

## the Power of Machine Vision

# **HDevelop User's Guide**





**Building Vision for Business** 

HDevelop, the interactive development environment of HALCON, Version 8.0.4

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.

Edition	1	July 1997	
Edition	2	November 1997	
Edition	3	March 1998	(HALCON 5.1)
Edition	4	April 1999	(HALCON 5.2)
Edition	5	October 2000	(HALCON 6.0)
Edition	6	June 2002	(HALCON 6.1)
Edition	6a	December 2002	(HALCON 6.1.1)
Edition	7	December 2003	(HALCON 7.0)
Edition	7a	July 2004	(HALCON 7.0.1)
Edition	8	July 2005	(HALCON 7.1)
Edition	8a	April 2006	(HALCON 7.1.1)
Edition	8b	December 2006	(HALCON 7.1.2)
Edition	9	June 2007	(HALCON 8.0)
Edition	9a	October 2007	(HALCON 8.0.1)
Edition	9b	April 2008	(HALCON 8.0.2)
Edition	9c	March 2009	(HALCON 8.0.3)

Copyright © 1997-2010 by MVTec Software GmbH, München, Germany



Microsoft, Windows, Windows NT, Windows 2000, Windows XP, Windows 2003, Windows Vista, and Visual Basic are either trademarks or registered trademarks of Microsoft Corporation.

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

More information about HALCON can be found at:

http://www.halcon.com/

# **About This Manual**

This manual is a guide to HDevelop, the interactive development environment for the HALCON machine vision library. It provides all necessary information to understand HDevelop's basic philosophy and to use HDevelop.

This manual 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. However, 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.

This manual does not assume that you are an expert in image processing. Regardless of your skills, it is quite easy to work with HDevelop. Nevertheless, it is helpful to have an idea about the functionality of *graphical user interfaces*  $(GUI)^1$ , and about some basic image processing aspects.

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 and provides a quick overview of the graphical user interface.

Running Example Programs

This chapter explains how to find and run 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 the procedure of development in HDevelop.

- **Graphical User Interface** This chapter explains the graphical user interface of HDevelop and how to interact with it.
- Matching Assistant This chapter describes how to use the Matching Assistant for shape-based matching.
- HDevelop Language

This chapter explains syntax and semantics of the language used in HDevelop programs.

<sup>&</sup>lt;sup>1</sup>consult your operating system's documentation for general information.

### Code Export

This chapter explains the export of a HDevelop program to C, C++, Visual Basic, Visual Basic .NET, or C#.

## • Tips & Tricks

This chapter describes keycodes, warning and error windows, and restrictions.

# Contents

1	Introducing HDevelop1.1Facts about HDevelop1.2HDevelop Procedures1.3Parallel HDevelop1.4Terminology & Usage	<b>9</b> 10 10 11 11
2	Getting Started	13
3	Running Example Programs	17
4	Acquiring Images with HDevelop         4.1       Reading Images From Files         4.2       Viewing Images         4.3       Image Acquisition Assistant         4.3.1       Acquiring Images From Files or Directories         4.3.2       Acquiring Images Through Image Acquisition Interfaces         4.3.3       Modifying the Generated Code	<b>21</b> 22 23 23 25 29
5	Programming HDevelop5.1Start a New Program5.2Enter an Operator5.3Specify Parameters5.4Getting Help5.5Add Additional Program Lines5.6Understanding the Image Display5.7Inspecting Variables5.8Improving the Threshold Using the Gray Histogram5.9Edit Lines5.10Re-Execute the Program5.11Save the Program5.12Selecting Regions Based on Features5.14Summary	<b>31</b> 31 32 32 33 34 36 37 37 37 38 39 39 39 39 40 42
6	Graphical User Interface         6.1       Main Window	<b>43</b> 43

	6.2	Menu Bar	46
		6.2.1 Menu File	16
		6.2.2 Menu Edit	56
		6.2.3 Menu Execute	13
		6.2.4 Menu Visualization	18
		6.2.5 Menu Procedures	37
		6.2.6 Menu Operators	39
		6.2.7 Menu Suggestions 9	<b>)</b> 4
		6.2.8 Menu Assistants	<b>)</b> 5
		6.2.9 Menu Window	<b>)</b> 7
		6.2.10 Menu Help	<del>)</del> 9
	6.3	Tool Bar	)1
	6.4	Program Window	)2
		6.4.1 The Program Area	)2
		6.4.2 PC, IC, and Break Points	)4
		6.4.3 Creating and Editing Procedures	)5
	6.5	Operator Window	13
		6.5.1 Operator Name Field	4
		6.5.2 Parameter Display	4
		6.5.3 Control Buttons	17
	6.6	Variable Window	17
		6.6.1 Iconic Variables	20
		6.6.2 Control Variables	22
	6.7	Graphics Window	24
	6.8	Help Window	27
	6.9	Zoom Window	31
	6.10	Gray Histogram Window	33
	6.11	Feature Histogram Window	37
	6.12	Feature Inspection Window	38
	6.13	Dialogs	10
		6.13.1 File Selection Dialog	10
		6.13.2 Unsaved Changes	11
7	Mate	hing Assistant 14	13
	7.1	Introducing the Matching Assistant of HDevelop 14	13
	7.2	How to Use the Matching Assistant of HDevelop 14	4
		7.2.1 Creating the Model $\ldots$ $1^2$	14
		7.2.2 Testing the Model $\ldots \ldots \ldots$	4
		7.2.3 Optimizing the Parameters $\ldots \ldots \ldots$	15
	7.3	Matching Assistant Reference	16
		7.3.1 The Menu File	16
		7.3.2 The Menu Create Model and the Tab Model Creation 14	18
		7.3.3 The Menu Use Model and the Tab Model Use	58
		7.3.4 The Menu and Tab Inspect	55
		7.3.5 The Menu and Tab Code Generation	6
		7.3.6 The Menu Help	58

8	HDe	velop Language 169
	8.1	Basic Types of Parameters
	8.2	Control Types and Constants
	8.3	Variables
	8.4	Operations on Iconic Objects
	8.5	Expressions for Input Control Parameters
		8.5.1 General Features of Tuple Operations
		8.5.2 Assignment
		8.5.3 Basic Tuple Operations
		8.5.4 Tuple Creation
		8.5.5 Simple Arithmetic Operations
		8.5.6 Bit Operations
		8.5.7 String Operations
		8.5.8 Comparison Operations
		8.5.9 Boolean Operations
		8.5.10 Trigonometric Functions
		8.5.11 Exponential Functions
		8.5.12 Numerical Functions
		8.5.13 Miscellaneous Functions
		8.5.14 Operation Precedence
	8.6	Reserved Words
	8.7	Control Flow Operators
	8.8	Limitations
9	Cod	e Export 199
	9.1	Code Generation for C++
		9.1.1 Basic Steps
		9.1.2 Optimization
		9.1.3 Used Classes
		9.1.4 Limitations and Troubleshooting
	9.2	Code Generation for C# (HALCON/.NET)
		9.2.1 Basic Steps
		9.2.2 Program Structure
		9.2.3 Limitations and Troubleshooting
	9.3	Code Generation for C# (HALCON/COM) 205
		9.3.1 Basic Steps
		9.3.2 Program Structure
		9.3.3 Limitations and Troubleshooting
	9.4	Code Generation for Visual Basic .NET (HALCON/.NET)
		9.4.1 Basic Steps
		9.4.2 Program Structure
		9.4.3 Limitations and Troubleshooting
	9.5	
		Code Generation for Visual Basic .NET (HALCON/COM) 210
		Code Generation for Visual Basic .NET (HALCON/COM)
		Code Generation for Visual Basic .NET (HALCON/COM)2109.5.1Basic Steps2109.5.2Program Structure211
		Code Generation for Visual Basic .NET (HALCON/COM)2109.5.1Basic Steps2109.5.2Program Structure2119.5.3Limitations and Troubleshooting212
	9.6	Code Generation for Visual Basic .NET (HALCON/COM)2109.5.1Basic Steps2109.5.2Program Structure2119.5.3Limitations and Troubleshooting212Code Generation for Visual Basic 6 (HALCON/COM)213

		9.6.1	Basic Steps	. 213
		9.6.2	Program Structure	. 213
		9.6.3	Limitations and Troubleshooting	. 215
	9.7	Code (	Generation for C	. 215
		9.7.1	Basic Steps	. 215
	9.8	Genera	al Aspects of Code Generation	. 217
		9.8.1	User-Defined Code Blocks	. 217
		9.8.2	Assignment	. 217
		9.8.3	'for' Loops	. 218
		9.8.4	Protected External Procedures	. 218
		9.8.5	System Parameters	. 219
		9.8.6	Graphics Windows	. 219
10	Tips	& Tric	ks	221
	10.1	Keyco	des	. 221
	10.2	Online	Help	. 221
	10.3	Warnir	ng and Error Windows	. 221
A	Glos	sary		225
B	Com	mand l	Line Switches	227
Inc	lex			229

## 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 sophisticated machine vision package suitable for product development, research, and education.

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

- *Rapid prototyping in the interactive environment HDevelop.* You can use HDevelop to find the optimal operators or parameters to solve your image analysis task, and then build the application using various programming languages, e.g., C, C++, C#, Visual Basic .NET, or Delphi.
- 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.
- *Execution of HDevelop programs or procedures using HDevEngine.* You can directly execute HDevelop programs or procedures from an application written in a programming language using HDevEngine. This is described in detail in the Programmer's Guide, part VI on page 161.
- *Export of an application as C, C++, Visual Basic, Visual Basic .NET, or C# source code.* Finally, you can export an application developed in HDevelop as C, C++, Visual Basic, 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 stand-alone (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

While developing programs, HDevelop actively supports the user in different ways:

- □ With the graphical user interface of HDevelop, operators and iconic objects can be directly selected, analyzed, and changed within one 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 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 conditions. 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 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.
- □ Variables with an automatic garbage collection are used to manage iconic objects or control values.

## 1.2 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 separate procedures.

A HDevelop procedure consists of an interface and a program body. Procedure interfaces resemble the interfaces of HALCON operators, i.e. they contain parameter lists for iconic and control input and output parameters. A procedure body contains a list of operator and procedure calls.

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.

Local and external procedures are differentiated in HDevelop. Local procedures are stored inside the HDevelop program while external procedures are stored separately and therefore can be shared between different HDevelop programs and, what is most advantageous, the modification of an external procedure immediatly affects all HDevelop programs using it. In order to manage a large collection of procedures, the procedures can be ordered in a hierarchical way, i.e., similar to the storing of operators thematically ordered chapters and sections can be composed. Furthermore, external procedures can be protected by passwords, so that they can be applied but not viewed or modified by unauthorized users.

## 1.3 Parallel HDevelop

In addition to the standard HDevelop, there is also a variant called Parallel HDevelop which is based on Parallel HALCON. The user interface is identical, but underneath Parallel HALCON automatically parallelizes operators if used on multi-processor or multi-core hardware. See the Programmer's Guide for more information. In the remainder of this document, when we refer to HDevelop you can substitute Parallel HDevelop if that is the variant you will be using.

## 1.4 Terminology & Usage

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

### Mouse Usage

- **click** A single click with the left mouse button, e.g., to mark and select items or to activate buttons. To select multiple items, hold down the <Ctrl> key and click the desired items. To select many items from a list, click the first item, hold down the <Shift> key and click the last item. All intermediate items are then also selected.
- **double-click** Two quick successive clicks with the left mouse button, e.g., to open dialogs of selected items. Double-clicks are mostly shortcuts for single clicks followed by an additional action.
- **right-click** A single click with the right mouse button to access additional functionality of the user interface, e.g., context-sensitive menus. Clicking the right mouse button also ends interactive drawing functions in HDevelop.
- **drag** Keeping the left mouse button pressed while moving the mouse and finally releasing the mouse button. Typically used to move items, resize windows, select multiple items at once, e.g., program lines, or to draw shapes.
- **drag-and-drop** HDevelop supports drag-and-drop of image files and HDevelop programs from other applications. You can, e.g., drag an HDevelop program icon from a file browser and drop it on the HDevelop window to load it.

12

- **middle mouse button** With three-button mice, the middle mouse button is mostly used under UNIX to paste text which has previously been selected into text fields. Otherwise, it provides the same functionality as the left mouse button or none at all.
- **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 be easily done using just the keyboard. The most common keyboard functions are listed in the appendix.

#### Windows

In the default window mode of HDevelop, windows can be freely moved inside the main window by dragging the title bar. They can be resized by dragging the window border. Windows can be focused by clicking inside the window area. This also raises the corresponding window to the front. Windows that are completely covered by other windows can be brought to the front by selecting them from the Window menu. The window title provides some buttons with additional functionality. Clicking the icon in the left edge of the window title opens a menu from which all window management functions (move, resize, minimize...) can be selected. The buttons on the right edge of the window to 1) minimize/restore, 2) maximize/restore, and 3) close the corresponding window (from left to right).

There is an alternative window mode called SDI (see also section 6.1 on page 45) which delegates the functionality of the window title to the window manager.

#### Abbreviations

BP break point

IC insert cursor

MDI multi-document interface

PC program counter

SDI single-document interface

**XLD** extended line description (see also chapter A on page 225)

## **Chapter 2**

# **Getting Started**

In this chapter the following topics are covered:

- running HDevelop
- specifying command line switches
- short introduction to the windows of 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 Windows "Start" menu:

Start ▷ Programs ▷ MVTec HALCON ▷ HDevelop

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

hdevelop

#### UNIX

Under UNIX, HDevelop is started from the shell:

hdevelop &





## **Command Line Switches**

Under both Windows and UNIX, HDevelop supports several command line switches to modify its startup behavior. You can add the 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. 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 B on page 227 for a listing of the available switches.

### **User Interface**

When HDevelop is started for the first time it looks similar to figure 2.1. The main window offers a menu and a tool bar for quick access to frequently used functions. The status bar at the bottom of the window displays messages and image properties. In addition, the following windows are available by default:

1. Graphics window

This window displays iconic data: images, regions, and XLDs. It provides its own tool bar to quickly zoom and pan the displayed image and a context menu to adapt the visualization settings. The context menu is available by right-clicking inside the window<sup>1</sup> It contains the most frequently used entries from the menu Visualization. You can open multiple graphics windows. The one marked with a lit bulb in the upper right corner is the active graphics window, i.e., it is the target for subsequent display operations. The graphics window works like an image stack: Images can be overlayed with regions or XLDs, or with images that have a reduced domain. The image stack and thus the graphics window can be explicitly cleared and rebuilt.

2. Operator window

You can select HALCON operators (and HDevelop procedures) in this window. The parameters of the selected operator can be specified, and the operator can be executed, entered in the current program, or both. You can also get online help for the selected operator from this window.

3. Program window

This window displays the current program. Program editing in HDevelop is line-based. Program lines can be modified by double-clicking them and editing them in the operator window. 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 break points in the current program in this column. These will halt the program execution at user-defined places.

4. 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. You can double-click iconic variables to display them in the active graphics window. Double-clicking on control variables spawns an inspection window with a list of the current values and statistical data.

There are many other windows which will be covered later in this manual.

<sup>&</sup>lt;sup>1</sup>Unless the context menu has been disabled in the preferences to prevent any interference with interactive drawing functions. See section 6.2.2.11 on page 62.

## **Chapter 3**

## **Running Example Programs**

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

One of the examples demonstrates many different capabilities of HALCON in one program. It is the only HDevelop example program that can be started from the "Start" menu under Windows. The UNIX path to this program is \$HALCONROOT/examples/hdevelop/explore\_halcon.dev. Running this program is very recommended to get a good overview of the many application areas 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.

V Open Example					
Category		Filter measure			
Pharmaceutical		Example	Short Description		
Precision engineering a Printing		fuzzy_measure_pin.dev	Measure pins of an IC using fuzzy measuring		
Rubber, synthetic mat		measure_ic_leads.dev	Measure leads of an IC		
Surveillance and security		measure_pin.dev	Measure pins of an IC		
Transport Wood and timber		pm_measure_board.dev	Locate IC on a board and measure pin distances		
····· industry.pack ····· Method ···· Operator ···· New in version	_				
I Keep dialog open		1)	Load In new HDevelop		

Figure 3.1: Open Example dialog.

17

### **Browse and Load Example Programs**

• Click File ▷ Open Example....

This will open the example program browser (see figure 3.1 on page 17). Similar to a file browser, it shows a tree of categories on the left and a list of example programs from the selected categories on the right. Categories that contain hidden subtopics are marked with a  $\boxplus$ . Double-click on a category label to open the subtopics (or click  $\boxplus$ ). Double-click again to close the subtopics (or click  $\boxdot$ ).

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

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

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

- Click on  $\boxplus$  next to Industry.
- Click on the subtopic Semiconductors. The examples belonging to the semiconductor industry are listed on the right.
- Enter the word measure into the Filter text field.

Note how the list is updated as you type. Now, you have a short list of example programs to select from. You may need to resize the example browser to fully read the short descriptions of the listed programs.

- Select measure\_ic\_leads.dev by clicking on it.
- Click Load. The selected example program is then loaded. Alternatively, you can load an example program by double-clicking on it. The example browser is closed unless Keep dialog open is checked.

The program lines of the loaded example program are now displayed in the program window. The PC is set to the first executable line of the program (leading comments are ignored). The variable window is also updated: It lists the variables that are used in the main procedure, which is initially the current procedure. The variables are currently uninstantiated, i.e., their current value is undefined. This is indicated by the question mark (?). Both windows are displayed in figure 3.2.

iconic variables control va	Iriables PC (program counte	r) current procedure
💥 Variable View - main (:::)	💷 🗵 🔣 Prog	aram Listing 📃 🗆 🗶
Iconic Variables		ain (:::) • Edit Interface
Image Rectangle	1 2 3 4 5	<pre>* Lead Measurement: Example for the application of t * including a lot of visualization operators. * First, the length of the leads is measured, then, *</pre>
Control Variables Rectangle - not (yet) initialized		* First, read in the image and initialize the progra- read_image (Image, 'ic_pin') get_image_pointer1 (Image, Pointer, Type, Width, He:
Width ?	10	p_reopen_window_Iit (0, 0, width, Height, 509, windo p_set_font (WindowHandle) dev display (Image)
Height ? WindowHandle ?	12	dev_set_draw ('margin') dev_set_line_width (3)
Row ?	14 15 16	* Define and display the rectangular ROIs within wh: $E_{\rm QW}$ = 55
All Auto User		

Figure 3.2: The variable and program window after loading the example program.

#### **Run Example Program**

• Click Execute > Run or click the corresponding button from the tool bar (see figure 3.3).

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  $\texttt{Execute} \triangleright \texttt{Run}$  again until the end of the program is reached.

- Click Reset Program Execution to reset the program to its initial state. (see figure 3.3).
- Using the button Step Over you can execute the program line by line and inspect the immediate effect of each instruction.



Figure 3.3: The basic execution buttons.

## **Chapter 4**

# **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.

## 4.1 Reading Images From Files

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

### **Drag-and-Drop**

The easiest way to read an image is to simply drag it from a file browser to the HDevelop window and drop it there. When the file is dropped, HDevelop opens the dialog Read Image (see figure 4.1).

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

Furthermore, a preview of the image and basic image properties are displayed in the dialog (width, height, color type, and number of channels). If you picked the wrong image, you can select another one from the same directory by pressing the button next to the file name. This will open a file browser native to the operating system, i.e., on Windows you may be able to switch to thumbnail view in this dialog. When another image is selected, the dialog is updated accordingly.

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



Figure 4.1: After dragging an image file onto the HDevelop window.

#### **Images from Selected Directories**

Apart from dragging and dropping images, there is an equivalent method from within HDevelop: Select File  $\triangleright$  Read Image to get a list of image directories to choose images from. Again, this will open a native file selection dialog. Browse to and select the desired image from there, and click OK to open up the dialog Read Image described above.

## 4.2 Viewing Images

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

Initially, the loaded image fills the graphics window entirely. The window itself is not resized so the aspect ratio of the image might be skewed. Using the tool box of the graphics window you can easily zoom the image, or change the window size with regard to the image.

We recommend to adapt the window size to the size of the image because otherwise the display is slowed down. The image size, the window size and the displayed part of the image are set with the tool bar icons of the graphics window (see figure 4.2).

An iconic view of the loaded image is also displayed in the variable window. When the image is cleared in the graphics window, it can always be restored by double-clicking this icon.



Figure 4.2: Tools in the graphics window.

## 4.3 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 4.3. 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.

Mage Acquisition : Image Acquisition 01	
File Acquisition Code Generation Help	
Source Connection Parameters Code Generation	
• Image Eile(s)	Recursive
Select File(s)	Select <u>D</u> irectory
C Image Acquisition Interface	
File	Detect
	0.00ms

Figure 4.3: 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, e.g., a camera. This is described in section 4.3.2 on page 25.

## 4.3.1 Acquiring Images From Files or Directories

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

23

#### 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 4.4). 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 Acquisition.





#### **Generate Code**

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

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

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

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

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

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

Source Conner	tion Parameters Code Gen	eration						
Acquisition								
Control <u>Fl</u> ow	Control Elow Acquire Images in Loop 🔽 Insert Code							
Acquisition Mode	Asynchronous Acquisition	[		Auto Dis	connect			
Variable Names								
Connection <u>H</u> andle	AcqHandle	Loop	Counter 🛛	index				
Image Object Image			Image Files ImageFiles					
Code Preview								
Insert Operator Procedure Line Replace Operato								
* Code generated by Image Acquisition 01								
ImageFiles := []								
ImageFiles[0] := 'C:/Program Files/MVTec/HALCON/images								

Figure 4.5: Specifying variable names and previewing the code.

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

## 4.3.2 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 Detect. HDevelop will then probe all the image acquisition interfaces and remove those that do not respond. Probing the interfaces is potentially dangerous. If there are unsaved changes in the current program, HDevelop will display a corresponding dialog. You can save the changes and proceed, ignore the warning and proceed, or abort the operation. After the interfaces have been probed, you can select the desired image acquisition interface from the list.

25

Source Connection Parameters Code Gen	eration	
C Image File(s)	Recursive	
Select File(s)	Select Directory	
Image Acquisition Interface		detect image acquisition devices
uEye	Detect 🖌	
		and select from the list

Figure 4.6: Source selection (example).

#### **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 4.7). 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).

Mage Acquisition : Image Acquisition 01	
File Acquisition Code Generation Help	
🖻 🗔 📲 👼 🖷 🛠 🙎	
Source Connection Parameters Code Generation	
Interface Library hAcqGigEVision.dll (Rev. 4.0 (gtl 1.0.3.1))	
Device PicSightP52 Port	0 🔻
Camera File default	
Trigger	Select
Resolution X Full  Y Full  Color Space	default 💌
Eield progressive Bit Depth	-1
Connect Snap Live Detect	Reset <u>A</u> ll
Display Image Normal 💌 🗆 Show frames per second during live acqu	isition
	9 85.5 ms

Figure 4.7: Connection parameters (example).

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 4.4 on page 24).

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

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

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

### **Set Device Parameters**

The tab card Parameters contains a list of parameters specific to the selected device. It becomes available once the connection to the device has been activated. See figure 4.8 for an example parameter list. Please refer to the HTML page of the selected interface for detailed information. You can click the help button of the assistant to get to the corresponding page automatically.

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

If Update Image 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.

V Image Acquisition : Image	Acquisition 01	_OX	
	I <del>⊈</del>   ?		
Source Connection	Parameters Code Generation ye.dll (3.2)	<u>R</u> efresh Reset <u>A</u> ll	<ul> <li>acquire new image at parameter change</li> </ul>
Category All Parameters	Visibility Beginner	Sort by Name	selection of parameters (optional) and sorting
digital_output edge_enhancement	disable		<ul> <li>reset parameter</li> </ul>
eeprom_content	99.92		<ul> <li>set parameter automatically</li> </ul>
external_trigger	False		
hip_vertical			
gain_boost gain_master			
gamma		1 104.9 ms	

Figure 4.8: Device-specific parameters (example).

### **Generate Code**

28

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

Control Flow

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

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

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

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

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

🐙 Image Acquisition : Image Acquisition 01							
Eile Acguisition Code Generation Help							
🗁 🔒 🕌 🖣	🖗 📧 🗬						
Source Connection Parameters Code Generation							
Acquisition							
Control <u>Fl</u> ow	Acquire Images in Loop 💽				Insert <u>C</u> ode		
Acquisition <u>M</u> ode	Asynchronous	Acquis	sition 💌		Auto Disconnect		
¥ariable Names							
Connection <u>H</u> andle	AcqHandle			Loop Counter	Index	_	
Image <u>O</u> bject	Image			Image Files	ImageFiles	_	
A Code Preview							
Insert Operator	Procedure	Line			Replace Operator		
* Code generate	main	1	* Code generated	by Image Acq	uisition 01	-	
open_framegrab	main	2	open_framegrabb	open_framegrabber ('uEye', 1, 1, 0, 0, 0, 0, 'interlaced', 8, 'default', -			
set_framegrabbe	main	3	set_framegrabber_param (AcqHandle, 'exposure', 99.9248)				
grab_image_star	main	4	grab_image_start (AcqHandle, -1)				
while (true)	main	5	while (true)				
grab_image_asy	main	6	grab_image_as	ync (Image, Ac	:qHandle, -1)		
					4 14	4.6 ms	

Figure 4.9: Code generation.

```
* Code generated by Image Acquisition 01
open_framegrabber ('uEye', 1, 1, 0, 0, 0, 0, 0, 'interlaced', 8, 'default', -1,
    'false', 'UI154x-M', '1', 0, -1, AcqHandle)
set_framegrabber_param (AcqHandle, 'exposure', 99.9248)
grab_image_start (AcqHandle, -1)
while (true)
    grab_image_async (Image, AcqHandle, -1)
* Do something
endwhile
close_framegrabber (AcqHandle)
```

## 4.3.3 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  $\triangleright$  Exit Assistant. If you close the assistant using the menu entry File  $\triangleright$  Close Dialog or using the close icon of the window, the assistant can be restored from the top of the menu Assistants.

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

## **Chapter 5**

# **Programming HDevelop**

This chapter explains how to use HDevelop to develop your own machine vision applications. It is meant to be actively followed in a running instance of HDevelop. In the following, it is assumed that the preferences of HDevelop are set to the default values. This is always the case after a fresh installation of HALCON. If you are uncertain about the current settings, you can always start HDevelop with the default settings by invoking it from the command line in the following way (see also chapter 2 on page 13):

```
hdevelop -reset_preferences
```

We begin with a very simple example. Given is the image displayed in figure 5.1. The objective is to count the clips and determine their orientation.



Figure 5.1: Paper clips.

## 5.1 Start a New Program

Start HDevelop or, if it is still running, click File > New Program to start a new program. HDevelop will notify you if there are unsaved changes in the current program. If it does, click Discard to throw away the changes and start anew. In case you rearranged the windows, click Window > Organize Windows to restore the default layout displayed in figure 2.1 on page 14.

32

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

## 5.2 Enter an Operator

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

🙀 Operator Window - Insert:	'read_cam_par'	-DX
Enter Operator or Procedure	read_cam_par	-
Parameters	read_data_code_2d_model	-
	read_funct_1d	
ComPorFile	read_gray_se	
	read_kalman	
🖶 CamParam CamPar	read_ncc_model	
	read_object_hodel_od_dxi	
_read_ocr_class_mlp		
	Apply Cancel	Help

Figure 5.2: Matching operators after typing read\_ and pressing <Return>.

## 5.3 Specify Parameters

After selecting an operator, its parameters are displayed in the operator window. They are grouped by iconic and control parameters. The icons next to the parameter names denote the parameter type: Input vs. output (see figure 5.3). 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 or press <Return> to submit the operator to the current program. This will do the following:

	💥 Operator Window - Insert: 'read_image'		
	Enter Operator or Procedure read_image	•	
	Parameters		
iconic output -	🕨 🖶 Image Image	🔹 image 🗲	<ul> <li>semantic type</li> </ul>
			– data type
control input —	▶ 🖻 FileName ['fabrik'	Filename.read (string)	
		▶	<ul> <li>file selection dialog</li> </ul>
			-
	Ok Enter Apply	Cancel Help	

Figure 5.3: Specifying parameters.

- 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 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 character \* is added to the window title to indicate unsaved changes in the current program.
- The image is displayed in the graphics window.
- The status bar is updated, i.e., the execution time of the operator **read\_image** is displayed and the format of the loaded image is reported.
- The output variable Clip is created and displayed in the variable window.
- The operator window is cleared and ready for the insertion of the next operator.

## 5.4 Getting Help

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

Double-click the first program line in the program window. The operator is displayed in the operator window for editing. Now click Help to open the HDevelop online help window. It will automatically jump to the documentation of the displayed operator (see figure 5.4). The reference manual is completely cross-linked. The navigation at the left part of the window provides quick access to the documentation. The tab card Contents presents the hierarchical structure of the reference manual (plus access to other HALCON manuals). The tab card Operators lists all operators for direct access. Enter any desired substring into Filter to quickly find an operator.

In the remainder of this chapter, try to use the online help as much as possible to get information about the used operators. The online help window is described in section 6.8 on page 127.



Figure 5.4: The online help window showing the documentation of the operator read\_image.

## 5.5 Add Additional Program Lines

#### Select the clips by thresholding

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

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

In figure 5.5 you can see that the input parameter Image is set to Clip automatically. For input variables with no default value, reasonable suggestions are inferred automatically by collecting previous output variables of the same type. Therefore, the name of the most recent matching output parameter will be suggested (most recent being the closest predecessor of the current program line). In this example, only Clip is available.

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

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

35

Second Contractor				-OX
Enter Operator o	or Procedure thresho	ld		•
Parameters				
<u>R</u>				
🞽 Image	Clip			• image
🐺 Region	Region			region
🖄 MinGray	128			number (real)
MaxGray	255			number (real)
Ok	Enter	Apply	Cancel	Help

Figure 5.5: Parameter guessing.

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

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

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

#### Successor

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

Again, the variable names suggested by HDevelop look reasonable, so press <Return>. This is equivalent to clicking the OK button (though this can be changed in the preferences of HDevelop). This time, two program lines are executed: The threshold operation and the connection operation. As noted above: Clicking OK executes from the PC to the IC.

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

## 5.6 Understanding the Image Display

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

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

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

A more convenient way to set many display properties at once is available through the menu entry Set Parameters.... It opens the settings window displayed in figure 5.6.

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



Figure 5.6: 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
🞇 Graphics Window: 3600 - Selected Iconic Object	💶 🗖 🗶 ConnectedRegions 📃 🗖 🗵
🌠 🛃 🖑 🔍 🔍 🔻 🗑 Fit 🔻 👰 100 % 👻	Region 1 (area: 100 center: (14.7,1.08 A
	Region 3 (area: 2554 center: (122.93);         Region 5 (area: 2 center: (107.5, 357.5         Region 6 (area: 2980 center: (193.913)         Region 7 (area: 2 center: (137, 406.5)))         Region 9 (area: 2 center: (138, 404))         Region 10 (area: 1 center: (138, 404))         Region 11 (area: 23 center: (148, 111))         Region 12 (area: 1 center: (184, 111))         Region 12 (area: 1 center: (208,672))         Region 13 (area: 2806 center: (270.67)         Region 14 (area: 1 center: (234,277))         Region 15 (area: 3 center: (324,111)         Region 16 (area: 1 center: (234,277))         Region 17 (area: 2 center: (304.5,152)         Region 18 (area: 4 center: (308.5,153)         Region 19 (area: 3 411 center: (396.5,152)         Region 19 (area: 3 411 center: (306.5,152)         Region 19 (area: 3 411 center: (306.5,153)         Region 19 (area: 3 411 center: (306.5,154)

Figure 5.7: Interactive inspection of an iconic variable containing regions.

whenever an image is added that uses the full domain. To clear the stack (and thus the graphics window) manually, click the clear icon (see figure 4.2 on page 23).

# 5.7 Inspecting Variables

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

Right-click on the icon ConnectedRegions and select Clear / Display to display only the connected regions in the graphics window. Right-click again and select Display Content > Select.... This menu entry opens a variable inspection window which lists the contents of the variable ConnectedRegions. The selected region of this inspection window is displayed in the graphics window using the current visualization settings. Set Draw to 'margin' and Shape to 'ellipse' and select some regions from the list. An example is illustrated in figure 5.7.

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 tools to find more appropriate settings interactively.

# 5.8 Improving the Threshold Using the Gray Histogram

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



Figure 5.8: Determining threshold bounds interactively using the gray histogram.

In the Output section of the gray histogram window, set Display to threshold. Now, you can try different threshold bounds by altering the values in Min and Max or by dragging the lines in the histogram area (see figure 5.8). 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.

# 5.9 Edit Lines

As previously noted, editing in HDevelop is line-based. All parameter modifications in the program are done through the operator window. To edit a line, double-click it in the program window. Afterwards, the parameters are displayed in the operator window for editing. If you click OK or Enter, the original line in the program is updated.

Double-click the second line of the program to adjust the threshold operation. Replace the value 30 with 56 and click Enter. The program line is updated in the program window.

# 5.10 Re-Execute the Program

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

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

# 5.11 Save the Program

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

# 5.12 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. 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 tools, which help 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 > Feature Histogram to open the tool. The checkbox at the top allows 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, i.e., the regions whose area falls between the values Min and Max, which are represented by the green and red vertical lines, respectively.

Specify the parameters in the Output section of the feature histogram window as shown in figure 5.9. Drag the green and red line to see how this affects the selected regions. From 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 6500. Add the following line to the program:

```
select_shape (ConnectedRegions, SelectedRegions, 'area', 'and', 4100, 6500)
```

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:



Figure 5.9: Selecting regions with a similar area in the feature histogram.

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

The operator orientation\_region returns a tuple of values: For each region in SelectedRegions a corresponding orientation value in Phi is returned. The operator area\_center in the same way returns the area, row and column of each input region as tuples. Again, run the program and inspect the calculated control variables. You can inspect multiple control variables in one inspection window. This is especially useful if the control variables all relate to each other as in this example. In the variable window select all control variables (hold down the <Ctrl>key), and right-click Inspect (see figure 5.10 on page 41).

# 5.13 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

				💥 Variat	-DX			
💥 Variable Vie	ew - main* (:::)				Phi	Area	Row	Column
Iconic Variable	es			0	0.691604	4876	126.439	182.198
000	200	200 20	2	1	2.4989	5091	167.206	438.396
-1-1	-	also als	Ĥ	2	-0.671646	5172	197.761	621.874
Control Variat	hies			3	0.36226	5212	271.818	238.464
Phi	[0.691604,	2.4989, -0.6716	546, O	4	1.98853	4804	370.505	533.539
Area	- [4876, 509	Inspect	804.	5	-1.86347	4767	399.981	339.445
Rom	F126 430	Clear Variable	271	6	1.47322	4916	450.672	667.683
KUW	[120,435,	Add to User Tab		7	-0.117317	5309	428.429	131.406
Loiumn	[182.198,		238	8	-0.148027	5964	503.389	461.329
	Sort by Name		L	9	-0.295232	5416	544.808	262.795
		<ul> <li>Sort by Occurrence</li> </ul>		10	1.71117	5011	664.751	427.176
	✓ Update Variables		L	11	2.55278	5265	660.569	590.18
		Cleanup		12	2.38878	4666	668.813	179.104
	n /\ Llser /							
				Types	real	integer	real	real

Figure 5.10: Inspecting control variables.

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 5.11. 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. 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.



Figure 5.11: Entering a loop in HDevelop.

Add the following lines to the program. They are automatically indented in the program window to highlight the nesting inside the **for** loop.

set\_tposition (3600, Row[Index], Column[Index])
write\_string (3600, deg(Phi[Index]) + ' degrees')

The instruction set\_tposition places the text cursor in the active graphics window at the center of the region corresponding to the loop index variable Index. The value 3600 is the so-called window handle of the target graphics window. This number is displayed in the title of the graphics window and can be different in your environment. The notation Row[Index] is another operation of the HDevelop language. It provides access to a single value of a tuple.

The instruction write\_string outputs a given string at the current text cursor position in the graphics window. 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 169.

# 5.14 Summary

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

# **Chapter 6**

# **Graphical User Interface**

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

# 6.1 Main Window

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

# Window Title

The main window of HDevelop is identified by the title HDevelop followed by the name of the current program (or unnamed if no file name has been specified yet). Unsaved changes in the current program are indicated with a trailing asterisk (\*) in the window title.

# Menu Bar

The menu bar at the top provides access to the functionality of HDevelop. The menus and their entries are described in the section "Menu Bar" on page 46.

# **Tool Bar**

The tool bar icons provide convenient shortcuts for frequently used functions. It is described in the section "Tool Bar" on page 101.

