This manual shows how to create HALCON extension packages based on new operators written in C.

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.

2. Edition November 1997
4. Edition April 1999
5. Edition October 2000

Microsoft, Windows, Windows NT, and Windows 2000 are either trademarks or registered trademarks of Microsoft Corporation.
Linux is a trademark of Linus Torvalds.
Sun and Solaris are either trademarks or registered trademarks of Sun Microsystems.
HP, HP-UX, and PA-RISC are either trademarks or registered trademarks of Hewlett-Packard Company.
Silicon Graphics, SGI, and IRIX are either trademarks or registered trademarks of Silicon Graphics, Inc..
DIGITAL UNIX, Alpha AXP, and Tru64 are either trademarks or registered trademarks of Compaq Computer Corporation.

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

More information about HALCON can be found at:

http://www.mvtec.com/halcon/
About This Manual

This manual describes how to extend HALCON by additional operators encapsulated in HALCON packages using the Extension Package Interface. Before starting to use the Extension Package Interface, MVTec strongly recommends that the user should be familiar with the standard HALCON system.

This manual is written for the expert HALCON user who wants to extend the system for specific requirements. Thus, the reader should be familiar with the standard HALCON system. Furthermore, C programming skills are required. Finally, the reader should know about his/her programming environment (that is how to invoke the compiler/linker etc.).

This manual is divided into the following parts:

- **Introduction**
  This chapter provides a short overview of HALCON packages and their creation. Furthermore, an example showing the integration of a simple operator is presented.

- **Operator Description (def-files)**
  This chapter summarizes the minimum required operator description used by the HALCON compiler hcomp as well as the complete operator description which is needed to provide a full integration of new operators in HDevelop and to generate documentation files.

- **Style Guide for Programming**
  This chapter introduces basic style guides for how to program HALCON operators. Especially, the HALCON memory management is explained.

- **HALCON Data Types**
  In this chapter the most important HALCON data structures to handle iconic data and control parameters are presented.

- **Handling Iconic Objects and Control Parameters**
  This chapters contains a set of routines to facilitate the programming of the operator interface and accessing the basic HALCON data structures.

- **Special Routines for Typical Supply Procedures**
  This chapter describes a set of convenience routines for standard situations.

- **Creating a New HALCON Package**
  The last chapter explains how to use the HALCON compiler hcomp to generate HALCON packages for different architectures.
Release Notes

Please note the latest updates of this manual:

  The name “C-Interface” has been changed to “Extension Package Interface” in order to prevent its confusion with the programming language interfaces of HALCON. The structure of the manual has been revised. Sections concerned with creating and installing extension packages were updated. In the chapter “Operator Description”, the new entries used by Parallel HALCON have been added. Furthermore, the introduction of Parallel HALCON caused changes and extensions concerning data types, routines, and macros in the corresponding chapters.


  The chapter “About This Manual” has been introduced. A complete revision of the manual has been done. Chapter 1 (“The Structure of HALCON Packages and the Extension Package Interface”) has been extended by section 1.3 (“Installing a Package”). The handling of XLD structures has been included in chapters 5 to 6. Chapter 5 (“Handling Iconic Objects and Control Parameters”) has replaced the chapter “Convenience Interface Macros” and chapter 6 (“Special Routines for Typical Supply Procedures”) has replaced the chapter “Elementary Interface Macros.” Chapter 7 (“Creating a New HALCON Package”) has been extended concerning UNIX systems.

  The semantics of `HALlocRL`, `HALlocRLNum`, and `HFreeRL` has been changed: These routines are now based on `HALloc` and thus provide permanent memory. The old semantics (memory for temporary data) is used for the new routines `HALlocRLTmp`, `HALlocRLNumTmp`, and `HFreeRLTmp`. Note that the new names better reflect the behaviour of the routines. Refer to section 3.2 for a detailed discussion.

  A mechanism for providing a user-defined error text for new operators has been included (see `HSetErrText` in section 3.6).

  **Do not use older HALCON versions together with packages based on HALCON 5.1 and vice versa.** Please update/re-compile all old packages.

- **2nd Edition (November 1997)**
  The extension concept of the HALCON system has been generalized. Instead of one user extension up to 99 extension packages can be handled by the new concept. This results basically in a new arrangement of the generated files and a modified use of the environment variable HALCONEXTENSIONS. Please (re-)read chapter 1 (“The Structure of HALCON Packages and the Extension Package Interface”) and chapter 7 (“Creating a New HALCON Package”) carefully.
# Contents

1 Introduction ........................................... 1
   1.1 HALCON Operators .................................. 2
      1.1.1 Using HALCON Operators in C and C++ .............. 2
      1.1.2 Internal Structure of HALCON Operators .......... 2
   1.2 HALCON Packages ................................ 4
      1.2.1 Packages and Parallel HALCON .................. 5
      1.2.2 Directory Structure of a HALCON Package ........... 5
      1.2.3 How to Create and Use a HALCON Package .......... 6
   1.3 Installing a HALCON Package ....................... 7
      1.3.1 Windows NT / 2000 .................................. 7
      1.3.2 Linux ............................................ 8
      1.3.3 Solaris ........................................... 8
      1.3.4 IRIX ............................................. 8
      1.3.5 HP-UX ........................................... 8
      1.3.6 DIGITAL UNIX (Tru64 UNIX) ........................ 8
   1.4 An Example ........................................ 9
   1.5 Additional Sources of Information .................. 13

