HALCON
a product of MVTec

HDevelop User's Guide

HALCON 25.11 *Progress*

MVTec Software GmbH

Building Vision for Business

HDevelop, the interactive development environment of HALCON, Version 25.11.0.0

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 © 1997-2025 by MVTec Software GmbH, Munich, Germany

Protected by the following patents: US 7,751,625, US 7,953,290, US 7,953,291, US 8,260,059, US 8,379,014, US 8,830,229, US 11,328,478. Further patents pending.

OpenGL® and the oval logo are either trademarks or registered trademarks of Hewlett Packard Enterprise in the United States and/or other countries worldwide.
Linux ® is a registered trademark of Linus Torvalds.
Microsoft, Windows, Microsoft .NET, Visual C++ and Visual Basic are either trademarks or registered trademarks of Microsoft Corporation.
Python® is a registered trademark of PSF.

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

More information about HALCON can be found at: https://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.

Symbols

The following symbols are used within the manual:

This symbol indicates a **tip**.

This symbol indicates an information you should **pay attention** to.

Contents

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, for example, 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 137.
- *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, for example, 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, for example, 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, for example, 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, for example, program lines, or to draw shapes.

drag-and-drop HDevelop supports drag-and-drop of image files, dictionaries, 3D object models, DL models, and HDevelop programs from other applications. You can, for example, drag an HDevelop program from a file browser and drop it on HDevelop 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 327.

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, for example, 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.

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 319)

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 Start menu.

You can also start HDevelop from the Windows command prompt or from the `Run` app (for example, press Windows logo key + R). This makes it easy to pass optional command line switches:

```
hdevelop
```

Linux

Under Linux, HDevelop is started from the shell:

```
hdevelop &
```

2.2 Start Dialog

Figure 2.1: Start dialog.

1. Quick access to HDevelop programs
2. Introductory material
3. Documentation

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

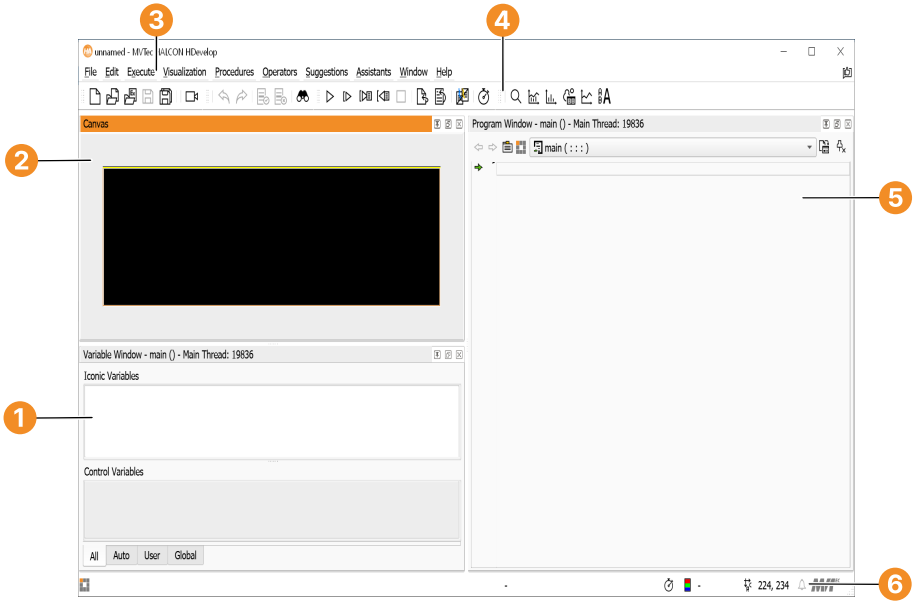


Figure 2.2: User interface.

When HDevelop is started for the first time it looks similar to figure 2.2. 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 160).

You can double-click iconic variables to display them in the active graphics window. Double-clicking control variables opens an inspection window with a formatted list of the current values and statistical data. See also Inspecting Variables (page 167).

2 Canvas window with graphics window

The canvas window is a container for graphics windows. It contains one empty graphics window, at first startup of HDevelop. Graphics windows are used to display iconic data like images, regions, and XLDs, for more information see also Graphics Window (page 73). Each graphics window provides its own toolbar to adapt the image display to your needs and to, for example choose between different zoom steps. 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 6.4 on page 66.

3 Menu

4 Tool bar

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 127).

When adding new lines or modifying existing lines, advanced autocompletion (page 130) 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.

6 Status bar

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 51). In the default layout, the program window, the graphics or canvas window, and the variable window are already docked. You can always restore your favorite default layout by clicking `Window` ▷ `Restore Default Layout Use` ▷ `Canvas` or ▷ `Restore Default Layout Use` ▷ `Docked Graphics Window`.

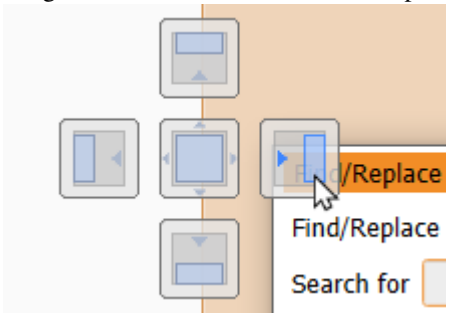
To dock a floating window, do the following:

1. Click and move the title bar of the window.
   → Drop indicators appear, showing possible docking positions: docking to the left, right, upper, or lower

part. The indicator in the middle tabs the window. The window joins the already docked window as a tab card.[1]

i

2. **When dragging the window and holding down the Ctrl key, the drop indicators disappear.**
3. Drag the window over the desired drop indicator.



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:

- Drag the window at the title bar/tab card and drop it apart from the drop indicators. The window will keep its current size.
- Or click Undock window in the top right corner of the window or double-click 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 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 6.4 on page 66.

[1] 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 or the other way round.

2.5 Running Example Programs

HALCON comes with many 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.

“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).

For more information on how to run programs with a floating canvas or graphics window, see section 6.4.1 on page 66.

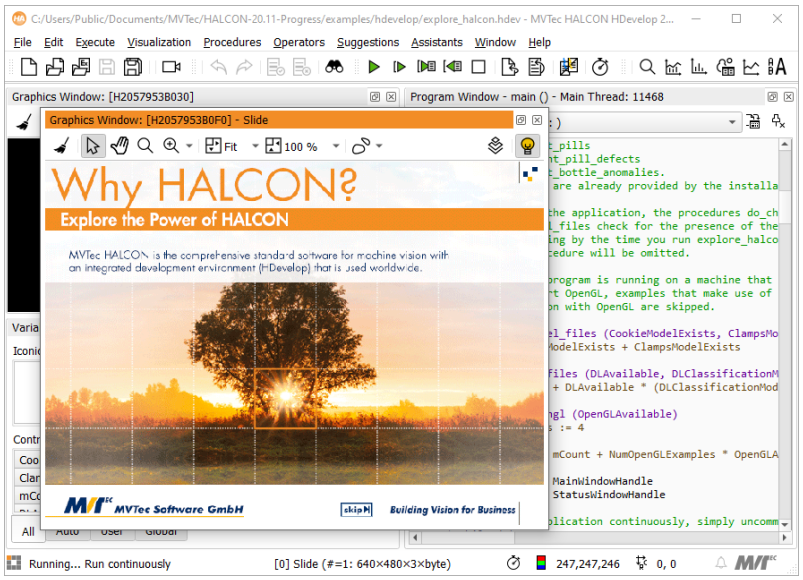


Figure 2.3: Explore the power of HALCON.

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.

To open the example program browser, click `File` ▷ `Browse HDevelop Example Programs...`
For more information, see section 6.5 on page 69. Click ▷ to run the program.

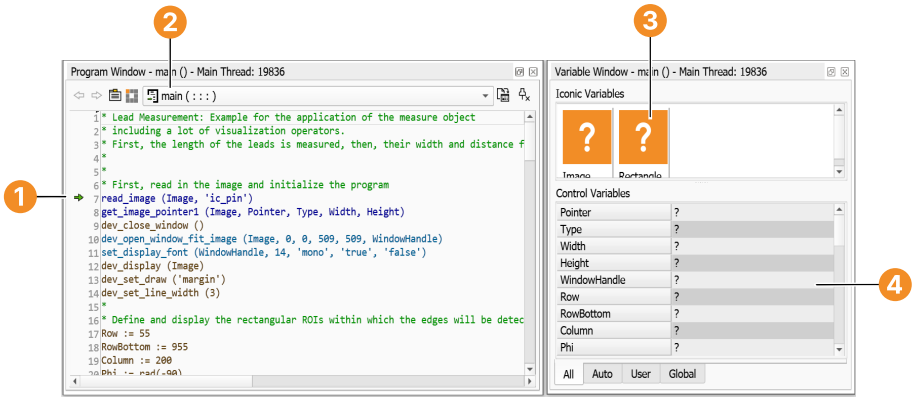


Figure 2.4: The variable and program window after loading the example program.

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, meaning their current value is undefined. This is indicated by the question mark (?). Both windows are displayed in figure 2.4.

Run Example Program

- Click `Execute` ▷ `Run` or click the corresponding button 1 from the tool bar (see figure 2.5).

  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 2.5).
- Using the button `Step Over` (2) you can execute the program line by line and inspect the immediate effect of each instruction.

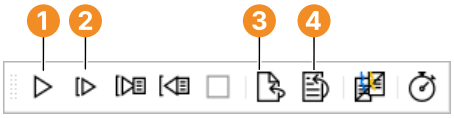


Figure 2.5: The basic execution buttons.

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
```

See appendix C.1 on page 323 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 during prototyping phase you often have a set of sample image files to work from. HDevelop supports an extensive variety 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).

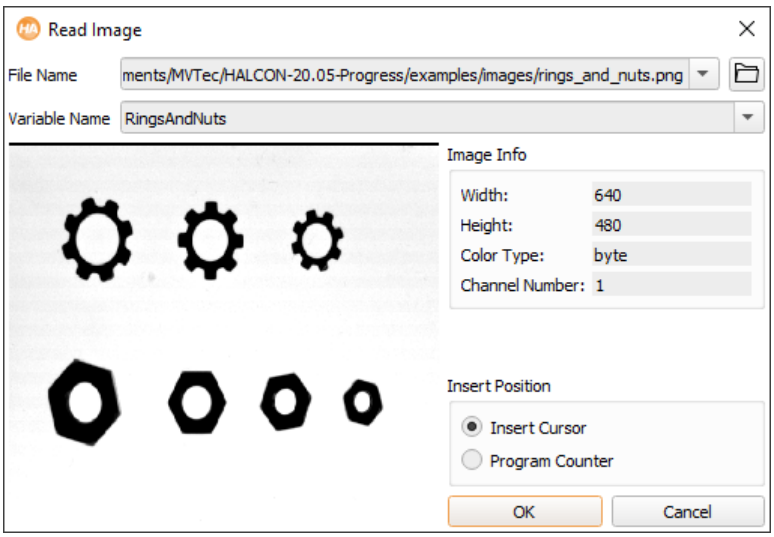


Figure 3.1: Read Image Dialog.

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 can 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. 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

Select File ▷ Read Image... to get the dialog described above.

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) is 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 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).

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, but the automatic display of images can be suppressed if desired, for example, 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 75) 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, Visualization ▷ 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 163).

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.

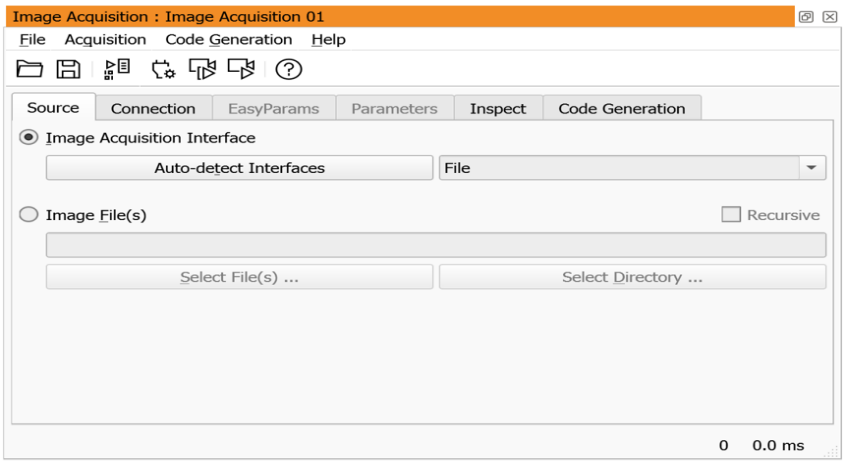


Figure 3.2: Image acquisition assistant.

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, for example, a camera. This is described in section 3.3.2 on page 27.

3.3.1 Acquiring Images From Files or Directories

You can specify a selection of image files or also drag and drop a complete 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.

`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`.

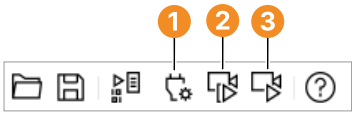


Figure 3.3: Browsing the selected images.

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.

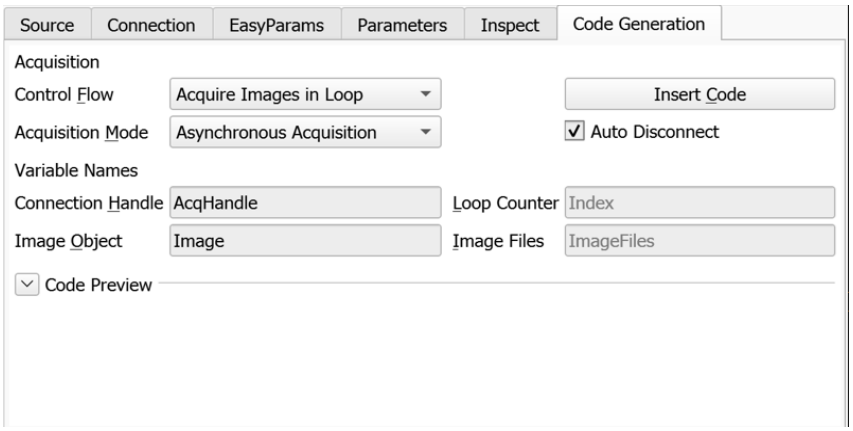


Figure 3.4: Specifying variable names for code generation.

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.

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 wrongly 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 cancel the process. 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.

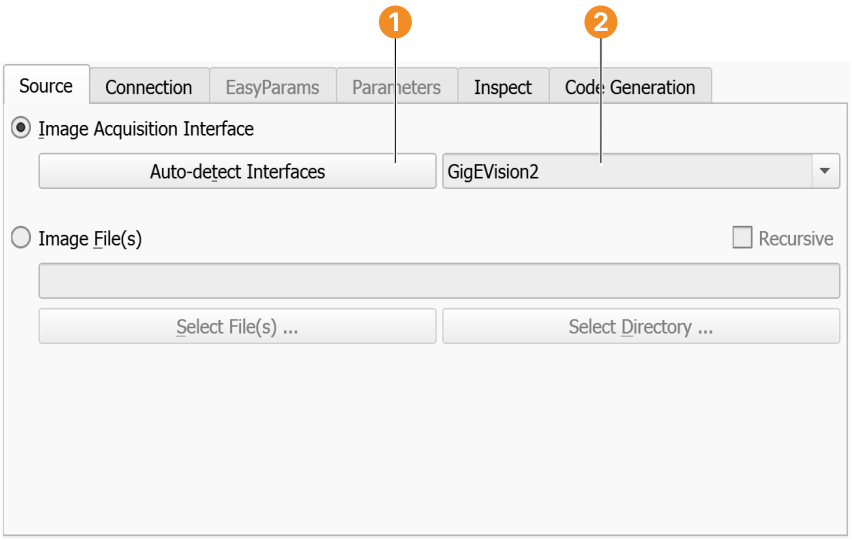


Figure 3.5: Source selection (example).

(1) `Auto-detect Interfaces`

(2) Interface list

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.

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 26).

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.

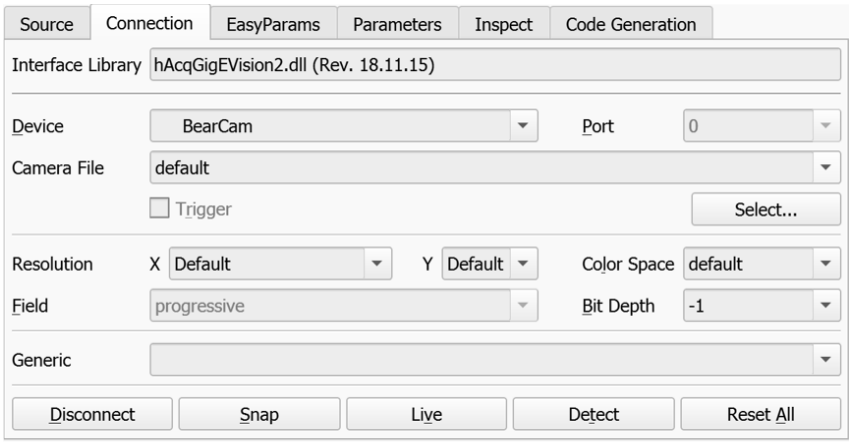


Figure 3.6: Connection parameters (example).

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.

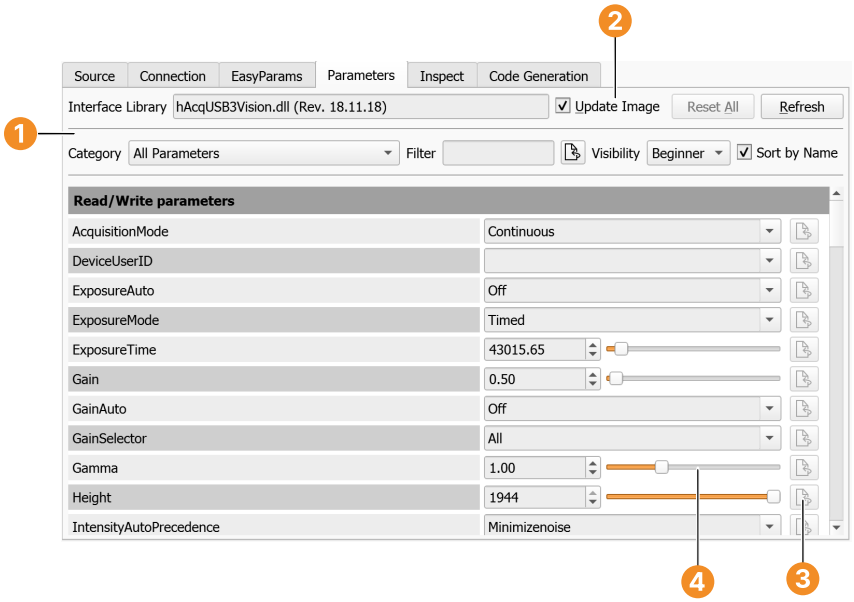


Figure 3.7: Device-specific parameters (example).

(1) Range of parameters

(2) `Update Image`

(3) `Refresh`

(4) Slider for definded range of values

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` (2) is checked, 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`.

EasyParams

The tab card `EasyParams` contains a preselection of the most important camera parameters. These parameters are, in most cases, enough to rapidly obtain an acceptable image in the `Image Acquisition Assistant`. Because the `EasyParams` rely on the GenICam features of your device, the tab card will only be available for GenICam-based image acquisition interfaces, and after connecting to a device.

Each one of the `EasyParams` internally controls multiple GenICam features of the connected device. See figure 3.8 for an example parameter list. Refer to Image Acquisition Interface Reference for detailed information. You can click the help button of the assistant to get to the corresponding page automatically.

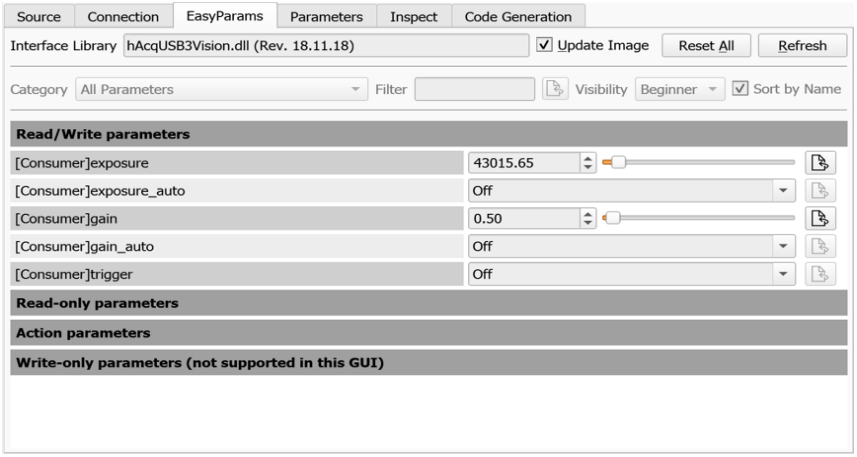


Figure 3.8: EasyParams.

Generate Code

The tab card `Code Generation` turns the settings made in the other tab cards 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)
```

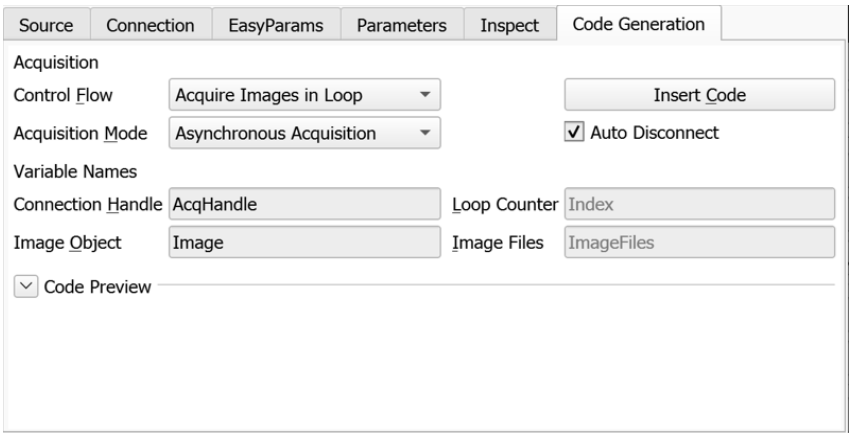


Figure 3.9: Code generation.

```
while (true)
    grab_image_async (Image, AcqHandle, -1)
    * Image Acquisition 01: Do something
endwhile
close_framegrabber (AcqHandle)
```

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 by using the menu entry `File ▷ Close Dialog` or 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 link 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 With HDevelop

This chapter explains how to use HDevelop to develop your own machine vision applications. Preferably, actively follow the examples 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.

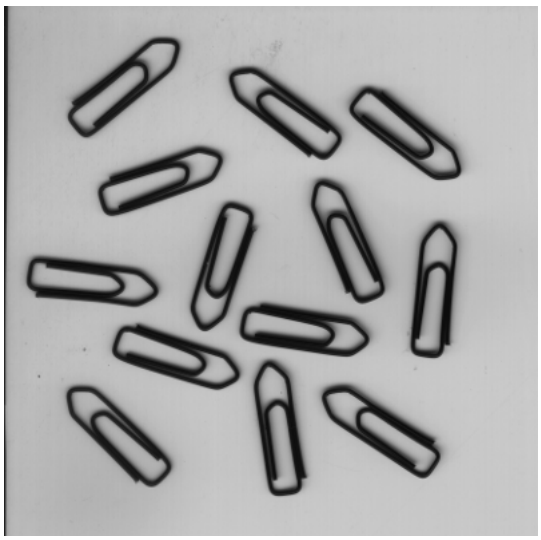


Figure 4.1: Paper clips.

4.1 Starting 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 chapter 3 on page 23 we already 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 Entering 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 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.

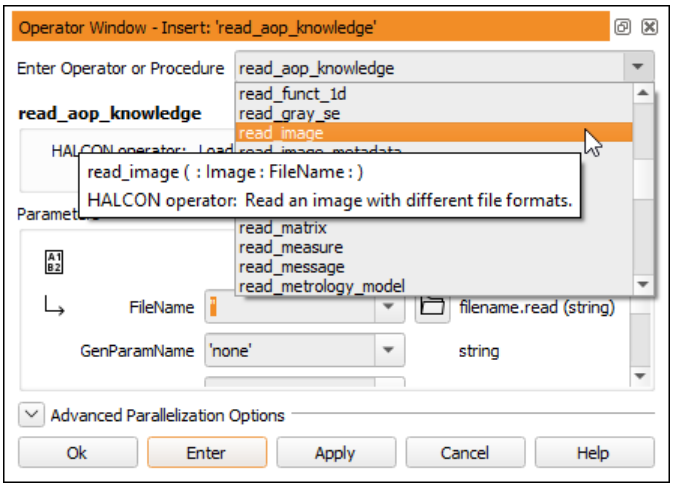


Figure 4.2: Matching operators after typing `read_` and pressing Return.

4.3 Specifying 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 (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.

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, and 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.

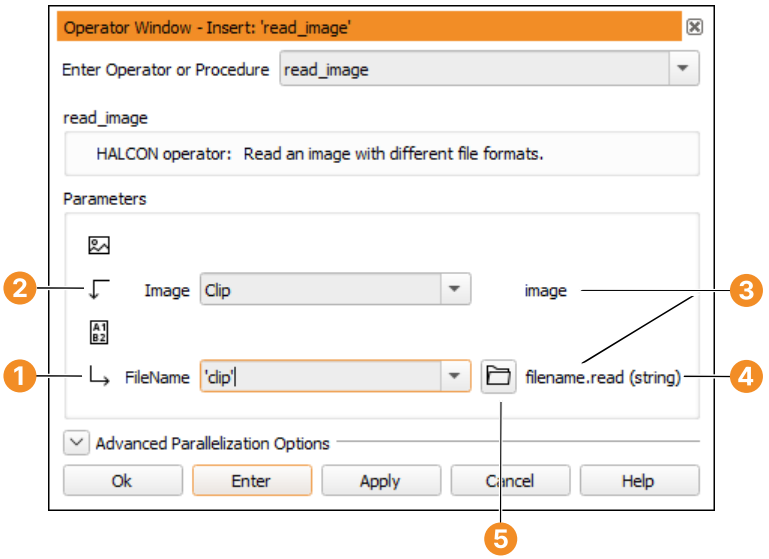


Figure 4.3: Specifying parameters.

1. Iconic output parameter
2. Control input parameter
3. Semantic type
4. Data type
5. File selection dialog

4.4 Getting Help

1. Double-click the first program line in the program window. The operator is displayed in the operator window for editing.
2. Click `Help` to open the HDevelop online help window. It will automatically jump to the documentation of the displayed operator. The reference manual is 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.
3. Enter any desired substring into `Find` to quickly find an operator.

The online help window is described in section 6.9 on page 81.

4.5 Adding Program Lines

Threshold

Thresholding is the step to separate the clips from the background, based on the gray value.

1. Enter `threshold` into the operator window. This is both the full name of an operator and part of other operator names.
2. Pressing Return shows a pre-selected list of matching operators with `threshold`
3. Press Return again 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.

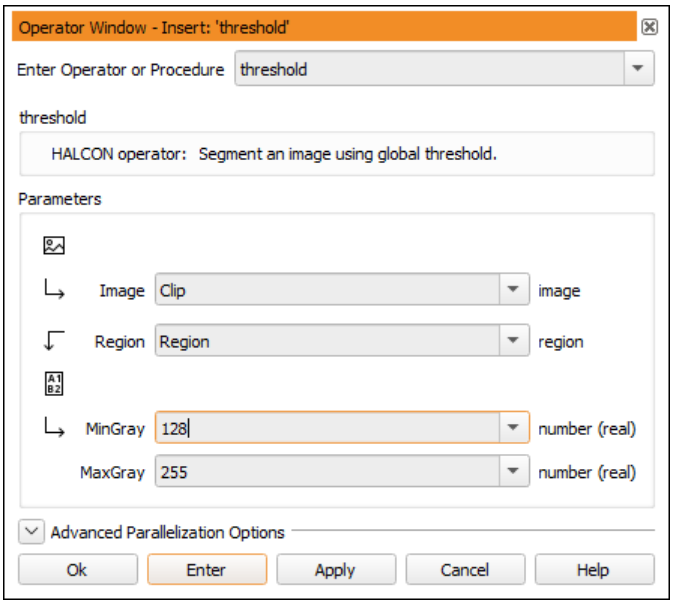


Figure 4.4: Parameter suggestions.

- 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, different settings can be tried and the result can be seen immediately. The selected pixels 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.
- The region is also displayed as an overlay in the graphics window. The selected pixels are displayed in red by default.
- 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 input parameters use variable names that will be added to the program at a later time.

Successor

- Click 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. Any operator selected through this menu is transferred to the operator window.
- Click the operator `connection`.
- By clicking `OK`. Two program lines are executed: The `threshold` operation and the `connection` operation. 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 the program, the graphics window displays three layers of iconic variables:

- the image `Clip`,
- the region `Region`,
- and the tuple of regions `ConnectedRegions`.

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 (`Lut`) and the display mode (`Paint`) can be set. The LUT specifies gray value mappings. Right-clicking 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` is still displayed in filled red.

To set many display properties at once click the menu entry `Set Parameters`. It opens the settings window displayed in figure 4.5.

Clicking the button `Reset` restores the default visualization settings.

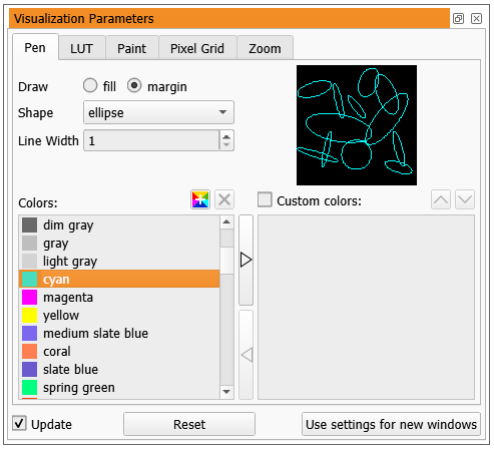


Figure 4.5: Changing the display parameters.

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 the graphics window) manually, click 🧹, see figure 6.10 on page 74.

4.7 Inspecting Variables

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

Right-click 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,

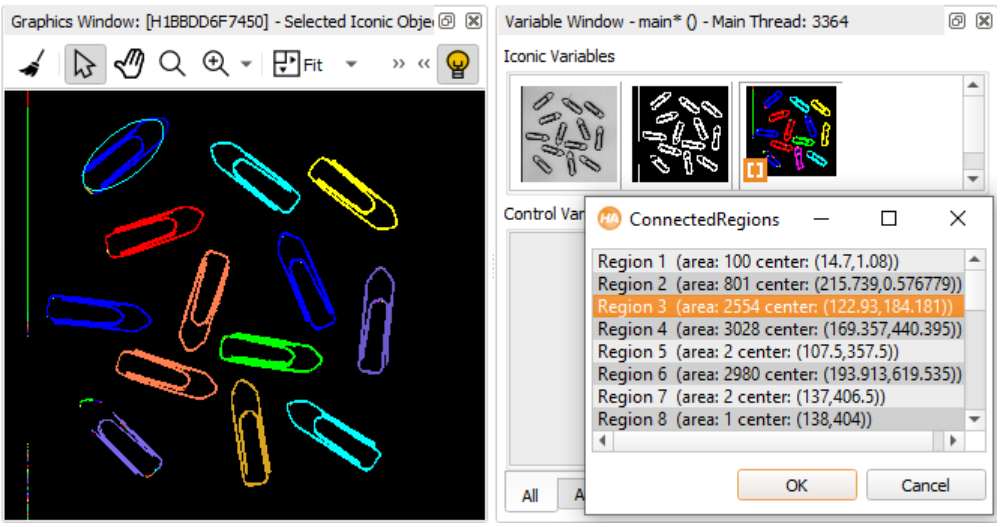


Figure 4.6: Interactive inspection of an iconic variable containing regions.

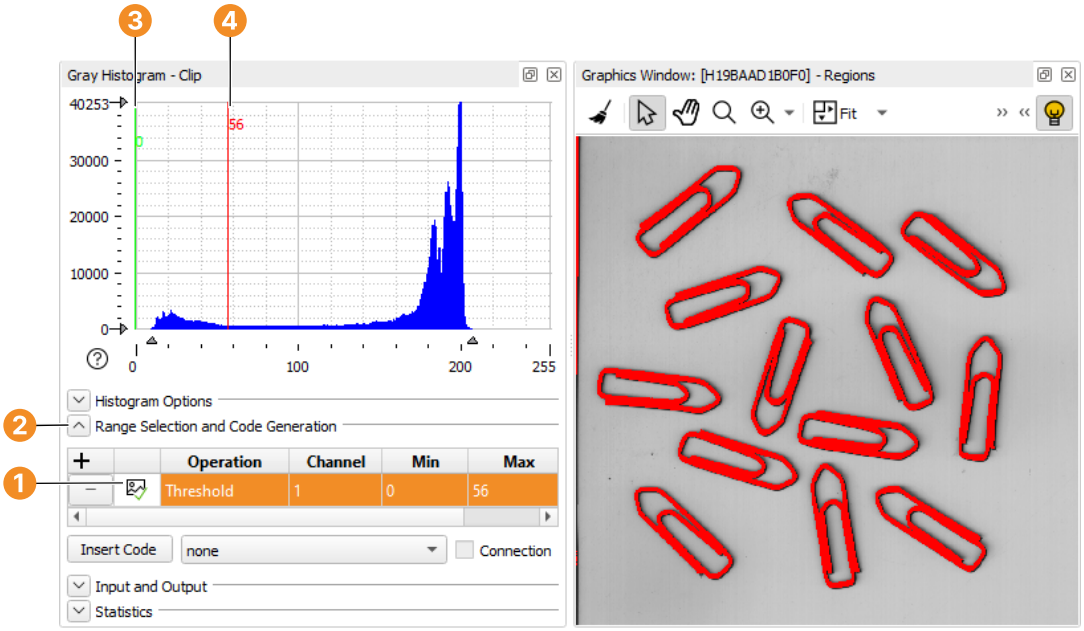


Figure 4.7: Determining threshold bounds interactively using the gray histogram.

1. `Threshold`
2. `Range Selection and Code Generation`
3. `Min` and `Max` Green line
4. `Min` and `Max` Red line

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.

Make sure `Range Selection and Code Generation` is visible in the histogram window, see figure 4.7. Select `Threshold` in the column `Operation` of the gray histogram window, and click the icon next to `Threshold` to visualize the operation. Now, you can try different threshold bounds by altering the values in `Min` and `Max` or by dragging the green and the red line 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 Editing 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 6.17.2 on page 128).

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 Executing a 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.

The processing time of the operator or procedure call is indicated in the status bar at the bottom (unless disabled).

Additional Information

Note the following when executing a program via `Execute` ▷ `Run`:

- 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.
- A breakpoint, stop instruction, or runtime error may interrupt the execution of your program; see below.
- 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 with a right click.

Further, the following **characteristics** apply during the execution of operator or procedure calls:

- You can initiate limited activities. For example, if you double-click variables in the variable window they will be visualized. You can modify parameters for the graphics windows as described in the menu `Visualization`. You can even modify the current procedure body.
- All user interaction except `Stop` is disabled during program execution if the latter was not started in the main procedure. HDevelop may respond slowly 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 6.16.14 on page 125). In any case, the values of all variables are shown in the variable window after the termination of the execution.
- While the program is running, the menu items `Run`, `Step Over`, `Step Into`, and `Step Out` (and the corresponding tool bar buttons) cannot be executed.
- Suggestions in the menu `Suggestions` are determined for the recently executed operator.
- After executing a program (regardless whether via `Run` or stepwise via `Step Over`), HDevelop is available for further transactions. Any user input that has been made during execution is handled then.

A running HDevelop **program stops** if one of the following conditions is met:

- You click `Execute` ▷ `Stop` (or the corresponding tool bar button).
- The last operator or procedure call in the current program (the main procedure body) is called. This is the usual way a program terminates. The PC is positioned behind this operator.

- A breakpoint is reached (see section 6.17 on page 127). 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 6.21.1 on page 162).
- A runtime error occurs. An input variable without a value or values outside a valid range are 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 section 5.6 on page 47) are obeyed. However, the code of the locked procedure will only be visible if the correct password is entered in the program window.
- The program waits via the operator `wait_seconds`. In this case, clicking `Run`, `Step Over`, or `Step Forward` continues the program immediately.
- HDevelop is closed.

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

4.11 Saving a 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 5; 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.

Similar to the gray histogram window, you can visualize the selected regions. 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 and red line 4 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)
```

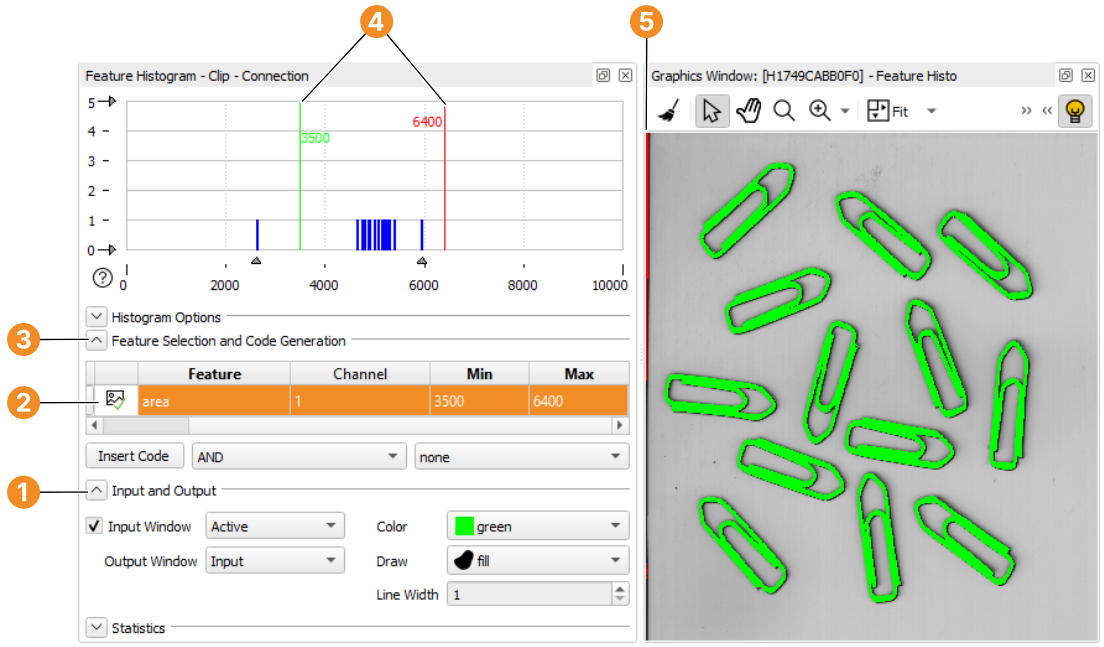


Figure 4.8: Selecting regions with a similar area in the feature histogram.

1. `Input and Output`
2. Click to enable visualization
3. `Feature Selection and Code Generation`
4. Green and red line that affects the selected regions
5. Contiguous area at the left edge of the image

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).

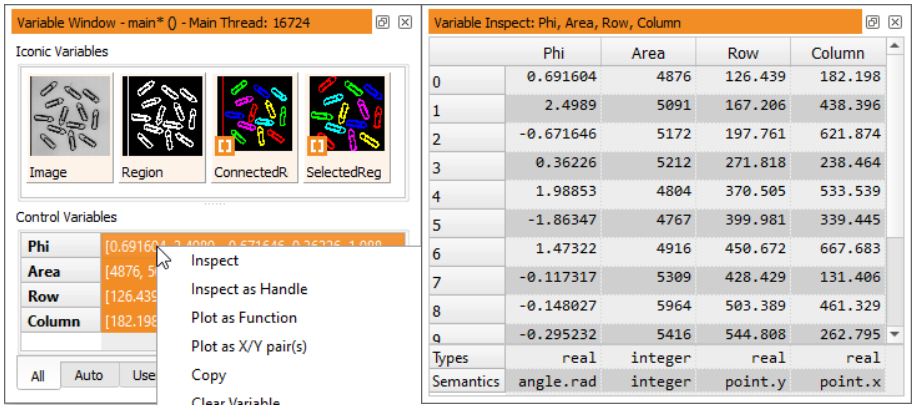


Figure 4.9: Inspecting control variables.

4.13 Opening the 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.

Generating Code for the Settings of a Graphics Window

You can use the `Insert Code...` command to insert the piece of code into the current program that will restore the current state of the graphics window or parts of it.

1. Prepare the state of the graphics window to use. For more information, see section 6.8 on page 73.
2. In the program window, place the cursor where you want to insert the code.
3. Click `Visualization` ▷ `Insert Code...`.
4. In the dialog, specify the settings to add:

   `Iconic objects from the graphics stack`

   All iconic objects that are currently visible in the active graphics window

   `Visualization parameters`

   The draw colors, draw mode, line width, region shape, LUT, and paint mode

   `Window geometry`

   The window size and the visible part

5. Click `Insert`.

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 247.

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', [], [])
```

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, for example, to find appropriate values for the operators.

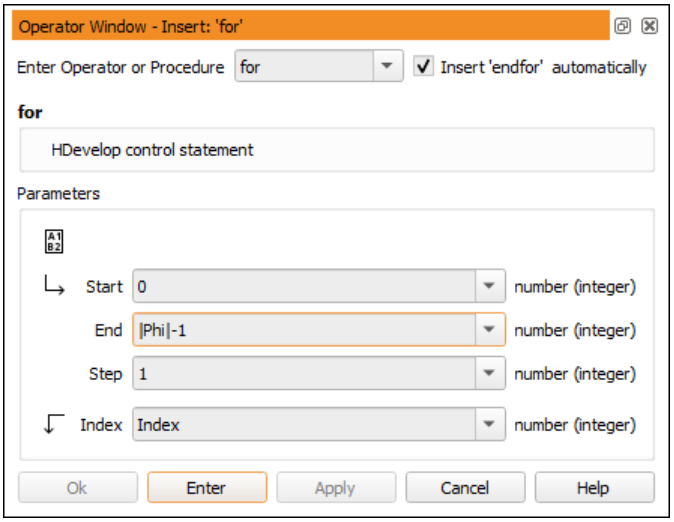


Figure 4.10: Entering a loop in HDevelop.

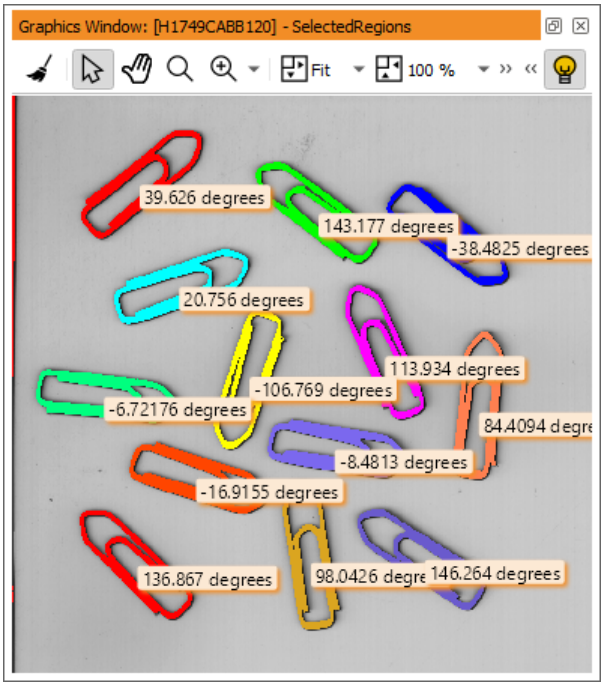


Figure 4.11: Final result of the example program.

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. 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.

- Each program file contains exactly one main procedure and zero to multiple local procedures.
- **Local procedures** are stored inside the HDevelop program and cannot be called from other programs or external procedures.
- **External procedures** are stored as separate files, and can be shared between different HDevelop programs. A modification immediately affects all HDevelop programs using it when reloaded. Each procedure file contains a single external procedure. The file name determines the name of the procedure. Thus, if the file name is changed, programs and other procedures using it will have to be adapted.
- **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 46).

5.2 File Types

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

5.2.1 File Tracking

When a file is changed outside of HDevelop, the file tracking alerts the change and offers to reload the file. Click `Ignore` to abort. To reload the changed files without using the file changes dialog, click `Reload All` within the File tabcard. For more information, see section 6.16.12 on page 123.

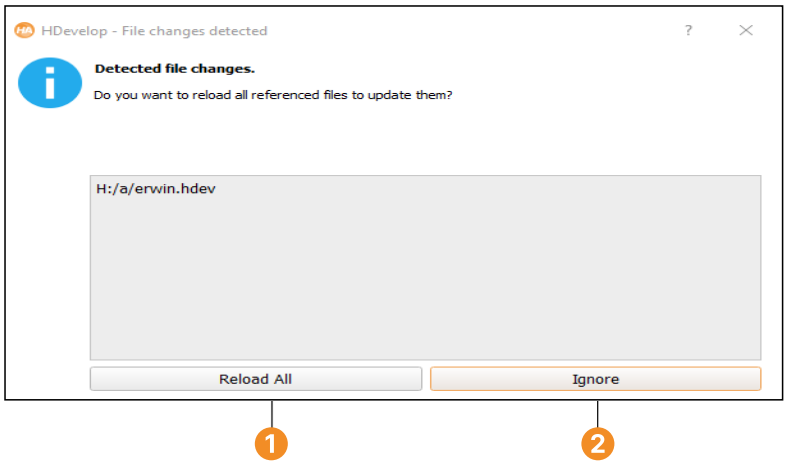


Figure 5.1: File tracking change dialog.

1. Reload All
2. Ignore

5.2.2 HDevelop Programs

.hdev This is the default file format for HDevelop programs. The "main" procedure and all local procedures. It stores programs in XML format and is suitable for revision control software.

.dev (Legacy) 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.

5.2.3 Procedure Files

.hdvp 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 (Legacy) 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 43).

5.2.4 Libraries

.hdpl This is the file format for procedure libraries. It stores libraries in XML format and is suitable for revision control software.

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.