## Window Area

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

- Program window (see page 102)
- Operator window (see page 113)
- Variable window (see page 117)
- Graphics window (see page 124)
- Online help window (see page 127)

#### **Status Bar**

The status bar at the bottom of the main window displays status information, e.g., context-sensitive information about a specific user action or the runtime of operator or procedure calls (unless time measurement has been deactivated in the preferences, see section "Runtime Settings" on page 72). The status bar is divided into the following five areas (from left to right):

- 1. Status icon: Shows the current run status of the program.
- 2. Messages and runtime information. For example, if you select an operator from the menu, the corresponding short description is displayed here.
- 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 multichannel images, the gray values of all channels are displayed separated by commas.
- 5. Image coordinates in the graphics window (row, column).

The status bar has its own context menu:

You can toggle whether execution messages are displayed in the status bar by clicking the entry Show Processing Time in the context menu of the status bar. To open the context menu, right-click in the message area of the status bar.

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.

The status bar is displayed in figure 6.1.

# Window Modes

There are two different window modes in HDevelop, which can be toggled in the menu Window:

- **MDI** (multiple-document interface): In this mode the main window contains all other windows and dialogs (with the exception of the online help window and modal dialogs which block other windows temporarily). You are free to move the windows according to your needs and preferences inside the main window. You may iconify and/or deiconify them. HDevelop provides basic window management functions in this mode.
- **SDI** (single-document interface): In this mode the main window contains only the menu bar, the tool bar, the status bar, and the program window. All other windows are independent. You may find this mode beneficial if you want to take advantage of special window manager capabilities under UNIX.

🌾 read_image (38.6 ms) ĸ	N	[0] Clip (#=1: 830×822×1×	byte)	5 <b>‡</b> ≹0,0	
run status	messages / runtime information	image details	gray value	coordinates	



# 6.2 Menu Bar

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

# 6.2.1 Menu File

This menu provides functions to load images and existing programs and to save recently created or modified programs and procedures, respectively. Furthermore, you may export HDevelop programs to C++, C, Visual Basic, Visual Basic .NET, or C#, and also print them.

6.2.1.1 New Program

Synopsis: Initialize a new HDevelop program.

```
Checks for: Unsaved changes (page 141)
```

Shortcut: Ctrl+N

This menu item deletes the current program including all local procedures. The contents of variables are deleted before removing them. In addition, all graphics windows except one are closed. The last graphics window is cleared. The display parameters for the remaining graphics window are reset to the default values stored in the preferences (see section "Visualization Settings" on page 71). The runtime settings of the preferences are reset to their default values (see section "Runtime Settings" on page 72).

6.2.1.2 Open Program...

Synopsis: Load an existing HDevelop program.

```
Checks for: Unsaved changes (page 141)
```



F

Shortcut: Ctrl+O

A file selection dialog (page 140) pops up to select an HDevelop program. Please note that only native HDevelop programs can be loaded. Thus, text, C, C++, Visual Basic, Visual Basic .NET, and C# versions of a file are rejected.

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

💥 Open Example			
Category		Filter measure	
Pharmaceutical Photogrammetry and r		Example	Short Description
Precision engineering a		fuzzy_measure_pin.dev	Measure pins of an IC using fuzzy measuring
Rubber, synthetic mat		measure_ic_leads.dev	Measure leads of an IC
Semiconductors 	me pm	measure_pin.dev	Measure pins of an IC
Transport Wood and timber		pm_measure_board.dev	Locate IC on a board and measure pin distances
Industry.pack     Method     Operator     New in version	•		
🦳 Keep dialog open			Load In new HDevelop



# 6.2.1.3 Open Example...

Synopsis: Load HDevelop example program by selecting it from a categorized list.

```
Checks for: Unsaved changes (page 141)
```

Selecting this menu item opens a dialog that allows you to load HDevelop example programs grouped by categories. The dialog is displayed in figure 6.2.

#### Browsing the Categories

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

#### Filtering the Matched Example Programs

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

# Loading an Example Program

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

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

# 6.2.1.4 Recent Programs

Synopsis: Load recently used HDevelop programs.

Checks for: Unsaved changes (page 141)

This submenu contains a list of the most recently used HDevelop programs. Simply click on a program name to load it.

# 6.2.1.5 Insert Program

Synopsis: Insert (parts of) another HDevelop program into the current program.

Selecting an entry in this submenu opens a file selection dialog (page 140) for the selection of an HDevelop program.

□ Insert All...

Synopsis: Insert a complete HDevelop program into the current program.

The main body of the selected program is inserted at the IC. All local procedures of the selected program are copied to the current program under their original name. If a local procedure of that name already exists, the suffix \_COPY\_1 (or \_COPY\_2, \_COPY\_3... for subsequent imports) is added to the imported local procedure. All invocations of the renamed procedure are updated automatically.

If the current program already contains an external procedure with the same name, the imported local procedure overrides the external procedure.

 $\Box$  Insert Procedures...

Synopsis: Insert local procedures of the selected program into the current program.

Via this menu item you can add local procedures from an HDevelop program file to the current program. All local procedures except the main procedure are loaded from the selected file. If the current program already contains a local procedure with the same name, the newly added procedure will be renamed by appending the suffix \_COPY\_1 to its name. If the current program already contains an external procedure with the same name, the external procedure with the same name, the external procedure.

□ Insert Mainbody...

Synopsis: Insert only the main procedure of the selected program into the current program.

Insert the main body of the selected program at the IC. No local procedures are imported. If the current program does not provide the procedures used in the imported program, the corresponding program lines are marked as invalid code and will not be executed. Invalid code is turned into valid code by providing the missing procedures.

# 6.2.1.6 Save

自

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

# Shortcut: Ctrl+S

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

• Main or local procedure selected in program window:

Save changes of the current HDevelop program. If no file name has been specified yet, a file selection dialog (page 140) will be opened. Local procedures are saved within the HDevelop program.

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

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

• External procedure selected in program window:

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

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

[]]:



Figure 6.3: Adding a new directory to the list of external procedure directories.

6.2.1.7 Save Program As...

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

**Shortcut:** Ctrl+Shift+S

A file selection dialog (page 140) is opened. You can specify a new file name and save the current program under that name. The new file name becomes the default name of the current program so that subsequent Save operations will use that name instead of the old.

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

6.2.1.8	Save	Procedure	As.	

Synopsis: Save current procedure as an external procedure or as an HDevelop program.

Using this menu entry you can save the currently selected procedure as an external procedure or an HDevelop program. A file selection dialog (page 140) is opened where you can select the file type:

• HDevelop procedures (\*.dvp)

The procedure is saved as an external procedure. If the target directory is not already configured in the external procedure directories (see Menu Edit > Preferences..., External Procedures (page 66)), HDevelop will suggest adding the directory to the list. An example dialog is displayed in figure 6.3. If you click No, HDevelop will not be able to access the saved procedure unless the directory is later added to the external procedure settings manually.

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

• HDevelop local procedure (\*.dev)

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

This menu item is disabled if the current procedure is the main procedure.

📡 Export 🛛 🔀
Export File and Format
HALCON/examples/hdevelop/Applications/FA/cbm_dip_switch.cs
C# - HALCON/.NET
Fyport Dange
Program      Current Procedure      External Procedures
External Procedure Options
Export Procedure Body
Encoding
Native C UTF-8     O
Keep dialog open     Export     Cancel



## 6.2.1.9 Save All

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

```
Shortcut: Ctrl+Alt+S
```

山

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

6.2.1.10 Export...

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

See also: hdevelop -convert (command line switch)

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 6.4.

The button next to the export file name opens a file selection dialog (page 140) to select a file name and an export format. The following formats are supported (file extension in parentheses):

- Text file (.txt)
- $C(.c) \triangleright$  see also section 9.7 on page 215
- C++  $(. cpp) \triangleright$  see also section 9.1 on page 199
- C# HALCON/.COM (.cs) ▷ see also section 9.3 on page 205
- C# HALCON/.NET (.cs) ▷ see also section 9.2 on page 203

- Visual Basic 6.0 HALCON/COM (.bas) ▷ see also section 9.6 on page 213
- Visual Basic .NET HALCON/COM (.vb) ▷ see also section 9.5 on page 210
- Visual Basic .NET HALCON/.NET (.vb) ▷ see also section 9.4 on page 208

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 exported. The following options are available:

- Program: The entire program is exported (main procedure and *all* local procedures). All *used* external procedures are exported depending on the setting of the external procedure options (see below).
- Current Procedure: The current procedure and all used local procedures are exported. All *used* external procedures are exported depending on the setting of the external procedure options (see below).
- External Procedures: All external procedures are exported depending on the setting of the external procedure options (see below).

The short description and chapter information of procedures are exported as comments. Arbitrary code can be embedded with special comment lines (see section 9.8 on page 217).

External Procedure Options Defines the export behavior for external procedures.

• Export Procedure Body: Determines whether only the procedure declaration or both the declaration and the procedure body is exported.

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

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

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

#### 6.2.1.11 Read Image

Synopsis: Read an image from a selected directory.

#### See also: read\_image

This submenu contains several directories from which images can be loaded. The directory denoted by . (a single dot) is the current working directory of HDevelop, i.e., the directory HDevelop was started from. Below that entry, the directories specified by the environment variables HALCONROOT and HALCONIMAGES are displayed.

Directories below the separator line are user-defined directories. Each time an image is loaded from a directory which is not already listed, that directory name is appended to the menu. This is convenient when several images from a non-standard directory must be read.

When clicking on an entry in this menu, a file selection dialog (page 140) of the given directory is opened. Depending on the operating system you may be able to switch to a thumbnail view in this dialog. See figure 6.5 for an example.

After selecting a file name, the dialog Read Image is opened. It displays a thumbnail of the selected image and some image properties. This is also displayed in figure 6.5. 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, a 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.

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.

# 6.2.1.12 Cleanup

ø

Synopsis: Clean up allocated resources that are no longer being used.

See also: variable window ▷ context menu (page 119)

This menu item deletes all unused variables (iconic and control data) from the current procedure. These are variables in the variable window that are no longer used in any operator or procedure call in the current procedure body. This can happen after the deletion of program lines or after editing variable names, because the corresponding variables are not deleted automatically. You may use this menu item during a longer editing session to reorganize your variable window (page 117).

# 6.2.1.13 Properties...

Synopsis: Display various properties of the current program.

The tab card General displays file properties of the current program, such as file name, path, creation and modification date, and write permission. It also shows the file size, the number of lines of code, used and unused local procedures, used external procedures and used protected procedures. This is displayed in figure 6.6.

The tab card Used Modules lists the HALCON modules used by the current program. Modules marked with a lit bulb are used. This window allows you to get an estimate of how many modules your application will need in a runtime license. Please refer to the Installation Guide for more information about modules and runtime licenses. See figure 6.6 for the corresponding dialog of an OCR example.

- Check only used procedures If checked, only used procedures are considered for the evaluation of the used modules. Otherwise, all procedures are considered.
- Copy to Clipboard Copy the names of the used modules to the system clipboard. This way the list can be easily pasted into other applications.

6.2.1.14 Print...

Synopsis: Print the current program or selected procedures.

Shortcut: Ctrl+P

The print dialog is displayed in figure 6.7.

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.

👷 Properties 📃 🗆 🗙		-DX
General Used Modules	General Used Modules	
File Name:     engraved.dev       Path:     33/examples/hdevelop/Applications/OCR       Created:     Fri Mar 9 17:23:57 2007       Last Modified:     Fri Mar 9 17:23:57 2007	✓ Check only used procedures     Foundation	
Writable:         yes           Size:         4689 Bytes           Lines of Code:         57           Used Local Procedures:         0           Unused Local Procedures:         0	3D Metrology Q BarCode Q DataCode Q OCR/OCV Q	
Used External Procedures: 0 Used Protected Procedures: 0	Matching Q Copy to (	Tipboard

Figure 6.6: Properties: General (left), Used Modules (right).

#### **Procedure Options**

Print Procedures Define whether procedures are printed or not.

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

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

The bodies of external procedures that are locked by a password (see section "Edit Status of External Procedures" on page 111) are not printed.

6.2.1.15 Quit

Synopsis: Quit HDevelop.

Checks for: Unsaved changes (page 141)

Shortcut: Ctrl+Q

See also: exit

Ο

This menu item terminates HDevelop.

# 6.2.2 Menu Edit

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

Werint	
Name: \\print\large	Printer Selection And Properties
Print Range	
● Program ○ Current Procedure ○ Selection	C External Procedures
Procedure Options	
✓ Print Procedures	
Used Local Procedures C All Local Procedures	ures
Used External Procedures	
Conly Procedure Interface	
	Print Cancel

6.2.2.1 Undo

Synopsis: Undo your previous editing activities.



Shortcut: Ctrl+Z

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

Undo Insert Program line at 23 (read\_image)

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

The undo item does not work for the password assignment for external procedures (see section "Edit Status of External Procedures" on page 111). To undo the password assignment you either have to remove the password as long as you can edit the procedure, or you quit HDevelop without saving the corresponding procedure.

# 6.2.2.2 Redo

Synopsis: Revoke undo activities.



# Shortcut: Ctrl+Y

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

# 6.2.2.3 Cut

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



# Shortcut: Ctrl+X

The highlighted program lines are removed from the selected procedure and placed in an internal buffer for later use. Additionally, for every procedure call line the corresponding procedure and all procedures that can be reached from it are copied to the buffer. This is necessary in order to obtain a consistent program when pasting procedure call lines to a program in which the corresponding procedures might not exist. The highlighted program lines are also copied to the system clipboard.

## 6.2.2.4 Copy

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

## Shortcut: Ctrl+C

The highlighted program lines are copied without affecting the program. Additionally, for every procedure call line the corresponding procedure and all procedures that can be reached from it are copied to the buffer. This is necessary in order to obtain a consistent program when pasting procedure call lines to a program in which the corresponding procedures might not exist. The highlighted program lines are also copied to the system clipboard.

## 6.2.2.5 Paste

Synopsis: Insert the contents of the internal buffer to the currently selected procedure at the IC.

Shortcut: Ctrl+V

To insert the internal buffer in the current HDevelop procedure body, you place the IC at the desired position and then select this menu item. If the buffer contains procedures that do not exist they are added to the current program. If the paste buffer contains local procedures that do not exist, they are added to the current program. If the paste buffer contains calls to external procedures, the paths to those procedures are copied, too. However, before an external procedure path is added during a paste action, you are asked whether or not you want to add that particular path to the external procedure paths. The mechanism of copying and pasting procedure call lines together with the corresponding procedures is an easy way to transfer procedures between different HDevelop programs. It also works between multiple instances of HDevelop. The contents of the internal buffer are kept, allowing this command to be repeated.

# 6.2.2.6 Delete

Synopsis: Delete the highlighted program lines from the program window.

#### Shortcut: Del

This menu item deletes all highlighted program lines without storing them in an internal buffer. The only way to get the deleted lines back into your program body is to use the menu item Undo.

## 6.2.2.7 Activate

Synopsis: Uncomment the highlighted program lines.

#### Shortcut: F3

All of the highlighted program lines that were previously commented using the Deactivate command are converted back to executable code. Comment lines created with the operator comment are unaffected by this command.



**T**B

📡 Find/Replace		_D×
Find/Replace 💿 Variables 🕓 Opera	ators 🔘 Texts	
Search for	▼ Find	Find All
Replace	Replace	Replace All
Find Parameter		
Scope C All 📀 Program C Cu	urrent Procedure	
Options 🗖 Case Sensitive 🛛 🕅 W	/hole Words	



#### 6.2.2.8 Deactivate

Synopsis: Comment out the highlighted program lines.

Shortcut: F4

The highlighted program lines are converted into comments. This is a quick way to suppress the execution of portions of the program for testing purposes. Comment lines created with the operator comment are unaffected by this command.

During testing, it is often useful to prevent some lines of the program from being executed. This can be achieved by selecting the appropriate lines in the program window and then selecting Deactivate. An asterisk is placed at the beginning of the selected lines, i.e., the lines appear as comments in the program window and have no influence on the program during runtime.

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

Note that you can insert a *real* comment into your program by using the operator comment.

#### 6.2.2.9 Find/Replace...

Synopsis: Find and replace text in the current program.

# Shortcut: Ctrl+F

This dialog provides comprehensive facilities for searching the program code. You can perform a full text search or search for variable names as well as operator (or procedure) calls. In addition, you can replace variable names and substitute operator or procedure calls. The dialog is displayed in figure 6.8.

#### Search Context

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

Variables Find program lines with variable names that match the search text.

Operators Find program lines with operator or procedure calls that match the search text.

Texts Full text search. Find program lines that match the search text anywhere. No replacing is allowed in this mode to prevent breaking the consistency of the program code.

#### Search Scope

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 "Edit Status of External Procedures" on page 111).

#### Search Parameters

- 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 hold matches operator calls to threshold. Check this box to match only whole words.

#### 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. All matched lines are listed at the bottom of the dialog along with the corresponding procedure name and line number. Click on a search result to jump to the corresponding procedure and highlight the matching program line. This function is recommended before doing a global replace to preview which program lines will be affected. An example is displayed in figure 6.9.

You can even select multiple lines from the search result by holding the <Ctrl> key. The following actions may be performed for all selected lines (either from the context menu of the search result or the corresponding menu entries or tool bar icons): Cut (page 57), Copy (page 58), Delete (page 58), Activate (page 58), and Deactivate (page 59).

#### 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. It is recommended to do 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 6.10 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.

# 6.2.2.10 Find Again

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

# Shortcut: Ctrl+G

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

Sind/Replace	- D×				
Find/Replace $\mathbb C$ Variables	<ul> <li>Operators</li> <li>C Texts</li> </ul>				
Search for threshold Find All					
Replace	Replace Replace All				
Find Parameter					
Scope 💿 All 🔿 Progra	n 🔘 Current Procedure				
Options 🔲 Case Sensitive	☐ Whole Words				
Find Results					
Procedure Line	Context				
p_do_ocr_color 25	threshold (Green, ForegroundRaw, 0, 220)				
p_do_ocr_color 34	bin_threshold (ImageMean, Cluster1)				
p_do_circles 12	fast_threshold (Image, Region, 0, 120, 7)				
p_do_measure_rim 10 threshold (Rim, Dark, 0, 128)					
p_do_clips 15 bin_threshold (Clip, Dark)					
p_do_pcb 19 dyn_threshold (ImageOpening, ImageClosing,					
p_do_resistors 31	threshold (DepthMean, Resistor, 158, 255)				
n do resistors 39	threshold (DenthMean Solder 125 158)				

Figure 6.9: Finding all occurrences of the search text.

💓 Find/Replace						
Find/Replace C Variables 💿 Operators C Texts						
Search for threshold	Find Find All					
Replace fast_threshold	Replace Replace All					
Find Parameter						
Scope 🔿 All 📀 Program 🔿 Current Proce	edure					
Options 🔲 Case Sensitive 🛛 🔽 Whole Words	;					
Parameter Replacement Scheme						
threshold	fast_threshold					
<b></b>						
🖻 💽 Image 📃 🖬 In	mage					
Region 💌 📫 R	tegion					
🖻 🕑 MinGray 🔽 🔿 M	1inGray					
P MaxGray	1axGray					
	linSize					
	III DIEU					

Figure 6.10: Replacing operator calls.



# 6.2.2.11 Preferences...

Synopsis: Set global preferences of HDevelop.

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

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

UNIX: \$HOME/.hdevelop/MVTec/HDevelop.ini

The dialog provides its own menu with the following entries:

Import Using this menu entry you can import a selection of preferences which were previously saved using the menu entry Export (see below). The dialog is displayed in figure 6.11.

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

- Export The export dialog is identical to the import dialog. Using the check boxes you can specify which settings will be saved to the selected file.
- Reset Selecting this menu entry resets all preferences (except the window geometry and layout) to the default settings. If you want to reset the window geometry as well you can start HDevelop with the following command line switch:

hdevelop -reset\_preferences

The functionality of the available tab cards is described in the following sections.



Figure 6.11: Import.

### $\Box$ User Interface

This box provides tab cards for setting up the appearance and language of the user interface.

#### Program Listing

Here, you can specify the font and the syntax highlighting used in the program window. You can choose one of the pre-defined color schemes, or set up your own by clicking on the colored buttons. Changing any color automatically switches to the color scheme *User defined*.

The 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.



Figure 6.12: User Interface > Program Listing.

# Fonts

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

- General: The font used throughout the user interface (menu entries, labels etc.)
- Help Window: The body font used in the help window (menu Menu Help > Help).
- Program Listing: The font used in the program window. This is the same font as on the tab card Program Listing (see above).
- 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.

#### Language

In this tab card you can change the language of the user interface. Please note that HDevelop needs to be restarted if a different language is selected.

#### Layout

- Show full path in main window title This checkbox determines whether the full path of the current program or only the file name is displayed in the title bar of the HDevelop window.
- Default for Organize Windows These combo boxes define the tiled layout of the four main windows of HDevelop when using the menu entry Menu Window ▷ Organize Windows (page 97).

#### $\Box$ External Procedures

#### Directories

Use this tab card to manage the list of directories that contain external procedures. The directories are scanned for external procedures in their listing order. For each directory, the total number of procedures is displayed. The number of loaded procedures is usually equal to the total number. However, external procedures are not loaded if a directory contains procedures with the same name as a preceding directory. The tab card is displayed in figure 6.13.

💥 Preferences						
File						
📲 User Interface	Directories	External Procedu	ires   M	1anage Passwords	Procedure Use	Unre 🖣 🕨
External Procedures	List of search (	directories for exte	rnal proce	edures		Rescan
Seneral Options	Add	1				Delete
Section Sections				1		
S Rancino Soccings	Directory	Procedures	Loaded			
	H:/procedure	s/ 15	15			
1						

Figure 6.13: External Procedures > Directories.

- Add Select an additional directory from the file selection dialog (page 140). This directory will be appended to the list. All subdirectories of the selected directory will be scanned as well.
- Delete Delete the selected directory from the list. Programs using any external procedure from that directory will no longer run.

Rescan Rescan all listed directories to reflect any changes in the file system.

### External Procedures

This tab card lists all external procedures in the order they are loaded from the configured directories (page 66). For each procedure, the state (loaded/unloaded), the search directory, the path relative to the search directory, the number of uses in the current program and the number of unsaved modifications is listed. The tab card is displayed in figure 6.14.

🕻 User Interface	Directori	es External Procedu	res Ma	nage Passwords	Passwords Procedure Use Unre		
External Procedures	i state						
General Options	# 🛆	Procedure Name	State	Search Directory	Relative Path		
♥ Visualization Settings ♥ Runtime Settings	0001	compare_control	loaded	H:/procedures/	compare_control.dvp		
	0002	compare_iconic	loaded	H:/procedures/	compare_iconic.dvp		
	0003	compare_images	loaded	H:/procedures/	compare_images.dvp		
	0004 0	compare_objs	loaded	H:/procedures/	compare_objs.dvp		
	0005	compare_regions	loaded	H:/procedures/	compare_regions.dvp		
	0006	compare_tuples	loaded	H:/procedures/	compare_tuples.dvp compare_xlds.dvp		
	0007	compare_xids	loaded	H:/procedures/			
	0008	compose_test_index	loaded	H:/procedures/	compose_test_index.dv		
	0009	decompose_test_index	loaded	H:/procedures/	decompose_test_index.		
	0010	determine_xld_type	loaded	H:/procedures/	determine_xld_type.dvp		
	0011	gen_test_image	loaded	H:/procedures/	gen_test_image.dvp		
	0012	gen_test_region	loaded	H:/procedures/	gen_test_region.dvp		
	0013	gen_test_region_compl	loaded	H:/procedures/	gen_test_region_compl.		

Figure 6.14: External Procedures > External Procedures.

#### Manage Passwords

Using this tab card, you can conveniently manage the editing status and passwords of all external procedures. The external procedures are divided into three categories (from left to right): External procedures without a password (unprotected), external procedures for which the password has already been entered in this session (unlocked), and external procedures that are locked with a password. For an explanation of the different states, see section "Edit Status of External Procedures" on page 111.

Using the arrow buttons between the columns or the left and right cursor key, you can move the selected external 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 "Protecting a Procedure" on page 111. The same password is applied to all selected procedures.

If you move procedures from the second to the third column, the bodies of the corresponding external procedures will be locked. They can only be accessed if the correct password is supplied. This can either be done from this dialog by simply moving the corresponding procedures back to the middle column and entering the password. Or, you can unlock procedures individually from the program window as described in section "Creating and Editing Procedures" on page 105.

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 external procedures that share the same password.

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 49) and Save All (page 51).



Figure 6.15: External Procedures > Manage Passwords.

#### 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 6.17.

👷 Preferences									_ 🗆 ×			
File												
NT User Interface	Dired	tories	External Procedures	Manage Passwords	Procedu	ire Use	Unresolved Procedure	e Calls 📔				
Sector Procedures	Show the names of procedures that are used or not used by the loaded program or a selected procedure.											
🤏 General Options	main			•	<li>used</li>	C unus	ed	🔽 local	🔽 external			
Visualization Settings					Namo							
Runtime Settings		do che	sk for model files		Name							
	2	n chow	Lintro									
	-	p_snow										
	3	p_inicial	ustatus window									
	4	p_snow	_status_window									
	5	p_ao_i	icate_cylinder_3d									
	6	p_do_k	cate_cookie_box									
	7	p_do_3	d_matching_clamps									
	8	p_do_lo	cate_engine_parts									
	9	p_do_d	etect_road_signs									
	10	p_do_d	heck_blister_mixed									
	11	p_do_e	cc200_finder_pattern_t	olerance								
	12	p_do_ir	spect_bottle_mouth									
	13	p_do_lo	cate_car_door									
	14	p_do_r	econstruct_connection_r	rod_uncalib								
	15	p_do_t	extline_orientation									
	16	p_do_n	ovelty_detection_gmm									
	17	p_do_fi	nd_ncc_model_defocuse	ed					-			

Figure 6.16: External Procedures > Procedure Use.

#### 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. The tab card is displayed in figure 6.16.

#### □ General Options

#### General Options

Select the behavior of pressing the [Return] key in the operator window: Defines the default button in the operator window. The following options are available:

- OK (Enter and execute): Enter the operator in the program window and execute it (the default behavior).
- Enter: Enter the operator without executing it.

The selected option may be toggled temporarily in the operator window by pressing <Ctrl>+<Return>, i.e., if "OK (Enter and execute)" is set, pressing <Ctrl>+<Return> in the operator window enters the current operator without executing it.

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



Figure 6.17: External Procedures > Unresolved Procedure Calls.

changes in the current program, the program will be saved before being executed. Use this option with care: You usually do not want to select this option if you are experimenting with a program, e.g., when trying out different parameter settings.

Number of recent program files: The number of recent program files that are displayed in the menu Menu File  $\triangleright$  Recent Programs can be adjusted by altering the 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.

Encoding for saving HDevelop programs and procedures: HDevelop can save programs and procedures in two different encodings. Upon loading, HDevelop detects the used encoding automatically. It is recommended to use UTF-8 encoding (Unicode) if the program contains international characters that exceed the ASCII standard. However, in order to load programs in older versions of HDevelop, you have to set the encoding to Native.

Precision for displaying real values: This option sets the number of significant digits that are displayed in the variable window (page 117) and variable inspection windows see "Inspecting and Editing Variables" on page 122.

#### Experienced User

• Show HALCON Low Level Error Messages:

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

• Ignore semantic type:

The lists of the parameter combo boxes in the operator window are extended so that they include variables whose semantic types do not match the semantic types of the corresponding parameters of the selected operator.

• 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.

• 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.

#### $\Box$ Visualization Settings

These settings define the default visualization settings for graphics windows when HDevelop is started. See the description of "Set Parameters..." on page 82. The dialog is displayed in figure 6.18.



Figure 6.18: Visualization Settings > Visualization Parameters.

#### $\Box$ Runtime Settings

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  $\triangleright$  New Program. The dialog is displayed in figure 6.19.





#### Give Error See also: dev\_set\_check

This check box specified 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 will be ignored.

#### Show Processing Time See also: dev\_update\_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.

This option can also be toggled from the context menu of the status bar (see page 44).

Update Program Counter See also: dev\_update\_pc

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

#### Update Variables See also: dev\_update\_var

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

Update Graphics Window See also: dev\_update\_window

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

Enable the Context Menu in the Graphics Window See also: dev\_set\_preferences

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

# 6.2.3 Menu Execute

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

### 6.2.3.1 Run

Synopsis: Execute the current program from the PC.

### Shortcut: F5

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

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

• You may initiate limited activities. For example, if you double-click variables in the Variable Window they will be visualized; you may modify parameters for the graphics windows as described in Menu Visualization; you may even modify the current procedure body.

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

• A variable window update during runtime will only be performed if it has not been suppressed (see section "Runtime Settings" on page 72). In any case, the values of all variables are shown in the variable window after the execution's termination.

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

You have the following possibilities to stop your HDevelop program:

- The program runs until the last operator or procedure call in the current program (i.e., the main procedure body) has been called. The PC is positioned behind this operator. This is the usual way a program terminates.
- The menu Menu Execute > Stop (or the corresponding tool bar button) has been pressed.
- A break point has been set (see section "Program Window" on page 102). In this case, the last operator or procedure call that will be executed is the one *before* the break point.
- Menu item Menu File ▷ Quit has been executed (see "Quit" on page 56).
- A runtime error occurred. An input variable without a value or values outside a valid range might be typical reasons. In this case the PC remains in the line of the erroneous operator or procedure call.
- A stop instruction is executed. The PC remains on the line containing the stop instruction. Note that stop instructions inside protected external procedures (see "Edit Status of External Procedures" on page 111) are obeyed. However, the code of the protected procedure will only be visible if the correct password is entered in the program window.

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

6.2.3.2 Run to Insert Cursor

Synopsis: Run from PC to IC.

Shortcut: Shift+F5

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

6.2.3.3 Step Over

Synopsis: Execute the next operator in the current program.

Shortcut: F6

 $|\rangle$ 

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

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

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

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

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

### 6.2.3.4 Step Forward

Synopsis: Execute the next line in the current program.

Shortcut: Shift+F6

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

6.2.3.5 Step Into

Synopsis: Step into HDevelop procedure.



### Shortcut: F7

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

6.2.3.6 Step Out

Synopsis: Step out of HDevelop procedure.

### Shortcut: F8

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

### 6.2.3.7 Stop

Synopsis: Stop program execution.

Shortcut: F9

HDevelop continues processing until the current operator has completed its computations. This may take a long time if the operator is taking a lot of time to execute. There is no way of interrupting a HALCON operator. The procedure and procedure call in which the program execution was stopped becomes the current procedure and procedure call, respectively. After interrupting a program you may continue it by selecting Run or Step Over, or Step Into if the next program line is a procedure call.

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

6.2.3.8 Call Stack...

圓

STO P

Synopsis: Visualize the calling hierarchy.

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

When you click on a procedure call that belongs to a protected external procedure (for protected external procedures see "Edit Status of External Procedures" on page 111), you can only see the procedure body if you enter the correct password in the program window.

# 6.2.3.9 Set Breakpoint

**Synopsis:** Add break point(s) at selected line(s).

This menu item sets a break point on the lines that are currently selected in the program. In most cases, however, it is more convenient to set individual break points by holding the <Ctrl> key and clicking in the left column of the program window as described in "PC, IC, and Break Points" on page 104.

🔀 CallStack	
p_set_font	
p_show_status_window	
main	

Figure 6.20: Call Stack.

6.2.3.10 Clear Breakpoint

Ø

-

**Synopsis:** Clear break point(s) at selected line(s).

These menu item clears break points on the lines that are currently selected in the program. In most cases, however, it is more convenient to clear individual break points by holding the <Ctrl> key and clicking in the left column of the program window as described in section "PC, IC, and Break Points" on page 104.



6.2.3.11 Clear All Breakpoints

Synopsis: Clear all break points in the current program.

6.2.3.12 Reset Program Execution

Synopsis: Reset program to its initial state.



Shortcut: F2

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

6.2.3.13 Reset Procedure Execution

Synopsis: Reset procedure execution.



**Shortcut:** Shift+F2

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

#### 6.2.3.14 Abort Procedure Execution

Synopsis: Abort execution of current procedure.

Έ

Shortcut: Shift+F8

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

# 6.2.4 Menu Visualization

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

6.2.4.1 Open Graphics Window...

~

Synopsis: Open a new graphics window.

See also: dev\_open\_window

When selecting this menu entry, a dialog window pops up. Here, you may specify some graphics window attributes. The dialog is displayed in figure 6.21. The position, size and background color of the new graphics window can be specified. For example, it is more convenient to have a white background while building graphics for slides or reports (see the HALCON operator dump\_window). If the window height and width are set to -1, the window size is set by HDevelop. It is taken from the persistent preferences of HDevelop (usually the size of the last graphics window in the previous HDevelop session). A position value of -1 specifies that the window position is determined by the window manager (in SDI mode).

💥 Open Graphics Window 📃 🗖 🗙							
Row	0	<u>*</u>					
Column	0	<u> </u>					
Width	-1	*					
Height	-1	*					
Background	black	•					
	Open Ca	ancel					

Figure 6.21: Specifying the parameters of the new graphics window.

The window handle of the new graphics window is displayed in its title bar. This number may be used in operators that require a window handle (e.g., dev\_set\_window or dump\_window). The handling of graphics windows is described in more detail in section "Graphics Window" on page 124.

### 6.2.4.2 Clear Graphics Window

Synopsis: Clear active graphics window.

See also: dev\_clear\_window

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



Synopsis: Close active graphics window.

See also: dev\_close\_window

### 6.2.4.4 Display

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

#### See also: dev\_display

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

### 6.2.4.5 Window Size

Synopsis: Set window size of active graphics window.

#### See also: dev\_set\_window\_extents

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

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

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

### 6.2.4.6 Image Size

(**B**)

Synopsis: Zoom image size of active graphics window.

The entry Fit scales the image to completely fill the graphics window.

A list of fixed percentages scales the image with respect to its natural size.

Double and Half double and half the current image size, respectively.

Aspect scales down the image size, so that its aspect ratio is maintained.

### 6.2.4.7 Colored

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

See also: dev\_set\_colored

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

6.2.4.8 Color

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

See also: dev\_set\_color

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

The default color is red.

### 6.2.4.9 Draw

Synopsis: Draw type of regions.

See also: dev\_set\_draw

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

### 6.2.4.10 Line Width

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

### See also: dev\_set\_line\_width

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

### 6.2.4.11 Shape

Synopsis: Specify representation shape for regions.

### See also: dev\_set\_shape

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

# 6.2.4.12 Lut

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

See also: dev\_set\_lut

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

# 6.2.4.13 Paint

Synopsis: Specify image visualization.

See also: dev\_set\_paint

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

### 6.2.4.14 Apply Immediately

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

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

### 6.2.4.15 Update Window

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

If this menu entry bears a checkmark, every object (image, region, or XLD) is displayed in the active graphics window during program execution. Otherwise, the active graphics window is not updated.



Figure 6.22: Visualization Parameters with multiple graphics windows.

### 6.2.4.16 Reset Parameters

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

Here, the display parameters of the active graphics window are set to their initial state (as defined in the preferences, see page 62). 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.

### 6.2.4.17 Set Parameters...

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

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

Select Graphics Window (only with multiple graphics windows) Keep in mind that 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.

- Update This check box corresponds to the setting of Menu Visualization > Apply Immediately. If it is checked, every change of a parameter will immediately lead to a redisplay of the image, regions, or XLD in the graphics window. Otherwise, the parameters become active for the next display of an object (double-click on an icon or execution of an operator).
- Reset Reset to the visualization settings defined in the Preferences (page 62).
- Use settings for new windows Make the current settings also the default settings for new graphics windows.

#### $\Box$ Pen settings

Here, the display modes for regions and XLDs are specified. You can select the color (single or multiple), the drawing mode (filled or outlined), the line width, and the shape of the regions.

You can select up to 12 colors by clicking the appropriate check box. They are used to emphasize the connectivity of different regions in the graphics window. If you choose a single color presentation you may specify this color by selecting it in the list box.

With the parameter Shape (default is original), you may specify 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 line width of the presented regions, XLDs, or lines is specified with help of the menu item Line Width.

For regions the draw mode can be specified: Either it might be *filled* (item fill) or *outlined* (item margin).

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

- "Draw", see also page 80
- "Colored", see also page 80
- "Color", see also page 80
- "Shape", see also page 81
- "Line Width", see also page 80

### □ LUT settings

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

See the description of the menu entry "Lut" on page 81.



Figure 6.23: Visualization Parameters: LUT settings.

### □ Paint settings

Here, you can select between several graphical presentations for images. Examples are contourline and 3D-plot. In the default mode the image will be displayed as is, i.e., as a picture.