2 Operator Description ('def'-files) ..................... 15
   2.1 Short Operator Description .......................... 15
   2.2 Extended Operator Description ....................... 16
      2.2.1 Header ......................................... 17
      2.2.2 short ......................................... 18
      2.2.3 abstract ...................................... 18
      2.2.4 module ....................................... 19
      2.2.5 chapter ...................................... 19
      2.2.6 functionality .................................. 20
      2.2.7 keywords ..................................... 20
      2.2.8 predecessor, successor, alternatives .............. 20
      2.2.9 see also ..................................... 21
      2.2.10 attention ..................................... 21
      2.2.11 result_state ................................... 21
      2.2.12 parallelization ................................ 22
      2.2.13 complexity .................................... 24
      2.2.14 example ...................................... 25
      2.2.15 references .................................... 25
   2.3 Describing the Parameters ........................... 25
      2.3.1 Name ........................................... 27
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.3.2</td>
<td>default_type</td>
<td>27</td>
</tr>
<tr>
<td>2.3.3</td>
<td>sem_type</td>
<td>27</td>
</tr>
<tr>
<td>2.3.4</td>
<td>multivalued</td>
<td>29</td>
</tr>
<tr>
<td>2.3.5</td>
<td>costs_weight</td>
<td>29</td>
</tr>
<tr>
<td>2.3.6</td>
<td>postprocessing</td>
<td>29</td>
</tr>
<tr>
<td>2.3.7</td>
<td>description</td>
<td>31</td>
</tr>
<tr>
<td>2.3.8</td>
<td>type_list</td>
<td>31</td>
</tr>
<tr>
<td>2.3.9</td>
<td>default_value</td>
<td>32</td>
</tr>
<tr>
<td>2.3.10</td>
<td>values</td>
<td>32</td>
</tr>
<tr>
<td>2.3.11</td>
<td>value_list</td>
<td>32</td>
</tr>
<tr>
<td>2.3.12</td>
<td>value_min, value_max</td>
<td>32</td>
</tr>
<tr>
<td>2.3.13</td>
<td>step_rec, step_min, value_function</td>
<td>32</td>
</tr>
<tr>
<td>2.3.14</td>
<td>value_number</td>
<td>33</td>
</tr>
<tr>
<td>2.3.15</td>
<td>assertion</td>
<td>34</td>
</tr>
<tr>
<td>2.3.16</td>
<td>multichannel</td>
<td>34</td>
</tr>
<tr>
<td>2.4</td>
<td>Text in 'def'-files</td>
<td>34</td>
</tr>
<tr>
<td>3</td>
<td>Style Guide for Programming</td>
<td>37</td>
</tr>
<tr>
<td>3.1</td>
<td>Basic Numeric Data Types</td>
<td>37</td>
</tr>
<tr>
<td>3.1.1</td>
<td>Local Variables / Temporary Results</td>
<td>37</td>
</tr>
<tr>
<td>3.1.2</td>
<td>Procedure Parameters</td>
<td>38</td>
</tr>
<tr>
<td>3.1.3</td>
<td>Arrays</td>
<td>38</td>
</tr>
<tr>
<td>3.2</td>
<td>Memory Management</td>
<td>38</td>
</tr>
<tr>
<td>3.2.1</td>
<td>Temporary Data</td>
<td>39</td>
</tr>
<tr>
<td>3.2.2</td>
<td>Permanent Data</td>
<td>42</td>
</tr>
<tr>
<td>3.2.3</td>
<td>Debugging</td>
<td>45</td>
</tr>
<tr>
<td>3.3</td>
<td>Structuring Programs</td>
<td>45</td>
</tr>
<tr>
<td>3.4</td>
<td>Name Conventions for Procedures</td>
<td>46</td>
</tr>
<tr>
<td>3.5</td>
<td>Input / Output</td>
<td>46</td>
</tr>
<tr>
<td>3.6</td>
<td>Error Handling</td>
<td>47</td>
</tr>
<tr>
<td>3.7</td>
<td>Notes on Image Processing Operators</td>
<td>48</td>
</tr>
<tr>
<td>4</td>
<td>HALCON Data Types</td>
<td>49</td>
</tr>
<tr>
<td>4.1</td>
<td>Pixel Data ('Himage')</td>
<td>49</td>
</tr>
<tr>
<td>4.2</td>
<td>Region Data ('Hrregion')</td>
<td>51</td>
</tr>
<tr>
<td>4.3</td>
<td>XLDs ('Hcont', 'Hpoly')</td>
<td>52</td>
</tr>
<tr>
<td>4.4</td>
<td>Control Parameters ('Hpar', 'Hcpar')</td>
<td>54</td>
</tr>
<tr>
<td>5</td>
<td>Handling Iconic Objects and Control Parameters</td>
<td>59</td>
</tr>
<tr>
<td>5.1</td>
<td>Basic Access to Iconic Input Objects</td>
<td>59</td>
</tr>
<tr>
<td>5.1.1</td>
<td>HGetObj</td>
<td>60</td>
</tr>
<tr>
<td>5.1.2</td>
<td>HGetComp</td>
<td>62</td>
</tr>
<tr>
<td>5.1.3</td>
<td>HGetRL</td>
<td>63</td>
</tr>
<tr>
<td>5.1.4</td>
<td>HGetImage</td>
<td>63</td>
</tr>
<tr>
<td>5.1.5</td>
<td>HGetXLD</td>
<td>64</td>
</tr>
<tr>
<td>5.2</td>
<td>Additional Routines for Accessing Input Image Objects</td>
<td>66</td>
</tr>
<tr>
<td>5.2.1</td>
<td>HGetDRL</td>
<td>66</td>
</tr>
<tr>
<td>5.2.2</td>
<td>HGetFDRL</td>
<td>66</td>
</tr>
<tr>
<td>5.2.3</td>
<td>HGetURL</td>
<td>66</td>
</tr>
</tbody>
</table>
5.2.4 HGetDImage ........................................ 67
5.2.5 HGetObjNum ........................................ 67
5.3 Loop Macros for Accessing Single Input Objects .............. 69
  5.3.1 HAllObj ........................................ 69
  5.3.2 HAllComp ....................................... 70
5.4 Creating Objects and Writing Output Object Parameters ........ 71
  5.4.1 HCrObj ........................................ 71
  5.4.2 HCopyObj ...................................... 72
  5.4.3 HPutDRL ....................................... 73
  5.4.4 HPutImage ..................................... 73
  5.4.5 HDefObj ....................................... 74
  5.4.6 HPutDImage .................................... 75
  5.4.7 HCrlImage ...................................... 75
  5.4.8 HCrlXLD ........................................ 76
5.5 Reading and Writing Control Parameters ....................... 78
  5.5.1 HGetCPar ....................................... 78
  5.5.2 HGetSPar ....................................... 78
  5.5.3 HGetCParNum ................................... 79
  5.5.4 HGetPPar ....................................... 80
  5.5.5 HPutCPar ....................................... 81
  5.5.6 HPutPPar ....................................... 82
  5.5.7 HAllocStringMem ................................ 82
5.6 Auxiliary Macros and Procedures ................................ 83
  5.6.1 HCKP ................................ .......... 83
  5.6.2 HCKNoObj ...................................... 84
  5.6.3 HRLDecomp ..................................... 85
  5.6.4 HNumOfChannels ................................ 86
6 Special Routines for Typical Supply Procedures ............... 87
  6.1 Loop Macros ........................................ 87
    6.1.1 HALlReg ...................................... 88
    6.1.2 HALlSegm ..................................... 89
    6.1.3 HALlFilter ................................... 92
    6.1.4 HALlFilter2 .................................. 94
  6.2 Object Generation ................................... 95
    6.2.1 HNewRegion .................................. 95
    6.2.2 HPutRect ...................................... 97
    6.2.3 HDupObj ....................................... 97
7 Creating a New HALCON Package ................................ 99
  7.1 The HALCON Compiler ‘hcomp’ .......................... 99
    7.1.1 Selection of the Host Language .................. 99
    7.1.2 Creating the ‘help’-files ....................... 100
    7.1.3 Creating the Manuals ............................ 101
    7.1.4 Creating HTML Files ............................. 101
    7.1.5 Miscellaneous ................................ 102
  7.2 Generating HALCON Packages .............................. 102
    7.2.1 Creating the Operator Libraries ................ 102
    7.2.2 Creating the C Interface ....................... 103
Chapter 1

Introduction

HALCON can be extended by up to 99 additional operator packages. During the initialization of the system all packages indicated by the environment variable HALCONEXTENSIONS are automatically loaded. A package typically contains libraries with the new operators, their prototypes, the operator information needed by HDevelop, and the HTML online documentation. For the programming of such a package the HALCON Extension Package Interface is used, e.g., to manipulate parameters of HALCON operators and to read or write iconic data (images, regions, XLDs1) from the HALCON data base.

Furthermore, the operators provided by HALCON itself are also based on the functionality of the Extension Package Interface. Thus, the Extension Package Interface is both the interface between application programs and the operator layer of the the HALCON system and the interface between the operator layer and the object data base.

A reason why to extend the capabilities of HALCON by using the Extension Package Interface might be one of the following:

- Extension of the pool of image processing operators,
- Integration of special image processing hardware,
- Integration of a special graphics software package.

Once the user has extended HALCON by a package containing his/her own operators, they can be used within all supported host-languages, and the interactive tool HDevelop.

This chapter gives a short introduction to HALCON packages and operators, including an example describing the integration of a simple operator. The following chapters present detailed information that will allow you to write your own packages. As a common example, a package called halconuser will be used (see %HALCONROOT%/examples\extension_package\halconuser on the HALCON CD).

Except of UNIX specific sections in this manual file paths and environment variables are printed in the Windows NT/2000 convention, e.g.,

%HALCONROOT%/examples\extension_package\halconuser

1 eXtended Line Description; i.e. subpixel contours and polygons.
to denote the subdirectory `halconuser` containing an example package within the HALCON base directory referenced by the environment variable `HALCONROOT`. The same expression in UNIX convention would look like

```
$HALCONROOT/examples/extension_package/halconuser
```

## 1.1 HALCON Operators

This section describes the external interface of HALCON operators, i.e., how to use them in a programming language, and their internal structure.