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

In 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, ";" on Windows systems and ":" on Linux 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. 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. 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, such like 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

   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

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

   (a) Session locations:

   ```
   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:

- 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.
- 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 6.17.9 on page 146.

To manage the protection state of multiple procedures, see section 6.16.8 on page 119.

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. Procedures can be ordered by chapters and sections just like operators.

The description of procedures can be formatted using Markdown syntax.

See section 6.17.8 on page 142 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 6.17.8.1 on page 142.

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 6.16.12 on page 123). 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 6.1.9 on page 63). 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 (for example, 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. 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 6.16.12 on page 123).

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, for example, 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, for example, the inspection of variables. However, for thread IDs returned by the qualifier `par_start` (see section 8.11.1 on page 283), 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 6.17.8.2 on page 145.

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.

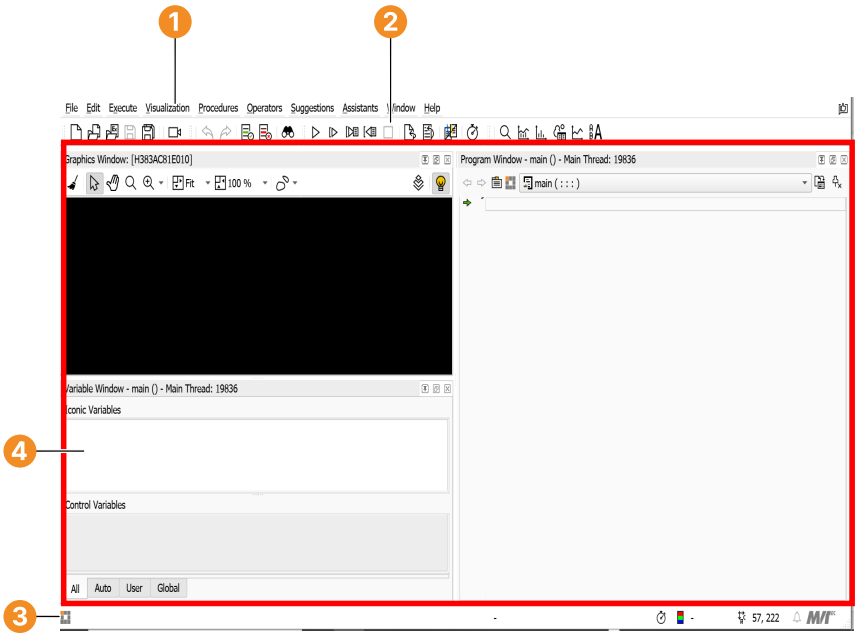


Figure 6.1: HDevelop main window.

The HDevelop main window contains the following components: The window title shows the given name of the current program. Unsaved changes in the current program are indicated with a trailing asterisk (*) in the window title.

- 1 Menu

  The menu at the top provides access to the functionality of HDevelop. The menus and their entries are described in section 6.1.
  At the right end of the menu, there is a feedback button.

- 2 Tool Bar

  The tool bar icons provide convenient shortcuts for frequently used functions, see section 6.2 on page 64.

- 3 Status Bar

  The status bar displays 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 6.16.14 on page 125).
  For some very time-consuming operators, like `trainf_ocr_class_mlp` or `train_class_svm`, the status bar also displays a progress bar for more information see section 6.3 on page 65.

- 4 Window Area

  The main part of the window is reserved for the dockable windows and dialogs of HDevelop, like the Canvas Window, the File Selection Dialog, the Graphics Window, and so on.

6.1 Menu

The menu of the main window provides access to the complete functionality of HDevelop. Every menu item opens a drop-down menu with optional submenus. You can either access the menu by clicking an item, or 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.

6.1.1 Menu `File`

This menu provides functions to load existing programs and to save recently created or modified programs and procedures. See section 5.2 on page 44 for the supported file types. Here, HDevelop programs can be exported to C++, C, Visual Basic.NET, C#, or plain text and also be printed. The menu also provides access to the supplied example programs and allows you to read arbitrary images.

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 dialogs `Save Program As...` or `Save Procedure As...`. 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 121).

Menu Entry	Description	Shortcut
New Program	Initialize a new HDevelop program.	Ctrl+N
Open Program...	Load an existing HDevelop program. After you have loaded a program, the corresponding file name is added to the top of the menu `Recent Programs`.	Ctrl+O
Browse HDevelop Example Programs...	Load HDevelop example programs grouped by categories. To scan through several example programs quickly, select `Keep dialog open`. More info (page 69)	Ctrl+E
Recent Programs	Load recently used HDevelop programs. The number menu entries can be customized in the preferences (see `General Options` ▷ `General Options`).	
Open Procedure For Editing...	Load an external procedure or library. 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. More info (page 45)	
Close Procedure	Closes the procedure, 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 procedure is not reachable via the procedure path, it will be closed.	
Close All Procedures	Closes all 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.	

Menu Entry	Description	Shortcut
Save	Saves changes of the current HDevelop program or the currently selected external procedure. Local procedures are saved within the HDevelop program. To save modified external procedures as well, select the menu entry `Save All`, or select the corresponding procedure in the program window and click `Save` again. A modified external procedure is marked with an asterisk (*) in the program window.	Ctrl+S
Save Program As...	Save the current HDevelop program to a new file. The file name of the program you save is added to the menu `Recent Programs`.	Ctrl+Shift+S
Save Procedure As...	Save the current procedure as... ...an external procedure (`*.hdvp` or `*.dvp`): 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. ...an HDevelop program (`*.hdev` or `*.dev`): An empty main procedure is added to the target file, and the procedure is added to the program as a local procedure.	Ctrl+Shift+P,S or Ctrl+Shift+P,Ctrl+Shift+S
Save All	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.	Ctrl+Alt+S
Export Program...	Export program code to a programming language or as a text file. More info (page 69)	Ctrl+Shift+O,X or Ctrl+Shift+O,Ctrl+Shift+X
Export Library Project...	Export library/local procedures as a C++ or C# project. More info (page 303)	
Read Image...	Add an image file to the current program. More info (page 24)	Ctrl+R
Properties	Display various properties of the current program. More info (page 111)	
Print	Print the current program or selected procedures. More info (page 111)	Ctrl+P
Quit	Terminate HDevelop. See also: `exit`.	Ctrl+Q
Reload All	Reloads all changed files and resets the program. The PC is set back to the first executable line of the main procedure. All variables are reset.	

6.1.2 Menu `Edit`

This menu holds the functions to modify the current HDevelop procedure body as shown in the program window. You can also access the preferences of HDevelop from this menu.

Menu Entry	Description	Shortcut
Undo	Only *editing* activities can be undone, not file operations.	Ctrl+Z
Redo	Undo activities.	Ctrl+Y
Cut	Cuts selected text to clipboard. For procedure calls, the corresponding local procedures and the paths to external procedures are also copied.	Ctrl+X or Shift+Del
Copy	Copies selected text to clipboard. For procedure calls, the corresponding local procedures and the paths to external procedures are also copied.	Ctrl+C or Ctrl+Ins
Paste	Pastes clipboard content. Whether local procedures are also pasted depends on the setting of Copy local procedures in full-text editor (page 122). Before an external procedure path is added, you are asked whether or not you want to add that path to the external procedure paths.	Ctrl+V or Shift+Ins
Activate	Re-enable the highlighted program lines previously disabled with `Deactivate`. Comment lines created with the operator `comment` are unaffected.	F3
Deactivate	Disable the highlighted program lines. An asterisk at the beginning of the selected lines marks them as comments. Comment lines created with the operator `comment` are unaffected.	F4
Find/Replace...	Opens the Find/Replace dialog for full-text search, to search for variable names, or to search for operator or procedure calls. More info (page 72)	Ctrl+F
Find Again	Repeats the search specified via the menu item `Find/Replace...`.	Ctrl+G
Set/Clear Bookmark	Toggle bookmarking of selected program lines. Bookmarks are not saved with the program.	Ctrl+F11
Next Bookmark	Jumps to the next bookmarked program line, either in the current procedure or in the first following procedure containing bookmarks. Procedures are traversed in alphabetical order except for the main procedure always coming first.	F11
Previous Bookmark	Jumps to the last bookmarked program line, either in the current procedure or in the first preceding procedure containing bookmarks. Procedures are traversed backwards in alphabetical order except for the main procedure always coming first.	Shift+F11
Manage Bookmarks	Opens the Quick Navigation window to manage bookmarks and navigate to the bookmarked program lines. More info (page 155)	Ctrl+Shift+O,F11 or Ctrl+Shift+O,Ctrl+Shift+F11
Invalid Lines	Opens the Quick Navigation window to manage invalid program lines. More info (page 153)	Ctrl+Shift+O,F12 or Ctrl+Shift+O,Ctrl+Shift+F12
Preferences	Opens dialog to set global preferences. More info (page 112)	Ctrl+Shift+O,S or Ctrl+Shift+O,Ctrl+Shift+S

6.1.3 Menu Execute

This menu item provides 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.

Menu Entry	Description	Shortcut
▷ Run	Execute the program from the program counter (PC) ➪ to the next breakpoint, `stop` instruction, or the last program line. More info (page 37). Further, you can adjust the runtime behavior in the Runtime Settings (page 125).	F5
Run To Insert Cursor	Execute the program from the PC to the insert cursor (IC) ▶. Breakpoints or `stop` instructions between PC and IC stop the execution.	Shift+F5
Step Over	Execute the next operator or procedure in the current program.	F6
Step Forward	Execute the program line per line. In case of loops, only the first run is single-stepped.	Shift+F6
Step Into	Step into a procedure. Click `Step Into` with the PC on a procedure call line.	F7
Step Out	Step out of a procedure.	F8
☐ Stop	Stop the program execution. Processing continues until the current operator has finished its computations. Some operators support timeouts, in which case their intermediate results are discarded.	F9
☐ Stop After Procedure	Same as `Stop` but finishes the current procedure (if any) first.	Shift+F9
Attach To Process...	Debug an external HDevEngine application. More info (page 299)	
Stop Debugging	Stop debugging an external HDevEngine application.	
Thread View / Call Stack	Visualize started threads and the calling hierarchy. More info (page 160)	Ctrl+Shift+O,C or Ctrl+Shift+O,Ctrl+Shift+C
Set / Clear Breakpoint	Set or clear breakpoints at the selected program lines. You can also set breakpoints by holding the Ctrl key and clicking in the left column of the program window as described in section 6.17.3 on page 133.	F10
Activate / Deactivate Breakpoint	Toggles the state of the breakpoints in the selected program lines. This can help to switch between continuous run mode and debug mode. To deactivate/activate the breakpoints in many procedures at once, use the tab `Breakpoints` of the Quick Navigation Window (see section 6.18.3 on page 154).	Shift+F10
Clear All Breakpoints	Both line-based breakpoints as well as breakpoints on variables are cleared.	
Manage Breakpoints	Opens the Quick Navigation Window to manage breakpoints. More info (page 154)	Ctrl+Shift+O,F10 or Ctrl+Shift+O,Ctrl+Shift+F10
Reset Program Execution	The PC is set to the first executable line of the main procedure. All variables are reset.	F2
Reset Procedure Execution	Useful for debugging procedures without affecting the calling procedures. The PC is set back to the first executable line of the current procedure. All variables of the current procedure are reset.	Shift+F2
Abort Procedure Execution	The PC is set back to the first executable line of the current procedure. All variables of the current procedure are reset.	Shift+F8
Activate / Deactivate Profiler	Toggle the profiler section of the program window, which displays runtime statistics, like the total and average processing time of each program line. More info (page 148)	Ctrl+Shift+F,F or Ctrl+Shift+F,Ctrl+Shift+F
Profiler Display	Select what is being displayed in the profiler section of the program window. More info (page 149)	

Menu Entry	Description	Shortcut
Reset Profiler	Reset profiler data.	Ctrl+Shift+F,R or Ctrl+Shift+F,Ctrl+Shift+R
Show Runtime Statistics	Evaluate runtime statistics.	Ctrl+Shift+F,S or Ctrl+Shift+F,Ctrl+Shift+S

6.1.4 Menu `Visualization`

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

Menu Entry	Description	Shortcut
Open Graphics Window...	Specify the position, size, and background color of the new graphics window. More info (page 75). See also `dev_open_window`.	Ctrl+Shift+G,O or Ctrl+Shift+G,Ctrl+Shift+O
Clear Graphics Window	Clear active graphics window. The history (previously displayed objects) of the window is also cleared. See also `dev_clear_window`.	Ctrl+Shift+G,Del or Ctrl+Shift+G,Ctrl+Shift+Del
Close Graphics Window	Close active graphics window. See also `dev_close_window`.	Ctrl+Shift+G,Q or Ctrl+Shift+G,Ctrl+Shift+Q
Display	Select iconic variable to be displayed in active graphics window. This submenu lists all instantiated iconic variables (images, regions, XLDs) for quick selection. See also `dev_display`.	
Window Size	Set window size of active graphics window. `Double` and `Half` change the size to half and double its current window size, respectively. `Aspect Ratio 1:1` scales down the current window size, so that the aspect ratio of the displayed image is maintained. See also `dev_set_window_extents`.	
Image Size	This submenu offers a list of zoom options to scale the image and to control its resizing behavior. More info (page 75)	
Colored	Color regions and XLDs as an easy way to display multiple regions or XLDs. Each region is displayed in a different color, where the number of different colors (3, 6, or 12) is specified in the submenu; default: 12 colors. If all regions are still displayed with one color, use the operator `connection` beforehand. You can check this also with the operator `count_obj`. See also `dev_set_colored`.	
Color	Choose a color for displaying segmentation results (regions and XLDs), text created with `write_string`, and general line drawings (for example, 3D plots, contour lines, and bar charts); default color: red. The number of available colors depends on the graphics display (that is, 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 `Apply Changes Immediately` is selected. See also `dev_set_color`.	
Draw	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`. See also `dev_set_draw`.	
Line Width	Determine the line width for painting XLDs, borders of regions, or other types of lines. See also `dev_set_line_width`.	
Shape	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. See also `dev_set_shape`.	

Menu Entry	Description	Shortcut
Lut	Specify a look-up table 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. See also dev_set_lut.	
Paint	Define the mode to display gray value images. See the menu item Set Parameters below. See also dev_set_paint.	
Attach To Canvas / Detach From Canvas	Add the selected graphics window to or remove it from the canvas.	
Apply Changes Immediately	If 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.	
⚙ Set Parameters	Opens the Visualization Parameters of the active graphics window. More info (page 76)	Ctrl+Shift+G,P or Ctrl+Shift+G,Ctrl+Shift+P
Reset Parameters	Reset the visualization parameters of the active graphics window to the default settings (as defined in the preferences; see section 6.16 on page 112). 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.	
Record Interactions	If enabled, any subsequent changes to the visualization settings will insert the corresponding operator call(s) into the current program. Unlike Insert Code..., which dumps the current settings in one block, Record Interactions allows you to 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.	Ctrl+I
Insert Code...	Insert code into the current program that can be used to restore the current settings of the graphics window. More info (page 40)	Ctrl+Shift+G,I or Ctrl+Shift+G,Ctrl+Shift+I
Update Window	Specify the update behavior of the active graphics window. Check In Run Mode to update images, regions, or XLDs during program execution. The submenu In Single Step Mode specifies the update behavior when stepping through the program. 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	
Position Precision	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. The upper left *edge* of the image is displayed as -0.5, -0.5.	
Pixel Grid	To easier visualize the pixels when an image is zoomed, a Pixel Grid can be set.	

Menu Entry	Description	Shortcut
Save Window...	The graphics window is saved "as is" (including displayed regions and XLDs). In the file dialog, you can select between the output formats TIFF, BMP, JPEG, PNG, or PostScript. See also `dump_window`.	Ctrl+Shift+G,S or Ctrl+Shift+G,Ctrl+Shift+S

Submenu `Tools`

Menu Entry	Description	Shortcut
Zoom Window	Open zoom window for image details and pixel inspection. More info (page 177)	Ctrl+Shift+O,Z or Ctrl+Shift+O,Ctrl+Shift+Z
New Zoom Window	Open additional zoom window. More info (page 177)	
Gray Histogram	Display gray value histogram of active graphics window, can also be used to select thresholds interactively and to set the range of displayed gray values dynamically. More info (page 99)	Ctrl+Shift+O,H or Ctrl+Shift+O,Ctrl+Shift+H
Feature Histogram	Interactively inspect feature histograms. More info (page 104)	Ctrl+Shift+O,F or Ctrl+Shift+O,Ctrl+Shift+F
Feature Inspection	Inspect shape and gray value features of individual regions. More info (page 105)	Ctrl+Shift+O,I or Ctrl+Shift+O,Ctrl+Shift+I
Line Profile	Display line profile of active graphics window. More info (page 107)	Ctrl+Shift+O,L or Ctrl+Shift+O,Ctrl+Shift+L
OCR Training File Browser	Browse OCR training files. More info (page 86)	Ctrl+Shift+O,T or Ctrl+Shift+O,Ctrl+Shift+T

6.1.5 Menu Procedures

This menu contains all functionality that is needed to create, modify, copy, or delete HDevelop procedures. To save procedures, refer to the File menu (section 6.1.1 on page 52).

Menu Entry	Description	Shortcut
Create New Procedure	Opens the procedure interface window to create a new internal or external procedure. More info (page 135)	Ctrl+Shift+P,C or Ctrl+Shift+P,Ctrl+Shift+C
Duplicate...	Copy a procedure under a different name. Choose the procedure to be copied via Source. Enter the name of the copied procedure into Target. 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. Duplicating procedures that are protected with a password is also possible; see section 5.6 on page 47. The associated password is also used for the duplicated procedure.	
Edit Procedure Interface	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 127). The interface of protected procedures can only be edited after the corresponding password has been entered; see section 5.6 on page 47.	Ctrl+Shift+P,I or Ctrl+Shift+P,Ctrl+Shift+I
Delete Current	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.	Ctrl+Shift+P,Del or Ctrl+Shift+P,Ctrl+Shift+Del
Delete All Unused Local	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.	
Insert Used As Local	The external procedures used in the current program are copied as local procedures. The external procedure files are left untouched.	
Insert All As Local	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. If your program contains protected external procedures, HDevelop issues a warning and inserts only the procedures that are not locked. For changing the status of a procedure, see section 5.6 on page 47.	
Make All External	Convert all local procedures into external procedures. Former local procedures are stored as external procedures in a selectable directory of the list of external procedure directories. More info (page 115)	
Manage Procedures	Opens the dialog Preferences ▷ Procedures to configure procedure settings. More info (page 117)	
Edit Procedure	Select a procedure for editing in the program window. This submenu lists all procedures in submenus, grouped by chapter and section title. More info (page 142)	

6.1.6 Menu Operators

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

For detailed information about all operators and procedures, we recommend reading the corresponding sections of the reference manual. To get there quickly, select an operator or procedure from the menu, and then press F1.

Menu Entry	Description
Control	Select control structures, meaning, operators that control the program flow like `if`, `else`, `for`, `while`.
Develop	These operators allow you to interact with the user interface. The operators start with the prefix `dev_` to distinguish them from the underlying basic HALCON operators (for example, `dev_set_color` and `set_color`).
1D Measuring, 2D Metrology, ...	Select HALCON operators and procedures sorted by categories.
Legacy	List obsolete HALCON operators, provided for backward compatibility.
Unclassified	List all HALCON operators and procedures not listed in any of the other categories, like 1D Measuring, 2D Metrology, etc.

6.1.7 Menu Suggestions

This menu shows another possibility how to select HALCON operators via suggestions. Assuming that you have already selected an operator in a previous step, depending on this operator, the following suggestions are offered.

Suggestions are separated into groups as described below.

Menu Entry	Description	Shortcut
Alternatives	Suggests alternative operators. For example, replace mean_image with operators such as binomial_filter, gauss_filter, or smooth_image.	
See also	Shows operators that have some *connection* to the current operator. For example, the operator gen_lowpass, which is used to create a lowpass filter in the frequency domain, is a reasonable reference to a Gaussian filter.	
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).	
Successors	Shows possible successor operators. Image processing tasks often result in a "natural" sequence of operators. For example, you use a thresholding after executing an edge filter or you execute a region processing (for example, morphological operators) after a segmentation.	
Keywords	Opens the tab card Index of the help window. More info (page 83)	Ctrl+Shift+H,K or Ctrl+Shift+H,Ctrl+Shift+K

6.1.8 Menu `Assistants`

This menu assembles assistants for specific machine vision tasks. The general concept of the assistants is described in the chapter Assistants (page 179).

The following assistants are available:

Menu Entry	Description
Open New Image Acquisition	Acquire images from file or from image acquisition devices. More info (page 180)
Open New Calibration	Calibrate your camera setup using calibration images. More info (page 184)
Open New Matching	Locate objects. More info (page 202)
Open New Measure	Measure 1D objects. More info (page 219)
Open New OCR	Classic optical character recognition using classifiers. More info (page 231)

6.1.9 Menu Window

This menu offers support to manage the sub-windows of the main window, such as the program, operator, variable, graphics window(s), and possible other dialogs.

At the bottom of the menu, all open windows are listed. Click an entry to bring the corresponding window to the front.

Menu Entry	Description	Shortcut
Open Graphics Window Docked	The graphics window will be opened docked. See graphics window (page 73), and docked windows (page 17) for more information. See also dev_open_window.	Ctrl+Shift+O,G or Ctrl+Shift+O,Ctrl+Shift+G
Open Graphics Window in Canvas	The graphics window will be opened within the canvas. See graphics window (page 73) for more information. See also dev_open_window.	Ctrl+Shift+O,B or Ctrl+Shift+O,Ctrl+Shift+B
Open Program Window	See program window (page 127) for more information. See also dev_open_tool.	Ctrl+Shift+O,P or Ctrl+Shift+O,Ctrl+Shift+P
Open Variable Window	This menu item is grayed out if the variable window is already open. See variable window (page 160) for more information. See also dev_open_tool.	Ctrl+Shift+O,V or Ctrl+Shift+O,Ctrl+Shift+V
Open Operator Window	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. See operatorWindow (page 91) for more information.	Ctrl+Shift+O,O or Ctrl+Shift+O,Ctrl+Shift+O
Open Output Console	The output console window contains a log of the most recent messages. More info (page 95)	Ctrl+Shift+O,E or Ctrl+Shift+O,Ctrl+Shift+E
Open Quick Navigation	Manage program lines on certain criteria. More info (page 152)	Ctrl+Shift+O,R or Ctrl+Shift+O,Ctrl+Shift+R
Open Canvas Window	The canvas window is a container for graphics windows. It allows you to organize multiple graphics windows side-by-side. More info (page 66)	Ctrl+Shift+O,A or Ctrl+Shift+O,Ctrl+Shift+A
Canvas Options	Configure the canvas window. More info (page 66)	
Cascade Floating Windows	Arrange 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 122).	Ctrl+Shift+W,C or Ctrl+Shift+W,Ctrl+Shift+C
Bring Main Window To Front	Raises the main window to the top. Works only if your windows are minimizable.	Ctrl+Shift+W,W or Ctrl+Shift+W,Ctrl+Shift+W
Bring Others To Front	Raises all floating windows to the top. Works only if your windows are minimizable.	Ctrl+Shift+W,B or Ctrl+Shift+W,Ctrl+Shift+B
Full Screen	Toggle full screen mode.	Ctrl+Shift+W,F or Ctrl+Shift+W,Ctrl+Shift+F
Restore Default Layout Use	The canvas and the docked graphics window are docked top left.	

6.1.10 Menu Help

This menu offers a version check and access to the HALCON documentation.

Menu Entry	Description	Shortcut
Help	Open the Help Window. More info (page 81)	F1
Context Help	Based on the currently focused window or tab card, the corresponding help page is opened.	Shift+F1
Start Dialog	Open the Start dialog, which helps you getting started with HALCON. More info (page 16)	Ctrl+Shift+H,S or Ctrl+Shift+H,Ctrl+Shift+S
Telemetry Dialog	Open the Telemetry dialog.	
HALCON Reference	Open the reference documentation of HALCON operators and classes, available in HDevelop, .NET, Python, C++, and C syntax.	Ctrl+Shift+H,R or Ctrl+Shift+H,Ctrl+Shift+R
HDevelop User's Guide	Open a guide to HDevelop, the interactive programming environment for the HALCON machine vision library.	Ctrl+Shift+H,U or Ctrl+Shift+H,Ctrl+Shift+U
HDevelop Language	Open the introduction to the syntax and semantics of the HDevelop language in the HDevelop User's Guide.	Ctrl+Shift+H,L or Ctrl+Shift+H,Ctrl+Shift+L
Search Documentation	Search for keywords in the documentation. More info (page 81)	Ctrl+Shift+H,F or Ctrl+Shift+H,Ctrl+Shift+F
HALCON News (WWW)	Check the latest news about HALCON on the MVTec website.	Ctrl+Shift+H,W or Ctrl+Shift+H,Ctrl+Shift+W
Check For Updates	Open a version check in your web browser.	Ctrl+Shift+H, C
About	Shows version and license information, like the HALCON edition, license type, license expiration date, and the like.	Ctrl+Shift+H,A or Ctrl+Shift+H,Ctrl+Shift+A

6.2 Tool Bar

	Action	Shortcut	Info
	New Program	Ctrl+N	section 6.1.1 on page 53
	Open Program...	Ctrl+O	section 6.1.1 on page 53
	Browse HDevelop Example Programs...	Ctrl+E	section 6.1.1 on page 53
	Save	Ctrl+S	section 6.1.1 on page 53
	Save All	Ctrl+Alt+S	section 6.1.1 on page 53
	Image Acquisition Assistant		section 7.1 on page 180
	Undo	Ctrl+Z	section 6.1.2 on page 54
	Redo	Ctrl+Y	section 6.1.2 on page 54
	Activate	F3	section 6.1.2 on page 54
	Deactivate	F4	section 6.1.2 on page 54
	Find/Replace...	Ctrl+F	section 6.7 on page 72
	Run	F5	section 6.1.3 on page 56
	Step Over	F6	section 6.1.3 on page 56
	Step Into	F7	section 6.1.3 on page 56
	Step Out	F8	section 6.1.3 on page 56
	Stop	F9	section 6.1.3 on page 56
	Reset Program Execution	F2	section 6.1.3 on page 56
	Reset Procedure Execution	Shift+F2	section 6.1.3 on page 56
	Thread View / Call Stack	Ctrl+Shift+O,C or Ctrl+Shift+O,Ctrl+Shift+C	section 6.20 on page 160
	Activate Profiler	Ctrl+Shift+F,F or Ctrl+Shift+F,Ctrl+Shift+F	section 6.1.3 on page 56
	Zoom Window	Ctrl+Shift+O,Z or Ctrl+Shift+O,Ctrl+Shift+Z	section 6.1.4 on page 58
	Gray Histogram	Ctrl+Shift+O,H or Ctrl+Shift+O,Ctrl+Shift+H	section 6.1.4 on page 58
	Feature Histogram	Ctrl+Shift+O,F or Ctrl+Shift+O,Ctrl+Shift+F	section 6.1.4 on page 58
	Feature Inspection	Ctrl+Shift+O,I or Ctrl+Shift+O,Ctrl+Shift+I	section 6.1.4 on page 58
	Line Profile	Ctrl+Shift+O,L or Ctrl+Shift+O,Ctrl+Shift+L	section 6.1.4 on page 58
	OCR Training File Browser	Ctrl+Shift+O,T or Ctrl+Shift+O,Ctrl+Shift+T	section 6.1.4 on page 58

6.3 Status Bar

The status bar is divided into the following areas:

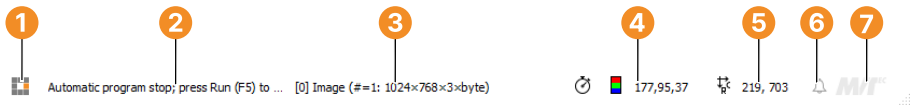


Figure 6.2: The status bar.

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 section 6.16.12 on page 123), 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 6.1.9 on page 63.

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 the logo to open our website `https://www.halcon.com`.

6.4 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 opened, all new graphics windows opened with `dev_open_window` will automatically show up on the canvas. You can add any additional graphics window to the canvas manually by clicking its context menu entry `Attach To Canvas`. To remove a graphics window from the canvas, open its context menu and choose `Detach From Canvas`.

The canvas displays graphics window states as follows:

Active: A yellow stroke highlights the active graphics window, see figure 6.3.

Selected: An orange colored frame indicates the selected graphics window figure 6.3.

i **The canvas window cannot be tabbed with other windows.**

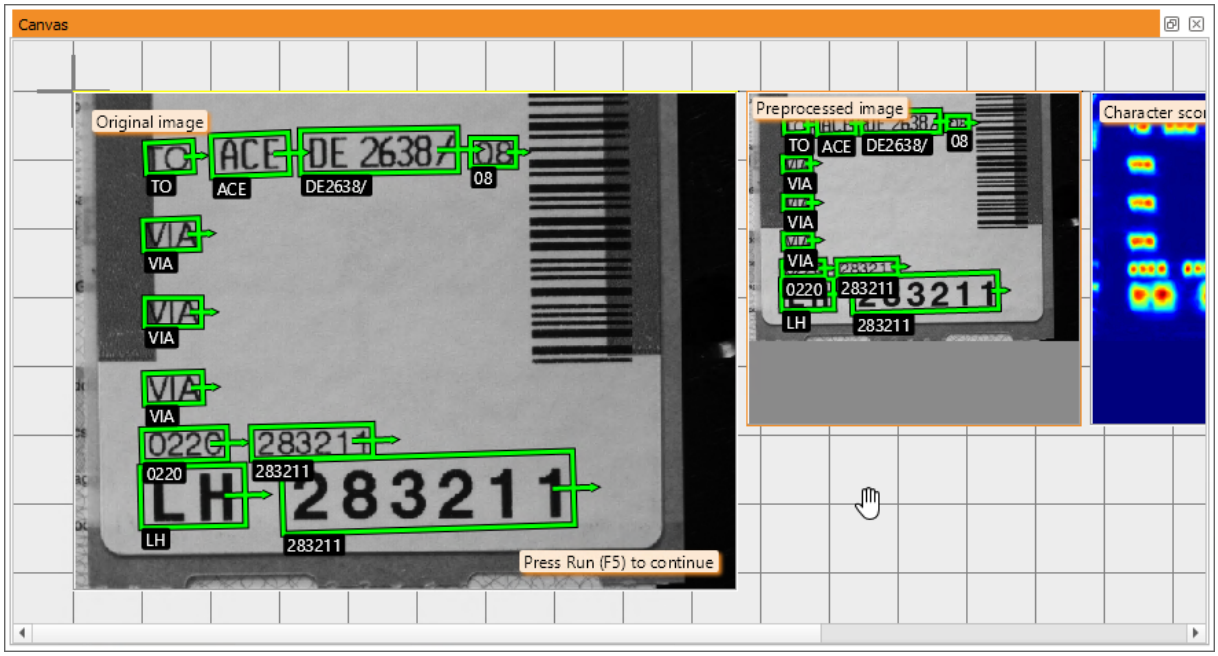


Figure 6.3: The canvas window.

6.4.1 `Canvas Options`

To set the canvas options click the canvas tab within the preferences dialog figure 6.4, right-click the canvas; or right-click any graphics window on the canvas and choose `Canvas Options`. Alternatively, choose `Canvas Options` from the menu `Window`:

- `Fit All Graphics Windows In View`

  The view is zoomed in or out once, so that all graphics windows on the canvas are visible at a glance with optimal use of the available space.

  i **To fit the view not only once, but continuously**, enable both canvas options `Auto-fit By Zooming Out` and `Auto-fit By Zooming In`.

  i **The current zoom factor** is displayed in the bottom right corner of the canvas window. Click the zoom factor to reset the view to 100%.

- `Auto-fit By Zooming Out`

  The view is zoomed out automatically, so that all graphics windows on the canvas are always visible at a glance without scrolling. This means, as soon as you try to move a graphics window across the borders of the current view, the view adapts automatically by decreasing the zoom factor.

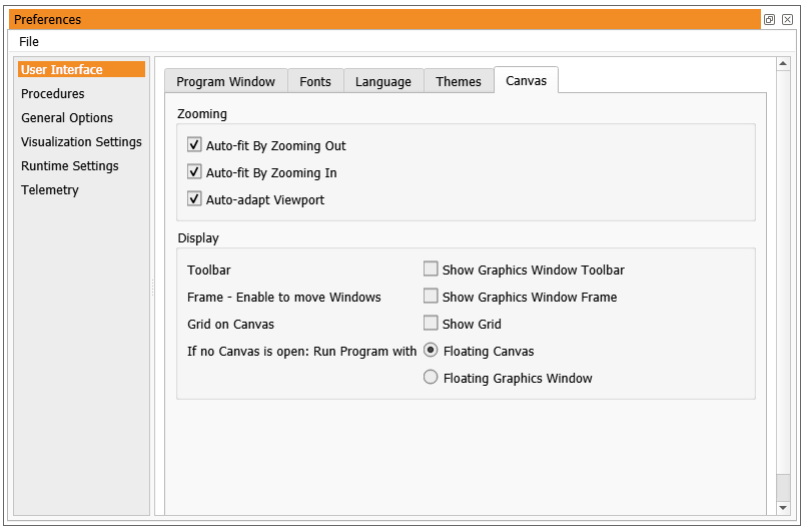


Figure 6.4: Canvas Options.

- `Auto-fit By Zooming In`

  The view is zoomed in automatically, so that all graphics windows on the canvas are visible at a glance with optimal use of the available space. This means, as soon as you move a graphics window towards the center of the current view, the view adapts automatically by increasing the zoom factor to maximum 100%.

- `Auto-adapt Viewport`

  The canvas view is focused automatically on the selected graphics window. Thus, the view follows the selected graphics window. To scroll the canvas view manually, disable this option.

- `Show Grid`

  The grid helps you arranging and resizing your graphics windows. The cross in the grid visualizes the reference point (0, 0) for the graphics window position.

- `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.

- `Run Program with Floating Canvas`

  Enabling the `Floating Canvas` will automatically open a floating canvas window, as soon as a code line in the script opens a graphics window, when `dev_open_window` is executed. All graphics windows will then be displayed in the canvas window. This option is available within the preferences dialog within the tabcard `Canvas` in the `User Interface` settings, see figure 6.4 on page 67. It is recommended to use this to visualize examples, and the results of your program. Keep in mind that if a canvas window is already open, all graphics windows will be displayed there. When a canvas was opened by a script, it will be closed when a reset is executed by pressing F2. Should you want to open a canvas with a distinct size, dev_open_tool ('canvas',...) should be used.

- `Run Program with Floating Graphics Window`

  Enabling the `Floating Graphics Window` will open all graphics windows separately. Keep in mind that if a canvas window is already open, all graphics windows will be displayed there. This option is available within the preferences dialog within the tabcard `Canvas` in the `User Interface` settings, see figure 6.4 on page 67.

- `Canvas reset behavior`

  When a canvas was opened via script it will be closed, when the F2 key is pressed. To open the canvas in a distinct size dev_open_tool ('canvas') should be used, and also the size of an already opened canvas can be adjusted by this.

6.4.2 Zooming

To zoom the content of the canvas window:

1. Click inside the canvas window (either in the background area or inside a graphics window on the canvas).
   → The canvas window gets the focus, indicated by its frame turning from gray to orange.
2. Press Ctrl+Shift++ to zoom in, or press Ctrl+Shift+- to zoom out. Alternatively, hold Ctrl or Ctrl+Shift and scroll up to zoom in, or scroll down to zoom out. **You can change the zoom direction of the mouse wheel** via the general option Action when spinning the mouse wheel down (page 121). Note that this setting also applies to the graphics window.
   → The current zoom factor is displayed in the bottom right corner of the canvas window.
   Click the zoom factor to reset the view to 100%.

6.4.3 Moving

To scroll the canvas:

1. Disable the canvas option (page 66) `Auto-adapt Viewport`.
2. Drag (page 12) the canvas. Use the middle mouse button for dragging if your mouse is located over a graphics window on the canvas.

To move a graphics window on the canvas:

1. Enable the canvas option `Show Graphics Window Frame`.
2. Drag the frame of the graphics window.

6.4.4 Resizing

To adapt the size of the graphics window on the canvas:

1. Enable the canvas option (page 66) `Show Graphics Window Frame`.
2. Drag (page 12) the bottom right corner of the graphics window.

   Resizing is not possible during draw operations, like `draw_rectangle1`.

6.5 Browse Example Programs Dialog

Using this dialog, you can browse and load the example programs delivered with HALCON. To open the example program browser, click `File` ▷ `Browse HDevelop Example Programs...` (see figure 6.5).

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.

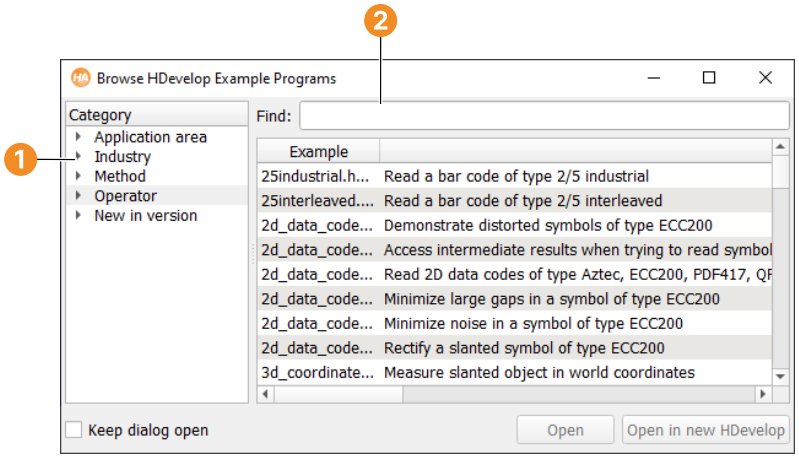


Figure 6.5: HDevelop example programs.

1 Category name

2 `Find` text field

Browse the categories: Click a topic 1 to select it and display its example programs. You can select multiple topics at once by holding the Ctrl key while clicking the categories.

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

Assume you are looking for a measuring example from the semiconductor industry:

1. Double-click `Industry`.
2. Click the subtopic `Semiconductors`. The examples belonging to the semiconductor industry are listed on the right.
3. 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.
4. Select `measure_ic_leads.hdev` by clicking it.
5. Click `Open`. The selected example program is then loaded. Alternatively, you can load an example program by double-clicking it. The example browser is closed unless `Keep dialog open` is selected.

6.6 Export Program Dialog

See also: `hdevelop -convert` (command line switch (page 323)).

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 6.6.

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

Format	Syntax	Extension	More information
Text File	HALCON/HDevelop	`.txt`	Plain text export
C	HALCON/C	`.c`	section 10.2.4 on page 313
C++	HALCON/C++	`.cpp`	section 10.2.1 on page 307
Visual Basic .NET	HALCON/.NET	`.vb`	section 10.2.3 on page 311
C#	HALCON/.NET	`.cs`	section 10.2.2 on page 309

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.

- `Program`: The entire program is exported the 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 314).

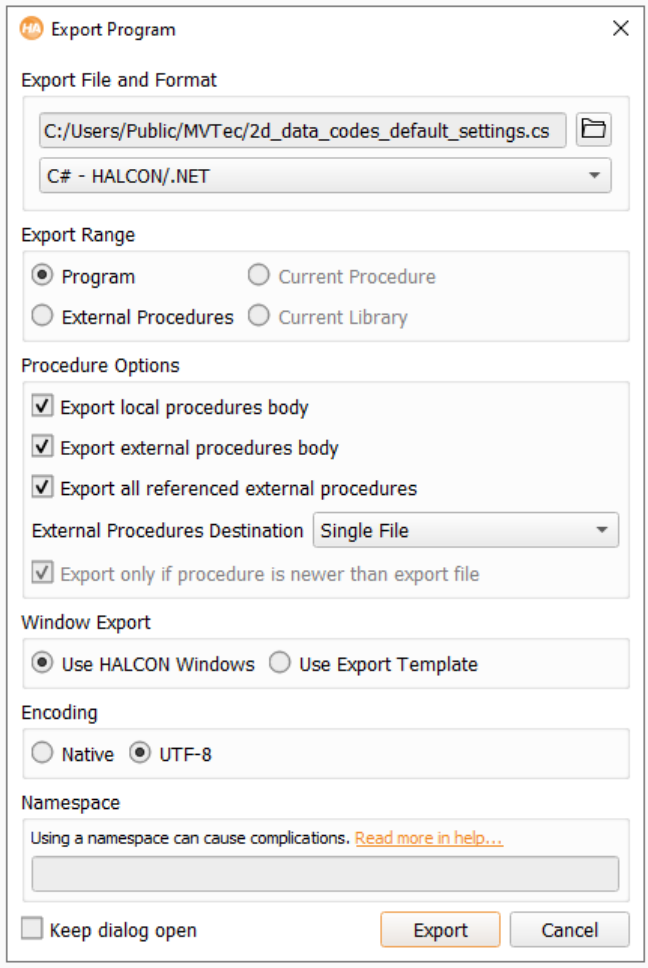


Figure 6.6: `Export`.

`Procedure Options`

Defines the export behavior for procedures.

- `Export local procedures body`: If checked, both the declaration and the body of local procedures are 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 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.

- `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.

`Namespace`

Can be used to avoid name conflicts between exported code and existing names in the client application, or to avoid name conflicts between multiple exported libraries in the same client application.

- If this field is empty no namespace is used.
- If this field is filled all code, generated during this one export operation, will be wrapped using exactly this one name.
- Template projects expect exported code without namespace, and a template export with namespace only works with adapted projects.
- When performing multiple exports with different namespaces the exports for each namespace must be self-contained including all dependencies. Exported procedure calls will not "see" exported procedures from the other namespace. The solution here would be to export the procedure call, with all dependencies even if it can lead to code duplication.
- The separate exports cannot share global variables. To avoid this: Do not use global variables, or use multiple "global def" declarations, and do not rely on sharing of global variables between libraries.

⚠ This icon appears in some cases, when an export may not be working. Hovering over the icon provides more information.

`Keep dialog open`

Checking this box keeps the dialog open for subsequent exports.

6.7 Find/Replace Dialog

The Find/Replace 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 6.7.

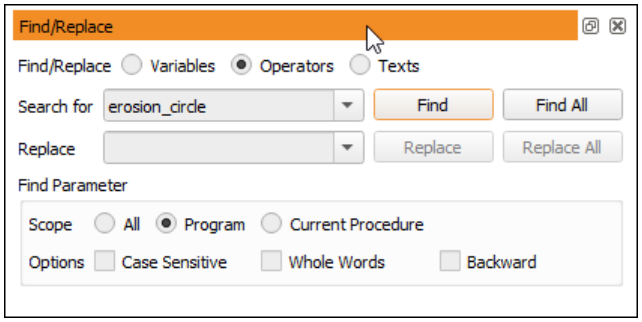


Figure 6.7: Looking for operator calls containing “grab_image”.

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

Context	Description
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:

Scope	Description
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 5.6 on page 47.

The following options specify how the search is performed:

Option	Description
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.

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 shown in the tab card `Find Results` of the Quick Navigation window.

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.

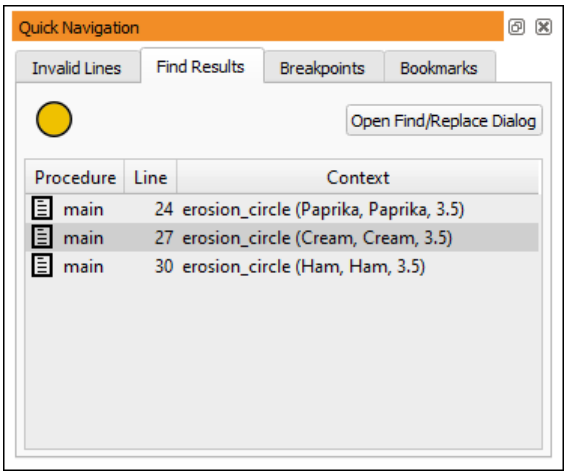


Figure 6.8: Finding all occurrences of the search text.

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 (from the context menu of the search result): `Cut` (page 54), `Copy` (page 54), `Delete` (page 54), `Activate` (page 54), `Deactivate` (page 54), and `Bookmark Selection`. Use `Bookmark Selection` to add bookmarks to the selected program lines. This 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.

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 6.9 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.

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

6.8 Graphics Window

This window displays iconic data: images, regions, and XLDs. You can open several graphics windows. The active graphics window is shown by in the window's tool bar.

Figure 6.10 (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.

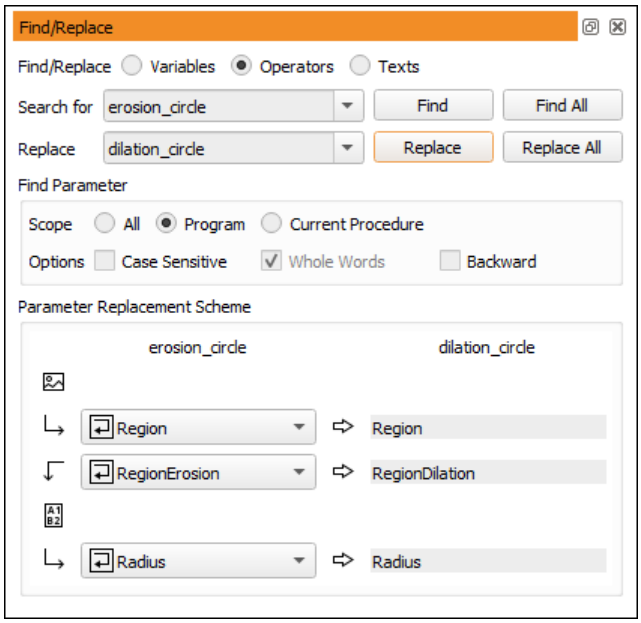


Figure 6.9: Replacing operator calls.

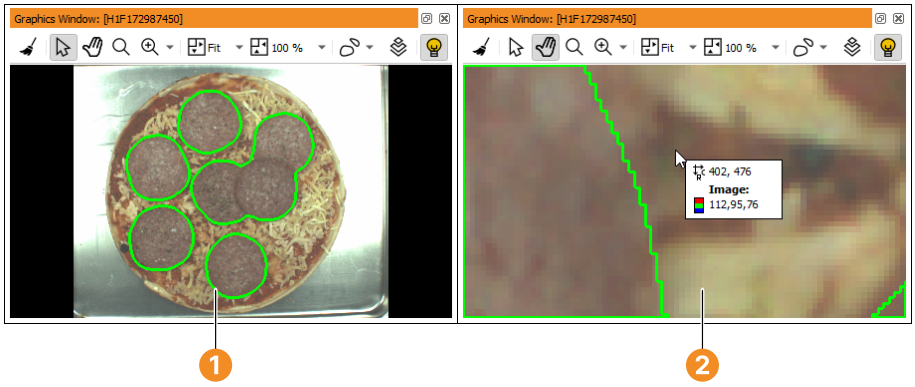


Figure 6.10: Graphics window displaying a tasty pizza.

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 section 6.3 on page 65). Sometimes, it is desirable to display this information close to the mouse cursor. This can be achieved by holding down the Ctrl key.

Please note this does not work when the zoom in and out tool is selected since pressing Ctrl inverts the corresponding zoom action.

Figure 6.10 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 Zoom settings (page 78)) or the operator `dev_set_part`. 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 Visualization Parameters (page 76)) applies to the currently active graphics window *only*. The parameter settings

of all other open graphics windows remain unchanged.

Each window has a history that contains *all objects* and *all 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 an icon are part of the history.

Additional Information

When opening a graphics window via `Visualization` ▷ `Open Graphics Window...` and 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).

6.8.1 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 6.10 on page 74, 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. The value can be selected from the menu attached to the small arrow. Additionally, the resizing behavior can be selected.

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 can also change the size of the graphics window, for example, 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. 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 can 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.

 Set window size. Clicking this icon sets the window size to the shown value. The value can be selected from the menu attached to the small arrow. `Double` and `Half` change the size to half and double its current window size, respectively. `Aspect Ratio 1:1` scales down the current window size, so that the aspect ratio of the displayed image is maintained.

 Draw ROIs and XLDs interactively; see section 6.19 on page 155.

 Toggle 3D plot mode; see below (page 80).

 Active graphics window.

 Non-active graphics window. Click the icon to activate the corresponding graphics window. Only one graphics window can be active at a time.

If you want to specify display parameters for a window, you can select the menu item `Visualization` in the menu. Here you can set the appropriate parameters by clicking the desired item (see section 6.1.4 on page 56). 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.

6.8.2 Visualization Parameters

Click `Set Parameters` (via the context menu or the menu `Visualization`) to open the `Visualization Parameters`. This dialog 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 dialog. 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.

`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 an icon or execution of an operator).

`Reset` Reset to the visualization settings defined in the `Preferences` (page 112).

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

6.8.2.1 Pen settings

The pen settings specify the drawing mode, filled or outlined, the shape, the line width, and the colors of regions and XLDs.

For regions the draw mode can be 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 6.13 on page 79). 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 6.16.13 on page 125).

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 section 6.4 on page 56
- "`Colored`", see also section 6.4 on page 56
- "`Color`", see also section 6.4 on page 56
- "`Shape`", see also section 6.1.4 on page 58
- "`Line Width`", see also section 6.1.4 on page 58

6.8.2.2 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`" in section 6.1.4 on page 58.

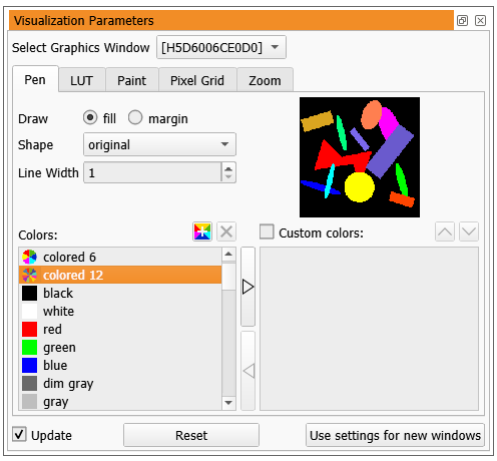


Figure 6.11: `Visualization Parameters` with multiple graphics windows.

6.8.2.3 `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 6.16 on page 81 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 section 6.8 on page 73.

6.8.2.4 `Pixel Grid` settings

This tab card lets you set a `Pixel Grid` to easier visualize the pixels, when zooming in. You cannot configure the `Pixel Grid` within HDevelop using the operator `set_window_param`.

6.8.2.5 `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.

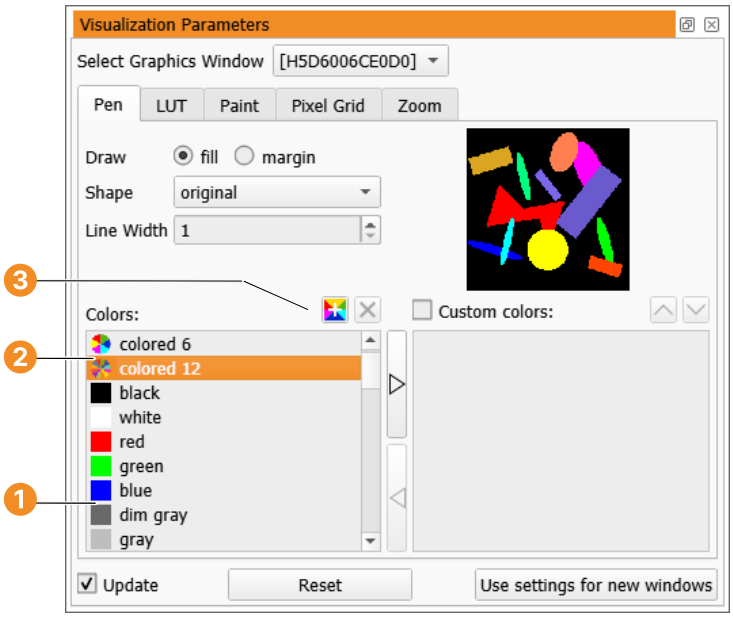


Figure 6.12: `Pen` settings.

1. Predefined colors
2. Color sets
2. Add/remove user-defined color

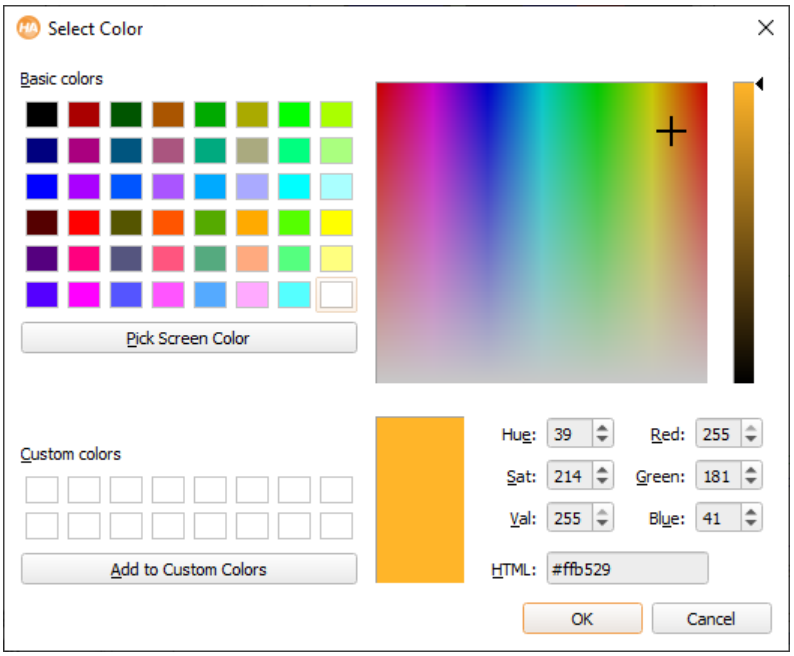


Figure 6.13: Defining a new color.

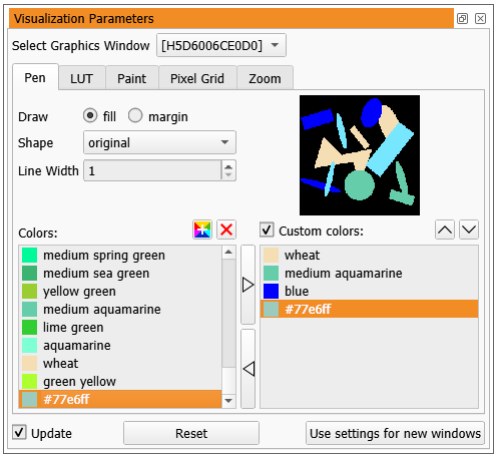


Figure 6.14: Custom color set.

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, click the button `Reset`.

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

6.8.3 Context Menu

The context menu can be enabled/disabled from the preferences (page 125).

The entries of the context menu are a subset of the menu `Visualization` (page 56). **Graphics windows attached to the canvas contain the additional context menu entry `Canvas Options`.** For more information, see section 6.4.1 on page 66.

6.8.4 3D Plot Mode

Clicking activates an interactive 3D plot mode. It displays meaningful information for height field images, such as images that encode height information as gray values. The greater the gray value, the higher the corresponding image point. Figure 6.19 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.

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 can 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.

`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 6.8.2.3 on page 78 for the other paint modes that can be selected in this window.

6.8.5 Special Keyboard Shortcuts

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

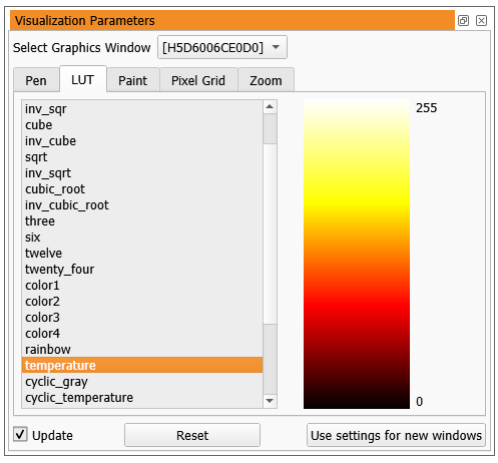


Figure 6.15: `Visualization Parameters`: LUT settings.

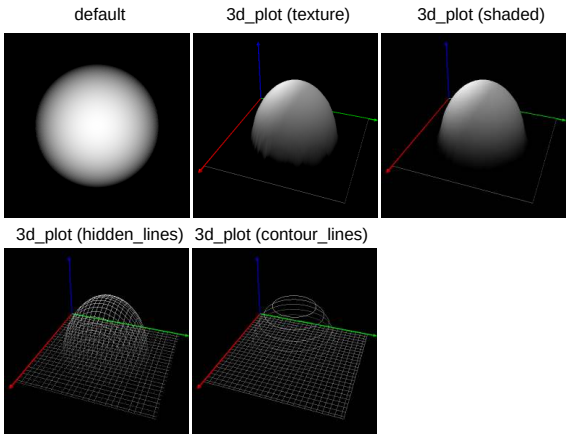


Figure 6.16: Comparison of the different paint settings.

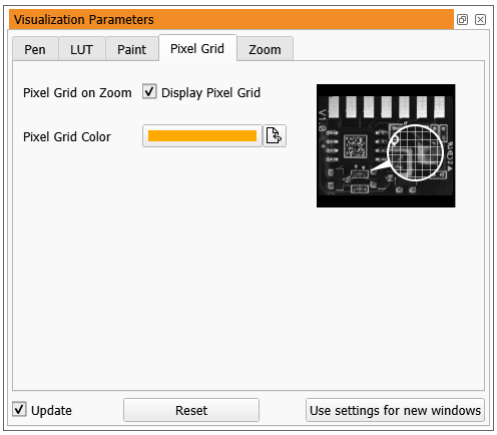


Figure 6.17: `Pixel Grid` settings.

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.2 on page 327.

6.9 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 Up and Down keys in the navigational panels to select the previous and the next entry, respectively. This allows you to examine, for example, 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 a node of the tree to display the associated document. See figure 6.21 on page 84 for an example.

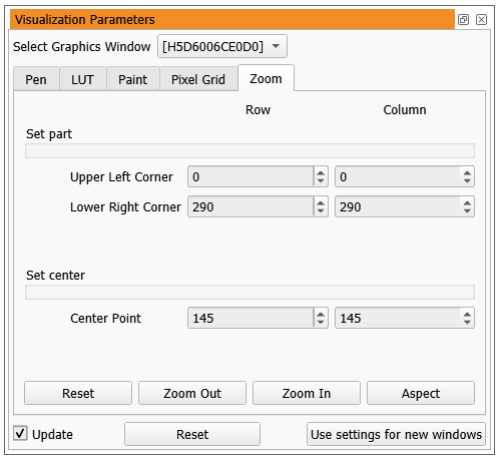


Figure 6.18: `Visualization Parameters`: Zoom settings.

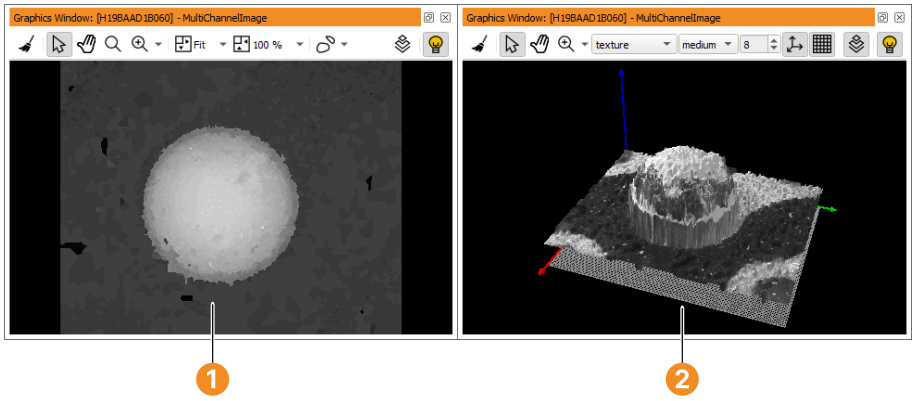


Figure 6.19: Default vs. 3D plot mode.

1 Default image display

2 3D plot mode

`Operators`

This tab card lists all operators in alphabetical order. Click 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 matches the query.

The query can 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:

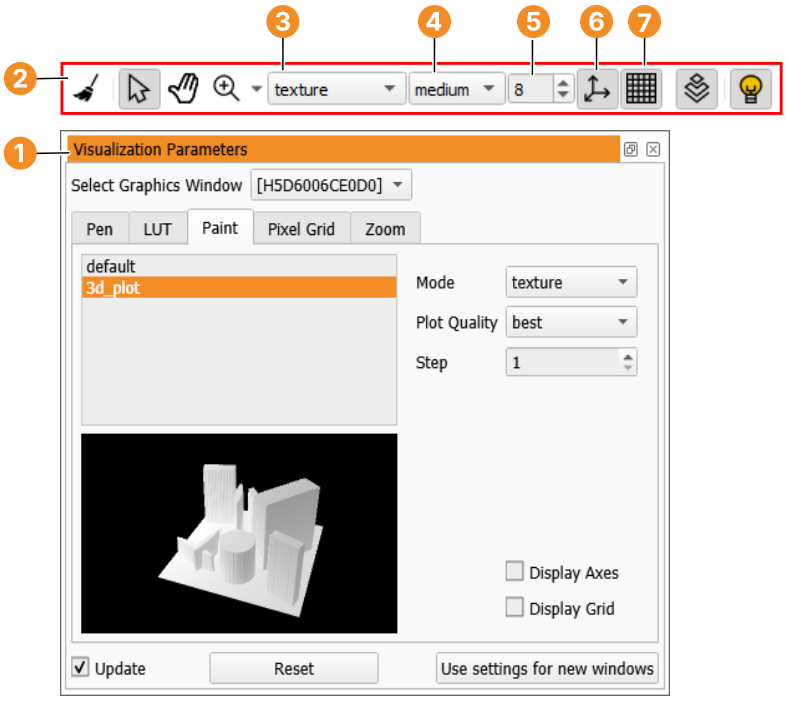


Figure 6.20: 3D plot mode settings.

1 Visualization parameters

2 Tool bar

3 Mode

4 Quality

5 Step

6 Axes

7 Grid

