the Measure Assistant you can, for example,

- easily set parameters and perform a visual inspection,
- quickly measure distances between edges along a straight line or circular arc,
- choose the results you need (page 252),
- perform a calibration to transfer your results to world coordinates (page 253), and
- if necessary, use advanced features to, e.g., optimize your measuring under difficult conditions with fuzzy measuring (page 255).

Using the Measure Assistant is simple: all you need is to setup the measure task to detect all relevant edges and add the resulting code to your application.

When you start a project for the first time, read "How to Use the Measure Assistant of HDevelop".

When looking for an overview over all Measure Assistant elements, please refer to the reference (page 258).

In this online help, the following special terms are used:

Edge An edge defines a gray-value transition that is either

- positive, which means that it changes from dark to bright, or
- negative, which means that it changes from bright to dark.

Edge Pair An edge pair always consists of two edges, one with a positive gray-value transition and one with a negative gray-value transition.

1D Measuring (also called 1D metrology or caliper) This is a fast and easy-to-use method for measuring objects in one dimension, along a line or circular arc. It is based on extracting edges perpendicular to a line of measurement and measuring, e.g., positions and the distances between them.

Region of Interest (ROI) This region specifies where to look for edges and the direction needed to determine the gray-value transition.

Fuzzy Measure Fuzzy Measuring allows a more detailed specification of the object parts to be measured by selecting edges according to different criteria. This can also be useful for images that contain unwanted gray-value edges due to, e.g., reflections on the surface ▷ `Fuzzy Measure` (page 255).

Fuzzy Score The so-called "fuzzy score" is part of fuzzy measuring and describes an evaluation of the quality of an edge. It is more than just an edge amplitude because it is the result of specifying different criteria and areas for edges as well as a tolerance. The score is calculated as a median value. A "good" edge is completely within a previously specified area for fuzzy measuring and has a fuzzy score of "1". If an edge is on the border of a fuzzy measure area, the `Fuzzy Score` describes how far the edge is still on the "good" side. The value can range from 0 to 1. Tolerances can be defined by the user. ▷ `Fuzzy Score` (page 256).

7.4.2 How to Use the Measure Assistant of HDevelop

The Measure Assistant allows you to easily choose the dimensions of an object you need in just a few steps:

- set up the measure task,
- extract edges,
- if desired, calibrate your system to obtain measuring results in world coordinates (page 253).

7.4.2.1 Set up the measure task

It is very easy to set up the measure task with the Measure Assistant:

- Choose your `Image Source` on the `Input` tab by either using the content of the `Graphics Window`, loading an `Image File`, or acquiring the image of an object using the Image Acquisition Assistant.
- Create an ROI using the buttons `Draw Line` or `Draw Circular Arc`, which you find in the toolbar of the Measure Assistant. Place the ROI perpendicular to the edges you want to measure.
- Now, that you have located the ROIs, you can extract edges within the chosen region(s) and therefore use the functionality of the `Edges` tab.

The `Input` tab offers two more options that you can use if necessary. If you already have calibration data available, you can load the data as described in the paragraph Calibration Source (page 253), otherwise just return to the `Input` tab later. A more advanced feature is changing the mode of the expected gray-value range which is only relevant if your image data uses more than 8 bits. Read more about this topic under `Expected Gray Value Range` (page 254).

In case your edge extraction is not as successful as expected, further improvements can be made using the fuzzy measuring function of the assistant (page 255).

□ Image Source

The `Input` tab lets you choose images from the following sources:

- If your image has already been opened in the `Graphics Window`, you can activate `Graphics Window` to continue working with the currently displayed image.
- You can load an image file by activating `Image File` or choosing `Load Image` from the menu `File` and the tool bar, respectively, and either typing in the image path or using the `Browse` button on the right to choose an image from a file.
- Another option is to activate the radio button for the Image Acquisition Assistant. Being connected to this assistant, you can acquire the image you want to measure in and you can even choose to work with a `Live` image. The same operations can be performed by clicking on the `Snap` or `Live` symbols in the tool bar.

Note that the measuring works on a single channel. For color RGB images, the red channel will be used. A color transformation can be performed with the operator `trans_from_rgb`.

Once you have loaded your image, you can continue to create an ROI. If your image data exceeds 8 bit, you can change the expected gray-value range (page 254). If you want to load calibration data or calibrate now, proceed to the section `Calibration Source` (page 253).

□ Create an ROI

Create an ROI by using the buttons `Draw Line` or `Draw Circular Arc` in the tool bar. The shape of the ROI should be chosen according to the shape of the object to be inspected.

After having chosen the ROI shape, you “draw” the ROI in the `Graphics Window` by keeping the left mouse button pressed. Then, modify the ROI until it has the correct shape. For linear ROIs you can modify the length by “dragging” the line’s end points or move it by dragging its center. For circular ROIs you can additionally modify the radius by dragging one of the four circle points that are located at 0, 90, 180, and 270 degrees. Once you are

finished, click the right mouse button, to confirm your choice. The ROI will already display edges if those can be detected with default parameters.

Remember that the ROI should run perpendicular to the edges you want to measure.

If you have previously prepared and saved an ROI, you can reuse it by choosing `Load ROI`.

In order to delete one or more ROIs you can either mark them and just press the delete button on your keybord or click on the tool bar buttons for `Delete Selected ROI Item` or `Delete All ROIs`.

You can view and edit the ROI data, which includes all the data about the exact position of the ROI(s), using the tool bar button `View ROI Data`. Editing an ROI by changing the ROI data is useful if an ROI should be modified more precisely than is possible by drawing in the graphics window.

Once you are satisfied with the shape of your ROI, proceed to the edge extraction on the `Edges` tab.

7.4.2.2 Extract Edges

After you have prepared the measure task (page 250), you continue to choose parameters on the `Edges` tab such that you can detect the edges between which you want to measure:

- On the tab `Edge` under `Edge Extraction` you can specify the parameters that are used to extract edges.
- `Edge Selection` allows you to group edges to pairs or choose edges with certain features.
- In order to improve the visibility of your ROI and edges, you can change the display parameters.

Now, you can select edges you want to measure and afterwards proceed to the `Results` tab to view your measuring results.

Open the `Line Profile` window (page 180) to inspect edges along an ROI. If necessary, i.e., if your edge extraction is not as successful as expected, you can refine the determination of the edges you need for measuring by using the `Fuzzy Measure` (page 255) option you find on the `Fuzzy` tab. Fuzzy measuring allows you to specify certain ranges within which edges are labeled “good”.

□ Edge Extraction

You can optimize edge extraction by adapting the following parameters on the `Edges` tab:

- With `Min. Edge Amplitude` you can specify how strong an edge must be to be selected. Adapt this parameter such that only your desired edges are selected. Note that very small values should only be used with high quality images, otherwise false edges might be caused by noise.
- `Smoothing` is helpful to reduce noise and therefore the detection of false edges. Please note, however, that smoothing distorts the edge profiles, i.e, edges are detected at a slightly wrong positions and the accuracy decreases. When using the line profile tool (page 180), smoothing can either be applied with the Measure Assistant or it can be performed with the smoothing slider within the line profile window. The smoothing values are immediately transfered to the Measure Assistant and the line profile window, respectively.
- With `ROI Width` you can specify how many pixels alongside, i.e., perpendicular, to the line or arc of measuring are used to detect edges. A larger value helps to reduce noise; however, if the edges themselves are not perpendicular to the line or arc of measure you must choose a smaller value, otherwise the edge will “loose its strength”. The general rule here is to choose the ROI always perpendicular and as wide as possible. This will lead to both a high precision and accuracy. If changing the ROI width leads to an edge length that is inconvenient for the display, e.g., because it is very short or very long, you can choose another length for the display instead of the ROI width under `Display Parameters`.
- With the `Interpolation Method` you influence the accuracy of the edge extraction, but also the processing time. Processing time and accuracy both increase from `nearest_neighbor` over `bilinear` up to `bicubic`. Activating `Highest Accuracy` influences the way the interpolation is calculated, it takes a bit more processing time and also slightly improves accuracy.

If one or more edges cannot be found without further processing, or if false edges are detected, the `Line Profile` (page 180) which can be opened with the `View Line Profile` button on the upper right edge of the `Edges` tab can provide the missing details to solve the problem. In its display, the line profile shows a green line with a gray

top for each successfully detected edge that is above a certain level to qualify as an edge. Potential edges that do not qualify as edges are displayed as vertical red lines. It may also be helpful to view the line profile along an ROI while changing parameters for edge detection, to see exactly how your actions influence the line profile and therefore possibly change which edges are detected. This profile can consequently be a basis for the decision which steps should be taken next, i.e., what would be necessary to enhance weak edges or suppress false edges.

In order to select which edges or edge pairs from your current results are relevant, proceed to `Edge Selection`. If you are already satisfied with the detected edges, continue to the `Results` tab.

□ Edge Selection

`Edge Selection` allows you to specify what kind of edges you are looking for. First you decide whether you want to find single edges or edge pairs. Depending on your choice, you have different options for the drop-down menus `Transition` and `Position`. These options will be explained for edge pairs in the following paragraph and subsequently for single edges.

You can activate `Group Edges to Pairs` to find an even number of edges with alternating transitions. The parameter `Transition` lets you select which edges are chosen to resolve ambiguity: The default setting of `all` will always accept the first matching edge. The values `positive` and `negative` restrict the transition of the very first edge. The `*_strongest` variants will pick the strongest edge instead of the first edge in a sequence of edges with the same transition. `Position` lets you decide whether you are looking for `all` edge pairs, or just want to detect the `first` or `last` edge pair.

If you are looking for single edges (instead of edge pairs) you can choose with `Transition` whether you want edges with `all` transitions (i.e. `positive` and `negative`) or just `positive` or just `negative`. Using `Position`, you can then determine whether you are looking for `all` edges or just want to detect the `first` or the `last` edge.

Once you are satisfied with your selection, you can view your measuring results on the `Results` tab.

7.4.2.3 Display Parameters

Changing display parameters on the `Edges` tab may help you to make the ROI in your image better visible and therefore easier to work with.

You can change

- the `Region Color` and also activate `Show Region` to see the actual size of your region which may be convenient if you do not use the ROI width as edge length.
- the `Edge Color` and also activate `Use Shadows` if this is beneficial for the visibility.
- the `Edge Length`, which is the ROI width in default mode, to improve the visibility of the edges, and
- the `Line Width`.

Once you are satisfied with your detected edges as well as with the display parameters, continue to the `Results` tab.

7.4.3 Results

The `Results` tab lets you choose which results are relevant for your application and displays these results in a spreadsheet.

The tab consists of three parts:

- `Feature Selection`,
- `Feature Processing`, and
- `Edge Data`.

Once you are satisfied with the results, continue to the `Code Generation` tab to receive the code for your application. Otherwise, go back to adapt your edge extraction (page 251) or calibrate your system.

7.4.3.1 Feature Selection

Feature selection lets you choose the features that are relevant for your application. The results of those features are measured and displayed below. All features are activated by default. It is, however, useful to deactivate those which are not needed to improve performance and readability.

You can choose between:

- `Position`,
- `Amplitude`,
- `Distance`,
- `Pair Width`, and
- `Fuzzy Score`.

Once you have chosen your features, you can either continue to `Feature Processing` if you want your results transformed into world coordinates, or, proceed to `Edge Data` to view the results for each ROI.

7.4.3.2 Feature Processing

Here, you can decide whether you want to receive your results in world coordinates and if so, you can choose a suitable `Unit`. This step will be unavailable if you have not calibrated your setup. If you want to transform your measurement results into world coordinates, you should now choose a calibration source or calibrate your camera system and load the data into the Measure Assistant (page 199) and then go back to `Transform Image into World Coordinates` under `Feature Processing`.

☐ Calibration Source

If you need results in world coordinates, a calibration is necessary. The `Input` tab lets you load calibration data if available or opens the Calibration Assistant to calibrate live.

Choosing `Calibration Files` allows you to load calibration data from file which is useful if you have already performed a calibration for your application. You can either type in the path to the parameter names (*.cal) and the camera pose (*.dat) or use the `Browse` buttons next to the input boxes to load them from file.

Choosing Calibration Assistant (page 199) allows you to use calibration data from a calibration assistant that quickly guides you through a calibration. A new assistant will be opened unless there is already one available.

If you are just finished with your `Input` (page 250), continue by creating an ROI (page 250). If you have already extracted your edges, proceed to examine your results under `Edge Data`.

7.4.3.3 Edge Data

`Edge Data` displays the measurement results. In order to see the different results for multiple ROIs, choose the ROI you want to view the results of by clicking on the name of that ROI in the `Active ROI` field. The results can then be examined below.

If you want to use measurement results in other documents, you can simply mark entries, use Ctrl+C to copy those entries to the global clipboard and subsequently paste them into any other document.

Once you are satisfied with the measuring results, proceed to the `Code Generation` tab.

7.4.4 Code Generation

`Code Generation` produces the code that is necessary to perform the chosen measurement tasks within an HDevelop program. On the `Code Generation` tab you can choose between several options and change parameter names, which has a direct effect on the code that is generated.

First, you can choose to `Initialize Acquisition` or `Initialize Calibration` automatically in your code. Depending on whether you have previously used the Image Acquisition Assistant or the Calibration Assistant to

acquire images or get calibration data, respectively, you can now decide whether you want the generated code from these assistants integrated in your code. Such an automatic integration of code results, e.g., in the automatic opening of the framegrabber if Image Acquisition Assistant is activated. If code for image acquisition or calibration already exist, the checkboxes should be deactivated. If Image Acquisition Assistant or Calibration Assistant are not in use, `Initialize Acquisition` and `Initialize Calibration` are grayed out.

Under `General Options`, you can decide on the `Alignment Method`, i.e., whether an alignment is needed for your application. To learn more about this advanced setting, please refer to the description in the section Alignment (page 258).

You can furthermore change variable names for

- General parameters,
- ROI coordinates, and
- Measurement results.

Once you are finished changing parameters, click the `Insert Code` button under `Measuring` to generate the code.

Finally, integrate the code into your HDevelop program.

7.4.4.1 Change Variable Names

If desired, you can change the default variable names for general parameters, ROI coordinates and measurement results or replace them with your own variable names.

Once you are finished changing your variable names, proceed to the `Code Preview`.

7.4.4.2 Code Preview

Before clicking the `Insert` button to include your code into the program window, you can preview the code in the `Code Preview` table. You can now step through the table which consists of the columns

- `Insert Operator`, which shows the operator that will be inserted once you press the `Insert` button,
- `Procedure`, which shows the corresponding procedure,
- `Line`, referring to the line number within the code, and
- `Replace Operator` which shows previously generated code that will be replaced.

7.4.5 Advanced Measuring Tasks

This section deals with more complicated measuring tasks. Therefore, you might find this section useful

- if you need to handle images with more than 8 bit,
- if standard edge detection does not detect the edges in your image and you might want to learn more about fuzzy measuring, or
- if your object can appear shifted and/or rotated in the image (page 258).

7.4.5.1 Expected Gray Value Range

If your image data exceeds 8 bits, it might be useful to choose a minimum and maximum gray value and change the mode that handles the expected gray-value range. On the `Input` tab, there are three modes available that allow you to control how much adaption of the program to gray-value ranges is necessary:

- The default mode is the `adaptive` mode, which checks the gray values of an image and automatically adapts these values for each image. This mode is useful if the gray-value range is unknown or differs between images. The downside to this default setting is, that the highest value might differ from image to image which results in the fact that also the curves in the graphs that define the edges appear shifted which can be confusing.

- The `increasing` mode checks the values for the first image and keeps these values if the gray-value range of the following images is either the same or smaller. It only corrects the values for a wider range which is only a problem if values should be adapted for an image with data that is significantly smaller than the one of the previous images and the values on the graph are therefore so close together that it is impossible to distinguish the edges.
- The simplest mode is the `fixed` mode as it uses only the gray-value range that has been entered and does not adapt any values. This mode is a good choice if it is known that all images have the same gray-value range or deviations do not contain necessary information in those values that exceed the given gray-value range so that their variation of range can be disregarded. You can check your camera to see if 10, 12 or 14 bits are used and choose your values accordingly.

Depending on the mode you have chosen, the minimum and maximum values will differ. The gray-value range that is chosen here directly affects the `Min. Edge Amplitude` as well the graphs on the Fuzzy tab.

The `Reset` button allows you to set the values back to their default.

If you are not finished with your `Input` (page 250) yet, proceed to either add calibration images or perform a calibration, respectively (page 253), if you want to transform your results in to world coordinates, or continue to create an ROI (page 250) and then step to the next tab to extract edges (page 251).

7.4.5.2 Fuzzy Measuring

So far, an edge amplitude was used for choosing edges. This is, however, sometimes not sufficient. When, e.g., reflections are part of the image, it might be necessary to further specify the features of the edges that should be detected. Such features, like position, contrast, pair width or mean gray value can be selected and graded by using fuzzy measuring.

Fuzzy measuring is based on fuzzy logic and allows a more specific determination of edge selection by assigning a certain score to each edge that determines whether this edge is a member of a particular fuzzy set. For most applications, however, it is, not necessary to use fuzzy measuring because the general edge detection functions are sufficient to detect the right edges. If you want to learn more about fuzzy measuring, please refer to the Solution Guide III on 1D Measuring. All different fuzzy features are explained in the section “Features that Can Be Used to Control the Selection of Edges and Edge Pairs”.

In order to use the fuzzy measuring function of the Measure Assistant, you first have to enable it by activating the `Use Fuzzy Measure (Advanced)` checkbox on top of the `Fuzzy` tab.

You can then proceed to select the following fuzzy membership criteria:

These options apply to both edges and edge pairs:

- `Fuzzy Contrast`, evaluates the amplitude of the edges and
- `Fuzzy Edge Position`, lets you choose “good” positions.

When working with edge pairs, additionally the following criteria can be activated:

- `Fuzzy Pair Center Position` chooses edge pairs with a center of a certain position,
- `Fuzzy Pair Width` (page 257) chooses pairs of a certain width,
- `Fuzzy Pair Gray Mean` (page 257), selects pairs of a certain mean gray value.

`Fuzzy Threshold` and `Reference Pair Width` are two general settings that can be specified and will then be applied for all activated criteria. Both settings are grayed out until at least one criterion is activated.

The `Fuzzy Threshold` is a value between 0.1 and 1 that selects the minimum fuzzy score. Each active fuzzy set, i.e. the values added to a fuzzy membership criterion after enabling it, will be evaluated. The final `Fuzzy Score` is the geometric mean of the individual scores.

`Reference Pair Width` helps you to adapt your values to changes in the setup. More information on how to use `Reference Pair Width` and the `Normalize` function can be found in the section “Advanced Fuzzy Features” (page 258).

How to specify the values that are to be evaluated as “good” by fuzzy measuring is described in more detail below.

☐ Specify Good Values

Fuzzy measuring works with “good values”. To determine whether a value is “good”, all edges receive a score which is a number between 0 and 1, depending on your chosen tolerance. There are different possibilities how you can specify those values that lead to the score which can be viewed on the `Results` tab (page 252):

- Click into the `Values` list and type in the values that you want as “good values”.
- Another solution is to use the `Add Current` button and add values corresponding to the currently extracted edges. You can then inspect and, if necessary, edit each single value simply by clicking on it and modifying the number.

E.g. if you want to measure the distance between wires but due to reflection you get one wrong edge pair you can use `Fuzzy Pair Width` to specify the pair width. Enter the wire’s width into the `Values` list to detect the right edge. Alternatively, you can load the current values into the list and delete the wrong one.

Whatever solution you choose to obtain your values, you can always delete values by clicking on them in the `Values` list and then using the `Remove` button. If you want to start over, you simply click on the `Remove All` button to delete all values.

It is also important that you choose the tolerance which defines how far from your chosen good value an edge can be to be still classified as “good”.

In the graph beneath `Tolerance`, you can see the values corresponding to all extracted edges. These are displayed as little crosses and the curve that was defined by your good values and the allowed tolerance is displayed as well.

Once you have determined all relevant edges, continue to the `Results` tab (page 252).

☐ Fuzzy Contrast

Enable `Fuzzy Contrast` to choose edges with specific amplitudes. Then specify your “good” values as described in the section “Specify Good Values”.

Once all relevant edges have been found, continue to the `Results` tab (page 252).

☐ Fuzzy Edge Position

Enable `Fuzzy Edge Position` to choose edges of a certain position. Under subtype, you can determine the kind of edge position that is relevant for your application. You have the choice between:

- `position` which lets you define the position of your edges with the starting point of the ROI set to 0,
- `position_center` which lets you define the position of your edges with the center of the ROI set to 0,
- `position_end` enables you to choose values with the end point of the ROI set to 0,
- `position_first_edge` which defines the first edge as value 0, or
- `position_last_edge` which defines the last edge as value 0.

Please note the description about how to specify “good” values.

You can learn how to use the `Normalized` option in the paragraph “Advanced Fuzzy Features” (page 258).

Once all relevant edges have been found, continue to the `Results` tab (page 252).

☐ Fuzzy Pair Center Position

Enable `Fuzzy Pair Center Position` to choose edge pairs with a center of a certain position. Under subtype, you can determine the kind of pair center position that is relevant for your application. You have the choice between:

- `position_pair` which lets you define the position of your edge pairs with the starting point of the ROI set to 0,
- `position_pair_center` which lets you define the position of your edge pairs with the center of the ROI is set to 0,

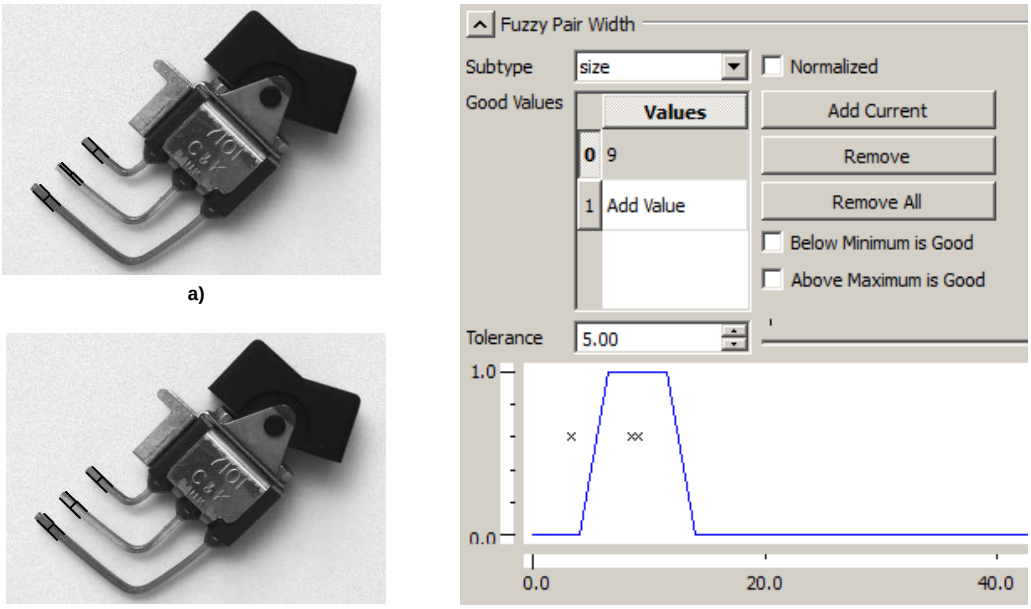


Figure 7.5: How to use fuzzy measuring to improve the edge detection in an image.

- `position_pair_end` which enables you to choose values with the end point of the ROI set to 0,
- `position_first_pair` which defines the first edge pair as value 0, and
- `position_last_pair` which defines the last edge pair as value 0.

Please note the description about how to specify “good” values (page 256).

You can learn how to use the `Normalized` option in the paragraph “Advanced Fuzzy Features”.

Once all relevant edges have been found, continue to the `Results` tab (page 252).

☐ Fuzzy Pair Width

`Fuzzy Pair Width` lets you select pairs of a certain width. You can choose between the subtypes

- `size` which evaluates the width of the edge pairs, i.e. the distance between the two edges of a pair,
- `size_diff` which evaluates the signed difference between the reference pair width and the actual width of the edge pairs, and
- `size_abs_diff` which evaluates the absolute difference between the desired reference pair width and the actual width of the edge pairs.

The images of a dip switch show how to use fuzzy measuring to improve edge detection. Due to surface reflections, one of the edge pairs is not detected properly. It is, however, known that the pair width is 9. Setting `Fuzzy Pair Width` to 9 results in the exclusion of the wrong edge and therefore also in the detection of all the right edge pairs.

Please note the description about how to specify “good” values (page 256).

You can learn how to use the `Normalized` option in the paragraph “Advanced Fuzzy Features”.

Once all relevant edges have been found, continue to the `Results` tab (page 252).

☐ Fuzzy Pair Gray Mean

`Fuzzy Pair Gray Mean` lets you select pairs of a certain mean gray value.

Please note the description about how to specify “good” values (page 256).

Once all relevant edges have been found, continue to the `Results` tab (page 252).

□ Advanced Fuzzy Features

The advanced features explained below enable you to adapt your fuzzy settings more easily to a different camera with a higher resolution, a different distance between object and camera, or in general an image that is larger or smaller.

The `Normalized` option can be activated for

- `Fuzzy Edge Position` (page 256),
- `Fuzzy Pair Center Position` (page 256), and
- `Fuzzy Pair Width` (page 257)

which are then converted to factors and multiplied with `Reference Pair Width`. The translation from regular to normalized values is automatically calculated when activating or deactivating the `Normalized` button. Therefore, corresponding graphs do not change. Choose the `Reference Pair Width` corresponding to the width of your reference pair. When changing your setup, adapt the `Reference Pair Width` to your new image size (e.g. if your image is now double the previous size, also double the reference `Reference Pair Width`).

7.4.5.3 Alignment

If the object in your image can appear shifted or rotated, an alignment, which can be chosen on the `Code Generation` tab under `Alignment Method` ▷ `Affine Transformation`, is necessary. An alignment allows you to automatically relocate your ROI in an image where the object occurs in a different position. Note, however, that the Measure Assistant does not perform the alignment by itself, but only generates code to align the measurements if a transformation matrix is available from other methods such as Template Matching. For general information, please read the following paragraph about alignment with the Measure Assistant. For more detailed information, please refer to the documentation on matching in the Solution Guide on Matching, Chapter 2.4 “Use the Results of Matching”.

To perform an alignment with the Measure Assistant, please follow the steps that are described in this paragraph. When generating code with an activated `Affine Transformation` option, the assistant produces code that requires a transformation matrix. To find the object within the image, you can perform a matching as explained in the Solution Guide on Matching or use the Matching Assistant to guide you through the matching task. By teaching a template, the object can be found in any position and orientation. After this successful matching, the affine transformation can be constructed with the operator `vector_angle_to_rigid` using the difference between the original and new model positions. This operator produces the homogeneous transformation matrix that describes the transformation from the old to the new position.

If the object in your image is only shifted but not rotated, the suitable `Alignment Method` would be `Translation Only` which also needs less processing time than `Affine Transformation`.

`No Alignment` is needed if the object always appears in the same position, or the intent is to find the object location by measuring.

You can now continue to change variable names (page 254) if necessary, or preview the generated code (page 254) before inserting it into your HDevelop program.

7.4.6 Measure Assistant Reference

The Measure Assistant consists of the following elements:

Pull-down menus:

- `File`
- `Measuring` (page 260)
- `Code Generation` (page 253)
- `Help` (page 261)

Tool bar with a selection of important buttons:

- `Load Assistant Settings` (page 260)

- Save Current Assistant Settings (page 260)
- Insert Code (page 261)
- Load Image
- Snap
- Live
- Draw Line
- Draw Circular Arc
- Line Profile (page 180)
- View ROI Data
- Delete Selected ROI Item
- Delete All ROIs (page 261)
- Help (page 261)

Tabs with the dialogs for most of the tasks that can be performed with the Measure Assistant:

- Input (page 261)
- Edges (page 262)
- Fuzzy (page 262)
- Results (page 262)
- Code Generation (page 262)

Furthermore it provides a status bar at the bottom in which messages are displayed. Please note that the status bar does not provide a scrolling mechanism; if the displayed message is to long, move the mouse over it, so that a tool tip displaying the full message pops up. Alternatively, if the message is only slighly larger than the status bar, you can also drag the left or right border of the Measure Assistant window to enlarge it.

Images are displayed in the graphics window of HDevelop.

7.4.6.1 The Menu File

Via the menu File you can

- load an image,
- load camera parameters,
- load assistant settings,
- save current assistant settings,
- close a dialog, and
- exit the assistant.

□ Load Image

You can load an image from a file by the menu item File ▷ Load Image or via the corresponding button of the tool bar. To acquire images from a camera, you can also use the Snap and Live buttons in the tool bar.

□ Load Camera Parameters

If you have saved your camera parameters before, you can load them by the menu item File ▷ Load Camera Parameters to use them for a calibration (page 199).

☐ Load Assistant Settings

If you have saved the settings of a former Measure Assistant session, you can load them again by the menu item `File` ▷ `Load Assistant Settings` or via the corresponding button of the tool bar.

☐ Save Current Assistant Settings

You can save the current settings of a Measure Assistant using the menu item `File` ▷ `Save Current Assistant Settings` or the corresponding button of the tool bar. Then you can load them again in a later session.

☐ Close Dialog

When closing the Measure Assistant dialog with the menu item `File` ▷ `Close Dialog` or the `X` in the top right corner of the window, the current settings are stored for the duration of the current HDevelop session. That is, as long as you do not exit HDevelop, you can again open the Measure Assistant with the same settings. In contrast to this, when you exit the Measure Assistant, the settings are lost also for the current HDevelop session.

☐ Exit Assistant

When you exit the Measure Assistant with the menu item `File` ▷ `Exit Assistant`, the assistant's dialog is closed and the current settings are lost unless you have stored them via the menu item `File` ▷ `Save Current Assistant Settings`. If you want to close the dialog but keep its settings for the current HDevelop session, you should use the menu item `Close Dialog` instead.

7.4.6.2 The Menu `Measuring`

Via the menu `Measuring` you can

- draw a linear ROI,
- draw a circular ROI,
- view the line profile (page 180),
- delete a selected ROI items,
- delete all ROIs,
- load a ROI from file, and
- save a ROI to file.

☐ Draw Line

To create a linear ROI, select the menu item `Measuring` ▷ `Draw Line` (also accessible as tool bar button). For more information about how to draw a linear ROI, please refer to the section "Create ROI" (page 250). You can also check the ROI data via the tool bar button `View ROI Data` (read more about ROI data in the section `Create ROI` (page 250).

☐ Draw Circular Arc

To create a circular ROI, select the menu item `Measuring` ▷ `Draw Line` (also accessible as tool bar button). For more information about how to draw a circular ROI, please refer to the section "Create ROI" (page 250). You can also check the ROI data via the tool bar button `View ROI Data` (read more about ROI data in the section `Create ROI` (page 250).

☐ Delete Selected ROI Item

You can delete an ROI item via the menu item `Measuring` ▷ `Delete Selected ROI Item` or via the corresponding button of the tool bar.

□ Load ROI from File

Via the menu item `Measure` ▷ `Load ROI from File`, you can load a previously saved ROI from a file.

□ Save ROI to File

If you want to reuse an ROI, you can save it to a file via the menu item `Measure` ▷ `Save ROI to File`.

□ Delete All ROIs

You can delete all ROIs via the menu item `Measuring` ▷ `Delete All ROIs` or via the corresponding button of the tool bar.

7.4.6.3 The Menu `Code Generation`

Via the menu `Code Generation` you can

- insert code into the program window of HDevelop according to the current settings of the Measure Assistant,
- release generated code lines in the program window,
- delete generated code lines from the program window as long as you did not release them, and
- open the dialog for the code preview inside the tab `Code Generation`.

□ Insert the Generated Code Lines

Via the menu item `Code Generation` ▷ `Insert Code` (also accessible as tool bar button or as button inside the tab `Code Generation`), you can insert the code that is generated according to the current settings of the Measure Assistant into the program window.

□ Release the Generated Code Lines

Via the menu item `Code Generation` ▷ `Release Generated Code Lines` you can release the generated and inserted code lines. After releasing the code lines, all connections between the Measure Assistant and the program window of HDevelop are lost. That is, changes, e.g., the deletion of code lines, can only be applied directly in the program window and not from within the Measure Assistant any more.

□ Delete the Generated Code Lines

Via the menu item `Code Generation` ▷ `Delete Generated Code Lines` you can delete the code lines that you have previously generated and inserted into the program window of HDevelop from within the Measure Assistant. Note that this works only as long as you have not yet released the code lines.

□ Preview of the Generated Code Lines

Via the menu item `Code Generation` ▷ `Show Code Preview` you can open the dialog for the `Code Preview` in the tab `Code Generation`.

7.4.6.4 The Menu `Help`

Via the menu `Help` you can access the online documentation.

7.4.6.5 The Tab `Input`

The `Input` tab consists of the following subdivisions:

- `Image Source` (page 250)
- `Expected Gray Value Range` (page 254)
- `Calibration Source` (page 253)

7.4.6.6 The Tab Edges

The Edges tab includes:

- Edge Extraction (page 251)
- Edge Selection (page 252)
- Display Parameters (page 252)

7.4.6.7 The Tab Fuzzy

The Fuzzy tab includes the following subdivisions:

- General Options (page 255)
- Fuzzy Contrast (page 256)
- Fuzzy Edge Position (page 256)
- Fuzzy Pair Center Position (page 256)
- Fuzzy Pair Width (page 257)
- Fuzzy Pair Gray Mean (page 257)

7.4.6.8 The Tab Results

The Results tab consists of the following subdivisions:

- Feature Selection (page 253)
- Feature Processing (page 253)
- Edge Data (page 253)

7.4.6.9 The Tab Code Generation

The Code Generation tab includes the following subdivisions:

- Measuring (page 258)
- Variable Names (General) (page 254)
- Variable Names (ROI coordinates) (page 254)
- Variable Names (Measurement results) (page 254)
- Code Preview (page 254)

7.5 OCR Assistant

7.5.1 Introducing the OCR Assistant of HDevelop

The OCR Assistant of HDevelop is a front-end to HALCON's optical character recognition. Using the OCR Assistant you can, for example,

- easily and quickly set parameters for optical character recognition (OCR) with the Quick Setup (page 264),
- segment text by choosing parameters suitable for the characters appearance (page 265)
- choose a pretrained classifier or train your own classifier (page 268),
- choose the kind of results you need (page 272), and

- generate code for your OCR application (page 274).

Note, the assistant is currently limited to OCR presented in the Solution Guide I, chapter 'OCR' and does not include Deep OCR.