### 1.1.1 Using HALCON Operators in C and C++

There are two different modes of calling operators from HALCON/C\(^2\): One way is to call operators for simple applications with only one value per control parameter (simple mode). This is the easiest way and sufficient for many applications. When using complex procedures that need more than one value per parameter, data is transferred within tuples (Tuple). Iconic objects are always represented as tuples (Object) so that one image object can contain several images, regions, or XLDs.

There exists a special interface for C++ supporting a flexible management of several types. Especially, the handling of multi-value control parameters is transparent to the user. Moreover, this interface enfolds an appropriate class hierarchy for handling different image analysis data.

However, the defined class hierarchy has a fixed mapping between the HALCON operators and the provided classes. This mapping follows predefined rules and is therefore partly generic. A user-defined HALCON operator cannot be linked to an arbitrary position inside the class hierarchy. Thus, these operators are integrated in C++ as C functions using the generic data type `Object` in order to pass arguments to and from the operator. This kind of integration is also used if you export your HDelay program as C++ code.

For further information please take a look at the corresponding manuals.

### 1.1.2 Internal Structure of HALCON Operators

HALCON operators (like the new operators in `halconuser`) typically consist of two procedures: One procedure – the supply procedure – receives the input data, tests its consistency, passes it to the processing action procedure and returns the output data after the processing. If the input data contains any composed objects, e.g., image tuples, the supply procedure has to extract the single parts. An object key can represent an unlimited number of iconic objects and every image object may consist of several components (one region and an unlimited number of gray value channels; see also Fig. 5.2 on page 61). Generally, a user may assume that a HALCON operator is able to handle single iconic objects as well as composed objects (i.e. multivalue: `optional` is set; see page 29) and that it can work on multi-channel images (i.e. multichannel: `optional` is set; see page 34). So, the author of a new HALCON operator should implement appropriate mechanisms for extracting the needed data from composed

\(^2\)The HALCON version that allows the usage of HALCON operators within C programs.
1.1. HALCON OPERATORS

objects (or multi-channel images) within the supply procedure. The name of this procedure designates the internal C-procedure name specified in the def-file (cf. sections 2 and 2.2).

After preprocessing the input data, the action procedure is called within the supply procedure. This procedure performs the specific image processing. In most cases the action procedure receives only the already extracted single components (e.g., regions of type `image`, channels of type `image`, or pointers to the raw image data) and parameter values. The results are returned to the supply procedure where they are passed to the generated interface that returns them to the calling system. Any supply procedure returns an error code of type `error`. Thus, it is necessary to return an error code, which is `error` if no error occurred. In Windows NT and Windows 2000, the supply procedures (like `cUserThreshold()` in `cipuserthreshold.c`) must be explicitly exported by the operator DLL `package.dll`. This is done by the macro `HUserExport` as a prefix to `error`, see Fig. 1.6. In UNIX environments `HUserExport` may be omitted, but in this case the package will not run on Windows NT and Windows 2000 systems.

Only one parameter (a so called procedure handle) is passed to a supply procedure. All other input data (iconic objects and control values) is read and written with the help of this identifier. Moreover, it allows the unique identification of a HALCON operator call while running HALCON on parallel hardware. Instead of further parameters, all in- and output to and from the supply procedure is done with the help of several (internal) buffers, as illustrated in Fig. 1.1.

![Figure 1.1: Data flow within HALCON.](image)

The buffer contains input data as well as output data. Control parameters (containing all kind of control values like integers, floating-point values or strings) are stored directly in the buffers, whereas all iconic data (including images, regions and XLDs) are represented by an object key. Accesses to iconic data can be done with the help of data base procedures by using these object keys. The Extension Package Interface provides special procedures and macros for reading and writing elements of the parameter buffer within the supply procedure. Every parameter belongs to one of the four classes
CHAPTER 1. INTRODUCTION

- input object
- output object
- input control
- output control

that determines the name of the suitable procedures/macros (e.g.: HGetCPar is used to get an input control value). To identify a specific parameter within a class, its number is passed to the procedure/macro. This is the number of the parameter within the parameter-list of a HALCON operator counted for every parameter class (e.g.: to get the value of the second input control parameter, the number 2 must be passed to HGetCPar).

The read-/write access on the buffer is done automatically by the generated interface. The code of this interface is generated by the HALCON compiler (see page 11 and section 7.1) from a so-called definition file (def-file). Any programmer of a new HALCON operator must provide such a definition file, which contains all relevant information e.g., concerning in- and output parameters: information about the parameter class (input-/output, object/control), about the parameter type, etc. The syntax and semantics of the definition file is explained in chapter 2.

1.2 HALCON Packages

Similar to the standard HALCON system a programmer has to provide (at least) two libraries (DLLs in Windows NT / 2000, shared libraries in UNIX environments) in order to extend HALCON by a new operator package (see chapter 7).

- A library containing the new operators (written in C). Most of this manual is about how to write such operators based on internal data structures / procedures of the HALCON system. This library has the same name as the package itself, in our example halconuser.
- Libraries encapsulating the generated interface code (for C and C++). These libraries (i.e., the operator library and one of the interface libraries) have to be linked to new applications in order to provide the interface of the new operators to the host language of your choice. Most of the work concerning these libraries is done by the HALCON compiler hcomp, see section 7.1 for details. In the example package these libraries are called halconuser and halconusercpp.

The names of all these libraries are derived from the name of the package itself: A package package contains the libraries halconuser, packagec, and packagecpp.

HDevelop uses the C language interface, that is the library halconuser, during initialization to integrate the new operators in its runtime environment.

---

3 This is true for the Windows NT/2000 version. For UNIX systems the prefix lib is added to the libraries resulting in libpackage, libpackagec, and libpackagecpp.
1.2.1 Packages and Parallel HALCON

Extension packages can also be used in Parallel HALCON. What’s more, Parallel HALCON will automatically parallelize your new operators if you add the corresponding information in the definition file (see section 2.2.12).

Similar to the HALCON libraries, which exist in two versions (e.g., halcon.dll and parhalcon.dll), you must provide separate versions of your package libraries to be used together with Parallel HALCON. The names of these libraries must start with the prefix par; e.g., the Parallel HALCON version of the example package halconuser consists of the libraries parhalconuser.d11, parhalconusersrc.d11, and parhalconusercpp.d11 under Windows NT / 2000 or libparhalconuser.so, libparhalconusersrc.so, and libparhalconusercpp.so under UNIX.

The Parallel HALCON version of the libraries is created analogously to the standard HALCON version with only small differences. See section 7.2 for details.

1.2.2 Directory Structure of a HALCON Package

A HALCON package resides in a directory with the same name as the package itself. In our example this is the directory halconuser in %HALCONROOT%\examples\c_interface. This directory must at least contain the following subdirectories:

- **lib\%ARCHITECTURE%**: In this subdirectory the libraries encapsulating the new operators (package) and the corresponding language interfaces (packagec and packagecpp) reside. **Note, that neither the name of these libraries nor their position within the package should be changed.** Otherwise the HALCON system cannot load the package. The environment variable ARCHITECTURE is set during the installation of HALCON; it describes the platform HALCON is running on. The following table gives an overview of the currently supported platforms and the corresponding values of ARCHITECTURE.