```
+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. The list of index entries can be filtered by entering a string 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.

`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

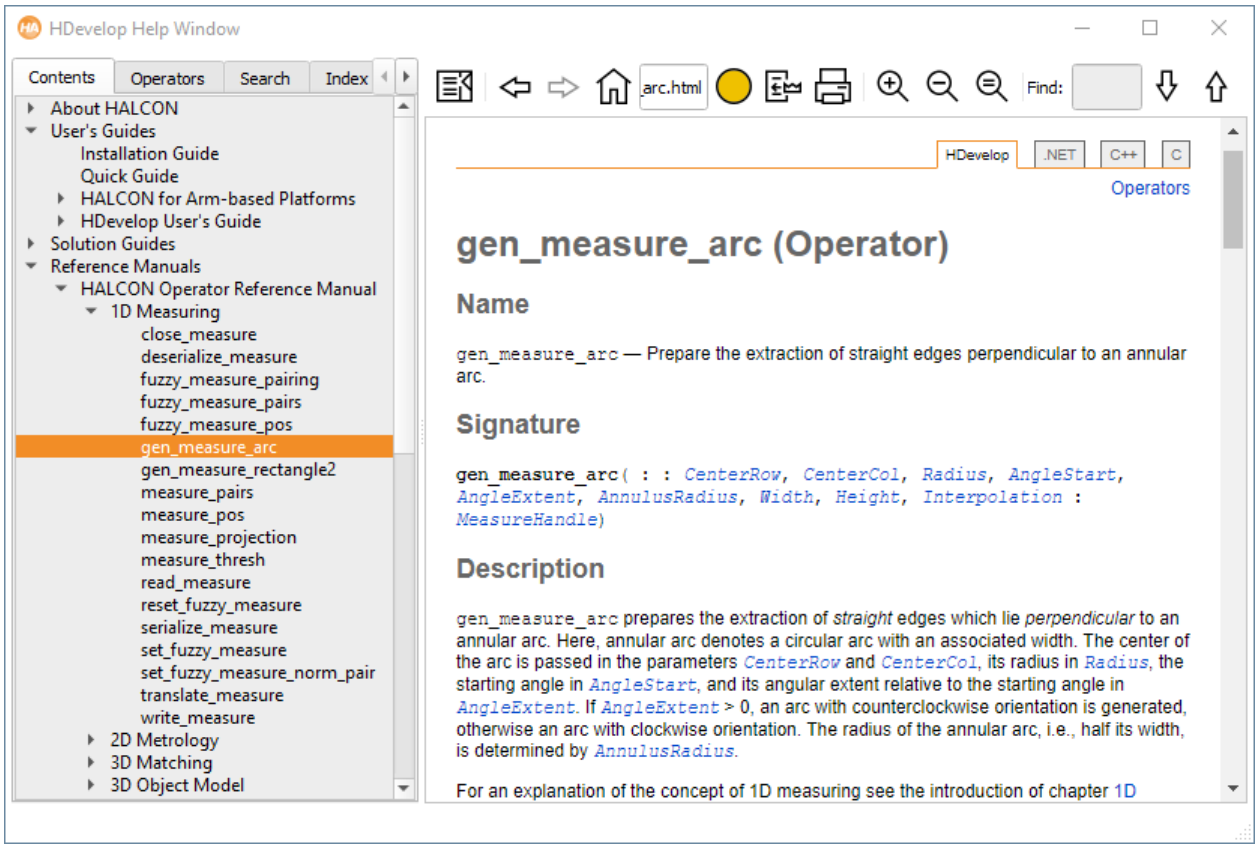


Figure 6.21: Help window with the contents tab card.

	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.
	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.

6.10 OCR Training File Browser

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, for example, if a sample has been assigned to a wrong symbol (please follow the links for a definition of the terms *symbol* (page 232) and *sample* (page 232)).

This tool is a useful addition to the OCR Assistant (page 231) 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 231) 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 231), 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 88) within the training file browser.

6.10.1 Windows of the Training File Browser

The OCR Training File Browser is composed of three windows (see figure 6.22). 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.

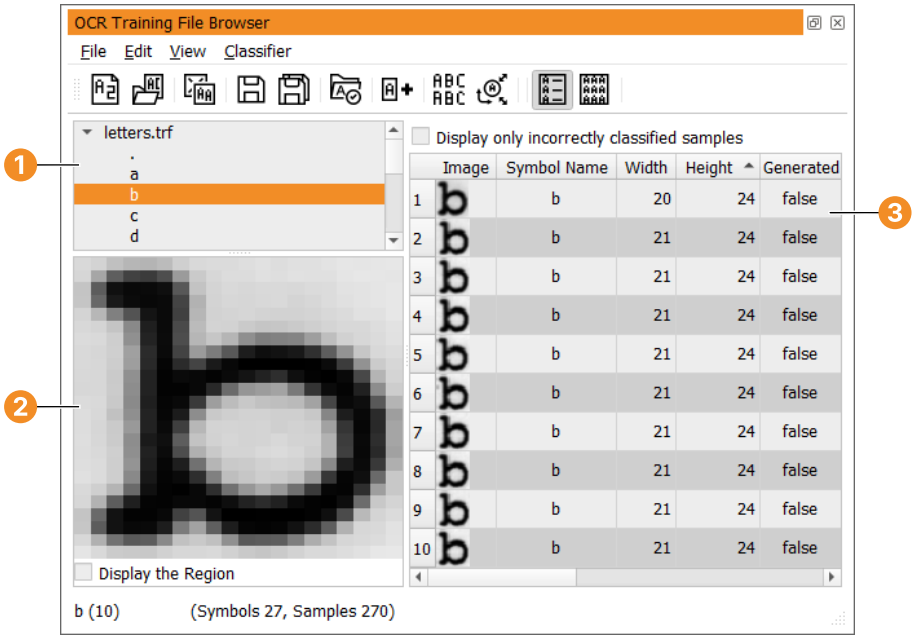


Figure 6.22: The OCR Training File Browser.

1 Zoomed sample window

2 Training file window

3 Sample inspection window

6.10.1.1 Training File Window

This window of the OCR Training File Browser enables you to view and edit training files (page 88) and symbols (page 88).

6.10.1.2 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.

6.10.1.3 Sample Inspection Window

The Sample Inspection Window enables you to view and edit samples (page 89). 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.

The columns show the following:

- `Image`: iconic sample
- `Symbol Name`: class to which the sample was assigned
- `Width`: width of the iconic sample (in pixels)
- `Height`: height of the iconic sample (in pixels)
- `Generated`: whether the sample is a generated variation (page 90) or not
- `Read Symbol`: result of reading the sample with the selected OCR classifier
- `Confidence`: a measure for the reliability of the read symbol (0: no confidence, 1: highest confidence)
- `Correct`: whether a sample has been read correctly. `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`.

6.10.2 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 231).
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 231)* 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 can include adding new samples, deleting samples, combining existing samples for a new training file or adding sample variations (page 90) as well as checking for classification problems. Use the Training File Window to edit the training file and the Sample Inspection Window to inspect and edit samples.
6. *Save changes*, for example, 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 231).

Note that all actions can be undone or redone by selecting `Undo` or `Redo` in the menu `Edit` to reverse or repeat changes, respectively.

6.10.3 Actions Within the Training File Browser

6.10.3.1 Training Files

The Training File Window (page 87) provides the following options for editing training files:

- *Open/Close training files*: When opening the OCR Training File Browser via the OCR Assistant (page 231), 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.

6.10.3.2 Symbols

The Training File Window (page 87) provides the following options for editing symbols:

- **Add a symbol**

  To add a new symbol to a training file, either
  - 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 either
  - 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 87).
- **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, either
  - 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 via drag and drop
  - 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.

  Moving a symbol (and therefore all samples that are assigned to this symbol) to another symbol is equal to assigning a new symbol name explicitly.

6.10.3.3 Samples

The Sample Inspection Window (page 87) 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, click the feature that should be used for sorting (for example, `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 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 231).*

Edit the Samples

- **Copy a sample**

  Select one or more samples in the Sample Inspection Window (page 87). In the Training File Window (page 87), they can then be copied to
  - another symbol in the same training file or
  - another symbol in a different training file.

  To copy and paste a sample, either
  - drag and drop the sample while pressing Shift,
  - 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 87). They can then be moved in the Training File Window (page 87) via drag and drop to
  - another symbol in the same training file or to
  - another symbol in a different training file

- **Delete a sample**

  Select one or more samples in the Sample Inspection Window (page 87). 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 87).)

6.10.3.4 Generating Sample Variations

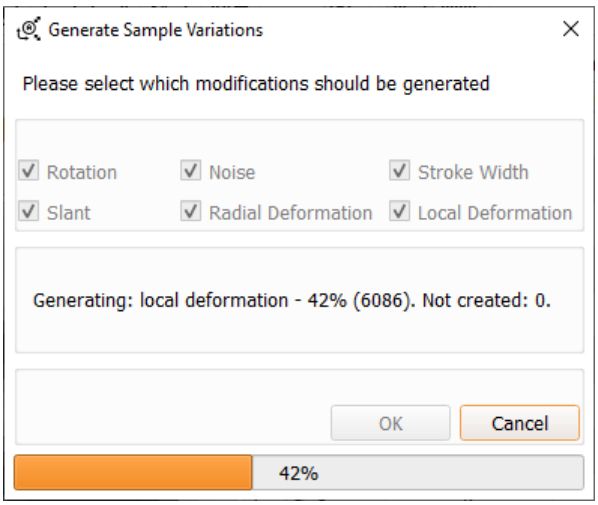


Figure 6.23: The Generate Sample Variations dialog.

The best classification results are achieved if the OCR font is trained using real data from the target application. This can 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 87) to generate variations for this sample or
- **a certain symbol** in the Training File Window (page 87) 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 87) 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)

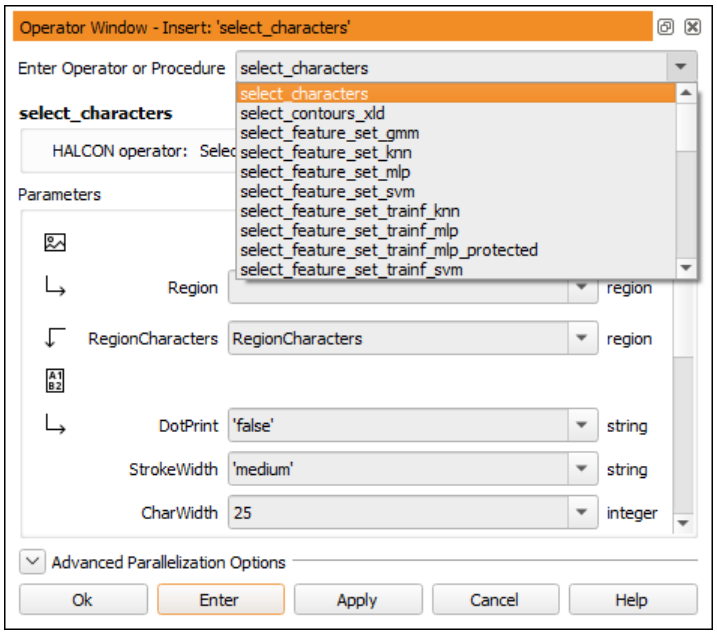


Figure 6.24: Selecting an operator after typing `select_`.

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 87) where they are marked with `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.

6.11 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 can 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.
- 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 8.11.1 on page 283).
- The row of buttons at the bottom allows you to control the parameter display.

6.11.1 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.

6.11.2 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, with the name, value, and semantic type of each parameter.

You can switch between a one-column (default) or two-column layout of the parameter display. In the two-column layout, the iconic parameters are shown on the left and the control parameters on the right.
In the following description, "column" refers to the order of the entries, which is independent of the layout.

- 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.
- 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 can 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 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 section 6.16.12 on page 123.

  This column can also contain action buttons for special semantic types, for example, 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.

Please refer to the following rules on how parameters obtain their values and how you can 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 can specify new variable names, meaning, 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 can 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.

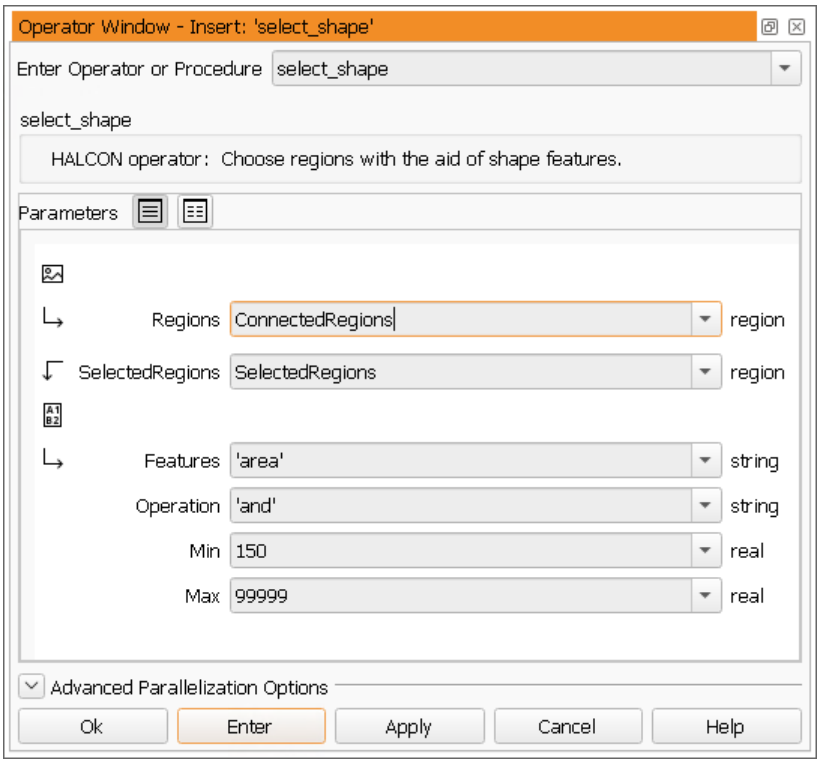


Figure 6.25: Specifying parameters for the operator `select_shape`.

Control input parameters These parameters normally possess a default value. As an alternative, you can 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. You can build up expressions with these values. You can use expressions in HDevelop similar to the use of expressions in C or in Pascal. You will find a detailed description in section 8.5 on page 253.

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 can 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 can 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 appendix D on page 327 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 can 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.

6.11.3 Control Buttons

Below the parameter display, you find five buttons that comprise the following functions:

OK Click `OK` to execute the operator or procedure call with the specified parameters. 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 did not disable this option (compare section section 6.16.14 on page 125). 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 the execution is canceled. The operator window then remains open to allow appropriate changes.

Enter / Replace Click `Enter` and 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 section 6.17.2 on page 128), the button label changes to `Replace`. When clicked, the original program line is replaced.

Apply 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.

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 Click `Cancel` to clear the contents of the operator window. Thus, there are neither changes in the program nor in any variables.

Help Click `Help` to invoke the online help for the selected operator or procedure. For this the system activates the online help window (see Help Window).

6.12 Output Console Window

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 6.26 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 48)) compiler, and file operations.

(Warning) Warning events related to the program execution, the just-in-time (JIT (page 48)) compiler, file operations, HALCON low-level errors, and invalid program lines.

(Error) Program execution errors, HALCON errors, and internal errors.

(Exception) Exception events.

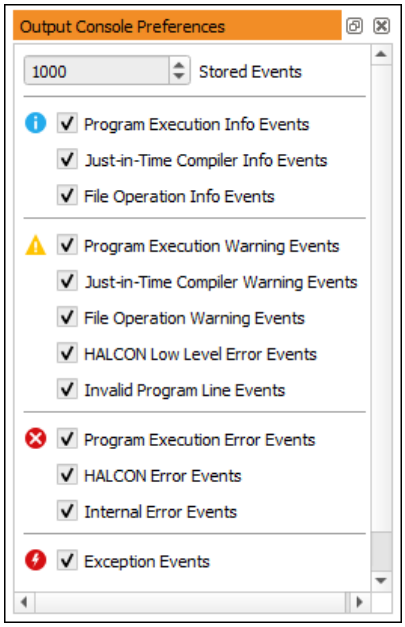
The tool bar includes the following action buttons:

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.



6.13 Plot Windows

HDevelop uses plot windows for the visualization of data in several areas. Differences specific to a given task are described in the corresponding sections listed below:

- Inspection of 1D functions and tuple data (section 6.22.7 on page 171).
- Visualization of gray value histograms (section 6.13.2 on page 99).
- Visualization of line profiles (section 6.13.5 on page 107).
- Visualization of feature histograms (section 6.13.3 on page 104).

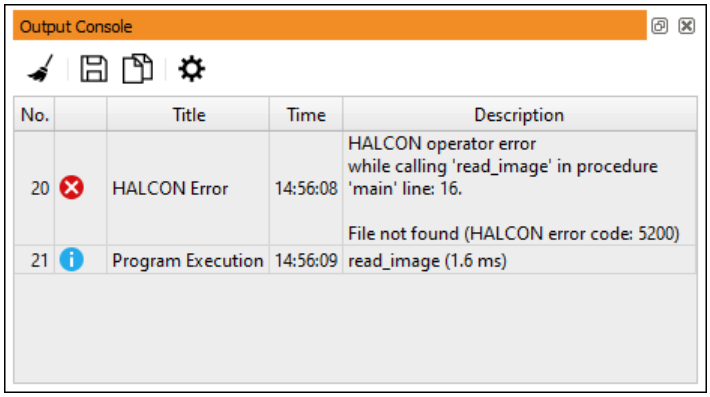


Figure 6.26: `Output Console`.

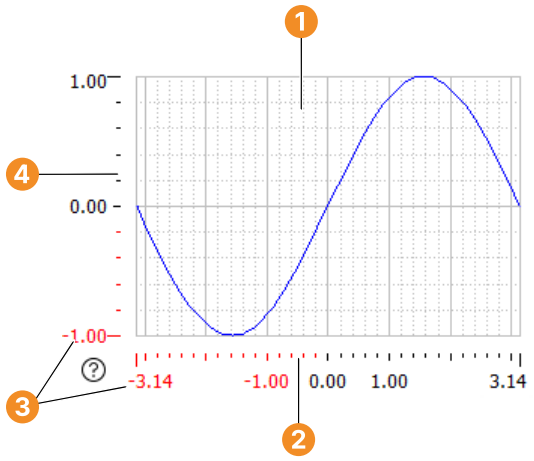


Figure 6.27: Example plot window.

1. Plot area
2. x-axis
3. Axis limits
4. y-axis

- Visualization of fuzzy measure graphs (section 7.4.5.2 on page 225).

An example plot window is illustrated in figure 6.27.

6.13.1 Interacting With Plot Windows

6.13.1.1 Mouse-based Operations

The following mouse-based operations are supported to manipulate the plot display:

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.

6.13.1.2 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 can be set to linear or logarithmic mode independently.

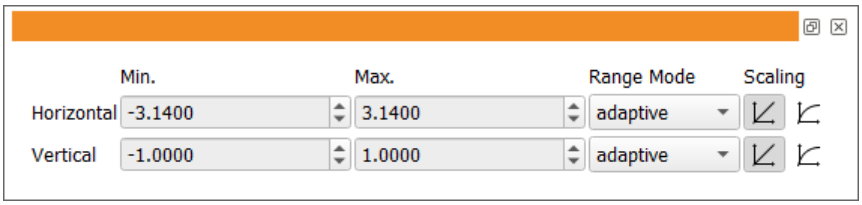


Figure 6.28: Plot bounds.

6.13.1.3 Plot Window Context Menus

The following context menu entries are valid for all plot windows. Depending on the plot window type, additional entries can 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 97)).
	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.

6.13.2 Gray Histogram Window

The gray histogram window is a 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. See also: Menu Visualization ▷ Gray Histogram. 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, for example, 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 y-axis (1) represents the frequency of the gray values. 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, for example, 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 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, that is, the gray value which 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. The range of input values is divided in quantization steps to obtain a meaningful histogram, and, as a consequence, the histogram's "peak value" can actually represent a whole range of input values.

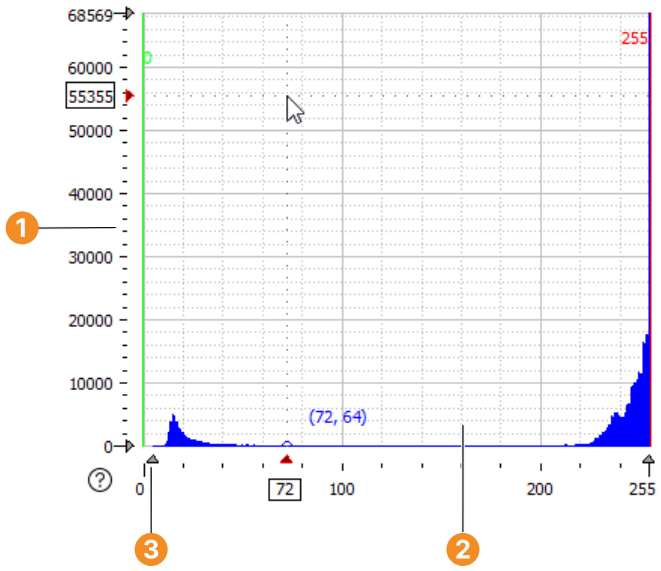


Figure 6.29: Gray histogram.

1. y-axis
2. Plot area
3. x-axis

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. 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 (for example, 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 (for example, 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 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`

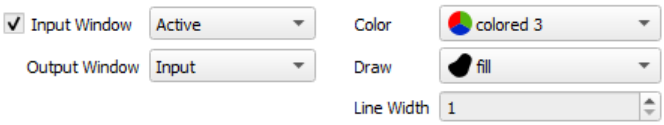
These controls define the visible area of the histogram and the way it is displayed.

- `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 6.13.2.2.

`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 56)
- `Draw` (page 56)
- `Line Width` (page 58)

Sometimes, it is desirable to suppress the updating of the histogram when new image data is available, for example, 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.

6.13.2.1 Context Menus

The common context menu entries are described in section 6.13.1.3 on page 98.

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

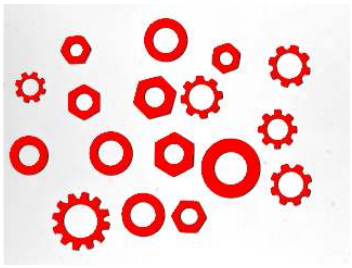
Note that histogram quantization is still performed using equal-sized linear bins if the x-axis is set to logarithmic scale.

Y-Axis

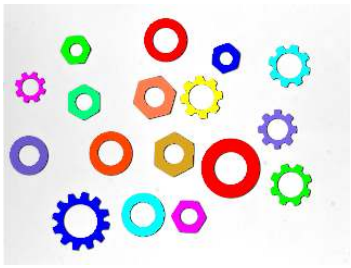
In logarithmic mode no negative values are permitted when dragging the plot area using the mouse.

6.13.2.2 Interactive Visual Operations

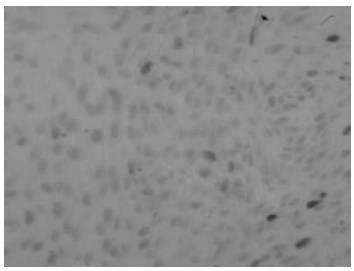
The selected range of gray values can be used for two major purposes: Thresholding (segmentation) and scaling the gray values. This is illustrated using the images *rings/mixed_03.png*, and *meningg5.png*.



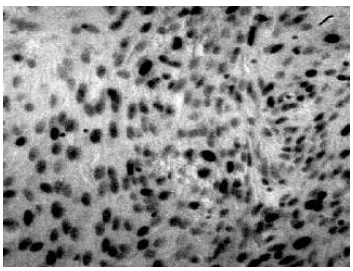
threshold



threshold + connection



no scale



scale

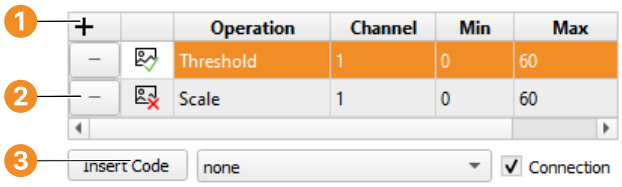
The setting of `Output Window` specifies the graphics window that is used to visualize the gray values between the green line and the red line: You can select the originating window (`Input`), the active graphics window (`Active`), or an arbitrary window ID from the list.

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, for example, 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` can 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 can specify its own range of gray values.



1 + button

2 – button

3 Icon

6.13.2.3 Threshold Operation

The image from which the histogram was computed is segmented with a `threshold` operation using 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 56), `Draw` (page 56), and `Line Width` (page 58)).

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, set the visualization color to a color different from black or white. 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.

There are two possibilities to select threshold parameters displaying the segmentation for an original image, derived from another image.

You can either 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.

Or, 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
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.

6.13.2.4 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. The combo box `Output Window` determines the graphics window, in which the result is displayed. Use this to interactively set a "window" of gray values that should be displayed with a large dynamic range. Note that, not more than one scale operations can be visualized in the graphics window at the same time.

Click `Insert Code` to generate HDevelop code that performs the currently visualized scale operation in your program. The code is inserted at the IC.

6.13.3 Feature Histogram Window

See also: Menu `Visualization` ▷ `Feature Histogram`.

This window provides a tool for the inspection of feature histograms. In contrast to the gray value histogram, 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*, for example, 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.

The default feature selection displays the histogram of the area, 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`. For more information about region features see section 6.13.4.

See figure 6.30. 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, , 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 6.13.2 on page 99 (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, for example, 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

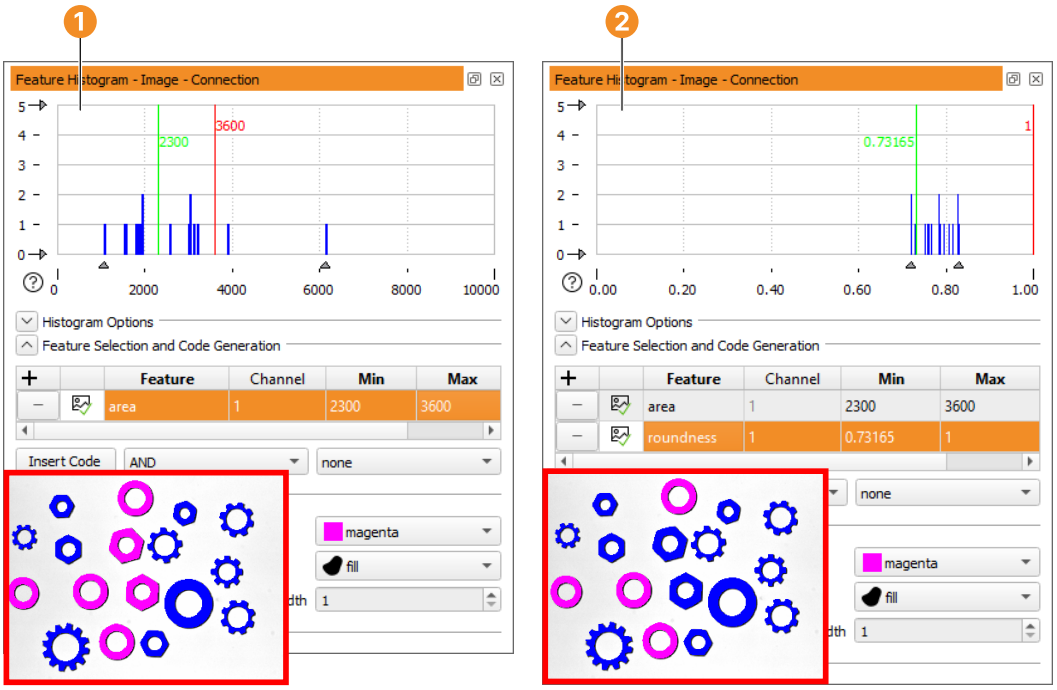


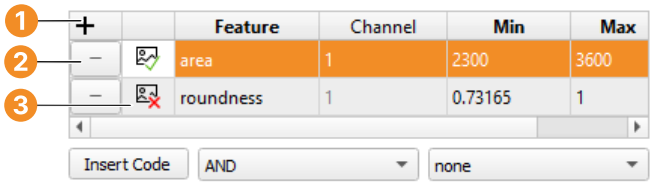
Figure 6.30: Combining different features selections.

Step 1 select regions of similar size

Step 2 restrict the selection to round regions

feature: For features with floating-point values, for example, the orientation, the probability that two regions or XLDs have the same feature value is very small, in contrast to features with integer values, for example, 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`.



1 + button

2 − button

3 Icon next to the operation

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.

6.13.4 Feature Inspection Window

See also: Menu `Visualization` ▷ `Feature Inspection`.

This tool is used for the inspection of shape and gray value features of individual regions and XLDs. It can, be used to determine thresholds for operators that select regions based on these features, for example, `select_shape` or `select_gray`.

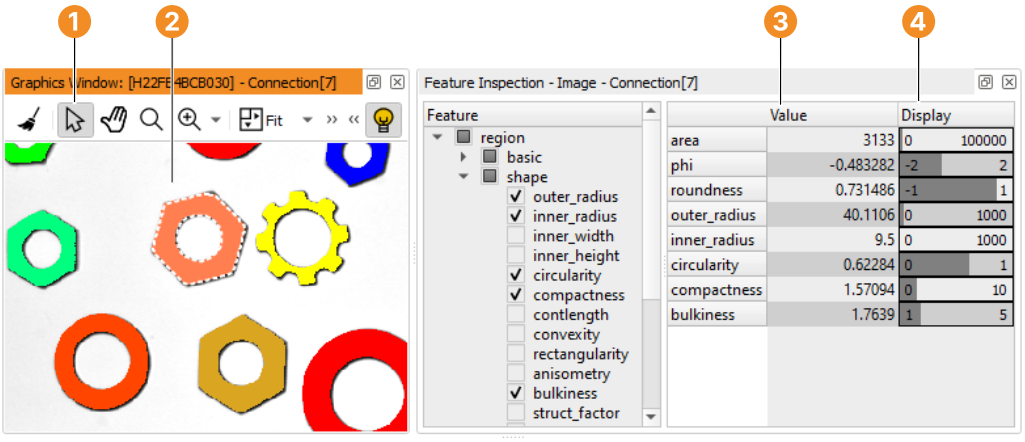


Figure 6.31: Inspection of selected features.

1 Select

2 Selected region

3 Feature value of selected region

4 Range visualization

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 6.13.2 on page 99). You can display an image or region by double-clicking 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 (for example, 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.

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 categories.

- Region features: This group contains features that describe the selected region, for example, 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, for example, minimum and maximum gray value, mean gray value, anisotropy and entropy.
- XLD features: This group contains features that describe the selected XLD (for example, 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. Select `Show Minimum/Maximum`, which is available in the context menu of the right side of the window.

See figure 6.31 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.

6.13.5 Line Profile 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 6.32. Using the line profile is helpful in particular for optimizing edge detection in the Measure Assistant (page 221) or when checking the focus of your camera (see the section 'Focusing Your Camera' (page 110)). The displayed line profile of the ROI is described in the section 'Line Profile Display'. 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.

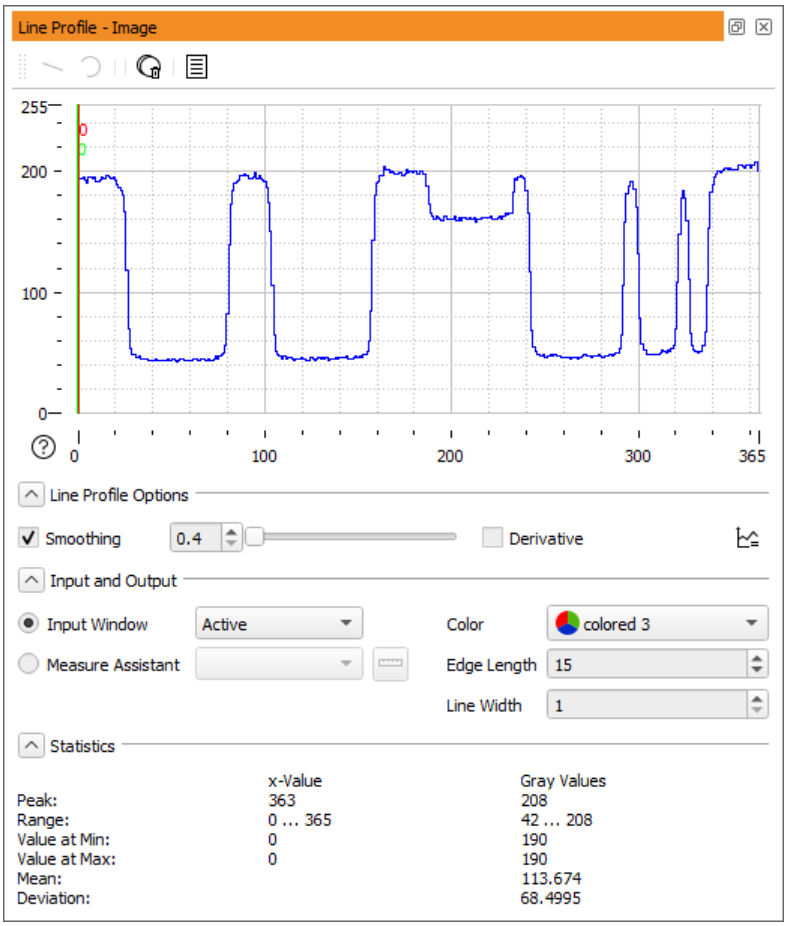


Figure 6.32: The Line Profile Window.

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.

6.13.5.1 ROI Menu of the Line Profile Window

If the Measure Assistant (page 219) 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 an 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.

6.13.5.2 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, for example, 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, for example, 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 6.13 on page 96.

6.13.5.3 Context Menus

The common context menu entries are described in section 6.13.1.3 on page 98. 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.