Using the OCR Assistant is simple: either choose the `Quick Setup` to load an image and perform an OCR by setting basic parameters or use the sections `Image Source` and `Region of Interest` (page 265) where you can also load a sample and mark the text that should be read with a rectangle. Then improve the segmentation (page 265) by adapting relevant parameters, choose a pretrained font or performing your own training (page 268) and finally add the resulting code to your application (page 274).

When looking for an overview of all OCR Assistant elements, please refer to the reference (page 275). The general process of an OCR application is visualized in figure 7.6. This figure shows how a sample is found in an image via segmentation and can be directly classified if an OCR classifier is available. The sample is then assigned to a certain class, a symbol. If no suitable classifier is available, samples can be added to a training file from which a classifier can be trained that can subsequently be used to classify a sample. The symbol class is typically equivalent to a simple character. Therefore, a sample that is assigned to a symbol class results in a certain character that is read.

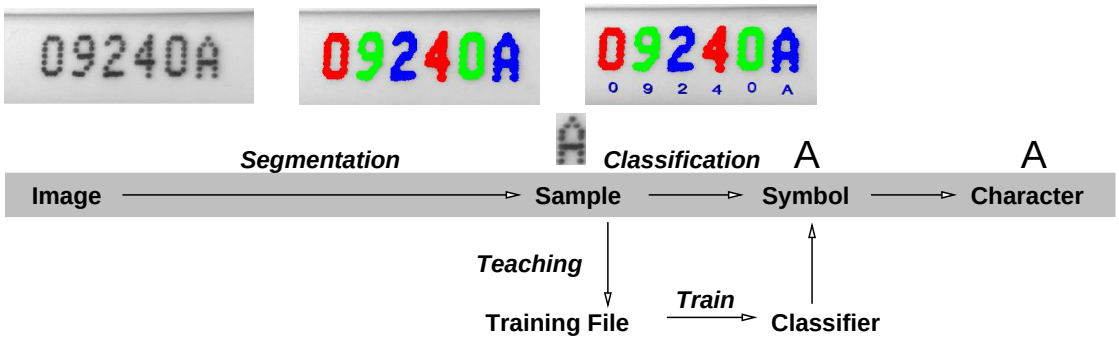


Figure 7.6: Process of an OCR application.

In this online help, the following special terms are used:

Optical Character Recognition (OCR) *Optical Character Recognition* is the technical term for reading, i.e., identifying symbols. In HALCON, OCR is defined as the task to assign an interpretation to regions of an image. These regions typically represent single *characters* and therefore we consider this as reading single symbols.

Sample A *sample* is the smallest individual object that is detected during *segmentation* and typically represents a simple *character*. It belongs to a certain class, a *symbol*.

Segmentation Both for the training and for the online reading process, *samples* must be extracted from the image. This step is called *segmentation*. This means that the OCR operators like `do_ocr_single_class_svm` do not search for the characters within a given region of interest, but expect a segmented region, which then will be assigned to a symbol class with a classifier.

Training The *training* consists of two important steps: First, for each *character* a number of samples is selected and stored in so-called training files. In the second step, these files are input to create a new *OCR classifier*. HALCON provides pretrained *OCR classifiers*, i.e., ready-to-use classifiers, which already solve many OCR applications. These *OCR classifiers* can be found in the subdirectory 'ocr' of the directory where you have installed HALCON.

Font A *font* describes a certain typeset, that differs from other fonts by certain features of the *characters*. A *classifier* can be trained for these special features of the font so that *characters* belonging to this *font* can be read successfully.

OCR Classifier An *OCR classifier* is trained to classify a certain set of *characters*, defined by certain characteristics (e.g. Pharma.omc). HALCON provides you with a set of pretrained *OCR classifiers*, which are based on a large amount of training data from various application areas. These OCR classifiers for *fonts* allow you to read text in documents, on pharmaceutical or industrial products, dot prints, and even handwritten numbers. Furthermore, HALCON includes pretrained *OCR classifiers* for fonts like OCR-A and OCR-B. You can also use your own classifiers and train an *OCR classifier* with HDevelop.

Symbol A *symbol* is a class one or more samples are assigned to with the help of an *OCR classifier*.

Character A *character* can be a single letter, number or special sign, like a hyphen. It belongs to a certain *Font*.

7.5.2 Setup

The OCR Assistant allows you to quickly and easily set up your OCR task in just a few steps:

- start easily with the `Quick Setup`,
- choose an image source,
- specify the position of a text with a region of interest.

7.5.2.1 Quick Setup

The `Quick Setup` is an easy way of setting up an OCR application. However, this preconfiguration of segmentation parameters is based on a basic description of the image content that does not take all factors into consideration that may influence a segmentation. For this reason, the `Quick Setup` alone may not be sufficient to segment and read characters in an image. Therefore, it is recommended and usually necessary to improve an OCR application by adapting parameters.

To get started, it is very easy to set up an OCR task with the OCR Assistant's `Quick Setup` in five steps:

1. **Load a sample image** by clicking the icon right beside the text or alternatively acquire a live image by selecting the Image Acquisition Assistant (page 194) under `Image Source` on the same tab.
2. **Mark the position of the text to be read** using a rectangle. The purpose of this step is to indicate the text geometry (like size and rotation). Use an axis-aligned rectangle for text in horizontal or vertical direction and a rotated rectangle for rotated text. Alternatively, set the rectangle via the `Region of Interest` dialog on the same tab. Allow for a small border. Note that if you are using the `Quick Setup` the rectangle must enclose the characters that should be read as precisely as possible and it has to include at least three characters.
3. Now that the region of the text is located, **enter the text that you expect to be read** in the text field. Remember to enter the text exactly as it is in the sample, including all separators, line breaks and correct capitalization. If the text contains large gaps these should be mimicked for a better recognition.
4. **Describe some basic properties of the text**. These properties are essential for a successful segmentation with a minimum number of parameter adaptations. Without this information, the samples in the image may not be segmented at all (e.g., for characters that are printed bright on dark background, the image has to be inverted internally). Check if characters are bright on dark background, composed of individual dots or if the text is structured, i.e. if letters, digits or separators appear at fixed positions. Furthermore, check whether the background is textured, noisy or cluttered.
5. Check if all information that was required so far is filled in correctly according to the features of the sample image and then **click the** `Apply Quick Setup` **button** when finished.

If the `Quick Setup` was successful, the found characters are read and the `Results` tab (page 272) is automatically opened. Otherwise, the segmentation tab opens and allows you to adapt parameters while simultaneously observing the segmentation results.

7.5.2.2 Image Source

The `Image Source` dialog on the `Setup` tab lets you choose images from different sources:

- If your image has already been opened in the `Graphics Window`, you can **activate** `Graphics Window` **to continue working with the currently displayed image**.

- You can **load an image file by activating** `Image File` or choosing `Load Image` from the menu `File` and the toolbar, respectively, and either typing in the image path or using the `Browse` button on the right to select an image from a file.
- Another option is to activate the radio button for the Image Acquisition Assistant (page 194). Being connected to this assistant, you can **acquire the image you want to use for OCR and you can even choose to work with a** `Live` **image**. The same operations can be performed by clicking on the `Snap` or `Live` symbols in the toolbar.

Note that the OCR works on a single channel image. For color RGB images, the red channel will be used. A color transformation can be performed with the operator `trans_from_rgb` later in the generated code.

Once you have loaded your image, you can continue to create an ROI.

7.5.2.3 Region of Interest

Create an ROI by using the buttons `Draw Axis-aligned Rectangle` or `Draw Rotated Rectangle` in the tool bar or in the drop-down menu `OCR`, respectively or by choosing a shape within the `Region of Interest` section of the `Setup` tab. The shape of the ROI should be chosen according to the shape of the object to be inspected.

After having chosen the ROI shape, you "draw" the ROI in the `Graphics Window` by keeping the left mouse button pressed. Then, modify the ROI until it has the correct shape. Once you are finished, click the right mouse button, to confirm your choice. To select a rotated rectangle with a correct reading direction, check that the arrow that is displayed within the rectangle corresponds to the reading direction of the characters. All characters that can be read with default parameters within the ROI are displayed immediately.

Note that the ROI should be chosen such that it also contains a small border around the characters to be recognized.

If you have previously prepared and saved an ROI, you can reuse it by selecting `Load ROI` from the `OCR` menu.

In order to delete one or more ROIs, you can either mark them and just press the delete button on your keybord or click on the buttons for `Delete Selected ROI Item` or `Delete All ROIs` on the `Setup` tab.

You can view and edit the ROI data, which includes all the data about the exact position of the ROI(s), using the tool bar button `View ROI Data`. Editing an ROI by changing the ROI data is useful if an ROI should be modified more precisely than is possible by drawing in the graphics window.

Once you are satisfied with the shape of your ROI, proceed to adjust parameters on the `Segmentation` tab.

7.5.3 Segmentation

This section describes how to set segmentation parameters for an OCR application. The tab `Segmentation` allows you to adjust several kinds of parameters for optical character recognition. The success of these parameter adaptions can be viewed immediately in the image after the `Setup` (page 264) has been completed (i.e., if at least one image has been loaded (page 264) and an ROI has been specified). Adjusting parameter settings is also recommended when using the `Quick Setup` (page 264) to achieve the highest possible character recognition quality.

Via the `Segmentation` tab, you can

- choose a special symbol appearance,
- set the parameters that define the symbol size,
- select special symbol shape features,
- set parameters for symbol fragmentation,
- specify text orientation range,
- set parameters for text layout (page 267),
- view content of the ROI in the inspection (page 267) field and apply corrections if necessary,
- and reset (page 268) all parameters.

7.5.3.1 Symbol Appearance

This section allows you to choose up to three options that specify the way the characters are printed.

Select

- `Light-On-Dark` whenever the gray values of the printed characters are brighter than the background gray values,
- `Dot Print` if the character print is composed of single dots (e.g. from matrix printers), and
- `Imprinted / Embossed` for print that is visible more by its relief rather than its pigment.

7.5.3.2 Symbol Size

Three characteristics of a symbols size, i.e., the

- maximum width of a single character,
- maximum height of a single character,
- and the so-called `Stroke Width`, defining the maximum thickness of the lines of a character

can be adjusted either by directly entering a number into the text field or by using the slider on the right.

Each value can be reset to the default value by clicking on the button on the right side of the slider.

7.5.3.3 Symbol Shape

This dialog allows you to choose up to three options that specify the shape of a symbol.

Select

- `Interpunctuation` for a text containing punctuation like, e.g., comma or period,
- `Separators` for a text containing separators such as hyphens or slashes, and
- `ALL UPPER CASE` for a text containing numbers and upper case letters only.

7.5.3.4 Symbol Fragmentation

Adjustments in this dialog help distinguish clutter from character fragments.

Set the `Min. Fragment Size` to define the minimum size of a connected structure that is not regarded as clutter either by entering a value in the text field or using the slider on the right. This value can be reset to the default value by clicking on the button on the right side of the slider.

`Symbol Fragmentation` also contains two checkboxes that should be deactivated if the settings are not helpful for a special application. By default, fragments are connected to form characters. Border fragments, i.e., fragments extending from outside into the region of interest, are eliminated.

7.5.3.5 Text Orientation

If the orientation of the text is unknown, i.e., it appears in variable orientations during the application, activate `Line Orientation`. Select an angle range by adjusting the minimum and maximum angle in degrees to automatically correct for text that is rotated within a given range. Note that not only the orientation but also the reading direction is also relevant for setting a value for `Text Orientation`. This is especially important for vertical text.

If text may appear slanted, activate `Symbol Slant` to automatically correct for text that is printed in slant or italics.

Each value can be reset to the default value by clicking on the corresponding buttons on the right side.

7.5.3.6 Text Layout

This dialog offers several options to make adjustments specifying a certain layout for text to be read. Those adjustments apply to characteristics of the text lines as they appear within the image.

For the text line layout, it is possible to

- set the `Baseline Tolerance`, i.e., the allowed amount of deviation from a perfect line, via text field or slider,
- set the `Max. Number of Lines`, and therefore limit the number of text lines that will be searched for, via text field or slider,
- use a text field next to `Line Structure` to set the maximum number of words and characters when searching for lines (e.g. '3 4-5' indicates a line consisting of three letters, then a space, then 4-5 letters), and
- activate the checkbox of `Eliminate Background Lines` if background lines exist, e.g. in certain forms or scanned documents, and shall be ignored.

The values of `Baseline Tolerance` and `Max. Number of Lines` can easily be reset to the default values via the buttons on the right side of the sliders.

7.5.3.7 Inspection

The `Inspection` section helps you to find a solution to a segmentation problem. Therefore, the embedded window shows only the content of the region of interest. By choosing an option from the combo box, intermediate results for different parameters as well as filtered images or symbol candidates can be viewed. By adapting parameters on this tab, changes in the segmentation process may be observed within this window which may be more helpful than just viewing the final result in the `Graphics Window`.

Choose from the combo box to view

- `Orientation Correction`,
- `Slant Correction`,
- `Filtered Image`,
- `Extracted Foreground`,
- `Best Symbol Candidates`,
- `Line Geometry`, and
- `Symbol Results`.

The following table can help you to solve segmentation problems by adapting the correct parameters.

Intermediate Step	Observation	Possible Solution
Orientation Correction	The text appears rotated.	Adapt the minimum and maximum angle for `Line Orientation` in the dialog `Text Orientation` or clip the ROI.
Slant Correction	The text appears slanted, e.g. printed in italics.	Adapt the minimum and maximum angle for `Symbol Slant` in the dialog `Text Orientation`.
Filtered Image	The characters are hard to detect in the filtered image.	There may be not enough contrast to segment the characters. Adapting the `Stroke Width` may help to solve this problem as well as it influences the filter size. Try to enhance the image quality.
Extracted Foreground	Not all characters are detected as foreground. One character is detected as two parts of the foreground.	Adapt the parameters in the dialogs `Symbol Size` and `Symbol Fragmentation` until they correspond to the appearance of the characters. Try to enhance the image quality if, e.g., the contrast is low.

Best Symbol Candidates	No symbol candidates or not all symbol candidates are found.	Note, that it is not necessary that all symbol candidates are found here as only the best candidates are displayed. Check if the parameters in the dialogs `Symbol Appearance`, `Size` and `Shape` correspond to the appearance of the characters (i.e., if the characters appear white on black background, the corresponding parameter has to be set).
Line Geometry	The extracted line does not match the text orientation.	Adapt the `Baseline Tolerance` in the dialog text layout and check if candidates are displayed.
Symbol Results	No results or incorrect results are found.	Check if the parameters in the dialogs `Symbol Appearance`, `Size` and `Shape` correspond to the appearance of the characters (i.e., if the characters are composed of single dots (dot print), the corresponding parameter has to be set).

Please note that if there are problems with the OCR, first of all the image quality should be checked. Only if there is nothing to enhance about the quality and therefore the training images are representative for the images in the application, parameters should be changed.

7.5.3.8 Reset

To simply set everything in this tab back to the default settings, which is recommended if a new OCR application is started with the OCR Assistant, just press the `Reset All` button. To reset single values on this tab, use the corresponding buttons on the right side of the selections.

7.5.4 OCR Classifier

This section deals with the classifier that is used for the OCR application and with its settings. The tab consists of the following sections:

- `OCR Classifier`
- `Teaching`
- `Training`
- `Basic Features`
- `Advanced Training Parameters and Features` (page 271)

7.5.4.1 Select or Save an OCR Classifier

When using a previously trained classifier, you can choose between either

- a `Pretrained Classifier` from one of the classifiers that are available for HALCON (see Solution Guide I, chapter 'OCR' for a more detailed description of the available pretrained OCR classifiers) or
- a previously trained classifier from file that can be selected via the `Browse` button next to the combo box for HALCON classifiers.

When deciding which pretrained classifier should be used, it may help to view details of the OCR classifier by clicking on the button with the magnifying lens located right of the selection for pretrained classifiers.

If a new classifier should be trained, a `Training File` has to be used as a basis for training a new classifier. Therefore, select the radio button `Training File` and continue by loading a training file with the `Browse` button next to the text field. To start the creation of a new training file, click on the button `New`. After a new training file has been created, samples can be added to the training file and then it has to be saved via the button `Save` before it can be used for training. To look at a currently or previously created training file and maybe edit it, open the `Training File Browser` (page 185) which provides an overview of the content of training files which can easily be modified.

7.5.4.2 Teaching

This section allows you to 'teach' characters which are displayed in the window right next to the text field.

Steps for teaching:

1. Use the text field to **enter characters that correspond to the segmented characters in the image**. As you enter the characters into the text field, the image of the corresponding character is displayed on the right side of the text field. The characters are highlighted in the Graphics Window as well.
2. After you have entered the characters, **press the button** `Add To Training Data`.
3. **Repeat this process for each sample image.**
4. After having added a representative number of samples, **click the button** `Train Now` **in the section** `Training`.
5. Once the classifier has been trained for the first time, **results of the OCR are available in the text field at the bottom of the** `Teaching` **dialog** to assist with further teaching.
6. If the results of the OCR are correct, **click the button** `Suggestion` **to quickly add the results to the text field during further teaching**.

If you want to view the results of the OCR classification, continue to the `Results` tab (page 272).

If you want to improve the OCR classification, you can change `Basic Features` that may improve a classification and train again. You can also select a different classifier and change parameters that influence single classifiers via the dialog `Advanced Training Parameters and Features` (page 271).

If you want to save data for archival purposes during training, you can activate `Save Ground Truth`. If activated, each time data is added to the training file, a copy of the source image, assistant settings, and teach data will be stored in a subdirectory next to the training file.

7.5.4.3 Training

The `Training Dialog` gives you the opportunity to perform a training via the button `Train Now` and is connected to the more interactive `Teaching` dialog above.

Other than starting the training, the status of the training can be viewed including

- the `Number of Samples` that were used in the selected training file,
- the name and path of the `Classifier`,
- and the status of the classifier, i.e., 'untrained'(not trained yet), 'training' (during training), 'trained' (after successful training), 'outdated' (new training data is available that has not been trained yet), 'read only' (classifier has been loaded) and 'failed' (the last training has not been successful).

7.5.4.4 Basic Features

This dialog enables you to choose several basic features that affect the classification.

First, you can adjust three features, that influence the interpolation of symbols to a fixed pattern size to provide a pixel feature of constant dimension that is used for classification. These features are:

- `Pattern Width`,
- `Pattern Height`,
- `Interpolation`

Furthermore, the three pattern features can be activated:

- `Gray Values`,

- Symbol Region, and
- Gradient.

And the following three symbol features can be activated:

- Ratio (page 271),
- Anisometry, and
- Convexity.

□ Pattern Width and Pattern Height

With the parameters Pattern Width and Pattern Height you set the fixed pattern size which is used for classification, i.e., the size to which characters are scaled. This pattern size influences the parameter Gray Values as this, if activated, uses the interpolated gray values. Depending on how much the characters are scaled, or if they are not scaled at all i.e., how much Pattern Width and Pattern Height differ from the size of the character in the image, a different Interpolation method might be suitable for the application.

Setting a bigger size generally helps to distinguish more characters. If the value is, however, chosen to big, overspecification may be a problem. In this case the amount of time necessary for the training and the time necessary for the recognition will also increase.

□ Interpolation

The parameter Interpolation lets you choose the interpolation mode, i.e., the adaptation of characters in the image to the pattern size. It also influences the parameter Gray Values as this, if activated, uses the interpolated pattern. The most suitable interpolation method largely depends on the values that were chosen as Pattern Width and Pattern Height, i.e., the scale factors.

Parameter	Effect	Usage
Constant (Default)	Bilinear interpolation in an image with a mean filter.	Recommended choice if characters are scaled down, but not by a large amount. This interpolation method achieves a high precision without requiring to much processing time.
Weighted	Bilinear interpolation in an image with a Gaussian filter.	Recommended choice if characters are scaled down by a large amount and a very high precision is required. Note that this interpolation method requires more processing time.
Bilinear	Interpolation, using the values of the 4 closest pixels in diagonal direction.	Can be used if characters are not scaled very much.
Nearest Neighbor	No interpolation is performed.	Fast but not very precise interpolation. Should only be used if the image is blurred.

For more detailed information on this parameter see also affine_trans_image.

□ Pattern Features Used for Classification

In this part of the dialog, three characteristics defining the classification of pattern features can be activated:

Activate

- Gray Values to use the gray values of the interpolated pattern,
- Symbol Region to use the symbol regions of the interpolated pattern, and
- Gradient to use gradient features instead of pixel patterns (8 directions sampled from a fixed 5x5 grid).

□ Symbol Features Used for Classification

In this part of the dialog, three characteristics defining the classification of symbol features can be activated:

Activate

- `Ratio` if the ratio (width to height) of a symbol should be used,
- `Anisometry` if the anisometry of the equivalent ellipse of the symbol shape should be used, and
- `Convexity` if the convexity of the symbol shape should be used.

7.5.4.5 Advanced Training Parameters and Features

This selection allows a precise specification of the classifier as well as of the classification parameters.

The following table can help you choose suitable classifier parameters.

Parameter	Method	Selection	Explanation
Classifier Type	All	MLP (Multilayer Perceptron), SVM (Support Vector Machine), kNN (k-Nearest Neighbor), Box (Hyperboxes); (default = MLP (if no classifier has been trained before) or the last classifier that has been trained)	Choose a method for the classification of the symbols. The last classifier that has been trained is loaded automatically. The settings for this classifier are preserved even if one of the pretrained classifiers is used in between.
Preprocessing	MLP, SVM	None, Normalization, Principal Components, Canonical Variates; (default = None)	Preprocessing can either normalize a value range of a feature or transform the feature space to decrease the number of dimensions. `Principal Components` and `Canonical Variates` can be used to reduce the amount of data without losing a large amount of information, while additionally optimizing the separability of the classes after the data reduction.
Components	MLP, SVM	Number of components (default = 10)	Dimension of the reduced feature space when using appropriate preprocessing. For preprocessing with principal components or canonical variates the length of the data is determined in *Components*. It is ignored if *Processing* was set to `None` or `Normalization`.
Hidden Units	MLP	Number of hidden units (1...150, default = Auto)	Number of neurons hidden in the middle layer of the MLP. The more input data you are using, the higher this value should be. In many cases, very small values of `Hidden Units` already lead to very good classification results. If `Hidden Units` is chosen too large, the MLP learns the training data very well, but does not return very good results for unknown data. In the 'auto' mode, the hidden value is estimated by a heuristic algorithm which is based on the number of characters. If the number of characters is really high, the number of hidden units might be estimated too high, which leads to a very slow training.
Iterations	MLP	The maximum number of iterations for training a MLP classifier (default = 200)	Select a sufficient number of iterations to create an MLP classifier that performs well in the subsequent application.
Weight Tolerance	MLP	MLP training continues while weights still change more than this value between iterations (default = 1)	Choose a realistic value for `Weight Tolerance`. If this value is chosen too small in the training, the training will take longer. If `Weight Tolerance` is set very high, the training will abort quickly and the classification results will not be very useful either.

Error Tolerance	MLP	MLP training continues while error still changes more than this value between iterations (default = 0.01)	Choose a realistic value for `Error Tolerance`. If this value is chosen too small in the training, the training will take longer. If `Error Tolerance` is set very high, everything is accepted which means that the results may not be useful either.
Mode	SVM	One vs. All, One vs. One (default = One vs. All)	The voting method used to combine the binary support vector machine classifiers. `One vs. All` creates a classifier where each class is compared to the rest of the training data. During testing, the class with the largest output is chosen. `One vs. One` creates a binary classifier between each single class. During testing a vote is cast and the class with the majority of the votes is selected.
Specialization (Gamma)	SVM	The Gamma parameter of the radial basis kernel function. 0.01, 0.02, 0.05, 0.1, 0.5 (default = 0.02)	It specifies the amount of influence of a support vector upon its surroundings. A big value means a small influence of surroundings, each training vector becomes a support vector and training/classification times grow. A too small value leads to few support vectors.
Regularization (Nu)	SVM	Regularization constant of an SVM (default 0.05)	The Nu parameter of the radial basis kernel function. One typical strategy is to select a small *Specialization (Gamma)* and *Regularization (Nu)* pair and consecutively increase the values as long as the recognition rate increases.
Number of Trees	kNN	Number of trees in a tree structure (default = 4)	The number of trees used by the kNN classifier. If more trees are used, the classification becomes more robust but the runtime also increases.

Other features that can be chosen are `pixel`, `pixel_invar`, `pixel_binary`, `gradient_8dir`, `projection_horizontal`, `projection_horizontal_invar`, `projection_vertical`, `projection_vertical_invar`, `ration`, `anisometry`, `width`, `height`, `zoom_factor`, `foreground`, `foreground_grid_9`, `foreground_grid_16`, `compactness`, `convexity`, `moments_region_2nd_invar`, `moments_region_2nd_rel_invar`, `moments_region_3rd_invar.`, `moments_central`, `moments_gray_plane`, `phi`, `num_connect`, `num_holes`, `cooc`, `num_runs`, and `chord_histo`.

For more information about the effect of these parameters, please refer to the reference documentation of the HALCON operators `create_ocr_class_mlp`, `create_ocr_class_svm`, `create_ocr_class_knn`, and `create_ocr_class_box`.

7.5.5 Results

The `Results` tab lets you select features concerning the classification results.

The tab consists of four parts:

- `Word Processing`,
- `Feature Selection`,
- `Display Parameters`, and
- `Results` (page 274).

Once you are satisfied with the results, proceed to the `Code Generation` (page 274) tab to receive the code for your application. Otherwise, go back to adapt your segmentation (page 265) or OCR classifier parameters (page 268).

7.5.5.1 Word Processing

To use word processing, activate Enable on the right side of this dialog. To correct the result of the OCR classification according to certain rules, the assistants' word processing offers two options:

- Use a Regular Expression that restricts the allowed text or
- load a Dictionary File to compare the text with allowed words (use the Browse button on the right to load a dictionary file).

Note that, in contrast to the operators do_ocr_word_*, the entire word is used in the Regular Expression automatically, i.e. the given expression is surrounded by '^...$'.

Furthermore, the maximum number of characters that will be corrected can be selected via Max. Corrections. For up to this number of characters if the word does not match the expression one or more alternatives (i.e., the second best class and so on) are considered, depending on the number set for Symbol Alternatives.

Note that if high numbers of Max. Corrections and Symbol Alternatives are chosen the number of possible corrections may be very, very high. Therefore, if the number of characters to be read is very high, the number of allowed corrections is internally clipped to 5, 3, or 1 if the alternatives multiplied with the number of characters are equal or greater than 30, 60 or 90, respectively.

The necessary corrections also influence the Score (the more corrections the lower the score) of the result that can be viewed in the Results tab.

For more information on how to use the word processing parameters, please refer to the reference documentation of the operators do_ocr_word_*.

7.5.5.2 Feature Selection

You can select one or more of the following features, depending on whether you have enabled Word Processing or not. The results for this feature are then displayed in the Results section.

Without word processing, you can enable

- Symbol to view the class determined for the symbol,
- Confidence to see with which confidence the class of the symbol was determined, and
- Alternatives to view which classes best match the symbol.

With word processing enabled, you can select

- Word to display the text that was read, including corrections,
- Uncorrected Word to see the text that was read without corrections, and
- Score to see how well the uncorrected text matched the expected format on a scale from 0 to 1.0 (where 0 is failure to correct and 1 is a match without corrections).

7.5.5.3 Display Parameters

To adapt the visualization, several display parameters can be adjusted.

The two combo boxes Symbol Color and Text Color let you select colors for text visualization. Via Symbol Color, a color, or color scheme that will be used to mark the extracted characters, can be selected. Text Color on the other hand lets you choose the color of the text that is read.

Use the dialog Display Font that opens when clicking on the font button to choose the font as well as further font settings for the text that is read

Several display functions can be turned on and off. If activated,

- Symbol Regions displays symbol regions,
- Text displays the text that is read,
- Bounding Boxes displays a bounding box for each character, and
- Shadows adds shadows to the displayed read text to enhance its visibility.

7.5.5.4 Results

The Results section consists of two windows. The first one shows the active ROI and also lets you choose between the different ROIs so you can view their results. The results of each ROI are displayed in the text field below. Columns will be displayed according to the features chosen in Feature Selection.

7.5.6 Code Generation

Code Generation produces the code that is necessary to perform the chosen OCR tasks within an HDevelop program. On the Code Generation tab you can choose between several options and change parameter names, which has a direct effect on the code that is generated.

Under General Options, you can decide on the Generation Mode, on the Alignment Method and whether code from the Image Acquisition Assistant should be integrated into the generated code.

You can furthermore change variable names and see a preview of the code that will be generated in Code Preview.

Once you are finished changing parameters, click the Insert Code button under on the right side of the General Options dialog to generate the code.

Finally, integrate the code into your HDevelop program.

7.5.6.1 General Options

In this dialog you can select the Generation Mode and the Alignment Method if required.

Generation Mode lets you select the task for which the code is generated, i.e.

- Text Reading or
- OCR Classifier Training.

You can also enable alignment, if it is needed for your application, i.e. if you want to use it to align the ROIs. To learn more about this setting, please refer to the description in the section Alignment.

Activate Initialize Acquisition if you have previously used the Image Acquisition Assistant (page 194) to acquire images and want to integrate the generated code from this assistant in your code. Such an automatic integration of code results, e.g., in the automatic opening of the framegrabber if Image Acquisition Assistant is activated. If code for image acquisition already exist, the checkboxes should be deactivated. If Image Acquisition Assistant is not in use, Initialize Acquisition is grayed out.

□ Alignment

If the characters in your image can appear shifted or rotated, an alignment, which can be chosen on the Code Generation tab under Alignment Method, is necessary. An alignment allows you to automatically relocate your ROI in an image where the object occurs in a different position. Note, however, that the OCR Assistant does not perform the alignment by itself, but only generates code to align the measurements if a transformation matrix is available from other methods such as Template Matching. For general information, please read the following paragraph about alignment with the OCR Assistant. For more detailed information, please refer to the documentation on matching in the Solution Guide on Matching, Chapter 2.4 “Use the Results of Matching”.

To perform an alignment with the OCR Assistant, please follow the description in this paragraph. When generating code with an activated Affine Transformation option, the assistant produces code that requires a transformation matrix. To find the characters within the image, you can perform a matching as explained in the Solution Guide on Matching or use the Matching Assistant (page 217) to guide you through the matching task and generate the alignment code. No Alignment is needed if the characters always appear in the same position. You can now continue to change variable names if necessary, or preview the generated code before inserting it into your HDevelop program.

7.5.6.2 Change Variable Names

If desired, you can change the default variable names or replace them with your own variable names.

Once you are finished changing your variable names, proceed to the Code Preview.

7.5.6.3 Code Preview

Before clicking the `Insert` button to include your code into the program window, you can preview the code in the `Code Preview` table. You can now step through the table which consists of the columns

- `Insert Operator`, which shows the operator that will be inserted once you press the `Insert` button,
- `Procedure`, which shows the corresponding procedure,
- `Line`, referring to the line number within the code, and
- `Replace Operator`, which shows previously generated code that will be replaced.

7.5.7 OCR Assistant Reference

The OCR Assistant consists of the following elements:

Pull-down menus:

- `File`
- `OCR` (page 277)
- `Code Generation` (page 279)
- `Help` (page 278)

Tool bar with a selection of important buttons:

- `Load Assistant Settings`
- `Save Current Assistant Settings` (page 277)
- `Insert Code` (page 278)
- `Load Image`
- `Snap`
- `Live`
- `Draw Axis-aligned Rectangle` (page 277)
- `Draw Rotated Rectangle` (page 277)
- `Load OCR Classifier`
- `New Training File`
- `Load Training File`
- `Save Training File`
- `Browse Training Files` (page 185)
- `Train Now` (page 277)
- `Help` (page 278)

Tabs with the dialogs for most of the tasks that can be performed with the Measure Assistant:

- `Setup` (page 264)
- `Segmentation` (page 265)
- `OCR Classifier` (page 268)
- `Results` (page 272)
- `Code Generation` (page 274)

Furthermore it provides a status bar at the bottom in which messages are displayed. Please note that the status bar does not provide a scrolling mechanism; if the displayed message is to long, move the mouse over it, so that a tool tip displaying the full message pops up. Alternatively, if the message is only slighly larger than the status bar, you can also drag the left or right border of the OCR Assistant window to enlarge it.

Images are displayed in the graphics window of HDevelop.

7.5.7.1 The Menu `File`

Via the menu `File` you can

- load an image,
- load an OCR classifier,
- create a new training file,
- load a training file,
- save a training file,
- load assistant settings,
- save current assistant settings,
- close a dialog, and
- exit the assistant.

☐ Load Image

You can load an image from a file by the menu item `File` ▷ `Load Image` or via the corresponding button of the tool bar. To acquire images from a camera, you can also use the `Snap` and `Live` buttons in the tool bar.

☐ Load OCR Classifier

If you have a pretrained classifier you want to use, you can load this classifier by the menu item `File` ▷ `Load OCR Classifier` or via the corresponding button of the toolbar.

☐ New Training File

You can create a new training file that can be used to train a classifier by the menu item `File` ▷ `Load OCR Classifier` or via the corresponding button of the toolbar. Alternatively, press the button `New` in the dialog `OCR Classifier` on the `OCR Classifier` tab. This automatically activates the radio button `Training File` which allows you to choose a file for the new training file or open the `Training File Browser` (page 185) which allows you to comfortably view and edit training files.

☐ Load Training File

If you have saved a training file of a former OCR Assistant session, you can load the training file again by the menu item `File` ▷ `Load Training File` or via the corresponding button of the toolbar.

☐ Save Training File

You can save a training file using the menu item `File` ▷ `Save Training File` or the corresponding button of the tool bar. Then you can load the training file again in a later session.

☐ Load Assistant Settings

If you have saved the settings of a former OCR Assistant session, you can load them again by the menu item `File` ▷ `Load Assistant Settings` or via the corresponding button of the tool bar.

□ Save Current Assistant Settings

You can save the current settings of an OCR Assistant using the menu item File ▷ Save Current Assistant Settings or the corresponding button of the tool bar. Then you can load them again in a later session.

□ Close Dialog

When closing the OCR Assistant dialog with the menu item File ▷ Close Dialog or the X in the top right corner of the window, the current settings are stored for the duration of the current HDevelop session. That is, as long as you do not exit HDevelop, you can again open the OCR Assistant with the same settings. In contrast to this, when you exit the OCR Assistant, the settings are lost also for the current HDevelop session.

□ Exit Assistant

When you exit the OCR Assistant with the menu item File ▷ Exit Assistant, the assistant's dialog is closed and the current settings are lost unless you have stored them via the menu item File ▷ Save Current Assistant Settings. If you want to close the dialog but keep its settings for the current HDevelop session, you should use the menu item Close Dialog instead.