<table>
<thead>
<tr>
<th>ARCHITECTURE</th>
<th>Operating System (Platform)</th>
</tr>
</thead>
<tbody>
<tr>
<td>i586-nt4</td>
<td>Windows NT 4.0 (^5) on Intel Pentium or compatible</td>
</tr>
<tr>
<td>sparc-sun-solaris7</td>
<td>Solaris 7 on Sparc Workstations</td>
</tr>
<tr>
<td>mips-sgi-irix6.5</td>
<td>IRIX 6.5 on SGI Workstations (Mips processors)</td>
</tr>
<tr>
<td>hppa1.1-hp-hpux10</td>
<td>HP-UX 10.x on Workstations (at least PA 1.1 processors)</td>
</tr>
<tr>
<td>alpha-dec-osf4.0</td>
<td>DIGITAL UNIX 4.0 (^6) on Alpha processors</td>
</tr>
<tr>
<td>i586-linux2.2</td>
<td>Linux 2.2 on Intel Pentium (or compatible)</td>
</tr>
</tbody>
</table>

- **bin\%ARCHITECTURE%**: For Windows NT and Windows 2000, this subdirectory has to contain the DLLs package.d11, packagec.d11, and packagecpp.d11 corresponding to the above libraries. We also recommend to place compiled example programs or additional utilities of a package in this subdirectory.

\(^4\)For UNIX systems the prefix lib has to be added to the libraries resulting in libpackage, libpackagec, and libpackagecpp.

\(^5\)or Windows 2000

\(^6\)now called Tru64 UNIX
• help: This subdirectory contains files with data for the online access to the knowledge base. They must be generated and placed in this directory in order to use a package inside of HDevelop. Without these files HDevelop cannot retrieve the information needed to access the operators of a package.

We also recommend to provide additional subdirectories:

• include: In this directory the generated include files containing the prototypes for the operators of a package should be placed.

• doc: In this directory the documentation of the package should be placed (see also $%\text{HALCONROOT}\%\text{doc}$ for comparison). Especially, it is necessary to put the generated HTML files in $\text{doc}\text{html}\text{reference}\text{hdevelop}$ in order to enjoy the online help from the operator window of HDevelop.

• examples: This directory might contain some typical example programs (e.g., HDevelop dev-files) demonstrating the use of the new operators.

• images: If the provided examples need specific images, they should be placed in this directory. In that case we recommend to include the path to this subdirectory in the environment variable HALCONIMAGES in order to allow access to these images without using an absolute file path in the programs.

### 1.2.3 How to Create and Use a HALCON Package

In summary, the following steps are necessary to create a HALCON package with the name package integrating new operators into the HALCON system. As noted in section 1.2.1, to create a Parallel HALCON package, you must create a second version of the libraries. A detailed description of the steps can be found in the referenced sections of this manual:

1. Description of the new operator(s) in one or more def-file(s) (cf. chapter 2).

2. Creation of a library (called package) containing the supply procedures in C and the corresponding action procedures (cf. chapter 3–6). This library has to be placed in the package subdirectory\footnote{In the Windows NT/2000 version, the DLLs corresponding to the link libraries have to be placed in bin or bin\%ARCHITECTURE\%.} or lib\%ARCHITECTURE\%.

3. Creation of a library containing the language dependent operator interface (package\c and package\cpp) based on the def-file(s) with the help of hcomp (cf. section 7.1). These libraries have to be placed in the subdirectory lib or lib\%ARCHITECTURE\% of the package.

4. Generation of files for online access to the knowledge base with the help of hcomp. These files are used, e.g., by HDevelop. They have to be placed in the subdirectory hhelp of the package.

5. Generating the HTML reference files with the help of hcomp (optional). The generated files have to be placed in the subdirectory doc\html\reference\hdevelop, c, or cpp of the package.

7. Extension of the package list in the environment variable HALCONEXTENSIONS by the complete file path of the new package, e.g.,

```
%HALCONROOT%\examples\extension_package\halconuser
```

For specific architectures additional environment variables have to be modified, see sections 1.3 and 7.2.5.

8. Linking of the generated libraries together with the HALCON library to the image analysis application (cf. chapter 7). For this, you have to modify additional environment variables, depending on the operating system (see below). Please note, that HDevelop is able to access the new operators dynamically without linkage.

### 1.3 Installing a HALCON Package

In order to install a HALCON package you have to copy the package to your hard disk and add the complete path of the package to the environment variable HALCONEXTENSIONS, e.g.,

```
%HALCONROOT%\examples\extension_package\halconuser
```

Note, that the delimiter between paths in an environment variable is a semicolon on Windows NT or Windows 2000 systems and a colon on UNIX systems.

**Never change the name of a package or the corresponding names of the libraries or DLLs contained in a package.**

These names are encoded *within* the libraries/DLLs. If you change the names this information will not match any longer. Thus, the loader of the operating system will fail to open the dynamic libraries. In order to use the new package within HDevelop you have to **restart** the program. For generating a standalone application using the package you have to link the C or C++ interface library of the package to the application code, see sections 7.2 and 7.2.5.

If the package contains images used e.g., within example programs you might want to include the corresponding directory (e.g., *images* within the package) in the environment variable HALCONIMAGES to access those images without specifying a complete path.

For UNIX architectures you have to modify additional environment variables in order to use a HALCON package (see below).

### 1.3.1 Additional Information for Windows NT and Windows 2000

To be able to link the package DLL to your application program, the complete DLL file path of the new package, e.g.,

```
%HALCONROOT%\examples\extension_package\halconuser\bin\i586-nt4
```

has to be added to the environment variable PATH.

**Do not copy a package DLL into the Windows system directories**, as it would be loaded twice in this case!
1.3.2 Additional Information for Linux

On UNIX PCs running Linux you have to include the package library subdirectory `lib/i586-linux2.2` in the environment variable `LD_LIBRARY_PATH` in order to use a package. This has to be done in any case – regardless whether you plan to use a package within HDevelop only or you want to create stand-alone applications. Otherwise the operating system will fail to open the interface library `libpackagec.so` or `libpackagecpp.so` needed to access the package.

1.3.3 Additional Information for Solaris

On UNIX workstations running Solaris you have to add the package library subdirectory `lib/sparc-sun-solaris7` to the environment variable `LD_LIBRARY_PATH` in order to run stand-alone applications based on the package. Otherwise the loader will fail to access the interface library `libpackagec.so` or `libpackagecpp.so` linked to the application, see section 7.2.5.

Note, that it is not necessary to modify `LD_LIBRARY_PATH` in order to use the package within HDevelop.

1.3.4 Additional Information for IRIX

On UNIX workstations running IRIX you have to include the package library subdirectory `lib/mips-sgi-irix6.5` in the environment variable `LD_LIBRARY_PATH` in order to run stand-alone applications based on the package. Otherwise the loader will fail to access the interface library `libpackagec.so` or `libpackagecpp.so` linked to the application, see section 7.2.5.

Note, that it is not necessary to modify `LD_LIBRARY_PATH` in order to use the package within HDevelop.

1.3.5 Additional Information for HP-UX

On UNIX workstations running HP-UX you have to include the package library subdirectory `lib/hppa1.1-hp-hpux10` in the environment variable `SHLIB_PATH` in order to use a package. This has to be done in any case – regardless whether you plan to use a package within HDevelop only or you want to create stand-alone applications. Otherwise the operating system will fail to open the interface library `libpackagec.s1` or `libpackagecpp.s1` needed to access the package.

1.3.6 Additional Information for DIGITAL UNIX (Tru64 UNIX)

On UNIX workstations running DIGITAL UNIX (now called Tru64 UNIX) you have to include the package library subdirectory `lib/alpha-dec-osf4.0` in the environment variable `LD_LIBRARY_PATH` in order to run stand-alone applications based on the package. Otherwise the loader will fail to access the interface library `libpackagec.so` or `libpackagecpp.so` linked to the application, see section 7.2.5.
Note, that it is *not* necessary to modify `LD_LIBRARY_PATH` in order to use the package within HDevelop.