6.13.5.4 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 219), 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 6.33).

`Derivative`: Display the derivation of the line profile.

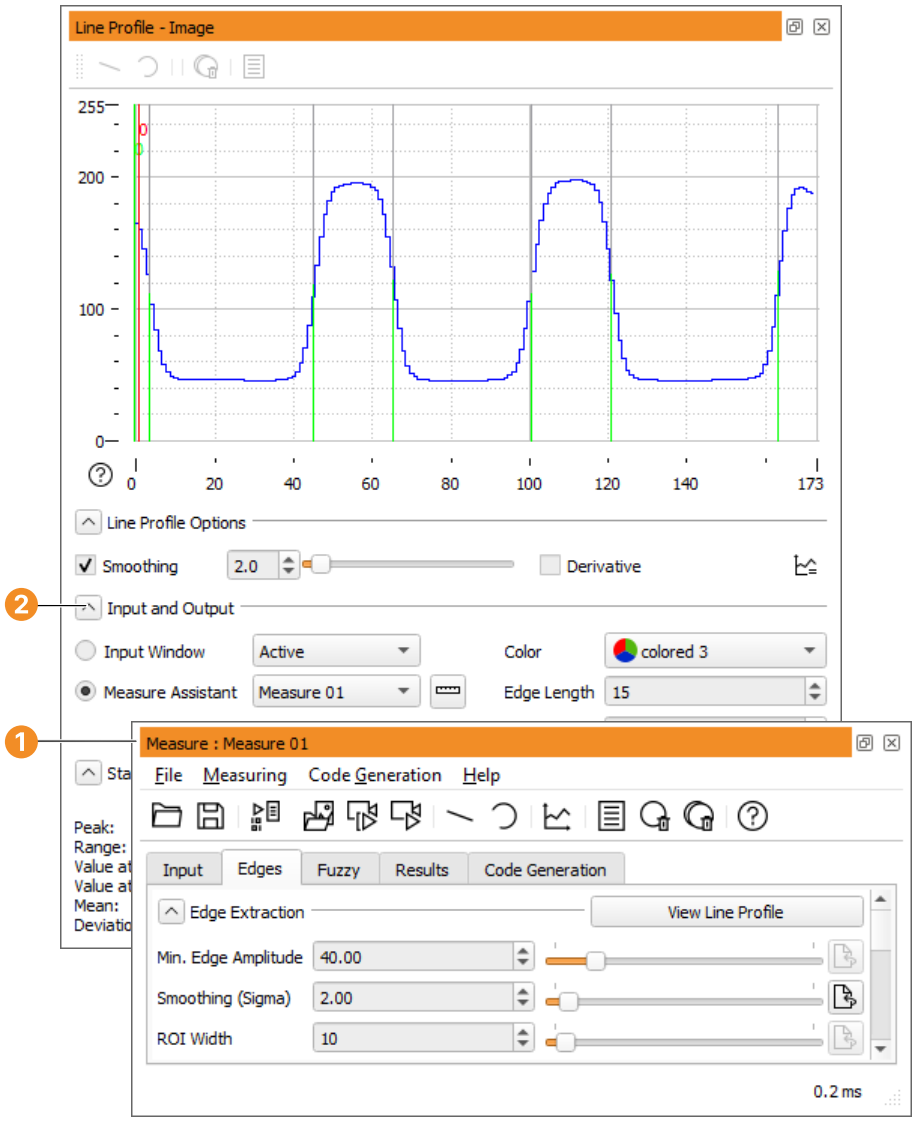


Figure 6.33: Line Profile dialog.

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 .

6.13.5.5 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 219) 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.

6.13.5.6 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.

6.13.5.7 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 6.34).

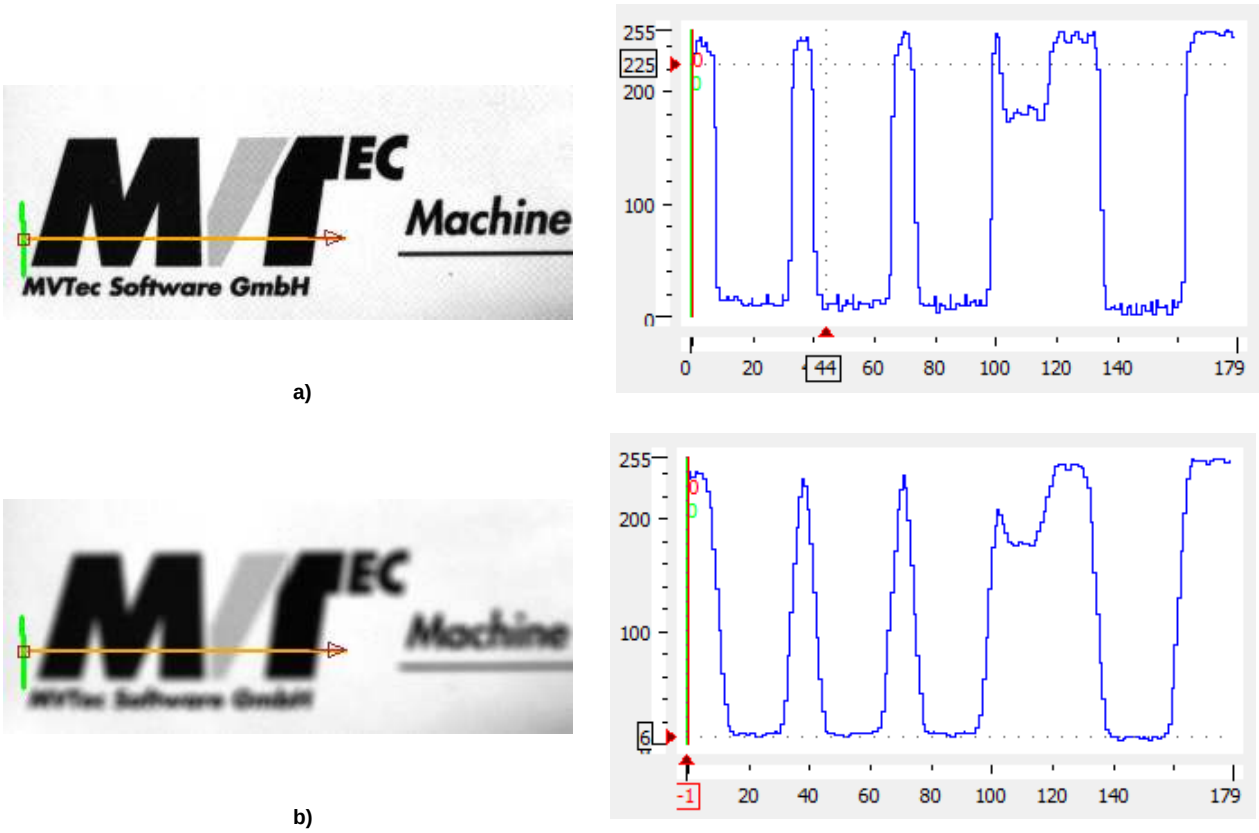


Figure 6.34: a) profile in a sharp image; b) profile in an unsharp image.

6.14 Properties Dialog

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 6.35.

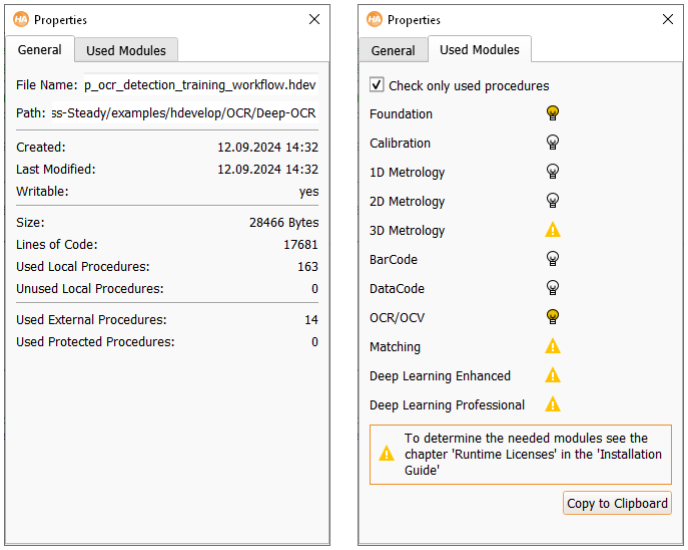


Figure 6.35: `Properties`: `General` (left), `Used Modules` (right).

The tab card `Used Modules` lists the HALCON modules used by the current program. Modules marked with are used. Modules marked with are potentially in use by dynamic licensing. 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, section 6.4.2 on page 31 for more information about modules, runtime licenses, and dynamic licensing. See figure 6.35 for the corresponding dialog of a 3D object processing 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.

6.15 Print Dialog

The print dialog is displayed in figure 6.36.

Print Range

`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.

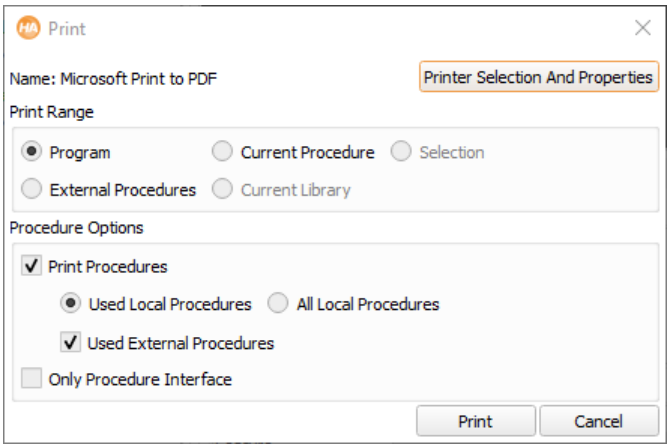


Figure 6.36: `Print`.

- `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 5.6 on page 47) are not printed.

6.16 Preferences Dialog

HDevelop maintains a set of preferences that are persistent between sessions. You can customize the appearance of the 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.

i **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`

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>`.

`Export` Using the check boxes you can specify which settings will be saved to the selected file.

`Import` In the import dialog, you can select a file with previously saved HDevelop preferences (default file extension: `.hdp`). This selection of preferences was saved using the menu entry `Export`. 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.

`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.

6.16.1 User Interface ▷ Program Window

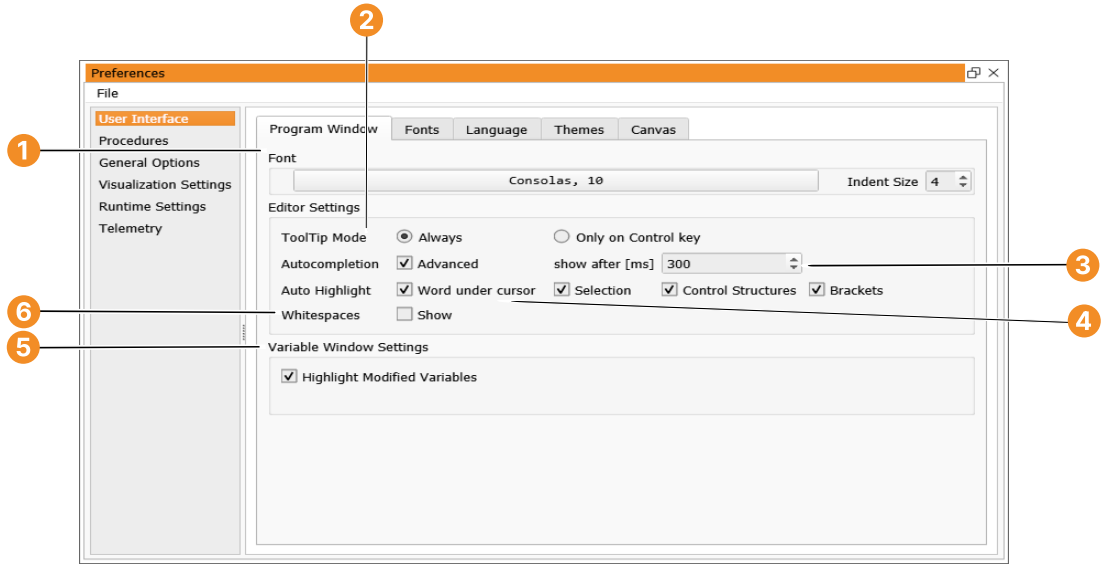


Figure 6.37: User Interface ▷ Program Window.

1 **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.

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

2 **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 are displayed. When placed over an instantiated iconic variable, an icon is displayed. When placed over the program line column, warning messages may be displayed, for example, if the used operator is deprecated. This behavior can 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.

3 **Autocompletion**:

- **Advanced**

  Enables advanced autocompletion in the program window (see section 6.17.2 on page 128).

- **show after [ms]:**

  This value defines the delay before the autocompletion is displayed.

4 **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.

5 `Variable Window Settings`:

Enables the highlighting of modified variables in the variable window (page 160) 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.

6 `Whitespaces`

Enables the visualization of whitespaces within the program window. By default this option is turned off.

6.16.2 `User Interface` ▷ `Fonts`

In this tab card, the font settings of HDevelop can 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 64) ▷ `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.

6.16.3 `User Interface` ▷ `Language`

By default, HDevelop uses the language that is specified as the display language of the operating system. If the language is changed via these dialogs, the operating system locale is ignored.

- `HDevelop GUI`

  Changes the language of HDevelop's Graphical User Interface and the language of the Program and Procedure Reference within HDevelop's help window. The new language will be applied the next time HDevelop is started.

- `Operator Reference`

  Changes the language of the HALCON Operator Reference within HDevelop's help window (page 81), as well as the language of the operator descriptions within the autocompletion and the operator window (page 91).

  The new language of the operator descriptions is applied immediately. Note that some changes, like the language of the operator window, only take effect after the next time HDevelop is started.

6.16.4 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.

6.16.5 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 45).

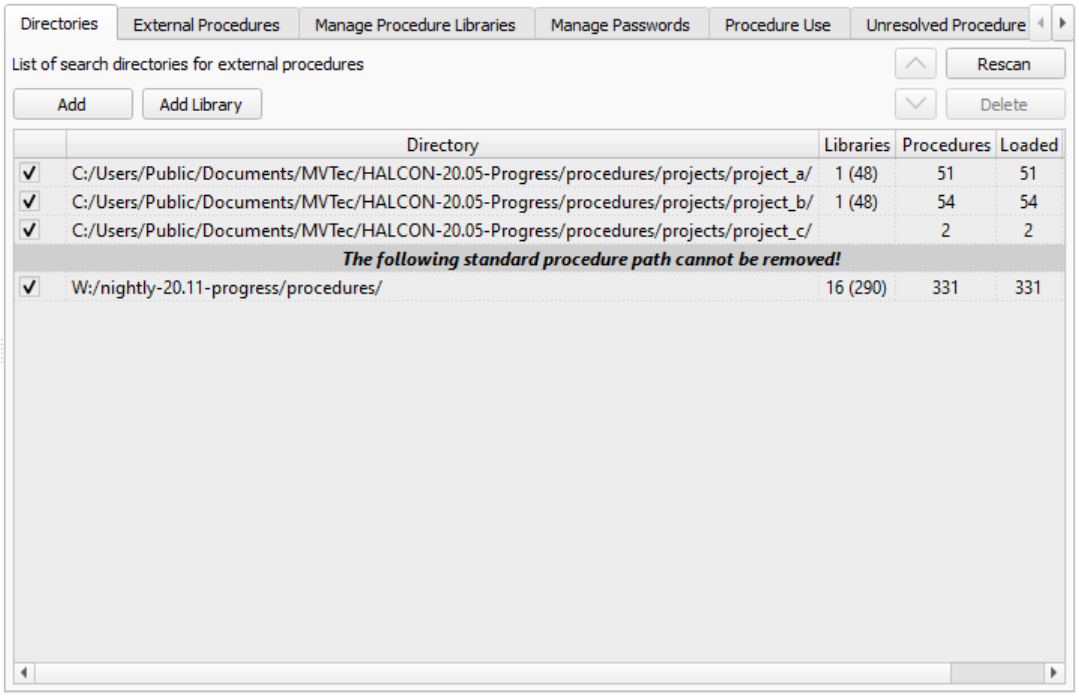


Figure 6.38: Procedures ▷ Directories.

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, that is, 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. 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.

6.16.6 `Procedures` ▷ `External Procedures`

This tab card lists all external procedures in the order they are loaded from the configured directories (page 115). For each procedure, the following information is displayed:

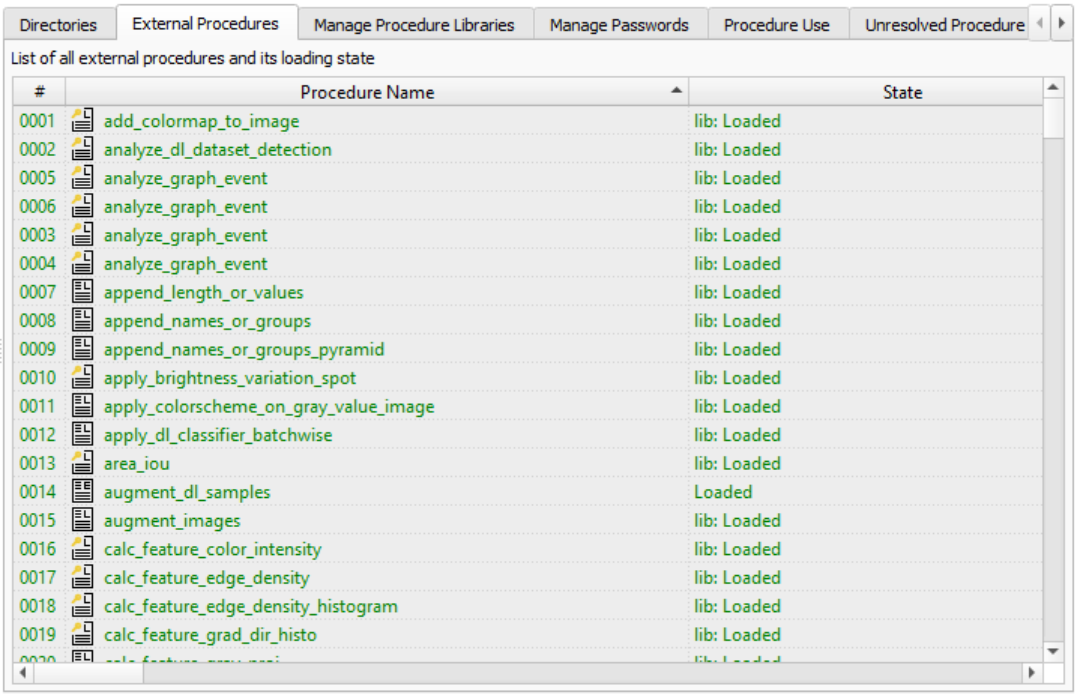


Figure 6.39: `Procedures` ▷ `External Procedures`.

Column	Meaning
#	Number of the external procedure.
Activated	The check box toggles the activation status of the corresponding procedure, that is, whether the procedure can be resolved. It can be used to temporarily disable procedures, for example, 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, for example, 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.

6.16.7 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).

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, to mark them for moving.

Copy the selected procedures, to 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.

For more information about procedures and their properties, see section 5.1 on page 43 and section 5.6 on page 47.

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).

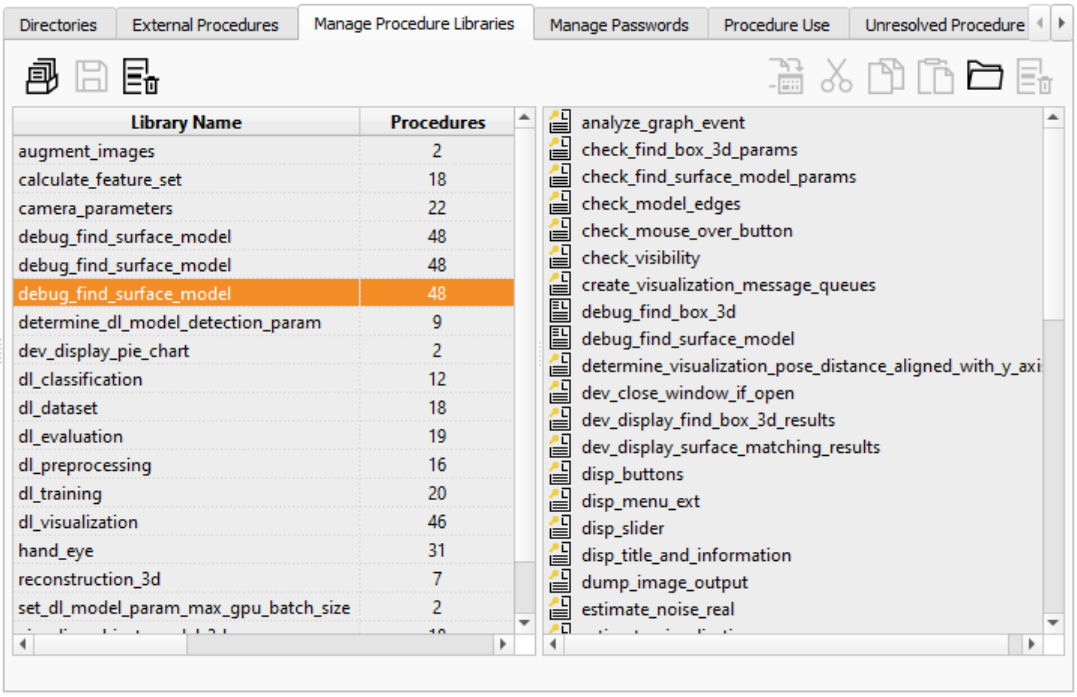


Figure 6.40: `Procedures` ▷ `Manage Procedure Libraries`.

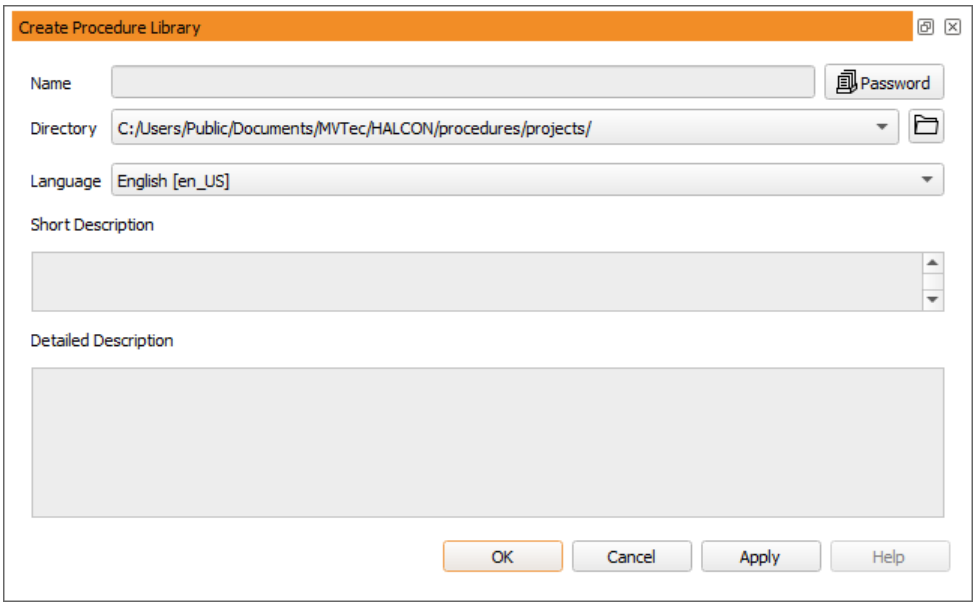


Figure 6.41: Create a new procedure library.

Create a new procedure library (or edit existing libraries)

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 6.17.9 on page 146).

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).

6.16.8 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 47.

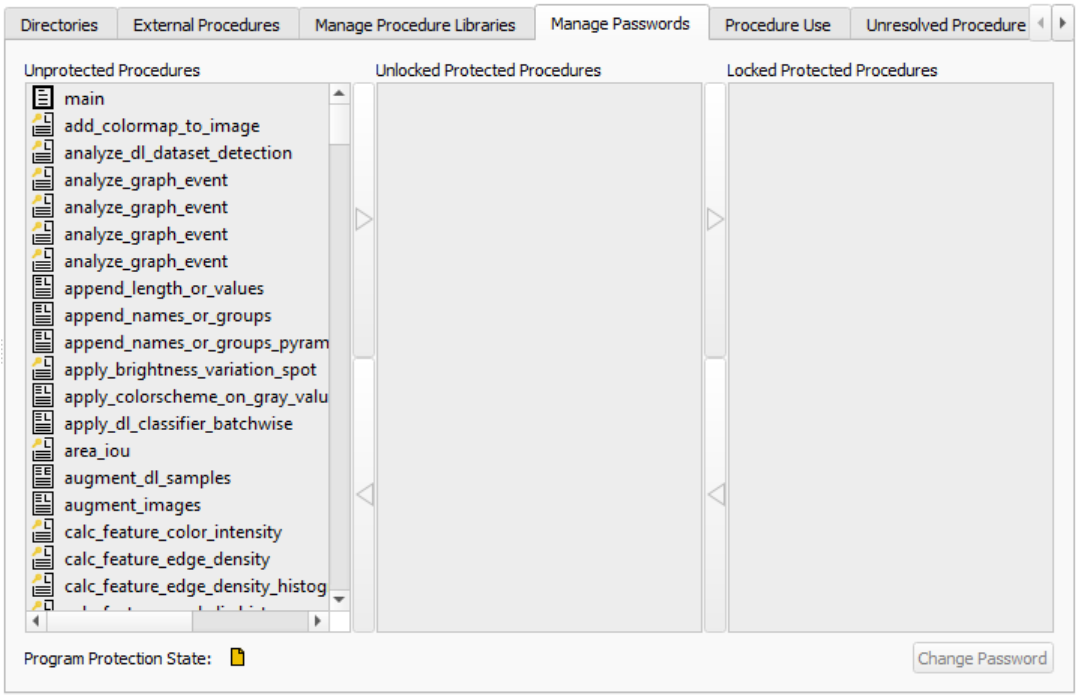


Figure 6.42: Procedures ▷ Manage Passwords.

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 6.17.9 on page 146. 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. 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 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 6.17.9 on page 147.

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 6.60 on page 147, 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 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 53).

6.16.9 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 6.43.

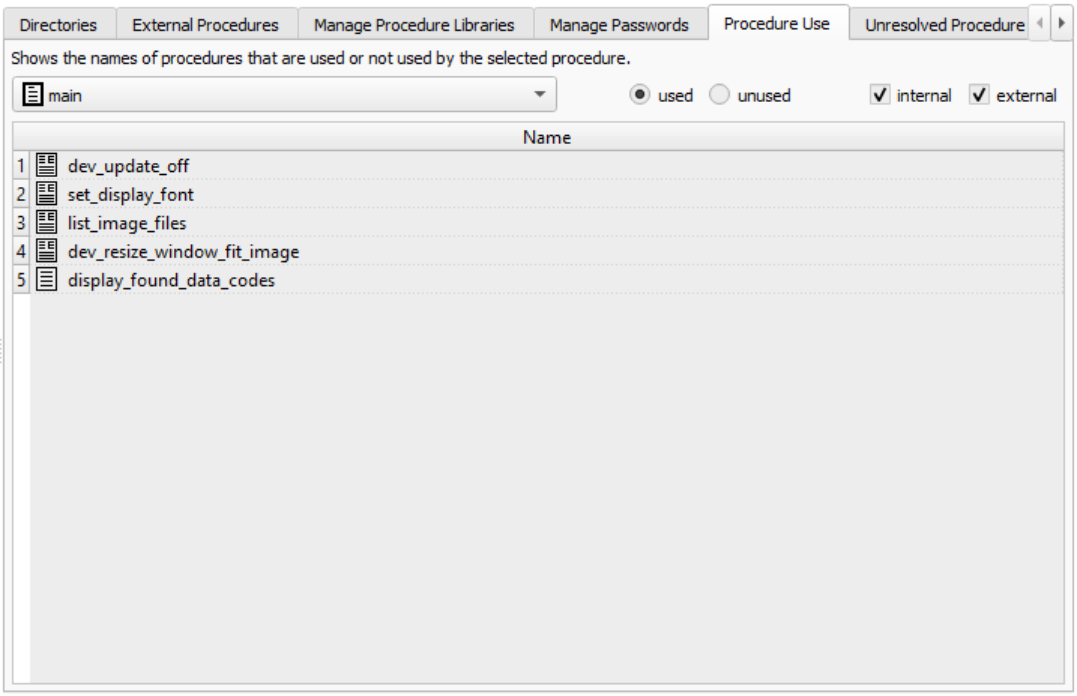


Figure 6.43: Procedures ▷ Procedure Use.

6.16.10 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.

6.16.11 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, as they are entered. See table 8.2 on page 249 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 160) and variable inspection windows, see section 6.22 on page 167.

- Precision for displaying mouse position values

  If set to a value greater than 0, subpixel mouse positions are enabled. See Position Precision (page 58) for more information.

- Default resize mode for new Graphics window

  This option controls the default resize mode for newly opened graphics windows. See section 6.1.4 on page 58 for more information about the resize modes. The default resize mode applies only to graphics windows that are opened via menu, 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 73), the zoom window (page 177), the gray histogram window (page 99), the feature histogram window (page 104), and the line profile window (page 107). 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 164).

- Return relative file paths when appropriate

  If enabled, HDevelop will try to convert file names selected from file selection dialogs 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 top level directory. This will be useful in cases where, for example, 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, for example, 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, that is 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 appendix C.1 on page 323).
- `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, for example, 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 5.2 on page 44). 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 5.2 on page 44.
- `Default file format for saving HDevelop procedures`

  Sets the default file format for new external procedures, see section 5.2 on page 44.
- `Update windows in single-step mode`

  Specifies the window update policy when stepping through the program. The different options are described in section 6.1.4 on page 58.
- `Maximum number of subthreads`

  Specifies the maximum number of subthreads that be started using the `par_start` qualifier (see section 8.11 on page 283).
- `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 6.44 for an illustration of the window offset.
- `Origin of coordinates`

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

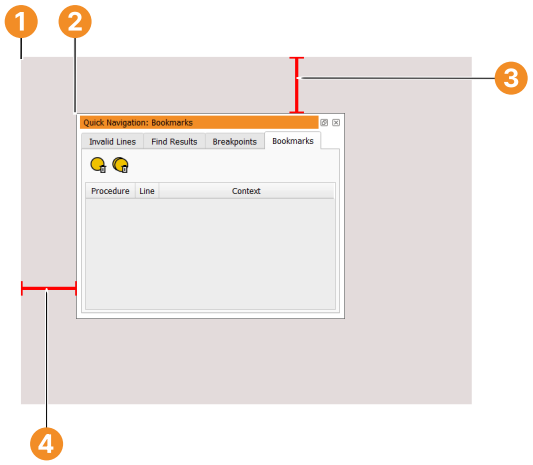


Figure 6.44: Window offset.

1. Reference point for the window position by default (upper left corner of the desktop)
2. The origin of the HDevelop window is its upper left corner
3. Value definded by the operator values for `Row` here (0, 0)
4. Value definded by the operator values for `Column` here (0, 0)

- `Main display`

  The main display is usually defined in the display settings of your operating system. **If you have only one monitor, this is the main display.** This display acts as the active desktop. It shows the taskbar and most items will open on the main display by default.

- `Display showing HDevelop`

  The display that currently shows the HDevelop main window (page 51). **This setting is only relevant if you use multiple monitors.** Otherwise, the `Display showing HDevelop` is the same as the `Main display`.

- `HDevelop`

  HDevelop's main window (page 51).

Please also note the setting `Window open offset`.

- `Floating windows behavior`

  Specifies the accessibility of floating windows. To apply the new settings, a restart of HDevelop is required. This only refers to graphic, tool and assistant windows.

  - `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.)

6.16.12 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 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. 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 section 6.11.2 on page 92.

- 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 5.9 on page 48. The JIT compilation is globally enabled or disabled using this setting.

- File tracking behavior

  By default, the file tracking is set to System file tracking. This option may limit the number of tracked files. When Application file tracking is activated, a timer checks for changed files. This option can affect the operating system performance. Switch off the file tracking by choosing No file tracking.

6.16.13 Visualization Settings

The tab cards in this category define the default visualization settings for graphics windows when HDevelop is started. See section 6.8.2 on page 76. The dialog is displayed in figure 6.45.

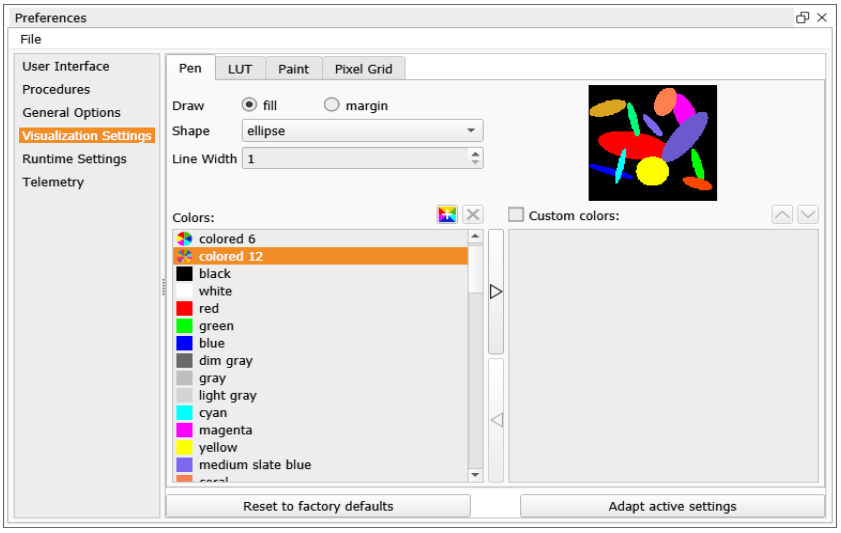


Figure 6.45: Visualization Settings ▷ Visualization Parameters.

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

`Adapt active settings` Adapt the user-defined colors and custom color sets permanently. To define your own colors and color sets, use the `Visualization Parameters` dialog (see section 6.8.2.1 on page 77).

6.16.14 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 section 6.3 on page 65).

- `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 can 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 can 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, for example, 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 can 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 can be disabled. This is desirable, for example, 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 6.4 on page 57.

6.16.15 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, that is, 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, for example, for performance measuring purposes. At other times, you would like to stretch the delays, for example, 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

6.16.16 Telemetry

You can help us improve your HALCON user experience by anonymously sharing usage data with MVTec. This will also allow you to receive additional information on our latest or upcoming products. 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, such as 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, such as selected language, HALCON version and edition, workflow information, like export and tool usage.

`HDevelop Help usage statistics` The sent data comprises workflow information inside HDevelop Help, like 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`.

6.17 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 63)). To switch between selected procedures rapidly, program windows support multiple tabs.

The program window (see figure 6.46 on page 129) is divided into three areas:

- The main part of the program window contains the program code of the current HDevelop procedure. See section 6.17.2.
- The column at the left side displays line numbers. It also contains the PC, the IC, and optionally, one or more breakpoints. Bookmarked lines and warning indicators are displayed. See section 6.17.3 on page 133.
- 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. The number of parameters as well as their names and types, can be modified. See section 6.17.5 on page 135 for a detailed description.

6.17.1 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.

	Action	Shortcut	Description
	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 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.
	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 or Ctrl+Enter	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.

6.17.2 Editing Programs

The program window supports full text editing. Continuous portions of text can be selected by dragging with the mouse. An existing selection can 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, for example, 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.

Figure 6.46: 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

Comments

Comments start with an asterisk (*) as the first character. You can 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 can 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 can span several lines for readability. A line can 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 (for example, `if` or `while`), the indent is increased by the indent size. The indent size is specified in the preferences (see `Program Window` (page 113)). It defaults to four spaces. If a control structure is closed (for example, by entering `endif` or `endwhile`), the indent of the current line is decreased by the indent size.

Advanced Autocompletion

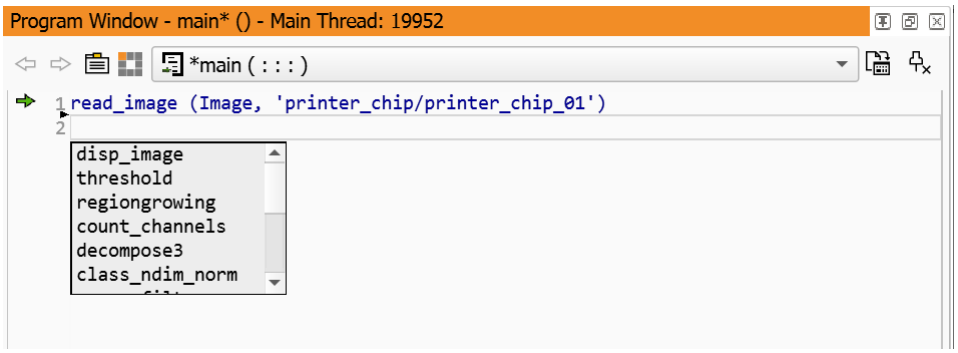
The program window provides advanced autocompletion to support you in several ways.

- **Getting started**

  HDevelop's autocompletion can assist you getting started with a completely empty program. Press Tab to get a list of typical first operators like `dev_open_window` and `read_image`.

- **Suggestions for empty code lines**

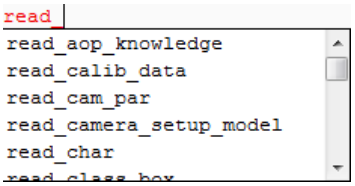
  Possible successors of the previous HALCON operator are suggested after pressing Tab.



Press Shift+Tab to get suggestions for possible predecessors.

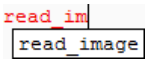
- **Typing support**

  When you start typing a new line, HDevelop will suggest a list of matching operator names:

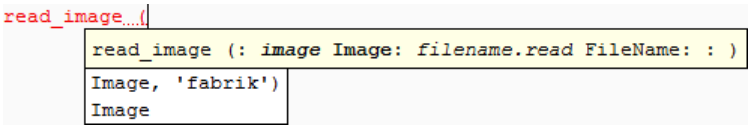


  Note that the line is highlighted as invalid (red in the default color scheme) because it is still incomplete.

  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:

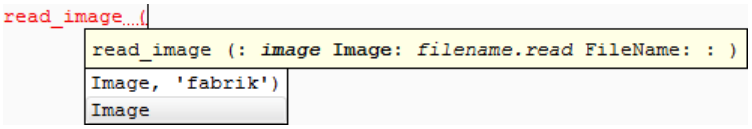


  Once the cursor moves inside the parentheses, the suggestion list changes from operator mode, to parameter mode. The signature of the selected operator is displayed, and the parameter corresponding to the cursor position is highlighted in bold.

  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.

  Press Tab to select the first suggestion,

  or press Up or Down to step through the list entries,



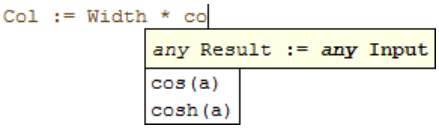
and press Tab or Return to select the highlighted entry. Then, enter a comma or press Tab again to get suggestions for the second parameter:

Double-clicking the browse button in the suggestion list opens up a file selection dialog to specify the file name parameter. These browse buttons are included in the suggestions lists of all parameters that specify a file name.

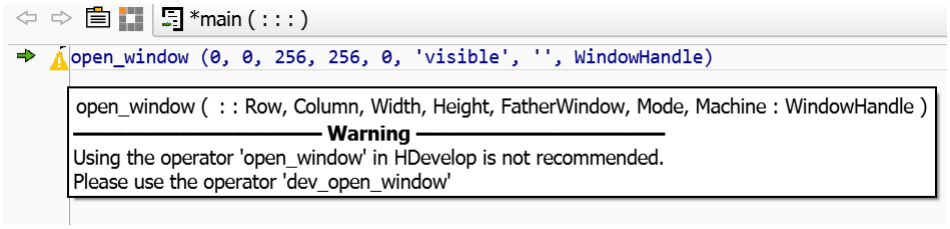
In this example, we want to load the image `clip`, so none of the suggestions fits. Just type the file name in single quotes (`'clip'`) and press Tab to complete the parameter list. The closing parenthesis is inserted automatically:

```
read_image (Image, 'clip')
```

When entering expressions, the advanced autocompletion also suggests operations supported by the HDevelop language (see chapter 8 on page 247).



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, for example, the control statement of a `for` loop, the corresponding end statement, for example, `endfor` of the block is automatically inserted by pressing Return.

Special Keyboard Shortcuts in the Program Window

Tab		*Cursor at the beginning of line:* Adjust indentation of current line
		Selected text: Indent corresponding code lines one level
	AC	*Operator suggestions:* Complete to highlighted suggestion or to longest common string from suggestion list
Shift+Tab		*Selected text:* Outdent corresponding code lines one level
Ctrl+Return		Execute current line (same as clicking `Apply` in operator window; see section 6.11.3 on page 95)
Shift+Return		*Cursor at the end of line:* Execute current line and insert new line (depends on the setting of the `Select the behavior of pressing the [Return] key` option; see section 6.16.11 on page 121)
Ctrl+F		Open find/replace dialog with selected text (see section 6.7 on page 72).
Escape	AC	Hide suggestion list
Ctrl+Space	AC	Re-display suggestion list based on cursor position or selection
Up	AC	Highlight previous entry in suggestion list
Down	AC	Highlight next entry in suggestion list
	AC	*Parameter suggestions:* Complete to highlighted suggestion or first suggestion if no suggestion is highlighted
Return	AC	*Cursor at the end of a block statement*: Insert the corresponding end statement of the block

Entries marked with AC are only available if advanced autocompletion is enabled.

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. See the previous section for an example (`read_image`).

The operators `info_framegrabber` and `open_framegrabber` provide a button to detect the available image acquisition interfaces automatically (see also Image Acquisition Assistant (page 180)).

6.17.3 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 indicates 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 can 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 6.47 on page 136, the PC can 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.

6.17.4 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`, for example, copy and paste lines, and Menu `Execute`, for example, 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, for example, 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.

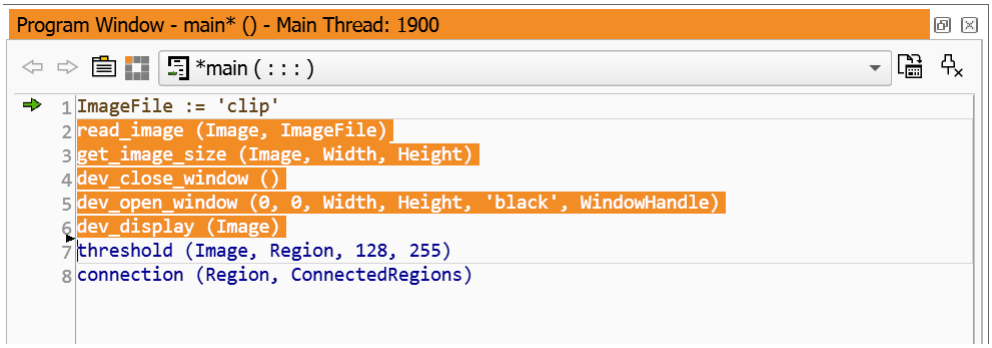
	Action	Shortcut		Description
	Open Operator Window	Ctrl+Shift+Space		Open the current operator or procedure call in the operator window for editing.
	Help			If the line under the mouse cursor contains an operator call, the corresponding page will be opened in the online help window. This is a shortcut 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 and 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 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. 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 59).
	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 125).
	Set/Clear Bookmark	Ctrl+F11		See Set/Clear Bookmark (page 54).
	Set/Clear Breakpoint	F10		See Set/Clear Breakpoint (page 56).
	Activate/Deactivate Breakpoint	Shift+F10		See Activate/Deactivate Breakpoint (page 56).
	Set/Clear Breakpoint on Variable			Toggle breakpoint on variable under mouse cursor. See section 6.21.1 on page 162.

	Action	Shortcut	Description
	Activate/Deactivate Breakpoint on Variable		Toggle activation of breakpoint on variable under mouse cursor.
	Manage Breakpoints	Ctrl+Shift+O,F10 or Ctrl+Shift+O,Ctrl+Shift+F10	See Manage Breakpoints (page 154).
	Add Watch		Add the variable under the mouse cursor to the user tab of the variable window.
	Print...	Ctrl+P	See Print... (page 111).

6.17.5 Creating Procedures

Procedures can 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 59) 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.

6.17.5.1 Setting Up the General Settings of a Procedure

See also: Procedures ▷ Create New Procedure (page 59) / Edit Procedure Interface (page 59).

Name The procedure name must start with a letter and may consist of alphanumeric characters and underscores. If you enter an invalid name, such as 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 46).

Password You can optionally protect procedures by a password. Protected procedures can 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 47 for more information.

Type This check box determines the procedure type (see section 5.1 on page 43). 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 can be specified explicitly (.hdvp or .dvp; see section 5.2 on page 44).

External and library procedures can be reused in other HDevelop programs. You can change the procedure type at any time.

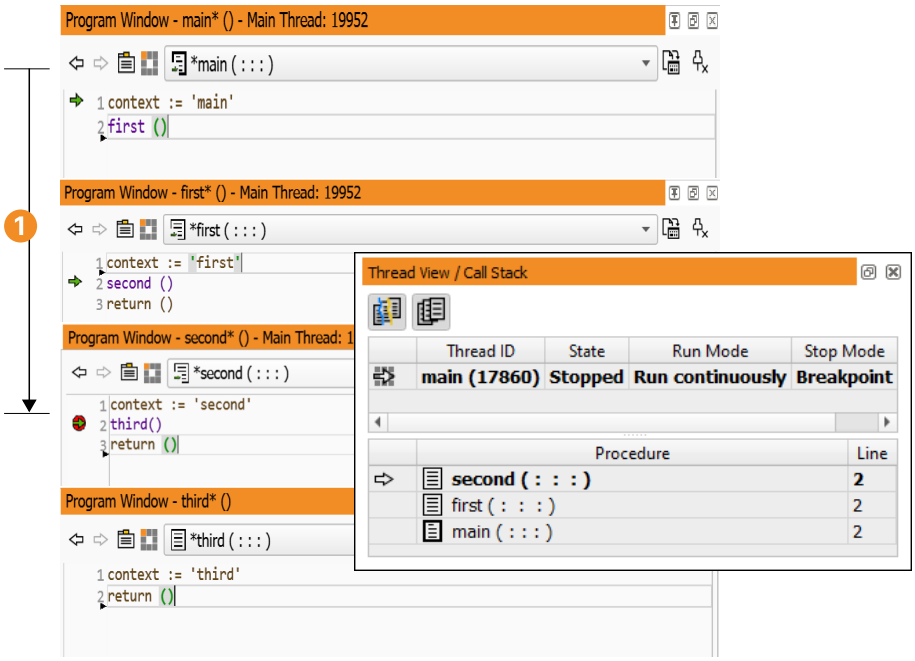


Figure 6.47: PC and call stack.

1 Execute by pressing F5

`Directory` For external procedures a target directory has to be specified.

The first directory specified in the procedure preferences (see section 6.16.5 on page 115) 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.