7.5.7.2 The Menu OCR

Via the menu OCR you can

- draw an axis-aligned rectangle,
- draw a rotated rectangle,
- browse training files (page 185),
- train a classifier,
- delete a selected ROI items,
- delete all ROIs,
- load a ROI from file, and
- save a ROI to file.

□ Draw Axis-aligned or Rotated Rectangle

To create either an axis-aligned or a rotated rectangle, select the menu items OCR ▷ Draw Axis-aligned Rectangle or Draw Rotated Rectangle (also accessible as tool bar button), respectively. For more information about how to draw a rectangular ROI, please refer to the section "Create ROI" (page 265). You can also check the ROI data via the button View ROI Data in the Region Of Interest section on the Setup (page 264) tab (read more about ROI data in the section Create ROI (page 265).

□ Train Now

Train a classifier for OCR once you have loaded training data via the menu item Train Now or the corresponding button on the OCR Classifier tab in the section Training.

□ Delete one or all ROIs

You can delete an ROI item via the menu item OCR ▷ Delete Selected ROI Item via the corresponding button in the Region Of Interest section on the Setup tab (read more about ROI data in the section Create ROI (page 265). If you want to delete all ROI items, select Delete All ROIs.

□ Load ROI from File

Via the menu item OCR ▷ Load ROI from File, you can load a previously saved ROI from a file.

□ Save ROI to File

If you want to reuse an ROI, you can save it to a file via the menu item OCR ▷ Save ROI to File.

7.5.7.3 The Menu `Code Generation`

Via the menu `Code Generation` you can

- insert code into the program window of HDevelop according to the current settings of the OCR Assistant,
- release generated code lines in the program window,
- delete generated code lines from the program window as long as you did not release them, and
- open the dialog for the code preview inside the tab `Code Generation`.

☐ Insert the Generated Code Lines

Via the menu item `Code Generation` ▷ `Insert Code` (also accessible as tool bar button or as button inside the tab `Code Generation`), you can insert the code that is generated according to the current settings of the OCR Assistant into the program window.

☐ Release the Generated Code Lines

Via the menu item `Code Generation` ▷ `Release Generated Code Lines` you can release the generated and inserted code lines. After releasing the code lines, all connections between the OCR Assistant and the program window of HDevelop are lost. That is, changes, e.g., the deletion of code lines, can only be applied directly in the program window and not from within the OCR Assistant any more.

☐ Delete the Generated Code Lines

Via the menu item `Code Generation` ▷ `Delete Generated Code Lines` you can delete the code lines that you have previously generated and inserted (page 261) into the program window of HDevelop from within the OCR Assistant. Note that this works only as long as you have not yet released the code lines.

☐ Preview of the Generated Code Lines

Via the menu item `Code Generation` ▷ `Show Code Preview` you can open the dialog for the `Code Preview` in the tab `Code Generation`.

7.5.7.4 The Menu `Help`

Via the menu `Help` you can access the online documentation.

7.5.7.5 The Tab `Setup`

The `Setup` tab consists of the following subdivisions:

- `Quick Setup` (page 264)
- `Image Source` (page 264)
- `Region of Interest` (page 265)

7.5.7.6 The Tab `Segmentation`

The `Segmentation` tab includes:

- `Symbol Appearance` (page 266)
- `Symbol Size` (page 266)
- `Symbol Shape` (page 266)
- `Text Orientation` (page 266)
- `Text Layout` (page 267)
- `Inspection` (page 267)

7.5.7.7 The Tab `OCR Classifier`

The `OCR Classifier` tab includes the following subdivisions:

- `OCR Classifier` (page 268)
- `Teaching` (page 269)
- `Basic Features` (page 269)
- `Advanced Training Parameters and Features` (page 271)

7.5.7.8 The Tab `Results`

The `Results` tab consists of the following subdivisions:

- `Word Processing` (page 273)
- `Feature Selection` (page 273)
- `Display Parameters` (page 273)
- `Results` (page 274)

7.5.7.9 The Tab `Code Generation`

The `Code Generation` tab includes the following subdivisions:

- `General Options` (page 274)
- `Variable Names` (page 274)
- `Code Preview` (page 275)

Chapter 8

HDevelop Language

This chapter introduces the syntax and the semantics of the HDevelop language. In other words, it illustrates what you can enter into a parameter slot of an operator or procedure call. In the simplest case this is the name of a variable, but it might also be an arbitrary expression like `sqrt(A)`. Besides, control structures (like loops) and the semantics of parameter passing are described.

Note that the HALCON operators themselves are not described in this chapter. For this purpose refer to the HALCON reference manual. All program examples used in this chapter can also be found in the directory `%HALCONEXAMPLES%\hdevelop\Manuals\HDevelop`.

8.1 Basic Types of Parameters

HALCON distinguishes two kinds of data: control data (numbers, strings, or handles) and iconic data (images, regions, etc.) By further distinguishing *input* from *output parameters*, we get four different kinds of parameters. These four kinds always appear in the same order in the HDevelop parameter list. In the reference manual operator signatures are visualized in the following way:

```
operator (iconic input : iconic output : control input : control output)
```

As you see, iconic input objects are always passed first, followed by the iconic output objects. The iconic data is followed by the control data, and again, the input parameters succeed the output parameters.

Any of the four types of parameters may be empty. For example, the signature of `read_image` reads

```
read_image ( : Image : FileName : )
```

The operator `read_image` has one output parameter for iconic objects `Image` and one input control parameter `FileName`. The parameter types are reflected when entering operators in the operator window. The actual operator call displayed in the HDevelop program window is:

```
read_image (Image, 'Name')
```

The parameters are separated by commas. Input control parameters can either be variables, constants or expressions. An expression is evaluated *before* it is passed to a parameter that receives the result of the evaluation. Iconic parameters must be variables. Control output parameters must be variables, too, as they store the results of an operator evaluation.

8.2 Control Types and Constants

All non-iconic data is represented by so called *control data* (numbers, strings or handles) in HDevelop. The name is derived from their respective functions within HALCON operators where they *control* the behavior (the effect) of image processing (e.g., thresholds for a segmentation operator). Control parameters in HDevelop may contain arithmetic or logical operations. A control data item can be of one of the following data types: `integer`, `real`, `string`, `boolean`, and `handle`.

`integer` The data type `integer` is used under the same syntactical rules as in C. Integer numbers can be input in the standard decimal notation, in hexadecimal by prefixing the number with `0x`, and in octal by prefixing the number with `0` (zero).

For example:

```
4711
-123
0xbeef (48879 in decimal notation)
073421 (30481 in decimal notation)
```

Data items of type `integer` are converted to their machine-internal representations, that is the C type `long` (4 or 8 bytes).

`real` The data type `real` is used under the same syntactical rules as in C.

For example:

```
73.815
0.32214
.56
-17.32e-122
32E19
```

Data items of type `real` are converted to their machine-internal representations, that is the C type `double` (8 bytes).

`string` A `string` is a sequence of characters that is enclosed in single quotes (’). Special characters, like the line feed, are represented in the C-like notation, as you can see in table 8.1 (see the reference of the C language for comparison). You can enter arbitrary characters using the format `\x`*nn* where *nn* is a two-digit hexadecimal number, or using the format `\0`*nnn* where *nnn* is a three-digit octal number. Less digits may be used if the string is unambiguous. For example, a line feed may be specified as `\xa` unless the string continues with another hexadecimal digit (0-F).

For example: The string `Sobel's edge-filter` has to be specified as `'Sobel\'s edge-filter'`. A Windows directory path can be entered as `'C:\\Programs\\MVTec\\Halcon\\images'`

`boolean` The constants `true` and `false` belong to the data type `boolean`. The value `true` is internally represented by the number 1 and the value `false` by 0. This means, that in the expression `Val := true` the effective value of `Val` is set to 1. In general, every integer value other than 0 means `true`. Please note that some HALCON operators take logical values for input (e.g., `set_system`). In this case the HALCON operators expect string constants like `'true'` or `'false'` rather than the boolean values `true` or `false`.

`handle` Handles are references to complex data structures, e.g., a connection to an image acquisition device or a model for shape-based matching.

In addition to these general types, there are special constants and the type `tuple`, which are specific to HALCON or HDevelop, respectively. Since HALCON 12.0, HDevelop also supports the variable type `vector` (see section 8.6 on page 303).

constants There are constants for the return value (result state) of an operator. The constants can be used together with the operator `dev_error_var` and `dev_set_check`. These constants represent the normal return value of an operator, so-called *messages*. For errors no constants are available (there are plenty of error numbers internally, see the Extension Package Programmer’s Manual).

Meaning	Abbreviation	Notation
line feed	NL (LF)	\n
horizontal tabulator	HT	\t
vertical tabulator	VT	\v
backspace	BS	\b
carriage return	CR	\r
form feed	FF	\f
bell	BEL	\a
backslash	\	\\
single quote	’	\’
arbitrary character (hexadecimal)		\x*nn*
arbitrary character (octal)		\0*nnn*

Table 8.1: Surrogates for special characters.

Constant	Meaning	Value
H_MSG_TRUE	No error; for tests: (true)	2
H_MSG_FALSE	For tests: false	3
H_MSG_VOID	No result could be computed	4
H_MSG_FAIL	Operator did not succeed	5

Table 8.2: Return values for operators.

In table 8.2 all return messages can be found. Additionally, there are constants for the types of control data (see table 8.3). These can be compared to the result of a type operation to react to different types of control data (see section “Type Operations” on page 291).

tuple The control types are only used within the generic HDevelop type *tuple*. A tuple of length 1 is interpreted as an atomic value. A tuple may consist of several data items with *different* types. The standard representation of a tuple is a listing of its elements included into brackets. This is illustrated in figure 8.1 on page 284.

[] specifies the empty tuple. A tuple with just one element is to be considered as a special case, because it can either be specified in the tuple notation or as an atomic value: [55] defines the same constant as 55. Examples for tuples are:

```
[]
4711
0.815
'Text'
[16]
```

Constant	Meaning	Value
H_TYPE_INT	integer value	1
H_TYPE_REAL	real value	2
H_TYPE_STRING	string value	4
H_TYPE_MIXED	mixed value	8
H_TYPE_HANDLE	handle value	16
H_TYPE_ANY	empty tuple	31

Table 8.3: Type values for control data.

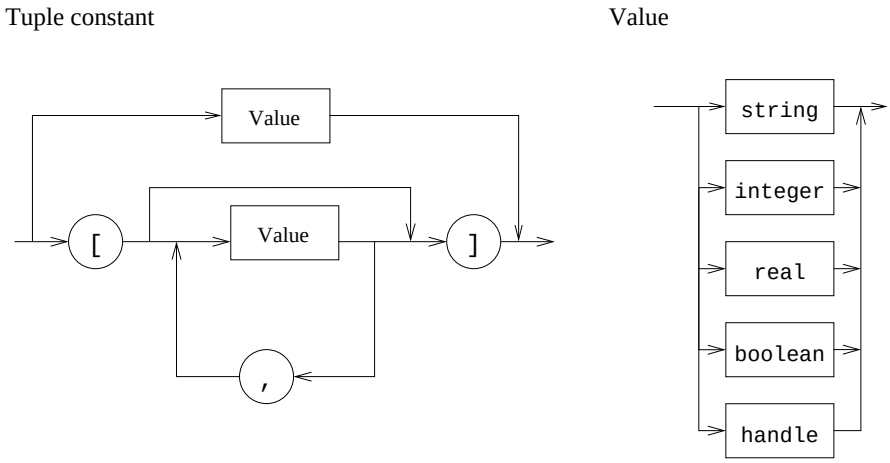


Figure 8.1: The syntax of tuple constants.

```
[100.0,100.0,200.0,200.0]
['FileName','Extension']
[4711,0.815,'Hugo']
```

8.3 Variables

The names of variables are built up as usual by composing letters, digits and the underscore '_'. The type of a variable (iconic or control variable) depends on its position in the parameter list in which the variable identifier is used for the first time (see also section 8.1 on page 281). The type of the variable is determined during the input of the operator parameters: whenever a new identifier appears, a new variable with the same identifier is created. Control and iconic variables must have different names. The value of a variable (iconic or control) is undefined until the first assignment defines it (the variable has not been instantiated yet). A read access to an undefined variable leads to a runtime error.

Instantiated variables contain tuples of values. Depending on the kind of the variable, the data items are either iconic objects or control data. The length of the tuple is determined dynamically by the performed operation. A variable can get new values any number of times, but once a value has been assigned the variable will always keep being instantiated, unless you select the menu item Menu `Execute` ▷ `Reset Program Execution`. The content of the variable is deleted before the variable is assigned new values.

8.3.1 Variable Types

The concept of different types of variables allows a first ("coarse") typification of variables (control or iconic data), whereas the actual type of the data (e.g., `real`, `integer`, `string`, etc.) is undefined until the variable gets assigned with a concrete value. Therefore, it is possible that the data type of a new data item differs from that of the old.

In HDevelop the type of a variable is defined in three different ways:

- explicitly
  - procedure parameter definitions
  - global variable declarations
- implicitly
  - usage in the code

Of these three possibilities, procedure parameter definitions is the most potent one and will overrule the two others in any case, followed by global variable declarations. Therefore, usage in the code is the least potent one. In the latter case the type of a variable is defined by those code lines, and by those lines only, where the value of the variable is written and where the exact type of the new value is known a priori (i.e., before run time).

Within its scope a variable must always have the same type (control, iconic) and the same dimension, otherwise this will cause an error:

- If the type of a variable is never properly defined, the variable will have an undefined type. All lines using the variable will become invalid, and the variable will not be displayed in the variable window.
- If the type of a variable is defined differently two or more times, the variable type will also be undefined, and the above said applies in that case, too.
- If the type of a variable is properly defined, but used in the wrong context (e.g., an iconic variable is used as a control input parameter), only the affected code line will become invalid.

8.3.2 Scope of Variables (local or global)

HDevelop supports local and global variables. All variables are local by default, i.e., they exist only within their procedure. Therefore, local variables with the same name may exist in different procedures without interfering with each other. In contrast, global variables may be accessed in the entire program. They have to be declared explicitly using the operator `global`.

The declaration

```
global tuple File
```

declares a global *control* variable named `File`, whereas

```
global object Image
```

declares a global *iconic* variable `Image`.

The keyword *def* allows you to mark one declaration explicitly as the place where the variable is defined, e.g., `global def object Image`. This is only relevant when exporting the program to a programming language. See the description of the operator `global` for more information.

Once the global variable is declared, it can be used just like a local variable inside the procedure it has been declared in. If you want to access a global variable in a different procedure, you have to announce this by using the same `global ...` call (otherwise, a local variable will be created).

```
main procedure:
  * declare global variables
  global tuple File
  global object Image
  ...
  File := 'particle'
  read_image(Image, File)
  process_image()
  * Image has been changed by process_image()
  * File remains unchanged
  ...

process_image procedure:
  * use global variable
  global object Image
  ...
  bin_threshold(Image, Region)
  File := 'fuse'
  read_image(Image, File)
  return()
```

Because procedures have to explicitly announce their use of global variables, existing procedures cannot be broken by introducing global variables in other parts of the program. By nature, the names of global variables have to be unique in the entire HDevelop program, i.e., all loaded external procedures, the main procedure and all local procedures. The variable window provides a special tab to list all global variables that are currently declared.

Symbol	Types
i	integer
a	arithmetic, that is: integer or real
b	boolean
s	string
v	all types (atomic)
t	all types (tuple)

Table 8.4: Symbols for the operation description.

8.4 Operations on Iconic Objects

Iconic objects are exclusively processed by HALCON operators. HALCON operators work on tuples of iconic objects, which are represented by their surrogates in the HALCON data management. The results of those operators are again tuples of iconic objects or control data elements. For a detailed description of the HALCON operators refer to the HALCON reference manual and the remarks in section 8.5.3 on page 289.

8.5 Expressions for Input Control Parameters

In HDevelop, the use of expressions like arithmetic operations or string operations is limited to control input parameters; all other kinds of parameters must be assigned by variables.

8.5.1 General Features of Tuple Operations

This section intends to give you a short overview over the features of tuples and their operations. A more detailed description of each operator mentioned here is given in the following sections.

Please note that in all following tables variables and constants have been substituted by letters which indicate allowed data types. These letters provide information about possible limitations of the areas of definition. The letters and their meaning are listed in table 8.4. Operations on these symbols can only be applied to parameters of the indicated type or to expressions that return a result of the indicated type.

The symbol names i, a, b, and s can denote atomic tuples (tuples of length 1) as well as tuples with arbitrary length.

Operations are normally described assuming atomic tuples. If the tuple contains more than one element, most operators work as follows:

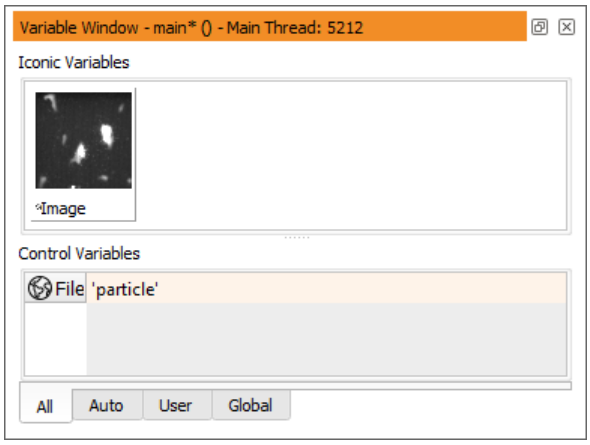


Figure 8.2: Global variables.

Input	Result
`5 * 5`	`25`
`[5] * [5]`	`25`
`[1,2,3] * 2`	`[2,4,6]`
`[1,2,3] * 2.1 + 10`	`[12.1,14.2,16.3]`
`[1,2,3] * [1,2,3]`	`[1,4,9]`
`[1,2,3] * [1,2]`	`runtime error`
`'Text1' + 'Text2'`	`'Text1Text2'`
`17 + '3'`	`'173'`
`'Text ' + 3.1 * 2`	`'Text 6.2'`
`3.1 * (2 + ' Text')`	`runtime error`
`3.1 + 2 + ' Text'`	`'5.1 Text'`
`3.1 + (2 + ' Text')`	`'3.12 Text'`

Table 8.5: Examples for arithmetic operations with tuples and strings.

- If one of the tuples is of length one, all elements of the other tuples are combined with that single value for the chosen operation.
- If both tuples have a length greater than one, both tuples must have the same length (otherwise a runtime error occurs). In this case, the selected operation is applied to all elements with the same index. The length of the resulting tuples is identical to the length of the input tuples.
- If one of the tuples is of length 0 (`[]`), a runtime error occurs.

In table 8.5 you can find some examples for arithmetic operations with tuples. Pay special attention to the order in which the string concatenations are performed. The basic arithmetic operations in HDevelop are `+`, `-`, `*`, `/`. Please note that `+` is a dimorphic operation: If both operands are numeric, it adds numbers. If at least one of the operands is a `string`, it concatenates both operands as strings.

8.5.2 Assignment

In HDevelop, an assignment is treated like an operator. To use an assignment you have to select the operator `assign (Input, Result)`. This operator has the following semantics: It evaluates `Input` (right side of assignment) and stores it in `Result` (left side of assignment). However, in the program text the assignment is represented by the usual syntax of the assignment operator: `Result := Input`. The following example outlines the difference between an assignment in C syntax and its transformed version in HDevelop:

The assignment in C syntax

```
u = sin(x) + cos(y);
```

is defined in HDevelop using the assignment operator as

```
assign (sin(x) + cos(y), u)
```

which is displayed in the program window as:

```
u := sin(x) + cos(y)
```

If the result of the expression does not need to be stored into a variable, the expression can directly be used as input value for any operator. Therefore, an assignment is necessary only if the value has to be used several times or if the variable has to be initialized (e.g., for a loop).

The assignment operator `assign_at (Index, Value, Result)` is used to modify tuple elements. The call:

```
assign_at (Radius-1, Area, Areas)
```

is not presented in the program text as an operator call, but in the more intuitive form as:

```
Areas[Radius-1] := Area.
```

As an example:

```
Areas := [1,2,3]
Areas[1] := 9
```

sets `Areas` to `[1,9,3]`.

To construct a tuple with `assign_at`, normally an empty tuple is used as initial value and the elements are inserted in a loop:

```
Tuple := []
for i := 0 to 5 by 1
  Tuple[i] := sqrt(real(i))
endfor
```

As you can see from the examples, the indices of a tuple start at 0.

An insertion into a tuple can generally be performed in one of the following ways:

1. In case of appending the value at the 'back' or at the 'front', the tuple concatenation operation `,` (comma) can be used. Here the operator `assign` is used with the following parameters:

   ```
   assign ([Tuple,NewVal],Tuple)
   ```

   which is displayed as

   ```
   Tuple := [Tuple,NewVal]
   ```

2. If the index position is somewhere in between, the operator `tuple_insert` has to be used.
   To insert the tuple `[11,12,13]` into the tuple `[1,2,3]` at position 1, use

   ```
   tuple_insert ([1,2,3], 1, [11,12,13], Result)
   ```

   resulting in `[1,11,12,13,2,3]`.

In the following example regions are dilated with a circle mask and afterwards the areas are stored into the tuple `Areas`. In this case the operator `assign_at` is used.

```
read_image (Mreut, 'mreut')
threshold (Mreut, Region, 190, 255)
Areas := []
for Radius := 1 to 50 by 1
  dilation_circle (Region, RegionDilation, Radius)
  area_center (RegionDilation, Area, Row, Column)
  Areas[Radius-1] := Area
endfor
```

Please note that first the variable `Areas` has to be initialized to avoid a runtime error. In the example `Areas` is initialized with the empty tuple (`[]`). Instead of `assign_at` the operator `assign` with tuple concatenation

```
Areas := [Areas,Area]
```

could be used, because the element is appended at the back of the tuple. More examples can be found in the program `assign.hdev`.

Operation	Meaning	HALCON operator
`t := [t1,t2]`	concatenate tuples	`tuple_concat`
`i := |t|`	get number of elements of tuple `t`	`tuple_length`
`v := t1[t2]`	select elements `t2` of tuple `t1`	`tuple_select`
`t := t[i1:i2]`	select from element `i1` to element `i2` of tuple `t`	`tuple_select_range`
`t := subset(t1,t2)`	select elements specified in `t2` from `t1`	`tuple_select`
`t := firstn(t,i)`	select first elements from `t` up to index `i`	`tuple_first_n`
`t := lastn(t,i)`	select elements from `t` from index `i` to the end	`tuple_last_n`
`t := select_mask(t1,t2)`	select all elements from `t1` where the corresponding mask value in `t2` is greater than 0	`tuple_select_mask`
`t := remove(t,i)`	remove elements specified in `i` from `t`	`tuple_remove`
`i := find(t1,t2)`	get indices of all occurrences of `t2` within `t1` (or `-1` if no match)	`tuple_find`
`i := replace(t1,t2,t3)`	replace elements	`tuple_replace`
`i := find_first(t1,t2)`	get indices of the first occurrences of `t2` within `t1` (or `-1` if no match)	`tuple_find_first`
`i := find_last(t1,t2)`	get indices of the last occurrences of `t2` within `t1` (or `-1` if no match)	`tuple_find_last`
`t := uniq(t)`	discard all but one of successive identical elements from `t`	`tuple_uniq`
`t := [i1:i2:i3]`	generate a sequence of values from `i1` to `i3` with an increment value of `i2`	`tuple_gen_sequence`
`t := [i1:i2]`	generate a sequence of values from `i1` to `i2` with an increment value of one	`tuple_gen_sequence`
`t := gen_tuple_const(i,t)`	generate a tuple with `i` values set to `t`	`tuple_gen_const`

Table 8.6: Basic operations on tuples (control data) and the corresponding HALCON operators.

8.5.3 Basic Tuple Operations

A basic tuple operation may be selecting one or more values, combining tuples (concatenation) or getting the number of elements (see table 8.6 for operations on tuples containing control data).

The concatenation accepts one or more variables or constants as input. They are all listed between the brackets, separated by commas. The result again is a tuple. Please note the following: `[[t]] = [t] = t`.

`|t|` returns the number of elements of a tuple. The indices of elements range from zero to the number of elements minus one (i.e., `|t|-1`). Therefore, the selection index has to be within this range. [1]

```
Tuple := [V1,V2,V3,V4]
for i := 0 to |Tuple|-1 by 1
  fwrite_string (FileHandle,Tuple[i]+'\n')
endfor
```

In the following examples the variable `Var` contains `[2,4,8,16,16,32]`:

[1] Please note that the index of objects (e.g., `select_obj`) ranges from 1 to the number of elements.

control	iconic
`[]`	`gen_empty_obj()`
`[t1,t2]`	`concat_obj(p1, p2, q)`
`|t|`	`count_obj(p, num)`
`t[i]`	`select_obj(p, q, i+1)`
`t[i1:i2]`	`copy_obj(p, q, i1+1, i2-i1+1)`

Table 8.7: Equivalent tuple operations for control and iconic data.

`[1,Var,[64,128]]`	`[1,2,4,8,16,16,32,64,128]`
`|Var|`	`6`
`Var[4]`	`16`
`Var[2:4]`	`[8,16,16]`
`subset(Var,[0,2,4])`	`[2,8,16]`
`select_mask(Var,[1,0,0,1,1,1])`	`[2,16,16,32]`
`remove(Var,[2,3])`	`[2,4,16,32]`
`find(Var,[8,16])`	`2`
`uniq(Var)`	`[2,4,8,16,32]`

Further examples can be found in the program `tuple.hdev`. The HALCON operators that correspond to the basic tuple operations are listed in table 8.6 on page 289.

Note that these direct operations cannot be used for iconic tuples, i.e., iconic objects cannot be selected from a tuple using `[]` and their number cannot be directly determined using `||`. For this purpose, however, HALCON operators are offered that carry out the equivalent tasks. In table 8.7 you can see tuple operations that work on control data (and which are applied via `assign` or `assign_at`) and their counterparts that work on iconic data (and which are independent operators). In the table the symbol `t` represents a control tuple, and the symbols `p` and `q` represent iconic tuples.

8.5.4 Tuple Creation

The simplest way to create a tuple, as mentioned in section 8.2 on page 282, is the use of constants together with the operator `assign` (or in case of iconic data one of its equivalents shown in table 8.7):

```
assign ([],empty_tuple)
assign (4711,one_integer)
assign ([4711,0.815],two_numbers)
```

This code is displayed as

```
empty_tuple := []
one_integer := 4711
two_numbers := [4711,0.815]
```

This is useful for constant tuples with a fixed (small) length. More general tuples can be created by successive application of the concatenation or the operator `assign_at` together with variables, expressions or constants. If we want to generate a tuple of length 100, where each element has the value 4711, it might be done like this:

```
tuple := []
for i := 1 to 100 by 1
  tuple := [tuple,4711]
endfor
```

Because this is not very convenient a special function called gen_tuple_const is available to construct a tuple of a given length, where each element has the same value. Using this function, the program from above is reduced to:

```
tuple := gen_tuple_const(100,4711)
```

A fast way to create a sequence of values with a common increment is to use tuple_gen_sequence. For example, to create a tuple containing the values 1..1000, use

```
tuple_gen_sequence(1,1000,1,Sequence)
```

An alternative syntax to the above is to write:

```
Sequence := [1:1:1000]
```

If the increment value is one (as in the above example), it is also possible to write:

```
Sequence := [1:1000]
```

If we want to construct a tuple with the same length as a given tuple there are two ways to get an easy solution. The first one is based on gen_tuple_const:

```
tuple_new := gen_tuple_const(|tuple_old|,4711)
```

The second one is a bit tricky and uses arithmetic functions:

```
tuple_new := (tuple_old * 0) + 4711
```

Here we get first a tuple of the same length with every element set to zero. Then, we add the constant to each element.

In the case of tuples with different values we have to use the loop version to assign the values to each position:

```
tuple := []
for i := 1 to 100 by 1
  tuple := [tuple,i*i]
endfor
```

In this example we construct a tuple with the square values from 1^2 to 100^2.

8.5.5 Type Operations

The type operations allow you to test or query the value type of control data. See table 8.3 on page 283 for the corresponding type constants.

There are also corresponding operations that test each element of the input tuple.

8.5.6 Basic Arithmetic Operations

See table 8.10 for an overview of the available basic arithmetic operations.

All operations are left-associative, except the right-associative unary minus operator. The evaluation usually is done from left to right. However, parentheses can change the order of evaluation and some operators have a higher precedence than others (see section 8.5.17).

The arithmetic operations in HDevelop match the usual definitions. Expressions can have any number of parentheses.

The division operator (a1 / a2) can be applied to integer as well as to real. The result is of type real, if at least one of the operands is of type real. If both operands are of type integer, the division is an integer division. The remaining arithmetic operators (multiplication, addition, subtraction, and negation) can be applied to either integer or real numbers. If at least one operand is of type real, the result will be a real number as well.

Examples:

Simple examples can be found in the program arithmetic.hdev.

Operation	Meaning	HALCON operator
`b := is_handle`	test for handle values	`tuple_is_handle`
`b := is_int(t)`	test for integer values	`tuple_is_int`
`b := is_mixed(t)`	test for mixed values	`tuple_is_mixed`
`b := is_number(t)`	test for numerical values	`tuple_is_number`
`b := is_real(t)`	test for real values	`tuple_is_real`
`b := is_string(t)`	test for string values	`tuple_is_string`
`b := is_valid_handle(t)`	test for valid handles	`tuple_is_valid_handle`
`s := sem_type(t)`	get semantic type	`tuple_sem_type`
`i := type(t)`	get type value	`tuple_type`

Table 8.8: Type operations.

Operation	Meaning	HALCON operator
`t := is_handle_elem`	elementwise test for handle values	`tuple_is_handle_elem`
`t := is_int_elem(t)`	elementwise test for integer values	`tuple_is_int_elem`
`t := is_real_elem(t)`	elementwise test for real values	`tuple_is_real_elem`
`t := is_string_elem(t)`	elementwise test for string values	`tuple_is_string_elem`
`t := sem_type_elem(t)`	get semantic type elementwise	`tuple_sem_type_elem`
`t := type_elem(t)`	get type value elementwise	`tuple_type_elem`

Table 8.9: Elementwise type operations.

8.5.7 Bit Operations

This section describes the operations for bit processing of numbers. The operands have to be integers.

The result of `lsh(i1,i2)` is a bitwise left shift of `i1` that is applied `i2` times. If there is no overflow this is equivalent to a multiplication by 2^{i2}. The result of `rsh(i1,i2)` is a bitwise right shift of `i1` that is applied `i2` times. For non-negative `i1` this is equivalent to a division by 2^{i2}. For negative `i1` the result depends on the used hardware. For `lsh` and `rsh` the result is undefined if the second operand has a negative value or the value is larger than 32. More examples can be found in the program `bit.hdev`.

8.5.8 String Operations

There are several string operations available to modify, select, and combine strings. Furthermore, some operations allow you to convert numbers (`real` and `integer`) to strings.

Operation	Meaning	HALCON operator
`a1 / a2`	division	`tuple_div`
`a1 * a2`	multiplication	`tuple_mult`
`i1 % i2`	modulus	`tuple_mod`
`a1 + a2`	addition	`tuple_add`
`a1 - a2`	subtraction	`tuple_sub`
`-a`	negation	`tuple_neg`

Table 8.10: Basic arithmetic operations.

Expression	Result
4/3	1
4/3.0	1.3333333
(4/3) * 2.0	2.0

Table 8.11: Example Expressions.

Operation	Meaning	HALCON operator
lsh(i1,i2)	left shift	tuple_lsh
rsh(i1,i2)	right shift	tuple_rsh
i1 band i2	bitwise and	tuple_band
i1 bxor i2	bitwise xor	tuple_bxor
i1 bor i2	bitwise or	tuple_bor
bnot i	bitwise complement	tuple_bnot

Table 8.12: Bit operations.

$ (string conversion)

See also: tuple_string.

Operation	Meaning	HALCON operator
v$s	convert v using specification s	tuple_string
v1 + v2	concatenate v1 and v2	tuple_add
strchr(s1,s2)	search character s2 in s1	tuple_strchr
strstr(s1,s2)	search substring s2 in s1	tuple_strstr
strrchr(s1,s2)	search character s2 in s1 (reverse)	tuple_strrchr
strrstr(s1,s2)	search substring s2 in s1 (reverse)	tuple_strrstr
strlen(s)	length of string	tuple_strlen
str_firstn(s,i)	cut the first characters of s up to position i	tuple_str_first_n
str_lastn(s,i)	cut the characters of s from position i	tuple_str_last_n
s{i}	select character at position i; 0 <= i <= strlen(s)-1	tuple_str_bit_select
s{i1:i2}	select substring from position i1 to position i2	tuple_substr
split(s1,s2)	split s1 in substrings at s2	tuple_split
regexp_match(s1,s2)	extract substrings of s1 matching the regular expression s2	tuple_regexp_match
regexp_replace(s1,s2,s3)	replace substrings of s1 matching the regular expression s2 with s3	tuple_regexp_replace
regexp_select(s1,s2)	select tuple elements from s1 matching the regular expression s2	tuple_regexp_select
regexp_test(s1,s2)	return how many tuple elements in s1 match the regular expression s2	tuple_regexp_test
s1 =~ s2	return how many tuple elements in s1 match the regular expression s2	tuple_regexp_test

Table 8.13: String operations.

$ converts numbers to strings or modifies strings. The operation has two operands: The first one (left of the $) is the number that has to be converted. The second one (right of the $) specifies the conversion. It is comparable to the format string of the `printf()` function in the C programming language. This format string consists of the following four parts

<flags><field width>.<precision><conversion>

or as a regular expression:

```
[-+ #]?([0-9]+)?(\.[0-9]*)?[doxXfeEgGsb]?
```

(which roughly translates to zero or more of the characters in the first bracket pair followed by zero or more digits, optionally followed by a dot which may be followed by digits followed by a conversion character from the last bracket pair).