If you have chosen a presentation mode, the window displays all possible parameters you may modify. For example, after selecting the item 3D-plot you can specify the following parameters:

- Row and
- Column (position of the center).
- Step (the distance of plot lines in pixels),
- Scale (height of 3D plot),
- Eye height,
- Eye distance (view point),
- Colored (use the gray value of a pixel to draw a line segment instead of one graphic color).

### $\Box$ 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.

🕵 Visualization Parameters			<u> </u>
Pen LUT Paint Z	oom		
default row		Row	0 🔺
contourline		Column	0
3D-plot 3D-plot_hidden		Step	7 主
3D-plot_point		Scale	150 👤
		Eye Height	110 🔹
		Eye Distance	160 👤
			Colored
Vpdate Reset		Use settings	for new windows

Figure 6.24: Visualization Parameters: Paint settings.

🔀 Visualization Parameters	
Pen LUT Paint Zoom	
Row Co	lumn
Set part	
Upper Left Corner 0 🔶 0	
Lower Right Corner 511 🚊 511	÷
Set center	
Center Point 255 📑 255	=
Reset Zoom Out Zoom In A	spect
Vpdate Reset Use settings for ne	ew windows

Figure 6.25: Visualization Parameters: Zoom settings.

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

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

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

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

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

6.2.4.18 Zoom Window

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

The zoom window is described in section "Zoom Window" on page 131.

6.2.4.19 New Zoom Window

Synopsis: Open additional zoom window.

See section "Zoom Window" on page 131.

6.2.4.20 Gray Histogram

Synopsis: Display gray value histogram of active graphics window.

Selecting this entry opens a sophisticated tool for the inspection of gray value histograms, which can also be used to select thresholds interactively and to set the range of displayed gray values dynamically. It is described in section "Gray Histogram Window" on page 133.

6.2.4.21 Feature Histogram

**Synopsis:** Interactive inspection of feature histograms.

This menu item opens a sophisticated tool for the inspection of feature histograms. It is described in section "Feature Histogram Window" on page 137.



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

This window provides a tool for the convenient inspection of shape and gray value features of individual regions. It is described in section "Feature Inspection Window" on page 138.

Ь.

100

86

6.2.4.23 Save Window ...

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

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

# 6.2.5 Menu Procedures

The menu Procedures contains all functionality that is needed to create, modify, copy, or delete HDevelop procedures. To save procedures, refer to the Menu File menu (page 46).

6.2.5.1 Create New Procedure

Synopsis: Create a new internal or external procedure.

Selecting this item opens the Procedure Interface Dialog window. The procedure interface dialog and the mechanism of creating procedures are described in section "Creating and Editing Procedures" on page 105.

### 6.2.5.2 Duplicate...

Synopsis: Copy a procedure under a different name.

Selecting this menu item opens a dialog with which it is possible to copy existing procedures. The dialog is displayed in figure 6.26. The combo box Source contains all local procedures in the current program and all external procedures. In the Target text field the name of the copied procedure can be entered. Clicking the OK button creates a copy of the source procedure, Cancel dismisses the dialog. The copy retains the status (local or external) of the source procedure. The copy of an external procedure is placed in the same directory as the source procedure.

<b>W</b> Duplicat	e Procedure	<u>?</u> ×
Source:		•
Target:		
	ОК	Cancel



Duplicating external procedures that are protected with a password (see "Edit Status of External Procedures" on page 111) is also possible. The associated password is also used for the duplicated procedure.

### 6.2.5.3 Edit Interface

Synopsis: Edit procedure interface.

This menu item opens the procedure interface window and displays the interface of the current procedure (or the first procedure from the list, if "main" is the current procedure). The menu item has the same effect as the button Edit Interface (page 107) in the Program Window (see page 102).

The interface of external procedures that are protected with a password can only be edited after the corresponding password has been entered (see section "Edit Status of External Procedures" on page 111.

6.2.5.4 Delete Current

Synopsis: Delete the current procedure.

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

### 6.2.5.5 Delete All Unused Local

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

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

### 6.2.5.6 Insert Used As Local

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

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

6.2.5.7 Insert All As Local

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

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

With this menu item, you can change all of your procedures to become local. If your program contains protected external procedures, HDevelop issues a warning and inserts only the procedures that are not locked. Individual procedures can be made local (or external) via the check box Local in the procedures interface (see section "Procedure Interface Dialog" on page 107). For changing the edit status of an external procedure see section "Edit Status of External Procedures" on page 111.

## 6.2.5.8 Make All External

Synopsis: Convert all local procedures into external procedures.

The formerly local procedures are now stored as external procedures in a selectable directory of the list of external procedure directories (see section "External Procedures" on page 66). If no directories are configured, you can select a target directory from a dialog. HDevelop will add the selected target directory to the list if you tell it to. Otherwise, the operation will be cancelled. For changing the edit status of an external procedure see section "Edit Status of External Procedures" on page 111.

6.2.5.9 External Procedure Settings...

# 4.E

Synopsis: Configure settings for external procedures.

When you select this menu item, the window Preferences...  $\triangleright$  External Procedures appears. With it, you can define one or more directories where external procedures are stored and searched for. Please note that by default no directory is defined. Inside the defined directories, also the contained subdirectories are scanned. Therefore, it is recommended to select directories with a restricted depth in order to speed up the search process. If you change the list of directories, the list of all available external procedures is updated.

See also section "External Procedures" on page 66.

# 6.2.5.10 Edit Procedure

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

This submenu lists all procedures in submenus grouped by chapter and section title (see section "Procedure Interface Dialog" on page 107). Procedures without a chapter title are listed directly in the menu Edit Procedure. If you click on a procedure name, it will become the current procedure in the program window. You can also select procedures in the combo box of the program window (page 102).

# 6.2.6 Menu Operators

Synopsis: Select HALCON operators and procedures.

This menu item comprises all HALCON and HDevelop operators including the HDevelop control constructs. Furthermore, procedures can be selected from a submenu at the bottom of this menu.

### 6.2.6.1 Control

Synopsis: Select control flow operators.

Here, you may select control structures for the program. This involves the execution of a program segment (henceforth named body) depending on a test (if, ifelse, and elseif) and the repetition of a program segment (for, while, and repeat) with controlled loop execution (break, continue). Furthermore, you may stop the program's execution at any position (stop) or terminate HDevelop (exit). The operators assign and insert do not influence the execution, but serve to specify values for control data (assignment). The operator comment is used to add a comment, that means any sequence of characters, to the program. The operator return terminates the current procedure call and returns to the calling procedure (see section "Creating and Editing Procedures" on page 105 for more information about HDevelop procedures).

Selecting a menu item displays the corresponding control construct in the operator window, where you can set the necessary parameters. After specifying all parameters you may transfer the construct into your program. A direct execution for loops and conditions is not possible, in contrast to other HDevelop and HALCON operators, because you have to specify the loop's and condition's body first to obtain useful semantics. If necessary, you may execute the program after the input with Step Over or Run. The IC is positioned after the construct head to ensure the input of the construct's body occurs in the correct place. This body is indented to make the nesting level of the control constructs visible, and thus to help you in understanding the program structure. The semantics for loops and conditions are shown in section "Control Flow Operators" on page 192.

#### Assignment

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

x = y + z;

is declared inside the operator window as

assign(y + z, x)

and displayed in the program window by

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

a[i] = v;

insert(a,v,i,a)

in the operator window, and is displayed as

in the HDevelop program window.

### Program termination

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

### **Comments**

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

### 6.2.6.2 Develop

Synopsis: Select operators specific to HDevelop.

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

All operators in this menu start with the prefix dev\_. It has been introduced to have a distinction to the underlying basic HALCON operators (e.g., dev\_set\_color and set\_color).

The effects of each operator are described as follows:

- dev\_open\_window, dev\_close\_window, dev\_clear\_window The operators dev\_open\_window and dev\_close\_window are used to open and to close a graphics window, respectively. During opening, the parameterization allows you to specify the window's size and position. The oper- ator dev\_clear\_window clears the active window's content and its history. This corresponds to the usage of the button Clear in the graphics window. Please note that dev\_open\_window and dev\_close\_window are not exported to Visual Basic, Visual Basic .NET, and C# because here one HWindowXCtrl is used.
- dev\_set\_window\_extents With this operator, you can set the size and position of the active HDevelop graphics window.

- dev\_set\_window This operator activates the graphics window containing the given ID. This ID is an output parameter of dev\_open\_window. After the execution, the output is redirected to this window. This operator is not needed for exported code in C++ or C, because here every window operation uses the ID as a parameter. The operator has no effect for exported code in Visual Basic, Visual Basic .NET, and C#.
- dev\_set\_color, dev\_set\_colored dev\_set\_color has the same effects as the menu item Menu Visualization ▷ Color (page 80). dev\_set\_colored is equal to the menu item Menu Visualization ▷ Colored (page 80).
- dev\_set\_draw This operator has the same effects as the menu item Menu Visualization ▷ Draw (page 80).
- dev\_set\_line\_width For an explanation see the menu item Menu Visualization ▷ Line Width (page 80).
- dev\_set\_lut For an explanation see the menu item Menu Visualization  $\triangleright$  Lut (page 81).
- dev\_set\_paint For an explanation see the menu item Menu Visualization > Paint (page 81). If you want to specify all possible parameters of a given paint mode, you have to specify them as a tuple, analogously to the HALCON operator set\_paint.
- dev\_set\_shape For an explanation see the menu item Menu Visualization ▷ Shape (page 81).
- dev\_set\_part This operator adjusts the coordinate system for image, region, XLD and other graphic output. This is done by specifying the upper left and the lower right corner coordinates. This specified part is shown in the entire graphics window. If the width or height of the specified rectangle has a negative value (e.g., Row1 > Row2), the result is equivalent to the menu Menu Visualization > Reset Parameters: the zoom mode is switched off, i.e., the *most recently* displayed image fills the whole graphics window. This feature of dev\_set\_part is *not* supported for exported C, C++, Visual Basic, Visual Basic .NET, and C# code.
- dev\_display Iconic variables are displayed in the active graphics window by this operator. It is reasonable to do this when the automatic output is suppressed (see dev\_update\_window below and Menu Edit > Preferences... > Runtime Settings (page 72).
- dev\_clear\_obj This operator deletes the iconic object stored in the HDevelop variable that is passed as the input parameter. In the variable window, the object is displayed as undefined (with a ? as its icon).
- dev\_inspect\_ctrl This operator opens an inspection window displaying the values of the variable passed to the operator. To inspect multiple variables at once, you can pass a tuple of variable names. In most cases a list dialog is opened, which shows *all* values of the variable (see also section "Inspecting and Editing Variables" on page 122). In the case of an image acquisition device handle, a description of this image acquisition device is opened. In addition, this dialog allows online grabbing of images (see page 122). This operator is not supported for exported code.
- dev\_close\_inspect\_ctrl This is the opposite operator to dev\_inspect\_ctrl, and closes the inspect window. This operator is not supported for exported code.

- dev\_map\_par, dev\_unmap\_par These operators open and close the parameter dialog, which can also be opened using the menu Menu Visualization ▷ Set Parameters.... This operator is not supported for exported code.
- dev\_map\_var, dev\_unmap\_var These operators iconify the variable window (dev\_unmap\_var), and retransform the iconified window to the normal visualization size, respectively (dev\_map\_var). This means that the variable window always remains visible on the display in one of the two ways of visualization. These operators can be executed with the help of the window manager. These operators are not supported for exported code.
- dev\_map\_prog, dev\_unmap\_prog Analogously to dev\_map\_var and dev\_unmap\_var, these operators iconify or deiconify the program window. These operators are not supported for exported code.
- dev\_update\_window, dev\_update\_var, dev\_update\_time, dev\_update\_pc Using these operators, you may configure the output at runtime. It corresponds to the settings in menu Menu Edit > Preferences... > Runtime Settings (page 72). These operators are not supported for exported code.
- dev\_set\_check This operator is equivalent to set\_check of the HALCON library. It is used to handle runtime errors caused by HALCON operators that are executed inside HDevelop. The parameter value 'give\_error', which is the default, leads to a stop of the program together with an error dialog if a value not equal to H\_MSG\_TRUE is returned. Using the value '~give\_error', errors or other messages are ignored and the program can continue. This mode is useful in connection with operators like get\_mposition, file\_exists, read\_image, or test\_region\_point, which can return H\_MSG\_FAIL.
- dev\_error\_var This operator specifies a variable that contains the return value (error code) of an operator after execution. This value can be used to continue, depending on the given value. dev\_error\_var is normally used in connection with dev\_set\_check. Note that, as the procedure concept in HDevelop only allows for local variables, the variable set by dev\_error\_var will only be valid in calls to the relevant procedure. Furthermore, every corresponding procedure call will have an own instance of the variable, i.e. the variable might have different values in different procedure calls. For an example how to use dev\_error\_var in connection with dev\_set\_check see %HALCONROOT%\examples\hdevelop\Graphics\Mouse\get\_mposition.dev.

Please note that operations concerning graphics windows and their corresponding operators have additional functionality in comparison to HALCON operators with corresponding names (without dev\_): graphics windows in HDevelop are based on HALCON windows (see open\_window in the HALCON reference manual), but in fact, they have an enhanced functionality (e.g., history of displayed objects, interactive modification of size, and control buttons). This is also true for operators that modify visualization parameters (dev\_set\_color, dev\_set\_draw, etc.). For example, the new visualization parameter is registered in the parameter window when the operator has been executed. You can easily check this by opening the dialog Menu Visualization  $\triangleright$  Set Parameters...  $\triangleright$  Pen and apply the operator dev\_set\_color. Here you will see the change of the visualization parameters in the dialog box. You have to be aware of this difference if you export dev\_\* to C, C++, Visual Basic, Visual Basic .NET, and C# code.

In contrast to the parameter dialog for changing display parameters like color, the corresponding operators (like dev\_set\_color) do not change the contents of the graphics window (i.e., they don't cause a redisplay). They are used to prepare the parameters for the *next* display action.

### 6.2.6.3 Classification, File, Filter, ...

Synopsis: Select HALCON operators.

In the these menu entries, you can find all HALCON operators, arranged in chapters and subchapters. This set of image analysis operators forms the most important part of HALCON: the HALCON library. HALCON operators implement the different image analysis tasks such as preprocessing, filtering, or measurement.

You may look for a detailed description of each operator in the HALCON reference manual. Operators in the menus Control and Develop are special operators of HDevelop. Thus, you will not find them in the reference manuals for HALCON/C, HALCON/C++, or HALCON/COM.

The menu has a cascade structure, according to the chapter structure of the HALCON reference manual. As this menu is built up dynamically when HDevelop starts, it might take some time until it is available. During the build-up time the menu is "grayed out". Selecting a chapter of the menu opens a pulldown menu with the corresponding subchapters or operators, respectively.

This operator hierarchy is especially useful for novices because it offers all operators sorted by thematic aspects. This might be interesting for an experienced user, too, if he wants to compare, e.g., different smoothing filters, because they reside in the same subchapter. To get additional information, a short description of an operator (while activating its name in the menu) is displayed in the status bar.

Note, that some operators are visible in the menus but should not be used, e.g., open\_window (in Menu Operators > Graphics > Window) or reset\_obj\_db (in Menu Operators > System > Database). If you select one of these operators, a warning text is displayed in the operator window. This warning will usually refer to a legal substitute. In the case of most of these operators, you should use the corresponding Develop operator (e.g., dev\_open\_window instead of open\_window) within HDevelop.

# 6.2.7 Menu Suggestions

Synopsis: Let HDevelop suggest operators based on the current operator.

This menu shows you another possibility how to select HALCON operators. But here they are proposed to you in a different manner. It is assumed that you have already selected an operator in a previous step. Depending on this operator, five different suggestions are offered.

Suggestions are separated into groups as described below.

### 6.2.7.1 Predecessors

Many operators require a reasonable or necessary predecessor operator. For example, before computing junction points in a skeleton (junctions\_skeleton), you have to compute this skeleton itself

(skeleton). To obtain a threshold image you usually use a lowpass filter before executing a dynamic threshold (dyn\_threshold). Using the watershed algorithms (watersheds), it is reasonable to apply a smoothing filter on an image first, because this reduces runtime considerably.

### 6.2.7.2 Successors

In many cases the task results in a "natural" sequence of operators. Thus, as a rule you use a thresholding after executing an edge filter or you execute a region processing (e.g., morphological operators) after a segmentation. To facilitate a reasonable processing, all the possible operators are offered in this menu item.

### 6.2.7.3 Alternatives

Since HALCON includes a large library, this menu item suggests alternative operators. Thus, you may, for example, replace mean\_image with operators such as gauss\_image, sigma\_image, or smooth\_image.

### 6.2.7.4 See also

Contrary to Alternatives, operators are offered here which have some *connection* to the current operator. Thus, the median filter (median\_image) is not a direct alternative to the mean filter (mean\_image). Similarly, the regiongrowing operator (regiongrowing) is no alternative for a thresholding. In any case, they offer a different approach to solve a task. References might consist of pure informative nature, too: the operator gen\_lowpass, which is used to create a lowpass filter in the frequency domain, is a reasonable reference to a Gaussian filter.

#### 6.2.7.5 Keywords...

This menu item gives access to HALCON operators through keywords which are associated with each operator. The tab card Keywords of the online help window is opened. It is described in section "Keywords" on page 129.

# 6.2.8 Menu Assistants

This menu assembles assistants for specific machine vision tasks. Each assistant provides a user interface tailored to the requirements of its task. Using this interface, the user can interactively set up and configure the assistant to solve a specific machine vision problem. Once the configuration is working satisfactorily, the assistant can be instructed to generate HDevelop code into the current program. You can also save an assistant's configuration for later use.

Currently, the following assistants are available:

• Image Acquisition: Using this assistant you can generate code to acquire images from different sources (files, directories, image acquisition interfaces). It is described in section 4.3 on page 23.

• Matching: Using this assistant you can generate code to perform shape-based matching in your HDevelop program. You can load a reference image to train a model. Using a selection of test images containing the model you can tweak a set of parameters to find the model in all variations permitted by the application. Furthermore, the parameters can be optimized to increase the processing speed. The assistant is described in the online help of HDevelop.

### **Common Features of all HDevelop Assistants**

Some features are common to all HDevelop assistants. First of all, you can open multiple assistants. Assistants of the same type are numbered consecutively, e.g., if you open two image acquisition assistants, they are labeled Image Acquisition 01 and Image Acquisition 02, respectively. When you open a new assistant, a menu entry is added to the top of the menu Assistants, from which the corresponding assistant can be restored if it has been closed. The current setup is lost and the menu entry disappears if the associated assistant is exited explicitly (see below). If you want to keep the setup for later sessions, you can always save it to a file.

Different assistants have different menus (usually corresponding to the available tab cards). These menus provide functionality specific to the assistant's task. There are also some menu entries that are available in every assistant. They are described in the following.

- File > Load Assistant Settings Using this entry, a previous configuration can be loaded from a file which has been generated using the menu entry Save Current Assistant Settings.
- File > Save Current Assistant Settings You can save the configuration of an assistant to a file for later use. The default extension for these configuration files is .das.
- File > Close Dialog The assistant is closed, but the current configuration is preserved. This menu entry performs the same function as the assistant's close button. You can restore a closed assistant by clicking the numbered entry in the menu Assistants which is generated when a new assistant is opened.
- File > Exit Assistant The assistant is quit. The resources used by the assistant are released. The link to the generated code is lost, i.e., it is not possible to restore the assistant unless the setup has been saved to a file. The menu entry in the menu Assistants is also removed.
- Code Generation > Insert Code 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.

# 6.2.9 Menu Window

This menu offers support to manage the sub-windows of the main window, i.e., the program, operator, variable, graphics window(s), and possibly other dialogs. At the bottom of the menu all open windows are listed. Clicking an entry here brings the corresponding window to the front.



6.2.9.1 Open Graphics Window

Synopsis: Open an additional graphics window.

See also: dev\_open\_window, and section "Open Graphics Window..." on page 78.



6.2.9.2 Open Program Listing

Synopsis: Open the program window.

See also: dev\_map\_prog

This menu item is grayed out if the program window is already open.



6.2.9.3 Open Variable Window

Synopsis: Open the variable window.

See also: dev\_map\_var

This menu item is grayed out if the variable window is already open.



6.2.9.4 Open Operator Window

Synopsis: Open the operator window.

This menu item is grayed out if the operator window is already open.

6.2.9.5 Organize Windows

Synopsis: Clean up window area of main window.

When selecting this item, the main window is split into four areas: by default, all graphics windows are cascaded to fit the upper left quarter, the operator window fits the upper right quarter, the variable window fits the lower left quarter, and the program window fits the lower right quarter. The positioning can be adjusted in the preferences (see page 65). All other windows that are currently open are cascaded at the center of the main window. In this arrangement, the four most important windows are placed in a non-overlapping fashion to provide maximum accessibility. It is therefore the default layout of HDevelop. The tiled layout is displayed in figure 6.27.



Figure 6.27: Tiled window layout of HDevelop.

# 6.2.9.6 Cascade Windows

Synopsis: Arrange windows in a cascade.

By selecting this item, HDevelop arranges the currently open windows in a cascade. The cascaded window layout is displayed in figure 6.28.

### 6.2.9.7 SDI/MDI

무

Synopsis: Switch between multiple-document interface (the default) and single-document interface.

The different modes are explained in section "Main Window" on page 43.

₩Н	Devel	op - unr	named										_ 0	×
Eile	Edit	Execute	<u>V</u> isua	lization	Proc	edures	Ope	rators	Sugg	estions	Assist	ants	Window	»
			5 B		<b>1</b>	$\langle \varphi \rangle$	B	₿					> 🖌	»
<b>%</b>	Graphic	s Window	v: 3600										-DX	
X	Progr	am Listing	)										-DX	
2	👯 Vari	able View	- main i	(:::)										
	<b>%</b> 0	perator												×
	Enter	r Operato	r or Pro	cedure								_		]   [
	Para	meters –												-
Γ-		Ok		En	ter	1	AD	alv		Cano	el 🚺		Help	1
	<u> </u>				9991		- PI	9.97°		sauro		<u> </u>	, i i u u u u	-
1950		-								-		Ť.	10,68	

Figure 6.28: Cascaded window layout of HDevelop.

# 6.2.10 Menu Help

Here, you may query information about HALCON itself and all HALCON and HDevelop operators.

# 6.2.10.1 Help

Synopsis: Open the online help window.

# Shortcut: F1

7

The help window provides access to the documentation of HDevelop and HALCON. In particular, the complete HALCON Reference Manual is available with extensive documentation of each operator. Another possibility of requesting information about the current operator is pressing the button Help inside the operator window (see section "Operator Window" on page 113).

The help window is described in section "Help Window" on page 127.

6.2.10.2 HALCON Reference

Synopsis: Display the HALCON Reference Manual in the online help window.

6.2.10.3 HDevelop Reference

Synopsis: Display the HDevelop Reference chapter in the online help window.

### 6.2.10.4 HDevelop Language

Synopsis: Display the HDevelop Language chapter in the online help window.

### 6.2.10.5 Search Documentation

Synopsis: Open the online help window and show the search tab to enter search queries.

The online help provides an integrated search engine. You can enter search queries there and search the HALCON documentation suite.

The search syntax is described in section "Help Window" on page 127.

6.2.10.6 HALCON News (WWW)

Synopsis: Visit the HALCON home page.

This menu item lets you check for the latest news about HALCON on MVTec's WWW server, e.g., whether new extension packages, image acquisition interfaces, or HALCON versions are available.

### 6.2.10.7 About

Synopsis: Display HDevelop version and licensing host IDs.

This menu item delivers information about the current HALCON and HDevelop version. Furthermore, it lists host IDs detected by the license manager (see the Installation Guide for more information).

# 6.3 Tool Bar

You use most icons in this tool bar to accelerate accessing important HDevelop features. These are features which you are performing many times while working with HDevelop. Hence, there are buttons to handle your HDevelop programs and to edit them. The most important buttons are used to start and to stop a program (or parts of a program).





# 6.4 Program Window

The program window (see figure 6.30 on page 104) is divided into three areas:

- At the top, you find elements for selecting procedures or editing the interface of existing procedures.
- Below this, the column at the left side contains the PC, the IC, and optionally, one or more break points.
- The main part of the program window contains the program code of the current HDevelop procedure.

These three parts are described in the following sections, but in the reverse sequence.

# 6.4.1 The Program Area

The main part of the program window contains the program code of the current HDevelop procedure. Here, the user can obtain information about the inserted operators or procedure calls. A program is built up such that every line contains exactly *one* operator or procedure call with its parameters, or an assignment. An exception are the conditional constructs if and ifelse, and the loop constructs while, for, and until. They contain two, and in case of ifelse three program lines, which enclose the body. Every line starts with an operator or procedure name, which is indented, if necessary, to highlight the structure created by the above mentioned control structures. After the operator or procedure name the parameters are displayed in parentheses. Parameters are separated by commas.

The program window is used to *visualize* program lines, but not to modify them. You cannot change a program body by modifying the text directly. Editing the program text in HDevelop is done in the operator window (see section "Operator Window" on page 113). The main reason for this principle is the advantage that it facilitates providing sophisticated help. Thus, many input errors can be avoided.

👷 Program Listing	Voperator Window - Edit: 'dev_open_window' in procedure 'main' at line 7
→ main (:::) Edit Interface	Enter Operator or Procedure dev_open_window
1 * clip.dev: Orientation of clips	Parameters
2 *	AT I
<pre> dev_update_window ('off') 4 read image (Clip, 'clip') </pre>	Row C rectangle.origin.y (integer)
5 get_image_pointer1 (Cliv, _, _, Width, Height) 6 dev close window ()	Column 0 rectangle.origin.x (integer)
7 dev_open_window (0, 0, Width/2, Height/2, 'black', Wind	Width Width/2 rectangle.extent.x (integer)
8 dev_display (Clip) 9 stop ()	Height Height/2 rectangle.extent.y (integer)
10 bin_threshold (Clip, Dark) 11 connection (Dark, Single)	Background 'black' color (string)
12 select_shape (Single, Selected, 'area', 'and', 5000, 10 13 dev_set_draw ('fill')	WindowHandle WindowID vindow(integer)
14 dev_set_colored (12) 15 dev_display (Selected)	
	Ok Replace Apply Cancel Help

Figure 6.29: Editing a program line.

# **Editing Program Lines**

To edit a line of a program, you double-click it with the left mouse button. In case of conditions and loops the operator line with the parameters has to be selected. For example, you have to double-click for in a for...endfor loop, but until in a repeat...until loop. You may edit only *one* operator or procedure call at a time.

After double-clicking a program line, note the following:

- The program line is highlighted in a slanted font. This serves as a reminder that you are altering an existing program line instead of adding a new one.
- The operator or procedure call of the program line is displayed in the operator window and can be edited there.
- 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.
- Unless the program line is deleted while still being edited, clicking OK or Replace in the operator window will replace the original program line. This is even the case if the corresponding program line is no longer in view, e.g., if a different procedure is selected in the program window.

If the program line is deleted before the changes are committed in the operator window, the edited line will be inserted as a new program line at the IC. If you are in doubt about the current status, check the window title of the operator window.

Figure 6.29 illustrates the editing process.

# Copy, Paste, Delete

Besides editing the parameters of a single operator or procedure call, single and multiple lines can be deleted, cut, or pasted in one step using simple mouse functions. To use this feature, you select one or more lines using the mouse:

• You select one line by clicking on it. Previously activated lines will then become deactivated.



Figure 6.30: Program example with the PC, IC, and a break point (BP).

- To activate more than one line you have to press the <Ctrl> key while clicking on the line.
- The <Shift> key is used to activate a sequence of lines using one mouse click: All lines between the most recent activation and the new one will become activated.

After the selection of lines, the edit function can be activated by either using the menu Menu Edit, or the tool bar (see section "Tool Bar" on page 101), or via the context menu of the program window (see page 105).

# 6.4.2 PC, IC, and Break Points

The column to the left of the displayed program body contains the PC, represented as a green arrow pointing to a program line, the IC (a black triangle between two program lines) and optionally one or more *break points* (BP–a red STOP sign).

The program counter resides in the line of the next operator or procedure call to execute. The IC indicates the position to insert a new program line. A break point shows the program line on which the program is stopped.

You may position or activate these three labels by clicking in the left column of the program window. That column itself is divided into three areas: Depending on the horizontal position of the mouse cursor, all three label types are available. The actual type is indicated through a change of the mouse cursor. At the leftmost position, break points can be placed. In the middle position, the PC can be placed. And finally, in the rightmost position, the IC can be placed. HDevelop assists you by displaying the icon that would be inserted. 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 <Shift>+<Ctrl> to place the PC.

# **Context Menu**

By clicking into the program window with the right mouse button you can open a context menu, which contains shortcuts to some of the actions of the menus Menu Edit, e.g., copy and paste lines, and Menu Execute, e.g., activate and deactivate lines or set and clear break points. 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 onto which you clicked with the right mouse button.

Note that any actions that modify the position of the PC will cause the call stack to pop all procedure calls until the current procedure call remains on top. This is relevant in case the current procedure call is not the top-most procedure call and is necessary to secure the consistency of the call stack. Modification of the PC can happen as well directly as described above or indirectly by, e.g., inserting a program line above the PC in the current procedure body.

The following entries of the context menu are not available elsewhere:

- Run Until Here Execute the lines from the PC to the line under the mouse cursor.
- Help If the line under the mouse cursor contains an operator call, the corresponding page is 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 If the line under the mouse cursor contains a procedure call, the corresponding procedure becomes the current procedure, i.e., it is displayed for editing.
- Show Caller This menu item lists all the places in the current program where the currently displayed procedure is called. Clicking on an entry takes you to the corresponding program line.

# 6.4.3 Creating and Editing Procedures

HDevelop always displays one procedure, the current procedure, at a time. The combo box on top of the program window displays the name of the current procedure. You can select another procedure from this box. The first element of the list is the main procedure, followed by the local procedures of the current program, followed by the external procedures. The procedures are sorted alphabetically. After being selected from the list, a procedure becomes the current procedure and the corresponding procedure call becomes the current procedure has multiple calls on the stack, the last of the procedure calls is displayed.



Figure 6.31: Context menu of the program window.



Figure 6.32: Display of a locked procedure in the program window.

If the procedure is locked, a password button is displayed instead of the procedure body. See figure 6.32 for an illustration. In order to access the code, the correct password has to be entered (see also section "Edit Status of External Procedures" on page 111).

While the body of the current procedure is visualized in the program window, the procedure *interface* can optionally be viewed or modified in the procedure interface dialog displayed in a separate window. The button Edit Interface at the top of the program window performs the same action as the menu item Menu Procedures  $\triangleright$  Edit Interface. Activating this button invokes the procedure interface dialog.

	💥 Create Procedu	re		
	Properties			
	Procedure Name		🗌 Local	🔁 Password
	Short Description			
	Chapter			
	Directory			- 🖻
	Parameters			
iconic parameters —	- 🖾			Reset
input —	- 🖻 🖻 🔪			Remove
output —	- 🗟 🗾 🔍			11011070
control parameters —	- 🖉 📃 🕺	> add new parameter		Move Up
input —	- 🖻 🖻 🧹			Move Down
output —	- 🗟 🖻 🤇			
			ОК	Cancel

only for external procedures

Figure 6.33: Creating a new procedure.

### 6.4.3.1 Procedure Interface Dialog

The procedure interface dialog enables you to define the interface of new procedures, and view and edit the interface of existing procedures. It is displayed in figure 6.33.

### □ Properties

- Procedure Name This text field specifies the name of the procedure. If the dialog is opened in order to create a new procedure, it contains a text field for the procedure name to be entered. If you edit an existing procedure, the name of the procedure is displayed in the combo box Procedure Name on top of the dialog. You can edit the interface of another procedure by selecting it from the combo box.
- Local The check box to the right of the combo box determines whether the procedure is a local or external procedure. Local procedures are always saved within the HDevelop program while external procedures are saved as stand-alone files.
- Password For external procedures additionally the edit status can be specified that determines if the procedure can be viewed and modified by all users or if it is protected by a password (see section "Edit Status of External Procedures" on page 111). Local procedures cannot be protected.

- Short Description The text field Short Description below the combo box can be used to enter an optional short description for the procedure. The short description of a procedure is treated like the short description of a HALCON operator, i.e., it is displayed in HDevelop's status bar when selecting a procedure from the menu.
- Chapter The text field Chapter can be used to specify chapter and subchapter, separated by a slash ('/'), so that your procedures can be displayed thematically ordered in the list at the bottom of the menus Menu Procedures and Menu Operators.
- Directory Next, the path is displayed where the procedure is stored if it is an external procedure. When storing the external procedure for the first time, this path corresponds always to the first path in the list of external procedure paths in the dialog Menu Procedures > External Procedure Settings... (see page 89). You can select any of the configured external procedure paths from the combo box. If you are editing an existing external procedure, the corresponding path is displayed but cannot be altered. Thus, once created external procedures can only be relocated in the file system.

You can also specify a new target directory by clicking the browse button next to the combo box. When you commit the new procedure later, HDevelop will suggest to add this directory to the list of external procedures automatically. If the addition is canceled, the new procedure will not be available unless you add the corresponding directory manually.

If the specified directory is a subdirectory of one of the pre-configured paths, it will not be added to the list. This is because subdirectories are automatically searched in HDevelop.

Note that the structure created by chapters and subchapters in the text field before 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.

### $\Box$ Parameters

The next part of the dialog is used for the procedure interface parameters. As mentioned earlier, HDevelop procedure interfaces have the same structure as HALCON operator interfaces, that is, they may contain parameters of the four categories iconic input, iconic output, control input, and control output in this order. The procedure interface dialog contains four separate areas that offer the necessary functionality for manipulating parameters. These areas correspond to above parameter classes and are independent of each other. Every area is marked with an icon that describes the parameter class. It contains a button for inserting new parameters, which are always appended at the end of the parameter list. The latter is displayed by an array of text fields containing the parameter names.

- Reset If you are creating a new procedure, clicking this button removes all entered parameters. If you are editing an existing procedure, the first click of this button restores the original interface, i.e., any changes to the parameters are undone. The second click removes all parameters.
- 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.
- OK Activating the button OK on 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 in order to create a new procedure or to modify the interface of 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 are editing an existing procedure interface, the checkbox Adapt program becomes available. If it is checked, all calls to the procedure in the current program are checked for consistency and updated if necessary. Note that if new parameters are added to an existing procedure interface, the corresponding procedure calls are modified by adding new variables as input parameters, which most likely will not be initialized at the time of the procedure call.

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 of the edited procedure are lost (with the exception of the editing status, see section "Edit Status of External Procedures" on page 111).

#### 6.4.3.2 Creating Procedures

Depending on the corresponding procedure options and a possible selection of lines in the program window, there are different ways on how a procedure can be created. When clicking Menu Procedures  $\triangleright$  Create New Procedure, the program lines marked in the program window are copied and inserted in the program body of the new procedure. This is illustrated in figure 6.34. If the last selected program line is not a return operator, a return call is added at the end of the procedure body. If no lines are selected in the program window, the newly created procedure body contains only the return operator.

By default, a local procedure is created. If you want to create an external procedure instead, you have to disable the check box Local and optionally specify chapter and subchapter. Additionally, you can select the path the procedure is stored in, which by default is the first path specified in the preferences (see section "External Procedures" on page 66). Further, section "Edit Status of External Procedures" on page 111 shows how to protect an external procedure by a password so that only authorized persons can view and modify it.

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 following different options can be selected in the combo box Selection Scheme to determine the procedure parameters:

Only In Only Out Variables that are exclusively input and output variables become input and output procedure parameters, respectively.

💥 Program Listing		Create Procedure	
→ main* (:::)	Edit Interface	Properties	
1 ImageFile := 'clip'		Procedure Name p_init_image	I∕ Local
2 read_image (Image, Image	File) Deinter Tune Nidth Heich	Short Description Initialize graphics window	
4 dev close window ()	e, Poincer, Type, wrach, heigh	Chapter Image/Init	
5 dev_open_window (0, 0, W	idth, He	Parameters Selection Scheme Only	In, All Out 💌
6 dev_display (Image) 7 threshold (Image, Region	. 0. 50) <sup>?</sup> Help	<b>F</b> U	Reset
8 connection (Region, Conn	ectedRec Show Procedure		Remove
9 select_shape (ConnectedR	egions, 🔉 Cut	📮 Image	Remove
	🗈 Сору		Move Up
	Paste		Move Down
	Delete	ImageFile 편	
	🗐 Activate	Pointer	
	E Deactivate	Туре	
	Create New Procedure	Width	
	California Caralan	WindowHandle	
	Set Program Counter  Set Insert Cursor	Ð	
	✓ Update Program Counter		
		✓ Replace selected program lines	Cancel
· ·			
Rogram Listin	g	×□_	
➡ main* (:::)		Edit Interface	
1 Image	File := 'clip' t image (Image ImageFile Poin	ter Tune Width Height WindowHendle)	
3 thres	hold (Image, Region, 0, 50)	cer, Type, widen, height, windownandie)	
4 conne	ction (Region, ConnectedRegions	)	
5 selec	t_shape (connectedRegions, sele	ctedRegions, 'area', 'and', 4200, 6000)	
		[+]	
<u> </u>			
Reprogram Listin	g		
E p_init_image	e (:Image:ImageFile:Pointer,Type,Width,Height,Win	dowHandle) Edit Interface	
1 read_	image (Image, ImageFile)		
2 get_1 3 dev c	mage_pointer1 (Image, Pointer, lose window ()	Type, Width, Height)	
4 dev_o	pen_window (O, O, Width, Height	, 'black', WindowHandle)	
5 dev_d	isplay (Image) n ()		
orecur			

Figure 6.34: Creating a procedure from selected lines.