### 1.4 An Example

The following example illustrates the steps described in section 1.2. Assume, one wants to implement a new operator named `user_thresh` to extract all pixels with a gray value larger than a threshold value specified in an input parameter. The use of `user_thresh` within a C program is illustrated in Fig. 1.2.

```c
main()
{
    Object  Image, Region;
    int     Thresh;

    read_image(&Image, "monkey");
    Thresh = 100;
    user_thresh(Image, &Region, Thresh);
}
```

Figure 1.2: Calling the operator `user_thresh` in C. A more detailed example is provided in `%HALCONROOT%\examples\c_interface\halconuser\source\testthreshold.c`.

To use `user_thresh` in this way within C-programs,

- The new operator must be implemented.
- The corresponding HALCON interface-code must be generated.
- The corresponding package must be created especially containing the operator and interface libraries.
- The application must be linked.

```c
user_thresh <- CIPUserThreshold[Image:Region:Threshold$i:];
```

Figure 1.3: Defining a new operator `user_thresh`.

First of all one has to create a definition file (file extension `.def`), as illustrated in Fig. 1.3 and Fig. 1.4. Fig. 1.3 shows only the absolute minimum of information that a definition file must contain in order to call a new operator within C-programs. When using C++ or HDevelop, an extended version is necessary as described in section 2.2. The minimal form of such an extended definition is illustrated in Fig. 1.4. The `.def`-file specifies:

- the name of the operator (`user_thresh`),

---

8To use it within C++ basically the same steps are necessary, see section 7.
user_thresh <- CIPUserThreshold[Image:Region:Threshold:]

short.german
  Schwellwertoperator.;

short.english
  Selection of gray values by thresholding.;

module
  basic;

chapter.german
  BenutzerErweiterungen;

chapter.english
  UserExtensions;

functionality
  image;

parameter
  Image: input_object;
  sem_type: image;

parameter
  Region: output_object;
  sem_type: region;

parameter
  Threshold: input_control;
  default_type: integer;
  multivalue: false;
  sem_type: number;

Figure 1.4: An example for an extended def-file of the operator user_thresh (minimal form), cf. %HALCONROOT%\examples\c_interface\halconuser\def\threshold.def.

- the name of the supply procedure that has to be implemented in C (CIPUserThreshold),
- names and types of the operator’s parameter(s). The names of the parameters are only important for the manuals and within HDevelop.

The minimal form of the extended operator def-file includes: A short description (short), the HALCON module (module) the operator belongs to, the mapping to a chapter (chapter) of the manual, the mapping to an object (functionality), and for every parameter its semantic type (sem_type). In addition, the type (default_type) and number (multivalue) must be specified for control parameters.

Our operator user_thresh has got the following parameters:

- one input object parameter (Image),
- one output object parameter (Region),
1.4. AN EXAMPLE

- one input control parameter (Threshold) of type int with exactly one value and
- no output control parameter.

```bash
comp -u -C -halconuser threshold.def
```

Figure 1.5: Call of `hcomp` for HALCON/C.

The operator definition within the def-file now is used by `hcomp` to generate the appropriate interface code. Fig. 1.5 shows the call of `hcomp`. Note, that within this manual `halconuser` is used as name of the HALCON package to be created. By calling `hcomp` with this options\(^9\) the files `Hhalconuser.c` (interface-code) and `Hhalconuser.h` (prototype of the new HALCON operator) are created.

In the next step one has to program the new HALCON operator. This is done by implementing the supply and action procedure. This results in the typical structure of HALCON operators as illustrated in Fig. 1.6.

```c
/* action procedure */
Herror IPBThreshold(proc_handle,region,image,
                  width,threshold,region_out)
{

  ...
}

/* supply procedure - always with this parameter! */
HUserExport Herror CIPUserThreshold(Hproc_handle proc_handle)
{
  HGetCPar(..,&Threshold,..); /* get control */
  /* parameter */
  HAL1Segm(proc_handle,&region,&image,1,i) /* work on all */
  /* input images */
{
  HCheck(proc_handle,IPBThreshold(...)); /* call action */
  /* procedure */
  HNewRegion(proc_handle,region_out); /* store (result) */
  /* region */
}
return(H_MSG_TRUE);
}
```

Figure 1.6: Sample source code of supply `CIPUserThreshold()` and action `IPBThreshold()` procedure.

Compiling the source code generates an object file (in the example `cipuserthreshold.o`) that should be integrated into the user extension library (`halconuser`). Moreover, the generated interface code must be converted to the corresponding language dependent interface library: In case of a C application\(^10\) this is `halconuser` containing `Hhalconuser.o`. The

\(^9\) see section 7.1 for a complete description.

\(^10\) HDevelop uses the C version as well.
file makefile in the directory %HALCONROOT%\examples\extension_package\halconuser contains a makefile for generating the example package itself under UNIX. makefile.nt is the corresponding file to be used via nmake under Windows NT and Windows 2000. The makefiles makefile_testprog and makefile_testprog.nt generate example applications using the new operators.

Now we can use our new operators within C-programs. But they are not yet available within HDevelop, because the online help files have not been generated so far. There is also still missing any kind of documentation, such as the HTML manual pages.

```
    hcomp -u -M threshold.def
```

Figure 1.7: Calling the HALCON compiler hcomp to create the help files.

The generation of the online help files (as they are used e.g., by HDevelop) can also be done with the help of the HALCON compiler hcomp, see Fig. 1.7. However, to do so an extended version of the def-file must be available. The generated help files must be placed in the subdirectory help of the package.

```
DEF_SOURCES = threshold.def
HCOMP = $(HALCONROOT)/bin/$(ARCHITECTURE)/hcomp

all: hdevelop c cpp

hdevelop: $(DEF_SOURCES)
    $(HCOMP) -L"t:all" $(DEF_SOURCES)
    mv *.html doc/html/reference/hdevelop

c: $(DEF_SOURCES)
    $(HCOMP) -L"c:all" $(DEF_SOURCES)
    mv *.html doc/html/reference/c

cpp: $(DEF_SOURCES)
    $(HCOMP) -L"cpp:all" $(DEF_SOURCES)
    mv *.html doc/html/reference/cpp
```

Figure 1.8: Makefile for generating HTML references.

The help files generated by now are used to access information about the new operators by calling specific HALCON operators like get_operator_info. This is of importance especially for constructing graphical user interfaces like HDevelop. For the user a more convenient way to access information online is to browse through HTML documents. The HTML documentation of HALCON operators can be generated by hcomp as well, see Fig. 1.8. In this example, HTML documents are generated in Trias (HDevelop) syntax (-L"t:all"), in C-syntax (-L"c:all"), and in C++- syntax (-L"cpp:all"). The HTML files have to be placed in the corresponding subdirectories doc\html\reference\hdevelop, c, or cpp of the package.

If you now add the package path to HALCONEXTENSIONS and start HDevelop the user extensions, e.g. user_threshold, are automatically available in the specified menu.
(i.e., chapter). In our example this menu is called UserExtensions as specified in \%HALCONROOT\%\examples\cInterface\halconuser\def\threshold.def.\textsuperscript{11} You can select and execute the user-defined operators like all the built-in operators of the HALCON system. The HTML documentation is available via the Help button within the operator window.

Examples of how to use the user defined extensions in HDevelop can be found in the subdirectory examples of \%HALCONROOT\%\examples\extension_package\halconuser (*.dev). Furthermore, the *.dev programs have been exported as C++ programs. The *.cpp files can be found in the subdirectory source. You can compile the C++ files using the already mentioned makefiles. These *.cpp files also serve as an example for the integration of user defined operators into the host language C++. Note, that before compiling the exported C++ code containing operators from the package halconuser, the corresponding user extension libraries must have been created. This can be done via the makefile in \%HALCONROOT\%\examples\extension_package\halconuser which will also generate the needed user extension C++ library halconusercpp.