Some conversion examples might show it best:

Input	Output
`23 $ '10.2f'`	`'     23.00'`
`23 $ '-10.2f'`	`'23.00     '`
`4 $ '.7f'`	`'4.0000000'`
`1234.56789 $ '+10.3f'`	`' +1234.568'`
`255 $ 'x'`	`'ff'`
`255 $ 'X'`	`'FF'`
`0xff $ '.5d'`	`'00255'`
`'total' $ '10s'`	`'     total'`
`'total' $ '-10s'`	`'total     '`
`'total' $ '10.3'`	`'       tot'`

flags Zero or more flags, in any order, which modify the meaning of the conversion specification. Flags may consist of the following characters:

- `-` The result of the conversion is left justified within the field.
- `+` The result of a signed conversion always begins with a sign, `+` or `-`.
- Space If the first character of a signed conversion is not a sign, a space character is prefixed to the result.
- `#` The value is to be converted to an "alternate form". For `d` and `s` (see below) conversions, this flag has no effect. For `o` conversion (see below), it increases the precision to force the first digit of the result to be a zero. For `x` or `X` conversion (see below), a non-zero result has `0x` or `0X` prefixed to it. For `e`, `E`, `f`, `g`, and `G` conversions, the result always contains a radix character, even if no digits follow the radix character. For `g` and `G` conversions, trailing zeros are not removed from the result, contrary to usual behavior.

width An optional string of decimal digits to specify a minimum field width. For an output field, if the converted value has fewer characters than the field width, it is padded on the left (or right, if the left-adjustment flag - has been given) to the field width.

precision The precision specifies the minimum number of digits to appear for integer conversions (the field is padded with leading zeros), the number of digits to appear after the radix character for the `e` and `f` conversions, the maximum number of significant digits for the `g` conversion, or the maximum number of characters to be printed from a string conversion. The precision takes the form of a period `.` followed by a decimal digit string. A null digit string is treated as a zero.

conversion A conversion character indicates the type of conversion to be applied:

`d`, `o`, `x`, `X` The integer argument is printed in signed decimal (`d`), unsigned octal (`o`), or unsigned hexadecimal notation (`x` and `X`). The `x` conversion uses the numbers and lower-case letters `0123456789abcdef`, and the `X` conversion uses the numbers and upper-case letters `0123456789ABCDEF`. The precision component of the argument specifies the minimum number of digits to appear. If the value being converted can be represented in fewer digits than the specified minimum, it is expanded with leading zeroes. The default precision is 1. The result of converting a zero value with a precision of 0 is no characters.

`f` The floating-point number argument is printed in decimal notation in the style `[-]`*ddd*`.`*ddd*, where the number of digits after the radix character, `.`, is equal to the precision specification. If the precision is omitted from the argument, six digits are output; if the precision is explicitly 0, no radix appears.

`e`,`E` The floating-point-number argument is printed in the style `[-]`*d*`.`*ddd*`e+`*dd*, where there is one digit before the radix character, and the number of digits after it is equal to the precision. When the precision is missing, six digits are produced; if the precision is 0, no radix character appears. The `E` conversion character produces a number with `E` introducing the exponent instead of `e`. The exponent always contains at least two digits. However, if the value to be printed requires an exponent greater than two digits, additional exponent digits are printed as necessary.

`g`, `G` The floating-point-number argument is printed in style `f` or `e` (or in style `E` in the case of a `G` conversion character), with the precision specifying the number of significant digits. The style used depends on the value converted; style `e` is used only if the exponent resulting from the conversion is less than -4 or greater than or equal to the precision. Trailing zeros are removed from the result. A radix character appears only if it is followed by a digit.

`s` The argument is taken to be a string, and characters from the string are printed until the end of the string or the number of characters indicated by the precision specification of the argument is reached. If the precision is omitted from the argument, it is interpreted as infinite and all characters up to the end of the string are printed.

In no case does a nonexistent or insufficient field width cause truncation of a field; if the result of a conversion is wider than the field width, the field is simply expanded to contain the conversion result.

Examples for the string conversion can be found in the program `string.hdev`.

+ (string concatenation)

See also: `tuple_add`.

The string concatenation (`+`) can be applied in combination with strings or all numerical types; if necessary, the operands are first transformed into strings (according to their standard representation). At least one of the operands has to be already a string so that the operator can act as a string concatenator. In the following example a file name (e.g., `'Name5.tiff'`) is generated. For this purpose two string constants (`'Name'` and `'.tiff'`) and an integer value (the loop-index `i`) are concatenated:

```
for i := 1 to 5 by 1
  read_image (Image, 'Name'+i+'.tiff')
endfor
```

str(r)chr

See also: `tuple_strchr`, `tuple_strrchr`.

`str(r)chr(s1,s2)` returns the index of the first (last) occurrence of one of the character in `s2` in string `s1`, or `-1` if none of the characters occur in the string. `s1` may be a single string or a tuple of strings.

str(r)str

See also: `tuple_strstr`, `tuple_strrstr`.

`str(r)str(s1,s2)` returns the index of the first (last) occurrence of string `s2` in string `s1`, or `-1` if `s2` does not occur in the string. `s1` may be a single string or a tuple of strings.

`strlen`

See also: `tuple_strlen`.

`strlen(s)` returns the number of characters in `s`.

`str_firstn`

See also: `tuple_str_first_n`.

`str_firstn(s,t)` returns the characters from the beginning of string `s` up to position `t`.

`str_lastn`

See also: `tuple_str_last_n`.

`str_lastn(s,t)` returns the character from position `t` of string `s` to the end.

`{}`

See also: `tuple_str_bit_select`, `tuple_substr`.

`s{i}` selects a single character (specified by index position) from `s`. The index ranges from zero to the length of the string minus 1. The result of the operator is a string of length one.

`s{i1:i2}` returns all characters from the first specified index position (`i1`) up to the second specified position (`i2`) in `s` as a string. The index ranges from zero to the length of the string minus 1.

`split`

See also: `tuple_split`.

`split(s1,s2)` divides the string `s1` into single substrings. The string is split at those positions where it contains a character from `s2`. As an example the result of

```
split('/usr/image:/usr/proj/image',':')
```

consists of the two strings

```
['/usr/image','/usr/proj/image']
```

Regular Expressions

HDevelop provides string functions that use Perl compatible regular expressions. Detailed information about them can be found in the Reference Manual at the descriptions of the corresponding operators, which have the same name but start with `tuple_`. In particular, at the description of `tuple_regexp_match` you find further information about the used syntax, a list of possible options, and a link to suitable literature about regular expressions.

`regexp_match`

See also: `tuple_regexp_match`.

`regexp_match(s1,s2)` searches for elements of the tuple `s1` that match the regular expression `s2`. It returns a tuple with the same size as the input tuple (exceptions exist when working with capturing groups, see the description of `tuple_regexp_match` in the Reference Manual for details). The resulting tuple contains the matching results for each tuple element of the input tuple. For a successful match the matching substring is returned. Otherwise, an empty string is returned.

`regexp_replace`

See also: `tuple_regexp_replace`.

`regexp_replace(s1,s2,s3)` replaces substrings in `s1` that match the regular expression `s2` with the string given in `s3`. By default, only the *first* matching substring of each element in `s1` is replaced. To replace all occurrences, the option `'replace_all'` has to be set in `s2` (see `tuple_regexp_replace`).

For example:

```
assign(regexp_replace(List, '\\.jpg$', '.png'), List)
```

substitutes file names that look like JPEG images with PNG images.

`regexp_select`

See also: `tuple_regexp_select`.

`regexp_select(s1,s2)` returns only the elements of the tuple `s1` that match the regular expression `s2`. In contrast to `regexp_match`, the original tuple elements instead of the matching substrings are returned. Tuple elements that do not match the regular expression are discarded.

For example:

```
assign(regexp_select(List, '\\.jpg$'), Selection)
```

sets `Selection` to all the strings from `List` that look like file names of JPEG images. Please note that the backslash character has to be escaped to be preserved.

`regexp_test`

See also: `tuple_regexp_test`.

`regexp_test(s1,s2)` returns the number of elements of the tuple `s1` that match the regular expression `s2`. Additionally, a short-hand notation of the operator is available, which is convenient in conditional expressions:

```
s1 =~ s2
```

8.5.9 Set Operations

The available set operations are listed in table 8.14.

Operation	Meaning	HALCON operator
`difference(t1,t2)`	difference set	`tuple_difference`
`intersection(t1,t2)`	intersection set	`tuple_intersection`
`symmdiff(t1,t2)`	symmetric difference set	`tuple_symmdiff`
`union(t1,t1)`	union set	`tuple_union`

Table 8.14: Set operations.

8.5.10 Comparison Operations

In HDevelop, the comparison operations are defined not only on atomic values, but also on tuples with an arbitrary number of elements. They always return values of type `boolean`. Table 8.15 shows all comparison operations.

`t1 == t2` and `t1 != t2` are defined on all types. Two tuples are equal (`true`), if they have the same length and all the data items on each index position are equal. If the operands have different types (`integer` and `real`), the

Operation	Meaning	HALCON operator	Alternative notation
t1 < t2	less than	tuple_less	
t1 > t2	greater than	tuple_greater	
t1 <= t2	less or equal	tuple_less_equal	
t1 >= t2	greater of equal	tuple_greater_equal	
t1 == t2	equal	tuple_equal	t1 = t2 (legacy)
t1 != t2	not equal	tuple_not_equal	t1 # t2 (legacy)

Table 8.15: Comparison operations.

1st Operand	Operation	2nd Operand	Result
1	==	1.0	true
[]	==	[]	true
''	==	[]	false
[1,'2']	==	[1,2]	false
[1,2,3]	==	[1,2]	false
[4711,'Hugo']	==	[4711,'Hugo']	true
'Hugo'	==	'hugo'	false
2	>	1	true
2	>	1.0	true
[5,4,1]	>	[5,4]	true
[2,1]	>	[2,0]	true
true	>	false	true
'Hugo'	<	'hugo'	true

Table 8.16: Examples for the comparison of tuples.

integer values are first transformed into `real` numbers. Values of type `string` cannot be mixed up with numbers, i.e., `string` values are considered to be not equal to values of other types.

The four comparison operations compute the lexicographic order of tuples. On equal index positions the types must be identical, however, values of type `integer`, `real`, and `boolean` are adapted automatically. The lexicographic order applies to strings, and the boolean `false` is considered to be smaller than the boolean `true` (`false` < `true`). In the program `compare.hdev` you can find examples for the comparison operations.

8.5.11 Elementwise Comparison Operations

These comparison operations compare the input tuples `t1` and `t2` elementwise. If both tuples have the same length, the corresponding elements of both tuples are compared. Otherwise, either `t1` or `t2` must have length 1. In this case, the comparison is performed for each element of the longer tuple with the single element of the other tuple. As a precondition for comparing the tuples elementwise two corresponding elements must either both be (integer or floating point) numbers or both be strings.

8.5.12 Boolean Operations

The boolean operations `and`, `xor`, `or`, and `not` are defined only for tuples of length 1. `l1 and l2` is set to `true` (1) if both operands are `true` (1), whereas `l1 xor l2` returns `true` (1) if exactly one of both operands is true. `l1 or l2` returns `true` (1) if at least one of the operands is `true` (1). `not l` returns `true` (1) if the input is `false` (0), and `false` (0), if the input is `true` (1).

Operation	Meaning	HALCON operator	Alternative notation
t1 [<] t2	less than	tuple_less_elem	
t1 [>] t2	greater than	tuple_greater_elem	
t1 [<=] t2	less or equal	tuple_less_equal_elem	
t1 [>=] t2	greater of equal	tuple_greater_equal_elem	
t1 [==] t2	equal	tuple_equal_elem	t1 [=] t2 (legacy)
t1 [!=] t2	not equal	tuple_not_equal_elem	t1 [#] t2 (legacy)

Table 8.17: Elementwise comparison operations.

1st Operand	Operation	2nd Operand	Result
[1,2,3]	[<]	[3,2,1]	[1,0,0]
['a','b','c']	[==]	'b'	[0,1,0]
['a','b','c']	[<]	['b']	[1,0,0]

Table 8.18: Examples for the elementwise comparison of tuples.

8.5.13 Trigonometric Functions

All these functions work on tuples of numbers as arguments. The input can either be of type integer or real. However, the resulting type will be of type real. The functions are applied to all tuple values, and the resulting tuple has the same length as the input tuple. For atan2 the two input tuples have to be of equal length or one of them of length 1. table 8.20 on page 300 shows the provided trigonometric functions. For the trigonometric functions the angle is specified in radians.

8.5.14 Exponential Functions

All these functions work on tuples of numbers as arguments. The input can either be of type integer or real. However, the resulting type will be of type real. The functions are applied to all tuple values and the resulting tuple has the same length as the input tuple. For pow and ldexp the two input tuples have to be of equal length or one of them of length 1. See table 8.21 for the provided exponential functions.

8.5.15 Numerical Functions

The numerical functions shown in table 8.22 on page 301 work on different data types.

The functions min and max select the minimum and the maximum values of the tuple values. All of these values either have to be of type string, or integer/real. It is not allowed to mix strings with numerical values. The resulting value will be of type real, if at least one of the elements is of type real. If all elements are of type integer the resulting value will also be of type integer. The same applies to the function sum that determines the sum of all values. If the input arguments are strings, string concatenation will be used instead of addition.

Operation	Meaning	HALCON operator
l1 and l2	logical 'and'	tuple_and
l1 xor l2	logical 'xor'	tuple_xor
l1 or l2	logical 'or'	tuple_or
not l	negation	tuple_not

Table 8.19: Boolean operations.

Operation	Meaning	HALCON Operator
sin(a)	sine of a	tuple_sin
cos(a)	cosine of a	tuple_cos
tan(a)	tangent of a	tuple_tan
asin(a)	arc sine of a in the interval $[-\pi/2, \pi/2], a \in [-1, 1]$	tuple_asin
acos(a)	arc cosine a in the interval $[-\pi/2, \pi/2], a \in [-1, 1]$	tuple_acos
atan(a)	arc tangent a in the interval $[-\pi/2, \pi/2], a \in [-\infty, +\infty]$	tuple_atan
atan2(a1,a2)	arc tangent a1/a2 in the interval $[-\pi, \pi]$	tuple_atan2
sinh(a)	hyperbolic sine of a	tuple_sinh
cosh(a)	hyperbolic cosine of a	tuple_cosh
tanh(a)	hyperbolic tangent of a	tuple_tanh
asinh(a)	inverse hyperbolic sine of a	tuple_asinh
acosh(a)	inverse hyperbolic cosine of a	tuple_acosh
atanh(a)	inverse hyperbolic tangent of a	tuple_atanh

Table 8.20: Trigonometric functions.

Operation	Meaning	HALCON operator
exp(a)	exponential function e^a	tuple_exp
exp2(a)	base 2 exponential function 2^a	tuple_exp2
exp10(a)	base 10 exponential function 10^a	tuple_exp10
log(a)	natural logarithm $\ln(a)$, a > 0	tuple_log
log2(a)	base 2 logarithm, $\log_2(a)$, a > 0	tuple_log2
log10(a)	base 10 logarithm, $\log_{10}(a)$, a > 0	tuple_log10
pow(a1,a2)	$a1^{a2}$	tuple_pow
ldexp(a1,a2)	$a1 \cdot 2^{\texttt{floor(a2)}}$	tuple_ldexp
tgamma(a)	gamma function $\Gamma(a)$	tuple_tgamma
lgamma(a)	logarithm of the absolute value of the gamma function $\log(\lvert\Gamma(a)\rvert)$	tuple_lgamma
erf(a)	error function $erf(a)$	tuple_erf
erfc(a)	complementary error function $erfc(a)$	tuple_erfc

Table 8.21: Exponential functions.

The functions mean, deviation, sqrt, cbrt, deg, rad, fabs, ceil, floor and fmod work with integer and real; the result is always of type real. The function mean calculates the mean value and deviation the standard deviation of numbers. sqrt calculates the square root of a number. cbrt calculates the cube root of a number. hypot calculates the hypotenuse of two numbers.

cumul returns the different cumulative sums of the corresponding elements of the input tuple, and median calculates the median of a tuple. For both functions, the resulting value will be of type real, if at least one of the elements is of type real. If all elements are of type integer the resulting value will also be of type integer. select_rank returns the element at rank i and works for tuples containing int or real values. The index i is of type int.

deg and rad convert numbers from radians to degrees and from degrees to radians, respectively.

real converts an integer to a real. For real as input it returns the input. int converts a real to an integer and truncates it. round converts a real to an integer and rounds the value. For integer it returns the input. The function abs always returns the absolute value that is of the same type as the input value.

The following example (file name: euclid_distance.hdev) shows the use of some numerical functions:

Operation	Meaning	HALCON operator
`min(t)`	minimum value of the tuple	`tuple_min`
`min2(t1,t2)`	elementwise minimum of two tuples	`tuple_min2`
`max(t)`	maximum value of the tuple	`tuple_max`
`max2(t1,t2)`	elementwise maximum of two tuples	`tuple_max2`
`sum(t)`	sum of all tuple elements or string concatenation	`tuple_sum`
`mean(a)`	mean value	`tuple_mean`
`deviation(a)`	standard deviation	`tuple_deviation`
`cumul(a)`	cumulative sums of a tuple	`tuple_cumul`
`median(a)`	median of a tuple	`tuple_median`
`select_rank(a,i)`	element at rank i of a tuple	`tuple_select_rank`
`sqrt(a)`	square root $\sqrt{a}$	`tuple_sqrt`
`cbrt(a)`	cube root $\sqrt[3]{a}$	`tuple_cbrt`
`hypot(a,b)`	hypotenuse $\sqrt{a^2+b^2}$	`tuple_hypot`
`deg(a)`	convert radians to degrees	`tuple_deg`
`rad(a)`	convert degrees to radians	`tuple_rad`
`real(a)`	convert `integer` to `real`	`tuple_real`
`int(a)`	truncate `real` to `integer`	`tuple_int`
`round(a)`	convert `real` to `integer`	`tuple_round`
`abs(a)`	absolute value of `a` (`integer` or `real`)	`tuple_abs`
`fabs(a)`	absolute value of `a` (always `real`)	`tuple_fabs`
`ceil(a)`	smallest integer value not smaller than `a`	`tuple_ceil`
`floor(a)`	largest integer value not greater than `a`	`tuple_floor`
`fmod(a1,a2)`	fractional part of `a1`/`a2`, with the same sign as `a1`	`tuple_fmod`
`sgn(a)`	elementwise sign of a tuple	`tuple_sgn`

Table 8.22: Numerical functions.

```
V1 := [18.8,132.4,33,19.3]
V2 := [233.23,32.786,234.4224,63.33]
Diff := V1 - V2
Distance := sqrt(sum(Diff * Diff))
Dotvalue := sum(V1 * V2)
```

First, the Euclidian distance of the two vectors `V1` and `V2` is computed, by using the formula:

$$d = \sqrt{\sum_i (V1_i - V2_i)^2}$$

The difference and the multiplication (square) are successively applied to each element of both vectors. Afterwards `sum` computes the sum of the squares. Then the square root of the sum is calculated. After that the dot product of `V1` and `V2` is determined by the formula:

$$\langle V1, V2 \rangle = \sum_i (V1_i * V2_i)$$

8.5.16 Miscellaneous Functions

`sort` sorts the tuple values in ascending order, that means, that the first value of the resulting tuple is the smallest one. But again: strings must not be mixed up with numbers. `sort_index` sorts the tuple values in ascending order, but in contrast to `sort` it returns the index positions (0..) of the sorted values.

Operation	Meaning	HALCON operator
sort(t)	sorting in increasing order	tuple_sort
sort_index(t)	return index instead of values	tuple_sort_index
inverse(t)	reverse the order of the values	tuple_inverse
number(v)	convert string to a number	tuple_number
environment(s)	value of an environment variable	tuple_environment
ord(a)	convert strings of length 1 into character codes (Unicode or ANSI)	tuple_ord
chr(a)	Inverse of ord(a)	tuple_chr
ords(s)	convert strings into character codes (Unicode or ANSI)	tuple_ords
chrt(i)	Inverse of ords(s)	tuple_chrt
rand(a)	create random numbers	tuple_rand

Table 8.23: Miscellaneous functions.

band
bxor bor
and
xor or
!= == # =
<= >= < >
+ -
/ * %
- (unary minus) not
$

Table 8.24: Operation precedence (increasing from top to bottom).

The function inverse reverses the order of the tuple values. Both sort and inverse are identical, if the input is empty, if the tuple is of length 1, or if the tuple contains only one value in all positions, e.g., [1,1,...,1].

is_number returns true for variables of the type integer or real and for variables of the type string that represent a number.

The function number converts a string representing a number to an integer or a real depending on the type of the number. Note that strings starting with 0x are interpreted as hexadecimal numbers, and strings starting with 0 (zero) as octal numbers; for example, the string '20' is converted to the integer 20, '020' to 16, and '0x20' to 32. If called with a string that does not represent a number or with a variable of the type integer or real, number returns a copy of the input.

environment returns the value of an environment variable. Input is the name of the environment variable as a string.

ord converts a tuple of strings of length 1 into a tuple of integers (Unicode character codes or ANSI codes). chr converts the Unicode character codes or the ANSI codes into a tuple of strings, each of length 1.

ords converts a tuple of strings into a tuple of integers (Unicode character codes or ANSI codes). chrt converts the Unicode character codes or the ANSI codes into a tuple with strings that are separated by the number 0.

8.5.17 Operation Precedence

See table 8.24 for the precedence of the operations for control data. Some operations (like functions, | |, t[], etc.) are left out, because they mark their arguments clearly.

8.6 Vectors

A vector is a container that can hold an arbitrary number of elements, all of which must have the exact same variable type (i.e., tuple, iconic object, or vector). The variable type “vector” is specific to HDevelop. It is available in HDevelop 12.0 or higher. Please note that programs utilizing vector variables cannot be executed in older versions of HDevelop.

A vector of tuples or objects is called one-dimensional, a vector of vectors of tuples or objects is two-dimensional, and so on. The type of a vector must not change within the program, i.e., its dimension has to remain constant, and vectors of tuples must not be assigned iconic objects or vice versa.

This is the definition of a vector in EBNF (Extended Backus-Naur Form) grammar:

```
vector     = "{" list "}" ;
list       = tuplelist | objectlist | vectorlist ;
tuplelist  = tuple, {",", tuple} ;
objectlist = object, {",", object} ;
vectorlist = vector, {",", vector} ;
tuple      = "[" control "]" ;
control    = string | integer | real | boolean ;
```

Construction of Vectors

A vector is defined by providing a comma-separated list of its elements in curly brackets.

```
vectorT := {[1], [2], [3]}      // one-dimensional vector
```

This is equivalent to

```
vectorT := {1, 2, 3}            // tuples of length 1 do not require square brackets
```

Of course, variable names or arbitrary expressions can be used instead of constants.

```
t1 := 1
vectorT := {t1, t1 * 2, 3}
```

The following example defines a vector of iconic objects.

```
read_image (Image, 'clip')
threshold (Image, Region, 0, 63)
connection (Region, ConnectedRegions)
vector0 := {Image, Region, ConnectedRegions}
```

The following example defines a two-dimensional vector variable.

```
vectorV := {vectorT, {[4,5,6,7], [8,9]}}
```

It is also possible to define vector variables using the `.at()` and `.insert()` operations (see below).

The list of the vector’s elements may also be empty, i.e., an empty vector `{}` is valid as in the following example:

```
vectorV2 := {{1,2}, {}}
```

Note, however, that an empty vector has no specific type. Therefore, all of the following three empty assignments are valid:

```
vectorO2 := vectorO
vectorT2 := vectorT
vectorV2 := vectorV

vectorO2 := {}
vectorT2 := {}
vectorV2 := {}
```

Assigning an empty vector to a vector variable is equivalent to the `.clear()` operation (see below). On the other hand, that means that the assignment of an empty vector to a variable is not sufficient to define the variable's type (see section "Variable Types" on page 284). Such a variable will have an undefined type (and, therefore, be invalid) unless its type is defined properly elsewhere in the program.

Accessing and Setting Vector Elements

A single vector element is accessed using the `.at()` operation, the argument of which ranges from 0 to the number of vector elements minus 1. Several `.at()` operations can be combined to access the subelements of multi-dimensional vectors. It is a runtime error to access non-existing vector elements.

```
tuple  := vectorT.at(0)          // tuple  := 1
region := vectorO.at(1)          // region := Region
vector := vectorV.at(0)          // vector := {[1,2,3]}
tuple  := vectorV.at(1).at(1)    // tuple  := [8, 9]
```

The `.at()` operation is also used to set vector elements. Writing to a non-existing vector element is allowed. If necessary, the vector is automatically filled with empty elements.

```
vectorT.at(2) := 33             // vectorT := {[1], [2], [33]}
vectorE.at(4) := 'text'        // vectorE := {[], [], [], [], 'text'}
```

The `at()` operation also allows you to construct a vector dynamically in a loop:

```
for i:= 0 to 5 by 1
  vecT.at(i) := gen_tuple_const(i,5)
endfor
```

The `.insert()` operation specifies an index position and a value. It shifts the values from the given index to the end by one positon, and sets the value at the index to the new value.

The `.remove()` operation performs the opposite operation. It removes the value at the specified position and moves all following values to the left.

```
vectorT.insert(1, 99)           // vectorT := {[1], [99], [2], [33]}
vectorT.remove(2)               // vectorT := {[1], [99], [33]}
```

Like with the `.at()` operation, the vector is automatically filled with empty elements if necessary.

```
vectorNew.insert(2, 3)          // vectorNew := {[], [], [3]}
```

The `.concat()` operation concatenates two vectors of the same type and dimension.

```
vectorC := vectorT.concat(vectorNew)   // vectorC := {[1], [99], [33], [], [], [3]}
```

Getting the Number of Vector Elements

The number of vector elements is queried using the `length()` operation.

```
i := vectorT.length()             // i := 3
j := vectorV.length()             // j := 2
k := vectorV.at(0).length()       // k := 3
```

Clearing a Vector Variable

The .clear() operation removes all elements from the corresponding vector variable. Note however that the cleared vector still keeps its variable type.

```
vectorT.clear()
* vectorT := vector0              // illegal: vectorT is a tuple vector
* vectorT := vectorV              // illegal: vectorT is one-dimensional
```

Modifying Vector Operations

The vector operations .clear(), .insert(), and .remove() are special in that they modify the input vector. Therefore, they may only be used within an independent program statement (a so-called executable expression) and not within expressions for assignments or input parameters. However, it is allowed to chain more than one modifying vector operation in an executable expression, e.g.:

```
v := {1, 2, 3}
v.insert(1, 5).insert(2, 4).remove(0)            // sets v to {5, 4, 2, 3}
```

Use the special operator executable_expression to enter a modifying vector operation into the operator window.

Testing Vector Variables for (In)equality

The operations == and != are used to test two vector variables for equality or inequality, respectively.

Converting Vectors to Tuples and Vice-versa

A vector variable can be flattened to a tuple using the convenience operator convert_vector_to_tuple. It concatenates all tuple values that are stored in the input vector and stores them in the output tuple.

```
convert_vector_to_tuple (vectorV, T)    // T := [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

The convenience operator convert_tuple_to_vector_1d stores the elements of the input tuple as single elements of the one-dimensional output vector.

```
convert_tuple_to_vector_1d (T, 1, V)   // V := {[1],[2],[3],[4],[5],[6],[7],[8],[9]}
```

8.7 Reserved Words

The identifiers listed in table 8.25 on page 307 are reserved words and their usage is strictly limited to their predefined meaning. They cannot be used as variable names.

8.8 Control Flow Operators

The operators introduced in this section execute a block of operators conditionally or repeatedly. Usually, these operators come in pairs: One operator marks the start of the block while the other marks the end. The code lines in-between are referred to as the body of a control flow structure.

When you enter a control flow operator to start a block, HDevelop also adds the corresponding closing operator by default to keep the program code balanced. In addition, the IC is placed between the control flow operators. This is fine for entering new code blocks. If you want to add control flow operators to existing code, you can also add the operators individually. Keep in mind, however, that a single control flow operator is treated as invalid code until its counterpart is entered as well.

In the following, <condition> is an expression that evaluates to an integer or boolean value. A condition is false if the expression evaluates to 0 (zero). Otherwise, it is true. HDevelop provides the following operators to control the program flow:

if ... endif This control flow structure executes a block of code conditionally. The operator if takes a condition as its input parameter. If the condition is true, the body is executed. Otherwise the execution is continued at the operator call that follows the operator endif.

To enter both if and endif at once, select the operator if in the operator window and make sure the check box next to the operator is ticked.

```
if (<condition>)
  ...
endif
```

if ... else ... endif Another simple control flow structure is the condition with alternative. If the condition is true, the block between if and else is executed. If the condition is false, the part between else and endif is executed.

To enter all three operators at once, select the operator ifelse in the operator window and make sure the check box next to the operator is ticked.

```
if (<condition>)
  ...
else
  ...
endif
```

abs	acos	acosh	and
asin	asinh	assign	assign_at
atan	atan2	atanh	band
bnot	bor	break	bxor
case	catch	cbrt	ceil
chr	chrt	comment	continue
cos	cosh	cumul	default
deg	deviation	else	elseif
endfor	endif	endswitch	endswitch
endtry	endwhile	environment	erf
erfc	exit	exp	exp10
exp2	export_def	fabs	false
find	floor	fmod	for
gen_tuple_const	global	HNULL	hypot
H_MSG_FAIL	H_MSG_FALSE	H_MSG_TRUE	H_MSG_VOID
H_TYPE_ANY	H_TYPE_HANDLE	H_TYPE_INT	H_TYPE_MIXED
H_TYPE_REAL	H_TYPE_STRING	if	ifelse
insert	int	inverse	is_handle
is_handle_elem	is_int	is_int_elem	is_mixed
is_number	is_real	is_real_elem	is_string
is_string_elem	is_valid_handle	ldexp	lgamma
log	log10	log2	lsh
max	max2	mean	median
min	min2	not	or
ord	ords	par_join	par_start
pow	rad	rand	real
regexp_match	regexp_replace	regexp_select	regexp_test
remove	repeat	replace	return
round	rsh	select_mask	select_rank
sem_type	sem_type_elem	sgn	sin
sinh	sort	sort_index	split
sqrt	stop	strchr	strlen
strrchr	strrstr	strstr	subset
sum	switch	tan	tanh
tgamma	throw	true	try
type	type_elem	uniq	until
while	xor		

Table 8.25: Reserved words.

`elseif` This operator is similar to the `else`-part of the previous control flow structure. However, it allows you to test for an additional condition. The block between `elseif` and `endif` is executed if <condition1> is false and <condition2> is true. `elseif` may be followed by an arbitrary number of additional `elseif` instructions. The last `elseif` may be followed by a single `else` instruction.

```
if (<condition1>)
  ...
elseif (<condition2>)
  ...
endif
```

This is syntactically equivalent and thus a shortcut for the following code block:

```
if (<condition1>)
  ...
else
  if (<condition2>)
    ...
  endif
endif
```

`while` **...** `endwhile` This is a looping control flow structure. As long as the condition is `true`, the body of the loop is executed. In order to enter the loop, the condition has to be true in the first place. The loop can be restarted and terminated immediately with the operator `continue` and `break`, respectively (see below).

To enter both `while` and `endwhile` at once, select the operator `while` in the operator window and make sure the check box next to the operator is ticked.

```
while (<condition>)
 ...
endwhile
```

`repeat` **...** `until` This loop is similar to the `while` loop with the exception that the condition is tested at the end of the loop. Thus, the body of a `repeat` ... `until` loop is executed at least once. Also in contrast to the `while` loop, the loop is repeated if the condition is false, i.e., *until* it is finally true.

To enter both `repeat` and `until` at once, select the operator `until` in the operator window and make sure the check box next to the operator is ticked.

```
repeat
  ...
until (<condition>)
```

`for` **...** `endfor` The `for` loop is controlled by a start and an end value and an increment value, step, that determines the number of loop steps. These values may also be expressions, which are evaluated immediately before the loop is entered. The expressions may be of type `integer` or of type `real`. If all input values are of type `integer`, the loop variable will also be of type `integer`. In all other cases the loop variable will be of type `real`.

Please note that the `for` loop is displayed differently in the program window than entered in the operator window. What you enter in the operator window as `for(`*start*`,`*end*`,`*step*`,`*index*`)` is displayed in the program window as:

```
for <index> := <start> to <end> by <step>
  ...
endfor
```

To enter both `for` and `endfor` at once, select the operator `for` in the operator window and make sure the check box next to the operator is ticked.

The start value is assigned to the index variable. The loop is executed as long as the following conditions are true: 1) The step value is positive, and the loop index is smaller than or equal to the end value. 2) The step

value is negative, and the loop index is greater than or equal to the end value. After a loop cycle, the loop index is incremented by the step value and the conditions are evaluated again.

Thus, after executing the following lines,

```
for i := 1 to 5 by 1
  j := i
endfor
```

`i` is set to 6 and `j` is set to 5, while in

```
for i := 5 to 1 by -1
  j := i
endfor
```

`i` is set to 0, and `j` is set to 1.

The loop can be restarted and terminated immediately with the operator `continue` and `break`, respectively. (see below).

Please note, that in older versions of HDevelop (prior to HALCON 11), the expressions for start and termination value were evaluated only once when *entering the loop*. A modification of a variable that appeared within these expressions had no influence on the termination of the loop. The same applied to the modifications of the loop index. It also had no influence on the termination. The loop value was assigned to the correct value each time the `for` operator was executed. See the reference manual of the `for` operator for more information.

If the `for` loop is left too early (e.g., if you press `Stop` and set the PC) and the loop is entered again, the expressions will be evaluated, as if the loop were entered for the first time.

In the following example the sine from 0 up to 6π is computed and printed into the graphical window (file name: `sine.hdev`):

```
old_x := 0
old_y := 0
dev_set_color ('red')
dev_set_part(0, 0, 511, 511)
for x := 1 to 511 by 1
    y := sin(x / 511.0 * 2 * 3.1416 * 3) * 255
    disp_line (WindowID, -old_y+256, old_x, -y+256, x)
    old_x := x
    old_y := y
endfor
```

In this example the assumption is made that the window is of size 512×512. The drawing is always done from the most recently evaluated point to the current point.

`continue` The operator `continue` forces the next loop cycle of a `for`, `while`, or `repeat` loop. The loop condition is tested, and the loop is executed depending on the result of the test.

In the following example, a selection of RGB color images is processed. Images with channel numbers other than three are skipped through the use of the operator `continue`. An alternative is to invert the condition and put the processing instructions between `if` and `endif`. But the form with `continue` tends to be much more readable when very complex processing with lots of lines of code is involved.