- Only In All Out Variables that are exclusively input variables become input parameters, while all output variables become output parameters. This is the default option in HDevelop.
- All In Only Out This setting is symmetrical to the previous setting.

All In All Out All input and output variables become input and output procedure parameters, respectively.

The classification of variables in the selected program lines is performed separately for iconic and control variables. If a variable is an input as well as an output variable, it is assigned to the first category, i.e., the corresponding procedure parameter becomes an input parameter.

If the option Replace selected program lines is checked, the selected program lines are replaced by an appropriate call of the new procedure. Otherwise, the 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 as stated above.

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.

#### 6.4.3.3 Edit Status of External Procedures

External procedures can be protected with a password. The bodies of protected external procedures can only be accessed if the correct password is supplied. They can be applied by all users but viewed and modified only by authorized people who know the password. The edit status of a procedure can be changed via the procedure interface dialog (page 107). To manage the edit status of multiple external procedures at once, click Menu Procedures > External Procedure Settings... and select the tab card Manage Passwords (page 68).

By default, new procedures are local procedures, which cannot be protected. To make them external, the check box Local has to be disabled. Now, the button Password becomes available.

#### Protecting a Procedure

If you want to protect a procedure with a password, do the following:

- Select the corresponding procedure in the program window.
- Click Edit Interface to edit the interface of the selected procedure.
- Make sure Local is unchecked. Only external procedures can be protected.
- 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 6.35 for an example. If both times the same password is used, clicking OK assigns the password. Otherwise, an error message is raised and you have to repeat the password assignment. When a protected procedure is finally saved, it is stored in a binary format.

When you start HDevelop the next time, the protected procedure is locked, i.e., when trying to edit the procedure, e.g., by selecting it from the combo box in the program window, a corresponding message is displayed in the program window. See figure 6.36 for an example. Additionally, a password button is displayed in the program window. Upon entering the correct password, the procedure is temporarily unlocked and stays unlocked as long as you do not close HDevelop.

N I dassing I d	٢
Please enter your password to lock the procedure.	
Password	
Confirm	
OK Cancel	

Figure 6.35: Entering a password to protect an external procedure.



Figure 6.36: A locked external procedure.

#### Changing the Edit Status of a Protected Procedure

To change the status of a protected procedure, you must first unlock it temporarily by entering the password. Then, you can use the procedure interface dialog (page 107) to change the password or remove the password to turn the protected external procedure into an unprotected external or even a local procedure. Click the button Password to change the edit status.

The following options are available:

- Lock You can lock the protected procedure so its body cannot be accessed in the current session without supplying the password again.
- Remove Selecting this option removes the password. When the procedure is saved, it is no longer protected. For turning a protected external procedure into a local procedure (without a password) it is sufficient to activate the check box Local in the procedure interface dialog (page 107).



Figure 6.37: Changing the edit status of a protected procedure.

New password The password window appears and you assign the new password by the same process you used for the old one.

Cancel The operation is cancelled without altering the status.

#### Warning

When working with protected procedures, be aware that the password cannot be reconstructed, so be very careful not to forget it and not to repeat a typing error when assigning it! Further, in some situations protected external procedures behave different from common external or local procedures. In particular, as they can not be viewed and modified by unauthorized users, they also can not 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 external 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. At last, if a protected external procedure contains any stop instructions, they are ignored.

# 6.5 Operator Window

This window is used to edit and display an operator or procedure call with all its parameters. Here you will obtain information about the number of the parameters of the operator or procedure, the parameter types, and parameter values. You can modify the parameter values according to your image processing tasks. For this you may use the values proposed by HDevelop or specify your own values.

The operator window consists of the following three 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 row of buttons at the bottom allows to control the parameter display.

💥 Operator - select_characters 📃 🗖 🗙				
Enter Operator or Procedure select_characters				
Parameters	select_characters select_contours_xld	-		
<u>N</u>	select_gray			
🖹 Region	select_grayvalues_rrom_channels select_lines select_lines_longest			
RegionCharacters R	eslect_matching_lines select_obj			
	select_region_point	-		
🖆 DotPrint 🔤	false'	string		
StrokeWidth	medium'	string		
CharWidth 2	5 💌	integer		
CharHeight 2	5 💌	integer		
<b>.</b>	· · ·	<b>I</b>		
Ok Enl	ter Apply Cancel	Help		

Figure 6.38: Selecting an operator after typing select\_.

## 6.5.1 Operator Name Field

The operator name field allows to select operators or procedures by specifying a substring 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 user-specified substring. 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 arranged in alphabetical order.

If there is an unambiguous search result, the operator or procedure is 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.

If you are already more familiar with HDevelop, it is useful to select an operator or procedure in the operator name field. However, in order to do so, you obviously have to be familiar with the operator names.

#### 6.5.2 Parameter Display

The parameter display is the main part of the operator window. It is empty in its initial state. If you have selected an operator or procedure call, HDevelop displays its parameter data, i.e., name, number, type, and default values, in the display.

💥 Operator Window - I	nsert: 'select_shape'	
Enter Operator or Proce	edure select_shape	-
Parameters		
<u> – – – – – – – – – – – – – – – – – – –</u>		
🖹 Regions	ConnectedRegions ]	region
🖶 SelectedRegions	SelectedRegions Regions to be examined.	region
🖆 Features	'area'	string
Operation	'and'	string
Min	150	real
Мах	99999	real
Ok	Enter Apply Cancel H	Help

Figure 6.39: Specifying parameters for the operator select\_shape.

- In the first column of the parameter display the parameter types are indicated by icons. Note that icons are not repeated if a parameter is of the same type as its predecessor. Hover the mouse cursor over the icons to get a tool tip.
- In the second column of the operator window you find the parameter names.
- The third column consists of the text fields, which contain variable names in case of iconic and control output parameters and expressions in case of control input parameters. If you want to change the suggestions offered by the system (variable names or default values), you may do so either manually or by pressing 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.
- The fourth column indicates the parameter's default semantic type and its data type in parentheses.

Hover the mouse cursor over the second to fourth column to get a short description for the corresponding parameter as a tool tip.

Please refer to the following rules on how parameters obtain their values and how you may specify them:

**Iconic input parameters** Possible inputs for these parameters are iconic variables of the corresponding list. If there is no need to execute the operator or procedure call immediately, you may even specify new variable names, i.e., names, that do not already exist in the variable window, but will be instantiated later by adding further operators or procedure calls to the program body. In any case, you have to specify iconic parameters exclusively with variable names. It is not possible to use expressions.

- **Iconic output parameters** These parameters contain default variables, which have the same names as the parameters themselves. If a variable with the same name as the output parameter is already being used, a number is added to the name to make it unique. Because the parameter names characterize the computed result very well, you may adopt these default names in many cases. Besides this, you are free to choose arbitrary names either by yourself or by opening the list (see above). If you use a variable that already has a value, this value is deleted during execution before overwriting it with new results. It is possible to specify a variable both in an input and output position.
- **Control input parameters** These parameters normally possess a default value. As an alternative, you may use the text field's button to open a combo box and to select a value suggestion. 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), and character strings (string).

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 brackets. Furthermore, you may build up expressions with these values. The possibilities of using tuples are very extensive. You may use expressions in HDevelop similar to the use of expressions in C or in Pascal. You will find a more detailed description in section "Expressions for Input Control Parameters" on page 173.

**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, either.

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 HALCON default values.

- Text input: To specify a parameter using your keyboard is the simplest but not the most often used method. Here you have to click into a text field with the left mouse button. This activates the field and prepares it for user input. Simultaneously, the writing position is marked by a vertical bar. Now you may input numbers, strings, expressions, or variables. There are some editing functions to help you doing input: <Backspace> deletes the character to the left and <Delete> deletes the one to the right. You may also select (invert) a sequence of characters in the text field using the mouse or holding <Shift> and using the cursor keys. If there is a succeeding input, the marked region is going to be deleted first and afterwards the characters are going to be written in the text field. See page 221 for a summary of the keyboard mappings.
- **Combo box selection:** Using this input method, you can obtain rapid settings of variables and constants. To do so, you have to click the button on the text field's right side. A combo box is opened, in which you may select an item. Thus, you are able to choose a certain variable or value without risking erroneous typing. This item is transferred to the operator name field. Previous entries are deleted. Afterwards, the combo box is closed. If there are no variables or appropriate values, the combo box remains closed.

#### 6.5.3 Control Buttons

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

Ok By clicking Ok you execute the operator or procedure call with the specified parameters. When doing so, the execution mode depends on the position of the PC: If the PC is placed above the insertion position, the system executes the program from the PC until the insertion position first. *Then*, the operator or procedure call that has been edited in the operator window is executed. The reason for this is that the parameter values that are used as input values for the currently edited operator or procedure call have to be calculated. If the PC is placed at or after the insertion position, only the currently edited operator or procedure call is executed.

The operator or procedure call is entered into the program window before it is executed. After the execution, the PC is positioned on the next executable program line after the edited operator or procedure call.

The computed output parameter values are displayed in the variable window. Iconic variables are shown in the current graphics window if you haven't suppressed this option (compare section "Runtime Settings" on page 72). Afterwards, the operator window is cleared. If you did not specify all parameters or if you used wrong values, an error dialog is raised and execution is canceled. In this case, the operator window remains open to allow appropriate changes.

- Enter / Replace By clicking the button Enter, the currently edited operator or procedure call is transferred into the program window without being executed. When editing existing program lines (through double-clicking in the program window, see page 103), the button label changes to Replace. When clicked, the original program line is replaced.
- Apply If you click Apply the operator is executed with the specified parameters, but not entered into or changed in the program. This enables you to determine the optimum parameters rapidly since the operator dialog remains open, and hence you can change parameters quickly. Note that this functionality is not available for procedure calls, and thus the button is grayed out in this case.

Unlike the button Ok, only the single line you edit or enter is executed, no matter where the PC is located. Thus, you have to ensure that all the input variables contain meaningful values. By pressing Apply, the corresponding output variables are changed or created, if necessary, to allow you to inspect their values. If you decide not to enter the line into the program body, some unused variables may thus be created. You can easily remove them by selecting Menu File  $\triangleright$  Cleanup.

- Cancel Clicking Cancel clears the contents of the operator window. Thus, there are neither changes in the program nor in any variables.
- Help Clicking Help invokes an appropriate help text for the selected operator. For this the system activates the online help window (see Help Window). Note that this functionality is not available for procedure calls, and thus the button is grayed out in this case.

# 6.6 Variable Window

There are two kinds of variables in HALCON, corresponding to the two parameter types of HALCON: iconic objects (images, regions, and XLDs) and control data (numbers, strings). The corresponding variables are called iconic and control variables. These variables may possess a value or may be undefined.



Figure 6.40: Variable window with instantiated iconic and control variables.

An undefined variable is created, for example, when loading a program or after inserting an operator with a new variable that is not executed immediately into a program. You may access these undefined variables only by writing to them. If you try to read such a variable, a runtime error occurs. If a variable obtains a value, the variable type is specified more precisely. A control variable that contains, for example, an integer is of type integer. This type might change to real or a tuple of integer after specifying new values for this variable. But it always remains a control variable. Similarly, this is the case for iconic variables, which may contain regions, images, or XLDs. You may assign new values to an iconic variable as often as you want to, but you cannot change its type so that it becomes a control variable.

In addition to classifying HDevelop variables by whether they are iconic or control variables, they can also be distinguished by whether they are interface parameters of the current procedure or local variables. Generally, both kinds of variables are treated equally.

New variables are created in the operator dialog area during specification of operator or procedure call parameters. Here, every sequence of characters without single quotation marks is interpreted as a variable name. If this name did not exist before, the variable is created in the operator dialog area by pressing Ok or Enter. The variable type is specified through the type of the parameter where it was used for the first time: Variables that correspond to an iconic object parameter create an iconic variable; variables for a control parameter create a control variable. Every time an operator or procedure call is executed, the results are stored in variables connected to its output parameters. This is achieved by first deleting the contents of the variable and then assigning the new value to it.

The variable window is similar to a watch window used in window-oriented debuggers. Inside this window you are able to keep track of variable values. Corresponding to the two variable types, there are two areas in the variable window. One for iconic data (above or left) and the other for control data (below or right).

All computed variables are displayed showing their iconic or control values (if the automatic update has

not been switched off, see section "Runtime Settings" on page 72). In case of a tuple result which is too long, the tuple presentation is shortened, indicated by three dots. In this case the full content of a variable can be displayed in an inspection window by double-clicking the value list. See also the following sections.

#### Switching Between Horizontal and Vertical Layout

You can toggle the orientation of the two parts of the variable window. To do this, double-click the dividing line between both parts. You can also drag that line to resize the parts.

#### **Managing Variables**

In large programs the variable window can become quite cluttered which makes watching selected variables difficult. Therefore, you can customize the selection of displayed variables. At the bottom of the variable window, three tabs are available:

- All: All variables are displayed at once.
- Auto: The variables of the current and the previous operator call are displayed. This is useful when single-stepping through the program, because only the variables relevant to the current context are displayed.
- User: A user-defined selection of variables is displayed. Variables may be added and removed using the context menu of the variable window (see below). If the tab User is active, variables may be added from a list in the context menu. In the other two tabs variables are added by selecting them first and clicking Add to User Tab in the context menu.

#### Context menu

In both parts of the variable window distinct context menus are available by right-clicking in the window. The entries that are common in both parts are described here.

- Clear Variable: The selected variables are cleared and appear as undefined.
- Add to User Tab: The selected variables are added to the tab User.
- Sort by Name: The variables are sorted in alphabetical order.
- Sort by Occurrence: The variables appear in the same order as they are defined in the program.
- Update Variables: Toggle whether variables will be updated during program execution. This is the same setting as in the runtime preferences (see page 72).
- Cleanup: Delete all unused variables (see page 54).

Only applicable when the tab User is selected:

- Add Variable: This submenu contains a list of all variables that are currently *not* displayed in the tab User. Clicking a variable name adds the variable to the tab.
- Remove from User Tab: The selected variables are removed from the tab User.



Figure 6.41: Displaying information about an iconic variable with a reduced domain.

# 6.6.1 Iconic Variables

The iconic variables are represented by icons, which contain an image, a region or an XLD, depending on the current value. The icons are created depending on the type of data according to the following rules:

• For images the icon contains a zoomed version of them, filling the icon completely. Due to the zooming onto the square shape of the icon, the aspect ratio of the small image might be wrong. If there is more than one image in the variable, only the *first* image is used for the icon. Similarly, for multi-channel images only the *first* channel is displayed. An exception is made for images with 3 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 6.41 for an illustration.

- Regions are displayed by first calculating the smallest surrounding rectangle and then zooming it so that it fills the icon using a border of one pixel. In contrast to images, the aspect ratio is always correct. This can lead to black bars at the borders. The color used to draw the region is always white without further modifications (except zooming).
- XLD data is displayed using the coordinate system of the largest image used so far. The color used for XLD objects is white on black background.

Because of the different ways of displaying objects, you have to be aware that the coordinates cannot be compared. The variable name is positioned below each icon. They are displayed in the variable window in the order of occurrence or name from left to right. If there is not enough space, a scrollbar is created, which you can use to scroll the icons.

#### **Displaying iconic variables**

Double-clicking an icon with the left mouse button displays the data in the active graphics window. If you use images of different sizes in a program, the system uses the following output strategy for an automatic adaption of the zooming: 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. If a partial zooming has been activated before (see section "Graphics Window" on page 124), it is going to be suppressed.

#### Displaying information about iconic variables

You can get information about an instantiated variable by placing the mouse pointer over the corresponding icon in the variable window. See also figure 6.41 on page 120 for an illustration. The information depends on the contents of the corresponding variable:

- Images: The image type and size and the number of channels is displayed. If the iconic variable contains multiple images, the properties of the first image are reported.
- Regions: The area and the center of the region is displayed. If the iconic variable contains multiple regions, the properties of the first region are reported.
- XLDs: The number of contour points and the length is displayed. If the iconic variable contains multiple XLDs, the properties of the first XLD are reported.

#### **Context Menu**

Clicking on an icon with the right mouse button opens a context menu with several options. 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. 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 6.42 for an example.

Normally, regions, images, and XLDs are represented in variable icons. Besides this there are three exceptions, which are shown by special icons:

- Undefined variables are displayed as a question mark (?) icon. You may *write to* but not read them, because they do not have any value.
- Brackets ([]) are used if a variable is instantiated but does not contain an iconic object (empty tuple). This may be the case using operators like select\_shape with "wrong" specified thresholds or using the operator gen\_empty\_obj. Such a value might be reasonable if you want to collect iconic objects in a variable gradually in a loop using concat\_obj. Here, an empty tuple is used as starting value for the loop.
- A last exception is an *empty region*. This is *one* region that does not contain any pixels (points), i.e., the area (number of points) is 0. You must not confuse this case with the empty tuple, because there the area is not defined. The empty region is symbolized by an empty set icon ( $\emptyset$ ).



Figure 6.42: Interactive channel selection from an RGB image.

# 6.6.2 Control Variables

To the right of the variable name you find its values in the default representation (you have to keep in mind that a floating point number without significant fractional part is represented as an integer, e.g., 1.0 is represented as 1). If you specify more than one value for one variable (tuple), they are separated by commas and enclosed by brackets. If the number of values exceeds an upper limit, the output is clipped. This is indicated by three dots at the end of the tuple. For undefined variables, their name and a ? are shown in the variable field. An empty tuple is represented by []. Both exceptions use the same symbols as the corresponding cases for the iconic variables.

#### **Inspecting and Editing Variables**

#### See also: dev\_inspect\_ctrl

Double-clicking a control variable opens a window that displays all its values in a tabular format. This is helpful if you have tuple variables with a large number of values that you want to inspect. Below the list, some statistical data may be displayed (minimum value, maximum value, sum of values, mean value, deviation, types, number of values, and the semantics if appropriate. You can select which statistical data is displayed by right-clicking on the statistics table and selecting the corresponding entries.

You can also select multiple control variables at once in the variable window by holding down the <Ctrl> key. To inspect these variables in a single inspection window, right click on the selected variables and select Inspect.

💥 Variable Ins	pect: ColumnEdgeFirst		×
	ColumnEdgeF	irst .	-
0		94.1577	
1		105.357	
2		116.023	
3		126.739	
4		137.489	
5		148.515	
6		159.023	
7		169.873	
8		181.002	-
Min		94.1577	
Max		876.281	
Sum		35979.5	
Mean		486.21	
Deviation		230.384	
Types		double	
Number		74	
Semantics		point.x	

Figure 6.43: Variable inspection.

An example inspection window is displayed in figure 6.43.

#### **Copying Values to the Clipboard**

Within the variable window, the context menu offers an entry for copying the values of the selected variable to the system clipboard. If the variable window has the keyboard focus, <Ctrl-C> can be used as an alternative. Tuples with zero or more than one values are returned in tuple notation: [..., ...]. If several variables are selected, the tuples of the different variables are separated by a new line.

#### **Inspecting Image Acquisition Device Handles**

For an image acquisition device handle, a dialog representing basic image acquisition device parameters is opened. Here you find the size, name, device, port, and other features of the image acquisition device. The toggle button Online allows 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. The dialog is displayed in figure 6.44.

💥 Variable View - main* (:::)		🙀 Acquisition In	ispect: AcqHandle 📃 🗖 🗙
Iconic Variables		Name	uEye (Rev. 3.2)
	-	Width	1280
		Height	1024
		ColorSpace	gray
	<b>_</b>	BitsPerChannel	8
Control Variables		Port	0
AcqHandle 282 63 6788 Inspect		Device	1
✓ Update Variables		Trigger	false
		CameraType	UI154x-M
All Auto		Online 🔪	
grab im	ages —	Status	

Figure 6.44: Inspecting an image acquisition device handle.

# 6.7 Graphics Window

This window displays iconic data. It has the following properties:

- The user may open several graphics windows.
- The active graphics window is shown by the lit bulb in the window's tool bar.
- Pressing the clear button clears the graphics window content and the history of the window.
- You close a graphics window using the close button of the window frame.

Figure 6.45 shows an example graphics window which is displaying a gray value image of a tooth rim overlaid with region data. 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.

Every HDevelop graphics window has its own visualization parameters. Thus, modifying the parameters (see section "Menu Visualization" on page 78) applies to the currently active graphics window *only*, i.e., the parameter settings of all other open graphics windows remain unchanged. Additionally, the new parameter settings are used as the default settings in all graphics windows yet to be opened.

The origin of the graphics window is the upper left 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 at that position are displayed in the status bar (see page 44). Sometimes, it is desired to display this information close to the mouse cursor. This can be achieved by holding down the <Ctrl> key (note: this does not work when the zoom in and out tool is selected since pressing <Ctrl> inverts the corresponding zoom action). Figure 6.46 shows the coordinate/gray value display.

Normally, the coordinate system of the graphics window corresponds to 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 tool bar of the graphics window or the menu Menu Visualization



Figure 6.45: Graphics Window.



Figure 6.46: Coordinate display in the graphics window.

 $\triangleright$  Set Parameters...  $\triangleright$  Zoom (see section "Menu Visualization" on page 78) or with the operator dev\_set\_part (see section "Develop" on page 91). Every time an image with another size is displayed, the coordinate system will be adapted automatically.

Each window has a history that contains all

- objects and
- display parameters

that have been displayed or changed since the most recent clearing or display of an image. This history is used for redrawing the contents of the window. The history is limited to a maximum number of 30 "redraw actions", where one redraw action contains all objects of one displayed variable.

Other output like text or general graphics like disp\_line or disp\_circle or iconic data that is displayed using HALCON operators like disp\_image or disp\_region are not part of the history, and are not redrawn. Only the object classes image, region, and XLD that are displayed with the HDevelop operator dev\_display or by double-clicking on an icon are part of the history.

You may change the size of the graphics window interactively by "gripping" the window border with the mouse. Then you can resize the window by dragging the mouse pointer. After this size modification the window content is redisplayed. Now, you see the same part of the window with changed zoom.

#### **Tool Bar Icons**



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.

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 6.45 on page 125, the displayed image of a tooth rim 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. Use the mouse wheel to zoom in and out.

A Magnifying glass. Click into the graphics window to magnify the area at the mouse cursor. See figure 6.47 for an illustration of this tool.



Zoom in or out. Click the small arrow next to the icon to switch between these tools.

Set image size. Clicking this icon sets the image size to the shown value. The value can be selected from the menu attached to the small arrow. See section "Image Size" on page 79 for additional information.



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. See section "Window Size" on page 79 for additional information.

ିହ୍ ହ Show state of graphics window (active / non-active). Non-active graphics window can be activated by clicking this button. Only one graphics window may be active any given time.

If you want to specify display parameters for a window, you may select the menu item Visualization in the menu bar. Here you can set the appropriate parameters by clicking the desired item (see section



Figure 6.47: Magnifying glass.

"Menu Visualization" on page 78). 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 Help Window

The help window provides access to the integrated online help of HDevelop. The window is split in two areas: On the left, navigational aids are available as tab cards. They are described below. On the right, the online help itself is displayed. Anyone familiar with a web browser will be able to navigate through the hypertext. The size of the two parts of the help window can be altered by dragging the dividing line.

With the help window you can easily browse the HALCON Reference Manual and the HDevelop Reference Manual. Furthermore, the complete offline documentation of HALCON, which is available in PDF format, can be accessed from this window. The help window also includes a full-text search engine to rummage both online and offline documentation.

#### Contents

This tab card presents the chapters and sections of the online documentation as a hierarchical tree. Click on a node of the tree to display the associated document.

#### Operators

This tab card lists all operators in alphabetical order. Click on an operator name to display the corresponding page from the Reference Manual. Enter any substring into the text field Filter to show only operator names matching that substring.





#### Search

Enter a search query into the text field, and click Search to start a full-text search. Both online documentation (HTML) and offline documentation (PDF) are searched. The search result is displayed below the query. The rank (in percent) indicates how well each found document satisfies the query.

The query may consist of one or multiple words. HDevelop will find all documents that contain any of the specified words.

To search for a phrase, enclose it in double quotes:

```
"radiometric calibration"
```

Boolean searches with and, or, and not can also be specified. To find all documents that say anything about filters except Gaussian filters, enter:

filter not gauss



Figure 6.49: Keywords and associated HALCON operators.

#### Keywords

This tab card gives access to HALCON operators by using keywords which are associated with each operator. It contains a list all known keywords and the selected operators below. The list of keywords can be filtered by entering any substring into the text field Filter.

After the suggestions for an operator have been generated, all keywords belonging to this operator are marked (reversed). Because there are many entries in the keyword list, you may see all marked keywords only by scrolling it. On the list below the keywords you will find all operators associated with at least one of these keywords. Clicking a keyword causes the addition of operators belonging to this keyword.

#### 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 Tool Bar Icons

Go back in the browse history.

Go forward in the browse history.

Go to the starting page of the HALCON Reference Manual.





Add the currently displayed document to the tab card Bookmarks (page 129).

Den the operating system dependent printer selection dialog to print the currently displayed page.

If the currently displayed document is the reference page of a HALCON operator, select this operator in the operator window.

- Syntax The online pages of the HALCON Reference Manual are available for the language interfaces HDevelop, C++, C, .NET, and COM. The displayed variant can be selected through this list box.
- Find 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 Highlight the next match.
- Prev Highlight the previous match.

#### Special Keyboard Shortcuts in the Help Window

go back in the browse history
go forward in the browse history
go to starting the page of the HALCON Refer-
ence Manual
enter operator into operator window
search text on the current page
print current page
increase font size
decrease font size
add current page to the bookmarks
highlight next link
highlight previous link
jump to highlighted link

Text in the help window can be selected, e.g., for copying, with the mouse or the keyboard. Click into the help window to place a text cursor. This text cursor can be moved around the page using the arrow keys. Holding down shift and pressing the arrow keys starts a text selection.



Figure 6.50: Zoom.

# 6.9 Zoom Window

Synopsis: Zoom window for image details and pixel inspection.

```
See also: Menu Visualization > Zoom Window
```

The zoom window enables the interactive inspection of image details. You can open up any number of zoom windows with different zoom levels (see Menu Visualization  $\triangleright$  New Zoom Window). The window also displays the gray values of each image channel at the mouse cursor position. Apart from this, the pixel type, the number of channels, and the current position of the mouse cursor are displayed. The percental scale can be selected from the combo box. It is related to the original size of the image.

There are multiple methods to navigate the zoom window:

Check Follow Mouse and move the mouse pointer over the image to select the zoomed area. Click once to keep the currently displayed area in the zoom window, when the mouse cursor moves out of the image window. Or, uncheck Follow Mouse and click (or drag) inside the image to select the zoomed area. The red square in the center of the zoom window indicates the position of the mouse cursor. The corresponding coordinates are also displayed at the bottom of the window.

You can select a particular pixel by single-clicking on it with the left mouse button. The zooming tool stores this position internally, and will redisplay the thus selected part of the image object when you leave the graphics window. This enables you to have a meaningful display in the zooming tool whenever you want to do actions outside of the graphics window.

For finer control of the zoomed area, click inside the zoom window to give it the focus and use the cursor keys to move pixel-wise. Press and hold the Alt key and use the cursor keys to move ten pixels at a time.

Click inside the zoom window to move relative to the center position. For example, clicking ten pixels above the center will move the view up ten pixels.

The lower part of the window contains a gauge to display the gray value of the center pixel graphically. The range goes from 0 (left) to 255 (right). Normally, the gray value of the first channel is displayed with a black bar. For images with multiple channels the gauge is split accordingly to show individual bars for each channel. Thus, for color images in RGB-space (three channels with red, green, and blue values) three colored bars are used. If the gray value is below 1, the gauge is light gray (background). If the value is above 255, the gauge is dark gray or colored for RGB images.

Above the gauge, the gray values are displayed as numbers. Up to five channels are displayed. If more than five channels are present, the remaining channel values are truncated.

Next to the gauge, the coordinates of the mouse position is displayed. Below these, the image size, pixel type, and the number of channels are shown.

The button next to the scale combo box enlarges the zoom window so that partially visible pixels at the border become fully visible.

# 6.10 Gray Histogram Window

Synopsis: Display gray value histogram of active graphics window.

See also: Menu Visualization > Gray Histogram

The gray histogram window is a sophisticated tool for the inspection of gray value histograms, which can also be used to select thresholds interactively and to set the range of displayed gray values dynamically.



Figure 6.51: Gray Histogram.

When opening the tool, the histogram of the image shown in the currently 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).
- Whenever image data is displayed overlaid with region data in a graphics window (the graphics window does not need to be active for this), you can click into any of the segmented regions, and the histogram of the image within that region will be computed and shown. If you click into a part of the image that is not contained in any of the overlaid regions, the histogram of the entire image will be displayed.
- The same mechanism is used for regions that have gray value information, e.g., image objects created by reduce\_domain or add\_channels. Here, the histogram of the image object you click into will be displayed.
- Channel When a multi-channel image, e.g., a RGB color image, is sent to the tool, by default the histogram of the first channel is displayed. The combo box Channel lets you select the channel from which to compute the histogram.
- Update Sometimes, it is desirable to suppress the updating of the histogram when new image data is available, e.g., if you want to select thresholds for a gradient image, but want to visualize the original image along with the segmentation (see below). In that case you can freeze the histogram by unchecking Update. The currently displayed histogram is preserved until Update is checked again in which case the histogram will be re-calculated from the active graphics window.

The main part of the tool is the area, in which the histogram of the image is displayed in blue. This area contains static parts and parts that can be interactively manipulated. The first static part is the horizontal coordinate axis, which displays the gray values in the image. For byte images, this range is always 0...255. For all other image types, e.g., real images, the horizontal axis runs from the minimum to the maximum gray value of the image, and the labeling of the axis is changed accordingly. To the left of the display, the vertical axis representing the frequency of the gray values is drawn. The final static parts of the display are three gray arrows. The two upward pointing arrows denote the maximum and minimum gray value of the image. The downward pointing arrow denotes the peak of the histogram, i.e., the gray value that occurs most frequently. This data is displayed in textual form within the Statistics area of the display. For int4, int8, or real images, the peak value is displayed as a value range in the Statistics. That is, the range of input values is divided in quantization steps to obtain a meaningful histogram, and, as a consequence, the histogram's "peak value" may actually represent a whole range of input values.

The dynamic parts of the histogram area are the two colored lines, which can be manipulated. The vertical green and red lines denote the minimum and maximum selected gray value of the histogram, respectively. The gray values on which the two vertical lines lie are displayed next to the lines in the same color.

Display linear histogram (the default).

Display logarithmic histogram.

Initially, the histogram is displayed at full vertical range, i.e., up to the peak value. The displayed part can be manipulated with the following buttons:



Zoom histogram display to a selected area. Click this button and drag an area inside the histogram to view that area.



Spread the histogram horizontally so that only the area between the lines is displayed.

Display the full histogram.



Reset the display of the histogram vertically.



ţ₿

Reset the display of the histogram horizontally.

Force minimum histogram width. Do not adjust the width of the histogram when resizing the window.

#### Histogram Options

- Quantization: Display the histogram quantized. The bucket size can be specified with the slider or entered into the spinner box.
- Smoothing: Display the histogram smoothed. The smoothing factor can be specified with the slider or entered into the spinner box.

Horizontal/Vertical: The visible part of the histogram can be specified parametrically by entering the minimum and maximum values into the spinner boxes. These values are adapted when the visible area is set with the buttons next to the histogram.

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

The corresponding range is adapted to the new values.

increasing

The corresponding range is only adapted if it has increased, i.e., the new minimum value is smaller, or the new maximum value is greater, or both.

fixed

The corresponding range is not changed at all.

Output

The selected range of gray values can be used for two major purposes: thresholding (segmentation) and scaling the gray values.

The gray values between Min and Max can be visualized either in the originating or the active graphics window as specified in Output Destination. The type of visualization is specified in the list box Display:

none: no visualization.

• threshold: Segment image. The image from which the histogram was computed is segmented with a threshold operation with the selected minimum and maximum gray value. Depending on the setting of the combo box Display, the segmentation result is either displayed in the graphics window, from which the image was originally sent (originating window), or in the active graphics window (active window). With the three combo boxes Color, Draw, and Line Width, you can specify how the segmentation results are displayed (see also Colored, Draw, and Line Width).

If you want to select threshold parameters for a single image, display the image in the active graphics window and open the histogram tool. For optimum visualization of the segmentation results, it is best to set the visualization color to a color different from black or white. Now, set Display to threshold and interactively drag the two vertical bars until you achieve the desired segmentation result. The parameters of the threshold operation can now be read off the two vertical lines.

If you want to select threshold parameters for an image that is derived from another image, but want to display the segmentation on the original image, e.g., if you want to select thresholds for a gradient image, two different possibilities exist. First, you can display the derived image, open the histogram tool, deselect Update, display the original image, and then select the appropriate thresholds. This way, only one window is needed for the visualization. For the second possibility you can display the derived image in one window, activate another window or open a new window, display the original image there, activate the first window again, open the histogram tool, activate the second window again, set Output Destination to active window, and select your thresholds. Although in this case it is not necessary to deselect Update, it is advantageous to do so, because this prevents the histogram from being updated if you click into a graphics window accidentally.

• connection: Display the connected regions of the selected gray values in the style specified with Color, Draw, and Line Width.

This display mode is similar to threshold. 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.

• scale: Map the gray values between Min and Max to the full range (usually 0...255). See also scale\_image.

The gray values of the image are scaled such that the gray value 0 of the scaled image corresponds to the selected minimum gray value and the gray value 255 to the selected maximum gray value. Again, the combo box Output Destination determines the graphics window, in which the result is displayed. This mode is useful to interactively set a "window" of gray values that should be displayed with a large dynamic range.

# 6.11 Feature Histogram Window

Synopsis: Interactive inspection of feature histograms.

#### See also: Menu Visualization > Feature Histogram

This window provides a sophisticated tool for the inspection of feature histograms. In contrast to the gray value histogram described above, this tool does not inspect individual pixels, but regions or XLDs; for these iconic objects, it displays the distribution of values of a selected *feature*, e.g., the area of an XLD or the mean gray value of the pixels within a region. The feature histogram can also be used to select suitable thresholds for the operators select\_shape and select\_shape\_xld interactively.

Upon opening, the tool displays the histogram of the area (default feature selection) of the regions or XLDs that were displayed most recently in the currently active graphics window. You can select various features in the combo box Feature; Further information about region features can be found in section "Feature Inspection Window" on page 138.

Most parts of the tool are built up similarly to the gray value histogram, which is described in detail in section "Gray Histogram Window" on page 133 (menu entry Menu Visualization  $\triangleright$  Gray Histogram); in the following, we concentrate on points specific to the feature histogram. An important point regards the "source" of the regions or XLDs: The feature histogram is calculated for the regions or XLDs that were displayed most recently in the graphics window. Thus, if you display an image, and there are no regions or XLDs, the histogram remains "empty". As soon as you display regions or XLDs on top of an image, the histogram is calculated. If you display regions or XLDs without an image, you can still calculate feature histograms, but only for shape features. Please keep in mind that only the most recently displayed regions or XLDs are the source of the histogram, not all objects currently displayed in the graphics window!

The histogram itself is displayed with the horizontal axis corresponding to the feature values and the vertical axis corresponding to the frequency of the values, i.e., to the number of regions or XLDs with a certain feature value.

When comparing feature histograms to gray value histograms, you will note a typical difference: Because in most cases the overall number of regions or XLDs is much smaller than the overall number of pixels, feature histograms often consist of individual lines, most of them having the height 1 Of course, this effect depends on the selected feature: For features with floating-point values, e.g., the orientation, the probability that two regions or XLDs have the same feature value is very small, in contrast to features with integer values, e.g., the number of holes.

You can influence the calculation of the histogram with the slider Quantization. The selected value is used to discretize the horizontal axis: Instead of determining the frequency of an "exact" feature value, regions with feature values falling within discrete intervals are summed. Graphically speaking, the horizontal axis is subdivided into "bins" with a width equal to the value selected with the slider Quantization.