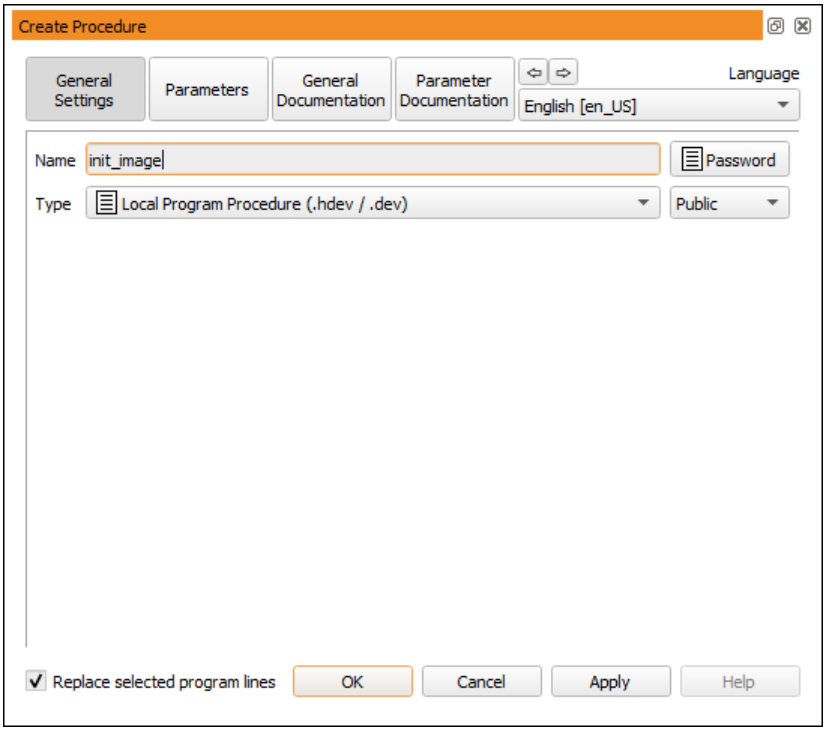


Figure 6.48: Dialog for creating a new procedure.

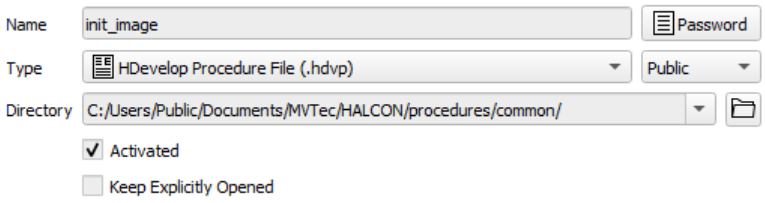


Figure 6.49: Settings for an external procedure.

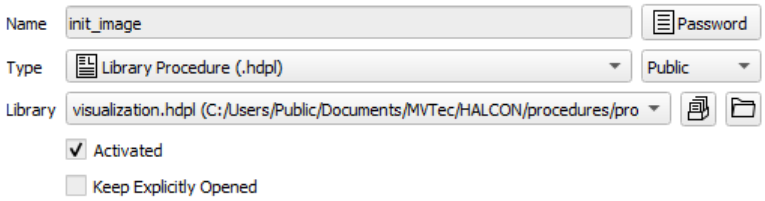


Figure 6.50: Settings for a library procedure.

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 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 45. 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 6.16.6 on page 116).

`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 6.16.5 on page 115).

The addition can either be permanent or for the current session only. This is further explained in section 5.4 on page 45.

6.17.5.2 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. They can 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 can 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.

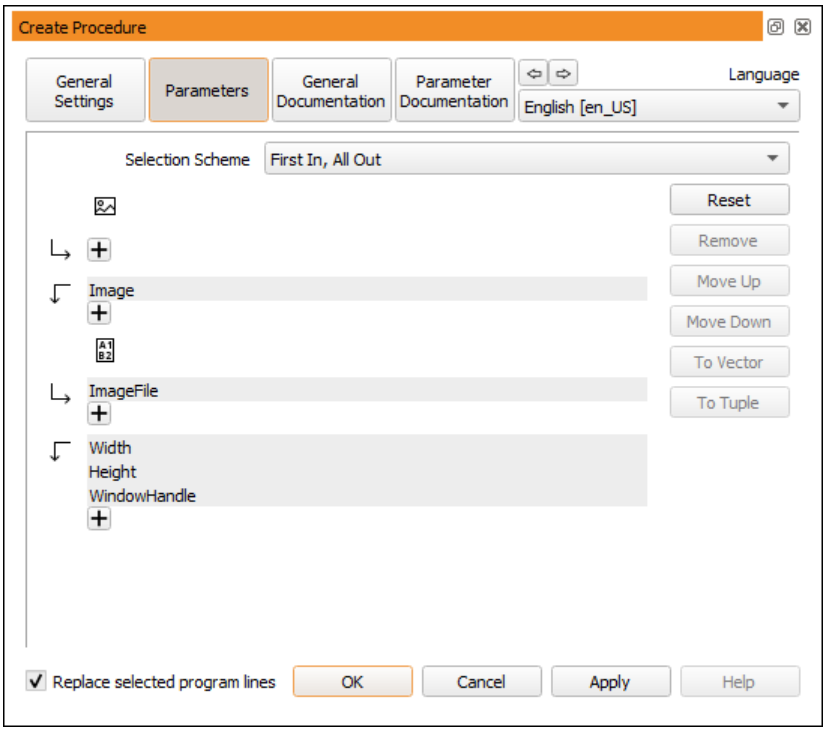


Figure 6.51: Procedure parameters.

`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.

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, and 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, for example, 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 8.6 on page 270 for a detailed description. Vector variables can 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. 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, for example, `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 can 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 can be decreased with this button (label changes to `Dim -`).

6.17.5.3 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 6.52 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 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.

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 5.6 on page 47).

`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.

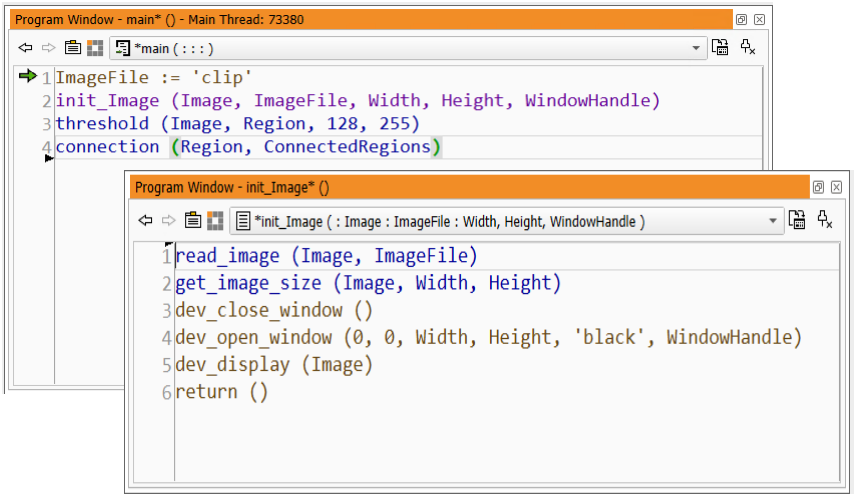


Figure 6.52: Resulting new procedure.

6.17.6 Editing Procedures

The combo box on top of the program window displays the name of the current procedure. You can search for specific procedures, and 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, followed by all referenced procedures and library procedures. The procedure groups are sorted alphabetically.

Procedures can be protected with a password. Those procedures 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 47.

To view and modify the interface of the current procedure

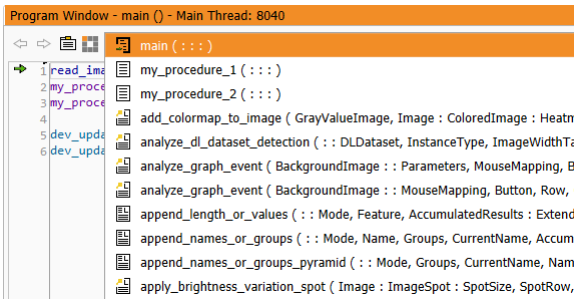


Figure 6.53: Searching procedures.

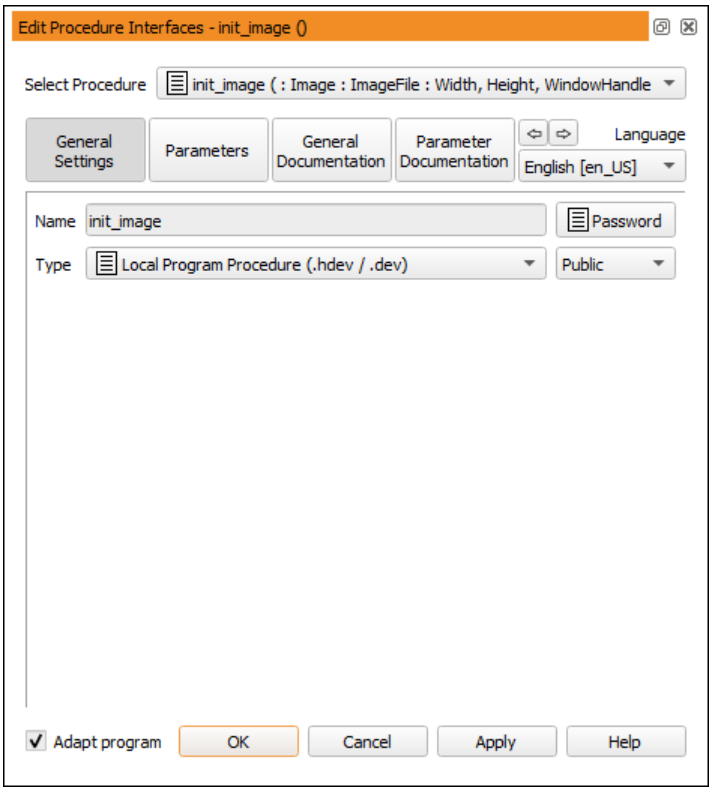


Figure 6.54: Editing a procedure.

- select `Procedures` ▷ `Edit Procedure Interface` (page 59)
- 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 6.17.5.1 on page 135
- `Parameters`: section 6.17.5.2 on page 137

The remaining buttons provide access to the documentation of the current procedure. This is described in section 6.17.8.

You can step through the individual tab cards of the dialog using the arrow buttons at the bottom of the dialog.

6.17.7 Side Effects of Procedure Changes

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.

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:

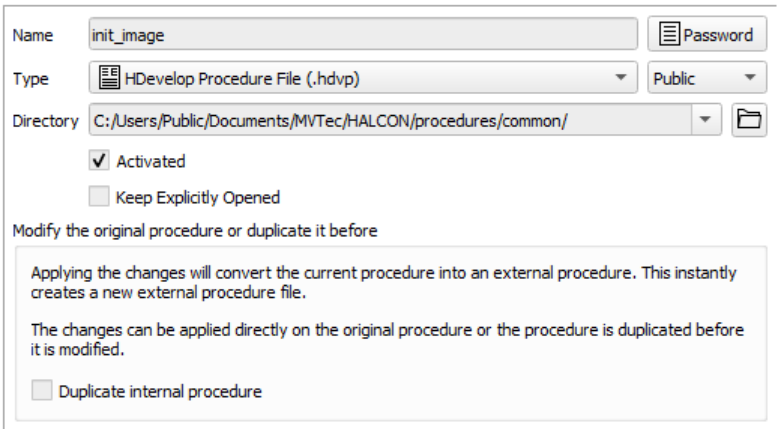


Figure 6.55: Procedure type changed from local to external.

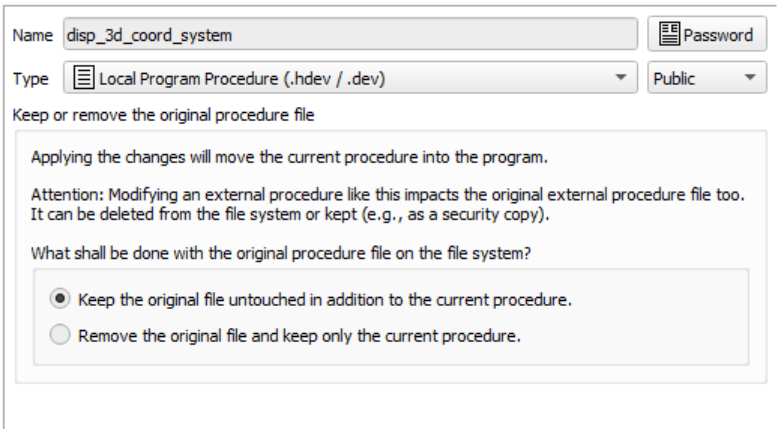


Figure 6.56: Procedure type changed from external to local.

- `Keep the original file untouched in addition to the current procedure.`
- `Remove the original file and keep only the current procedure.`

6.17.8 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 can 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`.

6.17.8.1 `General Documentation`

Procedures can be grouped by `Group` and `Chapters`.

`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

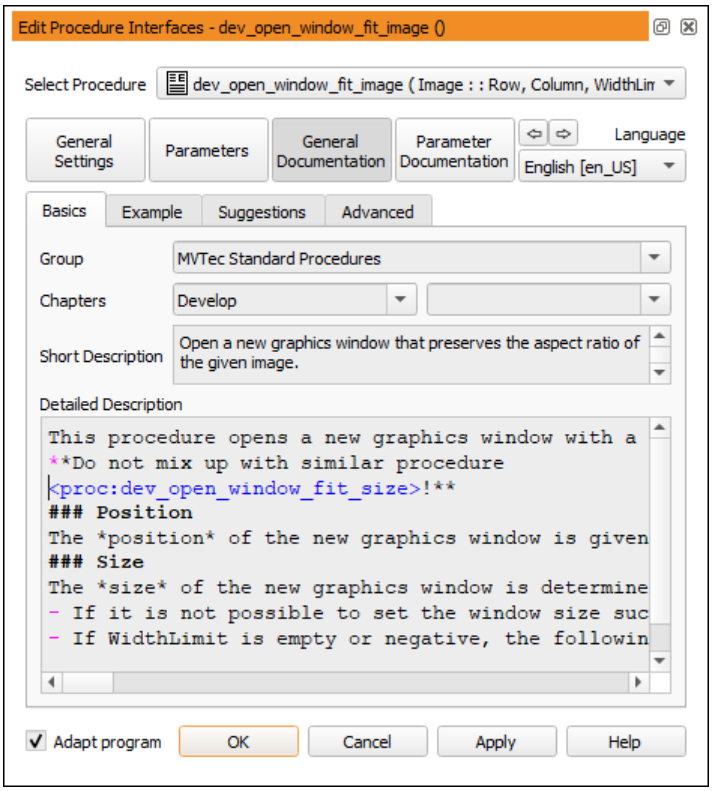


Figure 6.57: Editing the general documentation of a procedure.

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.

`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 6.17.8.1.

`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.

You can specify suggested successors, predecessors, and alternatives to the current procedure.

`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 6.17.8.1).

`Complexity` Notes about intricate details about the procedure usage. Supports Markdown syntax (see section 6.17.8.1).

`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 6.17.8.1).

`References` with recommended reading about certain aspects of the procedure. Supports Markdown syntax (see section 6.17.8.1).

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 (that is, 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)`
for example, `[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>` (for example, `<proc:train_dl_model>`)
- Operators: `<op:operator_name>` (for example, `<op:train_dl_model_batch>`)
- Table of contents: `<toc:chapter_name>` (for example, `<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.

6.17.8.2 `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.

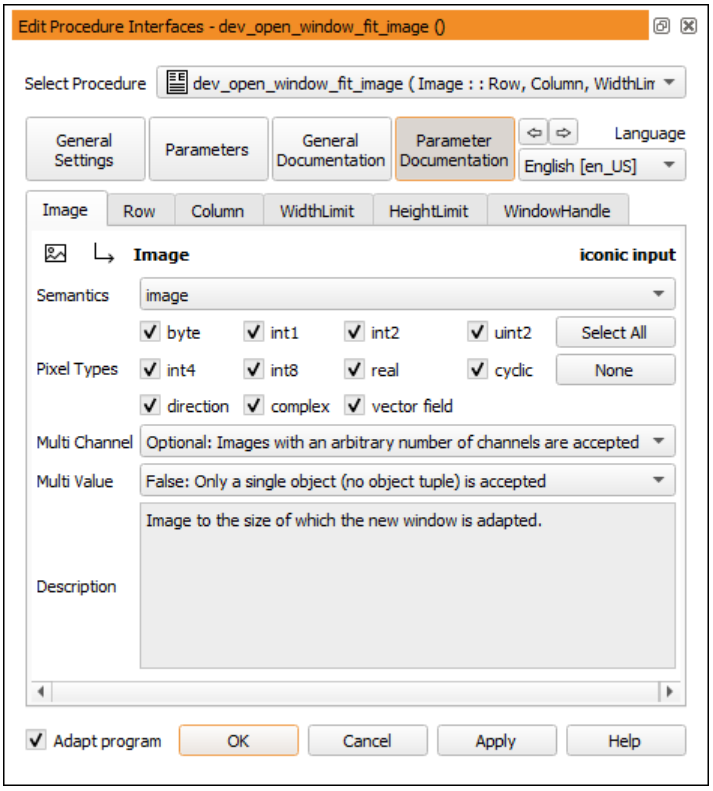


Figure 6.58: Editing the iconic parameter documentation of a procedure.

Control Parameter Documentation

Field	Meaning
`Semantics`	Specifies the semantic type of the parameter. For some semantic types, additional subtypes can 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.

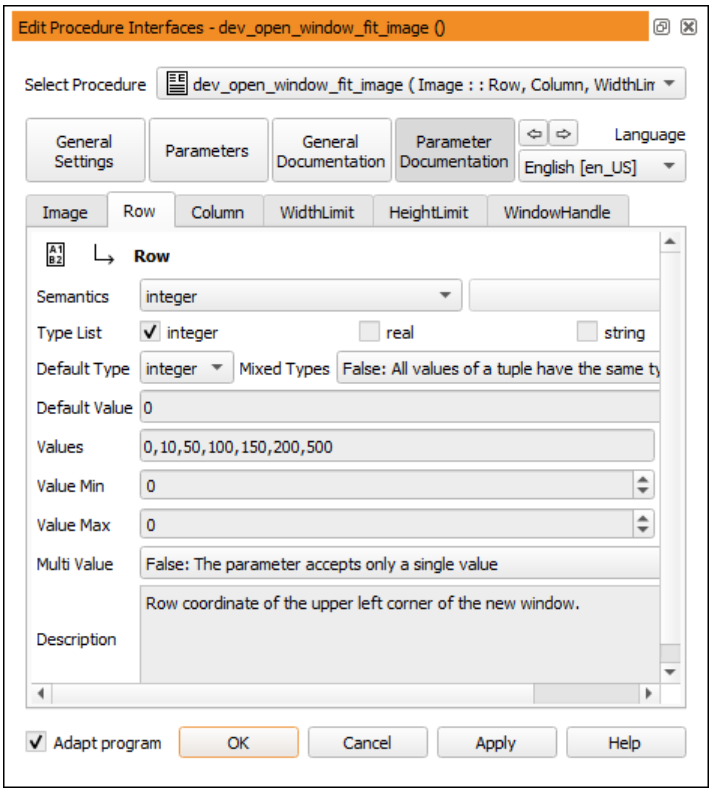


Figure 6.59: Editing the control parameter documentation of a procedure.

6.17.9 Protecting a Procedure

The concept of protected procedures is explained in section 5.6 on page 47. The status of a procedure can be changed via the procedure interface dialog (page 135). To manage the status of multiple procedures at once, click Menu `Procedures` ▷ `Manage Procedures` and select the tab card `Manage Passwords` (page 119).

If you want to protect a procedure with a password:

- 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 6.60). 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 can optionally lock the entire program, and protect all local procedures with the same password. The same mechanism works for library procedures: You can either protect the library procedures individually, or protect the entire library at once (see section 6.16.7 on page 118). 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.

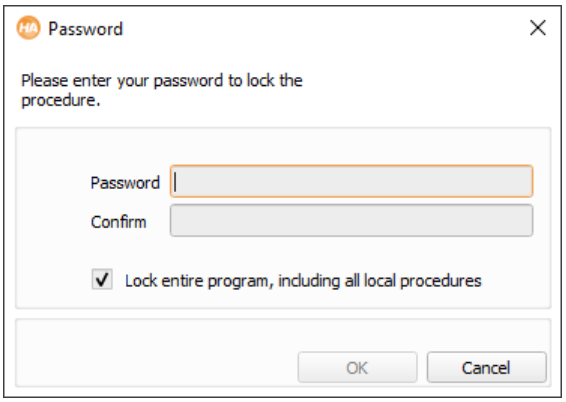


Figure 6.60: Entering a password to protect a procedure. The option “Lock entire...” is only available for the main() procedure.

When HDevelop is started again, the protected procedure is locked. When trying to edit the procedure, for example, by selecting it from the combo box in the program window, a corresponding message is displayed in the program window see figure 6.61). Additionally, a password button is displayed in the program window. The procedure remains unlocked for this session, that is, 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, it must be unlocked. Use the procedure interface dialog (page 135) to change the password or remove the password entirely.

`Lock` Lock the protected procedure and its body cannot be accessed in the current session without supplying the password again.

`Remove` Removes the password. When the procedure is saved, it is no longer protected.

`New password` The password window is displayed and a new password can be assigned.

`Cancel` The operation is canceled without altering the status.

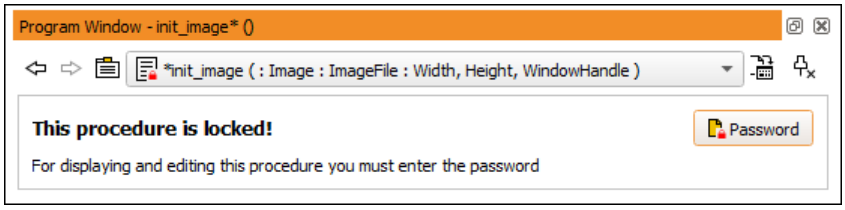


Figure 6.61: A locked procedure.

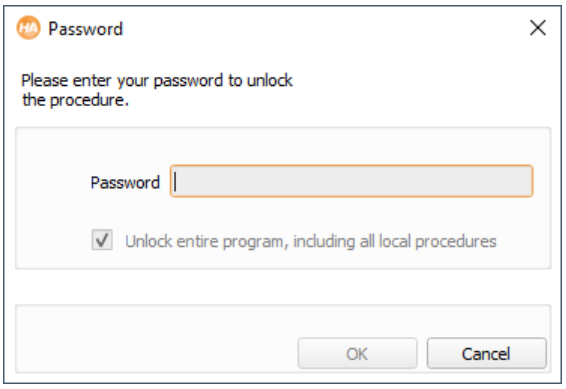


Figure 6.62: Changing the status of a protected procedure..

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!
The option “Unlock entire...” is only available if local procedures have been locked at once.
Note that, 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.

6.17.10 Profiler

The built-in profiler analyzes the runtime behavior of HDevelop programs. You can use the profiling data to evaluate the overall program execution, to optimize its performance, and to find bottlenecks.

The profiler measures the processing times in one of two modes:

- In the **operator time** mode, the profiler counts operator and procedure calls and measures the processing times of operator calls.

  The operator time is the appropriate measure if you plan to export your HDevelop program to a programming language.

- In the **execution time** mode, the profiler measures the operator time and additionally the overhead of each operator call inside HDevelop.

  The execution time is the appropriate measure if you run the program inside HDevelop or HDevEngine.

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 126).

To activate the profiler, click `Execute` ▷ `Activate Profiler` (page 56).
When the profiler is activated, each program execution collects profiling data.

To reset the profiler data, click `Execute` ▷ `Reset Profiler` (page 56),

To deactivate the profiler, click `Execute` ▷ `Deactivate Profiler` (page 56).

The profiling results are displayed in the program window. In addition, a summarized view of the runtime statistics is available.

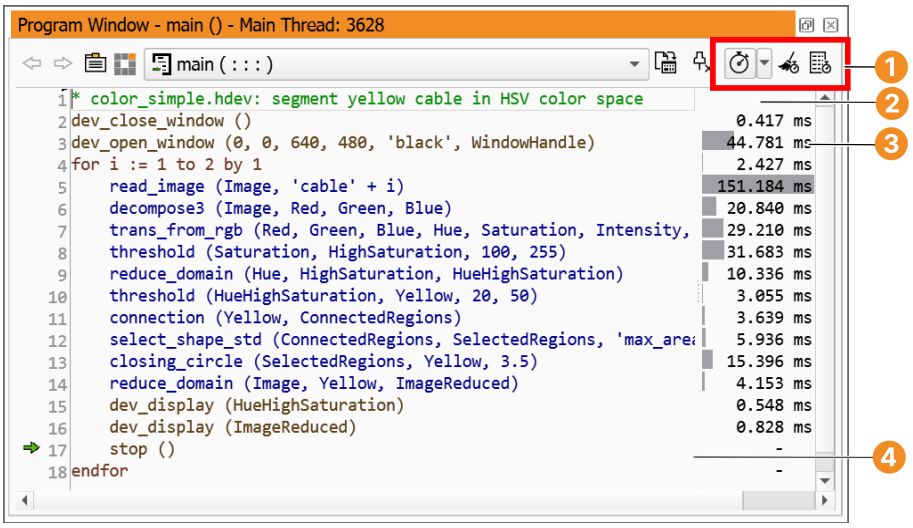


Figure 6.63: Profiling results of the first program execution.

1. If the profiler is enabled, the tool bar of the program window contains some additional profiler-related buttons
2. The profiling data is displayed in a separate column of the program window
3. 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
4. The display area can be resized by dragging its left edge

6.17.10.1 Profiler Display

To illustrate the way the profiler works, the HDevelop program `solution_guide/basics/color_simple.hdev` is loaded and executed (see figure 6.63).

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 6.17.10.3 on page 151.

`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, for example, in a loop. The average processing time is more meaningful if the program is reset and run multiple times.

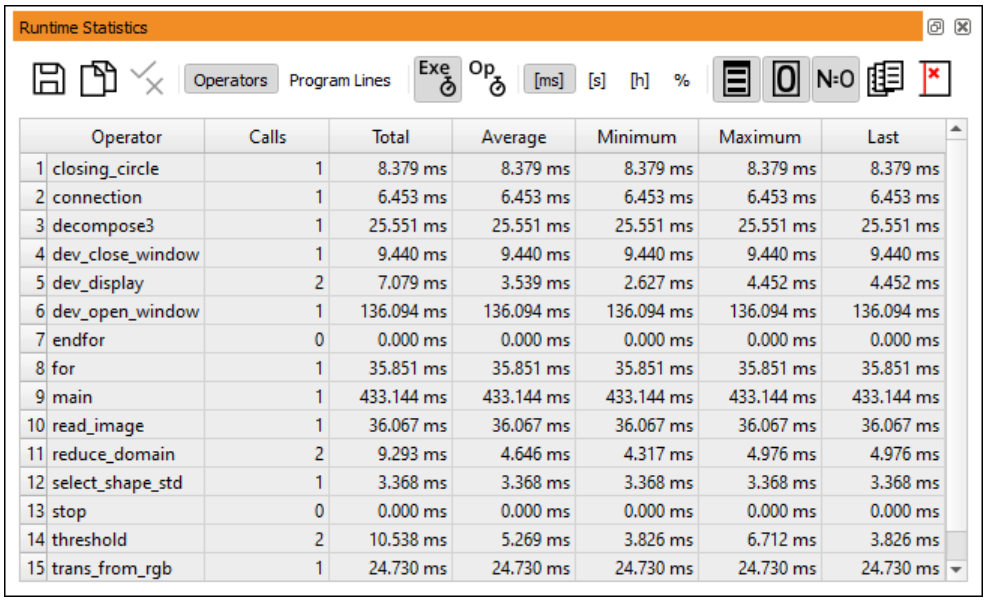
`Minimum Execution Time` Displays the minimum processing time of each program line. This value is only meaningful if the corresponding program line has been executed more than once, for example, in a loop.

`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, for example, in a loop.

`Last Execution Time` Displays the processing time of the last execution of each program line.

The processing time is displayed in one of two modes as explained in section 6.17.10 on page 148, depending on the setting in the context menu:

`Operator Time` Measures operator and procedure calls.



	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 6.64: Runtime statistics.

`Execution Time` Measures calls including the additional overhead inside HDevelop.

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, for example, 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 6.17.10.2.

`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 6.17.10.2).

6.17.10.2 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, that is, multiple calls of the same operator/procedure are summarized.

Note that the sum of all calls that comprise the program can be less than the measured duration of the `main` program as the latter also includes the overhead of the operating system.

`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 6.17.10.3). 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, 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 6.17.10.3). If this button is on, deactivated profiler lines will be displayed in light gray so that they can be activated again.

6.17.10.3 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 6.65).
- 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 6.66.

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 6.67. 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 6.68 on page 153. 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.

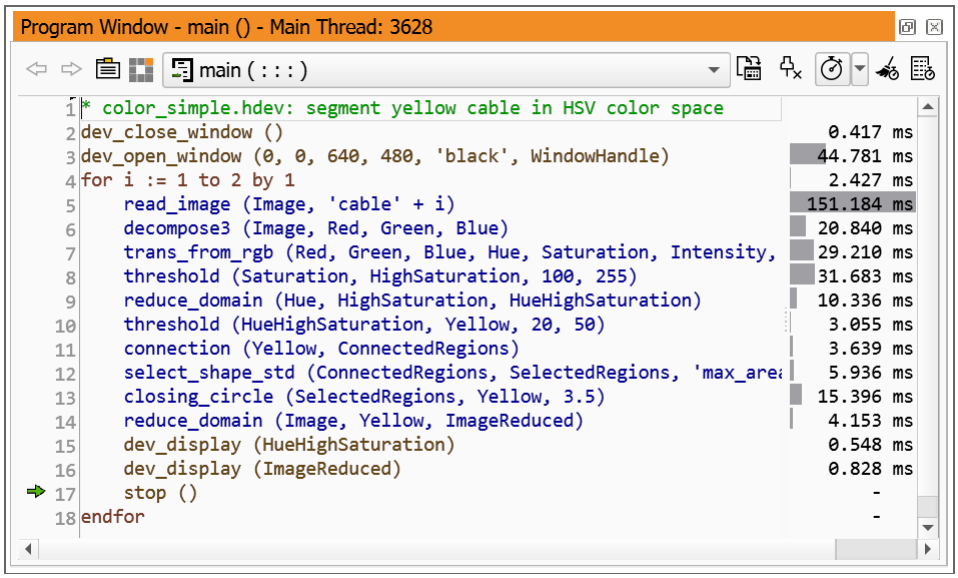


Figure 6.65: Selecting program lines in the profiler.

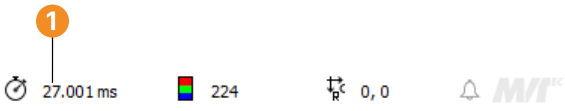


Figure 6.66: Profiler values.

1 Selected values are summed up in the status bar

6.18 Quick Navigation Window

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

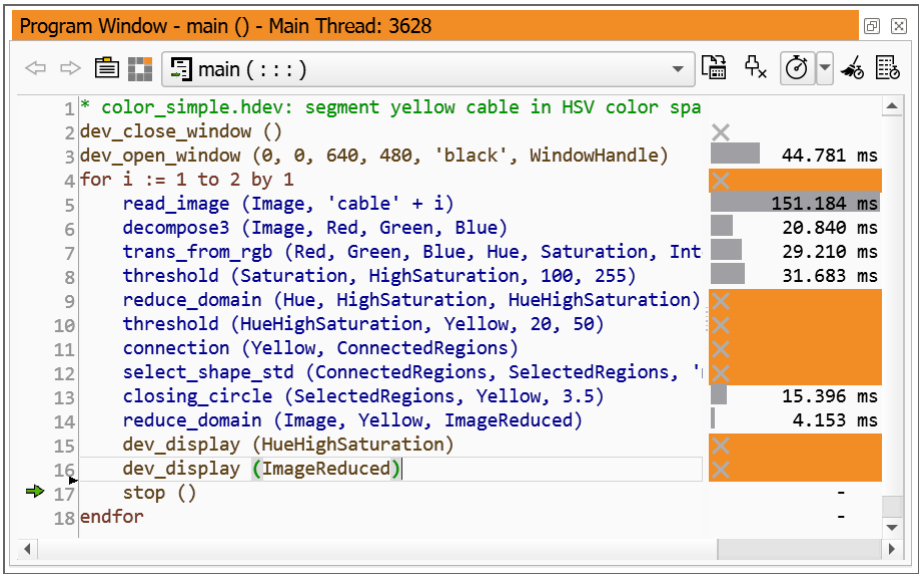
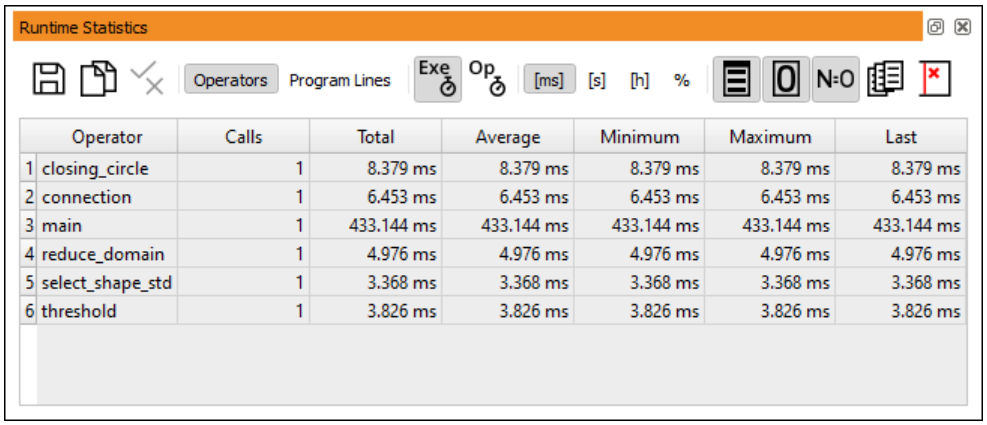


Figure 6.67: Activate selected program lines in the profiler.



	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 6.68: Runtime statistics of selected program lines.

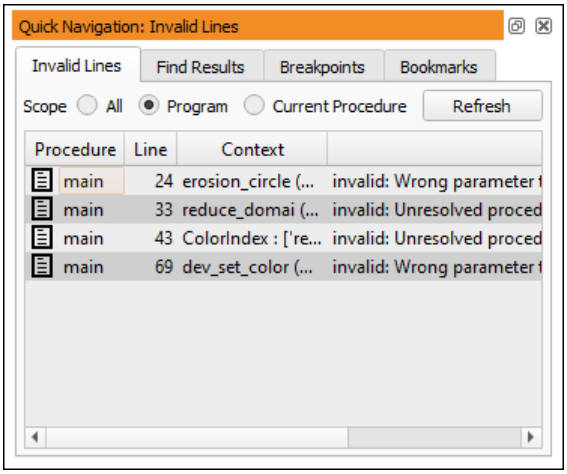


Figure 6.69: Managing program lines in the quick navigation window.

displayed in a program window tab, the corresponding tab is activated. Otherwise, the current view switches to the selected procedure.

It is possible to activate, deactivate, copy, cut, or delete one or more of the selected program lines directly from the quick navigation.

The “common properties” are available as tab cards:

6.18.1 Invalid Lines

This 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. Examples of invalid lines are unresolved procedure calls, operator or procedure calls with a wrong number or type of parameters, or syntax errors.

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 54)

- `Cut` (page 54)
- `Delete` (page 54) (also by pressing Del)
- `Activate` (page 54)
- `Deactivate` (page 54)

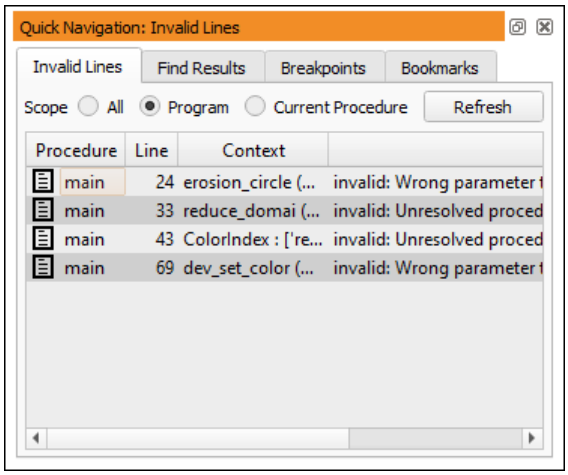


Figure 6.70: Managing invalid program lines in the quick navigation window.

6.18.2 Find Results

This tab card provides a listing of program lines that match a previous "find all" operation. Click the button `Open Find/Replace Dialog` to perform a "find all" operation. More info (page 72)

6.18.3 Breakpoints

This tab card lists all variables or program lines marked with a breakpoint (see figure 6.72). 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 can be activated or deactivated by clicking the check boxes.

See section 6.21.1 on page 162 for information about breakpoints on variables and section 6.17 on page 127 for line-based breakpoints.

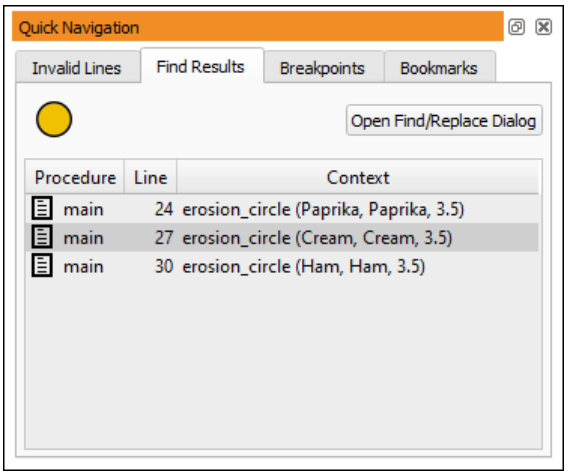


Figure 6.71: Search results of the “find all” operation.

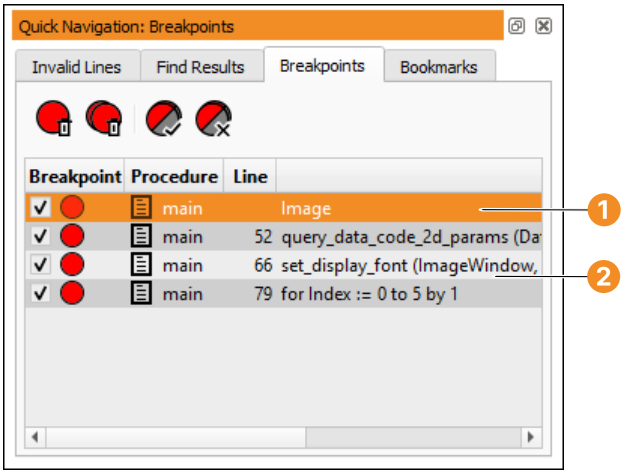


Figure 6.72: Managing breakpoints in the Quick Navigation window.

1 Breakpoint on variable

2 Line-based breakpoint

6.18.4 Bookmarks

This 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.

6.19 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

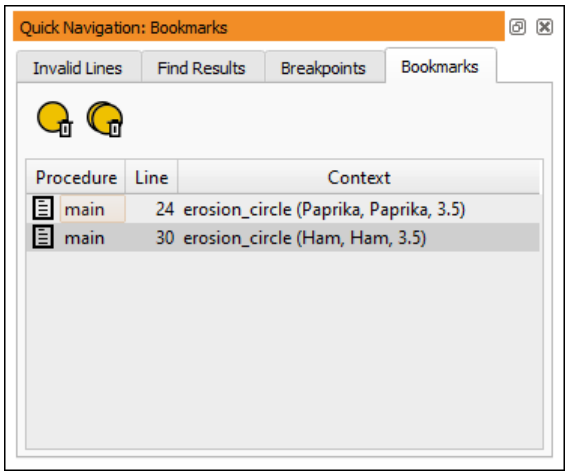


Figure 6.73: Managing bookmarks in the quick navigation window.

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 6.74 (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.

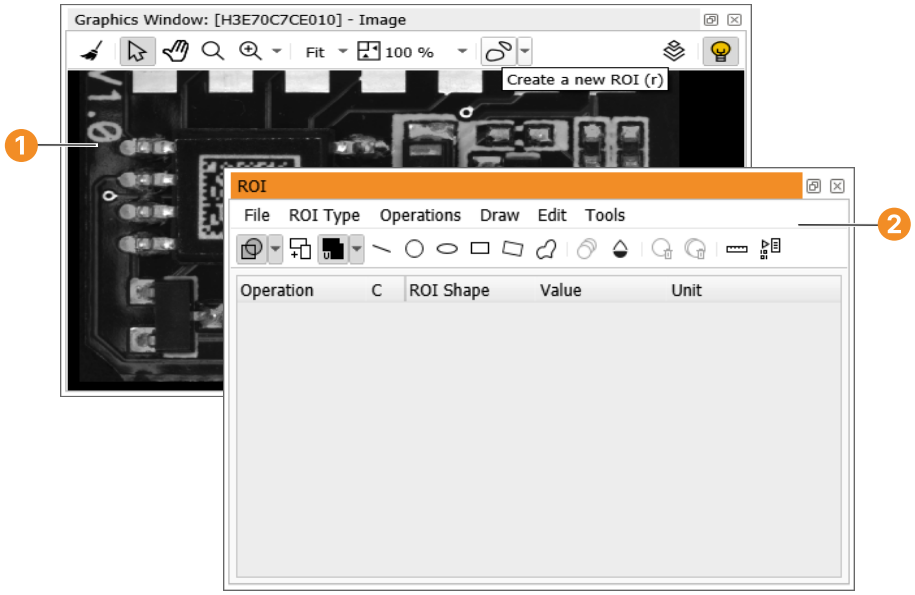


Figure 6.74: Base image for the creation of an ROI.

(1) Example image

(2) Opened dialog

Generating an ROI Interactively

- Click the tool bar icon to create a new figure.
  This will open the dialog shown in figure 6.74 (2) which will act as a tool box for your new figure.

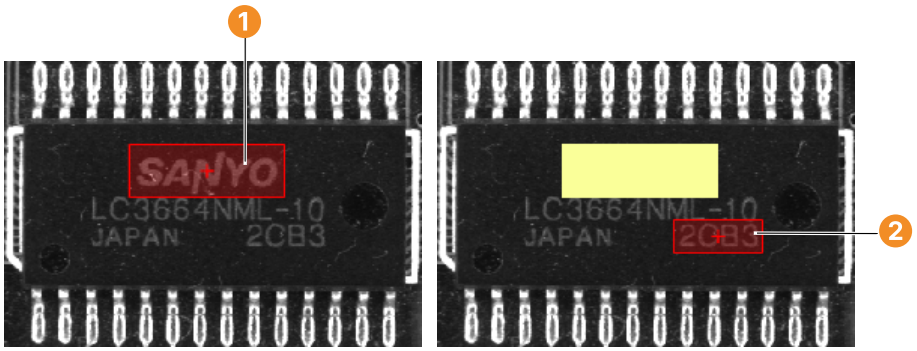


Figure 6.75: Drawing ROI rectangles.

1 Rectangle 1

2 Rectangle 2

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 6.75 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 6.75 2).