```
i := |Images|
while (i)
  Image := Images[i]
  count_channels (Image, Channels)
  if (Channels != 3)
    continue
  endif
  * extensive processing of color image follows
endwhile
```

break The operator break enables you to exit for, while, and repeat loops. The program is then continued at the next line after the end of the loop.

A typical use of the operator break is to terminate a for loop as soon as a certain condition becomes true, e.g., as in the following example:

```
Number := |Regions|
AllRegionsValid := 1
* check whether all regions have an area <= 30
for i := 1 to Number by 1
  ObjectSelected := Regions[i]
  area_center (ObjectSelected, Area, Row, Column)
  if (Area > 30)
    AllRegionsValid := 0
    break ()
  endif
endfor
```

In the following example, the operator break is used to terminate an (infinite) while loop as soon as one clicks into the graphics window:

```
while (1)
  grab_image (Image, FGHandle)
  dev_error_var (Error, 1)
  dev_set_check ('~give_error')
  get_mposition (WindowHandle, R, C, Button)
  dev_error_var (Error, 0)
  dev_set_check ('give_error')
  if ((Error = H_MSG_TRUE) and (Button != 0))
    break ()
  endif
endwhile
```

switch ... case ... endswitch The switch block allows you to control the program flow via a multiway branch. The branch targets are specified with case statements followed by an integer constant. Depending on an integer control value the program execution jumps to the matching case statements and continues to the next break statement or the closing endswitch statement. An optional default statement can be defined as the last jump label within a switch block. The program execution jumps to the default label if no preceding case statement matches the control expression.

```
...
switch (Grade)
  case 1:
    Result := 'excellent'
    break
  case 2:
    Result := 'good'
    break
  case 3:
    Result := 'acceptable'
    break
  case 4:
  case 5:
    Result := 'unacceptable'
    break
  default:
    Result := 'undefined'
endswitch
...
```

`stop` The operator `stop` stops the program after the operator is executed. The program can be continued by pressing the `Step Over` or `Run` button.

`exit` The `exit` operator *terminates* the HDevelop session.

`return` The operator `return` returns from the current procedure call to the calling procedure. If `return` is called in the main procedure, the PC jumps to the end of the program, i.e., the program is finished.

`try` **...** `catch` **...** `endtry` This control flow structure enables dynamic exception handling in HDevelop. The program block between the operators `try` and `catch` is watched for exceptions, i.e., runtime errors. If an exception occurs, diagnostic data about what caused the exception is stored in an exception tuple. The exception tuple is passed to the `catch` operator, and program execution continues from there. The program block between the operators `catch` and `endtry` is intended to analyze the exception data and react to it accordingly. If no exception occurs, this program block is never executed.

See section “Error Handling” on page 311, and the reference manual, e.g., the operator `try` for detailed information.

`throw` The operator `throw` allows you to generate user-defined exceptions.

8.9 Error Handling

This section describes how errors are handled in HDevelop programs. When an error occurs, the default behavior of HDevelop is to stop the program execution and display an error message box. While this is certainly beneficial at the time the program is developed, it is usually not desired when the program is actually deployed. A finished program should react to errors itself. This is of particular importance if the program interacts with the user.

There are basically two approaches to error handling in HDevelop:

- tracking the return value (error code) of operator calls
- using exception handling

A major difference between these approaches is the realm of application: The first method handles errors inside the procedure in which they occur. The latter method allows errors to work their way up in the call stack until they are finally dealt with.

8.9.1 Tracking the Return Value of Operator Calls

The operator `dev_set_check` specifies if error message boxes are displayed at all.

To turn message boxes off, use

```
dev_set_check('~give_error')
```

HDevelop will then ignore any errors in the program. Consequently, the programmer has to take care of the error handling. Every operator call provides a return value (or error code) which signals success or failure of its execution. This error code can be accessed through a designated error variable:

```
dev_error_var(ErrorCode, 1)
```

This operator call instantiates the variable `ErrorCode`. It stores the error code of the last executed operator. Using this error code, the program can depend its further flow on the success of an operation.

```
...
if (ErrorCode != H_MSG_TRUE)
  * react to error
endif
* continue with program
...
```

The error message related to a given error code can be obtained with the operator get_error_text. This is useful when reporting errors back to the user of the program.

If the error is to be handled in a calling procedure, an appropriate output control variable has to be added to the interface of each participating procedure, or the error variable has to be defined as a global variable (see section 8.3.2).

```
global tuple ErrorCode
dev_error_var(ErrorCode, 1)
...
```

8.9.2 Exception Handling

HDevelop supports dynamic exception handling, which is comparable to the exception handling in C++ and C#.

A block of program lines is watched for run-time errors. If an error occurs, an exception is raised and an associated exception handler is called. An exception handler is just another block of program lines, which is invisible to the program flow unless an error occurs. The exception handler may directly act on the error or it may pass the associated information (i.e., the exception) on to a parent exception handler. This is also known as *rethrowing an exception*.

In contrast to the tracking method described in the previous section, the exception handling requires HDevelop to be set up to stop on errors. This is the default behavior. It can also be turned on explicitly:

```
dev_set_check('give_error')
```

Furthermore, HDevelop can be configured to let the user choose whether or not an exception is thrown, or to throw exceptions automatically. This behavior is set in the preferences tab `General Options` ▷ `Experienced User`.

An HDevelop exception is a tuple containing data related to a specific error. It always contains the error code as the first item. The operator dev_get_exception_data provides access to the elements of an exception tuple.

HDevelop exception handling is applied in the following way:

```
...
try
  * start block of watched program lines
  ...
catch(Exception)
  * get error code
  ErrorCode := Exception[0]
  * react to error
endtry
* program continues normally
...
```

8.10 Parallel Execution

The HDevelop language supports the parallel execution of procedure and operator calls as subthreads of the main thread. Once started, subthreads are identified by a thread ID which is an integer process number depending on the operating system. The execution of subthreads is independent of the thread they have been started from. Therefore, the exact point in time when a specific threat finishes cannot be predicted. If you want to access data returned from a group of threads, it is required to explicitly wait for the corresponding threads to finish.

HDevelop limits the number of threads to 20 by default. This number can be modified in the preferences if required (see section "`General Options` ▷ `General Options`" on page 76). The main reason for limiting the number of simultaneous threads at all, is to prevent the user from inadvertently generating a huge number of threads due to a programming error. In that case, the system load as well as the memory consumption may grow so high that HDevelop can become unresponsive.

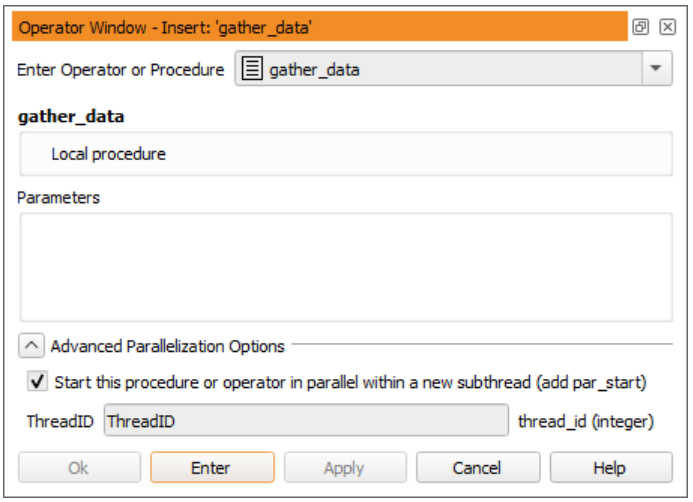


Figure 8.3: Operator window with parallelization options.

Please note that the thread count includes all threads that are “alive”. In particular it also includes threads that have already finished but are still being referenced by a variable. This has to be taken into account when tweaking the thread limit. See also section “Execution of Threads in HDevelop” on page 315 for information about the lifetime of a thread.

8.10.1 Starting a Subthread

To start a new thread, prefix the corresponding operator or procedure call with the `par_start` qualifier:

```
par_start <ThreadID> : gather_data()
...
```

This call starts the hypothetical procedure `gather_data()` as a new subthread in the background and continues to execute the subsequent program lines. The thread ID is returned in the variable `ThreadID` which must be specified in angle brackets. Unlike in HDevelop, in HDevEngine a given `ThreadID` is only valid within the procedure that started the thread.

Note that `par_start` is not an actual operator but merely a qualifier that modifies the calling behavior. Therefore, it is not possible to select `par_start` in the operator window.

If starting a new subthread would exceed the configured maximum number of threads (see above), an exception is thrown.

You can also start procedure or operator calls as a subthread from the operator window (see figure 8.3). To do this, open the section `Advanced Parallelization Options` at the bottom of the operator window, tick the check box and enter the name of the variable that will hold the thread ID. If you double-click on a program containing the `par_start` qualifier, the parallelization options will also be displayed in the operator window. For certain program lines (e.g., comments, declarations, loops, or assignments) `par_start` is not supported and the corresponding options will also not be available in the operator window. For a general description of the operator window see section “Operator Window” on page 138.

It is supported to start multiple threads in a loop. In that case the thread IDs need to be collected so that all threads can be referenced later:

```
ThreadIDs := []
for Index := 1 to 5 by 1
  par_start <ThreadID> : gather_data()
  ThreadIDs := [ThreadIDs, ThreadID]
endfor
```

It is often more convenient to collect the thread IDs in a vector variable (page 303):

```
for Index := 1 to 5 by 1
  par_start <ThreadIDs.at(Index - 1)> : gather_data()
endfor
```

Special care must be taken when the subthread returns data in an output variable. In particular, output variables must not be accessed in other threads while the subthread is still running. Otherwise the data is not guaranteed to be valid.

Likewise, its must be ensured that multiple threads do not interfere with their results. Suppose the procedure `gather_data` is started as multiple threads like above but returns data in an output control variable:

```
for Index := 1 to 5 by 1
  par_start <ThreadIDs.at(Index - 1)> : gather_data(Result)  // BEWARE!!!
endfor
```

In the above example, all the threads would return their result in the same variable which is certainly not what was intended. The final value of `Result` would be the (unpredictable) return value of the thread that finishes last, and all other results would be lost.

An easy solution to this problem is to collect the returned data in a vector variable as shown previously with the thread IDs:

```
for Index := 1 to 5 by 1
  par_start <ThreadIDs.at(Index - 1)> : gather_data(Result.at(Index - 1))
endfor
```

Here, each invocation of `gather_data` returns its result in a unique slot of the vector variable `Result`.

8.10.2 Waiting for Subthreads to Finish

Use the operator `par_join` to wait for the completion of a single thread or a group of threads.

As an example why this is necessary suppose we want to call a procedure that performs some magic calculation in the background and returns a count number as a result. In the subsequent program lines we want to use that number for further calculations.

```
par_start <ThreadID> : count_objects(num)
...
for i := 1 to num by 1     // BEWARE: num might be uninitialized
  ...
endfor
```

Simply relying on the subthread to be fast enough is most likely going to fail. Therefore, an explicit call to `par_join` is required beforehand.

```
par_start <ThreadID> : count_objects(num)
...
par_join(ThreadID)
for i := 1 to num by 1
  ...
endfor
```

Note that in HDevelop it is not strictly required to use `par_join` because the main thread will always outlive the subthreads. However, omitting it might lead to trouble if the program is going to be executed in HDevEngine (see Programmer's Guide, page 130) or exported to a programming language. Similarly, access to global variables might need some additional synchronization if the program is going to be exported.

Given the example from the previous section, waiting for the finishing of all the threads that were started in a loop is achieved using the following lines.

```
convert_vector_to_tuple(ThreadIDs, Threads)
par_join(Threads)
```

Please note that the thread IDs had been collected in a vector variable. Thus, the conversion to a tuple is necessary for par_join to work properly.

The par_join operator blocks the further execution of the procedure it has been called from until all specified threads have finished. In the subsequent program lines, results from the corresponding threads can then be accessed reliably.

8.10.3 Execution of Threads in HDevelop

In general, threads in HDevelop are only executed in parallel when the program runs continuously after pressing F5. In all other execution modes only the selected thread is started and all other threads remain stopped unless an explicit user interaction advances their execution. Active breakpoints, stop instructions, runtime errors or uncaught exceptions also cause all threads to stop so their current state can be evaluated. This convention enables a clearly defined debugging process because it eliminates uncontrollable side-effects from other threads. Any editing action in the program window will also cause a concurrently running program to stop.

Threads cannot be "killed" externally. They can be stopped between operator calls or by aborting interruptible operators. If any thread executes a long-running operator that cannot be interrupted at the time HDevelop tries to stop the program execution, a corresponding message will be displayed in the status line, and the corresponding thread will eventually stop after the operator has finished.

Selected Thread

Exactly one of the threads is the so-called *selected* thread; by default this corresponds to the main thread of the program. The position of the PC, the status of the call stack, and the state of the variables in the variable window are linked to the selected thread. The selected thread can change automatically to a thread that stops, e.g., by a breakpoint, stop instruction, an uncaught exception, or a draw operator.

How to select a specific thread is described in section "Inspecting Threads" on page 316. All run modes other then continuous execution apply only to the selected thread. Program lines unrelated to the selected thread will be grayed out in the program window.

Threads and Just-in-time Compiled Procedures

Procedures can be executed as compiled bytecode instead of being interpreted by the HDevelop interpreter. This is described in section "Just-In-Time Compilation" on page 45. There is one notable difference when debugging threaded HDevelop programs with compiled procedures. If the program is running continuously and is then being stopped (either by user action or a breakpoint/stop instruction), the current state of the compiled procedures (variables, PC) cannot be inspected. You can still step into the procedure calls but this will cause the corresponding thread to be re-executed which may cause unexpected side-effects. Note that this is not an issue when single-stepping into the thread calls in the first place because in that case procedures are always executed by the HDevelop interpreter.

Thread Lifetime

A thread exists as long as it is still being referenced by a variable even if its execution has finished. This is necessary to reference the thread in a par_join instruction, or to set back the PC of the corresponding thread for debugging. However, the lifetime of a thread ends if the PC is manually set back to a program line before its invocation. Apart from that the lifetime of all subthreads ends when the program is reset using F2. During its lifetime a thread is listed in the Thread View / Call Stack window from where it can be selected and managed.

Finished, but still "living" threads might cause the configured thread number limit to be exceeded unintentionally when new threads are started at a later time. Also, it can have a negative effect on the run time behavior if finished threads are "killed" at the same time new threads are being executed. To explicitly "kill" finished threads, it is sufficient to reset the variables that are referencing their thread IDs as in the following example.

```
for Index := 0 to 4 by 1
  par_start <ThreadIDs.at(Index)> : gather_data()
endfor
...
convert_vector_to_tuple (ThreadIDs, Threads)
par_join (Threads)
...
ThreadIDs := {}
Threads   := []
```

Error Handling

Each thread can specify its own error handling, e.g., using `dev_set_check`('~give_error'). New subthreads inherit the error handling mode from their parent thread. Exception handling using `try` .. `catch` works only within a thread, i.e., in the main thread it is not possible to catch an exception that is thrown in a subthread.

8.10.4 Inspecting Threads

The current execution status of the program and its threads is displayed in the combined thread view/call stack window. Select `Execute` ▷ `Thread View / Call Stack` or click in the tool bar (see also `Thread View / Call Stack` (page 85)). The upper half of the window lists all existing threads while the lower half displays the call stack of the selected thread. To illustrate the interaction with this window consider the following (silly) example.