#### **Automatic Selection**

As already noted, the region feature histogram facilitates the task of finding suitable threshold parameters for the operators <u>select\_shape</u>, <u>select\_gray</u>, and <u>select\_shape\_xld</u>: Select the entry <u>selected</u> in the combo box Display, choose suitable visualization parameters in the three combo boxes Color, Draw, and Line Width, and then position the two vertical lines such that the desired regions are high-lighted.

# 6.12 Feature Inspection Window

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

See also: Menu Visualization > Feature Inspection

This window provides a tool for the convenient inspection of shape and gray value features of individual regions and XLDs. It can, for instance, be used to determine thresholds for operators that select regions based on these features, e.g., select\_shape or select\_gray.

The strategy to determine the data from which to compute the features is very similar to that of the gray histogram inspection window (see section "Gray Histogram Window" on page 133). You can display an image or region by double-clicking on it in the variable window or you can select a region or an image which is already displayed by single-clicking it. If you display or click into an image, the gray value features of the entire image will be calculated. If you click into a region that is not underlaid with an image, only the shape features of this region will be displayed. If you click into a region that is underlaid with an image or into a region that has gray value information (e.g., from reduce\_domain or add\_channels), both the shape and gray value features of that region will be displayed. Finally, if you have overlaid an image with a region, but click into a part of the image that is outside the region, only the gray value features of the entire image will be calculated.

Use the "select" tool of the graphics window to select a region or XLD. The selected region or XLD is highlighted in the graphics window. The corresponding variable name and index are displayed in the title of the feature inspection window.

The gray value features of a multi-channel image are calculated from all channels independently.

The tree on the left side of the feature inspection window groups the features into several categories. At the top-most level, the following groups of features are distinguished:

- Region features: This group contains features that describe the selected region, e.g., area, center, and orientation.
- Gray value features: The feature values of this group are calculated from the gray values of the image *under* the selected region, e.g., minimum and maximum gray value, mean gray value, anisotropy and entropy.
- XLD features: This group contains features that describe the selected XLD (e.g., its dimensions or shape properties).

You can select the features to be inspected by ticking the corresponding check boxes in the tree. The selected features are displayed on the right side of the window. For each feature the calculated value



Figure 6.52: Inspection of selected features.

of the selected region or XLD is displayed (or the value for the entire image). The current value is also visualized as a gauge in a value range that can be set to the desired values. Simply select Show Minimum/Maximum, which is available in the context menu of the right side of the window.

See figure 6.52 for an illustration of a clip inspection. The range for the *area* feature has been set to [4000, 6200]. Individual clips 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 Dialogs

# 6.13.1 File Selection Dialog

The file selection dialogs opened by actions such as Open Program..., Save , or Read Image are native windows of the operating system and thus their appearance and internal functionality is beyond HDevelop's control. Their basic functionality is to browse the file system, and to select one or multiple files (or in some cases: directories). Usually, they have two buttons: The one labeled Open or OK confirms the selection and thus performs the initial action (e.g., loading a file) while the other (labeled Cancel) aborts the initial action.

As an example	, the dialog Menu	File ⊳ Open	Program	is explained.
---------------	-------------------	-------------	---------	---------------

Choose a HDeve	op program to op	en				<u>? ×</u>
Look jn:	🔁 hdevelop		•	( <del>-</del>	r 🔝	
My Recent Documents Desktop Bellini MVTec	Applications Classification Control Develop File Filter Graphics Manuals Manuals Matching Morphology Regions Segmentation System Tools	☐ XLD 🛣 explore_halcon.de	v			
	File <u>n</u> ame: Files of <u>type</u> :	HDevelop Programs (*.dev)			•	<u>O</u> pen Cancel

Figure 6.53: Example of a file selection dialog under Windows.

In the top-most text field you may specify a directory which contains your HDevelop programs. A combo box at the right hand side helps you browsing your directories. To move one directory level up, you press the button on the right hand side of this text field. The next button creates a new folder to store HDevelop programs. By pressing the last button you can activate or deactivate the option to see more details about your HDevelop programs, i.e., the program size, the program type, the date when the most recent user update occurred, and file attributes.

The middle text area displays all available HDevelop files to choose from. By clicking the left mouse button on a file name you select it. Double-clicking a file name opens the file immediately and displays it in the program window (page 102).

Furthermore, you may specify the file name in the text field below. The combo box for file type has no effect because only HDevelop programs with the extension .dev can be loaded. If you want to open

your file with a write protection, choose the check box at the bottom of this dialog window. To open your specified file, you press the button Open. This action deletes an already loaded HDevelop program and all created variables. The same actions as with File  $\triangleright$  New Program are performed. Now you can see the main procedure body of your new program in the program window. The file name is displayed in the title bar of the main window. All its (uninstantiated) variables are shown in the variable window. To indicate that they do not have any computed values, the system provides the iconic and control variables with a question mark. The program counter is placed on top of the program body and you are ready to execute the program. The visualization and options will be reset after loading (same as Menu File  $\triangleright$  New Program, see page 46).

You can cancel this task by pressing the corresponding button. By using one of the two buttons Open or Cancel the dialog window disappears.

# 6.13.2 Unsaved Changes

File operations that will delete the current program (such as loading a new program) trigger a security check. This security check prevents you from deleting the current program accidentally if the program has not been saved. A dialog box appears and asks whether you want to save the HDevelop program before its dismissal:





- Save Save the current program under its current name and proceed. If no name has been specified yet, a file dialog pops up to enter the name.
- Save As Save the current program under a different name and proceed.
- Discard Discard unsaved changes and proceed.
- Cancel Abort the current action.

# **GUI Reference**

# **Chapter 7**

# **Matching Assistant**

# 7.1 Introducing the Matching Assistant of HDevelop

The Matching Assistant of HDevelop is a front-end to HALCON's powerful *shape-based matching*, which lets you locate objects with sub-pixel accuracy at a high speed, even when they appear rotated, partly occluded, or under changing illumination. Using the Matching Assistant you can

- configure and test the matching process with a few mouse clicks and
- optimize the parameters interactively to get the maximum matching speed and recognition rate.

All you need is a single model image and a set of test images. The Matching Assistant further assists you by automatically calculating suitable parameter values based on your selections.

How to use the Matching Assistant is described here.

A reference to the elements of the Matching Assistant can be found here (page 146).

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

- **Matching** Matching is the process of locating an object described by a *model* in an image. The results of the matching process are the position and orientation of the object and the matching *score*.
- **Model** In order to locate an object, you must provide the Matching Assistant with an example image of the object. From this, the Matching Assistant creates the so-called model, an internal representation of the object containing only the information characterizing the object. This representation is then used when searching for the object in the test images.
- Model Image This is the image containing your example of the object to be searched for. This image should be a *characteristic* image of the object, i.e., the object should appear in its default position and orientation and not be occluded; furthermore, the image should not contain clutter. You can open this image via the menu item File ▷ Open Model Image (page 147).
- Model Region of Interest (ROI) This is the region in the model image which contains the object to be found. You can mark this region via the menu item Create Model ▷ Create ROI (page 149).

- **Test Image** You can test the performance of the matching process by providing test images via the menu item Use Model ▷ Test Images ▷ Load Test Images (page 158). These images should be representative images from your matching application, i.e., the object should appear in all allowed variations of its position, orientation, occlusion, and illumination.
- **Score** When comparing a region in a test image with the model, the Matching Assistant calculates a measure of similarity, the so-called *score*, which ranges between 0 (no similarity) and 1 (perfect similarity).

# 7.2 How to Use the Matching Assistant of HDevelop

By using the Matching Assistant, you can set up and optimize your matching application quickly and easily in three steps:

- Create the model,
- Test the model, and
- Optimize the matching speed.

We recommend to reset all parameters via the button Reset (page 157) inside the tab Model Creation (page 148) before starting with a new matching application.

## 7.2.1 Creating the Model

A model (page 143) is created in three steps:

- Open the so-called *model image* (page 143) via the menu item File ▷ Open Model Image (page 147), the corresponding button in the tool bar, or the text field and button of Model Image inside the tab Model Creation.
- Create an *ROI* (page 143) around the object either via the menu items at Create Model ▷ Create ROI (page 149) or via the corresponding buttons inside the tab Model Creation.
- Specify the parameter Contrast (page 151) inside the tab Model Creation (accessible also via Create Model ▷ Standard Model Parameters (page 150)) so that the model consists of enough points to be recognizable.

Alternatively, you can load a model (page 147) that you have saved (page 148) with the Matching Assistant or HALCON.

Now, you can test the model on *test images* (page 143).

#### 7.2.2 Testing the Model

After you created the *model* (page 143) you test it in the following steps:
- Load one or more *test images* (page 143) via the menu item Use Model ▷ Test Images ▷ Load Test Images (page 158) or via the button Load inside the dialog Test Images in the tab Model Use.
- Specify standard search parameters via the menu item Use Model ▷ Standard Model Use Parameters (page 161), which opens the corresponding dialog in the tab Model Use. Especially the number of object instances (page 162) to search for in an image should be specified. If the number of object instances varies from test image to test image, you can specify the number of visible objects (page 160) for each image separately; in this case the search parameter mentioned above should be set to 0 or to the maximum number of visible objects.
- Assure that all objects are found (page 160) in all test images.

Now, you can optimize the speed of the matching process by tuning the parameters.

# 7.2.3 Optimizing the Parameters

After you configured the *matching* (page 143) process such that the search is successful in all test images, you can start to optimize the parameters to speed up the matching as far as possible.

To support this process, the Matching Assistant allows to optimize the search parameters Minimum Score (page 162) and Greediness (page 163) automatically via the menu item Use Model  $\triangleright$  Optimize Recognition Speed (page 164), which can be accessed also via the tab Model Use.

If the reached recognition speed is not sufficient, you can try to modify parameters manually. However, please be aware that such a modification may result in a lower accuracy of the calculated position, orientation, or scale, or even prevent the Matching Assistant from finding the object! Therefore, we recommend to check whether the matching still succeeds in all *test images* (page 143) after each modification.

How the different parameters influence the recognition speed is described below. Please note that whenever you modify a model parameter, the internally stored model must be created anew; you must start this creation (and the search) explicitly using the button Find Model or the button Detect All in the tab Model Use. After each modification determine the resulting recognition speed using the dialog Optimize Recognition Speed (page 164).

The following modifications can speed up the matching processes:

Create Model > Standard Model Parameters (page 150):

• Number of Pyramid Levels (page 152)

Increase the value and check whether the matching still succeeds in all images.

• Allowed ranges of rotation (page 153) and scale (page 154)

Set the parameters Start Angle (page 153), Angle Extent (page 153), Min Row Scale (page 154), Max Row Scale (page 154), Min Column Scale (page 154), and Max Column Scale (page 154), according to the ranges probably needed for your images.

Create Model > Advanced Model Parameters (page 154):

• Minimum Contrast (page 157)

Increase the value and check whether the matching still succeeds in all images.

Optimization (page 156) (Point Reduction)

Select a higher reduction rate and check whether the matching still succeeds in all images.

• Angle Step (page 155) size and Scale Step (page 155) size

Increase the values and check whether the matching still succeeds in all images. Please note that the accuracy may suffer if you increase the step size!

Use Model ▷ Advanced Model Use Parameters (page 162):

• Subpixel (page 163)

If your application doesn't require sub-pixel accuracy, you can speed up the matching by selecting the value 'none'.

Last Pyramid Level (page 163)

Increase the value and check whether the matching still succeeds in all images. Note that as a result of this modification wrong instances of the model may be found. Furthermore, the accuracy of the calculated position, orientation, and scale may decrease.

# 7.3 Matching Assistant Reference

The Matching Assistant consists of the following elements:

- the pull-down menus File, Create Model (page 148), Use Model (page 158), Inspect (page 165), Code Generation (page 166), and Help (page 168),
- a tool bar with a selection of important buttons (Load Assistant Settings (page 148), Save Current Assistant Settings (page 148), Insert Code (page 167), Open Model Image, Display Model (page 149), Optimize Recognition Speed (page 164), and Determine Recognition Rate (page 165)),
- tabs with the dialogs for most of the tasks that can be done with the Matching Assistant (Model Creation (page 148), Model Use (page 158), Inspect (page 165), Code Generation (page 166)), and
- a status bar at the bottom in which messages are displayed. Futhermore, the status bar displays the matching results, i.e., the number of found instances, the needed time, and for each found instance the position, orientation, scale, and score. Please note that the status bar does not provide a scrolling mechanism; if the displayed message is to long, move the mouse over it, so that a tool tip displaying the full message pops up. Alternatively, if the message is only slighly larger than the status bar, you can also drag the left or right border of the Matching Assistant window to enlarge it.

Images and models are displayed in the graphics window of HDevelop.

# 7.3.1 The Menu File

Via the menu File you can

- open the model image,
- load an already existing shape model, •
- save a shape model,
- load formerly used and saved settings of the Matching Assistant, •
- save the current settings of the Matching Assistant for later use,
- close the Matching Assistant dialog (while retaining the current settings as long as the HDevelop session is active), and
- exit the Matching Assistant dialog (discarding the settings).

#### 7.3.1.1 **Opening the Model Image**

The so-called *model image* (page 143) is used to create the *model* (page 143) of the object you want to find later. This image should be a *characteristic* image of the object, i.e., the object should appear in its default position and orientation and not be occluded; furthermore, the image should not contain clutter.

When you select the menu item File > Open Model Image or press the corresponding button either in the tool bar or in the dialog Model Image in the tab Model Creation, a standard file selection box appears. The Matching Assistant can read the image file types TIFF, BMP, GIF, JPEG, PPM, PGM, PNG, and PBM.

The selected image is displayed automatically. Typically, the next step is to create a region of interest (page 149) around the object.

As an alternative to loading a model image and creating the model (page 144) interactively, the menu item File > Load Model can be used to load a model that you have saved with the Matching Assistant or HALCON.

# 7.3.1.2 Loading a Shape Model

As an alternative to opening a *model image* (page 143) and creating (page 144) the *model* (page 143) interactively, the menu item File > Load Model or the corresponding button Load in the tab Model Use can be used to load a model that you have saved with the Matching Assistant or HALCON.

Note that when you load the model from a file, all the menu items, buttons, and dialogs that enable you to change the model parameters or display the model image will not be selectable because a loaded model cannot be changed and contains no information about the image from which it was created. Thus, e.g., the menu items Create Model > Create ROI (page 149), Create Model > Standard Model Parameters (page 150), and Create Model > Advanced Model Parameters (page 154), and the Display button of the dialog accessed by Create Model ▷ Display Image Pyramid (page 149), which is used to inspect the model, are enabled.

#### 7.3.1.3 Saving a Shape Model

The menu item File  $\triangleright$  Save Model enables you to save the created model (page 144) in a file for later use. For example, the *model* (page 143) can be loaded into the Matching Assistant again in a later session with File  $\triangleright$  Load Model.

#### 7.3.1.4 Loading Assistant Settings

If you have saved the settings of a former Matching Assistant session, you can load them again by the menu item File > Load Assistant Settings or via the corresponding button of the tool bar.

#### 7.3.1.5 Save Current Assistant Settings

You can save the current settings of a Matching Assistant session using the menu item File  $\triangleright$  Save Current Assistant Settings or the corresponding button in the tool bar. Then, you can load them again in a later session.

#### 7.3.1.6 Close the Matching Assistant Dialog

When closing the Matching Assistant dialog with the menu item File  $\triangleright$  Close Dialog, the current settings are stored for the duration of the current HDevelop session. That is, aslong as you do not exit HDevelop, you can again open the Matching Assistant with the same settings. In contrast to this, when you exit the Matching Assistant, the settings are lost also for the current HDevelop session.

#### 7.3.1.7 Exit the Matching Assistant

When you exit the Matching Assistant with the menu item File > Exit Assistant, the assistant's dialog is closed and the current settings are lost unless you have not stored them via the menu item File > Save Current Assistant Settings. If you want to close the dialog so that the settings are retained for the current HDevelop session, you should use the menu item Close Dialog instead.

# 7.3.2 The Menu Create Model and the Tab Model Creation

Via the menu Create Model as well as the tab Model Creation you can

- create a model ROI,
- display the image pyramid, and
- specify standard (page 150) and advanced model parameters (page 154).

In the tab Model Creation you can additionally reset (page 157) the model.

# 7.3.2.1 Creating a Region of Interest Around the Object

Via the menu items in Create Model  $\triangleright$  Create ROI or the corresponding buttons in the tab Model Creation you can mark the region that serves as the model by *drawing* it on the displayed model image. The Matching Assistant provides different *ROI* (page 143) shapes: axis-parallel and arbitrarily oriented rectangles, circles and ellipses, as well as free-form shapes including polygons.

You draw rectangular, circular, and elliptic ROIs as follows: Select the corresponding drawing mode and click into the image. Then, move the mouse over the object while keeping the left mouse button pressed; the selected shape appears. After releasing the mouse button you can move the ROI by dragging its center (marked with a cross) with the left mouse button. Furthermore, you can edit the shape by dragging its boundaries. You finish the creation by clicking once with the right mouse button or by clicking the Stop button in the tool bar of the main window.

By selecting the menu item Create Model > Create ROI > Arbitrary Region or the corresponding button in the tab Model Creation you can create polygons and free-form shapes. To create a polygon click with the left mouse button to mark each corner point; a click with the right mouse button closes the polygon and finishes the creation. To create a free-form ROI draw it directly while keeping the left mouse button pressed; a click with the right mouse button closes the shape and finishes the creation. Note that in both cases you cannot edit the ROI after its creation!

In order to create an optimal model, please assure that the region of interest contains only characteristic parts of the object and no clutter!

After creating an ROI, you can specify standard model parameters. Typically, you now select what contrast (page 151) the points must have in order to be included in the model.

# 7.3.2.2 Displaying the Model Image

Pressing the button Display Model in the tool bar of the Matching Assistant, you can display the model image if available (if you loaded a shape model (page 147) from file, the model image is not available).

You can alternatively display the model image via the button Display in the dialog Display Image Pyramid of the tab Model Creation (accessed also via the menu item Create Model ▷ Display Image Pyramid). If you already created a model *ROI* (page 143), the model itself is displayed as well. When increasing the values for Image and Model using the sliders or the text boxes, you can display the pyramid levels (see the corresponding section about pyramid images).

# 7.3.2.3 Displaying the Image Pyramid

Using the dialog Display Image Pyramid (accessed via the menu item Create Model > Display Image Pyramid or directly inside the tab Model Creation), you can display the *model image* (page 143) (see how to display the model image) and inspect the pyramid of models and the corresponding images by

- selecting which model level is displayed,
- selecting which image level is displayed, and
- locking or unlocking model and image level.

## 7.3.2.4 Displaying the Model on the Different Pyramid Levels

You can select the desired pyramid level of the model by using the slider or text box for Model inside the dialog Display Image Pyramid of the tab Model Creation. The model is overlaid onto the pyramid image selected with the slider or text box Image within the same dialog. By default, the model and the image are displayed on the same pyramid level; you can unlock and again lock the levels using the lock/unlock button right to the sliders.

Note that the highest available pyramid level is determined automatically by the Matching Assistant based on the size of the model *ROI* (page 143); depending on the selected Contrast and Minimum Component Size (page 152), higher pyramid levels may not contain any model points.

Detailed information about the model image pyramid can be found here (page 152).

### 7.3.2.5 Displaying the Model Image on the Different Pyramid Levels

You can select the desired pyramid level of the model image using the slider or text box for Image inside the dialog Display Image Pyramid of the tab Model Creation. Onto this image, the model on the pyramid level selected with the slider or text box for Model within the same dialog is overlaid. By default, the model and the image are displayed on the same pyramid level; you can unlock and again lock the levels using the lock/unlock button right to the sliders.

Note that the highest available pyramid level is determined automatically by the Matching Assistant based on the size of the model *ROI* (page 143); depending on the selected Contrast and Minimum Component Size (page 152), higher pyramid levels may not contain any model points.

Detailed information about the model image pyramid can be found here (page 152).

### 7.3.2.6 Locking the Display of Model and Image Pyramid

By default, the pyramid levels of the displayed model and model image are locked. When pressing the unlock button right to the sliders, which are used for specifying the pyramid levels, you can select different pyramid levels for the model image and the model. When pressing the button again, both levels are locked again.

Detailed information about the model image pyramid can be found here (page 152).

### 7.3.2.7 Specifying Standard Model Parameters

Via the menu item Create Model  $\triangleright$  Standard Model Parameters the tab Model Creation is opened and you can specify basic parameters for the model, which describe the *appearance* of the object to recognize, e.g., the contrast of significant points or the allowed range of rotation.

By default, these parameters are set to values which work well for most tasks; by modifying them you can optimize the model for your application and speed up the search process.

The following parameters can be specified in this dialog:

• the Contrast which points must have in order to be included in the model,

- the Minimum Component Size (page 152) of model components,
- the number of Pyramid Levels (page 152) on which the model is created,
- the Start Angle (page 153) of the allowed range of rotation,
- the allowed range of rotation (Angle Extent (page 153)), and
- the scale range (page 154).

In most applications, specifying the standard parameters will already suffice. Therefore, you can directly test the model (page 144) now. Additionally, advanced model parameters can be specified via the menu item Create Model  $\triangleright$  Advanced Model Parameters (page 154).

## 7.3.2.8 The Model Parameter Contrast (Low/High)

The two parameters Contrast (Low) and Contrast (High) determine which pixels in the selected *ROI* (page 143) are included in the *model* (page 143); typically, the points corresponding to the contours of the object should be selected.

When you select a value, either by using the sliders or by entering a value in the text fields next to them, the included pixels are marked in the displayed image. In order to obtain a suitable model we recommend to choose the contrast in such a way that the *significant* pixels of the object are included, i.e., those pixels that characterize it and allow to discriminate it clearly from other objects or from the background. Please assure that no clutter is included, i.e., pixels that do not belong to the object!

You can use the parameters in two ways:

#### 1. Simple threshold:

Set both parameters to the same value. Then, all pixels with a contrast higher than this value are included in the model.

You can modify both parameters at the same time as follows: To increase the value, use the slider of Contrast (Low); then, the value Contrast (High) will follow automatically. Vice versa, to decrease the value use the slider of Contrast (High).

#### 2. Hysteresis threshold:

If there is no single contrast value that selects all significant object pixels without including clutter, try using different values for Contrast (Low) and Contrast (High). Then, pixels are selected in two steps: First pixels that have a contrast higher than Contrast (High) are selected; then, pixels that have a contrast higher than Contrast (Low) and that are connected to a high-contrast pixel, either directly or via another pixel with contrast above the lower threshold, are added.

We recommend to proceed as follows: Increase both values (using the slider of Contrast (Low)), until no clutter pixels are selected anymore. Then, decrease Contrast (Low) to add more object pixels. If significant object parts remain unselected, decrease Contrast (High).

Note that these parameters are used only to select model points in the model image. In the test images, the object may have a lower contrast.

You can also let the Matching Assistant select suitable values automatically based on the model image.

An additional method for removing clutter is to specify a minimum size for the model components. If you cannot find suitable parameter values that exclude the clutter, we recommend to create a new model ROI via the menu item Create Model  $\triangleright$  Create ROI (page 149).

#### 7.3.2.9 Letting the Matching Assistant Select a Suitable Value for Contrast

When you click the button Auto Select that is placed right beside the sliders for the parameters Contrast (Low/High) (page 151) the Matching Assistant selects suitable values for the contrast by trying to obtain many long and straight contour segments.

Note that you may need to set the value manually if certain model components should be included or suppressed because of application-specific reasons or if the object contains several different contrasts.

#### 7.3.2.10 The Model Parameter Minimum Component Size

The parameter Min Component Size specifies the minimum size, i.e., number of pixels, which contour parts must have to be included in the *model* (page 143). This parameter is useful to exclude clutter.

You can also let the Matching Assistant select a suitable value automatically based on the *model image* (page 143).

Note that the selected value is divided by two for each successive pyramid level.

# 7.3.2.11 Letting the Matching Assistant Select a Suitable Value for Minimum Component Size

When you click the button Auto Select that is placed right beside the slider for the parameter Minimum Component Size the Matching Assistant selects a suitable value for the minimum component size based on the model image.

### 7.3.2.12 The Model Parameter Pyramid Levels

To speed up the matching process, a so-called *image pyramid* is created, both for the model image and for the search images. The pyramid consists of the original, full-sized image and a set of downsampled images. For example, if the original image (first pyramid level) has the size 600x400, the second level image has the size 300x200, the third level 150x100, and so on. The object is then searched first on the highest pyramid level, i.e., in the smallest image. The results of this fast search are then used to limit the search in the next pyramid image, whose results are used on the next lower level until the lowest level is reached. Using this iterative method the search is both fast and accurate.

You can inspect the model image pyramid together with the corresponding models via the menu item Create Model > Display Image Pyramid (page 149), which opens the corresponding dialog of the tab Model Creation. We recommend to choose the highest pyramid level at which the model contains at least ten pixels (and still resembles the original shape). You can enter the value directly in the text field or by using the slider next to it. Alternatively, you can let the Matching Assistant select a suitable value automatically.

Note that the Matching Assistant can check whether the model contains enough points on the selected number of pyramid levels only when actually creating the model. In case the model does not contain enough model points a corresponding error dialog appears.

## 7.3.2.13 Letting the Matching Assistant Select a Suitable Value for Pyramid Levels

When you click the button Auto Select that is placed right beside the slider for the parameter Pyramid Levels (page 152) the Matching Assistant selects a suitable number of pyramid levels automatically, thus relieving you of the task of examining the model image pyramid.

Please note that in rare cases the automatic selection will yield a too low value and thereby slow down the search process, or a too high value, resulting in failures to recognize the object. In such a case we recommend to inspect the model image pyramid (page 149) and select a suitable value manually.

# 7.3.2.14 The Model Parameter Start Angle

With the parameter Start Angle you can specify the starting angle of the allowed range of rotation (unit:°). With another parameter you can specify the extent of the allowed range. Note that the range of rotation is defined relative to the model image, i.e., a starting angle of 0° corresponds to the orientation the object has in the model image. Therefore, to allow rotations up to +/-5°, e.g., you should set the starting angle to  $-5^{\circ}$  and the angle extent to  $10^{\circ}$ .

### 7.3.2.15 The Model Parameter Angle Extent

With the parameter Angle Extent you can specify how much the object is allowed to rotate (unit:°). With another parameter you can specify the starting angle of this allowed range. Note that the range of rotation is defined relative to the model image, i.e., a starting angle of 0° corresponds to the orientation the object has in the model image. Therefore, to allow rotations up to  $+/-5^{\circ}$ , e.g., you should set the starting angle to  $-5^{\circ}$  and the angle extent to  $10^{\circ}$ .

We recommend to limit the allowed range of rotation as much as possible in order to speed up the search process and to minimize the required memory. If the loaded test images (page 158) show the object in its extreme orientations, you can let the Matching Assistant determine the range of rotation, i.e., the Pose Bounds (page 166), by pressing the Run button of the tab Inspect and viewing the result in the Statistics output of the same tab.

Furthermore, you must limit the allowed range if the object is (almost) symmetrical. Otherwise the search process will find multiple, almost equally good matches on the same object at different angles; which match (at which angle) is returned as the best can therefore "jump" from image to image. The suitable range of rotation depends on the symmetry: For a cross-shaped or square object the allowed extent must be less than  $90^{\circ}$ , for a rectangular object less than  $180^{\circ}$ , and for a circular object  $0^{\circ}$ .

Note that if you have chosen a very large angle and scale range you may find it useful to switch off the complete pregeneration (page 157) of the model.

### 7.3.2.16 The Model Parameters for the Scale Range

The allowed range of scale is defined separately in row and column direction. Thus, it is described by the parameters:

- Minimum Row Scale
- Maximum Row Scale
- Minimum Column Scale
- Maximum Column Scale

In the model image, the scales all have the value 1.0.

Note that if you have chosen a very large angle extent (page 153) and scale range you may find it useful to switch off the complete pregeneration (page 157) of the model.

Depending on the specified parameters, the most efficient matching method is used. This method determines how the shape model is created in the generated code.

#### • Unscaled matching:

This method is used if all four scale factors are equal to 1.0.

• Scale invariant matching:

This method is used if all four scale factors are equal (but not 1.0) or locked.

• Anisotropic scale invariant matching:

This method is used if none of the above applies.

#### 7.3.2.17 Specifying Advanced Model Parameters

In most applications, specifying the Standard Model Parameters (page 150) will already suffice. The menu item Create Model  $\triangleright$  Advanced Model Parameters provides additional parameters that let you handle special cases like changing the contrast polarity or enable you to further optimize the model.

The following parameters can be specified in this dialog:

- the Angle Step at which the model is created,
- the scale steps at which the model is created,
- whether to use the polarity of the contrast (Metric (page 156)) in the model,
- whether to optimize the model (page 156) by using a reduced number of points,
- whether to pregenerate the model completely (page 157), and
- the Minimum Contrast (page 157) points must have in a search image to be compared with the model.

# 7.3.2.18 The Model Parameter Angle Step

The standard model parameters **Start Angle** (page 153) and **Angle Extent** (page 153) specify how much the object is allowed to rotate. To speed up the matching process the Matching Assistant precomputes instances of the model at intermediate angles in this range, at steps specified in the parameter **Angle Step**.

Note that each time you create a model *ROI* (page 143) or change the parameter Contrast (page 151), the Matching Assistant automatically selects a suitable value to obtain the highest possible accuracy. You can select a higher value manually. This may be useful to speed up the search process in special cases; please note however, that a large value may decrease the accuracy of the estimated orientation and even prevent the Matching Assistant from finding the object! You can restore the automatically selected value by clicking the button Auto Select.

If you already loaded test images (page 158) you can quickly test the effect of the selected parameter value via the menu item Inspect > Determine Recognition Rate (page 165).

# 7.3.2.19 Letting the Matching Assistant Select a Suitable Value for Angle Step

When you click the button Auto Select that is placed right beside the slider for the parameter Angle Step the Matching Assistant selects a suitable value for the angle step size to obtain the highest possible accuracy.

### 7.3.2.20 The Model Parameters Row Scale Step and Column Scale Step

The standard model parameters for the scale range (page 154) specify how much the object is allowed to be scaled in row and column direction. To speed up the matching process the Matching Assistant precomputes instances of the model at intermediate scales in this range, at steps specified in the parameters Row Scale Step and Column Scale Step.

Note that each time you create a model *ROI* (page 143) or change the parameter Contrast (page 151), the Matching Assistant automatically selects a suitable value to obtain the highest possible accuracy. You can select a higher value manually. This may be useful to speed up the search process in special cases; please note however, that a large value may decrease the accuracy of the estimated orientation and even prevent the Matching Assistant from finding the object! You can restore the automatically selected value by clicking the button Auto Select.

If you already loaded test images (page 158) you can quickly test the effect of the selected parameter value via the menu item Inspect > Determine Recognition Rate (page 165).

# 7.3.2.21 Letting the Matching Assistant Select a Suitable Value for Row Scale Step and Column Scale Step

When you click the button Auto Select that is placed right beside the sliders for the parameters Row/Column Step Size the Matching Assistant selects suitable values for both scale step sizes based on the model image.

### 7.3.2.22 The Model Parameter Metric

The parameter Metric lets you choose whether the *polarity of the contrast* is to be observed when comparing a test image with the model. By default, the polarity is used ('use\_polarity'), i.e., the points in the test image must show the same direction of the contrast as the corresponding points in the model.

You can choose to ignore the polarity *globally* ('ignore\_global\_polarity'), at the cost of a slightly lower recognition speed. In this mode, an object is recognized also if the direction of its contrast reverses, e.g., if your object can appear both as a dark shape on a light background and vice versa.

A third mode lets you ignore the polarity *locally* ('ignore\_local\_polarity'), i.e., objects are also recognized if the direction of the contrast changes only in some parts. This mode can be useful, e.g., if the object consists of a part with a medium gray value, within which either darker or brighter sub-objects lie. Please note, however, that the recognition speed decreases dramatically in this mode, especially if you allowed a large range of rotation (page 153).

Finally, you can choose to ignore the color polarity ('ignore\_color\_polarity') to apply shape based matching to multi-channel images.

If you already loaded test images (page 158) you can quickly test the effect of the selected parameter value via the menu item Inspect > Determine Recognition Rate (page 165).

### 7.3.2.23 The Model Parameter Optimization

After you created a model *ROI* (page 143), by default all points showing the required Contrast (page 151) (and belonging to components larger than the Minimum Size (page 152)) are selected for the *model* (page 143) and marked in the image. For particularly large models, i.e., a large number of model points, it might be useful to reduce the number of points using the parameter Optimization in order to speed up the *matching* (page 143) process and to reduce memory requirements. You can select a low, medium, or high point reduction; please note that regardless of your selection all points passing the contrast criterion are displayed, i.e., you cannot check which points are part of the model.

You can also let the Matching Assistant select a suitable value automatically based on the model image.

Another possibility to reduce the memory requirements of the model is to switch off the complete pregeneration of the model.

If you already loaded test images (page 158) you can quickly test the effect of the selected parameter value via the menu item Inspect > Determine Recognition Rate (page 165).

#### 7.3.2.24 Letting the Matching Assistant Select a Suitable Value for Optimization

When you click the button Auto Select that is placed right beside the slider for the parameter Optimization the Matching Assistant optimizes, i.e., reduces the number of model points based on the model image.

# 7.3.2.25 The Model Parameter Pregenerate Shape Model

The parameter Pregenerate Shape Model specifies whether the internal representation of the shape model is pregenerated completely whenever the model is created.

If you select a complete pregeneration by checking the check box Pregenerate Shape Model the model generation may require a substantial amount of time and memory. In contrast, if you switch off the complete pregeneration, the model creation will be very fast and the model will consume less memory.

The advantage of selecting a complete pregeneration is that the model can typically be found slightly faster than if the complete pregeneration is switched off. Typically, you may find it useful to switch off the complete pregeneration if your model uses a large angle and scale range.

# 7.3.2.26 The Model Parameter Minimum Contrast

In order to select significant object points for the *model* (page 143) you specified which Contrast (page 151) the points must show in the *model image* (page 143). With the parameter Minimum Contrast you can specify a separate minimum contrast for the *matching* (page 143) process itself, i.e., when searching for the object in the *test images* (page 143). The main use of this parameter is to exclude noise, i.e., gray value fluctuations, from the matching process.

Note that a low value for Minimum Contrast slows down the matching process because more points in the test image must be compared with the model. Therefore, we recommend to choose a value which is higher than the noise in the test images. You can also let the Matching Assistant select a suitable value automatically based on the model image.

Note that although this parameter is only used during the search, it is already included when creating the model in order to speed up the matching process.

If you already loaded test images you can quickly test the effect of the selected parameter value via the menu item Inspect > Determine Recognition Rate (page 165).

# 7.3.2.27 Letting the Matching Assistant Select a Suitable Value for Minimum Contrast

When you click the button Auto Select that is placed right beside the slider for the parameter Minimum Contrast the Matching Assistant selects a suitable value for the minimum contrast by evaluating the gray value fluctuations, i.e., the noise in the model image.

Note that an automatic determination only makes sense if the image noise during the recognition is similar to the noise in the model image. For this reason, it is typically not useful when using a synthetic model image (without noise).

# 7.3.2.28 Reset All Parameters

The button Reset inside the tab Model Creation resets all model and search parameters to their default settings and deletes the *model image* (page 143), the model *ROI* (page 143), and the *test images* (page 143).

# 7.3.3 The Menu Use Model and the Tab Model Use

Via the menu Use Model as well as the tab Model Use you can

- load test images,
- delete a selected test image,
- delete all test images at once,
- display the selected test image (page 160),
- access the test image settings in the tab Model Use,
- open the dialog for the standard (page 161) and advanced search parameters (page 162),
- open the dialog for the optimization of the recognition speed (page 164), and
- directly start to optimize the recognition speed (page 164).

In the tab Model Use you can additionally

- select a test image (page 160),
- specify the number of visible objects (page 160) in the image, and
- start the matching for a selected test image or
- for the whole sequence of test images.

### 7.3.3.1 Loading Test Images

The so-called *test images* (page 143) should be representative images from your matching application, i.e., the object should appear in all allowed variations of its position, orientation, occlusion, and illumination.

When you select the menu item Use Model  $\triangleright$  Test Images  $\triangleright$  Load Test Images (or click the corresponding button Load in the tab Model Use), a standard file selection box appears, in which you can select one or more images to load. The Matching Assistant can read the image file types TIFF, BMP, GIF, JPEG, PPM, PGM, PNG, and PBM. Please note that the test images must have the same size as the model image!