Please note, that on some systems not all users have writing permissions in the directories mentioned above. To experiment with the example package we recommend to create a private copy in your working directory. In such a case you must of course use the actual path to your copy when modifying environment variables.

### 1.5 Additional Sources of Information

For further information you may consult the following manuals:

- **Getting Started with HALCON**
  An introduction to HALCON in general, including how to install and configure HALCON.

- **HDevelop User’s Manual**
  An introduction to the graphical development environment of the HALCON system.

- **HALCON/C++ User’s Manual**
  How to use the HALCON library in your C++ programs.

- **HALCON/C User’s Manual**
  How to use the HALCON library in your C programs.

- **HALCON/COM User’s Manual**
  How to use the HALCON library in your COM programs, e.g., in Visual Basic.

- **Frame Grabber Integration Programmer’s Manual**
  A guide on how to integrate a new frame grabber in the HALCON system. Note that in some cases you might define new operators (using the Extension Package Interface) instead of using the standard HALCON Frame Grabber Integration Interface in order to exploit specific hardware features of a framegrabber board.

- **HALCON/HDevelop, HALCON/C++, HALCON/C, HALCON/COM**
  The reference manuals for all HALCON operators (versions for HDevelop, C++, C, and COM).

\textsuperscript{11}UserExtensions is integrated into the menu Operator of HDevelop.
All these manuals are available as PostScript (ps) and Acrobat Reader (pdf) documents. The reference manuals are available as HTML documents as well. For the latest version of the manuals please check

http://www.mvtec.com/halcon/

Please see also the example package

%HALCONROOT%\examples\extension_package\halconuser

which will be referenced many times within this manual.
Chapter 2

Operator Description (def-files)

The following chapters give a detailed description of the steps mentioned in section 1. This is done with respect to the examples within the directory %HALCONROOT%\examples\extension_package\halconuser as well as system-defined HALCON operators. We start with the generation of an operator description.

2.1 Short Operator Description

Before implementing an operator the author should think about the task of the operator and what kind of user interface is best for calling the operator. This specification is used to automatically generate the interface code for the desired host language with the HALCON compiler hcomp.

To do this, it is necessary to describe at least the following features beside the operator name in a def-file:

- the name of the operator as used within the application programming language,
- the name of the C-procedure (supply procedure) called by HALCON in order to start the processing (this procedure must be implemented later on),
- all iconic objects and control parameters that are needed by the operator as input (input parameters),
- all iconic objects and control parameters that are returned by the operator as result (output parameters),
- the types of these parameters,
- the number of values per parameter (there is only a distinction between parameters that need or can process exactly one value and those expecting more than one value).

There exist two kinds of parameters within HALCON: Iconic object parameters and parameters containing any other data – so called control parameters. By distinguishing input- and output-parameters we get four classes of parameters: Input iconic object parameters, output iconic object parameters, input control parameters, and output control parameters. Basically, specifying the number of parameters per class would suffice to describe the operator. The internal access (in particular within the supply procedure) on individual parameters works this way by using the parameter’s position within its class. But to refer to single parameters and add further information regarding a parameter, it is useful to give it a name.
The information specified up to now is already sufficient to integrate an operator within the HALCON/Extension Package Interface by using \texttt{/CW/CR/D3/D1/D4}. It describes the operator properly and may be specified as a “one line short version” of the \texttt{def} file according to the following syntax:

\begin{verbatim}
  extern_operator_name <- InternCProcedure
  [InputObjects:OutputObjects:InputCtrlParams:OutputCtrlParams];
\end{verbatim}

Chapter 2.2 explains the meaning of the single identifiers and the name convention. Please note, that in contrast to the header mentioned there, the short version header must be finished by a semicolon (;) and that the default type and the number of values is specified together with the name of a control parameter (InputCtrlParams and OutputCtrlParams). This is done by concatenating a dollar sign and an additional character to the name encoding the type and number of values (cf. the example in Fig. 2.1).

\begin{verbatim}
  user_inside <- CIPUserSelect[Regions:RegionSelected:Row$i,Column$i:];
\end{verbatim}

Figure 2.1: Short version of the definition of the operator \texttt{user_inside} (cf. \texttt{\%HALCONROOT\%\ex\c_interface\halconuser\def\regionfeatures.def}).

The following table shows how to code types and number of values into a character. Lower-case characters stand for \textit{exactly one value} in the specified parameter. This corresponds to assigning \texttt{multivalue: false} in the extended version. Capital letters specify parameters that contain \textit{more than one value} passed as a tuple (multivalue: true).

<table>
<thead>
<tr>
<th>Type</th>
<th>C-type</th>
<th>One Value</th>
<th>Unlimited Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>long</td>
<td>i</td>
<td>I</td>
</tr>
<tr>
<td>Floating-point Value</td>
<td>double</td>
<td>f</td>
<td>F</td>
</tr>
<tr>
<td>String</td>
<td>char *</td>
<td>s</td>
<td>S</td>
</tr>
</tbody>
</table>

The specification described so far is sufficient to integrate the operator into C applications. However, in general the new operators should also be usable by HDevelop or within C++ applications. To achieve this, an extended version of the operator description must be generated. This is described in the following section.

\section{2.2 Extended Operator Description}

An operator description within a \texttt{def} file must start with a header. The following entries are variable in their order. All of these slots begin with a determined keyword and are finished by a semicolon (;). Semicolons within the text not defining the end of a slot must be quoted with a backslash (\texttt{\}).

Some slots are language dependent: For example \texttt{short.english} contains a short description of the operator in English, \texttt{short.german} the same description in German. The currently supported languages are English and German. In order to allow an international usage of operators we strongly recommend always to provide all .english slots.
Note that all slots specified within the operator description are accessible online within the HALCON system using the operator `get_operator_info`, see the Reference Manuals for details. Describing the single parameters completes the operator description. Section 2.3 explains how to do this.

### 2.2.1 Header

The operator description starts with the header

```c
extern_operator_name <- InternCProcedure
    [InputObjects:OutputObjects:InputCtrlParams:OutputCtrlParams]
```

Note that in contrast to the short version described in section 2 the header doesn’t contain any information about the types of the control parameters (e.g. `$f$`) and does not end with a semicolon (`;`).

The following example is a short version of the operator description for the dynamic threshold operator `dyn_threshold`. For better legibility there should be a comment at the beginning of every operator description, like

```c
/**************************** dyn_threshold ****************************/

dyn_threshold <- CIPDynThreshold
    [OrigImage,ThresholdImage:RegionDynThresh:Offset,LightDark:]
```

Comments within `def`-files are indicated according to the C-syntax (`/* ... */`).

There are a couple of conventions/restrictions concerning the names of parameters: All word parts should begin with a capital letter (e.g., `RegionDynThresh`). They are directly concatenated without any separating character. Especially, they must not contain any underscore (`_`)!

The external operator name exclusively consists of lower-case characters with word parts separated by underscores. This operator name is used within C programs, low-level calls¹ of the operator in C++, or within HDevelop.