- Click to generate program code in the current program.

```
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.

ROI Opacity

You can adapt the opacity of the ROIs by dragging the slider.

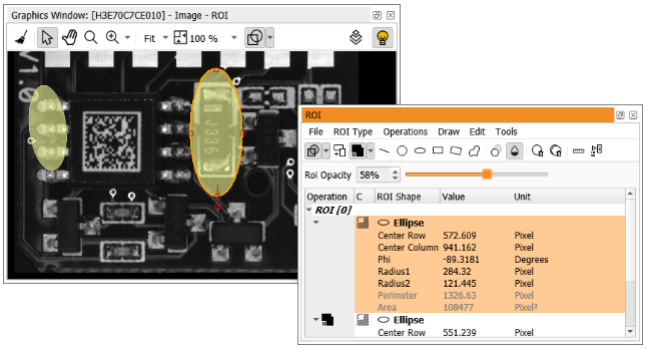


Figure 6.76: ROI Opacity.

Editing the Figure

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`.

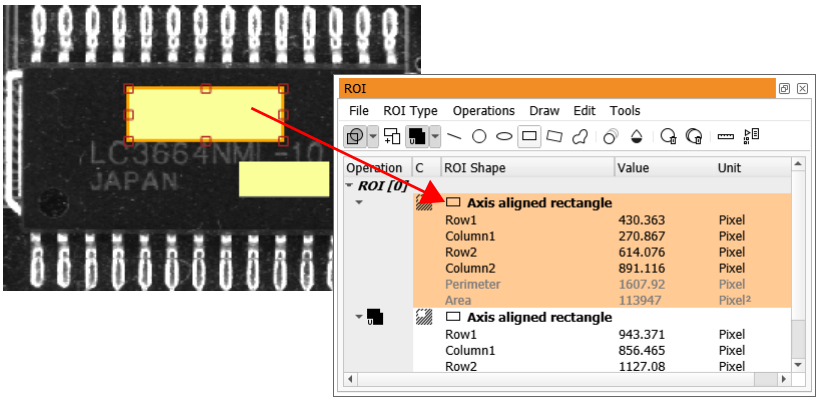


Figure 6.77: An ROI made of two rectangles.

You can make changes to a figure by clicking its individual constituent parts to select them. In figure 6.77, 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 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.

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 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).

For each element a masking mode can be toggled independently by clicking the corresponding icon in the tool box:

The default masking mode selects the *inside* of the element, for example, the inner area of a circle.

The complement masking mode selects the *outside* of the element, for example, 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).

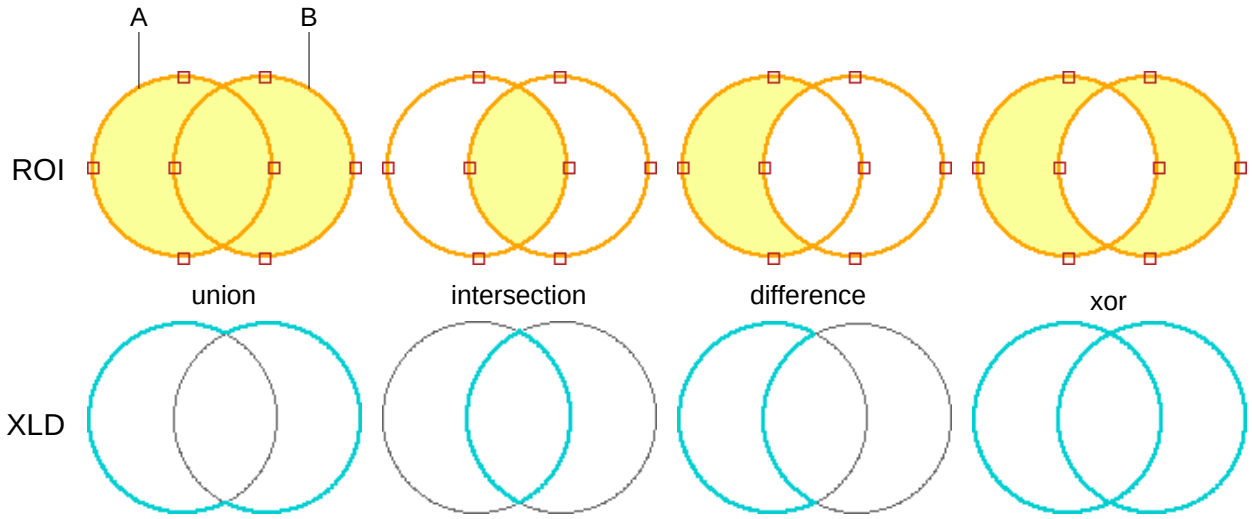


Figure 6.78: Set Operations.

(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	-

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 6.78 for an illustration):

- (union of A and B)
- (intersection of A and B)
- (difference of A minus B)
- (xor, 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, that is a known dimension has to be specified.

See figure 6.79 for an example. The length of the line is known, so it can be used for the calibration.

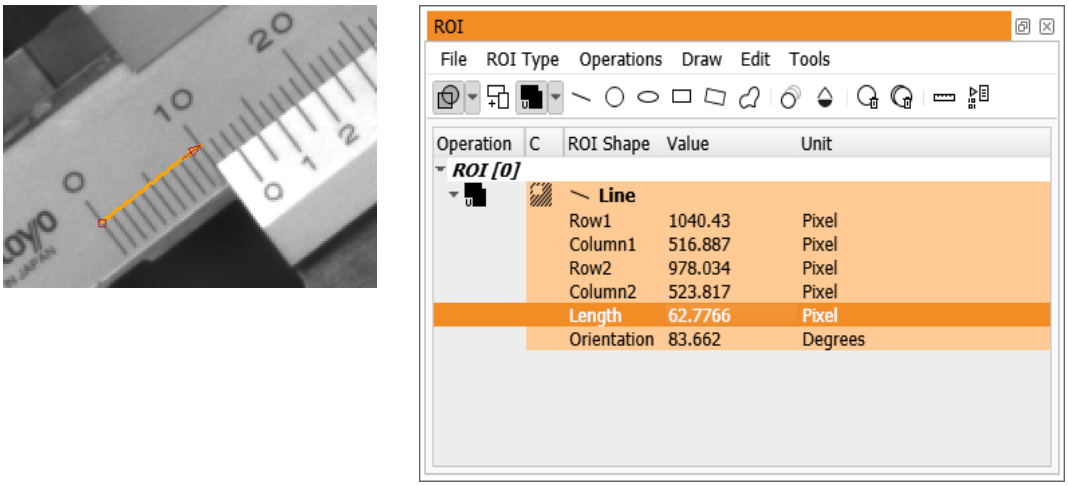


Figure 6.79: Performing 2D calibration.

- 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 the *Length* value in the data block of the line, and specify the known dimension, for example, 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.

6.20 Thread View / Call Stack

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 8.11.4 on page 287).

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 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 a procedure call that belongs to a locked procedure (see section 5.6 on page 47), you can only see the procedure body if you enter the correct password into the program window.

6.21 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 can possess a value or can 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 can 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.

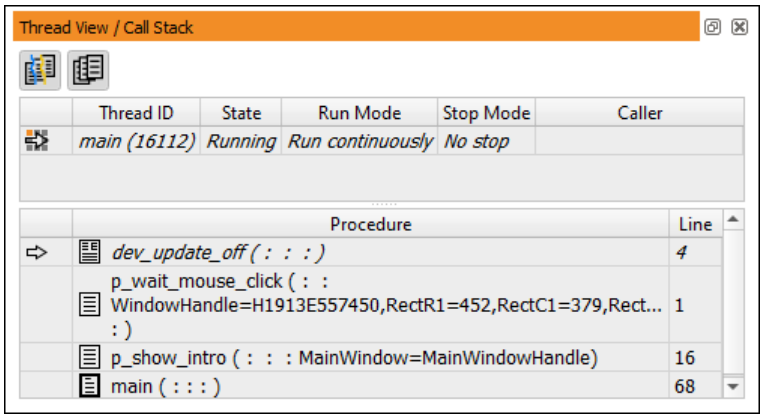


Figure 6.80: `Thread View / Call Stack`.

Similarly, this is the case for iconic variables, which can contain regions, images, or XLDs. You can 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). **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.

All computed variables are displayed showing their iconic or control values (unless the automatic update has been switched off, see section 6.16.14 on page 125). In case of a tuple result that is too long, the tuple presentation

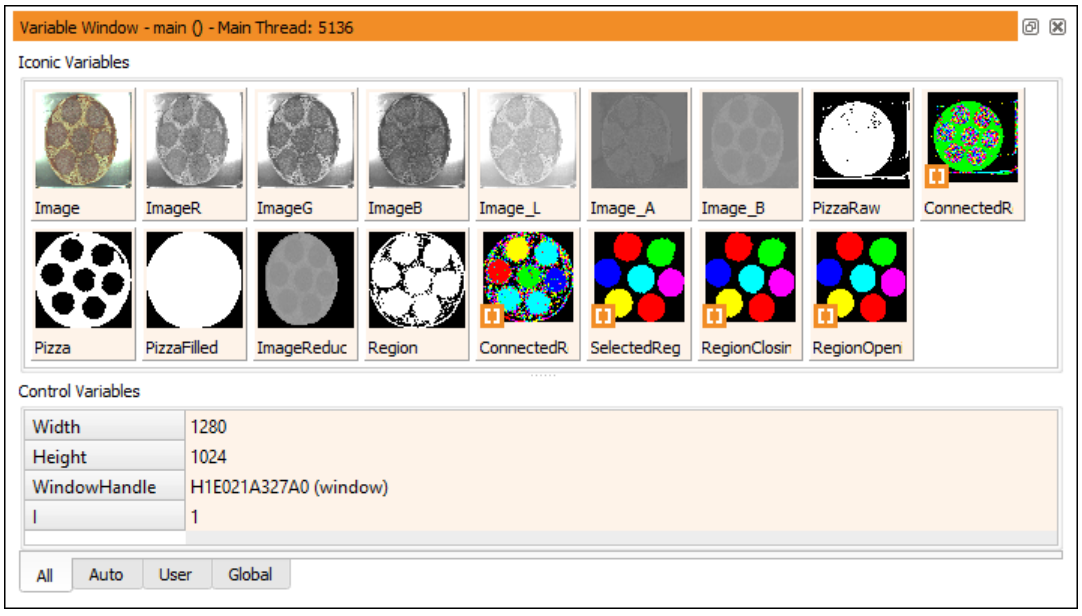


Figure 6.81: Variable window with instantiated iconic and control variables.

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.

6.21.1 Breakpoints on Variables

In addition to line-based breakpoints (see section 6.17 on page 127), breakpoints can also be set on variables. Right-click 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 8.5.10 on page 265):

- 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 (for example, 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
```

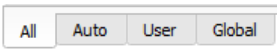
Using `clear_obj` does not trigger a variable breakpoint. Use `dev_clear_obj` instead.

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, for example, HDevelop displays both the old and the new value of a tuple.

6.21.2 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.



`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. If the tab `User` is active, variables can 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 an entry to jump to the corresponding location in the program window.

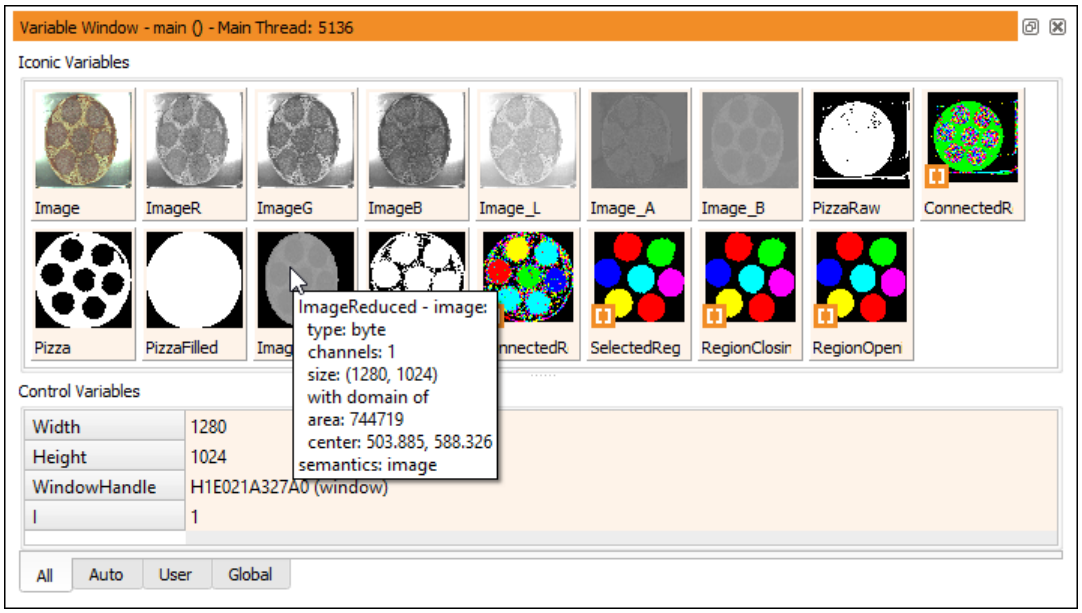


Figure 6.82: Displaying information about an iconic variable with a reduced domain.

6.21.3 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 6.82 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

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 58): 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 is 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 6.8 on page 73), it is 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 6.82 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 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, for example, 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 6.21.1 on page 162).

`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 72) 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 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 an entry to jump to the corresponding location in the program window.

`Sort by Name`**:** Sort all variables in alphabetical order.

`Sort by Occurrence`**:** Sort the variables in the same order as they are defined in the program.

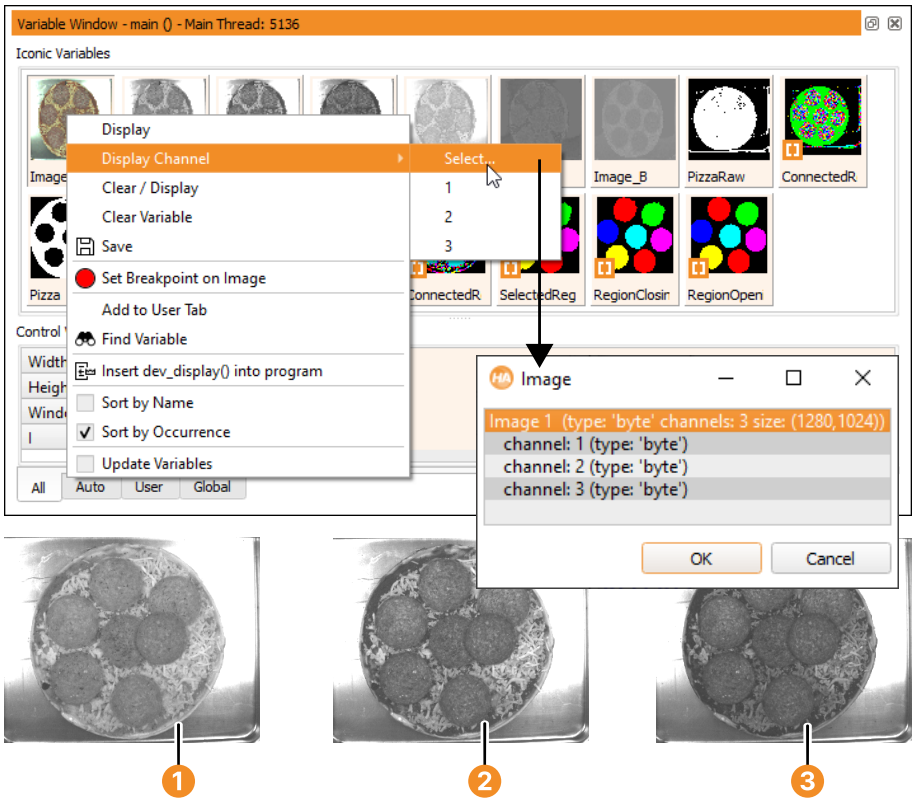


Figure 6.83: Interactive channel selection from an RGB image.

1 Channel 1: red

2 Channel 2: green

3 Channel 3: blue

`Update Variables`**:** Toggle whether variables will be updated during program execution. This is the same setting as in the runtime preferences (see section 6.16.14 on page 125).

`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 6.83 for an example.

Visualization of Iconic Variables

Usually, 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 can *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 can 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), that is 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 ∅.

6.21.4 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.

Context Menu

`Inspect`**:** Inspect the values of the selected control variables in an auxiliary window (see section 6.22). The display depends on the semantic type of the selected control variable:

- Tuples
- Vectors (page 169)
- Handles (page 170)
- Matrices (page 170)
- Poses (page 170)
- Image acquisition devices (page 171)
- Functions (page 171)
- 3D object models (page 176)

`Inspect as Handle` Force the generic handle inspect window for all types (see section 6.22.3 on page 170).

`Inspect as Tuple` Force the tuple inspect window for all types (see section 6.22.1).

`Plot as Function` Plot the tuple values (see section 6.22.7 on page 171). 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 6.21.1 on page 162).

`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 72) 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 an entry to jump to the corresponding location in the program window.

`Sort by Name`: Sort all variables in alphabetical order.

`Sort by Occurrence`: 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 section 6.16.14 on page 125).

`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`.

6.22 Variable Inspect

See also: `dev_inspect_ctrl`.

Double-clicking a control variable opens an inspection window next to the variable window. The type of the inspection window depends on the semantic type of the variable, for example, “Matrix Inspect” for matrices, “Handle Inspect” for handles, and so on.

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 the selected variables and select `Inspect`. For each inspected variable type, a new tab card is opened in the inspection window. Variables of the same semantic type are grouped in a single tab card.

6.22.1 Inspecting Tuples

Control variables containing tuples are displayed in a tabular format (see figure 6.84). This is especially useful for the inspection of tuples with a large number of values.

In addition to the tuple values, some statistical data can be displayed:

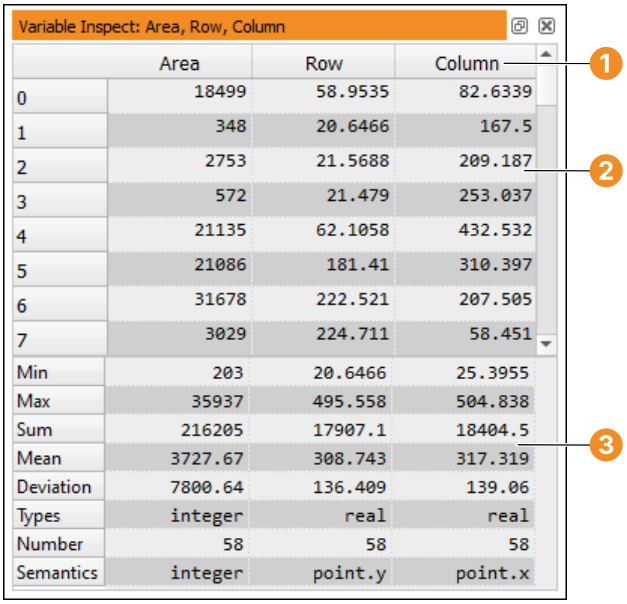


Figure 6.84: Control variable inspection.

1 Variable names

2 List of tuple values

3 Statistics - select from context menu

- 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 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 can be allowed, for example, matrix variables (see below) contain only real values.

You can also add new elements to a tuple: Instead of entering a single value, enter a list of values enclosed in square brackets (tuple notation).

For example:

```
Variable Inspect
  1
  2       -> [2,22,222]
  3
```

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
```

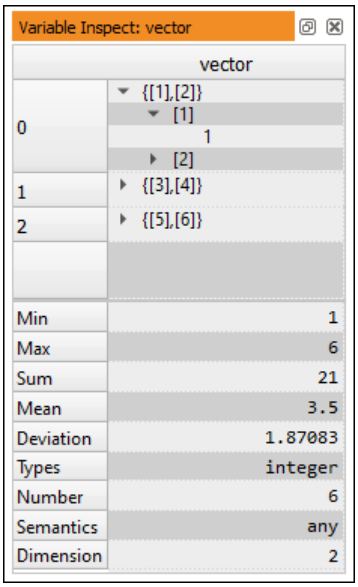


Figure 6.85: Vector variable inspection.

results in the following tuple:

```
1
3
```

6.22.2 Inspecting Vectors

Vectors are container variables for tuples, iconic objects, or vectors. See section 8.6 on page 270 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 6.85 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 6.22.1 on page 168).

In addition to the tuple values, some statistical data can 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 the statistical data to be displayed by right-clicking the statistics table and selecting the corresponding entries.

6.22.3 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, they cannot be modified in the dialog. They are updated automatically if they change in the background.

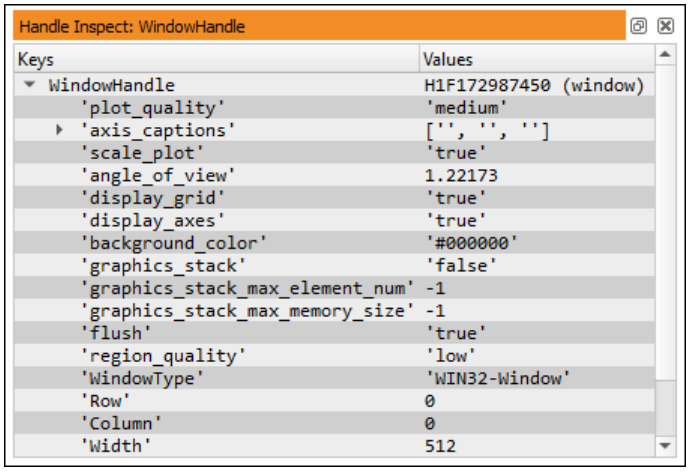


Figure 6.86: Inspection of a window handle variable.

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 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.

6.22.4 Inspecting Matrices

Control variables that reference a matrix are displayed in a tabular format as displayed in figure 6.87.

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).

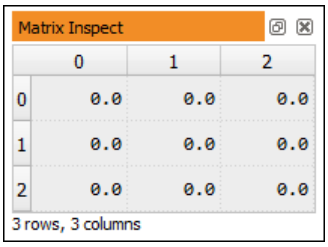


Figure 6.87: Inspection of a matrix control variable.

6.22.5 Inspecting Poses

Control variables that reference a pose are displayed in a tabular format as displayed in figure 6.88.

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).

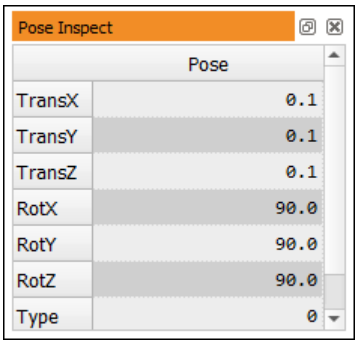


Figure 6.88: Inspection of a pose control variable.

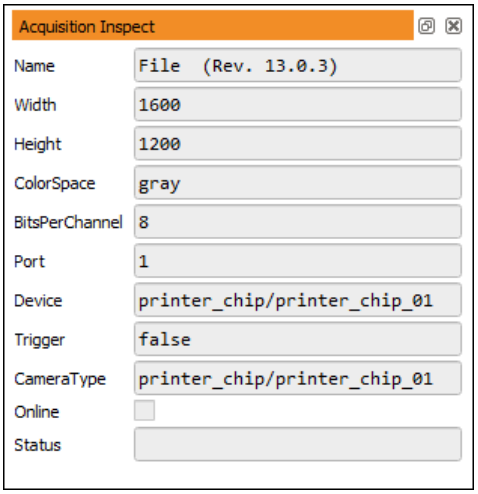


Figure 6.89: Inspecting an image acquisition device handle.

6.22.6 Inspecting Image Acquisition Device Handles

For an image acquisition device handle, a dialog with basic image acquisition device parameters is opened (see figure 6.89). 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.

6.22.7 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:

```
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`.

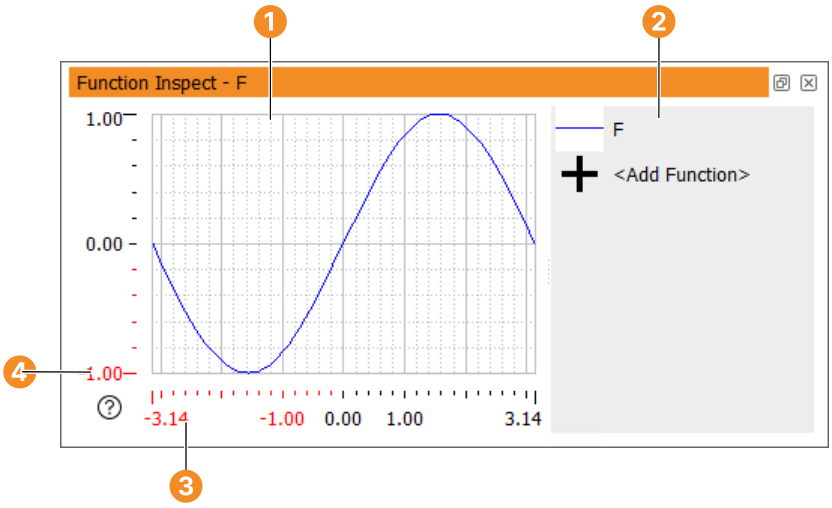


Figure 6.90: Inspecting a variable of semantic type `function_1d`.

1. Plot area
2. Plot list
3. x-axis
4. y-axis

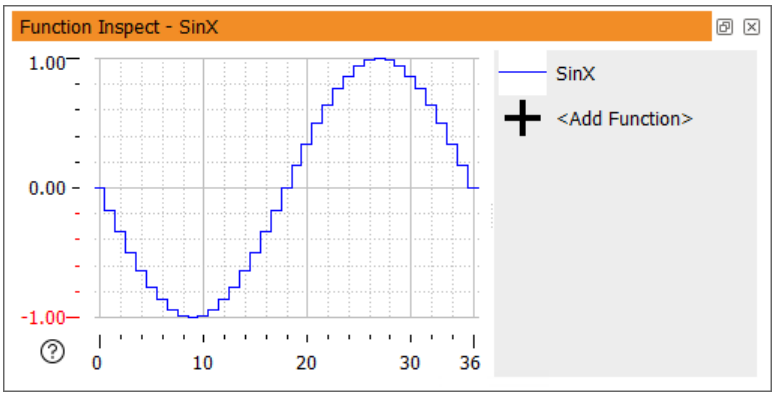


Figure 6.91: Inspecting a tuple with y values for equidistant x values.

Double-click `F` in the variable window to generate a function plot (see figure 6.90).

Right-click `SinX` in the variable window and select `Plot as Function` to generate a function plot of the y values (see figure 6.91). 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 can be adjusted with the mouse wheel over the plot area or one of the axes. The displayed part can 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)

	Action	Shortcut	Description
	`Reset Bounds`	W or Ctrl+Shift+W	Set width to default.

	Action	Shortcut	Description
	`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 6.92.

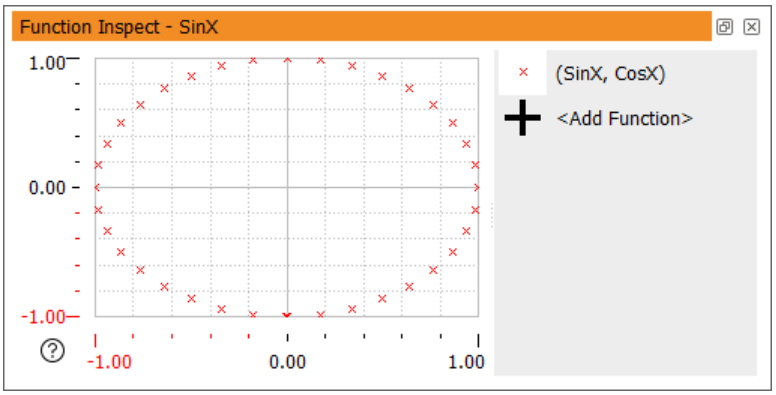


Figure 6.92: Example for a scatter plot using the variables `SinX` and `CosX`.