A dialog appears in the tab Model Use which enables you to test the matching on the loaded images.

# 7.3.3.2 Deleting a Test Image

When you select the menu item Use Model  $\triangleright$  Test Images  $\triangleright$  Delete Test Image or click the button Delete inside the dialog Test Images of the tab Model Use, the currently selected test image is deleted from the list of test images. You can select a test image (page 160) by clicking onto its index number or path in the text field left to the buttons.

You can also delete all test images at once.

## 7.3.3.3 Deleting All Test Images

When you select the menu item Use Model > Test Images > Delete All Test Images or click the button Delete All in the dialog Test Images of the tab Model Use, all test images are deleted from the list of test images.

You can also delete a selected test image (page 158).

## 7.3.3.4 Test Images

With the menu item Use Model > Test Images you can

- load (page 158) test images,
- delete (page 158) a selected test image or delete all test images,
- display an already selected test image, and
- open the dialog Test Images inside the tab Model Use

The dialog Test Images inside the tab Model Use you need to additionally

- select a test image for display or deletion,
- specify the number of visible objects for each image, and
- search for the model in the complete sequence of test images, in the currently selected test image, or automatically after each selection.

# 7.3.3.5 Searching for the Object in a Test Image

When you click the button Find Model in the dialog Test Images of the tab Model Use the object is searched for in the currently selected test image; the result is displayed in the graphics window.

Please note that if the button is clicked for the first time or after you changed a model parameter, the internally stored model is actually created, which takes some time. If the model creation takes a long time (i.e., if you have chosen a very large angle (page 153) and scale range (page 154)), you may find it useful to switch off the complete pregeneration (page 157) of the model.

You can also search for the object in the whole sequence of test images at once.

# 7.3.3.6 Searching for the Object in All Test Images

When you click the button Detect All inside the dialog Test Images of the tab Model Use, the object is searched for in the complete sequence of test images that were loaded (page 158) before. The results are displayed successively in the graphics window.

Please note that if the button is clicked for the first time or after you changed a model parameter, the internally stored model is actually created, which takes some time. If the model creation takes a long time (i.e., if you have chosen a very large angle (page 153) and scale range (page 154)), you may find it useful to switch off the complete pregeneration (page 157) of the model.

You can also search for the object in a single test image.

#### 7.3.3.7 Automatically Searching for the Object in the Test Images

If you check the box Always Find in the dialog Test Images of the tab Model Use (also accessible via the menu item Use Model  $\triangleright$  Test Images  $\triangleright$  Show Test Image Settings), the object is searched for automatically whenever you select a new test image.

Please note that if the matching process is started for the first time or after you changed a model parameter, the internally stored model is created, which takes some time. If the model creation takes a long time (i.e., if you have chosen a very large angle and scale range), you may find it useful to switch off the complete pregeneration (page 157) of the model.

#### 7.3.3.8 Selecting and Displaying a Test Image

You can select a test image by clicking with the left mouse button onto its number (index) or path in the text box of the dialog Test Images of the tab Model Use. The selected image is automatically displayed in the graphics window of HDevelop.

If the checkbox labelled Always Find is checked, the matching process is started automatically on the selected test image; its result is displayed in the graphics window.

If you want to redisplay the selected test image in a later step, e.g., after you displayed the model image (page 149) again, you can also display it via the menu item Use Model  $\triangleright$  Test Images  $\triangleright$  Display Selected Test Image without newly selecting it.

#### 7.3.3.9 Specifying the Number of Objects Visible in a Test Image

In the dialog Test Images (page 159) in the tab Model Use, you can specify how many objects are visible in the current test image using the corresponding text box that appears when clicking onto the currently displayed number of visible objects in the text field of the currently selected test image. The default value is 1.

If you select the corresponding recognition mode in the dialog accessed via Use Model  $\triangleright$  Go To Optimize Recognition Speed (page 164), the specified numbers of visible objects are used when determining the recognition rate, i.e., the recognition rate is 100% when the sum of all objects found in the test images is equal to the sum of the specified numbers.

#### 7.3.3.10 Assuring the Matching Success

After loading (page 158) the test images you can quickly test whether all objects are found successfully via the dialog Inspect > Determine Recognition Rate (page 165). If the matching succeeds in all test images, i.e., if a recognition rate of 100% is reached, you can start to optimize the speed (page 145) of the matching process.

If the matching fails in one or more test images, proceed as follows:

- Open the dialog Test Images (page 159) in the tab Model Use.
- Check the box Always Find.

- Step through the test images to determine the images where the matching fails.
- If an object is not found check whether one of the following situations causes your problem:
  - Is the object crossing the image border, i.e., does it lie partially outside the test image?

By default the objects must lie completely within the test image in order to be found. This behavior can be changed in the dialog Advanced Model Use Parameters in the tab Model Use via the parameter Shape models may cross the image border (page 164).

• Is the Matching Assistant too greedy ?

By default, the Matching Assistant uses a fast search heuristic which might *overlook* an object. Therefore, try reducing the corresponding parameter Greediness (page 163) in the dialog Advanced Model Use Parameters manually or automatically via the menu item Inspect > Optimize Recognition Speed (page 164).

• Is the object partly occluded?

If the object is to be recognized in this state nevertheless, try reducing the parameter Minimum Score in the dialog Standard Model Use Parameters in the tab Model Use manually or automatically via the menu item Inspect > Optimize Recognition Speed (page 164).

• Has the object a low contrast?

If the object is to be recognized in this state nevertheless, try reducing the parameter Minimum Contrast (page 157) in the dialog Standard Model Parameters (page 150) in the tab Model Creation.

• Do multiple objects overlap?

If the objects are to be recognized in this state nevertheless, try decreasing the Maximum Overlap (page 163) in the dialog Advanced Model Use Parameters in the tab Model Use.

- If the object is found but not at the expected position or orientation check the following:
  - If multiple matches are found on one and the same object, decrease the Maximum Overlap (page 163) in the dialog Advanced Model Use Parameters.
  - If an almost symmetric object is found at the *wrong* orientation try reducing the parameters specifying the allowed range of rotation (page 153) in the dialog Standard Model Parameters (page 150) in the tab Model Creation.

### 7.3.3.11 Specifying Standard Model Use Parameters

Via the menu item Use Model > Standard Model Use Parameters, you can specify

- the Minimum Score the object must have and
- the number of instances of the object that are searched for in an image (Maximum Number of Matches).

Additionally, advanced search parameters can be specified via the menu item Use Model > Advanced Model Use Parameters.

### 7.3.3.12 The Search Parameter Minimum Score

When comparing a region in a test image with the *model* (page 143), the Matching Assistant calculates a measure of similarity, the so-called *score* (page 144), which ranges between 0 (no similarity) and 1 (perfect similarity). With the parameter Minimum Score you can specify a minimum score that a match must reach.

Graphically speaking, the parameter specifies how much of the object, i.e., of the model points, must be visible. A part of the object may be *invisible* not only because it is occluded, but also if its contrast is lower than the selected minimum contrast value (page 157) or has the wrong polarity (page 156). A further cause of invisibility could be a (too) large angle step size (page 155).

The larger the value is chosen, the faster the search is, because candidate matches can be discarded earlier. Therefore, this parameter can be optimized easily: Starting from the maximum value, reduce the value until the object is found in all *test images* (page 143); in fact, this method is used by the Matching Assistant itself when you start the optimization via the menu item Inspect  $\triangleright$  Optimize Recognition Speed (page 164).

Choosing small values may cause the program to search for quite a while. In such a case we recommend to enter a larger value in the text box instead of using the slider.

Please note that by default the objects must lie completely within the test images in order to be found. This behavior can be changed via the parameter Shape models may cross the image border (page 164) in the dialog accessed via the menu item Use Model  $\triangleright$  Advanced Model Use Parameters.

### 7.3.3.13 The Search Parameter Maximum Number of Matches

The parameter Maximum Number of Matches specifies how many instances of the object are searched for in the image. Note that the parameter sets a maximum value, i.e., if more object instances are present in the image only the best instances of the specified number are displayed. If you specify the value 0, all found instances are displayed.

### 7.3.3.14 Specifying Advanced Model Use Parameters

Via the menu item Use Model > Advanced Model Use Parameters, you can specify:

- the Greediness of the search algorithm,
- how much the objects may overlap (Maximum Overlap),
- the accuracy (Subpixel) of the calculated position, orientation, and scale,
- the lowest pyramid level Last Pyramid Level to which the found matches are tracked, and
- whether objects that lie partially outside the image (Shape model may cross the image border (page 164)) should be searched.

# 7.3.3.15 The Search Parameter Greediness

The parameter Greediness influences the search algorithm used by the Matching Assistant. It ranges between 0 and 1. If you select a low value, the search is thorough but relatively slow. The higher the value, the faster the search algorithm becomes, but at the cost of thoroughness, i.e., an object might not be found even though it is visible in the image.

This parameter can be optimized easily: Starting from the value 0, increase the value until the matching fails in a test image, and then use the last value for which the object is found; in fact, this method is used by the Matching Assistant itself when you start the optimization via the menu item Use Model  $\triangleright$  Optimize Recognition Speed.

# 7.3.3.16 The Search Parameter Maximum Overlap

The parameter Maximum Overlap specifies how much two matches may overlap in the image; its value ranges between 0 and 1. Especially in the case of an almost symmetric object the allowed overlap should be reduced to prevent multiple matches on the same object.

# 7.3.3.17 The Search Parameter Subpixel

The parameter Subpixel allows to select the accuracy with which the position, orientation, and scale are calculated. If you select the value 'none', the position is determined only with pixel accuracy, and the accuracy of the orientation and scale is equal to the angle step size (page 155) and scale step size (page 155), respectively.

If you select the value 'interpolation', the Matching Assistant examines the matching scores at the neighboring positions, angles, and scales around the best match and determines the maximum by interpolation. Using this method, the position is therefore estimated with sub-pixel accuracy. The accuracy of the estimated orientation and scale depends on the size of the object: The larger the size, the more accurately the orientation and scale can be determined. For example, if the maximum distance between the center and the boundary is 100 pixel, the orientation is determined with an accuracy of about  $0.1^{\circ}$ .

Because the interpolation is very fast, you can select 'interpolation' in most applications.

When you choose the values 'least\_squares', 'least\_squares\_high', or 'least\_squares\_very\_high', a least-squares approximation is used instead of an interpolation, resulting in an even higher accuracy. However, this method requires additional computation time.

# 7.3.3.18 The Search Parameter Last Pyramid Level

With the parameter Last Pyramid Level you can select the lowest pyramid level to which the found matches are tracked. For example, when selecting the value 2, the matching starts at the highest pyramid level and tracks the matches to the second lowest pyramid level (the lowest pyramid level is denoted by a value of 1).

This mechanism can be used to speed up the matching. It should be noted, however, that in general the accuracy of the extracted position, orientation, and scale is lower in this mode than in the normal mode,

in which the matches are tracked to the lowest pyramid level. Hence, if a high accuracy is desired, the parameter Subpixel should be set to at least 'least\_squares'.

Note that if the lowest pyramid level to use is chosen too large, it may happen that the desired accuracy cannot be achieved, or that wrong instances of the model are found because the model is not specific enough on the higher pyramid levels to facilitate a reliable selection of the correct instance of the model. In this case, the lowest pyramid level to use must be set to a smaller value.

#### 7.3.3.19 The Search Parameter Shape models may cross the image border

With the parameter Shape models my cross the image border you can specify whether shape models that cross the image border, i.e., that lie partially outside the test images, should be searched.

If you switch off the check box Shape models may cross the image border the shape model will only be searched within those parts of the test images in which the shape model completely lies within the image.

If you switch on the check box Shape models may cross the image border the shape model will be searched for in all positions in which the model additionally lies partially outside the test images, i.e., in which the shape model extends beyond the image border. Here, points lying outside the image are regarded as being occluded, i.e., they lower the score. This should be taken into account while selecting the Minimum Score (page 162). Please note that the runtime of the search will increase in this mode.

### 7.3.3.20 Optimizing the Recognition Speed

When you select the menu item Use Model > Optimize Recognition Speed or click either the corresponding button in the tool bar or the button Run Optimization in the dialog Optimize Recognition Speed of the tab Model Use, the Matching Assistant automatically determines values for the parameters Minimum Score (page 162) and Greediness (page 163) to optimize the recognition speed. The speed is calculated as the average recognition speed over all test images. You can interrupt this process by clicking the button labelled Stop; please note however, that this event is processed only after the current search has finished.

The two parameters are optimized as follows: At the beginning, the greediness is set to 0 and the minimum score to 1. Then, the minimum score is decreased until the matching succeeds in all test images, i.e., until the recognition rate is 100%. Now, the greediness is increased as long as the matching succeeds. This process is repeated until the optimum parameters are found. You can lower the threshold of acceptance for the recognition rate manually using the corresponding slider or text box at the bottom of the dialog.

The Matching Assistant then displays the optimal minimum score and greediness and the reached recognition time. It automatically enters the parameter values in the dialogs Use Model  $\triangleright$  Standard Model Use Parameters (page 161) and Use Model  $\triangleright$  Advanced Model Use Parameters (page 162), respectively.

If a test image can contain more than one object, the term 'recognition rate' is ambiguous. Therefore, you can choose between three recognition modes:

165

- In each test image, at least one object is expected. The recognition rate is calculated as the percentage of test images which fulfill this condition, i.e., it is 100% if in all test images at least one object is found.
- In each test image, as many objects are expected as specified in the parameter Maximum Number of Matches (page 162) in the dialog accessed via Use Model > Standard Model Use Parameters (page 161). The recognition rate is calculated as the relation of found objects to the sum of expected objects over all images, i.e., it is 100% if in all test images (at least) Maximum Number of Matches objects are found.
- In each test image, as many objects are expected as specified manually (page 160) in the dialog Test Images (page 159) of the tab Model Use. The recognition rate is calculated as the relation of found objects to the sum of expected objects over all images, i.e., it is 100% if in each image exactly as many objects are found as specified.

Note that if you select Maximum Number of Matches = 0 and by mistake specify a lower number of visible objects than actually present in a test image, a recognition rate > 100% results, which completely confuses the optimization algorithm. You may handle this case by selecting the condition > = 100% for the recognition rate.

# 7.3.4 The Menu and Tab Inspect

Via the menu Inspect you can determine the recognition rate and the pose bounds of the object for the used set of test images. Besides the automatical determination of the recognition rate, the tab Inspect is opened. Alternatively, you can directly open the tab and select the button Run. Inside the tab, you can also specify the maximum number (page 162) of object instances the Matching Assistant should search for.

# 7.3.4.1 Determining the Recognition Rate

With the menu item Inspect > Determine Recognition Rate or when you click either the corresponding button in the tool bar or the button Run in the tab Inspect, the Matching Assistant determines the recognition rate by searching the object in all loaded test images. You can interrupt this process by clicking the button labelled Stop; please note however, that this event is processed only after the current search has finished.

The Matching Assistant then displays at Recognition Rate the recognition rate calculated for different criteria and at Statistics the mean, minimum, and maximum *score* (page 144), as well as the mean, minimum, and maximum matching time.

You can choose between three recognition modes:

- In each test image, at least one object is expected. The recognition rate is calculated as the percentage of test images which fulfill this condition.
- In each test image, as many objects are expected as specified in the parameter Maximum Number of Matches (page 162) in the dialog accessed via Use Model > Standard Model Use Parameters (page 161). The recognition rate is calculated as the relation of found objects to the sum of expected objects over all images (in percent).

Please keep in mind that if an image contains more objects than specified in the parameter Maximum Number of Matches, only the best Maximum Number of Matches instances are found! Therefore, if there are, e.g., two test images containing 1 and 3 objects, respectively, and you select Maximum Number of Matches = 2, the recognition rate will be 75%, i.e., 3 out of 4 expected objects.

• In each test image, as many objects are expected as specified manually (page 160) in the dialog accessed via Use Model > Standard Model Use Parameters (page 161). The recognition rate is calculated as the relation of found objects to the sum of expected objects over all images (in percent).

Before using this mode, please check the value specified for the parameter Maximum Number of Matches (page 162): If it is not set to 0, it should not be smaller than the maximum number of objects visible in a test image; otherwise, the recognition rate will be below 100%.

Note that if you select Maximum Number of Matches = 0 and by mistake specify a lower number of visible objects than actually present in a test image, a recognition rate > 100% results. To further extend this line of thought: If for some reason in another test image an object is not found, the two errors cancel each other out, i.e., the recognition rate is 100%! Therefore, we recommend to check whether the correct objects are found via the dialog Test Images (page 159) in the tab Model Use.

### 7.3.4.2 Determining the Pose Bounds

When you click the button Run in the tab Inspect, besides the recognition rate (page 165) the Matching Assistant determines so-called pose bounds, i.e., the range of positions, orientations, and scales in which the object appears in the test images. You can interrupt this process by clicking the button labelled Stop; please note however, that this event is processed only after the current search has finished.

If the test images cover the whole ranges of allowed orientations and scales of the object you can use the calculated ranges to optimize the parameters Angle Extent (page 153), Start Angle (page 153), and the parameters for the scale range (page 154) in the dialog accessed via the menu item Create Model > Standard Model Parameters (page 150); we recommend to use slightly larger values to get accurate results at the boundaries of the ranges.

In a corresponding HALCON program you can use the calculated range of positions as a region of interest and thus further speed up the matching process.

# 7.3.5 The Menu and Tab Code Generation

Via the menu Code Generation you can

- open the dialog Options inside the tab Code Generation, where options for the code generation can be set,
- open the dialog Variable Names inside the tab Code Generation, where the names for the used variables can be specified,
- insert code to the program window of HDevelop according to the current settings of the Matching Assistant,

- release the generated code lines in the program window,
- delete the generated code lines from the program window as long as you did not released them, and
- open the dialog for the code preview inside the tab Code Generation.

# 7.3.5.1 Specifying the Options for the Code Generation

Via the menu item Code Generation > Show Code Generation Options you can open the dialog for determining the options for the code generation inside the tab Code Generation. The dialog consists of the following parts:

- radio buttons for selecting whether the shape model is created at run time from the *model image* (page 143) or if an already existing shape model is to be loaded. For the first case, you can additionally select whether to use the model image and the *ROI* (page 143) that were specified inside the Matching Assistant or whether a new ROI has to be drawn at run time,
- a check box to select whether to display the detected model instances in a loop, and
- the button Insert Code to insert the code generated by the Matching Assistant into the program window of HDevelop.

## 7.3.5.2 Specifying the Variables for the Code Generation

Via the menu item Code Generation  $\triangleright$  Show Variables for Code Generation you can open the dialog for determining the variables used for the code generation inside the tab Code Generation. The dialog consists of several text fields for the individual variables needed for the code lines. The Matching Assistant automatically generates reasonable variable names, but you can change the individual names via the text fields.

### 7.3.5.3 Inserting the Generated Code Lines

Via the menu item Code Generation > Insert Code (also accessible as tool bar button or as button inside the tab Code Generation), you can insert the code that is generated according to the current settings of the Matching Assistant into the program window.

# 7.3.5.4 Releasing the Generated Code Lines

Via the menu item Code Generation > Release Generated Code Lines you can release the generated and inserted code lines. After releasing the code lines, all connections between the Matching Assistant and the program window of HDevelop are lost. That is, changes, e.g., the deletion of code lines, can then only be applied directly in the program window and not from within the Matching Assistant anymore.

## 7.3.5.5 Deleting the Generated Code Lines

Via the menu item Code Generation  $\triangleright$  Delete Generated Code Lines you can delete the code lines that you have previously generated and inserted to the program window of HDevelop from within the Matching Assistant. Note that this works only as long as you have not yet released the code lines.

#### 7.3.5.6 Preview of the Generated Code Lines

Via the menu item Code Generation > Show Code Preview you can open the dialog for the Code Preview in the tab Code Generation. Here, you have the possibility to, e.g., edit or replace individual operators of the code lines proposed by the Matching Assistant.

# 7.3.6 The Menu Help

Via the menu Help you can access this online documentation.

# **Chapter 8**

# **HDevelop Language**

This chapter introduces the syntax and the semantics of the HDevelop language. In other words, it illustrates what you can enter into a parameter slot of an operator or procedure call. In the simplest case this is the name of a variable, but it might also be an arbitrary expression like sqrt(A). Besides, control structures (like loops) and the semantics of parameter passing are described.

Note that the HALCON operators themselves are not described in this chapter. For this purpose refer to the HALCON reference manual. All program examples used in this chapter can also be found in the directory %HALCONROOT%\examples\hdevelop\Manuals\HDevelop.

# 8.1 Basic Types of Parameters

HALCON distinguishes two kinds of data: control data (numbers or strings) and iconic data (images, regions, etc.)

By further distinguishing *input* from *output parameters*, we get four different kinds of parameters. These four kinds always appear in the same order in the HDevelop parameter list. In the reference manual operator signatures are visualized in the following way:

operator ( iconic input : iconic output : control input : control output )

As you see, iconic input objects are always passed first, followed by the iconic output objects. The iconic data is followed by the control data, and again, the input parameters succeed the output parameters.

Any of the four types of parameters may be empty. For example, the signature of read\_image reads

```
read_image ( : Image : FileName : )
```

The operator read\_image has one output parameter for iconic objects read\_image.Image and one input control parameter read\_image.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 or strings) in HDevelop. The name is derived from their respective functions within HALCON operators where they *control* the behaviour (the effect) of image processing (e.g., thresholds for a segmentation operator). Control parameters in HDevelop may contain arithmetic or logical operations. A control data item can be of one of the following types: integer, real, string, and boolean.

integer and real The types integer and real are 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. For example:

4711 -123 0xfeb12 073421 73.815 0.32214 .56 -17.32e-122 32E19

Data items of type integer or real are converted to their machine-internal representations: real becomes the C-type double (8 bytes) and integer becomes the C-type long (4 or 8 bytes).

string A string is a sequence of characters that is enclosed in single quotes ('). The maximum string length is limited to 1024 characters. Special characters, like the line feed, are represented in the C-like notation, as you can see in table 8.1 (see the reference of the C language for comparison). You can enter arbitrary characters using the format  $\nn$  where *nn* is a two-digit hexadecimal number, or using the format  $\nn$  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  $\a$  unless the string continues with another hexadecimal digit (0-F).

For example: The string Sobel's edge-filter has be specified to 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 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

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	Ν	\\
single quote	,	\'
arbitrary character (hexadecimal)		$\xnn$
arbitrary character (octal)		0nnn

Table 8.1: Surrogates for special characters.

:= true the effective value of Val is set to 1. In general, every integer value other than 0 means true. Please note that some HALCON operators take logical values for input (e.g., set\_system). In this case the HALCON operators expect string constants like 'true' or 'false' rather than the boolean values true or false.

In addition to these general types, there are special constants and the type tuple, which are specific to HALCON or HDevelop, respectively.

**constants** There are constants for the return value (result state) of an operator. The constants can be used together with the operator dev\_error\_var and dev\_set\_check. These constants represent the normal return value of an operator, so called *messages*. For errors no constants are available (there are more than 400 error numbers internally, see the Extension Package Programmer's Manual).

In table 8.2 all return messages can be found.

**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 numerical data items with *different* 

Constant	Meaning	Value
H_MSG_TRUE	No error; for tests: (true)	2
M_MSG_FALSE	For tests: false	3
H_MSG_VOID	No result could be computed	4
H_MSG_FAIL	Operator did not succeed	5

Table 8.2: Return values for operators.

Tuple constant

```
Value
```



Figure 8.1: The syntax of tuple constants.

types. The standard representation of a tuple is a listing of its elements included into brackets. This is illustrated in figure 8.1.

[] 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

Names of variables are built up as usual by composing letters, digits and the underscore '\_'. The maximum length of a variable name is limited to 256 characters. The kind 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 169). The kind of the variable is determined during the input of the operator parameters: whenever a new identifier appears, a new variable with the same identifier is created. Control and iconic variables must have different names. The value of a variable (iconic or control) is undefined until the first assignment defines it (the variable has not been instantiated yet). A read access to an undefined variable leads to a runtime error (Variable <x> not instantiated).

HDevelop provides a pre-defined variable named \_ (single underscore). You can use this variable for output control parameters whose value you are not interested in. Please note that it is not allowed to use this variable for HDevelop-specific operators (chapters Control and Develop in the HALCON

reference manual). **It is not recommended** to use the variable \_ in programs that will later be exported to a foreign programming language.

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.

The concept of different kinds of variables allows a first ("coarse") typification of variables (control or iconic data), whereas the actual type of the data (e.g., real, integer, string, etc.) is undefined until the variable gets assigned with a concrete value. Therefore, it is possible that the type of a new data item differs from that of the old.

# 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 177.

# 8.5 Expressions for Input Control Parameters

In HDevelop, the use of expressions like arithmetic operations or string operations is limited to control input parameters; all other kinds of parameters must be assigned by variables.

# 8.5.1 General Features of Tuple Operations

This section intends to give you a short overview over the features of tuples and their operations. A more detailed description of each operator mentioned here is given in the following sections.

Please note that in all following tables variables and constants have been substituted by letters which indicate allowed data types. These letters provide information about possible limitations of the areas of definition. The letters and their meaning are listed in table 8.3. 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, l, 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 one, all elements of the other tuples are combined with that single value for the chosen operation.

Symbol	Types
i	integer
a	arithmetic, that is: integer or real
b	boolean
S	string
v	all types (atomic)
t	all types (tuple)

Table 8.3: Symbols for the operation description.

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 8.4: Examples for arithmetic operations with tuples and strings.

- If both tuples have a length greater than one, both tuples must have the same length (otherwise a runtime error occurs). In this case, the selected operation is applied to all elements with the same index. The length of the resulting tuples is identical to the length of the input tuples.
- If one of the tuples is of length 0 ([]), a runtime error occurs.

In table 8.4 you can find some examples for arithmetic operations with tuples. Pay special attention to the order in which the string concatenations are performed. The basic arithmetic operations in HDevelop are +, -, \*, /. Please note that + is a dimorphic operation: If both operands are numeric, it adds numbers. If at least one of the operands is a string, it concatenates both operands as strings.

# 8.5.2 Assignment

In HDevelop, an assignment is treated like an operator. To use an assignment you have to select the operator assign(Input, Result). This operator has the following semantics: It evaluates Input (right side of assignment) and stores it in Result (left side of assignment). However, in the program text the assignment is represented by the usual syntax of the assignment operator: Result := Input. The following example outlines the difference between an assignment in C syntax and its transformed version in HDevelop:

The assignment in C syntax

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

is defined in HDevelop using the assignment operator as

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

which is displayed in the program window as:

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

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

A second assignment operator is available: insert(Input, Value, Index, Result). It is used to assign tuple elements. If the first input parameter and the first output parameter are identical, the call:

insert (Areas, Area, Radius-1, Areas)

is not presented in the program text as an operator call, but in the more intuitive form as:

```
Areas[Radius-1] := Area.
```

As an example:

```
assign([1,2,3], Area)
assign(9, Areas)
insert(Areas, Area, 1, Areas)
```

sets Areas to [1,9,3].

To construct a tuple with insert, 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 **insert** has to be used. It takes the following arguments as input: first the tuple in which the new value should be inserted; then the new value and after that the index position as the third input parameter. The result (the fourth parameter) is almost identical with the input tuple, except of the new value at the defined index position (see the example above).

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 **insert** 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 in order to avoid a runtime error. In the example Areas is initialized with the empty tuple ([]). Instead of insert the operator assign with tuple concatenation

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

could be used, because the element is appended at the back of the tuple. More examples can be found in the program assign.dev.

[t1,t2] => t	concatenate tuples
t  => i	get number of elements of tuple t
t[i] => v	select element i of tuple t; $0 \le i \le  t $
t[i1:i2] => t	select from element i1 to element i2 of tuple t
<pre>subset(t,i) =&gt; t</pre>	select elements specified in i from t
<pre>remove(t,i) =&gt; t</pre>	remove elements specified in i from t
find(t1,t2) => i	get indices of all occurrences of t2 within t1 (or -1 if no match)
uniq(t) => t	discard all but one of successive identical elements from t

Table 8.5: Basic operations on tuples (control data).

# 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 8.5 for operations on tuples containing control data).

The concatenation accepts one or more variables or constants as input. They are all listed between the brackets, separated by commas. The result again is a tuple. Please note the following: [[t]] = [t] = t.

|t| returns the number of elements of a tuple. The indices of elements range from zero to the number of elements minus one (i.e., |t|-1). Therefore, the selection index has to be within this range. <sup>1</sup>

```
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 [2,2,3,'a', 'a',2,3,'b', 'b']:

```
[3,2,2,3,'a','a',2,3,'b',b',8,9]
      [3, Var, [8, 9]]
               |Var|
                      9
             Var[4]
                      'a'
           Var[4:6]
                      ['a',2,3]
subset(Var, [3, 6, 7])
                      ['a',3,'b']
remove(Var, [3, 6, 7])
                       [2,2,3,'a',2,'b']
    find(Var,[2,3])
                       [1,5]
                       [2,3,'a',2,3,'b']
          uniq(Var)
```

Further examples can be found in the program tuple.dev. The HALCON operators that correspond to the basic tuple operations are listed in table 8.6.

Note that these direct operations cannot be used for iconic tuples, i.e., iconic objects cannot be selected from a tuple using [] and their number cannot be directly determined using ||. For this purpose,

<sup>&</sup>lt;sup>1</sup>Please note that the index of objects (e.g., select\_obj) ranges from 1 to the number of elements.

Operation	HALCON operator
[t1,t2]	tuple_concat
t	tuple_length
t[i]	tuple_select
t[i1:i2]	<pre>tuple_select_range</pre>
<pre>subset(t,i)</pre>	<pre>tuple_select</pre>
remove(t,i)	tuple_remove
find(t1,t2)	tuple_find
uniq(t)	tuple_uniq

Table 8.6: Tuple operations and the corresponding HALCON operators.

control	iconic
[]	gen_empty_obj ()
[t1,t2]	<pre>concat_obj (p1, p2, q)</pre>
t	<pre>count_obj (p, num)</pre>
t[i]	<pre>select_obj (p, q, i+1)</pre>
t[i1:i2]	<pre>copy_obj (p, q, i1+1, i2-i1+1)</pre>

Table 8.7: Equivalent tuple operations for control and iconic data.

however, HALCON operators are offered that carry out the equivalent tasks. In table 8.7 you can see tuple operations that work on control data (and which are applied via assign or insert) 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 170, is the use of constants together with the operator assign (or in case of iconic data one of its equivalents shown in table 8.7):

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

This code is displayed as

```
empty_tuple := []
one_integer := 4711
two_numbers := [4711,0.815]
```

This is useful for constant tuples with a fixed (small) length. More general tuples can be created by successive application of the concatenation or the operator **insert** 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:

```
assign ([],tuple)
for (1,100,1,i)
  assign ([tuple,4711],tuple)
endfor
```

which is displayed as

```
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:

```
assign(gen_tuple_const(100,4711),tuple)
```

which is displayed as

```
tuple := gen_tuple_const(100,4711)
```

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:

assign(gen\_tuple\_const(|tuple\_old|,4711),tuple\_new)

which is displayed as

tuple\_new := gen\_tuple\_const(|tuple\_old|,4711)

The second one is a bit tricky and uses arithmetic functions:

assign((tuple\_old \* 0) + 4711,tuple\_new)

which is displayed as

 $tuple_new := (tuple_old * 0) + 4711$ 

Operation	Meaning	HALCON operator
a1 / a2	division	tuple_div
a1 * a2	multiplication	tuple_mult
a1 % a2	modulus	tuple_mod
a1 + a2	addition	tuple_add
a1 - a2	subtraction	tuple_sub
-a	negation	tuple_neg

Table 8.8: Basic arithmetic operations.

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:

```
assign([],tuple)
for (1,100,1,i)
   assign([tuple,i*i],tuple)
endfor
```

which is displayed as

```
tuple := []
for i := 1 to 100 by 1
   tuple := [tuple,i*i]
endfor
```

In this example we construct a tuple with the square values from  $1^2$  to  $100^2$ .

# 8.5.5 Simple Arithmetic Operations

See table 8.8 for an overview of the available simple arithmetic operations.

All operations are left-associative, except the right-associative unary minus operator. The evaluation usually is done from left to right. However, parentheses can change the order of evaluation and some operators have a higher precedence than others (see section 8.5.14).

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
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 8.9: Bit operations.

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

Simple examples can be found in the program arithmetic.dev.

### 8.5.6 Bit Operations

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

The result of lsh(i1,i2) is a bitwise left shift of i1 that is applied i2 times. If there is no overflow this is equivalent to a multiplication by  $2^{i2}$ . The result of rsh(i1,i2) is a bitwise right shift of i1 that is applied i2 times. For non-negative i1 this is equivalent to a division by  $2^{i2}$ . For negative i1 the result depends on the used hardware. For lsh and rsh the result is undefined if the second operand has a negative value or the value is larger than 32. More examples can be found in the program bit.dev.

### 8.5.7 String Operations

There are several string operations available to modify, select, and combine strings. Furthermore, some operations allow 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><width>.<precision><conversion>

or as a regular expression:

[-+ #]?([0-9]+)?(\.[0-9]\*)?[doxXfeEgGsb]?

(which roughly translates to zero or more of the characters in the first bracket pair followed by zero or more digits, optionally followed by a dot which may be followed by digits followed by a conversion character from the last bracket pair).

Input	Output	
23 \$ '10.2f'	, 23.00,	
23 \$ '-10.2f'	'23.00 '	
4 \$ '.7f'	'4.000000'	
1234.56789 \$ '+10.3f'	' <b>+</b> 1234.568'	
255 \$ 'x'	'ff'	
255 \$ 'X'	'FF'	
Oxff \$ '.5d'	'00255 <i>'</i>	
'total' \$ '10s'	' total'	
'total' \$ '-10s'	'total '	
'total' \$ '10.3'	' tot'	

Some conversion examples might show it best:

*flags* Zero or more flags, in any order, which modify the meaning of the conversion specification. Flags may consist of the following characters:

v\$s	convert v using specification s
v1 + v2	concatenate v1 and v2
<pre>strchr(s1,s2)</pre>	search character s2 in s1
<pre>strstr(s1,s2)</pre>	search substring s2 in s1
<pre>strrchr(s1,s2)</pre>	search character s2 in s1 (reverse)
<pre>strrstr(s1,s2)</pre>	search substring s2 in s1 (reverse)
strlen(s)	length of string
s{i}	select character at position i; 0 <= i <= strlen(s)
s{i1:i2}	select substring from position i1 to position i2
<pre>split(s1,s2)</pre>	split s1 in substrings at s2
regexp_match(s1,s2)	extract substrings of s1 matching the regular expression s2
<pre>regexp_replace(s1,s2,s3)</pre>	replace substrings of s1 matching the regular expression s2 with s3
<pre>regexp_select(s1,s2)</pre>	select tuple elements from s1 matching the regular expression s2
<pre>regexp_test(s1,s2)</pre>	return how many tuple elements in s1 match the regular expression s2

Table 8.10: String operations.

- + 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. This means that if the space flag and + flag both appear, the space flag is ignored.
- # The value is to be converted to an "alternate form". For d and s (see below) conversions, this flag has no effect. For o conversion (see below), it increases the precision to force the first digit of the result to be a zero. For x or X conversion (see below), a non-zero result has 0x or 0X prefixed to it. For e, E, f, g, and G conversions, the result always contains a radix character, even if no digits follow the radix character. For g and G conversions, trailing zeros are not removed from the result, contrary to usual behavior.
- *width* An optional string of decimal digits to specify a minimum field width. For an output field, if the converted value has fewer characters than the field width, it is padded on the left (or right, if the left-adjustment flag has been given) to the field width.
- *precision* The precision specifies the minimum number of digits to appear for integer conversions (the field is padded with leading zeros), the number of digits to appear after the radix character for the e and f conversions, the maximum number of significant digits for the g conversion, or the maximum number of characters to be printed from a string conversion. The precision takes the form of a period. followed by a decimal digit string. A null digit string is treated as a zero.
- conversion A conversion character indicates the type of conversion to be applied:
  - d, o, x, X The integer argument is printed in signed decimal (d), unsigned octal (o), or unsigned hexadecimal notation (x and X). The x conversion uses the numbers and lower-case letters 0123456789abcdef, and the X conversion uses the numbers and upper-case letters 0123456789ABCDEF. The precision component of the argument specifies the minimum number of digits to appear. If the value being converted can be represented in fewer digits than the specified minimum, it is expanded with leading zeroes. The default precision is 1. The result of converting a zero value with a precision of 0 is no characters.
  - f The floating-point number argument is printed in decimal notation in the style [-] ddd.ddd, where the number of digits after the radix character, ., is equal to the precision specification. If the precision is omitted from the argument, six digits are output; if the precision is explicitly 0, no radix appears.
  - e, E The floating-point-number argument is printed in the style [-]d.dde+dd, where there is one digit before the radix character, and the number of digits after it is equal to the precision. When the precision is missing, six digits are produced; if the precision is 0, no radix character appears. The E conversion character produces a number with E introducing the exponent instead of e. The exponent always contains at least two digits. However, if the value to be printed requires an exponent greater than two digits, additional exponent digits are printed as necessary.
  - g, G The floating-point-number argument is printed in style f or e (or in style E in the case of a G conversion character), with the precision specifying the number of significant digits. The style used depends on the value converted; style e is used only if the exponent resulting from the conversion is less than -4 or greater than or equal to the precision. Trailing zeros are removed from the result. A radix character appears only if it is followed by a digit.

s The argument is taken to be a string, and characters from the string are printed until the end of the string or the number of characters indicated by the precision specification of the argument is reached. If the precision is omitted from the argument, it is interpreted as infinite and all characters up to the end of the string are printed.