The four parameter classes (input/output object/control) are separated by colons (`:`). One class is described by a list of all parameter names within the class separated by commas. If there is no member of a class for an operator, the list remains empty. The example described above defines the following mapping between parameters and parameter classes:

<table>
<thead>
<tr>
<th>parameter class</th>
<th>keyword (cf. page 27)</th>
<th>parameter in the example</th>
</tr>
</thead>
<tbody>
<tr>
<td>input iconic objects</td>
<td>input_object</td>
<td>OrigImage, ThresholdImage</td>
</tr>
<tr>
<td>output iconic objects</td>
<td>output_object</td>
<td>RegionDynThresh</td>
</tr>
<tr>
<td>input control parameters</td>
<td>input_ctrl</td>
<td>Offset, LightDark</td>
</tr>
<tr>
<td>output control parameters</td>
<td>output_ctrl</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>

¹Within the HALCON/C++ class hierarchy corresponding `methods` use slightly different conventions: `extern_operator_name` is transformed into `ExternOperatorName`.
It is helpful to follow certain conventions, when choosing names for parameters. They should be meaningful (i.e. one should avoid names like ‘Aac’) and consequently should be given in English. It is convention to denote parameters that specify a position within an image matrix by \textit{Row(s)} (not \textit{Line}!) and \textit{Column(s)} or \textit{Col(s)}. Parameters defining a dimension are called \textit{Width} and \textit{Height}.

\subsection*{2.2.2 short}

The keyword \texttt{short} starts the short description of the operator, in our example

\begin{verbatim}
short.english
   Segment an image using a local threshold.;
\end{verbatim}

As all slots specified by the keyword \texttt{.english}, this slot contains purely textual information. Text can be written in different languages and can contain \LaTeX\-commands according to special syntactic rules that are defined in section 2.4.

\subsection*{2.2.3 abstract}

A more detailed description of the operator is given within the slot \texttt{abstract}. The description of the example is here shown as a shortened version:
2.2. EXTENDED OPERATOR DESCRIPTION

abstract.english

\UpRef{dyn_threshold} selects from the input image those regions in which the pixel fulfill a threshold condition. Let \$g_{\{o\}} = g_{\{\ParRef{OrigImage}\}}\$, and \$g_{\{m\}} = g_{\{\ParRef{ThresholdImage}\}}\$. Then the condition for \ParRef{LightDark} = \ValRef{‘light’} is:

@a
\begin{align*}
g_{\{o\}} & \ge g_{\{m\}} + \ParRef{Offset} \\
\end{align*}
\end{verbatim}

@l[[
\begin{align*}
g_{\{o\}} & \le g_{\{m\}} + \ParRef{Offset} \\
\end{align*}
\end{verbatim}

@e

For \ParRef{LightDark} = \ValRef{‘dark’} the condition is:

@a
\begin{align*}
g_{\{o\}} & \le g_{\{m\}} - \ParRef{Offset} \\
\end{align*}
\end{verbatim}

@l[[
\begin{align*}
g_{\{o\}} & \ge g_{\{m\}} - \ParRef{Offset} \\
\end{align*}
\end{verbatim}

@e

Finally, for \ParRef{LightDark} = \ValRef{‘equal’} it is:

@a
\begin{align*}
g_{\{m\}} = \ParRef{Offset} \le g_{\{o\}} \le g_{\{m\}} + \ParRef{Offset} \\
\end{align*}
\end{verbatim}

@l[[
\begin{align*}
g_{\{m\}} - \ParRef{Offset} & \le g_{\{o\}} \le g_{\{m\}} + \ParRef{Offset} \\
\end{align*}
\end{verbatim}

@e

This means that all points in \ParRef{OrigImage} for which the gray value is larger or equal to the gray value in \ParRef{ThresholdImage} plus an offset are aggregated into the resulting region.

Again, refer to section 2.4 for the special syntax of text.

2.2.4 module

module

basic;

This slot denotes the module inside HALCON the operator should belong to. For user extensions the module ‘basic’ is recommended.

2.2.5 chapter

To achieve a useful structuring of HALCON operators, they are all arranged in a hierarchy of chapters and sections. The assignment defined in the slot chapter is reflected in the reference manuals and in the menu Operator of HDvelop.
The structuring can be refined by specifying a section: `chapter[,section]`, e.g.,

```plaintext
chapter.english
Filter, Edges;
```

Note, that it is not allowed to insert operators and sections into the same chapter at the same level of hierarchy. So if you decide to use sections within a chapter, all operators within this chapter must be assigned to one of these sections. Furthermore, the chapter names may not contain spaces. If you need multi-word chapter names, please use a dash (`-`) between each word, e.g., `Affine-Transformations`.

### 2.2.6 functionality

```plaintext
functionality
  image;
```

This slot denotes the class of which the operator should become a method within an object oriented programming language like C++\(^2\). Generally this corresponds to the semantic type of the first parameter, e.g., an image object (`image`). The specified name is only a symbolic one. The actual class name depends on the programming language and is provided by the HALCON compiler `hcomp`. Possible values for `functionality` in the current version are:

- any, window, image, region, object, xld_cont, xld_poly, xld_para,
- xld_mod_para, and xld_ext_para.

### 2.2.7 keywords

Furthermore, one can assign a list of keywords to an operator. They are used e.g., by HDevelop to support the search for the proper operators for a given problem.

```plaintext
keywords.english
  Threshold, Gray-Value-Threshold, Dynamic-Threshold, Local-Threshold;
```

### 2.2.8 predecessor, successor, alternatives

The following slots also support interactive development of image processing application with HALCON. They are used to define potential, convenient or necessary predecessor (``predecessor``) and successor operators (``successor``) or to define alternatives (``alternatives``). All operators within these lists are referenced by their name.

\(^2\)User defined extensions are not inserted in the class hierarchy but considered as global methods.
2.2.9 see also

The slot `see also` contains a list of operators that are used for similar tasks or help to understand how an operator works.

```
see also
mean_image, smooth_image, gauss_image, connection, rank_region, dilatation1;
```

2.2.10 attention

The slot `attention` contains hints for using the operator or specific limitations.

```
attention.english
If \ParRef{Offset} is chosen from 0 to $\ldots$ 1 $@l^+1 dots1^0e$ usually a very noisy region is generated, requiring large storage. If \ParRef{Offset} is chosen too large (\$> 60, say) it may happen that no points fulfill the threshold condition (i.e. \ an empty region is returned). If \ParRef{Offset} is chosen too small (\$< -60, say) it may happen that all points fulfill the threshold condition (i.e. \ a full region is returned).
```

2.2.11 result_state

The slot `result_state` defines the result value of an operator and the corresponding exception handling. The syntactical rules for the text are again the same as described in section 2.4.

```
result_state.english
\OpRef{dyn_threshold} returns TRUE if all parameters are correct. The behavior with respect to the input images and output regions can be determined by setting the values of the flags `\ValRef{no_object_result}`, `\ValRef{empty_region_result}`, and `\ValRef{store_empty_region}` with `\OpRef{set_system}`. If necessary, an exception is raised.
```
2.2.12 parallelization

The slot parallelization contains information about the parallelization characteristics of an operator when used in Parallel HALCON (see the manual Getting Started with HALCON for more information about Parallel HALCON). On the one hand, this regards the possibility of using the operator in a parallel, for example, multithreaded application and on the other hand this regards the automatic parallelization which Parallel HALCON uses to speed up the operator’s processing when working with multi-processor hardware.

The slot parallelization consists of several sub-slots, each starting with a characteristic keyword followed by a colon and finishing with a semicolon. The example below shows the slot for the operator dyn_threshold, which is not “local”, needs neither complete nor mutual exclusion, and is parallelized on tuple level. The single subslots and their meaning will be described in the following sections.

```
parallelization
  process_exclusively: false;
  process_locally: false;
  process_mutual: false;
  method: split_tuple;
```

process_exclusively: true,false;
This subslot of parallelization assigns, whether an operator is processed completely exclusively (‘true’) by the main HALCON process/thread or not. An exclusive processing means that no other operator will be processed by Parallel HALCON while the exclusive operator is running. If an exclusive operator is started and other operators are still running, the processing of the exclusive operator is delayed until all other (currently running) operators have completed.