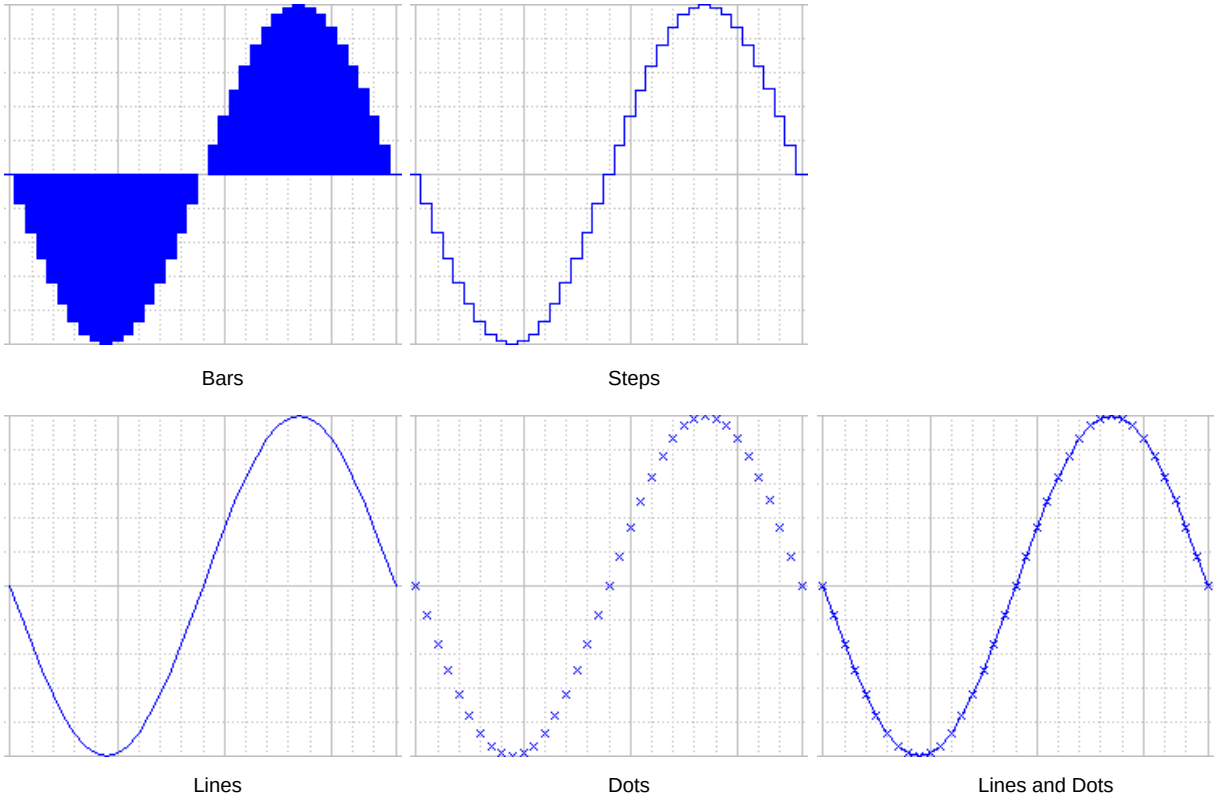


Figure 6.93: Plot styles.

```
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 6.93.

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.

6.22.8 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 6.94. See `get_object_model_3d_params` for a description of the displayed properties.

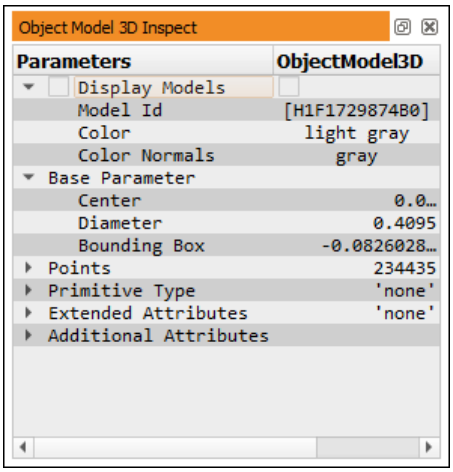


Figure 6.94: Parametric inspection of a 3D object model.

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 6.95.

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

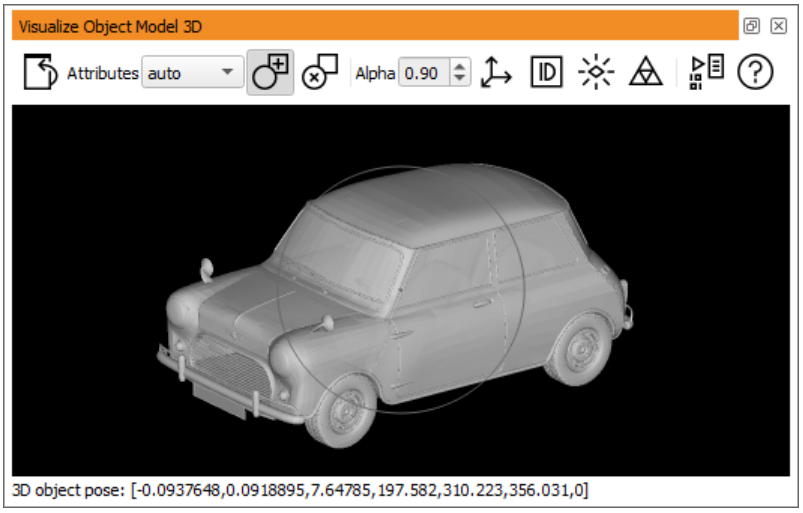


Figure 6.95: Visual inspection of a 3D object model.

corresponding slot and selecting a color name from the drop-down list. You can also enter a valid color name (page 321) 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.

6.23 Zoom Window

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 clicking it. 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.

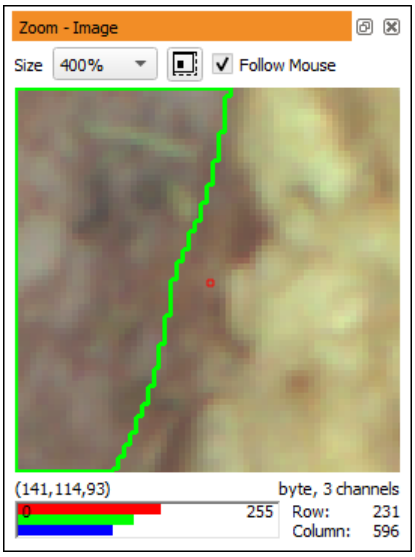


Figure 6.96: `Zoom`.

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.

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. When 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.

Common Features of all HDevelop Assistants

Because multiple assistants can be opened at the same time, assistants of the same type are numbered consecutively, for example, 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. If you want to keep the setup for later sessions, 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.

`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. 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, for example, 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, for example, the image format can be set. After establishing a connection to the selected 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`

`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`

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, that is, 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

When 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

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. 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 no range is defined, 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`

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`

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 179.

Menu `Acquisition`

`Connect` Connect / disconnect the selected image acquisition device. See section 7.1.2 on page 180.

`Snap` Acquire a single image. See section 7.1.2 on page 180.

`Live` Acquire images in live mode. See section 7.1.2 on page 180.

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 (for example, 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 195),
- generate code (page 196) 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 198).

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 when 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 188) a vision system, you extract information about it, for example, 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 194) 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, for example, 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 188) tab, you can choose the display parameters, like colors, as you prefer them. See also `Display Parameters` (page 194).

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 196).

Pose Estimation When 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 196).

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 (for example, in millimeters), as opposed to those of an image (in pixels). Example code is available from the `Code Generation` tab under `Sample Usage` (page 197).

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 191) 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

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.

Choosing the Task for Your Application If you want to calibrate all parameters, for example, 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 190).

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 188).

It is also possible to import parameters from a file. If you should decide to do this, just click the `Import Parameters` button.

When you have finished this last part of the `Setup` tab, proceed to the `Calibration` (page 188) 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 195) tab under `Calibration Status` (page 195) 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 Height` 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. 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 (page 190).

Optional parameters, which may be changed, are

- parameters concerning `Quality Issues` (page 191),
- `Display Parameters` (page 194), and
- `Calibration Plate Extraction Parameters` (page 194)

ATTENTION: Keep your camera setup (aperture, focus, pose) fixed when 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.

When the calibration images are available, you can push the `Calibrate` button and move on to the `Results` (page 195) 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 180). The assistant will then appear in a new window and support you with acquiring new calibration images.

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 Images

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.The image status gives information about the quality of each image. Details concerning quality can be found under `Quality`

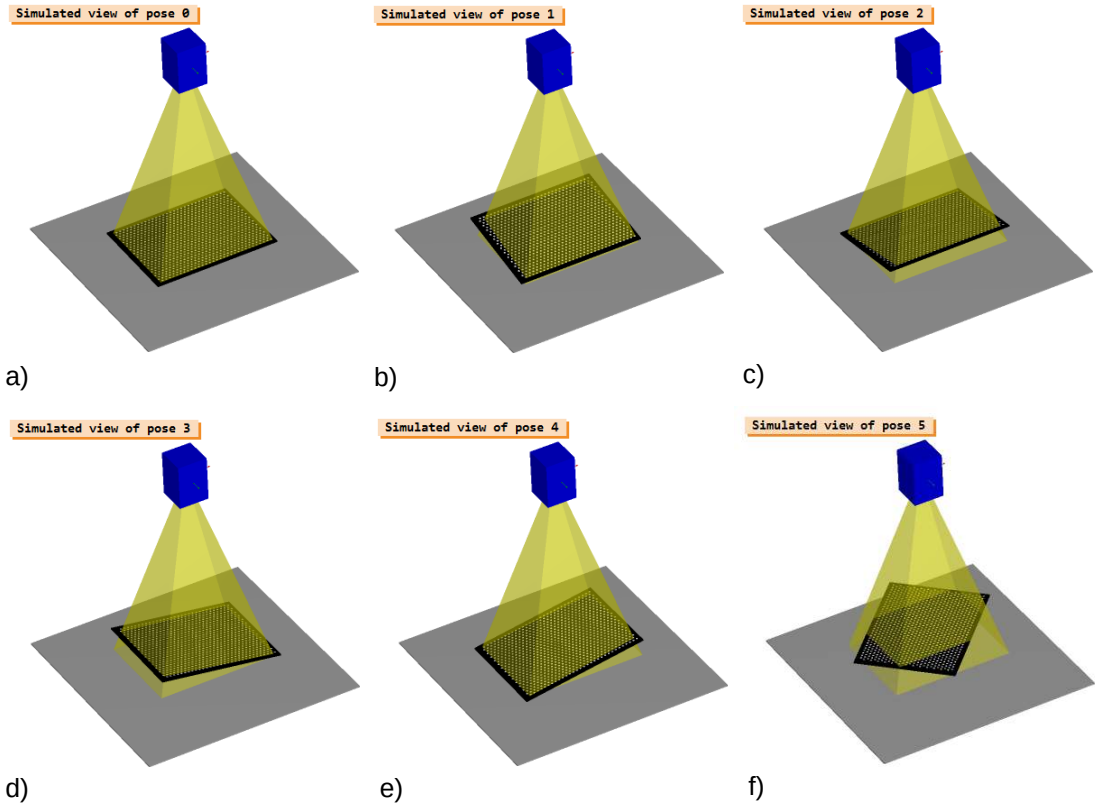


Figure 7.1: Acquisition of suitable calibration images using a calibration plate with hexagonally arranged marks. The plate is a) placed in the measurement plane, b) - e) tilted in different directions, and c) placed parallel to the measurement plane (with a rotation around the z axis).

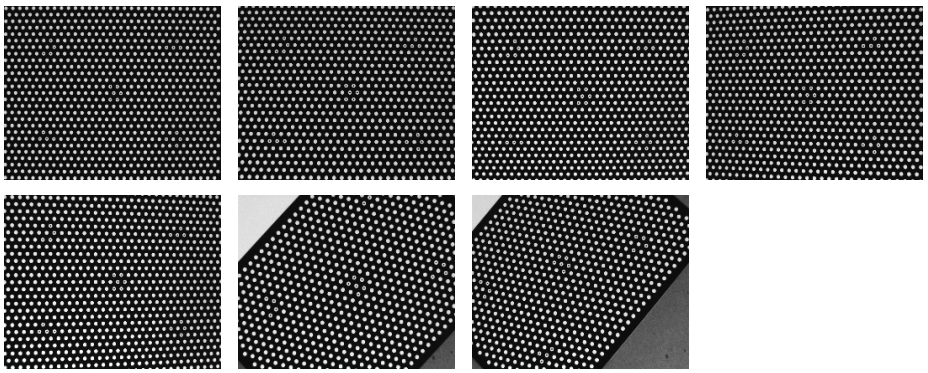
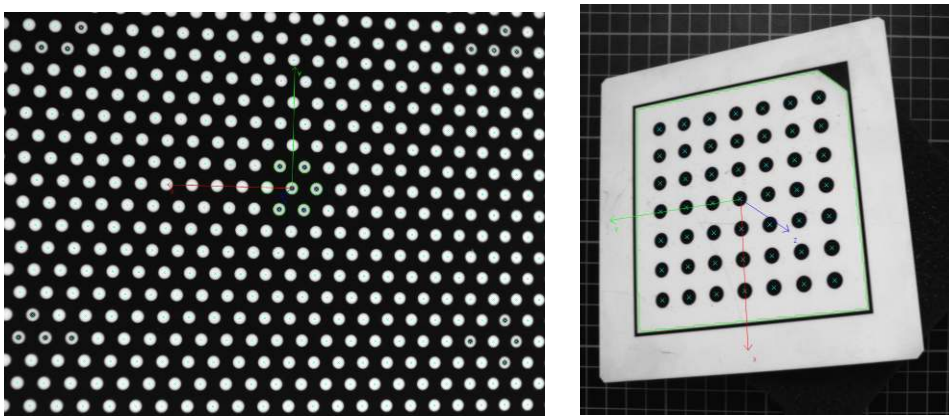


Figure 7.2: Example for suitable calibration images using a calibration plate with hexagonally arranged marks.

Issues (page 191).

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 have to be transferred for the existing 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.

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 194). 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.
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 194) 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. A result of 0% indicates a very defective image feature whereas 100% equals ideal quality. This can help to improve the 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, for example, 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, that is 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 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, that is 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

In the drop-down menus under `Display Parameters` you can 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

Some of the parameters under `Calibration Plate Extraction 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:

- `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.

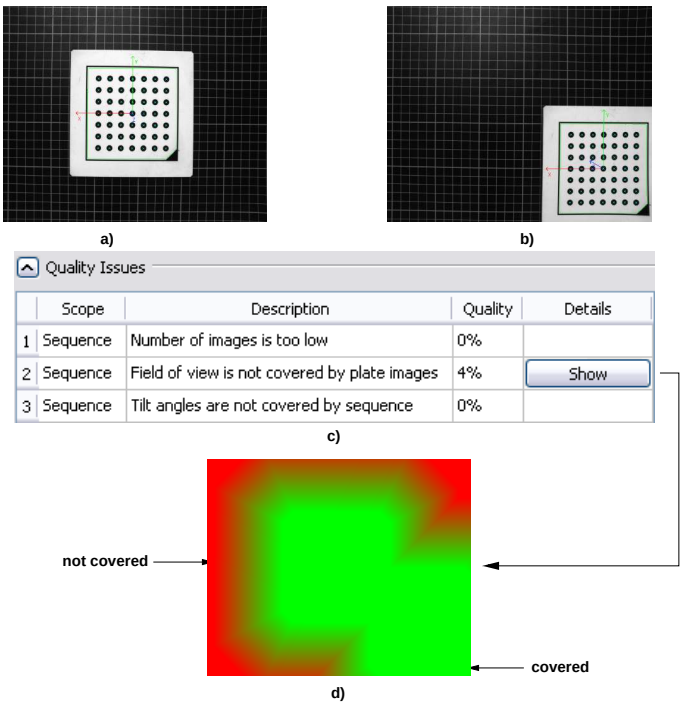


Figure 7.4: Output when not enough calibration images have been taken. .

a) and b): Calibration sequence consisting of two calibration plate images

c) `Show` button appears due to the fact that the coverage is not sufficient

d) Image shows which parts of the field of view are not covered by calibration plate images

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, for example, the precise focal length, the size of the camera chip, or the distortion caused by an imperfect lens, and *external* parameters, for example, the position and orientation of the vision system.

- `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.

When you are finished with the results, go on to the `Code Generation` (page 199) tab.

7.2.3.1 `Calibration Status`

This box displays whether the calibration was successful, and the `Mean Error` in pixels.

If you either delete calibration images (page 188), change `Calibration Plate Extraction Parameters` (page 194) or `Camera Parameters` (page 186) 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 190) again on the `Calibration` tab.

Mean Error Designates the average error in pixels during the calibration process. When 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 191).

7.2.3.2 `Camera Parameters`

The internal camera parameters include `Cell Width (Sx)` and `Cell Height (Sy)` in μm, `Focal Length` in mm, `Center Column (Cx)` and `Center Row (Cy)`, `Image Width` and `Image Height` in pixels. They also include `Kappa` in m^{-2} or instead of `Kappa`, the distortion parameters `Radial 2nd Order (K1)` in m^{-2}, `Radial 4th Order (K2)` in m^{-4}, `Radial 6th Order` in m^{-6}, `Tangential 2nd Order (P1)` and `Tangential 2nd Order (P2)` in 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 μ/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`

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, using the radio buttons. You can also decide whether 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 198)
- `Code Preview` (page 198)

When you are finished with configuring the options, *check the position of the insert cursor* and click `Insert Code` (page 199) under `Calibration` or `Sample Usage` to insert the code.

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 180), enable `Initialize Acquisition`.

When you are finished, *check the position of the insert cursor* and click `Insert Code` (page 199) 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` .

When you are finished, *check the position of the insert cursor* and click `Insert Code` (page 199) 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`.

When you are finished, *check the position of the insert cursor* and click `Insert Code` to insert the code into your HDevelop program.

7.2.4.5 Code Preview

Here, you can, for example, edit or replace individual operators of the code lines proposed by the Calibration Assistant.

For details, see also `Code Generation` (page 200) 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` (page 200)

Tool bar with a selection of important buttons:

- `Load Assistant Settings`

- Save Current Assistant Settings
- Insert Code
- Calibrate (page 188)
- Help

Tabs with the dialogs for most of the tasks that can be done with the Calibration Assistant:

- Setup
- Calibration
- Results
- Code Generation (page 201)

Furthermore, it provides a status bar at the bottom in which messages are displayed. The status bar also displays the *calibration results* (page 195), that is, 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

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 using 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.

7.2.5.2 The Menu Calibration

Via the menu Calibrate you can run a calibration as described in the section Calibrating (page 190).

7.2.5.3 The Menu Code Generation

Insert the Generated Code Lines

Code Generation ▷ Insert Code lets you insert the code that is generated according to the current settings of the Calibration Assistant into the program window. This is also accessible as tool bar button or as button inside the tab Code Generation. Inserting code via menu or tool bar will generate code for calibration and samples.

Release the Generated Code Lines

With `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. Changes, for example, 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 (page 199) 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 186)
- `Calibration Plate` (page 186)
- `Camera Parameters` (page 186)

7.2.5.6 The Tab `Calibration`

The `Calibration` tab includes:

- `Image Source` (page 188)
- `Calibration` (page 190)
- `Quality Issues` (page 191)
- `Display Parameters` (page 194)
- `Calibration Plate Extraction Parameters` (page 194)

7.2.5.7 The Tab `Results`

The `Results` tab includes the following subdivisions:

- `Calibration Status` (page 195)
- `Camera Parameters` (page 196)
- `Camera Pose` (page 196)
- `Display Results` (page 196)

7.2.5.8 The Tab `Code Generation`

The `Code Generation` tab includes the following subdivisions:

- `Calibration` (page 196)
- `Sample Usage` (page 197)
- `Variable Names` (page 198)
- `Code Preview` (page 198)

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 information on these matching methods and how to choose a suitable method for your application, please refer to the Solution Guide II-B, section 1 on page 7.

Using the Matching Assistant you can

- Configure (page 210) and test the matching process with a few mouse clicks and
- Interactively optimize the parameters for your application regarding the speed and recognition rate (page 218).

All you need is a single model image and a set of test images. The Matching Assistant guides and assists you setting up and optimizing your matching application. Its individual elements are explained in the part Matching Assistant Reference (page 204).

In this manual, the following special terms are used:

Matching Matching is the process of locating an object described by a *model* in an image. The main results of the matching process are the position and orientation of the found object instance and its matching score.

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. More information about alignment using shape-based matching can be found in the Solution Guide II-B, section 2.5.3 on page 33. To remove perspective or lens distortions, for example, if the camera observes the scene under an inclined angle, you must *rectify* the image first.

Model The *model* is an internal representation of the object containing only the information characterizing the object. This model is created by the assistant from an example image of the object, the *model image*, which is provided by you. The model is used when searching for the object in the *test images*. You can also provide the Matching Assistant with a *model*, see section 7.3.3.1 on page 204.

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. The object should appear in its default position and orientation and not be occluded.

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 205).

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. *Rectifying* 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*. These images should be representative images from your matching application. The object should appear in all allowed variations of its position, orientation, occlusion, and illumination.

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 in four steps.

We recommend resetting all model and search parameters via the toolbar button `Reset Model` (page 207) 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

Select the matching method from the pull-down menu in the toolbar (page 207) of the Matching Assistant.

7.3.2.2 Creating the Model

A *model* is created in three steps:

(1) Load the *model image*. This can be done via the menu item `File` ▷ `Load Model Image` or the corresponding button and text field under `Model` and `Model Source` inside the tab `Creation` (page 208).
Alternatively choose the Image Acquisition Assistant (page 180) as image source.

(2) Specify an *ROI* around the object. This can be done via the menu item `ROI` (page 205) or the corresponding buttons inside the tab `Creation` (page 208).

(3) Specify standard and advanced model parameter values that are available for your matching method. This can be done via the menu `Parameters` (page 206) or the tab `Parameters` (page 210).

Alternatively, you can load a model.

7.3.2.3 Testing the Model

The model can be tested by performing the following steps:

- Load one or more *test images*. This can be done via the menu item `Usage` ▷ `Test Images` ▷ `Load Test Images` (page 206) or the button `Load` in the tab `Usage` (page 214).
  Alternatively chose the Image Acquisition Assistant (page 180) to acquire images by activating the corresponding checkbox under `Test Image Source`.
- Specify standard search parameter values. This can be done via the menu item `Usage` ▷ `Standard Model Use Parameters` (page 206), or the menu item `Standard Use Parameters` (page 215) in the tab `Usage` (page 214).
- Assure that all objects are found in all *test images* by comparing the number of existing model instances with the number of found instances (applying for example, `Find Model` in `Usage` (page 214)) or simply determine the recognition rate (page 218).

7.3.2.4 Optimizing the Parameters

Configure your *matching* process such that the search is successful in all test images. To do so, you can start to optimize the parameters on the tabs `Parameters` and `Usage`.

You can also optimize your application regarding the speed. How to do so depends on the matching method and the specific application. We recommend to check following parameters which we see as especially promising:

- **shape-based matching**: The Matching Assistant allows you to optimize the search parameters `Minimum Score` (page 215) and `Greediness` (page 215). See Solution Guide II-B, section 3.2.9 on page 73.
- **deformable matching**: The Matching Assistant allows you to optimize the search parameters `Minimum Score` (page 215) and `Greediness` (page 215). See Solution Guide II-B, section 3.3.4 on page 83.

- **correlation-based matching**: The search parameter `Minimum Score` (page 215) can be tuned. See Solution Guide II-B, section 3.1.4 on page 50.
- **descriptor-based matching**: The model parameters `Fern Number` (page 210) and `Fern Depth` (page 210) can be tuned. Few ferns with a large depth enable a fast online matching which, however, requires more memory. See Solution Guide II-B, section 3.5.4 on page 100 for further optimization possibilities..

To find out if a speed up is necessary or successful, it is useful to check the `Statistics` on the tab `Inspect` (page 218).

Search parameters can also be automatically improved via the menu item `Usage` ▷ `Go To Optimize Recognition Speed` (page 206), or the menu item `Optimize Recognition Speed` (page 217) in the tab `Usage` (page 214). But we recommend checking whether the matching still succeeds in all *test images* after each modification.

7.3.2.5 Generate Code

Insert the code performing the chosen matching tasks within an HDevelop program. This code includes all of the previous adaptions and can be used in the final application.

The menu `Code Generation` (page 207) and the tab `Code Generation` (page 218) allow you to choose between several options and change parameter names, which has a direct effect on the generated code.

7.3.3 Matching Assistant Reference

The Matching Assistant shown in figure 7.5 on page 209 consists of the following elements:

- The following pull-down menus: `File`, `ROI`, `Parameters` (page 206), `Usage` (page 206), `Inspect` (page 207), `Code Generation` (page 207), and `Help` (page 207)
- A tool bar (page 207) for direct functionality access.
- The following tabs: `Creation` (page 208), `Parameters` (page 210), `Usage` (page 214), `Inspect` (page 218), and `Code Generation` (page 218)
- A status bar at the bottom in which messages are displayed. The status bar also displays the matching results. 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.

Note, not all options and parameters are supported for every matching method.

7.3.3.1 The Menu `File`

The menu `File` offers you the following options:

Menu Entry	Description
`Load Model Image`	Select a *model image* which will be used to create the model of the object you want to find later. The Matching Assistant can read the image file types `TIFF`, `BMP`, `GIF`, `JPEG`, `JPEG-XR`, `PPM`, `PGM`, `PNG`, and `PBM`. *Note*: Descriptor-based matching and correlation-based matching work on a single channel. For color RGB images, the red channel will be used.
`Load Model`	Load a *model* that you have saved with the Matching Assistant or HALCON in `DXF` format (or in the HALCON formats `SHM`, `NCM`, `DSM` or `DFM`). *Note*: A model loaded from file cannot be changed and contains no information about the image from which it was created. As a consequence, all the menu items, buttons, and dialogs that would enable you to change the model parameters or display the model image are not selectable.
`Save Model`	Save the created model (page 203) in a file.

Menu Entry	Description
Load Camera Parameters	Load camera parameters from file. For more information on camera parameters see "Calibration". *Restriction*: Only supported for descriptor-based matching and deformable matching.
Load Camera Pose	Load a camera pose from file. For more information on poses, see for example, "Transformation ▷ Poses" and Solution Guide III-C, section 2.1.4 on page 20. *Restriction*: Only supported for descriptor-based matching and deformable matching.
Display Image Pyramid Level	Inspect the model image pyramid and open the corresponding dialog in the tab `Creation` (page 209).
Load Assistant Settings	Load the settings of a previously saved Matching Assistant session. *Note*: 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	Save the current settings of a Matching Assistant session.
Close Dialog	Closes the Matching Assistant dialog. In doing so, 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 reopen the Matching Assistant with the same settings.
Exit Assistant	Close the assistant's dialog and exit the Matching Assistant without storing the current settings.

7.3.3.2 The Menu `ROI`

The menu `ROI` allows you to generate the *model region of interest* by drawing and modifying it on the displayed *model image*.

To draw objects, click into the image and 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. Finish the creation by clicking once with the right mouse button.

Menu Entry	Description
Draw Circle	Interactively create a circle, see `gen_circle` for its parameters.
Draw Ellipse	Interactively create an ellipse, see `gen_ellipse` for its parameters.
Draw Axis-aligned Rectangle	Interactively create an axis-aligned rectangle, see `gen_rectangle1` for its parameters.
Draw Rotated Rectangle	Interactively create an oriented rectangle, see `gen_rectangle2` for its parameters.
Draw Arbitrary Region	Create polygons and free-form shapes: • 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. • 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 can neither edit the ROI form nor the orientation after its creation!
Delete Selected ROI Item	Delete a selected model ROI.
Delete All ROIs	Delete all model ROIs.

Menu Entry	Description
Union	The ROI to be added and the existing total ROI form the new total model ROI by taking the union of the ROIs. *Note*: The behavior stays for this ROI also when modifying the model ROIs later.
Intersection	The ROI to be added and the existing total ROI form the new total model ROI by taking the intersection of the ROIs. *Note*: The behavior stays for this ROI also when modifying the model ROIs later.
Difference	The ROI to be added and the existing total ROI form the new total model ROI by taking the difference to the existing total ROI. *Note*: The behavior stays for this ROI also when modifying the model ROIs later.
XOR (symmetrical difference)	The ROI to be added and the existing total ROI form the new total model ROI by taking the difference of the ROIs. *Note*: The behavior stays for this ROI also when modifying the model ROIs later.
Load ROI from File	Load an ROI as model ROI.
Save ROI to File	Save the current model ROI.

7.3.3.3 The Menu Parameters

The menu Parameters allows you to adapt model parameters suitable to your matching application.

Note: Depending on the matching method, different parameters are available for adaption.

Menu Entry	Description
Standard Model Parameters	Open the dialog for specifying the standard model parameters (page 210) in the tab Parameters (page 210).
Advanced Model Parameters	Open the dialog for specifying the advanced parameters (page 212) in the tab Parameters (page 210).

7.3.3.4 The Menu Usage

The menu Usage allows you to apply your *model* on *test images* and adapt suitable search parameters.

Menu Entry	Description
Test Images ▷ Load Test Images	Select one or more images in a standard file selection box.
Test Images ▷ Remove Test Image	Remove the currently selected test image from the list of test images.
Test Images ▷ Remove All Test Images	Remove all test images from the list of test images.
Test Images ▷ Display Selected Test Image	Redisplay the selected test image without newly selecting it.
Test Images ▷ Show Test Image Settings	Open the dialog for Test Images (page 214) in the tab Usage (page 214).
Standard Model Use Parameters	Open the dialog for specifying the standard search parameters (page 215) in the tab Usage (page 214).
Advanced Model Use Parameters	Open the dialog for specifying the advanced search parameters (page 216) in the tab Usage (page 214).
Go To Optimize Recognition Speed	Open the dialog for specifying the Optimize Recognition Speed (page 217) in the tab Usage (page 214).

Menu Entry	Description
Optimize Recognition Speed	Run the optimization as done by Run Optimization (page 217) in the tab Usage (page 214).

7.3.3.5 The Menu Inspect

The menu Inspect allows you to determine the recognition rate and the pose bounds of the found object instances for the used set of *test images*.

Menu Entry	Description
Determine Recognition Rate	Open the tab Inspect (page 218) to run the recognition rate and statistics generation as if pressing on the button Run (page 218) in the tab Inspect (page 218).

7.3.3.6 The Menu Code Generation

The menu Code Generation allows you to generate and modify the HDevelop code for your specified matching application.

Menu Entry	Description
Show Code Generation Options	Open the dialog for determining the options for the code generation inside the tab Code Generation (page 218).
Show Variables for Code Generation	Open the dialog for determining the variables used for the code generation inside the tab Code Generation (page 218).
Insert Code	Insert the code that is generated according to the current settings of the Matching Assistant into the Program Window.
Release Generated Code Lines	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, for example, the deletion of code lines, can then only be applied directly in the Program Window and not from within the Matching Assistant anymore.
Delete Generated Code Lines	Delete the code lines that you have previously generated and inserted into the Program Window of HDevelop from within the Matching Assistant.
Show Code Preview	Open the dialog showing the Code Preview (page 219) in the tab Code Generation.

7.3.3.7 The Menu Help

The menu Help allows you to open the HALCON help window.

Menu Entry	Description
Help	Access the online documentation.

7.3.3.8 The Tool Bar

The tool bar provides buttons for direct access.

Menu Entry	Description.
Load Assistant Settings	Load the assistant settings as done via Load Assistant Settings in the menu File (page 204).
Save Current Assistant Settings	Save the current settings as done via Save Current Assistant Settings in the menu File (page 204).

Menu Entry	Description.
Insert Code	Insert the HDevelop code as done via Insert Code in the menu Code Generation.
Save Model	Save the current matching model as done via Save Model in the menu File (page 204).
Display Model	Display the current matching model as done via Model in the tab Creation (page 209).
Reset Model	Reset the model. This way, the parameters are reset to their default settings and the *model image*, the *model ROI*, and the *test images* are deleted.
Optimize Recognition Speed	Optimize the recognition speed as done via Optimize Recognition Speed in the tab Usage (page 214).
Determine Recognition Rate	Determine the recognition rate as done via Determine Recognition Rate in the tab Inspect (page 218).
Drop-down menu with the matching method	Select the matching method suiting your application.
Help	Open the HDevelop help window.

7.3.3.9 The Tab Creation

The tab Creation allows you to create or load a matching *model*. It consists of different sections.

Model

Determine the source type for your *model*.

Menu Entry	Description
Create from Image	Create a new model from an image.
Load	Load an already existing model, for more information see the menu entry Load Model Image (page 204).

Model Source

Determine the source for your *model*.

Menu Entry	Description
Graphics Window	Use the image currently shown in the graphics window (page 73).
File	Load a *model* (see the menu item Load Model (page 204) or a *model image* (see the menu item Load Model Image (page 204) from file. The following settings are available for loaded 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 (page 204), • or from an Acquisition Assistant (page 180). Note that when you load the model from a file, the loaded model cannot be changed and contains no information about the image from which it was created. As a consequence, all the menu items, buttons, and dialogs that would enable you to change the model parameters or display the model image will not be selectable.
Acquisition Assistant	Open a new acquisition assistant (page 180) session and use the resulting image.

Model ROI

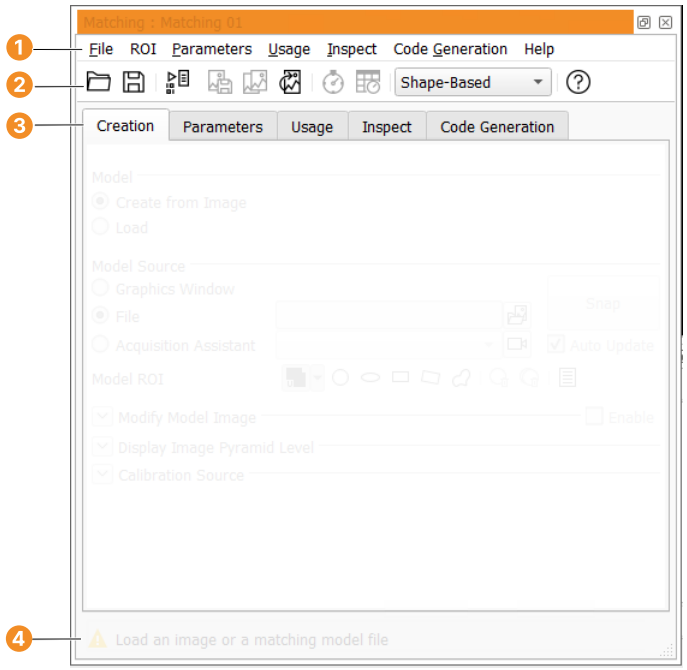


Figure 7.5: The matching assistant and its different elements.

1. Pull-down menus
2. Tool bar
3. Tabs
4. Status bar

Depending on the source type, you can draw an *ROI*. Next to the options explained in the `The Menu ROI` (page 205), you have the following option.

Menu Entry	Description
`Show List of ROI Shapes`	View the region shapes and therewith refine the ROI you have drawn. 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.

`Modify Model Image`

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.

Restriction: Only 8-bit (byte) model images can be modified! To modify images of another bit depth, a prior conversion is required.

Menu Entry	Description
Drop-down menu: `Inpaint Regions`	Remove unwanted structures by smoothing the area within the ROI. Select the shape of the modification ROI using the buttons on the right. You can draw the ROIs by positioning the mouse cursor as explained in the section about the `menu ROI` (page 205). Use the right mouse button to conclude the choice and view the effects of the inpainting. The shape should not only cover the area that is to be removed but also some of the "good" area around.
Drop-down menu: `Inpaint Regions Smooth`	Smooth the area within the ROI without completely removing structures. Select the shape of the modification ROI using the buttons on the right. You can draw the ROIs by positioning the mouse cursor as explained in the section about the `menu ROI` (page 205). Use the right mouse button to conclude the choice and view the effects of the inpainting. The shape should not only cover the area that is to be removed but also some of the "good" area around.
	Delete selected contours from your model. • `Remove edges by selection`: Remove edges by clicking on a contour

on the size of the model *ROI*; depending on the selected `Contrast` and `Min. Component Size`, higher pyramid levels may not contain any model points.

Menu Entry	Description
`Image`	Select the pyramid level of the displayed image using the sliders.
`Model`	Select the pyramid level of the displayed model contours using the sliders.
Lock/unlock button	Enforce the same level for the model image and the model or to enable different level selections.

`Calibration Source`

Select a calibration source.

Menu Entry	Description
`None`	No calibration data is used.
`Calibration Files`	Load your data files (files that end with `.cal` or `.dat`)
`Calibration Assistant`	Use the Calibration Assistant (page 184) for a guided calibration.

The Tab `Parameters`

The tab `Parameters` allows you to adapt various model parameters so the model suits your matching application.

Menu Entry	Description
`Standard Model Parameters`	Set the basic model parameters. An overview is given in the part Standard Model Parameters
`Advanced Model Parameters`	Set additional parameters that let you handle special cases and further optimize the model. An overview is given in the part Advanced Model Parameters (page 212)

Note: The supported parameters depend on the matching method.
For shape-based matching, the results of the generated code may vary from the generated results shown for the Test Images in the tab Usage (page 214) due to automatic value determination.

`Standard Model Parameters`

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.

Menu Entry	Description
`Contrast (High/Low)`	*SBM*: ✓, *CBM*: ×, *DBM*: ×, *DM*: ✓ The two parameters `Contrast (Low)` and `Contrast (High)` for shape-based matching and `Contrast` for deformable matching determine which pixels in the selected *ROI* are included in the *model*. 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. For information on the functionality of these parameters see their documentation in `set_generic_shape_model_param` (shape-based matching) and `create_planar_uncalib_deformable_model` (deformable matching), respectively. `Auto Select`: The Matching Assistant selects a suitable value automatically based on the model image. It selects the value by trying to obtain many long and straight contour segments. *Note*: 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.

Menu Entry	Description
`Min. Component Size`	*SBM*: ✓, *CBM*: ✕, *DBM*: ✕, *DM*: ✕ Specify the minimum size, that is, number of pixels, which contour parts must have to be included in the *model*. For information on the functionality of the parameter see its documentation in `set_generic_shape_model_param`. `Auto Select`: The Matching Assistant selects a suitable value automatically based on the model image.
`Pyramid Levels`	*SBM*: ✓, *CBM*: ✓, *DBM*: ✕, *DM*: ✓ Select the number of pyramid levels. You can enter the value directly in the text field or by using the slider next to it. See the Solution Guide II-B, section 2.3 on page 25 for more information about the image pyramid and its levels. See also `Display Image Pyramid` (page 209) for information how to inspect the model image pyramid. `Auto Select`: The Matching Assistant selects a suitable value automatically based on the model image. 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 209) and select a suitable value manually.
`Starting Angle`	*SBM*: ✕, *CBM*: ✓, *DBM*: ✕, *DM*: ✓ Specify the starting angle of the allowed range of rotation (unit: °). *Example*: To allow model rotations up to +/-5 °, for example, you should set the starting angle to -5 ° and the angle extent to 10 ° or angle end to 5 °. *Note*: The range of rotation is defined relative to the created model, that is, for a model created from an image a starting angle of 0 ° corresponds to the orientation the object has in the model image.
`Angle Extent`	*SBM*: ✕, *CBM*: ✓, *DBM*: ✕, *DM*: ✓ Specify how much the object is allowed to rotate (unit: °). *Example*: To allow model rotations of up to 10 °, for example, you should set the angle extent to 10 °.
`Min./Max. Angle`	*SBM*: ✕, *CBM*: ✕, *DBM*: ✓, *DM*: ✕ Define the range for the angle of rotation around the normal vector of the model. For further information see `create_uncalib_descriptor_model`, part “Simulation parameters” `'min_rot'` and `'max_rot'`.
`Min./Max. Column/Row Scale`	*SBM*: ✓, *CBM*: ✕, *DBM*: ✕, *DM*: ✓ Define the allowed scaling range. The allowed range of scale is defined separately in row and column direction. Thus, it is described by the parameters: • `Min. Column Scale` • `Max. Column Scale` • `Min. Row Scale` • `Max. Row Scale` The model template has scaling values of 1.0. This means, if the model was generated from a *model image*, the scaling values of the model instance in this image are 1.0. Depending on the specified parameters, the most efficient scaling method is used. This means: • **Unscaled matching**: This method is used if all four scale factors are equal to 1.0. • **Isotropically scaled matching**: This method is used if rows and columns have the same scaling factors (but not all are equal to 1.0). • **Anisotropically scaled matching**: This method is used if rows and columns have different scaling factors.

Menu Entry	Description
Min./Max. Scale	*SBM*: ✕, *CBM*: ✕, *DBM*: ✓, *DM*: ✕ Specify the scale range of the model. For further information see create_uncalib_descriptor_model, part "Simulation parameters" 'min_scale' and 'max_scale'.
Model Type	*SBM*: ✕, *CBM*: ✕, *DBM*: ✕, *DM*: ✓ Select between two model types, depending on the expectations concerning the deformation of the object: • local deformable • planar
Detector Type	*SBM*: ✕, *CBM*: ✕, *DBM*: ✓, *DM*: ✕ Specify the type of detector. The detector is used for the extraction of stable interest points within the image. Available types: • lepetit • harris • harris_binomial For more information on the detector types see create_uncalib_descriptor_model.
Radius	*SBM*: ✕, *CBM*: ✕, *DBM*: ✓, *DM*: ✕ Determine the radius of the circle in the Lepetit method. For further information see create_uncalib_descriptor_model and points_lepetit. *Restriction*: Only for detectors of type lepetit.
Min. Score	*SBM*: ✕, *CBM*: ✕, *DBM*: ✓, *DM*: ✕ Specify the minimum score a potential match must have to be returned as match. For more information and recommendations see Solution Guide II-B, section 3.5.4.4 on page 102. *Restriction*: Only for detectors of type lepetit.
Gradient Sigma	*SBM*: ✕, *CBM*: ✕, *DBM*: ✓, *DM*: ✕ Determine the amount of smoothing used for the gradient calculation. For further information see create_uncalib_descriptor_model and SigmaGrad in points_harris. *Restriction*: Only for detectors of type harris.
Gradient Mask Size	*SBM*: ✕, *CBM*: ✕, *DBM*: ✓, *DM*: ✕ Determine the amount of smoothing used for the gradient calculation. For further information see create_uncalib_descriptor_model and MaskSizeGrad in points_harris_binomial. *Restriction*: Only for detectors of type harris_binomial.
Threshold.	*SBM*: ✕, *CBM*: ✕, *DBM*: ✓, *DM*: ✕ Determine the minimum filter response for the points. For further information see create_uncalib_descriptor_model and Threshold in points_harris and points_harris_binomial. *Restriction*: Only for detectors of type harris and harris_binomial.
Fern Depth	*SBM*: ✕, *CBM*: ✕, *DBM*: ✓, *DM*: ✕ Specify the depth of the classification fern. For more information see the explanation of the 'depth' in Solution Guide II-B, section 3.5.3.2 on page 99.
Fern Number	*SBM*: ✕, *CBM*: ✕, *DBM*: ✓, *DM*: ✕ Specify the number of used fern structures. For more information see the explanation of the 'number_ferns' in Solution Guide II-B, section 3.5.3.2 on page 99.

Advanced Model Parameters

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.

Menu Entry	Description
`Angle Step`	*SBM*: ✓, *CBM*: ✓, *DBM*: ×, *DM*: ✓ Determine the step length within the selected range of angles. For further information see Solution Guide II-B, section 3.2.4.4 on page 61 (shape-based matching), `create_ncc_model` (correlation-based matching), and `find_planar_uncalib_deformable_model` (deformable matching). *Note*: Each time you create a model *ROI* or change the parameter `Contrast` (page 210), the Matching Assistant automatically selects a suitable value for `Angle Step` to obtain the highest possible accuracy. `Auto Select`: The Matching Assistant selects a suitable value automatically.
`Row/Column Scale Step`	*SBM*: ✓, *CBM*: ×, *DBM*: ×, *DM*: ✓ Determine the step length within the selected scaling in row and column direction. For further information see Solution Guide II-B, section 3.2.4.3 on page 60 (shape-based matching) and `find_planar_uncalib_deformable_model` (deformable matching). *Note*: Each time you create a model *ROI* or change the parameter `Contrast` (page 210), the Matching Assistant automatically selects a suitable value for `Row Scale Step` and `Column Scale Step` to obtain the highest possible accuracy. `Auto Select`: The Matching Assistant selects a suitable value automatically.
`Metric`	*SBM*: ✓, *CBM*: ✓, *DBM*: ×, *DM*: ✓ Specify how pixels are compared with the model, more precisely whether and how the polarity must be observed. For information on the functionality of the parameter and its possible values see Solution Guide II-B, section 3.2.4.5 on page 62 (shape-based matching) Solution Guide II-B, section 3.1.3.3 on page 49 (correlation-based matching), and Solution Guide II-B, section 3.3.3.4 on page 82 (deformable matching).
`Optimization`	*SBM*: ✓, *CBM*: ×, *DBM*: ×, *DM*: ✓ Determine the number of points by which the model is reduced. For further information see Solution Guide II-B, section 3.2.4.2 on page 59 (shape-based matching) and Solution Guide II-B, section 3.3.3.2 on page 82 (deformable matching). `Auto Select`: The Matching Assistant selects a suitable value automatically based on the model image.
`Min. Contrast`	*SBM*: ✓, *CBM*: ×, *DBM*: ×, *DM*: ✓ Determine the minimal contrast (gray value difference to neighboring pixels) a point in a search image must at least have in order to be compared with the model during the *matching* process. For information on the functionality of the parameter see `set_generic_shape_model_param` (shape-based matching) and Solution Guide II-B, section 3.3.3.4 on page 82 (deformable matching). `Auto Select`: The Matching Assistant selects a suitable value automatically based on the model image.
`Check Neighbor`	*SBM*: ×, *CBM*: ×, *DBM*: ✓, *DM*: × Determine the number of checked neighbors in the circle when using the Lepetit method. For further information see `create_uncalib_descriptor_model` and `CheckNeighbor` in `points_lepetit`. *Restriction*: Only for detectors of type `lepetit`.
`Neighbor Diff. Threshold`	*SBM*: ×, *CBM*: ×, *DBM*: ✓, *DM*: × Determine the threshold of gray value difference to each circle point when using the Lepetit method. For further information see `create_uncalib_descriptor_model` and `MinCheckNeighborDiff` in `points_lepetit`. *Restriction*: Only for detectors of type `lepetit`.
`Subpixel`	*SBM*: ×, *CBM*: ×, *DBM*: ✓, *DM*: × Turn the refinement for subpixel accuracy on or off. For further information see `create_uncalib_descriptor_model` and `SubPixel` in `points_lepetit` and `points_harris_binomial`. *Restriction*: Only for detectors of type `lepetit` and `harris_binomial`.

Menu Entry	Description
Patch Size	*SBM*: ×, *CBM*: ×, *DBM*: ✓, *DM*: × Specify the side length of the quadratic neighborhood that is used to describe the individual interest point. For more information see the explanation of the 'patch_size' in Solution Guide II-B, section 3.5.3.2 on page 99.
Tilt	*SBM*: ×, *CBM*: ×, *DBM*: ✓, *DM*: × Specify whether the projective transformations are used during the simulation. For more information see the explanation of the 'tilt' in Solution Guide II-B, section 3.5.3.2 on page 99.
Smoothing Sigma	*SBM*: ×, *CBM*: ×, *DBM*: ✓, *DM*: × Determine the amount of smoothing used for the integration of the gradients. For further information see create_uncalib_descriptor_model and SigmaSmooth in points_harris. *Restriction*: Only for detectors of type harris.
Alpha	*SBM*: ×, *CBM*: ×, *DBM*: ✓, *DM*: × Determine the weight of the squared trace of the squared gradient matrix. For further information see create_uncalib_descriptor_model and Alpha in points_harris and points_harris_binomial. *Restriction*: Only for detectors of type harris and harris_binomial.
Smoothing Mask Size	*SBM*: ×, *CBM*: ×, *DBM*: ✓, *DM*: × Determine the amount of smoothing used for the integration of the gradients. For further information see create_uncalib_descriptor_model and MaskSizeSmooth in points_harris_binomial. *Restriction*: Only for detectors of type harris_binomial.
Min. Size	*SBM*: ×, *CBM*: ×, *DBM*: ×, *DM*: ✓ Suppress small connected components of the model contours. For more information see the explanation of the 'min_size' in Solution Guide II-B, section 3.3.3.5 on page 82.
Part Size	*SBM*: ×, *CBM*: ×, *DBM*: ×, *DM*: ✓ Adjust the size of the model sub-parts. For more information see the explanation on the 'part_size' in Solution Guide II-B, section 3.3.3.5 on page 82.

7.3.3.10 The Tab Usage

The tab Usage allows you to apply your *model* on *test images* and adapt suitable search parameters.

Test Image Source

Determine the source type of your *test images*.

Menu Entry	Description
Image Files	Test images are loaded from file.
Image Acquisition Assistant	Use the Image Acquisition Assistant (page 180) to acquire your test images.

Test Images

You can select a test image from the text box of the dialog Test Images. The selected image is automatically displayed in the graphics window of HDevelop.

Menu Entry	Description
Load/Add	Load an image from file or add an image using the Image Acquisition Assistant (page 180).
Live Image	Add the image acquired by the live image acquisition mode of the Image Acquisition Assistant (page 180).
Remove	Remove the currently selected test image from the list.
Remove All	Remove all test images.

Menu Entry	Description
`Save`	Save the selected image if it has been added using the Image Acquisition Assistant (page 180).
`Save All`	Save all images added by the Image Acquisition Assistant (page 180).
`Set Reference`	Set the match in the chosen image as reference position for alignment. *Note*: This is only possible if a match is detected in the image. If no reference image is chosen, the model image will be used as basis for the alignment or rectfication.
`Detect All`	Search all objects in the complete sequence of test images that were loaded before (independent of the set `Maximum Number of Matches`). The results are displayed successively in the graphics window. *Note*: For descriptor-based matching this also sets the `Maximum Number of Matches` if it has not been set previously (it is set to 0). *Note*: This also sets the number of visible objects in the test image if it has not been set previously. *Note*: 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.
`Find Model`	Search the object in the currently selected test image. The result is displayed in the assistant window. *Note*: 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.
`Always Find`	Start the matching process automatically on the selected test image. The result is displayed in the graphics window. *Note*: If the matching process is started for the first time of after you changed a model parameter, the internally stored model is actually created, which takes some time.
Number of Visible Objects in Test Image	Specify how many objects are really visible in the current test image. To do so, click onto the currently displayed number of detected objects (`Visible`) in the text field of the currently selected test image. The specified numbers of visible objects are used when determining the recognition rate, that is, 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.

`Specifying Standard Search Parameters`

The menu item `Standard Use Parameters` allows you to specify search parameters.

The following table gives you an overview over the available standard search 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.

Menu Entry	Description
`Starting Angle`	*SBM*: ✓, *CBM*: ✕, *DBM*: ✕, *DM*: ✕ Specify the starting angle of the allowed range of rotation (unit: °). *Example*: To allow model rotations up to +/-5 °, for example, you should set the starting angle to -5 ° and the angle extent to 10 ° or angle end to 5 °. *Note*: The range of rotation is defined relative to the created model. For a model created from an image a starting angle of 0 ° corresponds to the orientation the object has in the model image.
`Ending Angle`	*SBM*: ✓, *CBM*: ✕, *DBM*: ✕, *DM*: ✕ Specify the ending angle of the allowed range of rotation (unit: °). *Example*: To allow model rotations up to +/-5 °, for example, you should set the ending angle to 5 ° and the starting angle extent to -5 °. *Note*: The range of rotation is defined relative to the created model. For a model created from an image an ending angle of 0 ° corresponds to the orientation the object has in the model image.
`Minimum Score`	*SBM*: ✓, *CBM*: ✓, *DBM*: ✓, *DM*: ✓ Specify the minimum score that a potential match must reach to be regarded as model instance in the image. For further information see Solution Guide II-B, section 2.5.6 on page 45.

Menu Entry	Description
`Maximum Number of Matches`	*SBM*: ✓, *CBM*: ✓, *DBM*: ✓, *DM*: ✓ Specify how many instances of the object are searched for in the image. If you specify the value 0, all found instances are displayed. *Note*: The parameter sets a maximum value, that is, if more object instances are present in the image only the best instances of the specified number are displayed.

`Specifying Advanced Model Search Parameters`

The menu item `Advanced Use Parameters` allows you to specify search parameters.

The following table gives you an overview over the available advanced search 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.

Menu Entry	Description
`Greediness`	*SBM*: ✓, *CBM*: ✗, *DBM*: ✗, *DM*: ✓ Determine how 'greedily' the search should be carried out. For more information see Solution Guide II-B, section 2.4.3 on page 29.
`Maximum Overlap`	*SBM*: ✓, *CBM*: ✗, *DBM*: ✗, *DM*: ✓ Determines by what fraction two instances may overlap at most in order to be considered as different instances and hence to be returned separately. For a schema see Solution Guide II-B, section 3.2.6.4 on page 66.
`Subpixel`	*SBM*: ✓, *CBM*: ✓, *DBM*: ✗, *DM*: ✓ Determine which type of subpixel refinement should be carried out. For more information and possible values see Solution Guide II-B, section 3.2.6.6 on page 69, (shape-based matching) `find_ncc_model` (correlation-based matching), and `find_planar_uncalib_deformable_model` (deformable matching).
`Max Deformation`	*SBM*: ✓, *CBM*: ✗, *DBM*: ✗, *DM*: ✗ Determine by how much an object is allowed to deviate from the model in order to be considered as a match. The maximal allowable object deformation is specified in pixels. For more information see Solution Guide II-B, section 3.2.6.6 on page 69.
`Last Pyramid Level`	*SBM*: ✓, *CBM*: ✓, *DBM*: ✗, *DM*: ✓ Determine the lowest pyramid level to which the found matches are tracked. For more information on the image pyramid see Solution Guide II-B, section 2.3 on page 25. *Note*: The lowest pyramid level is denoted by a value of 1 (shape-based and deformable) and 0 (correlation-based), respectively. *Example*: When selecting the value 2, the matching starts at the highest pyramid level and tracks the matches to the second lowest pyramid level.
`Timeout`	*SBM*: ✓, *CBM*: ✓, *DBM*: ✗, *DM*: ✗ Set a maximum runtime of the operators used to find the shape model. 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 is aborted.
`Shape models may cross the image border`	*SBM*: ✓, *CBM*: ✗, *DBM*: ✗, *DM*: ✗ Determine whether the models to be found may lie partially outside the image, that is, whether they may cross the image border, independent of the domain. For more information see the description of the parameter `border_shape_models` of `set_generic_shape_model_param`.
`Robust Pyramid Tracking`	*SBM*: ✓, *CBM*: ✗, *DBM*: ✗, *DM*: ✗ Enable a mode to automatically detect during the search the lowest image pyramid level on which at least one match can be found. For more information see the description of the parameter `pyramid_level_robust_tracking` of `set_generic_shape_model_param`.
`Score Type`	*SBM*: ✗, *CBM*: ✗, *DBM*: ✓, *DM*: ✗ Define the type of score to be evaluated. For more information on the parameter and possible values see Solution Guide II-B, section 3.5.4.6 on page 102.

Menu Entry	Description
`Descriptor Min. Score`	*SBM*: ✕, *CBM*: ✕, *DBM*: ✓, *DM*: ✕ Determine what score a potential match must at least have to be regarded as an instance of the model in the image. For more information on the parameter see Solution Guide II-B, section 3.5.4.4 on page 102.
`Guided Matching`	*SBM*: ✕, *CBM*: ✕, *DBM*: ✓, *DM*: ✕ Enable the usage of an already approximately known homography to 'guide' the matching process, which enhances the accuracy of the object recognition. For more information on the parameter see Solution Guide II-B, section 3.5.4.3 on page 101.

`Optimizing the Recognition Speed`

This section allows you to optimizes the recognition speed by means of automatically determining the values for the parameters `Minimum Score` (page 215) and `Greediness` (page 215).

How it works: 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, that is, 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 values for `Minimum Score` and `Greediness` and the reached recognition time. The found values are set as new search parameters.

The speed is calculated as the average recognition speed over all test images.

Menu Entry	Description
`Run Optimization`	Run the optimization.
`Stop`	Interrupt the process of optimizing the recognition speed; please note however, that this event is processed only after the current search has finished.
`Mode`	Specify how the recognition rate is defined. The following options are supported: • `Find number of instances specified in the table above`: In each test image, as many objects are expected as specified manually (page 214). The recognition rate is calculated as the relation of found objects to the sum of expected objects over all images. It is 100% if in each image exactly as many objects are found as specified. *Note*: If you select `Maximum Number of Matches` (page 215) = *0* and specify a lower number of visible objects than found 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. • `Find at least one model instance per image`: In each test image at least one model instance is expected. The recognition rate is calculated as the percentage of test images which fulfill this condition. It is 100% if in all test images at least one object is found. • `Find maximum number of model instances per image`: In each test image, as many objects are expected as specified in the parameter `Maximum Number of Matches` (page 215). The recognition rate is calculated as the relation of found objects to the sum of expected objects over all images. It is 100% if in all test images (at least) `Maximum Number of Matches` objects are found.
`Recognition Rate`	Determine the required recognition rate.

7.3.3.11 The Tab `Inspect`

The tab `Inspect` allows you to obtain the `Recognition Rate` and `Statistics`.

Menu Entry	Description
`Run`	Determines the recognition rate by searching for the object in all loaded test images.
`Stop`	Interrupt the process of determining the recognition rate; please note however, that this event is processed only after the current search has finished.

`Recognition Rate` The recognition rate calculated for different criteria. Note that it depends on the set `Maximum Number of Matches` (page 215) and specified number of visible objects (page 214).

`Statistics` The minimum, maximum and extent for different pose parameters of the found instances as well as for the score and execution time are shown. The range of positions, orientations, and scales in which the object appears in the test images are shown.

7.3.3.12 The Tab `Code Generation`

The tab `Code Generation` allows you to generate and modify the HDevelop code for your specified matching application.

`Options`

This section allows you to determine, which code parts are to be inserted.

Menu Entry	Description
`Insert Code`	Insert the code generated by the Matching Assistant into the Program Window of HDevelop, see the corresponding menu item (page 207).
`Create model from a model image at runtime`	In the application code the matching model will be created at runtime. Select the source for its *model image*: • `Use image`: Use the unmodified model image. • `Load modified image`: Load the modified model image (page 209) you saved previously. Select how the *model ROI* is set: • `Create ROI as in assistant`: Use the specified ROI. • `Generate Code for drawing an ROI at runtime`: A new ROI has to be drawn at runtime.
`Load model file:`	In the application code the matching model will be loaded from file.
`Load recently saved model file`	In the application code the recently saved model will be loaded from file.
`Initialize Acquisition`	Insert code to initializing an image acquisition device.
`Generate display code`	Display the detected model instances in a loop.
`Generate value retrieving code`	Generate code to retrieve the parameter values of the found matches in a loop. *Note:* For shape-based matching only.
`Generate affine transformation code`	Insert code for an affine transformation that can be used for *alignment*. *Note*: A reference image (page 214) has to be set on the `Usage` tab, otherwise the model image is used. *Restriction*: Only for shape-based matching.
`Generate rectification code`	Insert code for rectifying the image of detected model instances. *Note*: A reference image (page 214) has to be set on the `Usage` tab, otherwise the model image is used.

`Specifying the Variable Names for the Code Generation`

This section allows you to modify the names of individual variables needed for the code lines.

The dialog consists of a text field for every variable. The Matching Assistant automatically generates reasonable variable names, but you can change the individual names via the text fields.

`Code Preview`

This section allows you to preview the generated code and gives you the possibility to, for example, edit or replace individual operators of the code lines proposed by the Matching Assistant.

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 222),
- perform a calibration to transfer your results to world coordinates (page 223), and
- if necessary, use advanced features to, for example, optimize your measuring under difficult conditions with fuzzy measuring (page 225).

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 228).

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, for example, 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, for example, reflections on the surface ▷ `Fuzzy Measure` (page 225).

Fuzzy Score The 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 225).

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.

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 223), 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 224).

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 225).

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 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`.

When 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 224). If you want to load calibration data or calibrate now, proceed to the section `Calibration Source` (page 223).

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. When 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 keyboard or click 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.

When 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 220), 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 107) to inspect edges along an ROI. If necessary, that is, 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 225) 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 107), 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 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, for example, 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 107) 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, that is, 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 (`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.

When 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`.

When 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.

When 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 221) 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`.

When 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 184) 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 184) 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 220), continue by creating an ROI (page 220). 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 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.

When 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, for example, 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`, that is, 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 228).

You can furthermore change variable names for

- General parameters,
- ROI coordinates, and
- Measurement results.

When 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.

When 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 when 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 228).

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 220) yet, proceed to either add calibration images or perform a calibration, respectively (page 223), if you want to transform your results in to world coordinates, or continue to create an ROI (page 220) and then step to the next tab to extract edges (page 221).

7.4.5.2 Fuzzy Measuring

So far, an edge amplitude was used for choosing edges. This is, however, sometimes not sufficient. When, for example, 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` chooses pairs of a certain width,
- `Fuzzy Pair Gray Mean` (page 227), 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, 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 227).

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 222):

- 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 it and modifying the number.

For example 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 them in the `Values` list and then using the `Remove` button. If you want to start over, click 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.

When you have determined all relevant edges, continue to the Results tab (page 222).

Fuzzy Contrast

Enable Fuzzy Contrast to choose edges with specific amplitudes. Then specify your "good" values as described in the section "Specify Good Values" (page 225).

When all relevant edges have been found, continue to the Results tab (page 222).

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 (page 225).

You can learn how to use the Normalized option in the paragraph "Advanced Fuzzy Features".

When all relevant edges have been found, continue to the Results tab (page 222).

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,
- 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 225).

You can learn how to use the Normalized option in the paragraph "Advanced Fuzzy Features".

When all relevant edges have been found, continue to the Results tab (page 222).

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, that is 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.

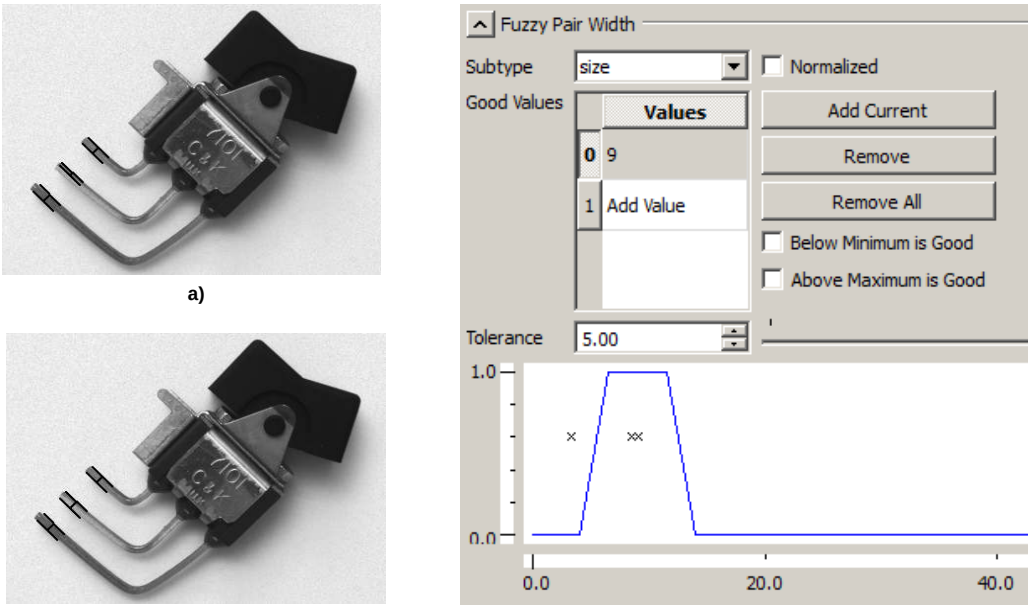


Figure 7.6: How to use fuzzy measuring to improve the edge detection in an image.

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 225).

You can learn how to use the `Normalized` option in the paragraph “Advanced Fuzzy Features”.

When all relevant edges have been found, continue to the `Results` tab (page 222).

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 225).

When all relevant edges have been found, continue to the `Results` tab (page 222).

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 226),
- `Fuzzy Pair Center Position` (page 226), and
- `Fuzzy Pair Width` (page 226)

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, for example, 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 appears 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 224) if necessary, or preview the generated code (page 224) before inserting it into your HDevelop program.

7.4.6 Measure Assistant Reference

The Measure Assistant Reference provides pull-down menus, a tool bar, tabs with dialogs for most of the tasks, and 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 slightly 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`

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 184).

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. 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 (page 228). 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

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 220). 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 220).

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 220). 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 220).

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

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, for example, 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 220)
- `Expected Gray Value Range` (page 224)
- `Calibration Source` (page 223)

7.4.6.6 The Tab `Edges`

The `Edges` tab includes:

- `Edge Extraction` (page 221)
- `Edge Selection` (page 222)
- `Display Parameters` (page 222)

7.4.6.7 The Tab `Fuzzy`

The `Fuzzy` tab includes the following subdivisions:

- `General Options` (page 225)
- `Fuzzy Contrast` (page 226)
- `Fuzzy Edge Position` (page 226)
- `Fuzzy Pair Center Position` (page 226)
- `Fuzzy Pair Width` (page 226)
- `Fuzzy Pair Gray Mean` (page 227)

7.4.6.8 The Tab `Results`

The `Results` tab consists of the following subdivisions:

- `Feature Selection` (page 222)
- `Feature Processing` (page 223)
- `Edge Data` (page 223)

7.4.6.9 The Tab `Code Generation`

The `Code Generation` tab includes the following subdivisions:

- `Measuring` (page 228)
- `Variable Names (General)` (page 224)
- `Variable Names (ROI coordinates)` (page 224)
- `Variable Names (Measurement results)` (page 224)
- `Code Preview` (page 224)

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`,
- segment text by choosing parameters suitable for the characters appearance (page 234)
- choose a pretrained classifier or train your own classifier (page 236),
- choose the kind of results you need (page 240), and
- generate code for your OCR application (page 241).

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` (page 233) and `Region of Interest` (page 233) where you can also load a sample and mark the text that should be read with a rectangle. Then improve the segmentation (page 234) by adapting relevant parameters, choose a pretrained font or performing your own training (page 236) and finally add the resulting code to your application (page 241).

When looking for an overview of all OCR Assistant elements, please refer to the reference (page 242). The general process of an OCR application is visualized in figure 7.7. 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.

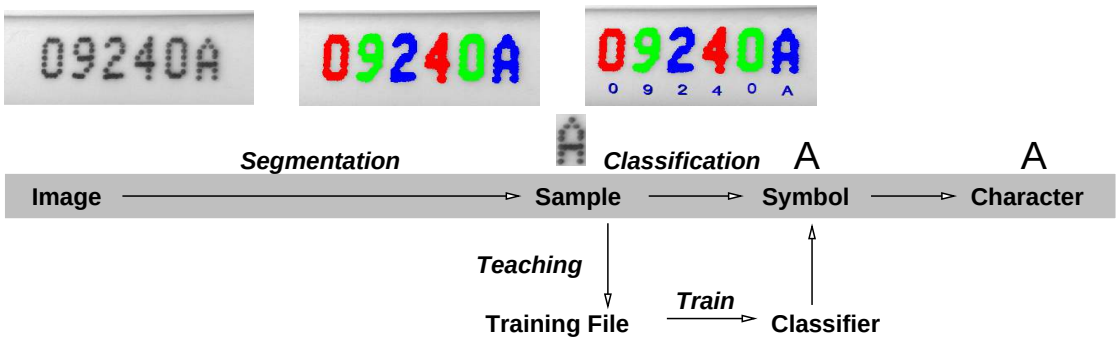


Figure 7.7: Process of an OCR application.

In this online help, the following special terms are used:

Optical Character Recognition (OCR) *Optical Character Recognition* is the technical term for reading and 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 training files. In the second step, these files are input to create a new *OCR classifier*. HALCON provides pretrained *OCR classifiers*, that are 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 (for example 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.

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 (page 234).

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 180) 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. **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 (for example, 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, that is 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 240) 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 180). 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 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.

When you have loaded your image, you can continue to create an ROI.

7.5.2.3 Region of Interest

1. Create a ROI by using the buttons `Draw Axis-aligned Rectangle` or `Draw Rotated Rectangle` in the tool bar or in the drop-down menu `OCR`. You can also choose a shape within the `Region of Interest` section of the `Setup` tab to create a ROI. The shape of the ROI should be chosen according to the shape of the object to be inspected.

2. 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. When you are finished, click the right mouse button, to confirm your choice.

3. 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 keyboard or click 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.

When 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 232) has been completed, if at least one image has been loaded and an ROI has been specified. Adjusting parameter settings is also recommended when using the `Quick Setup` (page 232) to achieve the highest possible character recognition quality.

7.5.3.1 Symbol Appearance

Here three options specifying the way the characters are printed, can be chosen.

- `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, for example, 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

Here three characteristics of a symbols size, the

- maximum width of a single character,
- maximum height of a single character,
- and the `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 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.

- `Interpunctuation` for a text containing punctuation like, for example, 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 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, 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, for example when 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 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.

- `Baseline Tolerance`, sets the allowed amount of deviation from a perfect line, via text field or slider.
- `Max.` sets the `Number of Lines`, and therefore limit the number of text lines that will be searched for, via text field or slider.
- `Line Structure` sets the maximum number of words and characters when searching for lines, for example '3 4-5' indicates a line consisting of three letters, then a space, then 4-5 letters.
- Activate the checkbox of `Eliminate Background Lines` if background lines exist, for example, in certain forms or scanned documents.

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`.

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, for example, 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, for example, 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. That is 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, that is, 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.

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 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 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 86) 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. **Click the button** `Add To Training Data`.
3. **Repeat this process for each sample image.**
4. **Click the button** `Train Now` **in the section** `Training`.

5. When 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 240).

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`.

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 (page 236) 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, 'untrained', 'training', 'trained', '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.

`Pattern Width` and `Pattern Height`

With the parameters `Pattern Width` and `Pattern Height` you set the fixed pattern size which is used for classification, that is, 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, that is, 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, that is 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`, as 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 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, ratio, 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, and create_ocr_class_knn.

7.5.5 Results

The Results tab lets you select features concerning the classification results.

When you are satisfied with the results, proceed to the Code Generation tab to receive the code for your application. Otherwise, go back to adapt your segmentation (page 234) or OCR classifier parameters (page 236).

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, that is 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, like 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 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 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` (page 240).

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.

7.5.6.1 General Options

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.

`Generation Mode` lets you select the task for which the code is generated, that is

- `Text Reading` or
- `OCR Classifier Training`.

Activate `Initialize Acquisition` if you have previously used the Image Acquisition Assistant (page 180) to acquire images and want to integrate the generated code from this assistant in your code. Such an automatic integration of code results, for example, 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 202) 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.

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 when 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.
- When you are finished changing parameters, click the `Insert Code` button on the right side of the `General Options` dialog to generate the code.
- Finally, integrate the code into your HDevelop program.

7.5.7 OCR Assistant Reference

The OCR Assistant provides pull-down menus. Tabs with the dialogs for most of the tasks that can be performed with the Measure Assistant, and 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 slightly 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`

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, click 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 86) 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

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 233). You can also check the ROI data via the button View ROI Data in the Region Of Interest section on the Setup (page 232) tab. Read more about ROI data in the section Create ROI (page 233).

Train Now

Train a classifier for OCR when 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 233). 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

Insert the Generated Code Lines

Via the menu item Insert Code, which is 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 `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, for example, 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 229) 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 `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 232)
- `Image Source` (page 233)
- `Region of Interest` (page 233)

7.5.7.6 The Tab `Segmentation`

The `Segmentation` tab includes:

- `Symbol Appearance` (page 234)
- `Symbol Size` (page 234)
- `Symbol Shape` (page 234)
- `Text Orientation` (page 234)
- `Text Layout` (page 235)
- `Inspection` (page 235)

7.5.7.7 The Tab `OCR Classifier`

The `OCR Classifier` tab includes the following subdivisions:

- `OCR Classifier` (page 236)
- `Teaching` (page 236)
- `Basic Features` (page 237)
- `Advanced Training Parameters and Features` (page 238)

7.5.7.8 The Tab `Results`

The `Results` tab consists of the following subdivisions:

- `Word Processing` (page 240)
- `Feature Selection` (page 240)
- `Display Parameters` (page 241)
- `Results` (page 241)

7.5.7.9 The Tab `Code Generation`

The `Code Generation` tab includes the following subdivisions:

- `General Options` (page 241)
- `Variable Names` (page 242)
- `Code Preview` (page 242)

Chapter 8

HDevelop Language

This chapter introduces the syntax and the semantics of the HDevelop language. It illustrates what can be entered 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)
```

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 of image processing, for example, 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).