```
for Index:= 1 to 5 by 1
  par_start <ThreadIDs.at(Index - 1)> : wait_seconds(Index)
endfor
wait_seconds(2)
stop()
```

After pressing F5, the program will start five subthreads, and eventually reach the `stop` instruction, leaving some subthreads still running while others will already have finished. The corresponding thread view is displayed in figure 8.4. Note that the unfinished threads are in a stopped state because of another thread (in this case caused by the `stop` instruction in the main thread).

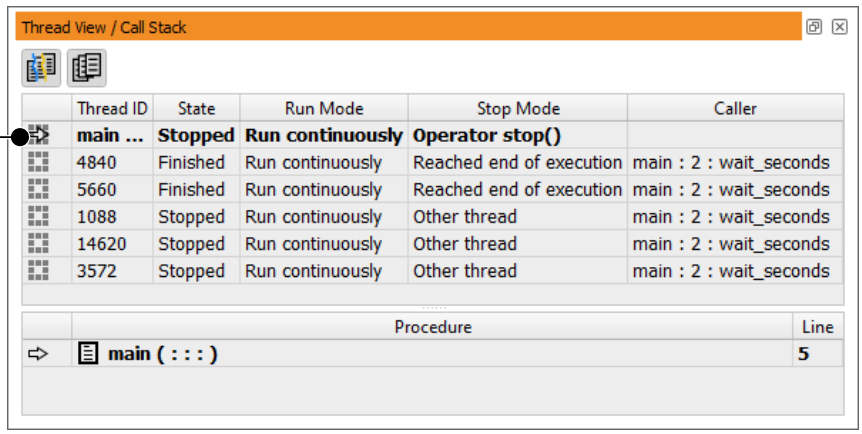


Figure 8.4: Inspecting threads.

The thread view lists the properties of all threads in a table. The status icon in the first column of each thread visualizes the current execution state. The currently selected thread (1) is marked by the arrow in the status icon and is also highlighted in bold text. The other five threads are the subthreads started from the main thread. To select another thread, double-click it in the thread view. This will also update the PC, the call stack, and the variables according to the selected thread. The active procedure of the selected thread will be displayed in the program window.

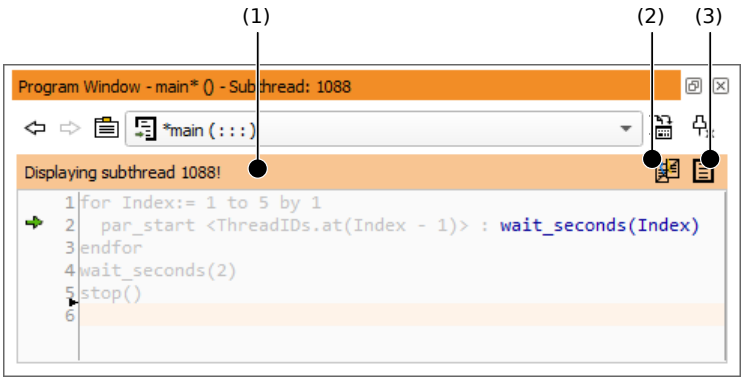


Figure 8.5: Selected subthread in the program window.

The meaning of the columns of the thread view is as follows:

Column	Description
Thread ID	The unique number assigned to the thread when it is started.
State	The current execution state of the thread.
Run Mode	The mode that the thread was started in the last time.
Stop Mode	The reason for stopping the thread.
Caller	The position (procedure and line number) from where the thread was started.
References (hidden by default)	The number of references to this thread.

Single-stepping over a program line containing a `par_start` will initialize the corresponding thread without actually starting it. To debug a specific thread, press F7 when the PC is on the corresponding `par_start` line. This will automatically make the new subthread the selected thread. If the PC is already past the invocation line, first select the thread in the thread view window. This will automatically display the correct procedure in the program window, with the PC on the first line if the thread was started by a procedure call. For threads started by an operator call like in the example above, the PC will be located on the corresponding call, and the rest of the program will be grayed out (see figure 8.5). The notification line (1) in the program window shows the thread ID of the selected subthread, and allows fast access to the thread view window (2). Clicking (3) switches back to the main thread.

8.10.5 Suspending and Resuming Threads

Threads can explicitly be suspended and resumed in the thread view window. Suspending a thread only works between operator calls, i.e., if the thread is currently running, the current operator will still be executed before the thread is frozen.

To suspend a thread, right-click on the thread entry and select `Suspend Thread`. Suspended threats are “frozen” in their current state and will defer subsequent run commands until the threads are resumed again, i.e., run commands will change the run status of the suspended threads but the actual execution will be prevented.

To resume a suspended thread again, right-click on the thread entry and select `Resume Thread`.

8.11 Summary of HDevelop Tuple Operations

Functionality	HDevelop Operation	HALCON operator
concatenation	`[t1,t2]`	`tuple_concat`
number of elements	`|t|`	`tuple_length`

select tuple elements	t1[t2]	tuple_select
select tuple slice	t[i1:i2]	tuple_select_range
select tuple elements	subset(t,i)	tuple_select
first elements	firstn(t,i)	tuple_first_n
last elements	lastn(t,i)	tuple_last_n
select by mask	select_mask(t1,t2)	tuple_select_mask
remove tuple elements	remove(t,i)	tuple_remove
lookup tuple values	find(t1,t2)	tuple_find
replace elements	replace(t1,t2,t3)	tuple_replace
find first occurrences	find_first(t1,t2)	tuple_find_first
find last occurrences	find_last(t1,t2)	tuple_find_last
unify tuple elements	uniq(t)	tuple_uniq
tuple creation	[i1:i2:i3]	tuple_gen_sequence
tuple creation	[i1:i2]	tuple_gen_sequence
tuple creation	gen_tuple_const(i1,i2)	tuple_gen_const
division	a1 / a2	tuple_div
multiplication	a1 * a2	tuple_mult
modulo	a1 % a2	tuple_mod
addition	a1 + a2	tuple_add
subtraction	a1 - a2	tuple_sub
negation	-a	tuple_neg
left shift	lsh(i1,i2)	tuple_lsh
right shift	rsh(i1,i2)	tuple_rsh
bitwise and	i1 band i2	tuple_band
bitwise xor	i1 bxor i2	tuple_bxor
bitwise or	i1 bor i2	tuple_bor
bitwise complement	bnot i	tuple_bnot
string conversion	v$s	tuple_string
string concatenation	v1 + v2	tuple_add
search character	strchr(s1,s2)	tuple_strchr
search character (reverse)	strrchr(s1,s2)	tuple_strrchr
search string	strstr(s1,s2)	tuple_strstr
search string (reverse)	strrstr(s1,s2)	tuple_strrstr
length of string	strlen(s)	tuple_strlen
first characters	str_firstn(s,i)	tuple_str_first_n
last characters	str_lastn(s,i)	tuple_str_last_n
select character	s{i}	tuple_str_bit_select
select substring	s{i1:i2}	tuple_str_bit_select
split string	split(s1,s2)	tuple_split
regular expression match	regexp_match(s1,s2)	tuple_regexp_match

regular expression replace	`regexp_replace(s1,s2,s3)`	`tuple_regexp_replace`
regular expression select	`regexp_select(s1,s2)`	`tuple_regexp_select`
regular expression test	`regexp_test(s1,s2)`	`tuple_regexp_test`
regular expression test	`s1 =~ s2`	`tuple_regexp_test`
difference	`difference(t1,t2)`	`tuple_difference`
intersection	`intersection(t1,t2)`	`tuple_intersection`
symmetric difference	`symmdiff(t1,t2)`	`tuple_symmdiff`
union	`union(t1,t2)`	`tuple_union`
less than	`t1 < t2`	`tuple_less`
greater than	`t1 > t2`	`tuple_greater`
less or equal	`t1 <= t2`	`tuple_less_equal`
greater or equal	`t1 >= t2`	`tuple_greater_equal`
equal	`t1 == t2`	`tuple_equal`
equal	`t1 = t2 (legacy)`	`tuple_equal`
not equal	`t1 != t2`	`tuple_not_equal`
not equal	`t1 # t2 (legacy)`	`tuple_not_equal`
less than (elementwise)	`t1 [<] t2`	`tuple_less_elem`
greater than (elementwise)	`t1 [>] t2`	`tuple_greater_elem`
less or equal (elementwise)	`t1 [<=] t2`	`tuple_less_equal_elem`
greater or equal (elementwise)	`t1 [>=] t2`	`tuple_greater_equal_elem`
equal (elementwise)	`t1 [==] t2`	`tuple_equal_elem`
equal (elementwise)	`t1 [=] t2 (legacy)`	`tuple_equal_elem`
not equal (elementwise)	`t1 [!=] t2`	`tuple_not_equal_elem`
not equal (elementwise)	`t1 [#] t2 (legacy)`	`tuple_not_equal_elem`
logical and	`l1 and l2`	`tuple_and`
logical xor	`l1 xor l2`	`tuple_xor`
logical or	`l1 or l2`	`tuple_or`
negation	`not l`	`tuple_not`
sine	`sin(a)`	`tuple_sin`
cosine	`cos(a)`	`tuple_cos`
tangent	`tan(a)`	`tuple_tan`
arc sine	`asin(a)`	`tuple_asin`
arc cosine	`acos(a)`	`tuple_acos`
arc tangent	`atan(a)`	`tuple_atan`
arc tangent2	`atan2(a1,a2)`	`tuple_atan2`
hyperbolic sine	`sinh(a)`	`tuple_sinh`
hyperbolic cosine	`cosh(a)`	`tuple_cosh`
hyperbolic tangent	`tanh(a)`	`tuple_tanh`
inverse hyperbolic sine	`asinh(a)`	`tuple_asinh`
inverse hyperbolic cosine	`acosh(a)`	`tuple_acosh`

inverse hyperbolic tangent	`atanh(a)`	`tuple_atanh`
exponential function	`exp(a)`	`tuple_exp`
base 2 exponential function	`exp2(a)`	`tuple_exp2`
base 10 exponential function	`exp10(a)`	`tuple_exp10`
natural logarithm	`log(a)`	`tuple_log`
base 2 logarithm	`log2(a)`	`tuple_log2`
base 10 logarithm	`log10(a)`	`tuple_log10`
power function	`pow(a1,a2)`	`tuple_pow`
ldexp function	`ldexp(a1,a2)`	`tuple_ldexp`
gamma function	`tgamma(a)`	`tuple_tgamma`
logarithm of the absolute value of the gamma function	`lgamma(a)`	`tuple_lgamma`
error function	`erf(a)`	`tuple_erf`
complementary error function	`erfc(a)`	`tuple_erfc`
minimum	`min(t)`	`tuple_min`
elementwise minimum	`min2(t1,t2)`	`tuple_min2`
maximum	`max(t)`	`tuple_max`
elementwise maximum	`max2(t1,t2)`	`tuple_max2`
sum function	`sum(t)`	`tuple_sum`
mean value	`mean(a)`	`tuple_mean`
standard deviation	`deviation(a)`	`tuple_deviation`
cumulative sum	`cumul(a)`	`tuple_cumul`
median	`median(a)`	`tuple_median`
element rank	`select_rank(a,i)`	`tuple_select_rank`
square root	`sqrt(a)`	`tuple_sqrt`
cube root	`cbrt(a)`	`tuple_cbrt`
hypotenuse	`hypot(a,b)`	`tuple_hypot`
radians to degrees	`deg(a)`	`tuple_deg`
degrees to radians	`rad(a)`	`tuple_rad`
integer to real	`real(a)`	`tuple_real`
real to integer	`int(a)`	`tuple_int`
real to integer	`round(a)`	`tuple_round`
absolute value	`abs(a)`	`tuple_abs`
floating absolute value	`fabs(a)`	`tuple_fabs`
ceiling function	`ceil(a)`	`tuple_ceil`
floor function	`floor(a)`	`tuple_floor`
fractional part	`fmod(a1,a2)`	`tuple_fmod`
elementwise sign	`sgn(a)`	`tuple_sgn`
sort elements	`sort(t)`	`tuple_sort`
sort elements (returns index)	`sort_index(t)`	`tuple_sort_index`
reverse element order	`inverse(t)`	`tuple_inverse`

test for numeric value	`is_number(v)`	`tuple_is_number`
string to number	`number(v)`	`tuple_number`
environment variable	`environment(s)`	`tuple_environment`
convert strings of length 1 into character codes	`ord(a)`	`tuple_ord`
inverse of ord(a)	`chr(a)`	`tuple_chr`
convert strings into character codes	`ords(s)`	`tuple_ords`
inverse of ords(s)	`chrt(i)`	`tuple_chrt`
random number	`rand(a)`	`tuple_rand`
test for handle values	`is_handle(t)`	`tuple_is_handle`
test for integer values	`is_int(t)`	`tuple_is_int`
test for mixed values	`is_mixed(t)`	`tuple_is_mixed`
test for numerical values	`is_number(t)`	`tuple_is_number`
test for real values	`is_real(t)`	`tuple_is_real`
test for string values	`is_string(t)`	`tuple_is_string`
test for valid handles	`is_valid_handle`	`tuple_is_valid_handle`
get semantic type	`sem_type(t)`	`tuple_sem_type`
get type value	`type(t)`	`tuple_type`
test for handle values (elementwise)	`is_handle_elem(t)`	`tuple_is_handle_elem`
test for integer values (elementwise)	`is_int_elem(t)`	`tuple_is_int_elem`
test for real values (elementwise)	`is_real_elem(t)`	`tuple_is_real_elem`
test for string values (elementwise)	`is_string_elem(t)`	`tuple_is_string_elem`
get semantic type (elementwise)	`sem_type_elem(t)`	`tuple_sem_type_elem`
get type value (elementwise)	`type_elem(t)`	`tuple_type_elem`

8.12 HDevelop Error Codes

21000 HALCON operator error
21001 User defined exception (’throw’)
21002 User defined error during execution
21003 User defined operator does not implement execution interface
21010 HALCON license error
21011 HALCON startup error
21012 HALCON operator error
21020 Format error: file is not a valid HDevelop program or procedure
21021 File is no HDevelop program or has the wrong version
21022 Protected procedure could not be decompressed
21023 Protected procedure could not be compressed and encrypted for saving
21024 Format error: file is not a valid HDevelop program
21025 Format error: file is not a valid HDevelop procedure
21026 Format error: file is not a valid HDevelop procedure library

21030 The program was modified inconsistently outside HDevelop.

21031 The program was modified outside HDevelop: inconsistent procedure lines.

21032 The program was modified outside HDevelop: unmatched control statements

21033 Renaming of procedure failed

21034 Locked procedures are not supported for the selected action.

21034 Password protection/locked procedures

21035 Procedures with advanced language elements are not supported for the selected action.

21035 Parallel execution statements, iconic assignments, or iconic comparisons

21036 Procedures with vector variables are not supported for the selected action.

21036 Vector variables

21040 Unable to open file

21041 Unable to read from file

21042 Unable to write to file

21043 Unable to rename file

21044 Unable to open file: invalid file name

21050 For this operator the parallel execution with par_start is not supported

21051 Thread creation failed

21052 Thread creation failed: exceeded maximum number of subthreads

21060 Iconic variable is not instantiated

21061 Control variable is not instantiated (no value)

21062 Wrong number of control values

21063 Wrong value type of control parameter

21064 Wrong value of control parameter

21065 Control parameter does not contain a variable

21066 Control parameter must be a constant value

21067 Wrong number of control values in condition variable

21068 Wrong type: Condition variable must be an integer or boolean

21070 Variable names must not be empty

21071 Variable names must not start with a number

21072 Invalid variable name

21073 Invalid name for a control variable: the name is already used for an iconic variable

21074 Invalid name for an iconic variable: the name is already used for a control variable

21075 An iconic variable is used in the wrong context: a control variable or a vector is expected

21076 A control variable is used in the wrong context: an iconic variable or a vector is expected

21077 An iconic vector variable is used in the wrong context: a control variable or a single value is expected

21078 A control vector variable is used in the wrong context: an iconic variable or a single value is expected

21080 For loop variable must be a number

21081 Step parameter of for loop must be a number

21082 End parameter of for loop must be a number
21083 Variable names must not be a reserved expression
21084 Case label value has already appeared in switch block
21085 Default label has already appeared in switch block
21086 The type of the variable could not be determined (no proper type definition found)
21087 The type of the variable could not be determined (conflicting type definitions found)
21088 The type of the variable could not be determined (conflicting type definitions found)
21090 A global variable with the specified name but a different type is already defined
21091 Access to an unknown global variable
21092 Access to an invalid global variable
21093 Invalid name for a global variable: the name is already used for a procedure parameter
21100 Access to an erroneous expression
21101 Wrong index in expression list
21102 Empty expression
21103 Empty expression argument
21104 Syntax error in expression
21105 Too few function arguments in expression
21106 Too many function arguments in expression
21107 The expression has no return value
21108 The expression has the wrong type
21109 The expression has the wrong type
21110 The expression has the wrong type: iconic expression expected
21111 The expression has the wrong type: iconic expression expected
21112 The expression has the wrong type: control expression expected
21113 The expression has the wrong type: control expression expected
21114 The expression has the wrong vector dimension
21115 The expression has the wrong vector dimension
21116 Vector expression expected
21117 Vector expression expected
21118 Single or tuple value expression expected instead of a vector
21119 Single or tuple value expression expected instead of a vector
21120 Expression expected
21121 lvalue expression expected
21122 Variable expected
21123 Unary expression expected
21124 Expression list expected
21125 Function arguments in parentheses expected
21126 One function argument in parentheses expected
21127 Two function arguments in parentheses expected
21128 Three function arguments in parentheses expected

21129 Four function arguments in parentheses expected

21130 Five function arguments in parentheses expected

21131 Right parenthesis ')' expected

21132 Right curly brace '}' expected

21133 Right square bracket ']' expected

21134 Unmatched right parenthesis ')' found

21135 Unmatched right curly brace '}' found

21136 Unmatched right square bracket ']' found

21137 Second bar 'l' expected

21138 Function name expected before parentheses

21139 Unterminated string detected

21140 Invalid character in an expression identifier detected

21141 Parameter expression expected

21142 Parameter expression is not executable

21143 Vector method after . expected

21144 Vector method 'at' after . expected

21145 Modifying vector methods are not allowed within parameters

21200 Syntax error in operator expression

21201 Identifier (operator or variable name) expected

21203 Syntax error in parameter list

21204 Parenthesis expected

21205 No parenthesis expected

21206 List of parameters in parenthesis expected

21207 Wrong number of parameters

21208 Unexpected characters at end of line

21209 Assign operator ':=' expected

21210 Expression after assign operator ':=' expected

21211 Expression in brackets '[ ]' for the assign_at index expected

21212 In for statement, after keyword 'by' expression for parameter 'Step' expected

21213 In for statement, after keyword 'to' expression for parameter 'End' expected

21214 In for statement, after assign operation (':=') expression for parameter 'Start' expected

21215 In for statement, after 'for .. := .. to ..' keyword 'by' expected

21216 In for statement, after 'for .. := ..' keyword 'to' expected

21217 In for statement, assign operation ':=' for initializing the index variable expected

21218 After 'for' keyword, assignment of 'Index' parameter expected

21219 In for statement, error after 'by' keyword in expression of parameter 'Step'

21220 In for statement, error after 'to' keyword in expression of parameter 'End' or the following 'by' keyword

21221 In for statement, error after assignment operation (':=') in expression of parameter 'Start' or the following 'to' keyword

21222 In for statement, invalid variable name in parameter 'Index' or error in the following assignment operation (':=')

21223 for statement not complete

21224 In for statement, space after 'for' expected

21225 In for statement, space after 'to' expected

21226 In for statement, space after 'by' expected

21227 Wrong type: The switch statement requires an integer value as parameter

21228 Wrong type: The case statement requires a constant integer value as parameter

21229 At the end of the case and the default statement a colon is expected

21230 Unknown operator or procedure

21231 Qualifier 'par_start' before ':' or '<ThreadID>' expected

21232 '<ThreadID>' variable in angle brackets after 'par_start' expected

21233 ThreadID variable after 'par_start<' expected

21234 Closing angle bracket ('>') after 'par_start<ThreadID' expected

21235 Colon (':') after 'par_start<ThreadID>' expected

21236 Operator or procedure call after 'par_start :' expected

21900 Internal value has the wrong type

21901 Internal value is not a vector

21902 Index into internal value is out of range

21903 Internal value is not instantiated

22000 Internal operation in expression failed

22001 Internal operation in a constant expression failed

22010 Parameters are tuples with different size

22011 Division by zero

22012 String exceeds maximum length

22100 Parameter is an empty tuple

22101 Parameter has more than one single value

22102 Parameter is not a single value

22103 Parameter has the wrong number of elements

22104 Parameter contains undefined value(s)

22105 Parameter contains wrong value(s)

22106 Parameter contains value(s) with the wrong type

22200 First parameter is an empty tuple

22201 First parameter has more than one single value

22202 First parameter is not a single value

22203 First parameter has the wrong number of elements

22204 First parameter contains undefined value(s)

22205 First parameter contains wrong value(s)

22206 First parameter contains value(s) with the wrong type

22300 Second parameter is an empty tuple

22301 Second parameter has more than one single value

22302 Second parameter is not a single value

22303 Second parameter has the wrong number of elements

22304 Second parameter contains undefined value(s)

22305 Second parameter contains wrong value(s)

22306 Second parameter contains value(s) with the wrong type

22400 Calling context was not set

22401 Accessing an invalid calling context

22402 Error while accessing calling context data

22500 Communication with external application failed

22501 Debug session with external application no longer valid

22502 An unexpected error occured in the external application

22503 Wrong password to unlock procedure in external application

23100 The generic parameter value is unknown

23101 The generic parameter name is unknown

30000 User defined exception

8.13 Emergency Backup

In case HDevelop ever crashes during program execution, the current program is saved in a temporary location. After restarting HDevelop, you can restore the backup file to continue your application. The exact location of the data depends on the operating system you are using:

Linux/macOS `/tmp/hdevelop_`*`login`* (substitute *login* with your login name)

Windows `%TEMP%\hdevelop`

Chapter 9

Remote Debugging

HDevelop supports the debugging of HDevelop code in stand-alone applications. This is referred to as "remote debugging". An external application can execute HDevelop procedures using HDevEngine. If it explicitly enables remote debug connections, HDevelop may attach to this application. Once the connection has been established, HDevelop operates in a special debugging mode (see figure 9.1). How to enable remote debugging is described in the Programmer's Guide, section 25.2 on page 193. The following sections describe the HDevelop side of remote debugging.

9.1 Requirements

To be able to use remote debugging, both the external application and HDevelop must be based on the same version of HALCON. Furthermore, the version of HDevelop must at least be as "large" as that of HDevEngine, i.e., if the external application uses HDevEngine XL, you cannot use HDevelop non-XL.

Please note that cross-platform debugging is supported, e.g., you can debug an external application running on Windows using HDevelop on Linux.

9.2 Attaching to an External Application

- Make sure the external application is running and has remote debugging enabled as described in the Programmer's Guide, section 25.2 on page 193.
- Run HDevelop.
- Click `Execute` ▷ `Attach To Process...`.

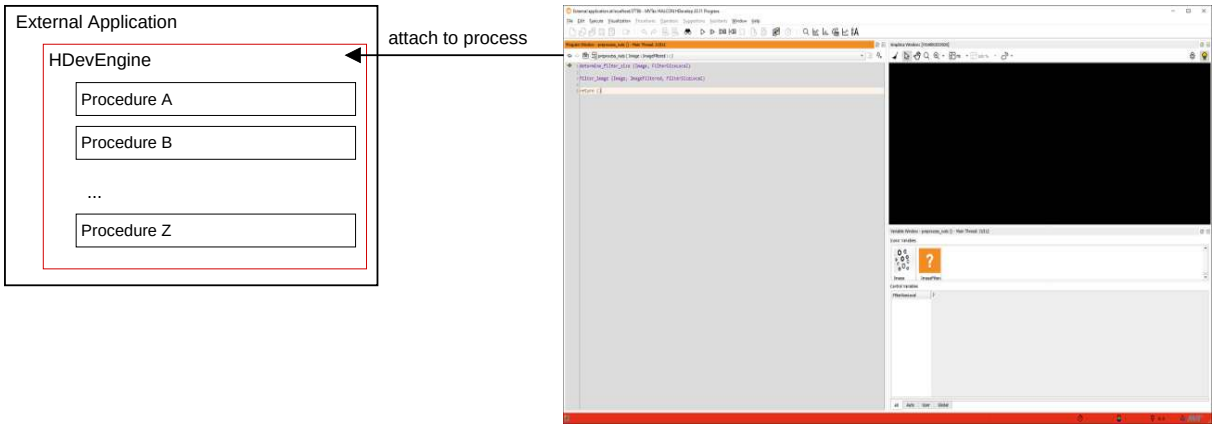


Figure 9.1: Remote debugging.

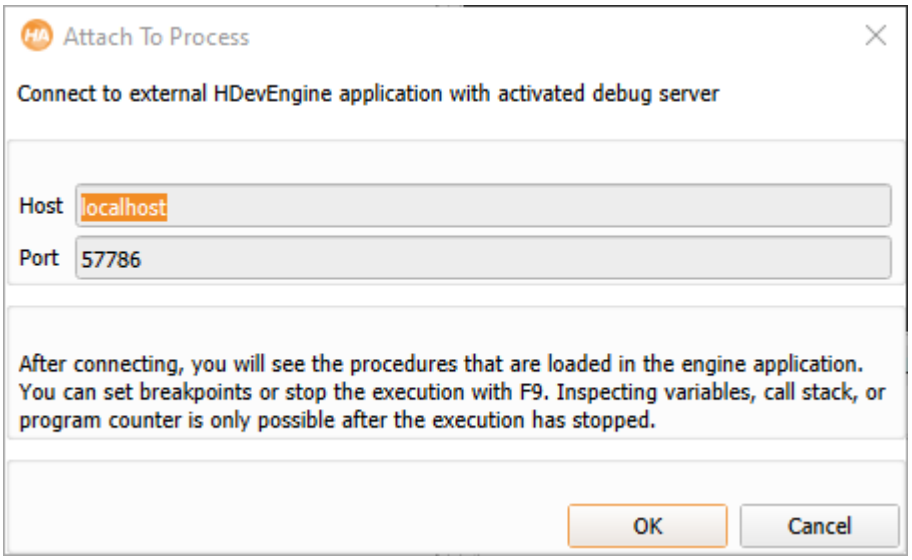


Figure 9.2: Attaching to an external application.

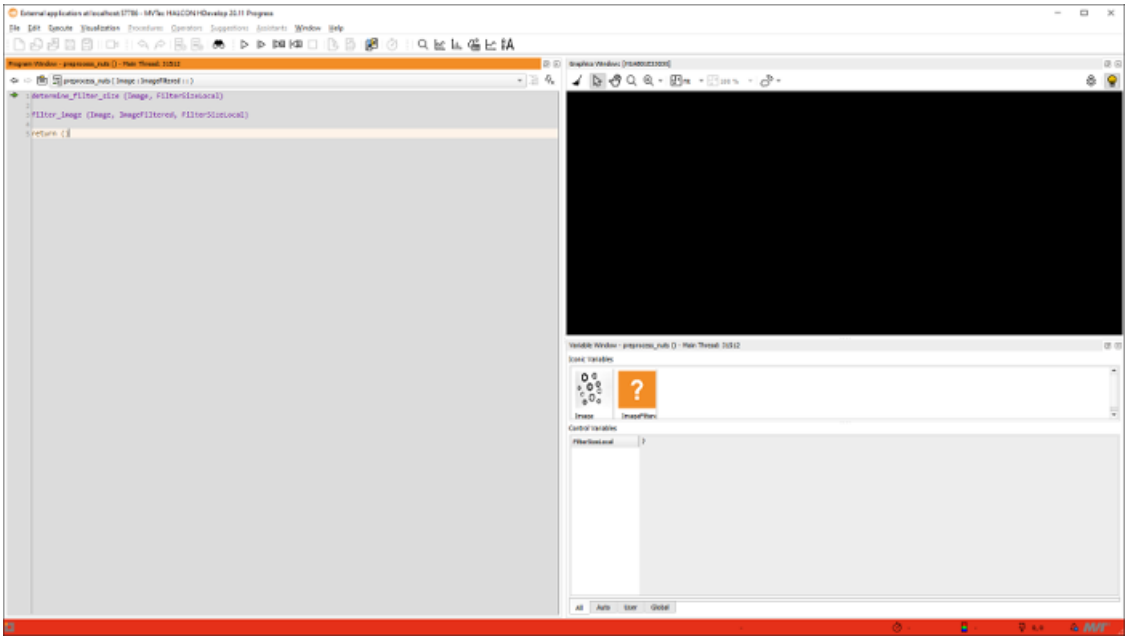


Figure 9.3: HDevelop running in remote debug mode.

- Enter the host name and connection port (see figure 9.2), and, optionally, the password. The default host name (localhost) can be used if both the external application and HDevelop run on the same machine.

Please note that your firewall must be configured to allow connections on the specified port. The default port number is 57786, but this can be changed in the external application. If a debug session is already active on the given machine, you can choose to take over the existing session. This will terminate the previous session.

After connecting, large parts of the user interface of HDevelop are grayed out: This applies to the functionality related to the normal operation of HDevelop, such as procedures, operators, assistants, and preferences. The special debug mode is indicated in the window title and by the red background color of the status bar (see figure 9.3). The background color of the program window also changes to indicate the read-only nature of the remote debug mode. This color can be customized in the preferences in normal mode.

9.3 Debugging

Remote debugging is in most parts similar to "local" debugging. The procedure execution is controlled by execution commands (e.g., `Stop`, `Step Over`, `Step Into` etc.). Debugging is only possible in stopped mode. If HDevEngine is in run mode (i.e., status bar reads "Running... External application"), the following applies:

- No PC is displayed.
- Variables are uninitialized.
- Application threads cannot be selected, and the call stack is empty.

Independent of whether run or stop mode is active, the loaded procedures can be viewed in the program window. As usual, breakpoints may be set on program lines and variables. Activated breakpoints persist as long as the external application is running. In contrast, deactivated breakpoints disappear on disconnect. HDevEngine stops at breakpoints only if HDevelop is actually attached.

Pressing `Stop` causes HDevelop to wait for the external application to execute code in HDevEngine. In stop mode you can execute program lines step-by-step and view intermediate results in both the variable and the graphics window. Other than that, the execution cannot be influenced:

- The PC cannot be repositioned.
- The values of variables cannot be altered.
- The procedures cannot be reset.
- The program code cannot be modified (see also section "Limitations" on page 330).

If the application executes procedure calls in the GUI thread, attaching a debugger and stopping the execution will freeze the application GUI.

9.4 Handling of Procedures

The program window provides access to all procedures loaded by HDevEngine during the debugging session. Unloading procedures will not cause the corresponding procedures to be removed from the list. This is to ensure usability even if the application loads/unloads procedures in a loop. You can disconnect and then reconnect the debugger to refresh the list.

The external application can load multiple copies of the same procedure. In HDevelop however, duplicate procedure names are shown only once. As a result, breakpoints set in a duplicate procedure will apply to all loaded copies. If another copy is loaded at a later time, the breakpoint will not be active for it.

9.5 Handling of Protected Procedures

Debugging protected procedures is also supported in HDevelop. If such a procedure is selected in the program window, you will be prompted for a password to unlock it. Unlocking protected procedures only has an effect inside HDevelop. The state of the procedures in the external application does not change. The unlocked state is only valid for the current debug session. If you or another user reconnects to the same process, the procedures will be locked again. There is no explicit way to relock an unlocked procedure during debugging. You can stop debugging and attach again to achieve this.

9.6 Error Handling

The behavior of handling errors in the HDevelop code of the external application depends on whether or not the exception is handled in the program code. If the exception is unhandled, the error message will be shown in HDevelop, and the application will wait for a `Step` or `Run` signal before raising `HDevEngineException` itself. If the exception is handled, there is no way to stop the execution.

9.7 Threading

Application threads are supported. After sending a stop signal to the application (`Stop` or F9), a random thread will be selected in HDevelop. Stepping over the last line of a procedure switches threads if another stopped HDevEngine thread is waiting. In case HDevEngine stops due to a breakpoint or an error, HDevelop will automatically switch to the corresponding thread.

The normal thread view in the dialog `Execute` ▷ `Thread View / Call Stack` is not available, but threads can be switched from this dialog by clicking the button `Show Application Threads...`.

Please note that subthreads started with `par_start` cannot be debugged.

9.8 Terminating a Remote Debug Session

Click `Execute` ▷ `Stop Debugging` to terminate the debugging session. The debug server will continue to run and accept further connections as long as HDevEngine keeps running. In case the external application is currently stopped at the moment the session is terminated, HDevelop will offer to resume the execution before disconnecting. Otherwise, the thread executing the procedure calls will be frozen until you reconnect and continue running it.

The debugging session will also be terminated if the external application itself stops the debug server or if the session is taken over by a subsequent connection.

9.9 Limitations

- Procedures cannot be edited in HDevelop.
- Name conflicts between procedures are not supported.
- Semantic types and special inspection widgets (e.g., 3D object models) are not supported.
- The graphics window is not always updated after each `Step`. As a workaround, you can double-click an iconic variable to display its current value.

Further limitations of remote debugging from the HDevEngine side are listed in the Programmer's Guide, section 25.2.4 on page 195.

Chapter 10

Code Export

The idea of code export or code generation is as follows: After developing a program according to the given requirements it has to be translated into its final environment. For this, the program is transferred into another programming language that can be compiled.

For C++ and C# developers there is a recommended approach for integrating HDevelop code into their own projects. Instead of exporting an entire HDevelop program to C++ or C#, a library project can be exported, which can be directly integrated into your own projects. The procedure can be called as simple as in HDevelop. The library project export is described in section 10.1.

Alternatively, HDevelop allows you to export an entire HDevelop program to the programming languages C++, Visual Basic .NET, C#, and C by writing the corresponding code to a file. This approach is described in section 10.2 on page 335.

10.1 Exporting Library Projects

Exporting a library project makes it easy to integrate HDevelop code into your projects. You can export a project containing either a selected procedure library or the local procedures of the current program. Watch our tutorial video and learn how to use the library project export to integrate HDevelop code into an existing Visual Studio application.

The following target languages are supported:

- C++
- C#

The project will be exported to a project directory with the following components:

- Wrapper code in the target language
- CMake file to build the project (CMake is freely available at `https://cmake.org/`)
- the selected procedure library (or HDevelop program containing the local procedures)

Additionally, for C++ the following files are generated:

- C++ header file
- CMake functions to resolve the HALCON environment

10.1.1 Requirements

10.1.1.1 C++

- CMake version 3.3 or higher is required to build the exported project.
- The exported code makes use of some C++11 elements. These will not work with old compilers. The minimum requirements for the target compilers are: VS 2013, GCC 4.8.6, clang 3.9, and ICC 16. Older compilers might work but are not supported.

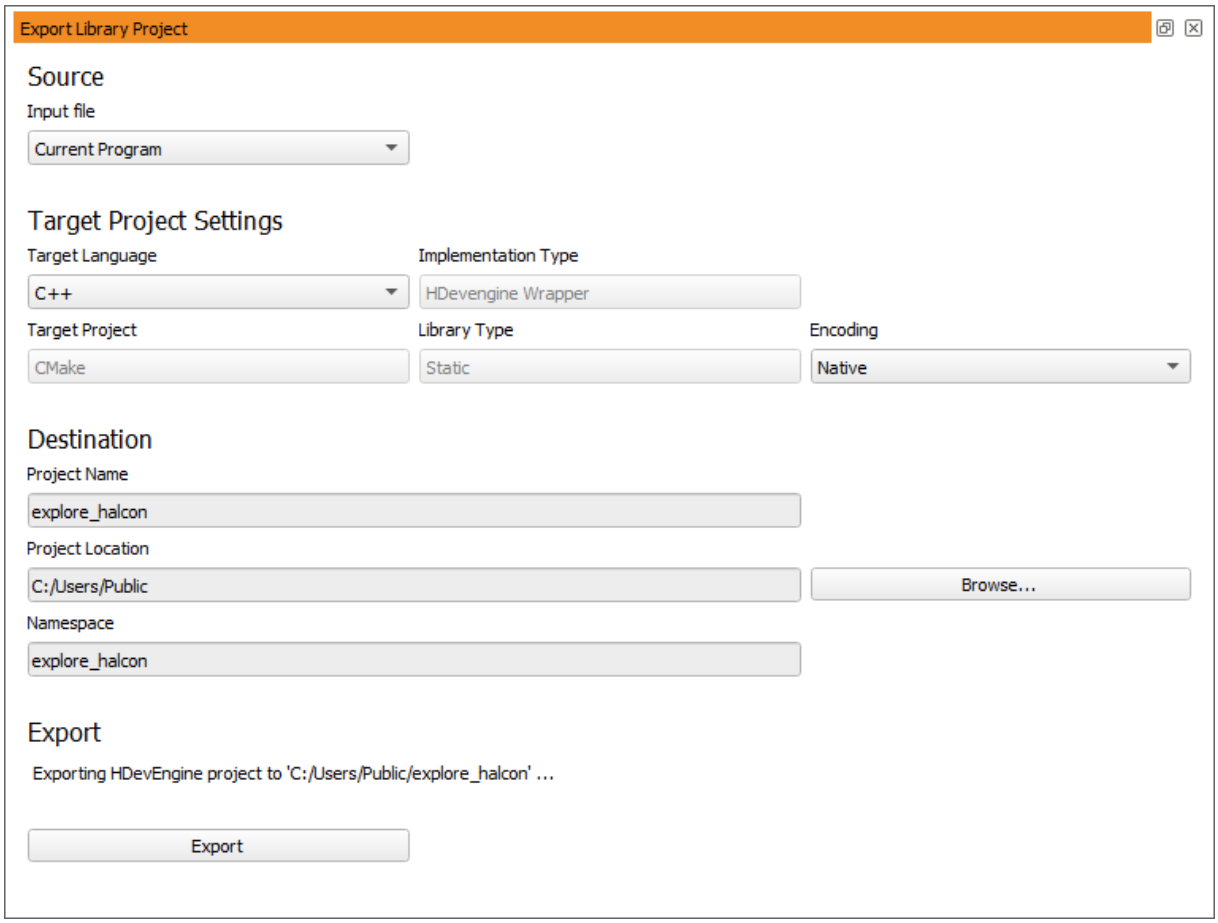


Figure 10.1: Export Library Project dialog.

10.1.1.2 C#

- CMake version 3.9 or higher is required to build the exported project.
- Only the project generators for VS 2010 and above are supported.

10.1.2 Project Preparation

Before you can export a library project, you need to create a procedure library containing all procedures that will be required in your application. Alternatively, create a HDevelop program containing the required functionality in local procedures. In the latter case, the main procedure will not be exported.

The declaration of public and private procedures defines the interface for your application. Public procedures can be called from your application. Private procedures can only be called from procedures within the library or HDevelop program.

10.1.3 Exporting a Library Project

Select `File` ▷ `Export Library Project` to open the export dialog (see figure 10.1).

`Source`

`Input file`

Select a procedure library to be exported. If `Current Program` is selected, the local procedures of the current program will be exported.

If the selected file becomes unavailable (e.g., because the procedure path settings have been changed in the meantime), an error message is displayed.

`Target Project Settings`

`Target Language`

Specify the target language of the exported project (C++ or C#).

`Encoding`

Specify the text encoding of the exported project (either `Native` or `UTF-8`). Selecting `Native` will use the current code page on Windows and UTF-8 on other platforms.

Currently, the other target project settings can not be modified. The library project will be exported to a CMake-based project. The exported library will be accessible through a HDevEngine wrapper. Building the exported project will result in a static library that can be linked to an application.

`Destination`

`Project Name`

Enter the project name. The project name specifies the name of a subdirectory of the project location, which will contain all generated files. If the project name is not a valid file name, an error message is displayed.

`Project Location`

Specify the destination directory of the exported project. Relative paths are allowed – the path will be relative to the current working directory (see `get_current_dir`).

`Namespace`

Specify the namespace of the exported library project. The namespace must be valid for the target language. Otherwise, an error message is displayed.

`Export`

Shows the final location of the library project based on the settings in `Destination`. After specifying the project properties, click the button `Export` to generate the library project. If you export from HDevelop XL, you will obtain an XL variant of the library project.

Previous exports at the same location will be **silently overwritten**.



The project directory will contain the following files:

- `CMakeLists.txt`
  CMake control file.
- `cmake` (C++ only)
  Directory containing CMake code for the resolution of HALCON and HDevEngine.
- `res_`*`project name`*
  Directory containing the exported procedure library or HDevelop program.
- `source`
  Directory containing the exported wrapper code and, for C++, a header file. In case of exporting `Current Program` with no local procedure at all, a project directory will still be created. However, there will not be any callable procedures/resources inside. If you experience this, try putting the relevant code of the main procedure into a local procedure and repeat the library export.

10.1.4 Using the Exported Library Project

10.1.4.1 Converting to Visual Studio

The CMake project can easily be converted to a Visual Studio project via the “Generate” option. The following example generates a project for Visual Studio 2015 (64-bit), but other generators are available as well.

- Open a command prompt
- Change to the exported project directory (containing `CMakeLists.txt`).

- Generate the Visual Studio project:

```
cmake . -G "Visual Studio 14 2015 Win64"
```

Run `cmake -G` to get a list of available generators.

10.1.4.2 Integrating the Project Into Your Own Projects

CMake

Move the exported project into a subdirectory of your CMake project and use it in your application project(s). Assuming the exported library has the name `MyLibrary`, a minimal `CMakeLists.txt` file for an executable project named `ExportTest` using the library look as follows:

```
project(ExportTest)
add_subdirectory(MyLibrary)
add_executable(ExportTest main.cpp)
target_link_libraries(ExportTest MyLibrary)
```

Visual Studio

- Add the library.
- Add the includes.

When linking the generated library project into the user application, both `hdevenginecpp.lib` and `halconcpp.lib` (C++), or `hdevenginedotnet.dll` and `halcondotnet.dll` (C#) must be added to the project as well. This is, because the HDevEngine symbols are not included in the exported library.

10.1.4.3 Updating the Exported Library Project



If you are working with CMake to configure and build your application, you can (re-)export directly into your application project file structure. However, be aware that HDevelop will **silently overwrite** any existing files in the target location.

10.1.4.4 Using the Exported Library in C++

- Use the function `SetResourcePath` to set the directory containing the exported scripts.
- Include the exported header file.
- Call the procedures.

Note that the procedures only use parameters of type `HObject`, `HTuple`, `HObjectVector`, and `HTupleVector`.

Use a Different Resource Location

`SetResourcePath` can be used in the application to specify a custom location of the HDevelop script or procedure library.

Working With Additional Procedure Libraries

If the exported procedure libraries use procedures contained in further procedure libraries, these additional procedure libraries will not be copied over to the project export target directory. You have to manually take care of copying these additional files or add procedure library paths pointing to the used additional procedure libraries in your application code when using the exported library.

To add additional procedure paths to HDevEngine, use the following call:

```
HDevEngineCpp::HDevEngine().AddProcedurePath(...)
```

10.1.4.5 Using the Exported Library in C#

- Use the function `ResourcePath` to set the directory containing the exported scripts.
- Call the procedures, e.g.,

```
MyNamespace.MyLibrary.my_procedure()
```

Note that the procedures only use parameters of type `HObject`, `HTuple`, `HObjectVector`, and `HTupleVector`.

Use a Different Resource Location

`ResourcePath` can be used in the application to specify a custom location of the HDevelop script or procedure library.

Working With Additional Procedure Libraries

If the exported procedure libraries use procedures contained in further procedure libraries, these additional procedure libraries will not be copied over to the project export target directory. You have to manually take care of copying these additional files or add procedure library paths pointing to the used additional procedure libraries in your application code when using the exported library.

To add additional procedure paths to HDevEngine, use the following call:

```
HalconDotNet::HDevEngine().AddProcedurePath(...)
```


10.2 Exporting entire HDevelop Programs

HDevelop allows you to export a developed HDevelop program to the programming languages C++, Visual Basic .NET, C#, and C. The following sections describe the general steps of program development using this feature for the languages

- C++ (section 10.2.1),
- C# - HALCON/.NET (section 10.2.2 on page 338),
- Visual Basic .NET - HALCON/.NET (section 10.2.3 on page 339),
- C (section 10.2.4 on page 341),

including some language-specific details of the code generation and optimization aspects.

Because HDevelop does more than just execute a HALCON program, the behavior of an exported program will differ in some points from its HDevelop counterpart. A prominent example is that in HDevelop, all results are automatically displayed, while in the exported programs you have to insert the corresponding display operators explicitly. Section 10.2.5 on page 342 describes these differences in more detail.

10.2.1 Code Generation for C++

This section describes how to create a HALCON application in C++, starting from a program developed in HDevelop.

10.2.1.1 Basic Steps

Program Export

The first step is to export the program using the menu `File` ▷ `Export...`. Here, select the language (`C++ - HALCON/C++`) and save it to a file. A file will be created that contains the HDevelop program as C++ source code. For every HDevelop procedure except the main procedure, the exported file contains a C++ procedure with the corresponding name. Iconic input and output parameters of a procedure are declared as `HObject` and `HObject*`, respectively, while control input and output parameters are declared as `HTuple` and `HTuple*`, respectively. All procedures are declared at the beginning of the file. The program body of the HDevelop main procedure is contained

in a procedure action() which is called in the function main(). action() and main() can be excluded from compilation by inserting the instruction #define NO_EXPORT_MAIN at the appropriate position in the application. Using the instruction #define NO_EXPORT_APP_MAIN only the main() procedure is excluded from compilation. This can be useful if you want to integrate exported HDevelop code into your application through specific procedure interfaces. In that case, there is typically no need to export the main procedure, which was probably used only for testing the functionality implemented in the corresponding 'real' procedures.

Besides the program code, the file contains all necessary #include instructions. All local variables (iconic as well as control) are declared in the corresponding procedures. Iconic variables belong to the class HObject and all other variables belong to HTuple.

Compiling and Linking in Windows Environments

The next step is to compile and link this new program. In the Windows environment, Visual C++ is used for the compiling and linking. Example projects can be found in the directory %HALCONEXAMPLES%\cpp.

If you want to use HALCON XL, you have to include the libraries halconxl.lib/.dll and halconcppxl.lib/.dll instead of halcon.lib/.dll and halconcpp.lib/.dll in your project (see the Programmer's Guide, chapter 7 on page 49, for more details).

Compiling and Linking in Linux Environments

To compile and link the new program (called, e.g., test.cpp) under Linux, you can use the example makefile, which can be found in the directory $HALCONEXAMPLES/cpp/console/makefiles, by calling

```
make PROG=test
```

Alternatively, you can set the variable PROG in makefile to test and then just type make.

You can link the program to the HALCON XL libraries by calling

```
make PROG=test XL=1
```

or just type make XL=1 if you set the variable PROG as described above.

For more details see the Programmer's Guide, chapter 7 on page 49.

Compiling and Linking in macOS Environments

In the macOS environment, Xcode is used for the compiling and linking. Example projects can be found in the directory /Users/Shared/Library/Application Support/HALCON-21.05/examples/cpp.

To compile and link the new program (called, e.g., test.cpp) under macOS from the command line, you can use the example makefile, which can be found in the directory $HALCONEXAMPLES/cpp/console/makefiles, by calling

```
make PROG=test
```

Alternatively, you can set the variable PROG in makefile to test and then just type make.

You can link the program to the HALCON XL libraries by calling

```
make PROG=test XL=1
```

or just type make XL=1 if you set the variable PROG as described above.

For more details see the Programmer's Guide, chapter 7 on page 49.

10.2.1.2 Optimization

Optimization might be necessary for variables of class `HTuple`. This kind of optimization can either be done in HDevelop or in the generated C++ code. In most cases optimization is not necessary if you program according to the following rules.

1. Using the tuple concatenation, it is more efficient to extend a tuple at the "right" side, like:

   ```
   T := [T,New]
   ```

   because this can be transformed to

   ```
   T = T.TupleConcat(New);
   ```

   in C++ and requires no creation of a new tuple, whereas

   ```
   T := [New,T]
   ```

   which is translated into

   ```
   T = HTuple(New).TupleConcat(T);
   ```

   would need the creation of a new tuple.

2. Another good way to modify a tuple is the operator `assign_at` (see section 8.5.2 on page 287). In this case HDevelop code like

   ```
   T[i] := New
   ```

   can directly be translated into the efficient and similar looking code

   ```
   T[i] = New;
   ```

10.2.1.3 Used Classes

There are only two classes that are used: `HTuple` for control parameters and `HObject` for iconic data. There is no need for other classes as long as the program has the same functionality as in HDevelop. When editing a generated program you are free to use any of the classes of HALCON/C++ to extend the functionality.

10.2.1.4 Limitations and Troubleshooting

Besides the restrictions mentioned in this section and in section 10.2.5 on page 342, please also check the description of the HDevelop operators in section 6.2.6.2 on page 102.

Exception Handling

In HDevelop, every exception normally causes the program to stop and report an error message in a dialog window. This might not be useful in C++. The standard way to handle this in C++ is by using the `try/catch` mechanism. This allows you to access the reason for the exception and to continue accordingly. HDevelop supports exception handling using `try/catch` blocks, which is exported transparently to C++. Therefore, it is the recommended method of handling errors in HDevelop programs that are going to be exported to C++.

For HDevelop programs containing error handling using `(dev_)set_check("~give_error")` the corresponding code is automatically included. Every operator call, for which it is assumed that the HALCON error mechanism is turned off, is enclosed in a `try` block followed by a `catch` block. The latter handles the exception and assigns the corresponding HALCON error number to the error variable activated by `dev_error_var` or to a local error variable, otherwise.

Please note that a call of `(dev_)set_check("~give_error")` has no influence on the operator call. The exception will *always* be raised.

10.2.2 Code Generation for C# (HALCON/.NET)

This section describes how to create a HALCON application in C#, starting from a program developed in HDevelop. HALCON can be used together with C# based on the .NET interface of HALCON. A detailed description of this interface can be found in the Programmer's Guide, part III on page 59.

10.2.2.1 Basic Steps

Export

The first step is to export the program using the menu `File` ▷ `Export...`. Here, select the language (`C# - HALCON/.NET`) and save it to file. The result is a new file with the given name and the extension "`.cs`".

The C# Template

If the file has been exported using the option `Use Export Template`, it is intended to be used together with the predefined C# project that can be found in the directory

```
%HALCONEXAMPLES%\c#\HDevelopTemplate
```

This project contains a form with a display window (`HWindowControl`) and a button labeled `Run`. Add the file generated by HDevelop to the project in the Solution Explorer (`Add Existing Item`). Now the project is ready for execution: Run the project and then press the button `Run` on the form, which will call the exported code.

Additional information about using the template can be found in the Programmer's Guide, section 12.3.1 on page 84.

10.2.2.2 Program Structure

If the program has been exported using the option `Use Export Template`, the file created by HDevelop contains a subroutine with the corresponding name for every HDevelop procedure except the main procedure, which is contained in the subroutine `action()`. Otherwise, the file is exported as a standalone application. Iconic input and output parameters of a procedure are passed as `HObject` and `out HObject`, respectively, while control input and output parameters are passed as `HTuple` and `out HTuple`, respectively. The subroutine `RunHalcon()` contains a call to the subroutine `action()` and has a parameter `Window`, which is of type `HTuple`. This is the link to the window on the form to which all output operations are passed. In addition, another subroutine is created with the name `InitHalcon()`. This subroutine applies the same initializations that HDevelop performs.

Most of the variables (iconic as well as control) are declared locally inside the corresponding subroutines. Iconic variables belong to the class `HObject` and control variables belong to `HTuple`.

Depending on the program, additional subroutines and variables are declared.

Stop

The HDevelop operator `stop` is translated into a subroutine in C# that creates a message box. This message box causes the program to halt until the button is pressed.

Used Classes

There are only four classes/types that are used: `HTuple` for control parameters and `HObject` for iconic data. In addition, there is the class `HWindowControl`. It is used inside the project for the output window and a variable of class `HTuple` directs the output to this window. Finally, the class `HOperatorSet` is used as a container for all HALCON operators. There is no need for other classes as long as the program has the same functionality as in HDevelop. When editing a generated program you are free to use any of the classes of HALCON/.NET to extend the functionality.

10.2.2.3 Limitations and Troubleshooting

Besides the restrictions mentioned in this section and in section 10.2.5 on page 342, please also check the description of the HDevelop operators in section 6.2.6.2 on page 102.

Variable Names

The export adds the prefix `ho_` to all local iconic and `hv_` to all local control variables, respectively, to avoid collisions with reserved words.

Exception Handling

In HDevelop, every exception normally causes the program to stop and report an error message in a dialog window. This might not be useful in C#. The standard way to handle this in C# is by using the `try/catch` mechanism. This allows you to access the reason for the exception and to continue accordingly. Thus, for HDevelop programs containing error handling (`(dev_)set_check("~give_error")`) the corresponding code is automatically included. Every operator call, for which it is assumed that the HALCON error mechanism is turned off, is enclosed in a `try` block followed by a `catch` block. The latter handles the exception and assigns the corresponding HALCON error number to the error variable activated by `dev_error_var` or to a local error variable, otherwise.

Please note that a call of `(dev_)set_check("~give_error")` has no influence on the operator call. The exception will *always* be raised. This is also true for messages like `H_MSG_FAIL`, which are not handled as exceptions in C++, for example.

Memory Management

The .NET Framework's runtime environment CLR (Common Language Runtime) has a mechanism called garbage collector, which is used by the CLR to remove no longer needed .NET objects from memory. As mentioned earlier, in the exported C# code every iconic object is represented by a .NET`HObject` object. From the garbage collector's point of view, a .NET`HObject` object is rather small. Thus, it might not be collected from memory although the underlying iconic object (e.g., an image) might in fact occupy a large portion of memory. In order to avoid memory leaks caused by this effect, in the exported code every iconic object is disposed explicitly before it is assigned a new value.

10.2.3 Code Generation for Visual Basic .NET (HALCON/.NET)

This section describes how to create a HALCON application in Visual Basic .NET, starting from a program developed in HDevelop. HALCON can be used together with Visual Basic .NET based on the .NET interface of HALCON. A detailed description of this interface can be found in the Programmer's Guide, part III on page 59.

10.2.3.1 Basic Steps

Export

The first step is to export the program using the menu `File` ▷ `Export...`. Here, select the language (`Visual Basic .NET - HALCON/.NET`) and save it to file. The result is a new file with the given name and the extension "`.vb`".

The Visual Basic .NET Template

If the file has been exported using the option `Use Export Template`, it is intended to be used together with the predefined Visual Basic .NET project that can be found in the directory