In no case does a nonexistent or insufficient field width cause truncation of a field; if the result of a conversion is wider than the field width, the field is simply expanded to contain the conversion result.

Examples for the string conversion can be found in the program string.dev.

### + (string concatenation)

The string concatenation (+) can be applied in combination with strings or all numerical types; if necessary, the operands are first transformed into strings (according to their standard representation). At least one of the operands has to be already a string so that the operator can act as a string concatenator. In the following example a file name (e.g., 'Name5.tiff') is generated. For this purpose two string constants ('Name' and '.tiff') and an integer value (the loop-index i) are concatenated:

```
for i := 1 to 5 by 1
    read_image (Image, 'Name'+i+'.tiff')
endfor
```

str(r)chr

See also: tuple\_strchr, tuple\_strrchr

str(r)chr(s1,s2) returns the index of the first (last) occurrence of one of the character in s2 in string s1, or -1 if none of the characters occur in the string. s1 may be a single string or a tuple of strings.

str(r)str

See also: tuple\_strstr, tuple\_strrstr

str(r)str(s1,s2) returns the index of the first (last) occurrence of string s2 in string s1, or -1 if s2 does not occur in the string. s1 may be a single string or a tuple of strings.

strlen

See also: tuple\_strlen

strlen(s) returns the number of characters in s.

### {}

See also: tuple\_str\_bit\_select

s{i} selects a single character (specified by index position) from s. The index ranges from zero to the length of the string minus 1. The result of the operator is a string of length one.

s{i1:i2} returns all characters from the first specified index position (i1) up to the second specified position (i2) in s as a string. The index ranges from zero to the length of the string minus 1.

split

See also: tuple\_split

split(s1,s2) divides the string s1 into single substrings. The string is split at those positions where it contains a character from s2. As an example the result of

split('/usr/image:/usr/proj/image',':')

consists of the two strings

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

#### **Regular Expressions**

HDevelop provides string functions that use Perl compatible regular expressions. Detailed information about them can be found in the Reference Manual at the descriptions of the corresponding operators, which have the same name but start with tuple\_. In particular, at the description of tuple\_regexp\_match you find further information about the used syntax, a list of possible options, and a link to suitable literature about regular expressions.

regexp\_match

See also: tuple\_regexp\_match

regexp\_match(s1,s2) searches for elements of the tuple s1 that match the regular expression s2. It returns a tuple with the same size as the input tuple (exceptions exist when working with capturing groups, see the description of tuple\_regexp\_match in the Reference Manual for details). The resulting tuple contains the matching results for each tuple element of the input tuple. For a successful match the matching substring is returned. Otherwise, an empty string is returned.

regexp\_replace

See also: tuple\_regexp\_replace

regexp\_replace(s1,s2,s3) replaces substrings in s1 that match the regular expression s2 with the string given in s3. By default, only the *first* matching substring of each element in s1 is replaced. To replace all occurrences, the option 'replace\_all' has to be set in s2 (see tuple\_regexp\_replace).

For example:

```
assign(regexp_replace(List, '\\.jpg$', '.png'), List)
```

substitutes file names that look like JPEG images with PNG images.

Operation	Meaning	HALCON operator
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	not equal	tuple_not_equal

Table 8.11: Comparison operations.

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

### 8.5.8 Comparison Operations

In HDevelop, the comparison operations are defined not only on atomic values, but also on tuples with an arbitrary number of elements. They always return values of type boolean. table 8.11 shows all comparison operations.

t1 = t2 and t1 # t2 are defined on all types. Two tuples are equal (true), if they have the same length and all the data items on each index position are equal. If the operands have different types (integer and real), the integer values are first transformed into real numbers. Values of type string cannot be mixed up with numbers, i.e., string values are considered to be not equal to values of other types.

The four comparison operations compute the lexicographic order of tuples. On equal index positions the types must be identical, however, values of type integer, real, and boolean are adapted automatically. The lexicographic order applies to strings, and the boolean false is considered to be smaller than the boolean true (false < true). In the program compare.dev you can find examples for the comparison operations.

### 8.5.9 Boolean Operations

The boolean operations and, xor, or, and not are defined only for tuples of length 1. 11 and 12 is set to true (1) if both operands are true (1), whereas 11 xor 12 returns true (1) if exactly one of both operands is true. 11 or 12 returns true (1) if at least one of the operands is true (1). not 1 returns true (1) if the input is false (0), and false (0), if the input is true (1).

### 8.5.10 Trigonometric Functions

All these functions work on tuples of numbers as arguments. The input can either be of type integer or real. However, the resulting type will be of type real. The functions are applied to all tuple values, and the resulting tuple has the same length as the input tuple. For atan2 the two input tuples have to be of equal length. table 8.14 shows the provided trigonometric functions. For the trigonometric functions the angle is specified in radians.

1st Operand	2nd Operand	Operation	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 8.12: Examples for the comparison of tuples.

Operation	Meaning	HALCON operator
11 and 12	logical 'and'	tuple_and
11 xor 12	logical 'xor'	tuple_xor
11 or 12	logical 'or'	tuple_or
not l	negation	tuple_not

Table 8.13: Boolean operations.

# 8.5.11 Exponential Functions

All these functions work on tuples of numbers as arguments. The input can either be of type integer or real. However, the resulting type will be of type real. The functions are applied to all tuple values and the resulting tuple has the same length as the input tuple. For pow and ldexp the two input tuples have to be of equal length.

See table 8.15 for the provided exponential functions.

# 8.5.12 Numerical Functions

The numerical functions shown in table 8.16 work on different data types.

The functions min and max select the minimum and the maximum values of the tuple values. All of these values either have to be of type string, or integer/real. It is not allowed to mix strings with numerical values. The resulting value will be of type real, if at least one of the elements is of type real. If all elements are of type integer the resulting value will also be of type integer. The same

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

Table 8.14: Trigonometric functions.

Operation	Meaning	HALCON operator
exp(a)	exponential function $e^{a}$	tuple_exp
log(a)	natural logarithm $\ln(a)$ , $a > 0$	tuple_log
log10(a)	decadic logarithm, $\log_{10}(a)$ , $a > 0$	tuple_log10
pow(a1,a2)	$a1^{a2}$	tuple_pow
ldexp(a1,a2)	$a1 \cdot 2^{a2}$	tuple_ldexp

Table 8.15: Exponential functions.

Operation	Meaning	HALCON operator
min(t)	minimum value of the tuple	tuple_min
min2(t1,t2)	element-wise minimum of two tuples	tuple_min2
max(t)	maximum value of the tuple	tuple_max
max2(t1,t2)	element-wise maximum of two tuples	tuple_max2
<pre>sum(t)</pre>	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
<pre>select_rank(a,i)</pre>	element at rank i of a tuple	<pre>tuple_select_rank</pre>
sqrt(a)	square root $\sqrt{a}$	tuple_sqrt
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)	element-wise sign of a tuple	tuple_sgn

Table 8.16: Numerical functions.

applies to the function sum that determines the sum of all values. If the input arguments are strings, string concatenation will be used instead of addition.

The functions mean, deviation, sqrt, deg, rad, fabs, ceil, floor and fmod work with integer and real; the result is always of type real. The function mean calculates the mean value and deviation the standard deviation of numbers. sqrt calculates the square root of a number.

cumul returns the different cumulative sums of the corresponding elements of the input tuple, and median calculates the median of a tuple. For both functions, the resulting value will be of type real, if at least one of the elements is of type real. If all elements are of type integer the resulting value will also be of type integer. select\_rank returns the element at rank i and works for tuples containing int or real values. The index i is of type int.

deg and rad convert numbers from radians to degrees and from degrees to radians, respectively.

real converts an integer to a real. For real as input it returns the input. int converts a real to an integer and truncates it. round converts a real to an integer and rounds the value. For integer it returns the input. The function abs always returns the absolute value that is of the same type as the input value.

The following example (file name: euclid\_distance.dev) 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 Euclidian distance of the two vectors V1 and V2 is computed, by using the formula:

$$d = \sqrt{\sum_{i} (V1_i - V2_i)^2}$$

The difference and the multiplication (square) are successively applied to each element of both vectors. Afterwards sum computes the sum of the squares. Then the square root of the sum is calculated. After that the dot product of V1 and V2 is determined by the formula:

$$\langle V1, V2 \rangle = \sum_{i} \left( V1_i * V2_i \right)$$

### 8.5.13 Miscellaneous Functions

sort sorts the tuple values in ascending order, that means, that the first value of the resulting tuple is the smallest one. But again: strings must not be mixed up with numbers. sort\_index sorts the tuple values in ascending order, but in contrast to sort it returns the index positions (0..) of the sorted values.

The function inverse reverses the order of the tuple values. Both sort and inverse are identical, if the input is empty, if the tuple is of length 1, or if the tuple contains only one value in all positions, e.g.,  $[1,1,\ldots,1]$ .

Operation	Meaning	HALCON operator
sort(t)	sorting in increasing order	tuple_sort
<pre>sort_index(t)</pre>	return index instead of values	<pre>tuple_sort_index</pre>
inverse(t)	reverse the order of the values	tuple_inverse
is_number(v)	test if value is a number	tuple_is_number
number(v)	convert string to a number	tuple_number
environment(s)	value of an environment variable	tuple_environment
ord(a)	ASCII number of a character	tuple_ord
chr(a)	convert an ASCII number to a character	tuple_chr
ords(s)	ASCII number of a tuple of strings	tuple_ords
chrt(i)	convert a tuple of integers into a string	tuple_chrt
rand(a)	create random numbers	tuple_rand

Table 8.17: Miscellaneous functions.

is\_number returns true for variables of the type integer or real and for variables of the type string that represent a number.

The function number converts a string representing a number to an integer or a real depending on the type of the number. Note that strings starting with 0x are interpreted as hexadecimal numbers, and strings starting with 0 (zero) as octal numbers; for example, the string '20' is converted to the integer 20, '020' to 16, and '0x20' to 32. If called with a string that does not represent a number or with a variable of the type integer or real, number returns a copy of the input.

environment returns the value of an environment variable. Input is the name of the environment variable as a string.

ord gives the ASCII number of a character as an integer. chr converts an ASCII number to a character.

ords converts a tuple of strings into a tuple of (ASCII) integers. chrt converts a tuple of integers into a string.

### 8.5.14 Operation Precedence

See table 8.18 for the precedence of the operations for control data. Some operations (like functions, |, t[], etc.) are left out, because they mark their arguments clearly.

# 8.6 Reserved Words

The identifiers listed in table 8.19 on page 193 are reserved words and their usage is strictly limited to their predefined meaning. They cannot be used as variable names.

```
band
bxor bor
and
xor or
# =
<= >= < >
+ -
/ * %
- (unary minus) not
$
```

```
Table 8.18: Operation precedence (increasing from top to bottom).
```

# 8.7 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 inbetween 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
```

ifelse (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.

abs	acos	and	asin
assign	atan	atan2	band
bnot	bor	break	bxor
catch	ceil	chr	chrt
comment	continue	cos	cosh
cumul	deg	deviation	else
elseif	endfor	endif	endtry
endwhile	environment	exit	exp
fabs	false	find	floor
fmod	for	gen_tuple_const	H_MSG_FAIL
H_MSG_FALSE	H_MSG_TRUE	H_MSG_VOID	if
ifelse	insert	int	inverse
is_number	ldexp	log	log10
lsh	max	max2	mean
median	min	min2	not
number	or	ord	ords
pow	rad	rand	real
regexp_match	regexp_replace	regexp_select	regexp_test
remove	repeat	return	round
rsh	select_rank	sgn	sin
sinh	sort	<pre>sort_index</pre>	split
sqrt	stop	strchr	strlen
strrchr	strrstr	strstr	subset
sum	tan	tanh	throw
true	try	uniq	until
while	xor		



```
if (<condition>)
    ...
else
    ...
endif
```

elseif This operator is similar to the else-part of the previous control flow structure. However, it
allows to test for an additional condition. The block between elseif and endif is executed if
<condition1> is false and <condition2> is true. elseif may be followed by an arbitrary number
of additional elseif instructions. The last elseif may be followed by a single else instruction.

```
if (<condition1>)
    ...
elseif (<condition2>)
    ...
endif
```

This is syntactically equivalent and thus a shortcut for the following code block:

```
if (<condition1>)
    ...
else
    if (<condition2>)
        ...
    endif
endif
```

while ... endwhile This is a looping control flow structure. As long as the condition is true, the body of the loop is executed. In order to enter the loop, the condition has to be true in the first place. The loop can be restarted and terminated immediately with the operator continue and break, respectively (see below).

To enter both while and endwhile at once, select the operator while in the operator window and make sure the check box next to the operator is ticked.

```
while (<condition>)
   ...
endwhile
```

repeat ... until This loop is similar to the while loop with the exception that the condition is tested at the end of the loop. Thus, the body of a repeat ... until loop is executed at least once. Also in contrast to the while loop, the loop is repeated if the condition is false, i.e., *until* it is finally true.

To enter both **repeat** and **until** at once, select the operator **until** in the operator window and make sure the check box next to the operator is ticked.

```
repeat
...
until (<condition>)
```

for ... endfor The for loop is controlled by a start and an end value and an increment value, step, that determines the number of loop steps. These values may also be expressions, which are evaluated immediately before the loop is entered. The expressions may be of type integer or of type real. If all input values are of type integer, the loop variable will also be of type integer. In all other cases the loop variable will be of type real.

Please note that the for loop is displayed differently in the program window than entered in the operator window. What you enter in the operator window as for(*start*, *end*, *step*, *index*) is displayed in the program window as:

```
for <index> := <start> to <end> by <step>
    ...
endfor
```

To enter both for and endfor at once, select the operator for in the operator window and make sure the check box next to the operator is ticked.

The start value is assigned to the index variable. The loop is executed as long as the following conditions are true: 1) The step value is positive, and the loop index is smaller than or equal to the end value. 2) The step value is negative, and the loop index is greater than or equal to the end value. After a loop cycle, the loop index is incremented by the step value and the conditions are evaluated again.

Thus, after executing the following lines,

```
for i := 1 to 5 by 1
    j := i
endfor
```

i is set to 6 and j is set to 5, while in

```
for i := 5 to 1 by -1
    j := i
endfor
```

i is set to 0, and j is set to 1.

The loop can be restarted and terminated immediately with the operator continue and break, respectively. (see below).

Please note, that the expressions for start and termination value are evaluated only once when *entering the loop*. A modification of a variable that appears within these expressions has no influence on the termination of the loop. The same applies to the modifications of the loop index. It also has no influence on the termination. The loop value is assigned to the correct value each time the for operator is executed.

If the for loop is left too early (e.g., if you press Stop and set the PC) and the loop is entered again, the expressions will be evaluated, as if the loop were entered for the first time.

In the following example the sine from 0 up to  $6\pi$  is computed and printed into the graphical window (file name: sine.dev):

```
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  $512 \times 512$ . 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 opeator break enables you to exit for, while, and repeat loops. The program is then continued at the next line after the end of the loop.

A typical use of the operator break is to terminate a for loop as soon as a certain condition becomes true, e.g., as in the following example:

```
Number := |Regions|
AllRegionsValid := 1
* check whether all regions have an area <= 30
for i := 1 to Number by 1
   ObjectSelected := Regions[i]
   area_center (ObjectSelected, Area, Row, Column)
   if (Area > 30)
      AllRegionsValid := 0
      break ()
   endif
endfor
```

In the following example, the operator break is used to terminate an (infinite) while loop as soon as one clicks into the graphics window:

```
while (1)
grab_image (Image, FGHandle)
dev_error_var (Error, 1)
dev_set_check ('~give_error')
get_mposition (WindowHandle, R, C, Button)
```

```
dev_error_var (Error, 0)
dev_set_check ('give_error')
if ((Error = H_MSG_TRUE) and (Button # 0))
        break ()
endif
endwhile
```

- stop The operator stop stops the program after the operator is executed. The program can be continued by pressing the Step Over or Run button.
- exit The exit operator *terminates* the HDevelop session.
- return The operator return returns from the current procedure call to the calling procedure. If return is called in the main procedure, the PC jumps to the end of the program, i.e., the program is finished.

# 8.8 Limitations

This section summarizes the restrictions of the HDevelop language:

- Maximum number of objects per parameter: 100000
- Maximum length of strings: 1024 characters
- Maximum length of a variable or procedure name: 256 characters

# **Chapter 9**

# **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.

HDevelop allows to export a developed HDevelop program to the programming languages C++, Visual Basic, Visual Basic .NET, C#, and C by writing the corresponding code to a file. The following sections describe the general steps of program development using this feature for the languages

- C++ (section 9.1),
- C# HALCON/.NET (section 9.2 on page 203),
- C# HALCON/COM (section 9.3 on page 205),
- Visual Basic .NET HALCON/.NET (section 9.4 on page 208),
- Visual Basic .NET HALCON/COM (section 9.5 on page 210),
- Visual Basic 6 HALCON/COM (section 9.6 on page 213),
- C (section 9.7 on page 215),

including some language-specific details of the code generation and optimization aspects.

Because HDevelop does more than just execute a HALCON program, the behavior of an exported program will differ in some points from its HDevelop counterpart. A prominent example is that in HDevelop, all results are automatically displayed, while in the exported programs you have to insert the corresponding display operators explicitly. Section 9.8 on page 217 describes these differences in more detail.

# 9.1 Code Generation for C++

This section describes how to create a HALCON application in C++, starting from a program developed in HDevelop.

# 9.1.1 Basic Steps

### 9.1.1.1 Program Export

The first step is to export the program using the menu File  $\triangleright$  Export.... Here, select the language (C++ - HALCON/C++) and save it to a file. A file will be created that contains the HDevelop program as C++ source code. For every HDevelop procedure except the main procedure, the exported file contains a C++ procedure with the corresponding name. Iconic input and output parameters of a procedure are declared as Hobject and Hobject\*, respectively, while control input and output parameters are declared as HTuple and HTuple\*, respectively. All procedures are declared at the beginning of the file. The program body of the HDevelop main procedure is contained in a procedure action() which is called in the function main(). action() and main() can be excluded from compilation by inserting the instruction #define NO\_EXPORT\_MAIN at the appropriate position in the application. Using the instruction #define NO\_EXPORT\_APP\_MAIN only the main() procedure is excluded from compilation. This can be useful if you want to integrate exported HDevelop code into your application through specific procedure interfaces. In that case, there is typically no need to export the main procedure, which was probably used only for testing the functionality implemented in the corresponding 'real' procedures.

Besides the program code, the file contains all necessary #include instructions. All local variables (iconic as well as control) are declared in the corresponding procedures. Iconic variables belong to the class Hobject and all other variables belong to HTuple.

### 9.1.1.2 Compiling and Linking in Windows Environments

The next step is to compile and link this new program. In the Windows environment, Visual C++ is used for the compiling and linking. Example projects can be found in the directory %HALCONROOT%\examples\cpp.

If you want to use Parallel HALCON, you have to include the libraries parhalcon.lib/.dll and parhalconcpp.lib/.dll instead of halcon.lib/.dll and halconcpp.lib/.dll in your project (see the Programmer's Guide, chapter 7 on page 71, for more details).

### 9.1.1.3 Compiling and Linking in UNIX Environments

To compile and link the new program (called, e.g., test.cpp) under UNIX, you can use the example makefile, which can be found in the directory \$HALCONROOT/examples/cpp , by calling

make PROG=test

Alternatively, you can set the variable PROG in makefile to test and then just type make.

You can link the program to the Parallel HALCON libraries by calling

make PROG=test PAR=1

or just type make PAR=1 if you set the variable PROG as described above.

For more details see the Programmer's Guide, chapter 7 on page 71.

### 9.1.2 Optimization

Optimization might be necessary for variables of class HTuple. This kind of optimization can either be done in HDevelop or in the generated C++ code. In most cases optimization is not necessary if you program according to the following rules.

1. Using the tuple concatenation, it is more efficient to extend a tuple at the "right" side, like:

T := [T,New]

because this can be transformed to

T.Append(New);

in C++ and requires no creation of a new tuple, whereas

T := [New, T]

which is translated into

T = New.Append(T);

would need the creation of a new tuple.

2. Another good way to modify a tuple is the operator insert (see section 8.5.2 on page 175). In this case HDevelop code like

T[i] := New

can directly be translated into the efficient and similar looking code

T[i] = New;

# 9.1.3 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.

# 9.1.4 Limitations and Troubleshooting

Besides the restrictions mentioned in this section and in section 9.8 on page 217, please also check the description of the HDevelop operators in section 6.2.6.2 on page 91.

### 9.1.4.1 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++. In addition, there are different default behaviors concerning the result state of operators.

### Messages

In HALCON/C++ only severe errors cause an exception handling which terminates the program and prints an error message. This might cause problems with minor errors, so called *messages* in HALCON. These messages are handled as return values of the operators and can have the following values, which are also available in HDevelop as constants:

H\_MSG\_TRUE H\_MSG\_FALSE H\_MSG\_FAIL H\_MSG\_VOID

One of these messages is always returned indicating the status of the operator. Normally, the result is H\_MSG\_TRUE. Some operators return H\_MSG\_FAIL like read\_image or read\_region to indicate that they could not open a file or there was no permission to read it. In this case the programmer has to check the return value and apply some adequate action. If the message H\_MSG\_FALSE is ignored, errors like

Halcon Error #4056: Image data management: object-ID is NULL

will happen in successive operators, because the predecessor operator did not calculate an appropriate value.

### **Errors**

In the case of hard errors (i.e., no message as described above) the program stops with an error message. To prevent this behavior the HDevelop operators dev\_error\_var and dev\_set\_check can be used to control the exception handling in the application. This works similarly in HDevelop and C++. One difference is caused by the dynamic evaluation of dev\_error\_var in HDevelop. This means that each time the operator is executed (e.g., in a loop) the use of the error variable might change. In contrast to this, in C++ special code is added to store the return values of operators. This code will therefore be static and cannot change during program execution. To understand how the code generation works let us have a look at a short example. Here at first the HDevelop program:

```
dev_set_check('~give_error')
dev_error_var(error,true)
threshold(image,region,100,255)
dev_error_var(error,false)
if (error # H_MSG_TRUE)
   write_string(WindowId,'error number = ' + error)
   exit()
endif
dev_set_check('give_error')
```

This program will be translated into

```
HTuple error;
set_check("~give_error");
error = threshold(image,&region,100,255);
if (error != 2)
{
    write_string(WindowId,HTuple("error number = ") + HTuple(error));
    exit(1);
}
set_check("give_error");
```

As can be seen, the operator dev\_error\_var is eliminated and replaced by the use of the error variable later on.

The points mentioned above might cause these two problems:

- If the second parameter of dev\_error\_var cannot be derived from the program (because no constant false or true are used but expressions, the value will be interpreted as true, that means: "start to use the variable". To avoid confusion use only the constants false or true as values for the second parameter.
- The usage of a variable starts after the first call of dev\_error\_var(ErrVariable,true). In C++ this means that all successive lines (i.e., lines "below"), until the first dev\_error\_var(ErrVariable,false) will have the assignment to ErrVariable. This might lead to a different behavior compared with HDevelop, if dev\_error\_var is called inside a loop, because here the operators inside the loop before dev\_error\_var might also use ErrVariable after the second execution of the loop body. Therefore: Try not to use dev\_error\_var inside a loop. Use it right at the beginning of the program.

# 9.2 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 83.

### 9.2.1 Basic Steps

### 9.2.1.1 Export

The first step is to export the program using the menu File ▷ Export.... Here, select the language (C# - HALCON/.NET) and save it to file. The result is a new file with the given name and the extension ".cs".

### 9.2.1.2 The C# Template

The exported file is intended to be used together with the predefined C# project that can be found in the directory

%HALCONROOT%\examples\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 11.3.1 on page 109.

# 9.2.2 Program Structure

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(). 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.

### 9.2.2.1 Stop

The HDevelop operator stop is translated into a subroutine in C# that creates a message box. This message box causes the program to halt until the button is pressed.

### 9.2.2.2 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.

# 9.2.3 Limitations and Troubleshooting

Besides the restrictions mentioned in this section and in section 9.8 on page 217, please also check the description of the HDevelop operators in section 6.2.6.2 on page 91.

### 9.2.3.1 Variable Names

The export adds the prefix ho\_ to all local iconic and hv\_ to all local control variables, respectively, in order to avoid collisions with reserved words.

### 9.2.3.2 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 to access the reason for the exception and to continue accordingly. Thus, for HDevelop programs containing error handling ((dev\_)set\_check("~give\_error")) the corresponding code is automatically included. Every operator call, for which it is assumed that the HALCON error mechanism is turned off, is enclosed in a try block followed by a catch block. The latter handles the exception and assigns the corresponding HALCON error number to the error variable activated by dev\_error\_var or to a local error variable, otherwise.

Please note that a call of (dev\_)set\_check("~give\_error") has no influence on the operator call. The exception will *always* be raised. This is also true for messages like H\_MSG\_FAIL, which are not handled as exceptions in C++, for example.

### 9.2.3.3 Memory Management

The .NET Framework's runtime environment CLR (Common Language Runtime) has a mechanism called garbage collector, which is used by the CLR to remove no longer needed .NET objects from memory. As mentioned earlier, in the exported C# code every iconic object is represented by a .NET HObject object. From the garbage collector's point of view, a .NET HObject object is rather small. Thus, it might not be collected from memory although the underlying iconic object (e.g., an image) might in fact occupy a large portion of memory. In order to avoid memory leaks caused by this effect, in the exported code every iconic object is deleted explicitly before it is assigned a new value.

# 9.3 Code Generation for C# (HALCON/COM)

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 COM interface of HALCON. A detailed description of this interface can be found in the Programmer's Guide, part IV on page 113.

Note that this export is only provided for backwards compatibility. We recommend to use the export based on HALCON/.NET (see section 9.2 on page 203).

# 9.3.1 Basic Steps

### 9.3.1.1 Export

The first step is to export the program using the menu File ▷ Export.... Here, select the language (C# - HALCON/COM) and save it to file. The result is a new file with the given name and the extension ".cs".

### 9.3.1.2 The C# Template

The exported file is intended to be used together with the predefined C# project that can be found in the directory

```
%HALCONROOT%\examples\c#\HDevelopTemplateCOM
```

This project contains a form with a display window (HWindowXCtrl) 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.

# 9.3.2 Program Structure

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(). Iconic input and output parameters of a procedure are passed as HUntypedObjectX and out HUntypedObjectX, respectively, while control input and output parameters are passed as object and out object, respectively. The subroutine RunHalcon() contains a call to the subroutine action() and has a parameter Window, which is of type HWindowX. 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 HUntypedObjectX and control variables belong to object.

Depending on the program, additional subroutines and variables are declared.

### 9.3.2.1 Arrays

If a single value is inserted into an object array, a special subroutine is called to ensure that the array is valid. If the array is too small or of the wrong type, it is recreated in the appropriate way.

### 9.3.2.2 Expressions

All parameter expressions inside HDevelop are translated into expressions based on the HALCON tuple operators. Therefore, an expression might look somewhat complex. In many cases these expressions can be changed to simple C# expressions. For example, TupleSub becomes a simple subtraction. To ensure that the exported program has the same effect in C#, this exchange is not applied automatically because the semantics are not always identical.

### 9.3.2.3 Used Classes

There are only six classes/types that are used: object for control parameters and HUntypedObjectX for iconic data. In addition, there is the container class HTupleX, which comprises all operators of HALCON processing tuples, in this case the data type object. Then, there are the classes HWindowXCtrl and its low-level content HWindowX. HWindowXCtrl is used inside the project for the output window and a variable of class HWindowX directs the output to this window. Finally, the class HOperatorSetX 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/COM to extend the functionality.

# 9.3.3 Limitations and Troubleshooting

Besides the restrictions mentioned in this section and in section 9.8 on page 217, please also check the description of the HDevelop operators in section 6.2.6.2 on page 91.

### 9.3.3.1 Variable Names

The export adds the prefix ho\_ to all local iconic and hv\_ to all local control variables, respectively, in order to avoid collisions with reserved words.

### 9.3.3.2 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 to access the reason for the exception and to continue accordingly. Thus, for HDevelop programs containing error handling ((dev\_)set\_check("~give\_error")) the corresponding code is automatically included. Every operator call, for which it is assumed that the HALCON error mechanism is turned off, is enclosed in a try block followed by a catch block. The latter handles the exception and assigns the corresponding HALCON error number to the error variable activated by dev\_error\_var or to a local error variable, otherwise.

Please note that a call of (dev\_)set\_check("~give\_error") has no influence on the operator call. The exception will *always* be raised. This is also true for messages like H\_MSG\_FAIL, which are not handled as exceptions in C++, for example.

When handling exceptions you also have to be aware that the COM interface always resets the output parameters at the beginning of the operator execution. Thus, when the exception occurs, output variables are set to Nothing. Therefore, you cannot use the values of variables used as output parameters of the operator causing the exception.

### 9.3.3.3 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 HUntypedObjectX object, which contains a reference to a COM HUntypedObjectX object. From the garbage collector's point of view, a .NET HUntypedObjectX object is rather small. Thus, it might not be collected from memory although the underlying iconic object (e.g., an image) might in fact occupy a large portion of memory. In order to avoid memory leaks caused by this effect, in the exported code every iconic object is deleted explicitly before it is assigned a new value.

# 9.4 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 83.

# 9.4.1 Basic Steps

### 9.4.1.1 Export

The first step is to export the program using the menu File ▷ Export.... Here, select the language (Visual Basic .NET - HALCON/.NET) and save it to file. The result is a new file with the given name and the extension ".vb".