The minimum and maximum possible values of an `integer` depend on whether the 64- or 32-bit version of HALCON is used. You can use the operator `tuple_number` to verify that a value fits into the range of the `integer` data type. For a reference, see table 8.1.

	64-bit	32-bit
Minimum value	−9223372036854775808	−2147483648
Maximum value	9223372036854775807	2147483647

Table 8.1: Minimum and maximum values of `integer`.

`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 8.2, 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`. Note that some HALCON operators take logical values for input, for example, `set_system`. In this case, the HALCON operators expect string constants like `'true'` or `'false'` rather than the boolean values `true` or `false`.

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 8.2: Substitutes for special characters.

`handle` Handles are references to complex data structures, for example, 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. HDevelop also supports the variable type `vector`, see section 8.6 on page 270.

constants For the return value (result state) of an operator, constants exist. 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 but there are plenty of error numbers internally, see the Extension Package Programmer's Manual, appendix A on page 105.

In table 8.3, all return messages can be found.

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 8.3: Return values for operators.

Additionally, there are constants for the types of control data, see table 8.4. These can be compared to the result of a `type` operation to react to different types of control data, see section 8.5.5 on page 259.

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 8.4: Type values for control data.

Finally, you can access minimum and maximum values of integers and floating-point numbers via the constants mentioned in table 8.5.

Constant	Meaning
H_FLOAT32_EPSILON	maximal relative approximation error for 32-bit floating-point numbers
H_FLOAT32_MAX	largest possible 32-bit floating-point number
H_FLOAT32_MIN	smallest possible 32-bit floating-point number
H_FLOAT32_MIN_POSITIVE	smallest possible positive floating-point number (32-bit)
H_FLOAT64_EPSILON	maximal relative approximation error for 64-bit floating-point numbers
H_FLOAT64_MAX	largest possible 64-bit floating-point number
H_FLOAT64_MIN	smallest possible 64-bit floating-point number
H_FLOAT64_MIN_POSITIVE	smallest possible positive 64-bit floating-point number
H_FLOAT_INFINITY	∞ (for example: $1.0/0.0$), shown as `inf` in variable window
H_FLOAT_NAN	Not a Number (NaN), shown as `nan` in variable window
H_FLOAT_NEG_INFINITY	$-\infty$ (for example: $-1.0/0.0$), shown as `-inf` in variable window
H_INT32_MAX	largest possible 32-bit integer
H_INT32_MIN	smallest possible 32-bit integer
H_INT64_MAX	largest possible 64-bit integer (error in 32-bit HALCON)
H_INT64_MIN	smallest possible 64-bit integer (error in 32-bit HALCON)
H_INT_MAX	H_INT64_MAX or H_INT32_MAX, depending on HALCON version
H_INT_MIN	H_INT64_MIN or H_INT32_MIN, depending on HALCON version

Table 8.5: Numeric constants.

For further explanation of the numeric constants, see the HDevelop example program `tuple_numeric_limits.hdev`. See also the description of the operator `tuple_constant`.

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.

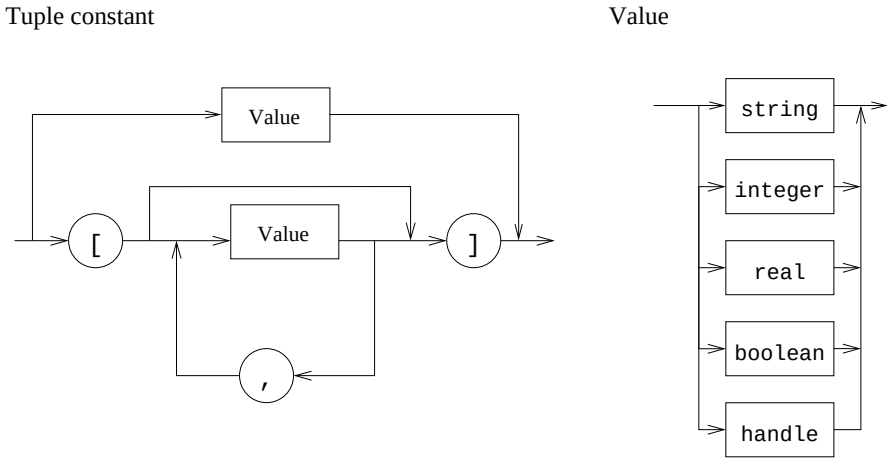


Figure 8.1: Syntax of tuple constants.

`[]` 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]
[100.0,100.0,200.0,200.0]
['FileName','Extension']
[4711,0.815,'Hugo']
```

8.3 Variables

The names of variables are built up 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 247, and 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 is undefined until the first assignment defines it, note that 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 for example, `real`, `integer`, `string` etc., is undefined until a concrete value is assigned to the variable. 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:

- procedure parameter definitions (explicitly)
- global variable declarations (explicitly)
- usage in the code (implicitly)

The order of precedence is as follows:

1. Procedure parameter definitions
2. Global variable declarations
3. Usage in the code

In the latter case, the type of a variable is defined by those code lines, where the value of the variable is used and where the exact type of the new value is known before run time.

Within its scope, a variable must always be properly defined. The following type definition errors are possible:

- A local variable is used, whose type has not been defined before.
  → The error 'The variable never gets initialized' [21089] is thrown. All lines using the variable will become invalid, and the variable will not be displayed in the variable window.
- The type of a variable is defined differently, for example, as iconic variable and later as control variable.
  → The error 'The type of the variable could not be determined (conflicting type definitions found)' [21088] is thrown. All lines using the variable will become invalid, and the variable will not be displayed in the variable window.
- The type of a variable is properly defined, but used in the wrong context, for example, an iconic variable is used as a control input parameter.
  → The error 'The type of the variable could not be determined (conflicting type definitions found)' [21088] is thrown. Only the affected code will become invalid.

8.3.2 Scope of Variables (local or global)

HDevelop supports local and global variables. All variables are local by default. They exist only within their procedure. 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 `global` statement.

When using HDevEngine, a global variable can extend beyond the scope of its program as HDevEngine manages the variables of *all* loaded programs and procedures. This can be used to transfer information between procedures. Each global variable can have only one value at a time for all procedure calls of the running HDevEngine instance. Therefore, take care not to overwrite the value of a variable of one program with that of another.
For more information about HDevEngine, see the Programmer's Guide, chapter 21 on page 137.

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, for example, `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
  ...
  auto_threshold (Image, Regions, 2)
  File := 'fuse'
  read_image(Image, File)
  return()
```

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. 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 8.6: 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 256.

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

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 8.6. 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:

- If one of the tuples is of length 1, all elements of the other tuples are combined with that single value for the chosen operation.

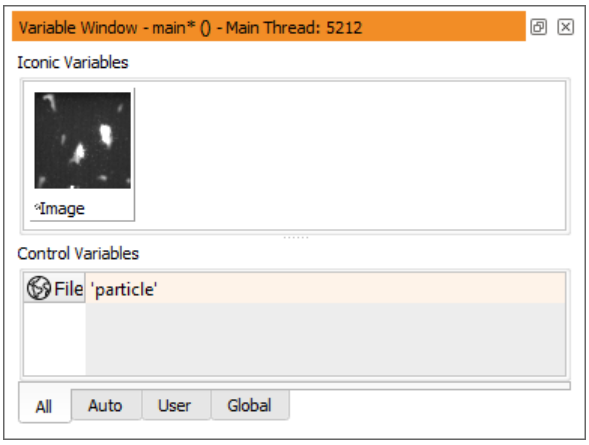


Figure 8.2: Global variables.

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 8.7: Examples for arithmetic operations with tuples and strings.

- If both tuples have a length greater than 1, 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 8.7 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 can either type it directly into the program window or use the operator `assign (Input, Result)`. Should you choose to work with the operator window, the 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);
```

Can be directly typed in the program window as:

```
u := sin(x) + cos(y)
```

Another possibility would be to use the assignment operator:

```
assign (sin(x) + cos(y), u)
```

If the result of the expression does not need to be stored in a variable, the expression can directly be used as input value for any operator, or be typed into the program window. Therefore, an assignment is necessary only if the value has to be used several times or if the variable has to be initialized, for example for a loop.

Modifying tuple elements:

Inside the program window enter:

```
Result[Index] := Value
```

Another possibility would be to use the assignment operator assign_at:

```
assign_at (Index, Value, Result)
```

When using the operator window enter the operator assign_at and fill in the fields.

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)
   ```

   or as in-line operation

   ```
   Result := insert([1,2,3], 1, [11,12,13])
   ```

   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 could be used because the element is appended at the end of the tuple:

```
Areas := [Areas,Area]
```

More examples can be found in the program assign.hdev.

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 8.8 for operations on tuples containing control data.

Operation	Meaning	HALCON operator
t := [t1,t2]	concatenate tuples	tuple_concat
i := |t|	get number of elements of tuple t	tuple_length
v := t[i]	select element i of tuple t	tuple_select
t := t[i1,i2]	select elements i1 and i2 of tuple t	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
t := insert(t1,i,t2)	insert elements from t2 at position i in t1	tuple_insert
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
t := rep(t,i)	repeat t i times	tuple_repeat
t := rep_elem(t,i)	repeat t i times elementwise	tuple_repeat_elem

Table 8.8: Basic operations on tuples (control data) and the corresponding HALCON operators.

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. Note the following: [[t]] = [t] = t.

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 8.9: Equivalent tuple operations for control and iconic data.

`|t|` returns the number of elements of a tuple. The indices of elements range from zero to the number of elements minus one (`|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,Var,[64,128]]`	`[1,2,4,8,16,16,32,64,128]`
`|Var|`	`6`
`Var[4]`	`16`
`Var[4,5]`	`[16,32]`
`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]`

Other examples can be found in the program `tuple.hdev`. The HALCON operators that correspond to the basic tuple operations are listed in table 8.8 on page 256.

Note that these direct operations cannot be used for iconic tuples. Iconic objects cannot be selected from a tuple using `[]` and their number cannot be directly determined using `||`. For this purpose, HALCON operators are offered that carry out the equivalent tasks. In table 8.9 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 248, is the use of constants together with the operator `assign` (or in case of iconic data one of its equivalents shown in table 8.9):

```
assign ([],empty_tuple)
assign (4711,one_integer)
assign ([4711,0.815],two_numbers)
```

[1] In contrast, the index of objects for example, `select_obj`, ranges from 1 to the number of elements.

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 of i from 1^2 to 100^2.

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 8.10: 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_nan_elem(t)`	elementwise test for NaNs	`tuple_is_nan_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 8.11: Elementwise type operations.

8.5.5 Type Operations

The type operations allow you to test or query the value type of control data as shown in table 8.10. See table 8.4 on page 249 for the corresponding type constants.

There are also corresponding operations that test each element of the input tuple as shown in table 8.11.

8.5.6 Basic Arithmetic Operations

See table 8.12 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. Parentheses can change the order of evaluation and some operators have a higher precedence than others, see section 8.5.17.

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 8.12: Basic arithmetic operations.

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:

Expression	Result
`4/3`	1
`4/3.0`	1.3333333
`(4/3) * 2.0`	2.0

Table 8.13: Example Expressions.

Simple examples can be found in the program `arithmetic.hdev`.

8.5.7 Bit Operations

This section describes the operations for bit processing of numbers. The operands have to be integers.

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 8.14: Bit operations.

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.

$ (string conversion)

See also: `tuple_string`.

$ 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:

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
str_replace(s1,s2,s3)	replace substrings of s1 matching s2 with s3	tuple_str_replace
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
join(s1,s2)	join substrings in s1 via separator in s2	tuple_join
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 8.15: String operations.

```
[-+ #]?([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` conversions, this flag has no effect. For `o` conversion, it increases the precision to force the first digit of the result to be a zero. For `x` or `X` conversion, 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 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 for example, '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,i)` returns the characters from the beginning of string `s` up to position `i`.

`str_lastn`

See also: tuple_str_last_n.

`str_lastn(s,i)` returns the character from position `i` of string `s` to the end.

`str_replace`

See also: tuple_str_replace.

`str_replace(s1,s2,s3)` replaces all substrings of `s1` that match `s2` with `s3`.

`{}`

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']
```

`join`

See also: `tuple_join`.

`join(s1,s2)` joins the substrings in `s1`. The separator in `s2` is placed between the substrings. As an example, the result of

```
join(['/usr/image', '/usr/proj/image'], [':', ';'])
```

consists of the two strings

```
['/usr/image:/usr/proj/image', '/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

Table 8.16 shows the provided set operations.

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 8.16: Set operations.

8.5.10 Comparison Operations

Table 8.17 shows the provided comparison operations.

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 8.17: 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 8.18: Examples for the comparison of tuples.

8.5.11 Elementwise Comparison Operations

Table 8.19 shows the provided elementwise comparison operations.

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 8.19: 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 8.20: Examples for the elementwise comparison of tuples.

8.5.12 Boolean Operations

Table 8.21 shows the provided boolean operations.

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 8.21: Boolean operations.

8.5.13 Trigonometric Functions

Table 8.22 shows the provided trigonometric functions.

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 8.22: Trigonometric functions.

8.5.14 Exponential Functions

Table 8.23 shows the provided exponential 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 $\ln(\mathtt{a})$, 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)`	$a1 \cdot 2^{\mathtt{floor(a2)}}$	`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 $erf(\mathtt{a})$	`tuple_erf`
`erfc(a)`	complementary error function $erfc(\mathtt{a})$	`tuple_erfc`

Table 8.23: Exponential functions.

8.5.15 Numerical Functions

Table 8.24 shows the provided numerical functions.

The following example (file name: `euclid_distance.hdev`) shows the use of some 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 Euclidean 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:

$$\langle V1, V2 \rangle = \sum_i (V1_i * V2_i)$$

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 8.24: Numerical functions.

8.5.16 Miscellaneous Functions

Table 8.25 shows the provided miscellaneous functions.

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
constant(s)	value of an HDevelop language constant	tuple_constant
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 8.25: Miscellaneous functions.

8.5.17 Operation Precedence

Table 8.26 shows the provided precedence of the operations for control data.

or
xor
and
bor
bxor
band
== != =~ = #
> < >= <=
[==] [!=] [=] [=] [#]
[>] [<] [>=] [<=]
+ -
* / %
- (unary minus) not
bnot
$
.

Table 8.26: Operation precedence (increasing from top to bottom).

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 (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. 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.

The list of the vector's elements may also be empty, that is, 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:

```
vector02 := vector0
vectorT2 := vectorT
vectorV2 := vectorV

vector02 := {}
vectorT2 := {}
vectorV2 := {}
```

Assigning an empty vector to a vector variable is equivalent to the `.clear()` operation. This means that the assignment of an empty vector to a variable is not sufficient to define the variable's type, see section 8.3.1 on page 251. 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 := vector0.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 position, 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, 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:

```
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 Dictionaries

A dictionary (often also called “dict”) is an associative, array-like container. Similarly to vectors, it can store an arbitrary number of values. In contrast, the values are each associated with unique keys (integers or strings), and each key can refer either to a tuple or to an iconic object.

For example:

```
create_dict (DictHandle)
* Add data.
set_dict_tuple (DictHandle, 'SingleInteger', 27)
set_dict_tuple (DictHandle, 'MixedTuple', ['The answer', 41])
set_dict_object (Image, DictHandle, 'SingleImage')
* Access by key.
get_dict_tuple (DictHandle, 'MixedTuple', Tuple)
* Update value.
RightAnswer := Tuple[1] + 1
set_dict_tuple (DictHandle, 'MixedTuple', ['The answer',RightAnswer])
```

Using Dictionary Init and Access Expressions

Since HALCON 21.11, you can alternatively use a “dot notation”-like syntax instead of the “operator syntax” above. These expressions allow you to create the dictionary and assign values in one go, for example. However, the alternative syntax does not offer additional functionality.

The init and access expressions for dictionaries come in two flavors – “dynamic” or “static syntax”. Due to the broader scope of the dynamic syntax notation, we introduce it first:

Create dict To create a dictionary, use the `dict` init expression.

```
<DictHandle> := dict{['<key>']: <value>}
```

For example:

```
DictHandle := dict{['SingleInteger']: 27, ['MixedTuple']: ['The answer', 42]}
```

Read dict To look up the value associated with a key, use the dict access expression.

```
<DictValue> := <DictHandle>.['<key>']
```

Write dict To assign a value to a key, use a dict access expression. If the key is not present in the dictionary yet, the corresponding key-value pair will be added.

```
<DictHandle>.['<key>'] := <NewDictValue>
```

The following code shows the example from above using dict expressions:

```
* Add data.
DictHandle := dict{['SingleInteger']: 27, ['MixedTuple']: ['The answer',  \
              41]}
read_image (DictHandle.['SingleImage'], 'printer_chip/printer_chip_01')
* Access dict's tuple element.
RightAnswer := DictHandle.['MixedTuple'][1] + 1
* Update dict value.
DictHandle.['MixedTuple'] := ['The answer',RightAnswer]
```

Note that the dict access expression can increase the runtime when handling larger tuples, because the tuple is copied internally. Therefore, instead of repeatedly accessing a tuple in a loop, it is more efficient to access the tuple outside the loop and subsequently iterate over the tuple content.

Simplifying the code using static syntax

Often, you can further simplify the expressions compared to "dynamic syntax" and use "static syntax" instead. To create a dictionary in static syntax notation, use

```
<DictHandle> := dict{<key>: <value>}
```

For example:

```
DictHandle := dict{SingleInteger: 27, MixedTuple: ['The answer', 42]}
```

The example from above now looks like this:

```
* Add data.
DictHandle := dict{SingleInteger: 27, MixedTuple: ['The answer', 41]}
read_image (DictHandle.SingleImage, 'printer_chip/printer_chip_01')
* Access dict's tuple element.
RightAnswer := DictHandle.MixedTuple[1] + 1
* Update dict value.
DictHandle.MixedTuple := ['The answer',RightAnswer]
```

Differences between dynamic and static syntax

Although each of the programs mentioned above leads to the same result, some tasks can only be achieved using dynamic syntax: In case of static syntax, `<key>` is always interpreted as a string. This means that you cannot access the value of a variable. If you want to do this, use the dynamic syntax. In addition, the reserved words as described in section 8.8 cannot be used with the static syntax. For example, if you have a key named `global`, you will have to use the dynamic syntax.

For example, see the following code:

```
ImagePath := 'printer_chip/printer_chip_01'
read_image (Image, ImagePath)
Params := dict{[ImagePath]: 1}
```

Here, `ImagePath` is a variable and used as the key in the dictionary assignment:
`<key>` = '`printer_chip/printer_chip_01`').

We can then extend the example like this:

```
ImagePathPrefix := 'printer_chip/printer_chip_0'

for Id := 1 to 5 by 1
  read_image (Image, ImagePathPrefix + Id)
  Params := dict{[ImagePathPrefix + Id]: Id}
endfor
```

Compare this with the following code using static syntax:

```
ImagePathPrefix := 'printer_chip/printer_chip_0'

for Id := 1 to 5 by 1
  read_image (Image, ImagePathPrefix + Id)
  Params := dict{ImagePathPrefix + Id: Id} // error!
endfor
```

In this case, `ImagePathPrefix` is interpreted as a string and the dictionary `Params` would contain the syntactically wrong key `'ImagePathPrefix' + 1`, while the output of `read_image` is not being used any further.

The following example shows another case requiring dynamic syntax:

```
Keys := ['a', 'b', 'c']

for i := 0 to |Keys| - 1 by 1
  Val := Params.[Keys[i]]
endfor
```

The same could be written with a single expression. When using multiple keys in one dict access or assignment expression, the elements to which the keys refer must all be tuples of length 1.

```
Keys := ['a', 'b', 'c']
Vals := Params.[Keys]
```

An empty list of keys yields an empty tuple.

```
Keys := []
* Vals will always be [] in this case.
Vals := Params.[Keys]
```

Assigning control values to multiple keys can be done in a single expression.

```
* Dict.a == 3 and Dict.b == 3 and Dict.c == 3
Dict.[['a', 'b', 'c']] := 3

* Dict.a == 1 and Dict.b == 2 and Dict.c == 3
Dict.[['a', 'b', 'c']] := [1, 2, 3]

* Dict.a == [] and Dict.b == [] and Dict.c == []
Dict.[['a', 'b', 'c']] := []

* Dict.a == [2, 4] and Dict.b == [1, 'x'] and Dict.c == [3, 7]
* Note that this broadcast assignment only works for even multiples of the number of keys.
Dict.[['a', 'b', 'c']] := [2, 4, 1, 'x', 3, 7]
```

Nesting dictionaries

The `dict` init expression allows arbitrary nesting. You can create further dictionaries within a dictionary. For example:

```
NestedDict := dict{Outer: dict{Inner: [1, 'two', dict{Innermost: 33}]}}
```

Restrictions

Currently, the following restrictions apply in regard to dict expressions:

- Accessing a dict element from a tuple select expression.
  `X := [5, dict{x: 6}][1].x // doesn't work!`
- In a `for` loop, a value of a dictionary cannot be used as index.
  `for MyDict.Index := 0 to 100 by 1 // doesn't work!`

- par_start cannot be used with a dictionary value.
  par_start<T1>: tuple_add(2,2,Dict.Key1) // doesn't work!
- par_start cannot write the thread handle into a dictionary.
  par_start<Dict.ThreadID>: tuple_add(2,2,Four) // doesn't work!

Value copy semantics

The data stored in the dictionary is a full owning copy in case of basic control values. In contrast, handles and iconic objects are copied as a shared reference copy. In particular, note the following:

Objects The copy is a reference, as in copy_obj. In particular, changes made using the operators set_grayval or overpaint_region affect the object stored in the dictionary as well.

Handles Storing any handle in the dictionary will copy the handle value, but not the resource behind the handle. This is illustrated in the following example:

```
create_generic_shape_model (ModelID1)
set_generic_shape_model_param (ModelID1, 'min_score', 0.42)
DictHandle := dict{sbm_model_handle: ModelID1}
ModelID2 := DictHandle.sbm_model_handle
set_generic_shape_model_param (ModelID1, 'min_score', 0.73)
get_generic_shape_model_param (ModelID2, 'min_score', MinScoreModel2)
```

ModelID1 and ModelID2 are both handles for the same model. As a consequence, the retrieved MinScoreModel2 is *0.73*, the value set a line above and not the value set before setting the model handle as dictionary entry.

8.8 Reserved Words

The identifiers listed in table 8.27 on page 278 are reserved words and their usage is strictly limited. Here are some examples where they cannot be used:

- As variable names.
  global := 55 // doesn't work!
- As key in static dictionary expressions.
  X := dict{global: 55} // doesn't work!
- As thread id in par_start.
  par_start<global>: tuple_div(3, 2, Out) // doesn't work!

8.9 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	constant
continue	cos	cosh	cumul
default	deg	deviation	dict
else	elseif	endfor	endif
endswitch	endswitch	endtry	endwhile
environment	erf	erfc	exit
exp	exp10	exp2	export_def
fabs	false	find	find_first
find_last	first_n	floor	fmod
for	gen_tuple_const	global	HNULL
hypot	H_FLOAT32_EPSILON	H_FLOAT32_MAX	H_FLOAT32_MIN
H_FLOAT32_MIN_POSITIVE	H_FLOAT64_EPSILON	H_FLOAT64_MAX	H_FLOAT64_MIN
H_FLOAT64_MIN_POSITIVE	H_FLOAT_INFINITY	H_FLOAT_NAN	H_FLOAT_NEG_INFINITY
H_INT32_MAX	H_INT32_MIN	H_INT64_MAX	H_INT64_MIN
H_INT_MAX	H_INT_MIN	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_nan_elem	is_number
is_real	is_real_elem	is_string	is_string_elem
is_valid_handle	join	last_n	ldexp
lgamma	log	log10	log2
lsh	max	max2	mean
median	min	min2	not
number	or	ord	ords
par_join	par_start	pow	rad
rand	real	regexp_match	regexp_replace
regexp_select	regexp_test	remove	rep
repeat	replace	rep_elem	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	str_distance
str_first_n	str_last_n	str_replace	subset
sum	switch	tan	tanh
tgamma	throw	true	try
type	type_elem	uniq	until
while	xor		

Table 8.27: 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`..

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, *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.

If the `for` loop is left too early for example, 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, for example, 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, and the program is finished.

try ... catch ... endtry This control flow structure enables dynamic exception handling in HDevelop. The program block, runtime errors, between the operators try and catch is watched for exceptions. 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 8.10, and the reference manual, for example, the operator try for detailed information.

throw The operator throw allows you to generate user-defined exceptions.

8.10 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.

8.10.1 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)
...
```

8.10.2 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 (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
...
```

8.11 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 "`General Options` ▷ `General Options`" section 6.16.11 on page 121. 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.

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 8.11.3 on page 286 for information about the lifetime of a thread.

8.11.1 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()
...
```

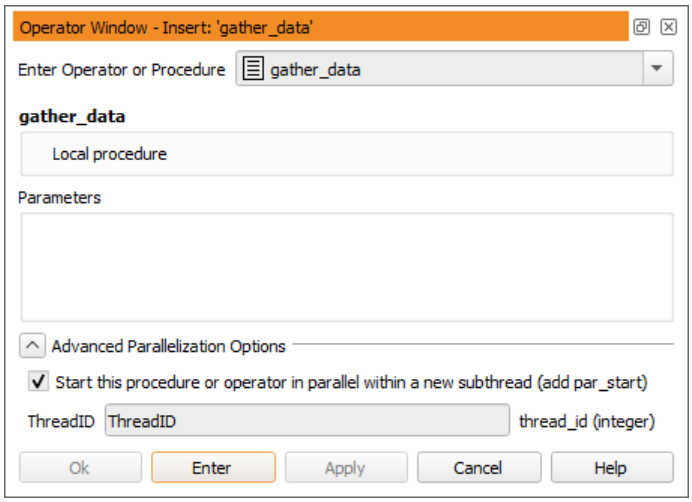


Figure 8.3: Operator window with parallelization options.

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 a program containing the `par_start` qualifier, the parallelization options will also be displayed in the operator window. For certain program lines, 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 6.11 on page 91.

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 270):

```
for Index := 1 to 5 by 1
  par_start <ThreadIDs.at(Index - 1)> : gather_data()
endfor
```

In contrast, **storing thread IDs in dictionaries or messages is not supported.**
For more information about dictionaries, see section 8.7 on page 273. For more information about messages, see the operator `create_message` and the technical note Parallel Programming.

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, it 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 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`.

8.11.2 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
```

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 138 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.

8.11.3 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, for example, by a breakpoint, `stop` instruction, an uncaught exception, or a draw operator.

How to select a specific thread see section 8.11.4. 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. For more information, see section 5.9 on page 48. 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, for example, 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. In the main thread it is not possible to catch an exception that is thrown in a subthread.

8.11.4 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 160)). 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).

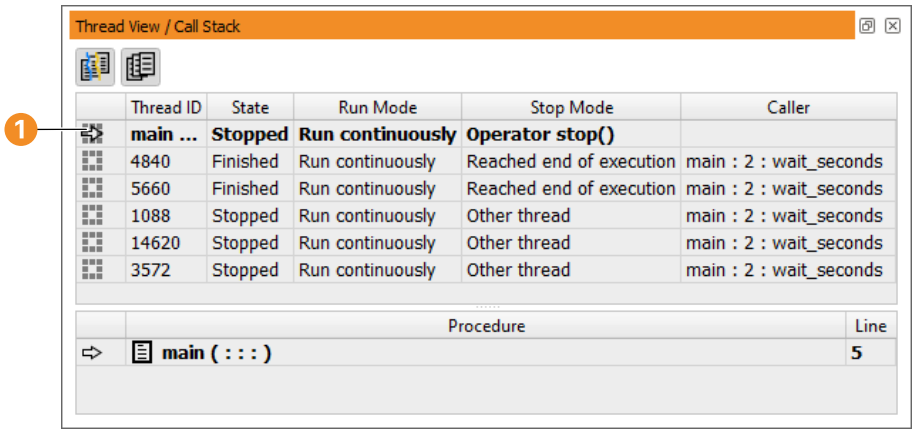


Figure 8.4: Inspecting threads.

1 Currently selected thread

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.

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.

Figure 8.5: Selected subthread in the program window.

1 Notification line

2 Thread view window

3 Switch back to main 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.

8.11.5 Suspending and Resuming Threads

Threads can explicitly be suspended and resumed in the thread view window. Suspending a thread only works between operator calls. If the thread is currently running, the current operator will still be executed before the thread is frozen.

To suspend a thread, right-click 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. 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 the thread entry and select `Resume Thread`.

8.12 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`

insert tuple elements	`insert(t1,i,t2)`	tuple_insert
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
repeat tuple	`rep(s,i)`	tuple_repeat
repeat tuple elements	`rep_elem(s,i)`	tuple_repeat_elem
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
join strings	`join(s1,s2)`	tuple_join
replace strings	`str_replace(s1,s2,s3)`	tuple_str_replace
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`
HDevelop constant	`constant(s)`	`tuple_constant`
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 NaN values (elementwise)	`is_nan_elem(t)`	`tuple_is_nan_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`

8.13 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)
21089 The variable never gets initialized
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 '|' 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 method or key expected after .
21144 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 This expression cannot be used as loop index

21228 Wrong type: The switch statement requires an integer value as parameter

21229 Wrong type: The case statement requires a constant integer value as parameter

21230 At the end of the case and the default statement a colon is expected

21231 Unknown operator or procedure

21232 Qualifier 'par_start' before ':' or '<ThreadID>' expected

21233 '<ThreadID>' variable in angle brackets after 'par_start' expected

21234 ThreadID variable after 'par_start<' expected

21235 Closing angle bracket ('>') after 'par_start<ThreadID' expected

21236 Colon (':') after 'par_start<ThreadID>' expected

21237 Operator or procedure call after 'par_start :' expected

21238 This expression cannot be used as ThreadID

21239 Dictionary expression cannot be used as parameter in par_start call

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 occurred 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

8.14 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. You can find the backup file highlighted in red under your `Recent Programs` list in the start dialog.

When restarting the computer, the locally saved backup files will be deleted.

The exact location of the data depends on the operating system you are using:

Linux `/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 201. 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. If the external application uses HDevEngine XL, you cannot use HDevelop non-XL.

Please note that cross-platform debugging is supported. 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 201.
- Run HDevelop.
- Click `Execute` ▷ `Attach To Process...`.

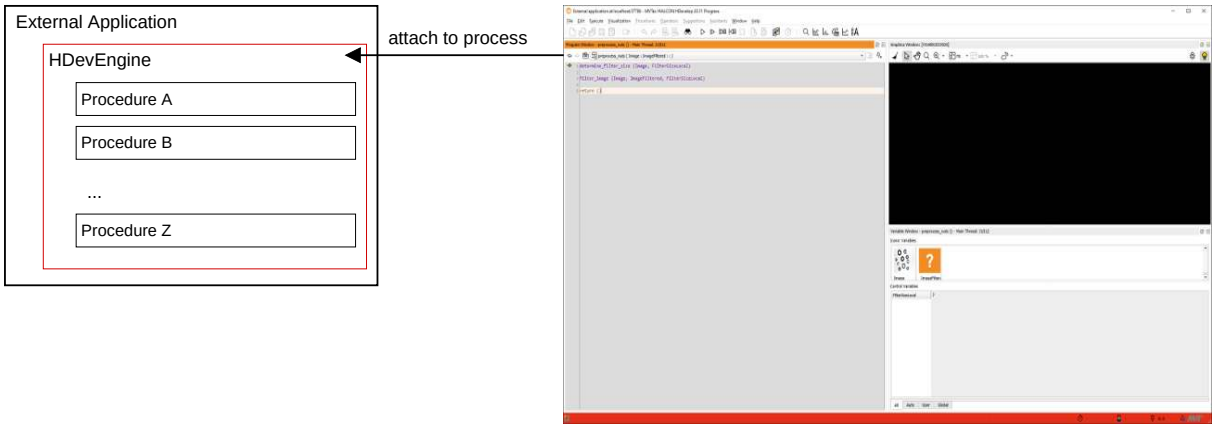


Figure 9.1: Remote debugging.

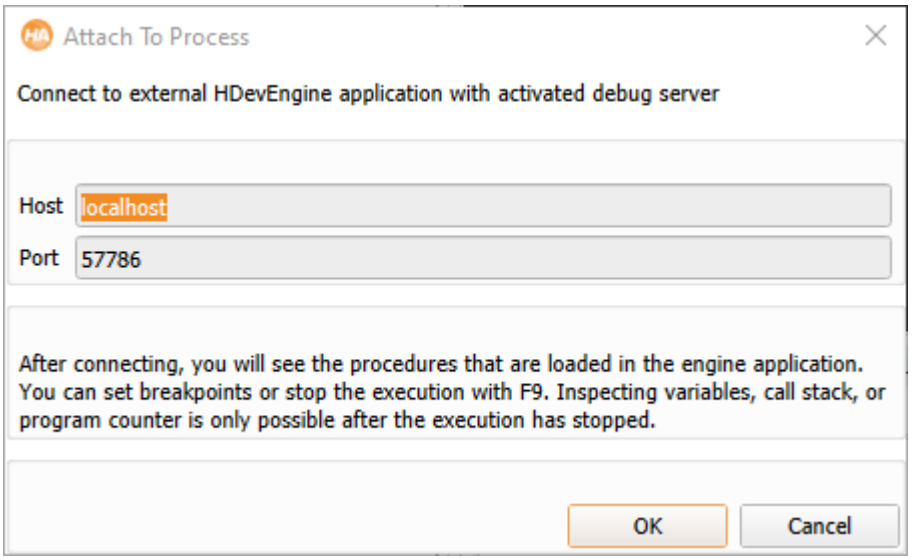


Figure 9.2: Attaching to an external application.

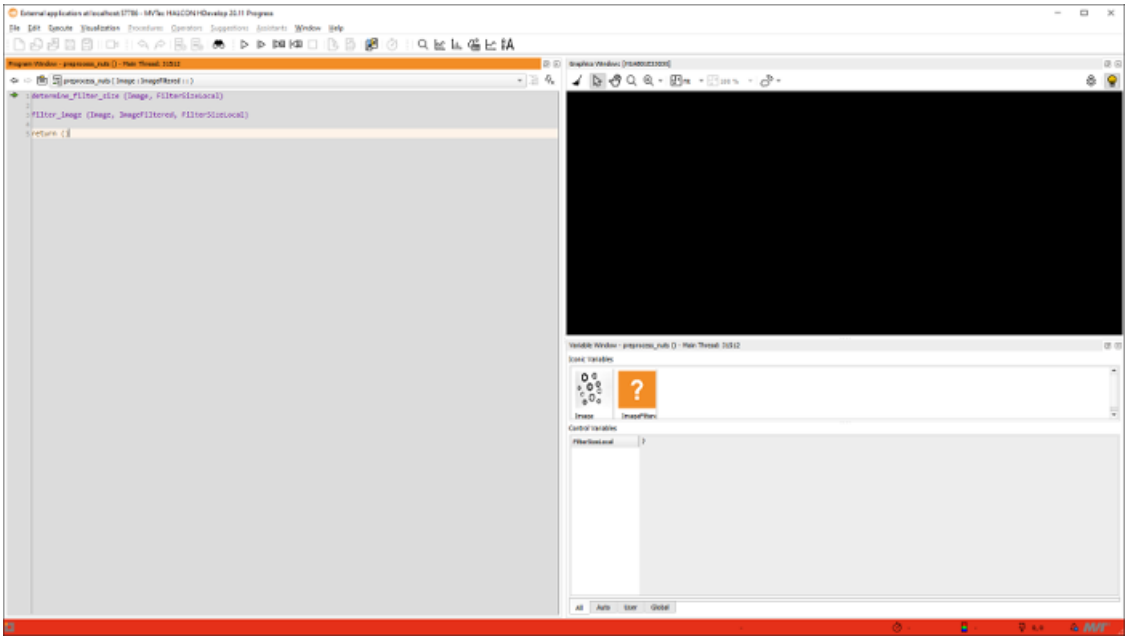


Figure 9.3: HDevelop running in remote debug mode.

- Enter the host name and connection port, and, optionally, the password. See figure 9.2. 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 (for example, `Stop`, `Step Over`, `Step Into` etc.). Debugging is only possible in stopped mode. If HDevEngine is in run mode, the 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 9.9).

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 re-lock 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, for example 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 203.

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 307.

10.1 Exporting Library Projects

See also: `hdevelop -export_project` (command line switch (page 323)).

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. Learn how to use the library project export to integrate HDevelop code into an existing Visual Studio application by watching our tutorial video or by taking an interactive online course at our MVTec Academy.

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.

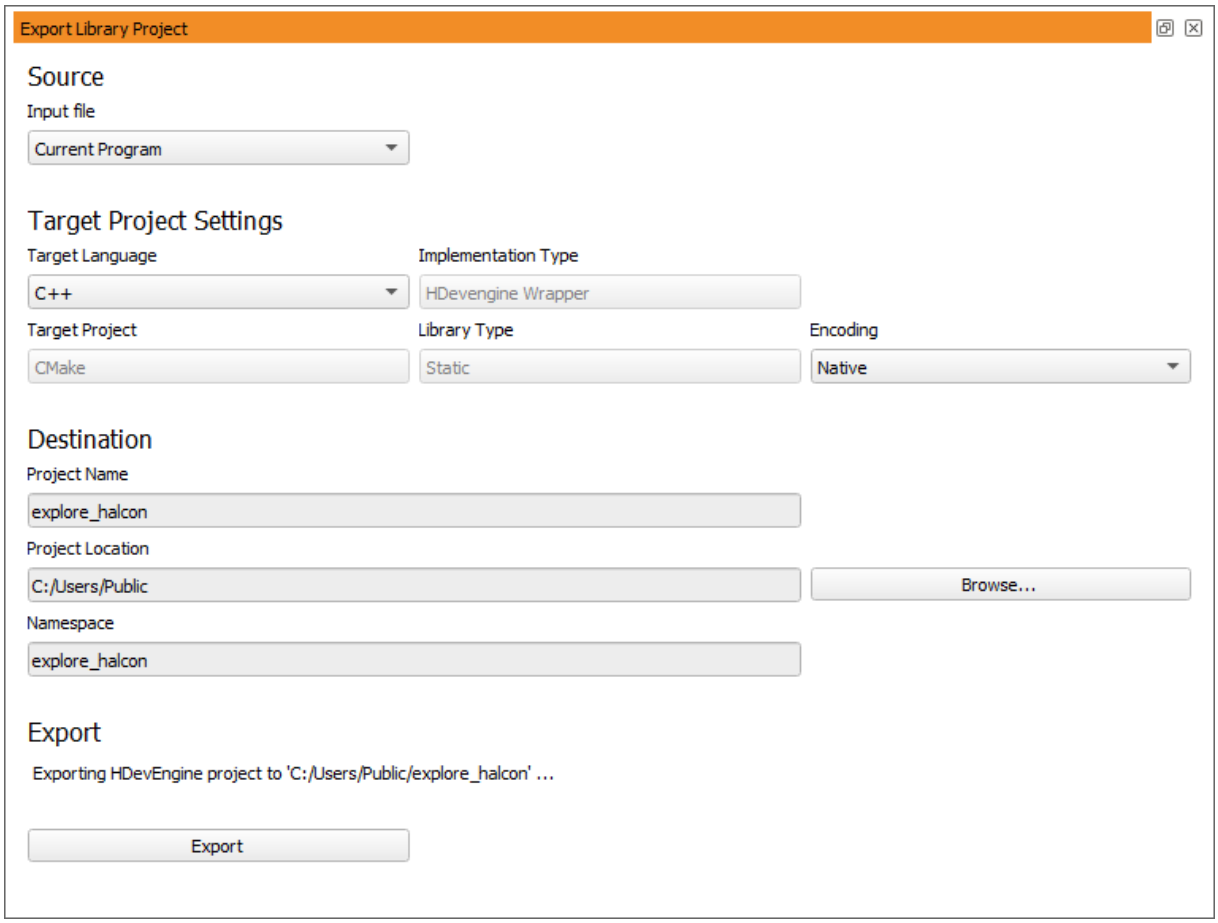


Figure 10.1: Export Library Project dialog.

- 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.

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 (for example, 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`.

Using 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 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, for example,

```
MyNamespace.MyLibrary.my_procedure()
```

Note that the procedures only use parameters of type `HObject`, `HTuple`, `HObjectVector`, and `HTupleVector`.

Using 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 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 including some language-specific details of the code generation and optimization aspects for the following languages:

- C++ (section 10.2.1),
- C# – HALCON/.NET (section 10.2.2 on page 309),
- Visual Basic.NET – HALCON/.NET (section 10.2.3 on page 311),
- C (section 10.2.4 on page 313),

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 314 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

To export the program, use the menu `File` ▷ `Export Program...` and select the language (`C++ - HALCON/C++`). The created file 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.
To exclude the `main()` procedure from compilation only, use the instruction `#define NO_EXPORT_APP_MAIN`. 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

To compile and link the new program (called, for example, `test.cpp`), you can use the example CMake file, which can be found in the directory `%HALCONEXAMPLES%\cpp\console` (Windows) or `$HALCONEXAMPLES/cpp/console` (Linux).

Open the `CMakeLists.txt` in an editor and add lines similar to the following after the `include(UseHalcon)` line:

```
add_executable(test.cpp)
target_link_libraries(test HALCON::CppInt)
```

If you want to build using HALCON XL, set the option `HALCON_XL` to `ON` or `1` during the configuration step. For this, use the following syntax:

```
cmake -DHALCON_XL=1 %HALCONEXAMPLES%/cpp/console
```

To configure the build and create the application, run the following commands:

```
mkdir build
cd build
cmake %HALCONEXAMPLES%/cpp/console
cmake --build .
```

For more information, 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.

- 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.

- Another good way to modify a tuple is the operator `assign_at` (see section 8.5.2 on page 254). 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 314, please also check the description of the HDevelop operators in section 6.1.6 on page 60.

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')` (or `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

To export the program, use the menu `File` ▷ `Export Program...` and select the language (`C# - HALCON/.NET`). The created file with extension "`.cs`" contains the HDevelop program as C# source code.

Using 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 86.

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

If the program has been exported using the option `Use Export Template`, 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.

If the program has been exported using the option `Use HALCON Windows`, the operator `stop` will be ignored as this export is mainly intended to compile and run standalone.

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 314, please also check the description of the HDevelop operators in section 6.1.6 on page 60.

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. For HDevelop programs containing error handling using `dev_set_check('~give_error')` (or `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 (for example, an image) might in fact occupy a large portion of memory. 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

To export the program, use the menu `File` ▷ `Export Program...` and select the language (`Visual Basic.NET - HALCON/.NET`). The created file with extension "`.vb`" contains the HDevelop program as Visual Basic.NET source code.

Using 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 86.

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

If the program has been exported using the option `Use Export Template`, 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.

If the program has been exported using the option `Use HALCON Windows`, the operator `stop` will be ignored as this export is mainly intended to compile and run standalone.

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 314, please also check the description of the HDevelop operators in section 6.1.6 on page 60.

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 315 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. For HDevelop programs containing error handling using `dev_set_check('~give_error')` (or `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 (for example, an image) might in fact occupy a large portion of memory. 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 8.11 on page 283), 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 Program...`. 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

The next step is to compile and link this new program.

To compile and link the new program (called, for example, `test.c`), you can use the example CMake file, which can be found in the directory `%HALCONEXAMPLES%\c` (Windows) or `$HALCONEXAMPLES/c` (Linux).

Open the `CMakeLists.txt` in an editor and add lines similar to the following after the `include(UseHalcon)` line:

```
add_executable(test test.c)
target_link_libraries(test HALCON::CInt)
```

If you want to build using HALCON XL, set the option `HALCON_XL` to `ON` or `1`.

To configure the build and create the application, run the following commands:

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.

```
mkdir build
cd build
cmake %HALCONEXAMPLES%/c
cmake --build .
```

For more details see the Programmer's Guide, chapter 19 on page 125.

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.

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, `x` and `X` are distinct variable names in HDevelop programs. If you export such a program to a case-insensitive target language, for example 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 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 47. 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.

10.2.5.5 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.

10.2.5.6 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 (for example, 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.

10.2.5.7 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.

10.2.5.8 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 24) (same behavior as dragging an image file onto a running instance of HDevelop).

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"
  ```

  Specified path names are put first in the list of directories that are searched for external procedures and libraries. Separate multiple directories by the system-dependent separator (";" on Windows and ":" on Linux). This is a temporary setting for the current session only. See also section 5.4 on page 45.

- 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
```

Command Line Options

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, for example, 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.

D.1 Canvas Window

Function	Shortcut	Alternative
Increase Zoom Factor	Ctrl+Shift++	
Decrease Zoom Factor	Ctrl+Shift+-	

D.2 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

Function	Shortcut	Alternative
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+=
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.3 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 HDevelop Main Window

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	

Function	Shortcut	Alternative
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	
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

Function	Shortcut	Alternative
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+=
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

Function	Shortcut	Alternative
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
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.5 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

Function	Shortcut	Alternative
Detailed View	Ctrl+Shift+V,D	Ctrl+Shift+V,Ctrl+Shift+D
Thumbnails	Ctrl+Shift+V,T	Ctrl+Shift+V,Ctrl+Shift+T

D.6 Plot Windows

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.7 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	

Function	Shortcut	Alternative
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	
Increment text size	Ctrl++	
Decrement 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.8 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	

Index