```
%HALCONEXAMPLES%\vb.net\HDevelopTemplate
```

This project contains a form with a display window (`HWindowControl`) and a button labeled `Run`. Add the file generated by HDevelop to the project in the Solution Explorer (`Add Existing Item`). Now the project is ready for execution: Run the project and then press the button `Run` on the form, which will call the exported code.

Additional information about using the template can be found in the Programmer's Guide, section 12.3.1 on page 84.

10.2.3.2 Program Structure

If the program has been exported using the option `Use Export Template`, the file created by HDevelop contains a subroutine with the corresponding name for every HDevelop procedure except the main procedure, which is contained in the subroutine `action()`. Otherwise, the file is exported as a standalone application. Iconic input and output parameters of a procedure are passed as `ByVal HObject` and `ByRef HObject`, respectively, while control input and output parameters are passed as `ByVal HTuple` and `ByRef HTuple`, respectively. The subroutine `RunHalcon()` contains a call to the subroutine `action()` and has a parameter `Window`, which is of type `HTuple`. This is the link to the window on the form to which all output operations are passed. In addition, another subroutine is created with the name `InitHalcon()`. This subroutine applies the same initializations that HDevelop performs.

Most of the variables (iconic as well as control) are declared locally inside the corresponding subroutines. Iconic variables belong to the class `HObject` and control variables belong to `HTuple`.

Depending on the program, additional subroutines and variables are declared.

Stop

The HDevelop operator `stop` is translated into a subroutine in Visual Basic .NET that creates a message box. This message box causes the program to halt until the button is pressed.

Exit

The HDevelop operator `exit` is translated into the Visual Basic .NET routine `End`. Because this routine has no parameter, the parameters of `exit` are suppressed.

Used Classes

There are only four classes/types that are used: `HTuple` for control parameters and `HObject` for iconic data. In addition, there is the class `HWindowControl`. It is used inside the project for the output window and a variable of class `HTuple` directs the output to this window. Finally, the class `HOperatorSet` is used as a container for all HALCON operators. There is no need for other classes as long as the program has the same functionality as in HDevelop. When editing a generated program you are free to use any of the classes of HALCON/.NET to extend the functionality.

10.2.3.3 Limitations and Troubleshooting

Besides the restrictions mentioned in this section and in section 10.2.5 on page 342, please also check the description of the HDevelop operators in section 6.2.6.2 on page 102.

Variable Names

In contrast to C, C++, or HDevelop, Visual Basic .NET has many reserved words. Thus, the export adds the prefix `ho_` to all iconic and `hv_` to all control variables, respectively, to avoid collisions with these reserved words. See also section 10.2.5.3 on page 343 about case sensitivity.

Exception Handling

In HDevelop, every exception normally causes the program to stop and report an error message in a dialog window. This might not be useful in Visual Basic .NET. The standard way to handle this in Visual Basic .NET is by using the `Try/Catch` mechanism. This allows you to access the reason for the exception and to continue accordingly. Thus, for HDevelop programs containing error handling (`(dev_)set_check("~give_error")`) the corresponding code is automatically included. Every operator call, for which it is assumed that the HALCON error mechanism is turned off, is enclosed in a `Try` block followed by a `Catch` block. The latter handles the exception and assigns the corresponding HALCON error number to the error variable activated by `dev_error_var` or to a local error variable, otherwise.

Please note that a call of `(dev_)set_check("~give_error")` has no influence on the operator call. The exception will *always* be raised. This is also true for messages like `H_MSG_FAIL`, which are not handled as exceptions in C++, for example.

Memory Management

The .NET Framework's runtime environment CLR (Common Language Runtime) has a mechanism called garbage collector, which is used by the CLR to remove no longer needed .NET objects from memory. As mentioned earlier, in the exported Visual Basic .NET code every iconic object is represented by a .NET `HObject` object. From the garbage collector's point of view, a .NET `HObject` object is rather small. Thus, it might not be collected from memory although the underlying iconic object (e.g., an image) might in fact occupy a large portion of memory. In order to avoid memory leaks caused by this effect, in the exported code every iconic object is deleted explicitly before it is assigned a new value.

Parallel Execution

If the program to be exported includes the parallel execution of procedure or operator calls using `par_start` (see section 8.10 on page 312), the exported code requires Visual Studio 2010 or higher.

10.2.4 Code Generation for C

This section describes how to create a HALCON application in C, starting from a program developed in HDevelop.

10.2.4.1 Basic Steps

Program Export

The first step is to export the program using the menu `File` ▷ `Export...`. Here, select the language (`C - HALCON/C`) and save it to file. A file will be created that contains the HDevelop program as C source code. For every HDevelop procedure except the main procedure, the exported file contains a C procedure with the corresponding name. Iconic input and output parameters of a procedure are declared as `Hobject` and `Hobject*`, respectively, while control input and output parameters are declared as `Htuple` and `Htuple*`, respectively. All procedures are declared at the beginning of the file. The program body of the HDevelop main procedure is contained in a procedure `action()` which is called in function `main()`. `action()` and `main()` can be excluded from compilation by inserting the instruction `#define NO_EXPORT_MAIN` at the appropriate position in the application. Using the instruction `#define NO_EXPORT_APP_MAIN` only the `main()` procedure is excluded from compilation. This can be useful if you want to integrate exported HDevelop code into your application through specific procedure interfaces. In that case, there is typically no need to export the main procedure, which was probably used only for testing the functionality implemented in the corresponding 'real' procedures.

Besides the program code, the file contains all necessary `#include` instructions. All local variables (iconic as well as control) are declared in the corresponding procedures. Iconic variables belong to the class `Hobject` and all other variables belong to `Htuple`.

Please note that in the current version the generated C code is not optimized for readability. It is output such that it always produces identical results as the HDevelop code.

Compiling and Linking in Windows Environments

The next step is to compile and link this new program. In the Windows environment, Visual C++ is used for the compiling and linking. Example projects can be found in the directory `%HALCONEXAMPLES%\c`.

If you want to use HALCON XL, you have to include the libraries `halconxl.lib/.dll` and `halconcxl.lib/.dll` instead of `halcon.lib/.dll` and `halconc.lib/.dll` in your project (see the Programmer's Guide, chapter 19 on page 119, for more details).

Compiling and Linking in Linux Environments

To compile and link the new program (called, e.g., `test.c`) under Linux, you can use the example `makefile`, which can be found in the directory `$HALCONEXAMPLES/c/makefiles`, by calling

```
make TEST_PROG=test
```

Alternatively, you can set the variable `TEST_PROG` in `makefile` to `test` and then just type `make`.

You can link the program to the HALCON XL libraries by calling

```
make TEST_PROG=test XL=1
```

or just type `make XL=1` if you set the variable `TEST_PROG` as described above.

For more details see the Programmer's Guide, chapter 19 on page 119.

Compiling and Linking in macOS Environments

In the macOS environment, Xcode is used for the compiling and linking. Example projects can be found in the directory `/Users/Shared/Library/Application Support/HALCON-21.05/examples/c`.

To compile and link the new program (called, e.g., `test.c`) under macOS from the command line, you can use the example `makefile`, which can be found in the directory `$HALCONEXAMPLES/c/makefiles`, by calling

```
make PROG=test
```

Alternatively, you can set the variable `PROG` in `makefile` to `test` and then just type `make`.

You can link the program to the HALCON XL libraries by calling

```
make PROG=test XL=1
```

or just type `make XL=1` if you set the variable `PROG` as described above.

For more details see the Programmer's Guide, chapter 19 on page 119.

10.2.5 General Aspects of Code Generation

In the following, general differences in the behavior of an HDevelop program and its exported versions are described.

10.2.5.1 Arbitrary Program Code

It is possible to embed arbitrary code into HDevelop programs. This code is ignored inside HDevelop. When you export the program to a programming language, the embedded code is passed through verbatim.

Program lines starting with `#` as the first character mark an arbitrary code line. The marker and the first space character following it are discarded when exporting the program. For example, the line

```
# Call MsgBox("Press button to continue",vbYes,"Program stop","",1000)
```

in HDevelop will result in

```
Call MsgBox("Press button to continue",vbYes,"Program stop","",1000)
```

in Visual Basic. The `#` may be followed by other special characters that further specify where the code block will be placed upon exporting. For example, the line

```
#^^ #define NO_EXPORT_APP_MAIN
```

puts the line

```
#define NO_EXPORT_APP_MAIN
```

at the very beginning of the exported program. Code lines in this format are collected from the `main` procedure first, followed by `#^^` lines in other procedures.

Prefix	Destination	export_def
#	The place of insertion	'in_place'
#^^	Beginning of the program	'at_file_begin'
#$$	End of the program	'at_file_end'
#^	Before the current procedure	'before_procedure'
#$	After the current procedure	'after_procedure'

Table 10.1: Embedding arbitrary code in HDevelop.

The recognized special markers are summarized in table 10.1.

If you are using the operator window to enter arbitrary code lines, you will have to select the special operator export_def. Its first parameter specifies the destination of the exported code line (see the last column of table 10.1 for reference). The second parameter is the code line itself. When you submit the operator to the program window, the operator call will be converted to the special prefix characters for better readability.

10.2.5.2 Assignment

In HDevelop each time a new value is assigned to a variable its old contents are removed automatically, independent of the type of the variable. In the exported code, this is also the case for iconic objects (HALCON/C++: Hobject, HALCON/.NET: HObject) and for the class HTuple (HALCON/C++, HALCON/.NET). Because C does not provide destructors, the generated C code calls the operators clear_obj and destroy_tuple to remove the content of iconic output parameters (Hobject) and control output parameters (Htuple) before each operator call. Memory issues regarding iconic objects in HALCON/.NET are described in section 10.2.3.3 (Visual Basic .NET) and section 10.2.2.3 (C#).

10.2.5.3 Variable Names

Variable names in HDevelop are case-sensitive, i.e., x and X are distinct variable names in HDevelop programs. If you export such a program to a case-insensitive target language (e.g., Visual Basic .NET), the development environment will complain about multiple declarations. Either plan ahead and avoid these variable names, or use the Find/Replace dialog to substitute conflicting variable names before exporting your program.

10.2.5.4 for Loops

Old HDevelop programs (prior to HALCON 11) have different semantics of for loops compared to other programming languages. If an old HDevelop program is opened, the for loops are labeled to be run in a compatibility mode to simulate the old behavior. To avoid confusions about the exported code, it is recommended to disable the compatibility mode by removing the special __use_internal_index__ tag. See the reference manual of the for operator for more information.

The following applies only to old HDevelop programs using the compatibility mode that are going to be exported. Because the problems are so rare and the generated code would become very difficult to understand otherwise, the code generation ignores the different semantics. These differences are:

1. In the programming languages, you can modify the loop variable (e.g., by setting it to the end value of the condition) to terminate the loop. This can't be done in HDevelop because here the current value is stored "inside" the for-operator and is automatically updated when it is executed again.
2. In the programming languages, you can modify the step range if you use a variable for the increment. This is also not possible with HDevelop because the increment is stored "inside" the for-operator when the loop is entered.
3. The last difference concerns the value of the loop variable after exiting the loop. In the programming languages, it has the value with which the condition becomes false for the first time. In HDevelop it contains the end value, which was calculated when the loop was entered.

Looking at the mentioned points, we recommend that you program according to the following rules:

1. Don't modify the loop variable or the step value inside the loop. If you need this behavior, use the `while`-loop.
2. Don't use the loop variable after the loop.

10.2.5.5 Protected Procedures

As described for the different programming languages, HDevelop procedures are exported automatically to procedures or subroutines of the selected programming language. This does not hold for the protected procedures described in section 5.6 on page 45. These procedures are protected by a password so that they cannot be viewed and modified by unauthorized users. Thus, as long as they are locked by the password, they can not be exported to any programming language.

10.2.5.6 System Parameters

You should know that HDevelop performs some changes of system parameters of HALCON by calling the operator `set_system` (see the reference manual). This might cause the exported program not to produce identical output. If such a problem arises, you may query the system parameters by means of `get_system` in HDevelop after or while running the original HDevelop version of the program. Depending to the problem, you can now modify relevant parameters by explicitly calling the operator `set_system` in the exported program.

10.2.5.7 Graphics Windows

HALCON provides a functionality that emulates the behavior of HDevelop graphics windows for HALCON windows. This HALCON window stack is accessible via class methods and functions in the HALCON interfaces, and code exported from HDevelop uses this functionality when opening, closing, setting, or accessing the active window. The HALCON window stack mechanism is threadsafe and can be shared among threads. Though, in a multithreaded application the user has to take care when switching the active window in different threads, as setting it in one thread will also set it for other threads.

For the .NET code export it is optional whether to export HDevelop programs as code using the HDevelop export example templates or as code using the previously described HALCON window stack functionality when doing graphics windows output. Additionally, in the latter case the exported code contains a main function and thus is usable as a standalone application. The HDevelop `Export` dialog allows you to select the corresponding option.

The graphics windows of HDevelop and the basic windows of the HALCON libraries

- HALCON/C++: class `HWindow`,
- HALCON/.NET: class `HWindowControl`, and
- HALCON/C: addressed via handles

have different functionality.

- **Multiple windows**
  If you use the operator `dev_open_window` to open multiple graphics windows in HDevelop, these calls will be converted into corresponding calls of `open_window` only if the export option `Use HALCON Windows` is selected.

  In the export of Visual Basic .NET, and C# programs using the option `Use Export Template`, all window operations are suppressed, because the exported code is intended to work together with the corresponding template. If you want to use more than one window in programs exported in this mode, you have to modify the code and project manually.

  Note that the export of programs containing multiple windows using the option `Use HALCON Windows` might be incorrect if the active graphics window was changed using the mouse during program execution. It is recommended to use the operator `dev_set_window` explicitly to achieve the same functionality.

- **Window size**
  In exported Visual Basic .NET, and C# programs, the size of the window on the form is predefined (512×512); thus, it will normally not fit your image size. Therefore, you must adapt the size interactively or by using the properties of the window.

- **Displaying results**
  Normally, the result of every operator is displayed in the graphics window of HDevelop. This is not the case when using an exported program. It behaves like the HDevelop program running with the option: "update window = off". We recommend inserting the operator `dev_display` in the HDevelop program at each point where you want to display data. This will not change the behavior of the HDevelop program but result in the appropriate call in the exported code.

  When generating code using the option `Use HALCON Windows`, close the default graphics window (using `dev_close_window`) and open a new one (using `dev_open_window`) *before* the first call of `dev_display` in order to assure a correct export.

- **Displaying images**
  In HDevelop, images are automatically scaled to fit the current window size. *This is not the case in exported programs*. For example, if you load and display two images of different size, the second one will appear clipped if it is larger than the first image or filled up with black areas if it is smaller. For a correct display, you must use the operator `dev_set_part` *before* displaying an image with `dev_display` as follows:

  ```
  dev_set_part (0, 0, ImageHeight-1, ImageWidth-1)
  dev_display (Image)
  ```

  In this example, `Image` is the image variable, `ImageHeight` and `ImageWidth` denote its size. You can query the size of an image with the operator `get_image_size`. Please consult the HALCON Reference Manuals for more details.

  Note that the operator `dev_set_part` (and its HALCON library equivalent `set_part`) is more commonly used for displaying (and thereby zooming) *parts* of images. By calling it with the full size of an image as shown above, you assure that the image exactly fits the window.

- **Changing display parameters**
  If you change the way how results are displayed (color, line width, etc.) in HDevelop interactively via the menu `Visualization`, these changes will not be incorporated in the exported program. We recommend inserting the corresponding `Develop` operators (e.g., `dev_set_color` or `dev_set_line_width`) in the HDevelop program explicitly. This will result in the appropriate call (`set_color`, `set_line_width`, etc.) in the exported code.

10.2.5.8 Unset Output Parameters in Procedures

HDevelop and exported code can behave differently regarding the values of output parameters of procedures in case an output parameter is not assigned a value inside the procedure. This can happen if the procedure returns with an exception before setting an output parameter or if the parameter was not set on purpose, in which case the value of the output parameter is not defined and thus HDevelop and the different code exports may behave differently. The main reason why the code exports are not adapted to behave like HDevelop is that that would imply a deterioration of runtime efficiency in the exported code.

10.2.5.9 Accessing Uninitialized Tuple or Vector Elements

When accessing uninitialized elements of a tuple or vector with HDevelop, an exception is raised. Due to performance reasons, there is no corresponding check in the HALCON language interfaces. Instead of raising an exception, an element of the default type is returned.

```
a := [0, 1]
b := a[2]
```

For example, this code snippet raises an exception in HDevelop, but might return an arbitrary value for *b* in one of the language interfaces. Thus, this coding style must be avoided.

Appendix A

Glossary

Boolean is the type name for the truth values `true` and `false` as well as for the related boolean expressions.

Body A body is part of a conditional instruction (`if`) or a loop (`while` or `for`) and consists of a sequence of operator calls. If you consider the `for`-loop, for instance, all operator calls, that are located between `for` and `endfor` form the body.

Control data Control data can be either numbers (↑`integer` and ↑`real`), character strings (↑`string`) and truth values (`boolean`). This data can be used as atomic values (i.e., single values) or as ↑tuples (i.e., arrays of values).

Empty region An empty ↑region contains no points at all, i.e., its area is zero.

Graphics window A graphics window is used in ↑HDevelop for displaying, e.g., ↑images, ↑regions, and ↑XLD.

HDevelop is an interactive program for the creation of HALCON applications.

Iconic data are image data, i.e., image arrays and data, which are described by coordinates and are derived from image arrays, e.g., ↑regions, ↑images and ↑XLD.

Image An image consists of one or more (multi-channel image) image arrays and a ↑region as the definition domain. All image arrays have the same dimension, but they can be of different pixel types. The size of the ↑region is smaller or equal than the size of the image arrays. The ↑region determines all image points that should be processed.

Iconic object Generic implementation of ↑iconic data in HALCON.

integer is the type name for integer numbers.

Operator data base The operator data base contains information about the HALCON operators. It is loaded at runtime from the binary files in `%HALCONROOT%\help`.

Program window In HDevelop the program window contains the program. It is used to edit (copy, delete, and paste lines) and to run or debug the program.

Operator window In the operator window of HDevelop the parameters of the selected operators can be entered or modified.

Real is the type name for floating point numbers. They are implemented using the C-type `double` (8 bytes).

Region A region is a set of image points without gray values. A region can be imagined as a binary image (mask). Regions are implemented using runlength encoding. The region size is not limited to the image size (see also `set_system('clip_region','true'/'false')` in the HALCON reference manual).

String is the type name for character strings. A string starts and ends with a single quote; in between any character can be used except single quote. The empty string consists of two consecutive single quotes.

Tuple A tuple is an ordered multivalue set. In case of ↑control data a tuple can consist of a large number of items with different data types. The term tuple is also used in conjunction with ↑iconic objects, if it is to be emphasized that several ↑iconic objects will be used.

Type ↑iconic variables can be assigned with data items of type ↑image, ↑region, and ↑XLD. The types of ↑control data items can be one of ↑`integer`, ↑`real`, ↑`boolean`, or ↑`string`.

Variable window In HDevelop the variable window manages the ↑control and ↑iconic data.

XLD is the short term for e*X*tended *L*ine *D*escription. It is used as a superclass for contours, polygons, and lines.

Appendix B

Color names

The following color names are predefined in HALCON.

Color Name	Color	Hex Triplet
white		#FFFFFF
red		#FF0000
green		#00FF00
blue		#0000FF
dim gray		#696969
gray		#BEBEBE
light gray		#D3D3D3
cyan		#00FFFF
magenta		#FF00FF
yellow		#FFFF00
medium slate blue		#7B68EE
coral		#FF7F50
slate blue		#6A5ACD
spring green		#00FF7F
orange red		#FF4500
dark olive green		#556B2F
pink		#FFC0CB
cadet blue		#5F9EA0
goldenrod		#DAA520
orange		#FFA500
gold		#FFD700
forest green		#228B22
cornflower blue		#6495ED
navy		#000080
turquoise		#40E0D0
dark slate blue		#483D8B
light blue		#ADD8E6

Color Name	Color	Hex Triplet
indian red		#CD5C5C
violet red		#D02090
light steel blue		#B0C4DE
medium blue		#0000CD
khaki		#F0E68C
violet		#EE82EE
firebrick		#B22222
midnight blue		#191970
sea green		#2E8B57
dark turquoise		#00CED1
orchid		#DA70D6
sienna		#A0522D
medium orchid		#BA55D3
medium forest green		#6B8E23
medium turquoise		#48D1CC
medium violet red		#C71585
salmon		#FA8072
blue violet		#8A2BE2
tan		#D2B48C
pale green		#98FB98
sky blue		#87CEEB
medium goldenrod		#EAEAAD
plum		#DDA0DD
thistle		#D8BFD8
dark orchid		#9932CC
maroon		#B03060
dark green		#006400
steel blue		#4682B4
medium spring green		#00FA9A
medium sea green		#3CB371
yellow green		#9ACD32
medium aquamarine		#66CDAA
lime green		#32CD32
aquamarine		#7FFFD4
wheat		#F5DEB3
green yellow		#ADFF2F

Appendix C

Command Line Usage

C.1 HDevelop

HDevelop understands many command line switches when started from a console or terminal window. Run the following command to get a list of all supported command line switches:

```
hdevelop -h
```

In general, HDevelop is started using

```
hdevelop [OPTIONS] [FILE]
```

FILE can be an HDevelop program, an external procedure, a library file, or an image file. The default action is to load the given file for editing. If an image file is given, it is preselected in the dialog `Read Image...` (page 57) (same behavior as dragging an image file onto a running instance of HDevelop).

This appendix presents some useful command line examples.

Load the HDevelop program *example.hdev* and run it immediately:

```
hdevelop -run example.hdev
```

Run the HDevelop program *example.hdev* and skip `stop` calls:

```
hdevelop -override_stop 0 -run example.hdev
```

Use the procedures and libraries from *C:/projects* in the next session:

```
hdevelop -external_proc_path:C:/projects"
```

The given path name(s) are put first in the list of directories that are searched for external procedures and libraries. Multiple directories are separated by the system-dependent separator (";" on Windows and ":" on Linux/macOS). This is a temporary setting for the current session only. See also page 43.

Open the protected program secret.hdev for editing (password: crypt):

```
hdevelop -unlock:crypt secret.hdev
```

Convert the old procedure example.dvp to the default procedure format:

```
hdevelop -convert example.dvp example.hdvp
```

Convert the procedure example.hdvp for use in older versions of HDevelop:

```
hdevelop -convert example.hdvp example.dvp
```

Convert the protected procedure secret.hdvp (password: crypt) to an unprotected procedure:

```
hdevelop -unprotect:crypt -convert secret.hdvp nosecret.hdvp
```

Protect the procedure example.hdvp with the password crypt:

```
hdevelop -protect:crypt -convert example.hdvp example.hdvp
```

Export the program example.hdev and all used procedures to C++:

```
hdevelop -convert example.hdev example.cpp
```

HDevelop accepts the following command line switches:

```
hdevelop [options]
 HDevelop options:
  <program>.{hdev,dev}
                    load the program for editing, converting, or running
  {<procedure_file>,<procedure_library_file>}
                    load the procedure or procedure library for editing
  -{protect,unprotect,unlock}:<password>
                    modify the protection state of an HDevelop program or
                    procedure
  -load_recent_prog load the recent program irrespective of the appropriate
                    option that was selected in the preferences dialog
  -load_no_prog     do not load the recent program irrespective of the
                    appropriate option that was selected in the preferences
                    dialog
  -run              start execution of the passed program
  <image_file>      load an image file with read_image
  -help             show this help info in a message box
  --help            show this help information on the console
  -version          show version information in a message box
  --version         show version information on the console
  -external_proc_path:<external procedure path(s)>
                    an external procedure path may point either to a directory
                    or a procedure library file
                    multiple external procedure paths are separated by semicolons
                    on windows systems, or by colons on all other systems
  -convert <source> <destination> [<options>]
                    convert an HDevelop program or procedure into a
                    file of the specified type
                    (not possible combinations:
                    convert dev, hdev, hdpl into dvp, hdvp
                    convert dev, hdev, dvp, hdvp into hdpl)
      source:       <src_file>.{hdev,dev}
                    <src_file>.{hdvp,dvp}
                    <src_file>.{hdpl}
      destination:  dest_file.<type>
                    <type>      write to <src_file>.<type>
                    - <type>    write to stdout
       type:        hdev,dev    HDevelop program
                    hdvp,dvp    HDevelop procedure
                    hdpl        HDevelop procedure library
                    c,cpp       C, C++
                    cs,vb       C#, VisualBasic.NET
```

```
                  txt         text
      options:    [-external_procs_only_interfaces]
                              export only the interface(s) of the procedures
                              (the declarations) without the bodies
                  [-no_export_of_referred_procs]
                              export only the passed program or procedure but
                              do not include any referred external procedures
                  [-no_msg_box]
                              suppress error messages
                  [-reset_free_text]
                              reset free text formating
                  [-delete_local_procs]
                  [-delete_unused_local_procs]
                  [-no_use_hdevelop_template]
                  [-encoding native]
                              exported string constants are encoded in native
                              local-8-bit encoding instead of UTF-8
                  [-encoding UTF-8]
                              exported string constants are encoded in UTF-8 (default)
                              Please note that non-ASCII characters in string constants
                              are exported as octal codes in order to guarantee that
                              the strings are correctly created on all systems,
                              independent on any compiler settings
 -reset_preferences
                  reset all persistent settings from previous sessions
 -add_preferences <file>
                  start HDevelop with additional preferences
                  from a file
                  (replaces -preferences <file> that became deprecated)
 -load_preferences <file>
                  reset all persistent settings and
                  start HDevelop with the preferences from a file
 -use_preferences <file>
                  start HDevelop with the preferences from file
                  and store all modified preferences to file
 -override_stop <time>
                  override stop() operator with wait_seconds(time)
 -override_wait <time>
                  replace time of wait_seconds() operator with <time>

Export library project options:
 -export_project
                  Export a library project
 -export_type <type>
                  Type of project. Currently supported:
                  cmake
 -input_file <file>
                  *.hdev or *.hdpl file used as project source
 -project_name <name>
                  Name of exported project
                  (must be a valid identifier in the target language)
 -output_folder <path>
                  Target path for exported files
 -encoding <encoding>
                  Set output encoding. Currently supported:
                  native
                  UTF-8
 -namespace <name>
                  Namespace of project
```

```
                    (must be a valid identifier in the target language)
 -language <language>
                    Set source code language. Currently supported:
                    cpp
                    cs

GUI options:
 -style[=] <style>
                    sets the application GUI style. Possible values are:
                    Windows Motif CDE Plastique Cleanlooks

X11 options:
 -display <display>
                    sets the X display (default is $DISPLAY).
 -geometry <geometry>
                    sets the client geometry of the first window that is shown.
 -graphicssystem {native|raster|opengl}
                    sets the graphics backend used (default: native).
 -{fn|font} <font>
                    defines the application font. The font should be specified
                    using an X logical font description.
 -{bg|background} <color>
                    sets the default background color and an application palette
                    (light and dark shades are calculated).
 -{fg|foreground} <color>
                    sets the default foreground color.
 -{btn|button} <color>
                    sets the default button color.
 -name <name>       sets the application name.
 -title <app_title> sets the application title.
 -visual TrueColor forces the application to use a  TrueColor visual on an
                    8-bit display.
 -ncols <count>     limits the number of colors allocated in the color cube on
                    an 8-bit display if the application is using the
                    QApplication::ManyColor color specification. If count is 216
                    then a 6x6x6 color cube is used (i.e., 6 levels of red,
                    6 of green, and 6 of blue); for other values, a cube
                    approximately proportional to a 2x3x1cube is used.
 -cmap              causes the application to install a private color map on an
                    8-bit display.
```

C.2 Hrun

Hrun is a command line utility to run HDevelop scripts.

This is especially useful if there is no HDevelop installed on your system, e.g., on embedded systems.

Hrun understands many command line switches when started from a console or terminal window. Run the following command to get a list of all supported command line switches:

```
hrun --help
```

In general, Hrun is started using

```
hrun [OPTIONS] [FILE]
```

Appendix D

Keyboard Shortcuts

The following sections list the keyboard shortcuts supported in HDevelop. On macOS, Cmd is used instead of Ctrl.

D.1 HDevelop

Function	Shortcut	Alternative
New Program	Ctrl+N	
Open Program...	Ctrl+O	
Save	Ctrl+S	
Save Program As...	Ctrl+Shift+S	
Save All	Ctrl+Alt+S	
Print Page	Ctrl+P	
Quit	Ctrl+Q	
Undo	Ctrl+Z	
Redo	Ctrl+Y	
Cut	Ctrl+X	Shift+Del
Copy	Ctrl+C	Ctrl+Ins
Paste	Ctrl+V	Shift+Ins
Delete	Del	
Activate	F3	
Deactivate	F4	
Find:	Ctrl+F	
Find Again	Ctrl+G	
Reset Program Execution	F2	
Reset Procedure Execution	Shift+F2	
Run	F5	
Run To Insert Cursor	Shift+F5	
Step Over	F6	
Step Forward	Shift+F6	

Function	Shortcut	Alternative
Step Into	F7	
Step Out	F8	
Abort Procedure Execution	Shift+F8	
Stop	F9	
Stop After Procedure	Shift+F9	
Set Breakpoint	F10	
Activate/Deactivate Breakpoint	Shift+F10	
Next Bookmark	F11	
Previous Bookmark	Shift+F11	
Set Bookmark	Ctrl+F11	
Activate Profiling	Ctrl+Shift+F,F	Ctrl+Shift+F,Ctrl+Shift+F
Number of Calls	Ctrl+Shift+F,N	Ctrl+Shift+F,Ctrl+Shift+N
Total Execution Time	Ctrl+Shift+F,L	Ctrl+Shift+F,Ctrl+Shift+L
Average Execution Time	Ctrl+Shift+F,A	Ctrl+Shift+F,Ctrl+Shift+A
Minimum Execution Time	Ctrl+Shift+F,I	Ctrl+Shift+F,Ctrl+Shift+I
Maximum Execution Time	Ctrl+Shift+F,X	Ctrl+Shift+F,Ctrl+Shift+X
Last Execution Time	Ctrl+Shift+F,M	Ctrl+Shift+F,Ctrl+Shift+M
Execution Time	Ctrl+Shift+F,E	Ctrl+Shift+F,Ctrl+Shift+E
Operator Time	Ctrl+Shift+F,O	Ctrl+Shift+F,Ctrl+Shift+O
Time	Ctrl+Shift+F,T	Ctrl+Shift+F,Ctrl+Shift+T
Percentage	Ctrl+Shift+F,P	Ctrl+Shift+F,Ctrl+Shift+P
Reset Profiler	Ctrl+Shift+F,R	Ctrl+Shift+F,Ctrl+Shift+R
Show Runtime Statistics	Ctrl+Shift+F,S	Ctrl+Shift+F,Ctrl+Shift+S
Activate/Deactivate display	Ctrl+Shift+F,V	Ctrl+Shift+F,Ctrl+Shift+V
Activate only selected	Ctrl+Shift+F,Y	Ctrl+Shift+F,Ctrl+Shift+Y
Set Parameters	Ctrl+Shift+G,P	Ctrl+Shift+G,Ctrl+Shift+P
Open Graphics Window...	Ctrl+Shift+G,O	Ctrl+Shift+G,Ctrl+Shift+O
Clear Graphics Window	Ctrl+Shift+G,Del	Ctrl+Shift+G,Ctrl+Shift+Del
Close Graphics Window	Ctrl+Shift+G,Q	Ctrl+Shift+G,Ctrl+Shift+Q
Save Window...	Ctrl+Shift+G,S	Ctrl+Shift+G,Ctrl+Shift+S
Insert Code...	Ctrl+Shift+G,I	Ctrl+Shift+G,Ctrl+Shift+I
Record Interactions	Ctrl+I	
400 %	Ctrl+Shift+G,4	Ctrl+Shift+G,Ctrl+Shift+4
200 %	Ctrl+Shift+G,2	Ctrl+Shift+G,Ctrl+Shift+2
100 %	Ctrl+Shift+G,1	Ctrl+Shift+G,Ctrl+Shift+1
50 %	Ctrl+Shift+G,5	Ctrl+Shift+G,Ctrl+Shift+5
Double	Ctrl+Shift+G,+	Ctrl+Shift+G,Ctrl+Shift++
Half	Ctrl+Shift+G,-	Ctrl+Shift+G,Ctrl+Shift+-
Aspect	Ctrl+Shift+G,=	Ctrl+Shift+G,Ctrl+Shift+=

Function	Shortcut	Alternative
Fit	Ctrl+Shift+G,Home	Ctrl+Shift+G,Ctrl+Shift+Home
Double	Ctrl+Shift+G,Right	Ctrl+Shift+G,Ctrl+Shift+Right
Half	Ctrl+Shift+G,Left	Ctrl+Shift+G,Ctrl+Shift+Left
Aspect	Ctrl+Shift+G,.	Ctrl+Shift+G,Ctrl+Shift+.
Save Procedure As...	Ctrl+Shift+P,S	Ctrl+Shift+P,Ctrl+Shift+S
Create New Procedure	Ctrl+Shift+P,C	Ctrl+Shift+P,Ctrl+Shift+C
Edit Procedure Interface	Ctrl+Shift+P,I	Ctrl+Shift+P,Ctrl+Shift+I
Delete Current	Ctrl+Shift+P,Del	Ctrl+Shift+P,Ctrl+Shift+Del
Browse HDevelop Example Programs...	Ctrl+E	
Read Image...	Ctrl+R	
Preferences	Ctrl+Shift+O,S	Ctrl+Shift+O,Ctrl+Shift+S
Open Graphics Window	Ctrl+Shift+O,G	Ctrl+Shift+O,Ctrl+Shift+G
Open Program Window	Ctrl+Shift+O,P	Ctrl+Shift+O,Ctrl+Shift+P
Open Variable Window	Ctrl+Shift+O,V	Ctrl+Shift+O,Ctrl+Shift+V
Open Operator Window	Ctrl+Shift+O,O	Ctrl+Shift+O,Ctrl+Shift+O
Gray Histogram	Ctrl+Shift+O,H	Ctrl+Shift+O,Ctrl+Shift+H
Feature Histogram	Ctrl+Shift+O,F	Ctrl+Shift+O,Ctrl+Shift+F
Feature Inspection	Ctrl+Shift+O,I	Ctrl+Shift+O,Ctrl+Shift+I
Line Profile	Ctrl+Shift+O,L	Ctrl+Shift+O,Ctrl+Shift+L
Thread View / Call Stack	Ctrl+Shift+O,C	Ctrl+Shift+O,Ctrl+Shift+C
Zoom Window	Ctrl+Shift+O,Z	Ctrl+Shift+O,Ctrl+Shift+Z
OCR Training File Browser	Ctrl+Shift+O,T	Ctrl+Shift+O,Ctrl+Shift+T
Export	Ctrl+Shift+O,X	Ctrl+Shift+O,Ctrl+Shift+X
Open Output Console	Ctrl+Shift+O,E	Ctrl+Shift+O,Ctrl+Shift+E
Open Quick Navigation	Ctrl+Shift+O,R	Ctrl+Shift+O,Ctrl+Shift+R
Manage Breakpoints	Ctrl+Shift+O,F10	Ctrl+Shift+O,Ctrl+Shift+F10
Manage Bookmarks	Ctrl+Shift+O,F11	Ctrl+Shift+O,Ctrl+Shift+F11
Invalid Lines	Ctrl+Shift+O,F12	Ctrl+Shift+O,Ctrl+Shift+F12
Full Screen	Ctrl+Shift+W,F	Ctrl+Shift+W,Ctrl+Shift+F
Next Window	Ctrl+Tab	Ctrl+>
Previous Window	Ctrl+Shift+Tab	Ctrl+<
Help	F1	
Context Help	Shift+F1	
Start Dialog	Ctrl+Shift+H,S	Ctrl+Shift+H,Ctrl+Shift+S
About	Ctrl+Shift+H,A	Ctrl+Shift+H,Ctrl+Shift+A
Keywords	Ctrl+Shift+H,K	Ctrl+Shift+H,Ctrl+Shift+K
HALCON News (WWW)	Ctrl+Shift+H,W	Ctrl+Shift+H,Ctrl+Shift+W
Search Documentation	Ctrl+Shift+H,F	Ctrl+Shift+H,Ctrl+Shift+F

Function	Shortcut	Alternative
HDevelop Language	Ctrl+Shift+H,L	Ctrl+Shift+H,Ctrl+Shift+L
HALCON Reference	Ctrl+Shift+H,R	Ctrl+Shift+H,Ctrl+Shift+R
HDevelop User's Guide	Ctrl+Shift+H,U	Ctrl+Shift+H,Ctrl+Shift+U

D.2 Program Window

Function	Shortcut	Alternative
Run Until Here	Shift+F5	
Set Insert Cursor	Ctrl+.	
Set Program Counter	Ctrl+,	
Show Procedure	Alt+Return	Alt+Enter
Show Procedure in New Tab	Alt+Ctrl+Return	Alt+Ctrl+Enter
Show Procedure in New Window	Alt+Shift+Return	Alt+Shift+Enter
Auto Indent	Ctrl+Shift+I	
PC Up	Ctrl+Up	
PC Down	Ctrl+Down	
Open Operator Window	Ctrl+Shift+Space	
Execute Current Line	Ctrl+Return	Ctrl+Enter
Go to Line	Alt+G	
Go to Program Counter	Alt+,	
Show main	Alt+Home	
Show Next Tab Card	Alt+Right	
Show Previous Tab Card	Alt+Left	
Select a Procedure	Alt+Up	
Move tab to the left	Ctrl+Alt+Shift+Left	
Move tab to the right	Ctrl+Alt+Shift+Right	
Incremment text size	Ctrl++	
Decremment text size	Ctrl+-	
Set Breakpoint	F10	
Activate/Deactivate Breakpoint	Shift+F10	
Set Bookmark	Ctrl+F11	
Back in History	Alt+Shift+Left	
Forward in History	Alt+Shift+Right	
New Tab	Alt+Ins	
List Open Tabs	Alt+Down	

D.3 HDevelop Help Window

Function	Shortcut	Alternative
Locate page	Ctrl+L	
Back	Alt+Left	
Forward	Alt+Right	
Home	Alt+Home	
Increase Zoom Factor	Ctrl++	
Decrease Zoom Factor	Ctrl+-	
Reset to normal size	Ctrl+0	
Bookmark	Ctrl+D	
Print Page	Ctrl+P	
Insert	Alt+Return	Alt+Enter
Find:	Ctrl+F	
Next	Down	
Previous	Up	

D.4 Graphics Window

Function	Shortcut	Alternative
Delete	Del	
Increase Zoom Factor	Ctrl++	
Decrease Zoom Factor	Ctrl+-	
Clear	c	Shift+c
Select Mode	s	Shift+s
Move Mode	m	Shift+m
Magnify Mode	z	Shift+z
Zoom In Mode	+	
Zoom Out Mode	-	
Draw ROI	r	Shift+r
3D Mode	3	
Show Axes	x	Shift+x
Show Grid	g	Shift+g
Activate	a	Shift+a
Close Graphics Window	Ctrl+Shift+G,Q	Ctrl+Shift+G,Ctrl+Shift+Q
400 %	Ctrl+Shift+G,4	Ctrl+Shift+G,Ctrl+Shift+4
200 %	Ctrl+Shift+G,2	Ctrl+Shift+G,Ctrl+Shift+2
100 %	Ctrl+Shift+G,1	Ctrl+Shift+G,Ctrl+Shift+1
50 %	Ctrl+Shift+G,5	Ctrl+Shift+G,Ctrl+Shift+5
Double	Ctrl+Shift+G,+	Ctrl+Shift+G,Ctrl+Shift++
Half	Ctrl+Shift+G,-	Ctrl+Shift+G,Ctrl+Shift+-
Aspect	Ctrl+Shift+G,=	Ctrl+Shift+G,Ctrl+Shift+=

Function	Shortcut	Alternative
Fit	Ctrl+Shift+G,Home	Ctrl+Shift+G,Ctrl+Shift+Home
Double	Ctrl+Shift+G,Right	Ctrl+Shift+G,Ctrl+Shift+Right
Half	Ctrl+Shift+G,Left	Ctrl+Shift+G,Ctrl+Shift+Left
Aspect	Ctrl+Shift+G,.	Ctrl+Shift+G,Ctrl+Shift+.

D.5 Plot Area

Function	Shortcut	Alternative
Zoom In	+	Ctrl+Shift++
Zoom Out	-	Ctrl+Shift+-
Fit Data Range	d	Ctrl+Shift+D
Zoom To Selection	s	Ctrl+Shift+S
Reset Bounds	r	Ctrl+Shift+R
Enter Bounds...	b	Ctrl+Shift+B
Show Mouse Position	p	Ctrl+Shift+P
Show Function Value At X	x	Ctrl+Shift+X
Show Function Value At Y	y	Ctrl+Shift+Y
Show Background Grid	g	Ctrl+Shift+G
Reset Bounds	w	Ctrl+Shift+W
Reset Bounds	h	Ctrl+Shift+H
Left	Shift+Left	Shift+4
Right	Shift+Right	Shift+6
Up	Shift+Up	Shift+8
Down	Shift+Down	Shift+2

D.6 Variable Inspect

Function	Shortcut	Alternative
Cut	Ctrl+X	Shift+Del
Copy	Ctrl+C	Ctrl+Ins
Paste	Ctrl+V	Shift+Ins
Delete	Del	
Insert	Ctrl+Shift+V	

D.7 Handle Inspect

Function	Shortcut	Alternative
Copy	Ctrl+C	Ctrl+Ins

D.8 OCR Training File Browser

Function	Shortcut	Alternative
New Training File	Ctrl+N	
Load Training File...	Ctrl+O,T	Ctrl+O,Ctrl+T
Save Training File	Ctrl+S	
Close Training File	Ctrl+L	
Save All Training Files	Ctrl+Alt+S	
Close All Training Files	Ctrl+Alt+L	
Save Training File As...	Ctrl+Shift+S	
Load OCR Classifier...	Ctrl+O,C	Ctrl+O,Ctrl+C
Undo	Ctrl+Z	
Redo	Ctrl+Y	
Cut	Ctrl+X	Shift+Del
Copy	Ctrl+C	Ctrl+Ins
Paste	Ctrl+V	Shift+Ins
Delete	Del	
Add Symbol Name	Ctrl+E,N	Ctrl+E,Ctrl+N
Generate Variations...	Ctrl+E,V	Ctrl+E,Ctrl+V
Add Samples From A System Font...	Ctrl+E,F	Ctrl+E,Ctrl+F
Detailed View	Ctrl+Shift+V,D	Ctrl+Shift+V,Ctrl+Shift+D
Thumbnails	Ctrl+Shift+V,T	Ctrl+Shift+V,Ctrl+Shift+T

Index

* (asterisk)
    external procedure, 53
    in window title, 33, 49
.NET, 338, 339
.avi, 26
.dev, 42
.dvp, 42
.hdev, 42
.hdpl, 42
.hdvp, 42
.seq, 26
#, 343
#$, 343
#$$, 343
#^, 343
#^^, 343
$, 293

Abort Procedure Execution, 88
About, 111
Acquisition
    menu (Image Acquisition Assistant), 198
    Snap, 195
Acquisition menu
    Live, 195
Acquisition Mode, 198
actionUpdateWin, 96
Activate, 61, 113, 120
Activate Profiler, 88, 113
Activate/Deactivate Breakpoint, 86, 120
Activate/Deactivate Breakpoint on Variable, 120, 146, 148
Adapt program, 125
Adaptive Range, 154, 172
Add to User Tab, 146, 148
Add Variable, 146, 148
Add Watch, 121
Advanced, 129
advanced autocompletion, 116
advanced fuzzy features, 258
advanced measuring tasks, 254, 265
advanced model parameters, 235
advanced parameters Detector Type, 238
advanced search parameters, 243
advanced training parameters, 271
alignment, 258
All, 144
Alternatives, 105
Always Find, 241
Angle Extent, 233
Angle Step, 236
Apply Changes Immediately, 92
assign_at, 337
assistant
    calibration, 193
    Close Dialog, 194
    Delete Generated Code Lines, 194
    Exit Assistant, 194
    image acquisition, 24, 193
    Insert Code for Selection, 194
    Load Assistant Settings..., 194
    matching, 193
    measure, 193
    OCR, 193
    Release Generated Code Lines, 194
    Save Current Assistant Settings..., 194
    Show Code Preview, 194
assistant settings, load, 214, 226
assistant settings, save, 214, 226
Assistants
    menu, 106
Attach To Process..., 85
Attention, 129
Auto, 144
Auto Disconnect, 198
Auto Indent, 120
Auto-detect Interfaces, 195
Autocompletion, 69
autocompletion, 116
AVI, 26

Back, 167
Back in History, 114
basic features, 269
Basics, 128
Bit Depth, 195
Bookmark, 167
boolean, 282, 347
    operations, 298
breakpoint, 17, 83
    activate/deactivate, 86
    clear all, 86
    manage, 86
    on variable, 143
    set, 86
Bring Main Window To Front, 110
Bring Others To Front, 110
Browse HDevelop Example Programs..., 19, 52, 113

C, 11, 341

compile and link (macOS), 342
compile and link (Linux), 341
compile and link (Windows), 341
C export, 341
C++, 11
compile and link (macOS), 336
compile and link (Linux), 336
compile and link (Windows), 336
C++ export, 335
C#, 11
C# export, 338
Calibrate, 215
calibration, 204
Calibration Assistant, 193
calibration plate, 201
calibration plate extraction parameters, 209
calibration source, 229, 253
calibration task, 201
camera parameters, 201
camera parameters, load, 225
camera pose, load, 226
Camera Type, 195
Canvas Options, 109
canvas window, 17, 161
Cascade Floating Windows, 109
catch, 311
categories
example programs, 52
Category, 196
channel
gray value, 97, 168
channel number, 169
channel selection
gray histogram, 176
Chapters, 128
check box Always Find, 241
check box Pregenerate Shape Model, 237
check box Shape models may cross the image border, 245
Check For Updates, 111
Clear / Display, 145
Clear All Breakpoints, 86
Clear Graphics Window, 89
Clear Variable, 146, 148
clipboard, 60
Close All Procedures, 53
Close Assistant, 215, 226
Close Dialog, 194, 260, 277
Close Graphics Window, 89
Close Procedure, 53
Code Generation, 215, 247, 253, 261, 274, 278, 331, 335
File, 25
Image Acquisition Assistant, 197
image acquisition interface, 28
Code Generation, preview, 215, 249, 261, 278
code lines, delete, 215, 248, 261, 278
code lines, insert, 215, 248, 261, 278
code lines, release, 215, 248, 261, 278
code options, 248
Code Preview, 197, 254, 275
code variables, 248
Color, 91, 155, 177
Color Space, 195
Colored, 90
Colors, 71
column, 50
Column Scale Step, 236
comment, 61, 102
#, 343
#$, 343
#$$, 343
#^, 343
#^^, 343
comparison
operations, 297
compilation of procedures, 45
Complexity, 129
Connect, 196
Connection, 177
Image Acquisition Assistant, 195
Connection Handle, 198
Context Help, 110
continuation
line, 116
Contrast, 232
Control, 101
control data, 347
Control Flow, 198
control flow
break, 310
continue, 309
elseif, 308
exit, 311
for ... endfor, 308
if ... else ... endif, 306
if ... endif, 306
operators, 305
repeat ... until, 308
return, 311
stop, 311
throw, 311
try ... catch ... endtry, 311
while ... endwhile, 308
coordinates
status bar, 50
Copy, 60, 120, 148
Copy History to Clipboard, 50
Create Model, 227
create model, 219
Create New Procedure, 99, 120
Create ROI, 250, 265
Cut, 60, 120

Deactivate, 61, 113, 120
debugging
remote, 327
Declared in, 146, 148
Decrease Zoom Factor, 168

Delete, 61, 120
Delete All ROIs, 260, 261, 277
Delete All Unused Local, 100
Delete Current, 100
Delete Generated Code Lines, 194, 215, 248, 261, 278
Delete Selected ROI, 277
Descriptor Min. Score, 245
Detailed Description, 129
Detect, 27, 196
Detect All, 240
Detector Type, 234
Detector Type, advanced, 238
determine pose bounds, 247
Determine Recognition Rate, 246
dev_ operators, 102
dev_display, 345
dev_open_window, 345
dev_set_check, 311
dev_set_part, 345
Develop, 102
Device, 195
Dim +, 125
Dim -, 125
Disabled, 197
Disconnect, 196
Display, 89, 145
Display Axes, 160
Display Channel, 145
Display Content:, 145
Display Grid, 160
Display Image, 197
Display Image Pyramid, 228
display parameters, 209, 252, 273
Display Selected Test Image, 241
Draw, 91, 177
Draw Axis-aligned Rectangle, 277
Draw Circular Arc, 260
Draw Line, 260
Draw Rotated Rectangle, 277
Draw1, 164
Duplicate..., 99

edge data, 253
edge selection, 252
Edit menu, 59
    Activate, 61
    Copy, 60
    Cut, 60
    Deactivate, 61
    Delete, 61
    Find Again, 64
    Find/Replace..., 62
    Invalid Lines, 65
    Manage Bookmarks, 65
    Next Bookmark, 64
    Paste, 60
    Preferences, 67
    Previous Bookmark, 65
    Redo, 60
    Set/Clear Bookmark, 64
    Undo, 60
Edit Procedure, 101
Edit Procedure Interface, 100, 114
else, 306
Emergency backup, 326
Enable the context menu in the Graphics window, 81
Enable the mouse wheel in the Graphics window, 81
Enable tooltip showing coordinates and gray value at the mouse cursor position in the Graphics window, 81
encoding, 56
    native, 56, 77
    UTF-8, 56, 77
endfor, 308
endif, 306
endtry, 311
endwhile, 308
Enter Bounds..., 155, 173
error handling, 311
escape
    strings, 282
Example, 129
example programs, 52
exception
    handling, 312
    throw directly, 79
Exception handling, 337, 339, 340
executable expression, 305
Execute
    Activate/Deactivate Breakpoint, 86
    Attach To Process..., 85
    menu, 82
    Stop Debugging, 85
Execute Current Line, 115
Execute menu
    Abort Procedure Execution, 88
    Activate Profiler, 88
    Clear All Breakpoints, 86
    Clear Breakpoint, 86
    Manage Breakpoints, 86
    Reset Procedure Execution, 87
    Reset Profiler, 88
    Reset Program Execution, 87, 284
    Run, 82
    Run To Insert Cursor, 83
    Set Breakpoint, 86
    Show Runtime Statistics, 88
    Step Forward, 84
    Step Into, 84
    Step Out, 84
    Step Over, 83
    Stop, 84
    Stop After Procedure, 85
    Thread View / Call Stack, 85
Exit Assistant, 194, 215, 226, 260, 277

expected gray value range, 254
exponential
    functions, 299
Export, 68
Export Library Project..., 56
Export..., 55
    UTF-8 encoding, 56
external procedure, modified, 53
extract edges, 251

false, 347
Feature Histogram, 98, 113
Feature Inspection, 98, 113
feature processing, 253
feature selection, 253, 273
Fern Depth, 235
Fern Number, 235
Field, 195
File, 26, 163, 214, 224, 259, 276
    menu, 51
file history, 52
File menu
    Browse HDevelop Example Programs..., 52
    Close All Procedures, 53
    Close Procedure, 53
    Export Library Project..., 56
    Export..., 55
    New Program, 51
    Open Procedure For Editing..., 53
    Open Program..., 52
    Print..., 58
    Properties, 58
    Quit, 59
    Read Image..., 57
    Recent Programs, 53
    Save, 53
    Save All, 55
    Save Procedure As..., 54
    Save Program As..., 54
file types, 42
Find, 62, 168
Find Again, 64
Find Model, 240
Find Variable, 146, 148
Find/Replace..., 113
Fit Data Range, 176, 182
Fit Graph, 155
Font, 68
for
    loop, 308
for, 343, 347
Forward, 167
Forward in History, 114
frames per second
    Image Acquisition Assistant, 197
Full Screen, 110
fuzzy contrast, 256
fuzzy edge position, 256
fuzzy measuring, 255
fuzzy pair center position, 256
fuzzy pair gray mean, 257
fuzzy pair width, 257

gen_tuple_const, 291
General Documentation, 128
General Settings, 126
generate code, 223
Generic, 195
get_system, 344
Give Error, 80
Global, 144
global, 285
global variables, 285
Go to Line, 114
Go to Program Counter, 115
graphics window, 158, 344, 347
    clear, 89
    close, 89
    colors, 90, 91
    image size, 90
    line width, 91
    open, 88
    regions, 91
    select iconic variable, 89
    window size, 89
Gray Histogram, 98, 113
gray value
    histogram, 98, 173
    inspection, 97, 168
    status bar, 50
Greediness, 243
Group, 128
Guided Matching, 246

HALCON
    example programs, 52
    modules, 58
HALCON News (WWW), 111
HALCON Reference, 111
HALCON XL, 336, 341, 342
HALCONIMAGES, 57, 194
HALCONROOT, 57, 194
HDevelop
    dev_ operators, 102
    example programs, 52
    language, 281
    runtime error, 83
    warning, 191
HDevelop Language, 111
hdevelop program export, 335
HDevelop User's Guide, 111
HDevelop.ini, 68
Help, 110, 114, 120
    About, 111
    Check For Updates, 111
    Context Help, 110
    HALCON News (WWW), 111
    HALCON Reference, 111
    HDevelop Language, 111

HDevelop User's Guide, 111
menu, 110
Search Documentation, 111
Start Dialog, 110
Highlight Modified Variables, 70
Highlight Selection, 176
history
of files, 52
Home, 167
Hrun, 354

IC, 17, 113
iconic data, 347
iconic object, 347
if, 306, 347
ifelse, 306
Ignore first image of live acquisition, 197
image, 347
Image Acquisition Assistant, 24, 113, 193, 194
Code Generation, 197
Connection, 195
frames per second, 197
Inspect, 197
Parameters, 196
Source, 194
Image Acquisition Interface, 195
File, 26
image coordinates
status bar, 50
Image File(s), 194
Image Files, 198
Image Object, 198
image properties
status bar, 50
image pyramid, display, 228
Image Size, 90
image source, 203, 250, 264
Import, 68
In Single Step Mode, 96
Increase Zoom Factor, 168
Increasing Range, 154, 172
Indent Size, 68
Input Window, 175
Insert All As Local, 100
Insert Code, 198, 215, 248, 261, 278
Insert Code for Selection, 194
Insert Code..., 96
insert cursor, 17
Insert dev_display() into program, 146
Insert operator into program, 167
Insert Plot Code for Graphics Window, 155, 173
Insert Used As Local, 100
InsertCode, 176
Inspect, 147, 246
Image Acquisition Assistant, 197
Inspect as Handle, 147
Inspect as Tuple, 147, 155
inspect matching results, 223
inspection, 267
Interface Library, 196
Invalid Lines, 65

JIT compilation, 45
Jump to Link, 168

Keep dialog open, 53, 56
keyboard
menu access, 51
shortcuts, 355
usage, 13
Keywords, 106

LANG, 70
Language, 70, 128
Last Pyramid Level, 244
LC_ALL, 70
LC_COLLATE, 70
LC_CTYPE, 70
library project export, 331
line continuation, 116
Line Profile, 98, 113, 180
Line Width, 91, 177
Linear Scale, 154, 172
Linux, 336, 341
List Open Tabs, 114
Live, 27, 196
Load Assistant Settings, 214, 226, 260, 276
Load Assistant Settings..., 194
Load Camera Parameters, 225, 259
Load Camera Pose, 226
Load Image, 259, 276
Load Model, 225
Load Model Image, 225
Load OCR Classifier, 276
Load ROI from File., 261, 277
Load Test Images, 240
Load Training File, 276
local procedure, 53
local variables, 285
Locate page, 167
Logarithmic Scale, 154, 172
look-up table, 91
loop
body, 347
Loop Counter, 198
LUT, 34
Lut, 91

macOS, 336, 342
main procedure, 335, 336, 338, 340, 341
main window, 49
window title, 49
Make All External, 100
Manage Bookmarks, 65
Manage Breakpoints, 86, 121
Manage Passwords, 74
Manage Procedures, 101
Matching Assistant, 193

Max Deformation, 244
Max. Column Scale, 234
Max. Row Scale, 234
Maximum Number of Matches, 243
Maximum Overlap, 244
Measure Assistant, 193
measure task, setup, 250
Measuring, 260
menu, 51
  Acquisition (Image Acquisition Assistant), 198
  Assistants, 106
  Edit, 59
  Execute, 82
  File, 51
  Help, 110
  Operators, 101
  Procedures, 99
  Suggestions, 105
  Visualization, 88
  Window, 106
messages
  status bar, 50
method selection, 218
Metric, 237
Min. Angle and Max. Angle, 235
Min. Column Scale, 234
Min. Contrast, 238
Min. Row Scale, 234
Min. Scale and Max. Scale, 235
Min. Score, 235
Minimum Component Size, 232
Minimum Score, 242
Mode, 160
model creation, 219, 227
model image, load, 225
model parameters, advanced, 235
model parameters, standard, 231
model use parameters, advanced, 243
model use parameters, standard, 242
modified
  external procedure, 53
  program, 49
Modify Model Image, 227
Modify Regions, 165
Move Down, 124
Move tab to the left, 114
Move tab to the right, 114
Move Up, 124

Name, 121
native encoding, 56, 77
New Program, 51, 113
New Tab, 114
New Training File, 276
New Zoom Window, 98
Next, 168
Next Bookmark, 64
Next Link, 168
No output, 176
Normal, 197
number of visible objects, 241

OCR
  assistant, 193
  classifier, 268
  menu, 277
  Training File Browser, 98, 113
  training files, 98
Open, 53
Open Canvas Window, 109
Open Graphics Window, 106
Open Graphics Window..., 88
Open in new HDevelop, 53
Open Operator Window, 107, 119
Open Output Console, 50, 107
Open Procedure For Editing..., 53
Open Program Window, 106
Open Program..., 52, 113
Open Quick Navigation, 108
open test images, 240
Open Variable Window, 106
operation
  precedence, 302
Operations, 165
operator data base, 347
operator window, 138, 347
Operators
  Control, 101
  Develop, 102
  Legacy, 105
  menu, 101
  submenus, 104
  Unclassified, 105
Optimization, 337
Optimization, 237
optimize parameters, 220
Optimize Recognition Speed, 246
Output Window, 176

Paint, 91
parameter
  expressions, 286
parameter Angle Extent, 233
parameter Angle Step, 236
parameter Column Scale Step, 236
parameter Contrast, 232
parameter Descriptor Min. Score, 245
parameter Detector Type, 234
Parameter Documentation, 131
parameter Fern Depth, 235
parameter Fern Number, 235
parameter Greediness, 243
parameter Guided Matching, 246
parameter Last Pyramid Level, 244
parameter Max Deformation, 244
parameter Max. Column Scale, 234
parameter Max. Row Scale, 234
parameter Maximum Number of Matches, 243
parameter Maximum Overlap, 244

parameter Metric, 237
parameter Min. Angle and Max. Angle, 235
parameter Min. Column Scale, 234
parameter Min. Component Size, 232
parameter Min. Contrast, 238
parameter Min. Row Scale, 234
parameter Min. Score, 235
parameter Minimum Score, 242
parameter Optimization, 237
parameter optimization, 220
parameter Patch Size, 239
parameter Pyramid Levels, 233
parameter Row Scale Step, 236
parameter Score Type, 245
parameter Start Angle, 233
parameter Subpixel, 244
parameter Tilt, 239
parameter Timeout, 245
Parameters
    Image Acquisition Assistant, 196
parameters, 229
parameters Min. Scale and Max. Scale, 235
Password, 121
Paste, 60, 120
Patch Size, 239
PC, 17, 113
PC Down, 115
PC Up, 115
Pin the program listing, 114
pixel
    type, 169
pixel info, 97, 168
Plot as Function, 147
Plot as X/Y pair(s), 147
Plot Quality, 160
plot windows (interaction), 169
Port, 195
pose bounds, determine, 247
Position Precision, 97
Predecessors, 105
Preferences, 67
preferences
    export, 68
    HDevelop.ini (persistence), 68
    import, 68
Pregenerate Shape Model, 237
Preview Code, 254, 275
Previous, 168
Previous Bookmark, 65
Print..., 58, 121, 168
procedure, 335, 338, 340, 341
procedure compilation, 45
procedures, 41
Procedures menu
    Create New Procedure, 99
    Delete All Unused Local, 100
    Delete Current, 100
    Duplicate..., 99
    Edit Procedure, 101
    Edit Procedure Interface, 100
    Insert All As Local, 100
    Insert Used As Local, 100
    Make All External, 100
    Manage Procedures, 101
process features, 253
Profiler Display, 88
program counter, 17
Program Protection State:, 74
program window, 17, 113, 347
Properties, 58
Pyramid Levels, 233
pyramid levels, lock model and model image, 229
pyramid levels, model, 228
pyramid levels, model image, 229
pyramid, display, 228

quality issues, 206
quick setup, 264
Quit, 59

Read Image..., 57
Recent Programs, 52, 53
recognition rate, determine, 246
recognition speed, optimize, 246
Record Interactions, 96
Recursive, 194
Redo, 60, 113
reference to assistant elements, 214, 223, 258, 275
References, 130
Refresh, 66, 196
regexp_match, 296
regexp_replace, 297
regexp_select, 297
regexp_test, 297
region, 347
    colors, 90, 91
    empty, 347
    line width, 91
    shape, 91
regions
    visualization, 91
regular expressions, 296
relative file paths, 77
Release Generated Code Lines, 194, 215, 248, 261, 278
remote debugging, 327
Remove, 124, 155
Remove All Test Images, 240
Remove from User Tab, 146, 148
Remove Test Image, 240
repeat, 308
replace
    Find/Replace..., 62
Replace selected program lines, 125
reserved words, 305
Reset, 68, 124
reset, 268
    graphics window, 95
Reset All, 27, 196

Reset Bounds, 154, 155, 172, 173
Reset Parameters, 95
Reset Procedure Execution, 87, 113
Reset Profiler, 88
Reset Program Execution, 87, 113
Reset to normal size, 168
Resolution, 195
Restore Default Layout, 109
Restrictions, 337, 338, 340
results, 274
results tab, 252, 272
ROI Type, 164
row, 50
Row Scale Step, 236
Run, 82, 113
Run To Insert Cursor, 83
Run Until Here, 119
runtime
    status bar, 50

Save, 53, 113, 146, 148
save
    local procedure, 53
Save All, 55, 113
Save Current Assistant Settings, 214, 226, 260, 277
Save Current Assistant Settings..., 194
Save Model, 225
Save Procedure As..., 54
Save Program As..., 54
Save ROI to File, 261, 277
Save Training File, 276
Save Window..., 99
scale range, 234
Scale Selection, 176
scale step size, 236
scope (of variables), 285
Score Type, 245
Scroll down, 115
Scroll down page-wise, 115
Scroll up, 115
Scroll up page-wise, 115
Search Documentation, 111
search object, 240
search parameters, advanced, 243
search parameters, standard, 242
See also, 105
Select a Procedure, 114
Select Directory ..., 195
select edges, 252
select features, 253
Select File(s) ..., 195
Select Graphics Window, 92
select method, 218
select test image, 241
Select..., 195
semantics, 281
sequence file, 26
set
    operations, 297
    up, 250
Set 1:1 Aspect Ratio, 154
Set 2D Calibration, 165
Set Insert Cursor, 114, 120
Set Parameters, 92
Set Program Counter, 114, 120
Set Reference, 240
Set/Clear Bookmark, 64, 120
Set/Clear Breakpoint, 86, 114, 120
Set/Clear Breakpoint on Variable, 120
    control, 148
    iconic, 146
set_system, 344
setup, 264
Shape, 91
shape model, load, 225
shape model, save, 225
Shape models may cross the image border, 245
Short Description, 129
shortcuts, 13
Show Background Grid, 155, 173
Show Caller, 120
Show Code Preview, 194, 215, 249, 261, 278
Show frames per second during live acquisition, 197
Show Function Value At X, 155, 173
Show Function Value At Y, 155, 173
Show main, 115
Show Min/Mean/Max, 197
Show Mouse Position, 155, 173
Show Next Tab Card, 114
Show Previous Tab Card, 114
Show Procedure, 120
Show Procedure in New Tab, 120
Show Procedure in New Window, 120
Show Processing Time, 50, 80
Show Runtime Statistics, 88
Snap, 27, 196
Sort by Name, 146, 148, 196
Sort by Occurence, 146, 148
Source
    Image Acquisition Assistant, 194
Source
    image, 24
speed up descriptor-based matching, 223
speed up matching, 220
speed up matching, correlation-based, 221
speed up matching, deformable, 222
speed up matching, shape-based, 221
split, 296
standard model parameters, 231
standard search parameters, 242
Start Angle, 233
Start Dialog, 110
start dialog, 16
status bar, 50
Step, 160
Step Forward, 84

Step Into, 84, 113
Step Out, 84, 113
Step Over, 83, 113
Stop, 84, 113, 196
stop
    HDevelop program, 83
    program, 83
Stop After Procedure, 85
Stop Debugging, 85
str_firstn, 296
str_lastn, 296
strchr, 295
string, 347
    concatenation, 287, 295
    operations, 292
    special characters, 282
strlen, 296
strrchr, 295
strrstr, 295
strstr, 295
Style, 155
Subpixel, 244
Successors, 105
Suggestions, 129
    Alternatives, 105
    Keywords, 106
    Predecessors, 105
    See also, 105
    Successors, 105
suppress error messages, 79
Swap (X, Y), 155
symbol appearance, 266
symbol fragmentation, 266
symbol shape, 266
symbol size, 266
syntax, 281

tab results, 252, 272
teaching, 269
terminology, 12
test image sequence, remove, 240
test image, display, 241
test image, remove, 240
test image, select, 241
test image, set reference, 240
Test Images, 239
test images, load, 240
test model, 219, 239
text layout, 267
text orientation, 266
Themes, 71
Thickness, 155
Thread View / Call Stack, 85, 113
throw, 311
Tilt, 239
Timeout, 245
To Object / To Tuple, 125
To Vector, 125
Tools, 97
Train Now, 277
training, 269
Trigger, 195
trigonometric
    functions, 299
true, 347
try, 311
tuple, 347
    concatenation, 288, 289
Type, 121
type, 348
    boolean, 347
    integer, 347
    real, 347
    string, 347

Undo, 60, 113
Ungroup, 155
Unicode, 56
Unit, 165
unnamed, 49
unsaved changes, 49
until, 308
Update Graphics Window, 81
Update Image, 196
Update Program Counter, 81, 120
Update Variables, 81, 146, 148
Update Window, 96
Use Model, 239
User, 144
User-defined Range, 154, 172
UTF-8 encoding, 56, 77

variable names, 254, 274
variable types, 284
variable window, 17
variable window, 142, 348
    layout, 143
    resize, 143
    tabs All, 144
    tabs Auto, 144
    tabs User, 144
variables, 284
view image pyramid, 228
view test image, 241
Visibility, 196
visible objects, 241
Visual Basic .NET, 11
Visual Basic .NET export, 339
Visualization
    menu, 88
    Record Interactions, 96
Visualization menu
    Apply Changes Immediately, 92
    Clear Graphics Window, 89
    Close Graphics Window, 89
    Color, 91
    Colored, 90
    Display, 89
    Draw, 91
    Feature Histogram, 98, 178

Feature Inspection, 98, 179
Gray Histogram, 98, 173
Image Size, 90
Insert Code..., 96
Line Profile, 98, 180
Line Width, 91
Lut, 91
New Zoom Window, 98
OCR Training File Browser, 98, 185
Open Graphics Window..., 88
Paint, 91
Position Precision, 97
Reset Parameters, 95
Save Window..., 99
Set Parameters, 92
Shape, 91
Update Window, 96
Window Size, 89
Zoom Window, 97, 168
Volatile, 197

Warning, 129
warning, 191
while
loop, 308
while, 344, 347
Window
Cascade Floating Windows, 109
menu, 106
Open Graphics Window, 106
Open Operator Window, 107
Open Output Console, 107
Open Program Window, 106
Open Quick Navigation, 108
Open Variable Window, 106
Window Size, 89
window title, 49
Windows, 336, 341
word processing, 273

X, 195
XLD, 348
colors, 90, 91
line width, 91

Y, 195

Zoom, 168
Zoom in, 115
Zoom In Mode, 155, 173
Zoom out, 115
Zoom Out Mode, 155, 173
Zoom To Selection, 176, 182
Zoom Window, 97, 113