### 9.4.1.2 The Visual Basic .NET Template

The exported file is intended to be used together with the predefined Visual Basic .NET project that can be found in the directory

```
%HALCONROOT%\examples\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 11.3.1 on page 109.

# 9.4.2 Program Structure

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(). 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.

### 9.4.2.1 Stop

The HDevelop operator stop is translated into a subroutine in Visual Basic .NET that creates a message box. This message box causes the program to halt until the button is pressed.

### 9.4.2.2 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.

### 9.4.2.3 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.

# 9.4.3 Limitations and Troubleshooting

Besides the restrictions mentioned in this section and in section 9.8 on page 217, please also check the description of the HDevelop operators in section 6.2.6.2 on page 91.

### 9.4.3.1 Variable Names

In contrast to C, C++, or HDevelop, Visual Basic .NET has many reserved words. Thus, the export adds the prefix  $ho_t$  to all iconic and  $hv_t$  to all control variables, respectively, in order to avoid collisions with these reserved words.

### 9.4.3.2 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 to access the reason for the exception and to continue accordingly. Thus, for HDevelop programs containing error handling ((dev\_)set\_check("~give\_error")) the corresponding code is automatically included. Every operator call, for which it is assumed that the HALCON error mechanism is turned off, is enclosed in a Try block followed by a Catch block. The latter handles the exception and assigns the corresponding HALCON error number to the error variable activated by dev\_error\_var or to a local error variable, otherwise.

Please note that a call of (dev\_)set\_check("~give\_error") has no influence on the operator call. The exception will *always* be raised. This is also true for messages like H\_MSG\_FAIL, which are not handled as exceptions in C++, for example.

### 9.4.3.3 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 H0bject object. From the garbage collector's point of view, a .NET H0bject object is rather small. Thus, it might not be collected from memory although the underlying iconic object (e.g., an image) might in fact occupy a large portion of memory. In order to avoid memory leaks caused by this effect, in the exported code every iconic object is deleted explicitly before it is assigned a new value.

# 9.5 Code Generation for Visual Basic .NET (HALCON/COM)

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 COM interface of HALCON. A detailed description of this interface can be found in the Programmer's Guide, part IV on page 113.

Note that this export is only provided for backwards compatibility. We recommend to use the export based on HALCON/.NET (see section 9.4 on page 208).

### 9.5.1 Basic Steps

### 9.5.1.1 Export

The first step is to export the program using the menu File ▷ Export.... Here, select the language (Visual Basic .NET - HALCON/COM) and save it to file. The result is a new file with the given name and the extension ".vb".

### 9.5.1.2 The Visual Basic .NET Template

The exported file is intended to be used together with the predefined Visual Basic .NET project that can be found in the directory

%HALCONROOT%\examples\vb.net\HDevelopTemplateCOM

This project contains a form with a display window (HWindowXCtrl) 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.

# 9.5.2 Program Structure

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(). Iconic input and output parameters of a procedure are passed as ByVal HUntypedObjectX and ByRef HUntypedObjectX, respectively, while control input and output parameters are passed as ByVal Object and ByRef Object, respectively. The subroutine RunHalcon() contains a call to the subroutine action() and has a parameter Window, which is of type HWindowX. 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 HUntypedObjectX and control variables belong to Object.

Depending on the program, additional subroutines and variables are declared.

### 9.5.2.1 Arrays

If a single value is inserted into an Object array, a special subroutine is called to ensure that the array is valid. If the array is too small or of the wrong type, it is recreated in the appropriate way.

### 9.5.2.2 Expressions

All parameter expressions inside HDevelop are translated into expressions based on the HALCON tuple operators. Therefore, an expression might look somewhat complex. In many cases these expressions can be changed to simple Visual Basic .NET expressions. For example, TupleSub becomes a simple subtraction. To ensure that the exported program has the same effect in Visual Basic .NET, this exchange is not applied automatically because the semantics are not always identical.

### 9.5.2.3 Stop

The HDevelop operator stop is translated into a subroutine in Visual Basic .NET that creates a message box. This message box causes the program to halt until the button is pressed.

### 9.5.2.4 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.

### 9.5.2.5 Used Classes

There are only six classes/types that are used: Object for control parameters and HUntypedObjectX for iconic data. In addition, there is the container class HTupleX, which comprises all operators of HALCON processing tuples, in this case the data type Object. Then, there are the classes HWindowXCtrl and its low-level content HWindowX. HWindowXCtrl is used inside the project for the output window and a variable of class HWindowX directs the output to this window. Finally, the class HOperatorSetX 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/COM to extend the functionality.

# 9.5.3 Limitations and Troubleshooting

Besides the restrictions mentioned in this section and in section 9.8 on page 217, please also check the description of the HDevelop operators in section 6.2.6.2 on page 91.

### 9.5.3.1 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  $hv_{to}$  to all control variables, respectively, in order to avoid collisions with these reserved words.

### 9.5.3.2 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 to access the reason for the exception and to continue accordingly. Thus, for HDevelop programs containing error handling ((dev\_)set\_check("~give\_error")) the corresponding code is automatically included. Every operator call, for which it is assumed that the HALCON error mechanism is turned off, is enclosed in a Try block followed by a Catch block. The latter handles the exception and assigns the corresponding HALCON error number to the error variable activated by dev\_error\_var or to a local error variable, otherwise.

Please note that a call of (dev\_)set\_check("~give\_error") has no influence on the operator call. The exception will *always* be raised. This is also true for messages like H\_MSG\_FAIL, which are not handled as exceptions in C++, for example.

When handling exceptions you also have to be aware that the COM interface always resets the output parameters at the beginning of the operator execution. Thus, when the exception occurs, output variables are set to Nothing. Therefore, you cannot use the values of variables used as output parameters of the operator causing the exception.

### 9.5.3.3 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 HUntypedObjectX object, which contains a reference to a COM HUntypedObjectX object. From the garbage collector's point of view, a .NET HUntypedObjectX object is rather small. Thus, it might not be collected from memory although the underlying iconic object (e.g., an image) might in fact occupy a large portion of memory. In order to avoid memory leaks caused by this effect, in the exported code every iconic object is deleted explicitly before it is assigned a new value.

# 9.6 Code Generation for Visual Basic 6 (HALCON/COM)

This section describes how to create a HALCON application in Visual Basic 6, starting from a program developed in HDevelop. HALCON can be used together with Visual Basic 6 based on the COM interface of HALCON. A detailed description of this interface can be found in the Programmer's Guide, part IV on page 113.

### 9.6.1 Basic Steps

### 9.6.1.1 Export

The first step is to export the program using the menu File  $\triangleright$  Export... Here, select the language (Visual Basic 6.0 - HALCON/COM) and save it to file. The result is a new file with the given name and the extension ".bas".

### 9.6.1.2 The Visual Basic 6 Template

The exported file is intended to be used together with the predefined Visual Basic 6 project that can be found in the directory

%HALCONROOT%\examples\vb\HDevelopTemplate

This project contains a form with a display window (HWindowXCtrl) and a button labeled Run. The file generated by HDevelop has to be added to this project. This is done by using the menu Project  $\triangleright$  Add Module  $\triangleright$  Existing and selecting the file. 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.

### 9.6.2 Program Structure

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(). Iconic 213

input and output parameters of a procedure are passed as ByVal HUntypedObjectX and ByRef HUntypedObjectX, respectively, while control input and output parameters are passed as ByVal Variant and ByRef Variant, respectively. The subroutine RunHalcon() contains a call to the subroutine action() and has a parameter Window, which is of type HWindowX. 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 HUntypedObjectX and control variables belong to Variant. The subroutine RunHalcon() has a parameter Window, which is of type HWindowX. This is the link to the window in the panel to which all output operations are passed.

Depending on the program, additional subroutines and variables are declared.

### 9.6.2.1 Arrays

If a single value is inserted into a Variant array, a special subroutine is called to ensure that the index is valid. If the array is too small it is resized.

### 9.6.2.2 Expressions

All parameter expressions inside HDevelop are translated into expressions based on the HALCON tuple operators. Therefore, an expression might look somewhat complex. In many cases these expressions can be changed to simple Visual Basic expressions. For example, TupleSub becomes a simple subtraction. To ensure that the exported program has the same effect in Visual Basic, this exchange is not applied automatically because the semantics are not always identical.

### 9.6.2.3 Stop

The HDevelop operator stop is translated into a subroutine in Visual Basic that creates a message box. This message box causes the program to halt until the button is pressed.

### 9.6.2.4 Exit

The HDevelop operator exit is translated into the Visual Basic routine End. Because this routine has no parameter, the parameters of exit are suppressed.

### 9.6.2.5 Used Classes

There are only six classes/types that are used: Variant for control parameters and HUntypedObjectX for iconic data. In addition, there is the container class HTupleX, which comprises all operators of HAL-CON processing tuples, in this case the data type Variant. Then, there are the classes HWindowXCtrl and its low-level content HWindowX. HWindowXCtrl is used inside the project for the output window and a variable of class HWindowX directs the output to this window. Finally, the class HOperatorSetX 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/COM to extend the functionality.

# 9.6.3 Limitations and Troubleshooting

Besides the restrictions mentioned in this section and in section 9.8 on page 217, please also check the description of the HDevelop operators in section 6.2.6.2 on page 91.

### 9.6.3.1 Variable Names

In contrast to C, C++, or HDevelop, Visual Basic has many reserved words. Thus, the export adds the prefix  $ho_t$  to all iconic and  $hv_t$  to all control variables, respectively, in order to avoid collisions with these reserved words.

### 9.6.3.2 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. The standard way to handle this in Visual Basic is by using the On Error Goto command. This allows to access the reason for the exception and to continue accordingly. Thus, for HDevelop programs containing error handling (dev\_error\_var) the corresponding code is automatically included.

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.

When handling exceptions you also have to be aware that the COM interface always resets the output parameters at the beginning of the operator execution. Thus, when the exception occurs, output variables are set to Nothing. Therefore, you cannot use the values of variables used as output parameters of the operator causing the exception.

# 9.7 Code Generation for C

This section describes how to create a HALCON application in C, starting from a program developed in HDevelop.

### 9.7.1 Basic Steps

### 9.7.1.1 Program Export

The first step is to export the program using the menu File  $\triangleright$  Export.... Here, select the language (C - HALCON/C) and save it to file. A file will be created that contains the HDevelop program as C

source code. For every HDevelop procedure except the main procedure, the exported file contains a C procedure with the corresponding name. Iconic input and output parameters of a procedure are declared as Hobject and Hobject\*, respectively, while control input and output parameters are declared as Htuple and Htuple\*, respectively. All procedures are declared at the beginning of the file. The program body of the HDevelop main procedure is contained in a procedure action() which is called in function main(). action() and main() can be excluded from compilation by inserting the instruction #define NO\_EXPORT\_MAIN at the appropriate position in the application. Using the instruction #define NO\_EXPORT\_APP\_MAIN only the main() procedure is excluded from compilation. This can be useful if you want to integrate exported HDevelop code into your application through specific procedure interfaces. In that case, there is typically no need to export the main procedure, which was probably used only for testing the functionality implemented in the corresponding 'real' procedures.

Besides the program code, the file contains all necessary #include instructions. All local variables (iconic as well as control) are declared in the corresponding procedures. Iconic variables belong to the class Hobject and all other variables belong to Htuple.

Please note that in the current version the generated C code is not optimized for readability. It is output such that it always produces identical results as the HDevelop code.

### 9.7.1.2 Compiling and Linking in Windows Environments

The next step is to compile and link this new program. In the Windows environment, Visual C++ is used for the compiling and linking. Example projects can be found in the directory %HALCONROOT%\examples\c.

If you want to use Parallel HALCON, you have to include the libraries parhalcon.lib/.dll and parhalconc.lib/.dll instead of halcon.lib/.dll and halconc.lib/.dll in your project (see the Programmer's Guide, chapter 18 on page 151, for more details).

### 9.7.1.3 Compiling and Linking in UNIX Environments

To compile and link the new program (called, e.g., test.c) under UNIX, you can use the example makefile, which can be found in the directory <code>\$HALCONROOT/examples/c</code>, by calling

make TEST\_PROG=test

Alternatively, you can set the variable TEST\_PROG in makefile to test and then just type make.

You can link the program to the Parallel HALCON libraries by calling

make TEST\_PROG=test PAR=1

or just type make PAR=1 if you set the variable TEST\_PROG as described above.

For more details see the Programmer's Guide, chapter 18 on page 151.
Prefix	Where the text following the prefix goes in the exported program
#	The place of insertion
#^^	Beginning of the program
#\$\$	End of the program
#^	Before the current procedure
#\$	After the current procedure

Table 9.1: Special comments in HDevelop.

## 9.8 General Aspects of Code Generation

In the following, general differences in the behavior of a HDevelop program and its exported versions are described.

## 9.8.1 User-Defined Code Blocks

HDevelop comments containing the **#** symbol as the first character are exported as plain text statements. 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 6. This feature can be used to integrate Visual Basic, Visual Basic .NET, C#, C++, or C code into a HDevelop program. Furthermore, some additional special comments are recognized to specify the destination of the plain text statements. For example, the line

\* #^^#define NO\_EXPORT\_APP\_MAIN

puts the given text at the very beginning of the exported program. Comments in this format are collected from the main procedure first, followed by #^^ comments in other procedures.

The recognized special comments are summarized in table 9.1.

### 9.8.2 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, HALCON/COM: HUntypedObjectX) and for the class HTuple (HALCON/C++, HALCON/.NET), the type Variant (Visual Basic 6), and the class object (HALCON/COM for .NET languages), as they all have a destructor that removes the stored data. 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 9.4.3.3 (Visual Basic .NET) and section 9.2.3.3 (C#). Memory issues regarding iconic objects in HALCON/COM are described in section 9.5.3.3 (Visual Basic .NET) and section 9.3.3.3 (C#).

However, problems arise if a tuple (variant) contains a *handle*, for example for a file, a window, or for OCR. In this case, the memory of the handle is automatically removed *but not the data to which it points*. In the exported programs, this data therefore has to be removed explicitly by calling the corresponding operators close\_\* like close\_ocr or close\_ocv. Please insert the close\_\* operators for all handles in use

- before a new value is assigned to a handle and
- at the end of the program.

## 9.8.3 for Loops

HDevelop and the programming languages have different semantics for loops, which may cause confusion. Because the problems are so rare and the generated code would become very difficult to understand otherwise, the code generation ignores the different semantics. These differences are:

- 1. In the programming languages, you can modify the loop variable (e.g., by setting it to the end value of the condition) to terminate the loop. This can't be done in HDevelop because here the current value is stored "inside" the for-operator and is automatically updated when it is executed again.
- 2. In the programming languages, you can modify the step range if you use a variable for the increment. This is also not possible with HDevelop because the increment is stored "inside" the for-operator when the loop is entered.
- 3. The last difference concerns the value of the loop variable after exiting the loop. In the programming languages, it has the value with which the condition becomes false for the first time. In HDevelop it contains the end value, which was calculated when the loop was entered.

Looking at the mentioned points, we recommend to program according to the following rules:

- 1. Don't modify the loop variable or the step value inside the loop. If you need this behavior, use the while-loop.
- 2. Don't use the loop variable after the loop.

### 9.8.4 Protected External 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

external procedures described in section 6.4.3.3 on page 111. 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.

### 9.8.5 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.

### 9.8.6 Graphics Windows

The graphics windows of HDevelop and the basic windows of the HALCON libraries

- HALCON/C++: class HWindow,
- HALCON/.NET: class HWindowControl,
- HALCON/COM: class HWindowXCtrl, 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 for C++ and C programs. In the export of Visual Basic, Visual Basic .NET, and C# programs, 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 Visual Basic, Visual Basic .NET, or C#, you have to modify the code and project manually.

Note that the export of programs containing multiple windows to C++ or C might be incorrect if the button Activate was used during program execution. Note also that HDevelop window operations that do *not* have a window handle parameter like dev\_open\_window, dev\_close\_window, or dev\_set\_line\_style are suppressed in all HDevelop procedures except the main procedure. In order to perform windows operations in HDevelop procedures aimed to be exported to C++ or C, the corresponding HALCON operators like open\_window, close\_window, or set\_line\_style should be used instead (in that case ignore the warning issued by HDevelop).

#### • Window size

In exported Visual Basic, Visual Basic .NET, and C# programs, the size of the window on the form is predefined ( $512 \times 512$ ); 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 to insert 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 for C++ or C, 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\_pointer1. 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 to insert the corresponding Develop operators (e.g., dev\_set\_color or dev\_set\_line\_width) in the HDevelop program explicitly. This will result in the appropriate call (set\_color, set\_line\_width, etc.) in the exported code.

## Chapter 10

# **Tips & Tricks**

This chapter contains helpful information for working with HDevelop.

## 10.1 Keycodes

In order to speed up the entering of values in the input fields of HDevelop (e.g., operator parameters), several keycodes are defined, which have special functions. These keyboard mappings are shown in table 10.1.

## 10.2 Online Help

Online documentation is available in PDF and partly in HTML format.

HDevelop provides an integrated online help window. You can conveniently browse the HTML-based documentation in this window view the HTML files in your web browser. In HDevelop you may call the online help window via the menu Help  $\triangleright$  Help or by pressing <F1>. The functionality is described in section 6.8 on page 127.

Besides HTML, the documentation is available in PDF format as well. To display the manuals, the Adobe file viewer Acrobat Reader is included in the distribution for Windows systems. If you click on a PDF document in the online help window, the registered application for viewing PDF files starts up automatically.

## 10.3 Warning and Error Windows

Warning and error windows are popups that make the user aware of user errors. Usually, they interrupt the faulty actions with a description of the error. For this purpose information about the kind of the error is determined during the execution. Figure 10.1 shows an example of an error window.

Text editing			
<home></home>	Move cursor to the beginning of the line.		
<end></end>	Move cursor to the end of the line.		
<left></left>	Move cursor left one character.		
<right></right>	Move cursor right one character.		
<ctrl> <left></left></ctrl>	Move cursor left one word.		
<ctrl> <right></right></ctrl>	Move cursor right one word.		
<delete></delete>	Delete single character to the right of the cursor position.		
<backspace></backspace>	Delete single character to the left of the cursor position.		
<ctrl> <backspace></backspace></ctrl>	Delete word to the left of the cursor.		
<ctrl> <delete></delete></ctrl>	Delete word to the right of the cursor.		
<shift> <left></left></shift>	Select character to the left of the cursor (or extend the selection		
	by one character).		
<shift> <right></right></shift>	Select character to the right of the cursor (or extend the selection		
	by one character).		
<ctrl> <shift> <left></left></shift></ctrl>	Select word to the left of the cursor (or extend the selection by		
	one word).		
<ctrl> <shift> <right></right></shift></ctrl>	Select word to the right of the cursor (or extend the selection by		
	one word).		
<ctrl> a</ctrl>	Select all.		
<ctrl> d</ctrl>	UNIX: Analogous to Delete		
<ctrl> e</ctrl>	UNIX: Move cursor to last character in line.		
<ctrl> k</ctrl>	UNIX: Delete all characters from current position to end of line.		
<ctrl> u</ctrl>	UNIX: Delete entire input line.		
GUI Navigation			
<ctrl> <tab></tab></ctrl>	Focus next window and bring it to the front.		
<shift> <tab></tab></shift>	Focus the previous window and bring it to the front.		
<tab></tab>	Select the following GUI element.		
<shift> <tab></tab></shift>	Select the previous GUI element.		
<space></space>	Activate the focused button (highlighted with a dashed border)		
<up></up>	Scroll up one line.		
<down></down>	Scroll down one line.		
<pg up=""></pg>	Scroll up one page.		
<pg down=""></pg>	Scroll down one page.		
<home></home>	Scroll to the beginning.		
<end></end>	Scroll to the end.		

Table 10.1: Keycodes for special editing functions.



Figure 10.1: Example for an error window.

## **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.
- **Button** A button is part of a graphical user interface. With the mouse the user can press a button to cause an action to be performed.
- **Control data** Control data can be either numbers (*finteger and freal*), character strings (*fstring*) and truth values (boolean). This data can be used as atomic values (i.e., single values) or as *fuples* (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  $\uparrow$ HDevelop for displaying, e.g.,  $\uparrow$ images,  $\uparrow$ regions, and  $\uparrow$ 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. The maximum length of a character string is limited to 1024 characters.
- **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 *\cap\control* and *\cap\control* data.
- **XLD** is the short term for eXtended Line Description. It is used as a superclass for contours, polygons, and lines.

## **Appendix B**

# **Command Line Switches**

HDevelop accepts the following command line switches:

```
hdevelop [options]
 HDevelop options:
  <program>.dev
                      - open the program for editing
  -run <program>.dev - load the program and start execution
  <image_file>
                      - load an image file with read_image
                      - show this help info in a message box
  -help
  -version
                      - show version information in a message box
  --help
                      - show this help information on the console
  --version
                      - show version information on the console
  -convert <program>.dev <program>.{cpp,c,cs,vb,bas,cs,vb,txt,dev}
                  [-COM]
                  [-external_proc_path:<external procedure path>]
                  [-external_procs_only_interfaces]
                      - convert an HDevelop program into a source
                        file of the specified type
  -reset_preferences - reset all persistent settings from
                        previous sessions
  -preferences <file> - start HDevelop with the preferences
                        from a file
 Qt 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.
  -{fn|font} <font>
                      - defines the application font. The font
                        should be specified using an X logical
                        font description.
```

-{bg background} <col< th=""><th>or&gt;</th></col<>	or>
-	sets the default background color and an
	application palette (light and dark shades
	are calculated).
-{fg foreground} <col< td=""><td>or&gt;</td></col<>	or>
-	sets the default foreground color.
-{btn button} <color></color>	
-	sets the default button color.
-name <name> -</name>	sets the application name.
-title <title> -</title>	sets the application title.
-visual TrueColor -	forces the application to use a TrueColor
	visual on an 8-bit display.
-ncols <count> -</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 2x3x1 cube is used.
-cmap -	causes the application to install a
-	private color map on an 8-bit display.

# Index

\* (asterisk) external procedure, 49 in window title, 33, 43 .NET, 203, 208 .avi, 26 .seq, 26 #, 217 #\$. 217 #\$\$, 217 #^. 217 #^^, 217 \$.181 IC. 15 PC, 15 Source image, 23 \_COPY\_1, 48, 49 Live, 26 Detect, 27 Reset All, 27 Snap, 26 File, 26 Abort Procedure Execution, 78 About, 100 Activate, 58 Add to User Tab, 119 Add Variable, 119 add\_channels, 134, 138 advanced model parameters, 154 advanced search parameters, 162 A11. 119 Alternatives, 95 Always Find, 160 Angle Extent, 153 Angle Step, 155 Apply Immediately, 81 area\_center, 40

assign, 90, 175, 176, 178 assistant Close Dialog, 96 Delete Generated Code Lines, 96 Exit Assistant, 96 image acquisition, 23, 95 Insert Code, 96 Load Assistant Settings, 96 matching, 95 Release Generated Code Lines. 96 Save Current Assistant Settings, 96 Show Code Preview. 96 assistant settings, load, 148 assistant settings, save, 148 assure success, 160 Auto, 119 AVI, 26 boolean, 225 boolean operations, 187 break, 90, 194-196 break point, 15, 74 clear, 77 clear all. 77 set. 76 button, 225 C, 9, 215 compile and link (UNIX), 216 compile and link (Windows), 216 export of HDevelop programs, 9, 215 C++, 9 compile and link (UNIX), 200 compile and link (Windows), 200 export of HDevelop programs, 199 C#. 9 export of HDevelop programs, 203, 205

Call Stack..., 76 Cascade Windows, 98 categories example programs, 47 channel gray value, 86, 131 channel number, 132 channel selection gray histogram, 134 check box Always Find, 160 check box Pregenerate Shape Model, 157 check box Shape models may cross the image border, 164 Cleanup, 54, 119 Clear All Breakpoints, 77 Clear Breakpoint, 77 Clear Graphics Window, 79 Clear Variable, 119 click. 11 clipboard, 57, 58 Close Assistant, 148 Close Dialog, 96 Close Graphics Window, 79 close window. 219 Code Generation. 166 Code generation, 199 code generation file, 24 image acquisition interface, 28 code generation, preview, 168 code lines, delete, 168 code lines, insert, 167 code lines, release, 167 code options, 167 code variables, 167 Color. 80 color graphics window, 80 Colored, 80 column. 44 Column Scale Step, 155 COM, 205, 210, 213 comment, 58, 59, 90, 91 comment, 59, 91 #. 217 #\$, 217 #\$\$, 217

#^, 217 #^^, 217 comparison operations, 186 concat\_obj, 121, 178 connection, 35, 80, 136 continue, 90, 194-196 Contrast. 151 Control. 90 control data, 225 control flow break, 196 continue, 196 elseif. 193 exit, 197 for ... endfor, 194 if ... else ... endif, 192 if ... endif, 192 operators, 192 repeat ... until, 194 return. 197 stop, 197 while ... endwhile, 194 coordinates status bar, 44 Copy, 58 variable values, 123 Copy History to Clipboard, 44 copy\_obj, 178 count\_obj, 80, 178 Create Model, 148 create model, 144 Create New Procedure, 87 Create ROI, 149 Cut, 57 Deactivate, 59 Delete, 58 Delete All Test Images, 159 Delete All Unused Local, 88 Delete Current, 88 Delete Generated Code Lines, 96, 168 Delete Test Image, 158 Delphi (Borland), 9 Detect All. 159 determine pose bounds, 166 Determine Recognition Rate, 165

dev\_ operators, 91 dev\_clear\_obj, 92 dev clear window, 79, 91 dev\_close\_inspect\_ctrl, 92 dev\_close\_window, 79, 91, 219, 220 dev\_display, 79, 92, 126, 220 dev\_display, 220 dev error var. 93, 171 dev\_error\_var, 202 dev\_inspect\_ctrl, 92, 122 dev\_map\_par, 93 dev\_map\_prog, 93, 97 dev\_map\_var, 93, 97 dev\_open\_window, 78, 91, 92, 94, 97, 219, 220dev\_open\_window, 220 dev\_set\_check, 72, 93, 171 dev\_set\_check, 202 dev\_set\_color, 80, 91-94, 220 dev\_set\_colored, 80, 92 dev\_set\_draw, 80, 92, 93 dev\_set\_line\_style, 219 dev\_set\_line\_width, 80, 92, 220 dev\_set\_lut, 81, 92 dev\_set\_paint, 81, 92 dev\_set\_part, 84, 92, 125, 220 dev\_set\_part, 220 dev\_set\_preferences, 73 dev\_set\_shape, 81, 92 dev\_set\_window, 78, 92 dev\_set\_window\_extents, 79, 91 dev\_unmap\_par, 93 dev\_unmap\_prog, 93 dev\_unmap\_var, 93 dev\_update\_pc, 72, 93 dev\_update\_time, 72, 93 dev\_update\_var, 72, 93 dev\_update\_window, 73, 92, 93 Develop, 91 disp\_circle, 126 disp\_image, 126 disp\_line, 126 disp\_region, 126 Display, 79 Display Image Pyramid, 149 Display Model, 149 Display Selected Test Image, 160

drag-and-drop, 11, 21 Draw, 80 dump\_window, 78 Duplicate...,87 dyn\_threshold, 95 Edit program line, 103 Edit Interface, 88 Edit menu Activate. 58 Copy, **58** Cut. 57 Deactivate. 59 Delete. 58 Find Again, 61 Find/Replace..., 59 Paste, 58 Preferences..., 62 Redo, 57Undo, 57Edit Procedure, 89 else, 192, 193 else, 192 elseif, 90, 193 Enable the Context Menu in the Graphics Window, 73 encoding, 53 native. 53, 70 UTF-8, 53, 70 endfor, 41, 75, 194, 195 endfor. 194 endif, 75, 192, 193, 196 endif, 192 endwhile, 75, 194 endwhile, 194 Error message, 221 escape strings, 170 example programs, 47 Exception handling, 202, 205, 207, 210, 212, 215 Execute menu Abort Procedure Execution, 78 Call Stack..., 76 Clear All Breakpoints, 77 Clear Breakpoint, 77

Reset Procedure Execution, 77 Reset Program Execution, 77, 173 Run. 73 Run to Insert Cursor, 74 Set Breakpoint, 76 Step Forward, 75 Step Into, 75 Step Out, 76 Step Over, 74 Stop, 76 exit, 56, 90, 91, 197 Exit Assistant, 96, 148 exponential functions, 188 Export, 63 Export..., 51 UTF-8 encoding, 53 External Procedure Settings..., 89 external procedure, modified, 49 External Procedures, 66 false. 225 Feature Histogram, 86 Feature Inspection, 86 File, 146 menu. 46 file history, 46 File menu Cleanup, 54 Export..., 51 Insert All..., 48 Insert Mainbody..., 49 Insert Procedures..., 49 Insert Program, 48 New Program, 46 Open Example..., 47 Open Program..., 46 Print..., 55 Properties..., 54 Quit, 56 Read Image, 53 Recent Programs, 48 Save , 49Save All, 51 Save Procedure As..., 50 Save Program As..., 50 file\_exists, 93

Find Again, 61 Find Model, 159 Find/Replace..., 59 for, 41, 42, 75, 90, 102, 103, 194-196 for loop, 194 for, 218, 225 gauss\_image, 95 gen\_empty\_obj, 121, 178 gen\_lowpass, 95 gen\_tuple\_const, 179 General Options, 69 get\_image\_pointer1, 220 get\_mposition, 93 get\_system, 219 Give Error, 72 graphics window, 124 graphics window, 15, 219, 225 clear. 79 close, 79 colors, 80 image size, 79 line width, 80 open, 78 regions, 80, 81 select iconic variable, 79 window size, 79 Gray Histogram, 86 gray value histogram, 86, 133 inspection, 86, 131 status bar, 44 Greediness, 163 GUI. 3 H\_MSG\_FAIL, 202 H\_MSG\_FALSE, 202 H\_MSG\_TRUE, 202 H\_MSG\_VOID, 202 HALCON example programs, 47 modules, 54 HALCON News (WWW), 100 HALCON Reference, 99 HALCONIMAGES, 53 HALCONROOT, 53

HDevelop procedures, 10 export, 219 external, 11 hierarchy, 11 local, 11 main, 10, 200, 204, 206, 208, 211, 213, 216.219 program export to C, 9, 215 export to C++, 199 export to C#, 203, 205 export to Visual Basic, 213 export to Visual Basic .NET, 208, 210 HDevelop dev\_operators, 91 example programs, 47 language, 169 runtime error, 74 warning, 141 HDevelop Language, 100 HDevelop Reference, 99 Help, 99, 105 About. 100 HALCON News (WWW), 100 HALCON Reference, 99 HDevelop Language, 100 HDevelop Reference, 99 Help, 99 Search Documentation, 100 history of files, 46

### IC, 102

iconic data, 225 iconic object, 225 if, 75, 90, 102, 192, 196 if, 192 if, 225 ifelse, 90, 102, 192 ifelse, 192 image, 225 image acquisition assistant, 23 Image Acquisition Assistant, 95 image coordinates status bar, 44 image properties status bar, 44 image pyramid, display, 149 Image Size, 79 Import, 62 insert, 90, 175, 176, 178, 179 insert. 201 Insert All As Local. 88 Insert All....48 Insert Code, 96, 167 insert cursor, 15 Insert Mainbody..., 49 Insert Procedures..., 49 Insert Program, 48 Insert Used As Local, 88 Inspect, 165 junctions\_skeleton, 94 Keep dialog open, 48, 53 keyboard, 12 keyboard menu access, 46 Keycodes, 221 Keywords..., 95 Last Pyramid Level, 163 Line Width, 80 Load Assistant Settings, 96, 148 Load in new HDevelop, 48 Load Model, 147 Load Test Images, 158 local procedure, 49 look-up table, 81 loop body, 225 Lut, 81 main window, 43 window title, 43 Make All External, 89 Manage Passwords, 68 Matching Assistant, 95 Max Column Scale, 154 Max Row Scale, 154

Maximum Number of Matches, 162

Maximum Overlap, 163

MDI, 98

**MDI. 45** mean\_image, 95 median\_image, 95 menu Assistants. 95 Edit. 56 Execute, 73 File, 46 Help, 99 Operators, 89 Procedures, 87 Suggestions, 94 Visualization, 78 Window, 97 menu bar, 46 messages status bar, 44 Metric, 156 Min Column Scale, 154 Min Row Scale, 154 Minimum Component Size, 152 Minimum Contrast, 157 Minimum Score, 162 Miscellaneous, 221 model creation, 144, 148 model image, display, 149 model image, open, 147 model parameters, advanced, 154 model parameters, standard, 150 model use parameters, advanced, 162 model use parameters, standard, 161 modified external procedure, 49 program, 43 mouse click, 11 multiple-document interface, 45 native encoding, 53, 70 New Program, 46 New Zoom Window, 86 number of visible objects, 160 Open, 48

Open Example..., Open Example..., Open Graphics Window, Open Graphics Window...,

Open Model Image, 147 Open Operator Window, 97 Open Program Listing, 97 Open Program..., 46 open test images, 158 Open Variable Window, 97 open\_framegrabber, 26 open\_window, 93, 94, 219 operating systems UNIX, 200, 216 Windows, 200, 216 operation precedence, 191 operator data base, 225 window, 113 operator window, 15 operator window, 226 Operators Control, 90 Develop, 91 submenus, 94 Optimization, 201 Optimization, 156 optimize parameters, 145 Optimize Recognition Speed, 164 Organize Windows, 97 orientation\_region, 40 Paint. 81 Parallel HALCON, 200, 216 parameter

expressions, 173 parameter Angle Extent, 153 parameter Angle Step, 155 parameter Column Scale Step, 155 parameter Contrast, 151 parameter Greediness, 163 parameter Last Pyramid Level, 163 parameter Max Column Scale, 154 parameter Max Row Scale, 154 parameter Maximum Number of Matches, 162 parameter Maximum Overlap, 163 parameter Metric, 156 parameter Min Column Scale, 154 parameter Min Row Scale, 154

parameter Minimum Component Size, 152 parameter Minimum Contrast, 157 parameter Minimum Score, 162 parameter Optimization, 156 parameter optimization, 145 parameter Pyramid Levels, 152 parameter Row Scale Step, 155 parameter Start Angle, 153 parameter Subpixel, 163 parameters, reset, 157 Paste, 58 PC. 102 pixel type, 132 pixel info, 86, 131 pose bounds, determine, 166 Predecessors, 94 preferences export, 63 import, 62 Preferences....62 Pregenerate Shape Model, 157 print HDevelop procedure, 55 HDevelop program, 55 procedure, 55 program, 55 Print..., 55 procedures (HDevelop), 10 export, 219 C. 216 C++, 200C#, 204, 206 Visual Basic, 213 Visual Basic .NET, 208, 211 external, 11 hierarchy, 11 local. 11 main, 10, 200, 204, 206, 208, 211, 213, 216.219 Procedures menu Create New Procedure, 87 Delete All Unused Local. 88 Delete Current, 88 Duplicate...,87 Edit Interface, 88 Edit Procedure, 89

External Procedure Settings..., 89 Insert All As Local. 88 Insert Used As Local. 88 Make All External. 89 program counter, 15 program window, 15, 102, 226 edit line. 103 Properties..., 54 pull-down menu, 46 Pyramid Levels, 152 pyramid levels, lock model and model image, 150pyramid levels, model, 150 pyramid levels, model image, 150 pyramid, display, 149

#### Quit, 56

Read Image, 53 read\_image, 21, 32-34, 53, 54, 93, 169 read\_image.FileName, 169 read\_image.Image, 169 Recent Programs, 46, 48 recognition rate, determine, 165 recognition speed, optimize, 164 Redo. 57reduce domain. 134, 138 reference to assistant elements, 146 regexp\_match, 185 regexp\_replace, 185 regexp\_select, 186 regexp\_test, 186 region, 226 colors, 80 empty, 225 line width, 80 shape, 81 regiongrowing, 95 regions visualization, 80 regular expressions, 185 Release Generated Code Lines, 96, 167 Remove from User Tab, 119 repeat, 90, 194, 196 repeat, 194replace Find/Replace..., 59

reserved words, 191 Reset, 63, 157 reset graphics window, 82 Reset Parameters. 82 Reset Procedure Execution, 77 Reset Program Execution, 77 reset\_obj\_db, 94 Restrictions, 201, 205, 207, 209, 212, 215 return, 90, 109, 197 ROI creation, 149 row, 44 Row Scale Step, 155 Run, 73 Run to Insert Cursor, 74 Run Until Here, 105 runtime status bar, 44 Runtime error, 202 Runtime Settings, 72 save local procedure, 49 Save, 49 Save All, 51 Save Current Assistant Settings, 96. 148 Save Model, 148 Save Procedure As..., 50 Save Program As..., 50 Save Window ..., 87 scale range, 154 scale step size, 155 scale\_image, 136 SDI, 98 SDI, 45 Search Documentation, 100 search object, 159 search parameters, advanced, 162 search parameters, standard, 161 See also, 95 Select Graphics Window, 82 select test image, 160 select\_gray, 138 select\_obj, 177, 178 select\_shape, 115, 121, 137, 138 select\_shape\_xld, 137, 138

semantics. 169 sequence file, 26 Set Breakpoint, 76 Set Parameters...,82 set check. 93 set\_color, 91, 220 set\_line\_style, 219 set\_line\_width, 220 set\_paint, 92 set\_part, 220 set\_system, 171 set\_tposition, 42 set\_system, 219 Shape, 81 shape model, load, 147 shape model, save, 148 Shape models may cross the image border, 164 shortcuts. 12 Show Caller, 105 Show Code Preview, 96, 168 Show Procedure, 105 Show Processing Time, 44, 72 sigma\_image, 95 single-document interface, 45 skeleton, 95 smooth\_image, 95 Sort by Name, 119 Sort by Occurrence, 119 split, 185 standard model parameters, 150 standard search parameters, 161 Start Angle, 153 status bar, 44 Step Forward, 75 Step Into, 75 Step Out, 76 Step Over, 74 Stop, 76 stop, 19, 73, 74, 90, 91, 197 stop HDevelop program, 74 program, 74 strchr, 184 string, 226 concatenation, 174, 184 operations, 181

```
special characters, 170
strlen. 184
strrchr. 184
strrstr. 184
strstr. 184
Subpixel, 163
Successors, 95
Suggestions
    Alternatives, 95
    Keywords...,95
    Predecessors, 94
    See also, 95
    Successors, 95
syntax, 169
terminology, 11
test image sequence, delete, 159
test image, delete, 158
test image, display, 160
test image, select, 160
Test Images, 159
test images, load, 158
test model, 144, 158
test_region_point, 93
threshold, 34, 35, 37, 136
trigonometric
    functions, 187
true, 225
tuple, 226
    concatenation, 176, 177
tuple_abs, 189
tuple_acos, 188
tuple_add, 180
tuple_and, 188
tuple_asin, 188
tuple_atan, 188
tuple_atan2, 188
tuple_band, 181
tuple_bnot, 181
tuple_bor, 181
tuple_bxor, 181
tuple_ceil, 189
tuple_chr, 191
tuple_chrt, 191
tuple_concat, 178
tuple_cos, 188
tuple_cosh, 188
```

tuple\_cumul, 189 tuple\_deg, 189 tuple\_deviation, 189 tuple\_div, 180 tuple\_environment, 191 tuple\_equal, 186 tuple\_exp, 189 tuple\_fabs, 189 tuple\_find, 178 tuple\_floor, 189 tuple\_fmod, 189 tuple\_greater, 186 tuple\_greater\_equal, 186 tuple\_int, 189 tuple\_inverse, 191 tuple\_is\_number, 191 tuple\_ldexp, 189 tuple\_length, 178 tuple\_less, 186 tuple\_less\_equal, 186 tuple\_log, 189 tuple\_log10, 189 tuple\_lsh, 181 tuple\_max, 189 tuple\_max2, 189 tuple\_mean, 189 tuple\_median, 189 tuple\_min, 189 tuple\_min2, 189 tuple\_mod, 180 tuple\_mult, 180 tuple\_neg, 180 tuple\_not, 188 tuple\_not\_equal, 186 tuple\_number, 191 tuple\_or, 188 tuple\_ord, 191 tuple\_ords, 191 tuple\_pow, 189 tuple\_rad, 189 tuple\_rand, 191 tuple\_real, 189 tuple\_regexp\_match, 185 tuple\_regexp\_replace, 185 tuple\_regexp\_select, 186 tuple\_regexp\_test, 186 tuple\_remove, 178

tuple\_round, 189 tuple\_rsh, 181 tuple\_select, 178 tuple\_select\_range, 178 tuple\_select\_rank, 189 tuple\_sgn, 189 tuple\_sin, 188 tuple\_sinh, 188 tuple\_sort, 191 tuple\_sort\_index, 191 tuple\_split, 185 tuple\_sqrt, 189 tuple\_str\_bit\_select, 184 tuple\_strchr, 184 tuple\_string, 181 tuple\_strlen, 184 tuple\_strrchr, 184 tuple\_strrstr, 184 tuple\_strstr, 184 tuple\_sub, 180 tuple\_sum, 189 tuple\_tan, 188 tuple\_tanh, 188 tuple\_uniq, 178 tuple\_xor, 188 type, 226 boolean. 225 integer, 225 real, 225, 226 string, 225, 226 Undo, 57Unicode, 53 UNIX, 200, 216 unnamed. 43 unsaved changes, 43 until, 102, 103, 194 until. 194 Update Graphics Window, 73 Update Program Counter, 72 Update Variables, 72, 119 Update Window, 81 Use Model, 158 User, 119 User Interface, 64 UTF-8 encoding, 53, 70

\_, 172 variable window. 15 variable window, 117, 226 layout, 119 resize, 119 tabs (All, Auto, User), 119 variables, 172 view image pyramid, 149 view model image, 149 view test image, 160 visible objects, 160 Visual Basic, 9 export of HDevelop programs, 213 Visual Basic .NET, 9 export of HDevelop programs, 208, 210 Visualization menu Apply Immediately, 81 Clear Graphics Window, 79 Close Graphics Window, 79 Color, 80 Colored. 80 Display, 79 Draw, 80 Feature Histogram, 86, 137 Feature Inspection, 86, 138 Gray Histogram, 86, 133 Image Size, 79 Line Width, 80 Lut, 81 New Zoom Window, 86 Open Graphics Window..., 78 Paint, 81 Reset Parameters, 82 Save Window ..., 87 Set Parameters...,82 Shape, 81 Update Window, 81 Window Size, 79 Zoom Window, 86, 131 Visualization Settings, 71 warning, 141 watersheds, 95 while, 75, 90, 102, 194, 196 while loop, 194 while, 218, 225

variable

Window

Cascade Windows, 98 Open Graphics Window, 97 Open Operator Window, 97 Open Program Listing, 97 Open Variable Window, 97 Organize Windows, 97 Window Size, 79 window title, 43 Windows, 200, 216 write\_string, 42, 80

#### XLD, 226

colors, 80 line width, 80

Zoom, 131 Zoom Window, 86