An exclusively processed operator is always processed without any parallelization by the main HALCON thread/process. Note that even if Parallel HALCON is reentrant for most of the operators, there are still some operators that have to be processed exclusively due to reasons of their implementation.

If no subslot process_exclusively is filled in with an operator, it is assumed that the operator needs no completely exclusive processing (“optimistic” assumption).

process_mutual: true,false;
This subslot of parallelization assigns, whether Parallel HALCON processes an operator under mutual exclusion (‘true’) against itself or not. Here, a mutual exclusion means that the same operator may not run twice or more at the same time; for example, there will only run one instance of adapt_template at a time, whereas other operators may run concurrently to adapt_template.

An exclusively processed operator is always processed without any parallelization by the main HALCON thread/process.

If no subslot process_mutual is filled in with an operator, it is assumed that the operator needs no mutual exclusion (“optimistic” assumption).

process_locally: true,false;
This subslot of parallelization assigns, whether an operator must be processed locally
2.2. EXTENDED OPERATOR DESCRIPTION

(‘true’) by a program thread, or whether it may called by any ("external") thread without
problems. The slot is mainly important when programming multithreaded applications
under Windows NT and Windows 2000. Here, there exists a direct mapping between
program threads and graphical elements, such as windows, dialog boxes or button controls. In short, a graphical element only exists in the context of its associated thread. This
can cause severe problems, for example, hang the application, if another thread tries to
perform user interactions via graphical elements which belong to other threads. For ex-
ample, troubles might occur under Windows NT/2000, if one thread opens a window via
open_window and another thread tries to get input from this window via draw_circle.

As it is not always obvious for all operators, whether they work with critical (graphical)
elements or not, the slot process_locally was introduced. If process_locally is
set to ‘true’, this signals the programmer that the corresponding operator must be used
carefully within multithreaded Windows NT/2000 applications. The most simple way
to avoid any problems with such operators is to process them all under mutual exclusion
within the main thread of the program.

As a side effect, Parallel HALCON processes an operator without parallelization regard-
less the settings of the subslots method and domain_split, if process_locally is set to
‘true’. This avoids problems with ‘local’ operators and multithreading and makes sense
because ‘local’ operators normally are responsible for graphical interaction and thus are
not suitable for parallelization.

If no subslot process_locally is filled in with an operator, it is assumed that the operator
must be processed “locally” (“pessimistic” assumption). However, this subslot should be
filled in in any case, because it contains a very useful information for programmers of
multithreaded applications.

method: none,split_tuple,split_channel,split_domain;

This subslot of parallelization can contain one or more of the strings above and
specifies, which type of (automatic) parallelization the operator is suitable for:

- **split_tuple:** Parallel HALCON does not automatically parallelize the operator.
- **split_tuple:** Parallel HALCON parallelizes the operator by splitting every input
  image tuple into several subsets of the tuple; the tuple subsets are then processed in
  parallel.
- **split_channel:** Parallel HALCON parallelizes by splitting every multichannel
  (input) image into several subsets of the channels; the subsets are then processed
  in parallel.
- **split_domain:** Parallel HALCON parallelizes by splitting the domain of every
  input image into several parts; the parts are then processed in parallel.

If method is set to anything other but ‘none’, the parameter slot costs_weight must be
filled in for every input control parameter and the parameter slot postprocessing must be
filled in for every output control parameter of the operator (see page 29)!

If method is set to ‘split_domain’, the slot domain_split (see below) must also be set
correctly and may not be omitted!

If the slot process_locally is set to ‘true’, method should be set to ‘none’, because the
operator won’t be parallelized then (see description of process_locally above).
Note that the slot \texttt{method} may contain any combination of the strings ‘split\_tuple’, ‘split\_channel’, and ‘split\_domain’, because an operator may be parallelized by using any combination of those three methods. However, if ‘none’ is specified with the slot \texttt{method}, no other string should be specified with it in order to keep the description consistent.

If no slot \texttt{method} is filled in with an operator, it is assumed that the operator is \textit{not} suitable for being automatically parallelized (‘pessimistic’ assumption).

\texttt{domain\_split}: an unsigned integer value;

This subslot of \texttt{parallelization} must (only) be set for operators that are suitable for being parallelized by splitting the domain of iconic input objects (i.e. slot \texttt{method} contains ‘split\_domain’). For such operators the value of \texttt{domain\_split} assigns, whether the input domain may be split disjunctively (value ‘0’) or not. In the latter case the domain is split into several overlapping stripes (note, that the splitting of domain regions is always done row-wise).

If the domain is split disjunctively, the value (\textgreater{} 0) of \texttt{domain\_split} assigns the index ([1..n]) of that input control parameter which determines the extent of the overlapping regions (in number of pixels) by its content. For example, for the operator \texttt{eliminate\_min\_max} the extent of the overlapping regions depends on the height of the filter mask. Therefore, the content of the second input control parameter (compare the operator’s description in the Reference Manuals) directly determines the extent of the overlapping regions. Thus, the slot \texttt{domain\_split} is set to ‘2’ for \texttt{eliminate\_min\_max}.

The example below shows the whole \texttt{parallelization} slot for \texttt{eliminate\_min\_max}.

```plaintext
parallelization
   process_exclusively: false;
   process_locally:    false;
   process_mutual:     false;
   method:             split\_tuple, split\_channel, split\_domain;
   domain\_split:      2;
```

If \texttt{method} contains ‘split\_domain’, the slot \texttt{domain\_split} must be filled in. In all other cases, the slot should be omitted!

Though, we recommend to specify the whole slot \texttt{parallelization}, it is also possible to completely leave it out in an operator’s description. In this case, it is assumed that the operator is \textit{reentrant}, i.e. it is \textit{not} processed under (complete or mutual) exclusion, and that it must be processed “locally”. The latter also means that \textit{no} parallelization is used when processing the operator. This assumption corresponds exactly to those which are used in the case of missing single subslots (see descriptions above).

\subsection{complexity}

The slot \texttt{complexity} describes the complexity of the operation in terms of number of points along a contour, the area of an image region etc.

```plaintext
complexity.english
   Let $F$ be the area of the input region. Then the runtime complexity is $O(F)$.
```

HALCON Extension Package Interface / 2000-11-16
2.2.14 example

It is possible to describe an example under the keyword example in order to illustrate the usage of an operator. An extension of the keyword specifies the programming language of the example.

```plaintext
example.trias
   /* Looking for regions with the diameter D: */

   mean_image(Image:Mean:D*2,D*2:) >
   dyn_threshold(Image,Mean:Seg:5,'light') >
   connection(Seg:Objects);
```

The extension .trias denotes the image processing language Trias that uses a syntax basically equal to the notation within HDevelop. Moreover, the extensions .c and .cpp can be used. The text of the example is processed unmodified, so it must not contain any \LaTeX-special characters. The only exception is the semicolon that needs a prefixed backslash, because it would signal the end of the example text otherwise. This must be considered especially for C- and C++-examples.

2.2.15 references

The slot references (not specified in the dyn_threshold example) is used to insert references to literature into the documentation, e.g.,

```plaintext
references
   R.M. Haralick, K.G. Shapiro: ‘‘Computer and Robot Vision’’;
   Vol. 1, Addison-Wesley Publishing Company, 1992;
```

2.3 Describing the Parameters

Again, the definition of dyn_threshold is used as an example. Every description of a single parameter starts with the keyword parameter and contains several slots. A slot begins with a characteristic keyword followed by a colon and is finished by a semicolon. All slots specified for the parameters of an operator are accessible online within the HALCON system using the operator get_param_info, see the Reference Manuals for details.
parameter
OrigImage: input_object;
description.english: Image to be segmented.;
sem_type: image;
type_list: byte,int2,int4,real;
multivalue: optional;

parameter
ThresholdImage: input_object;
description.english: Image containing the local thresholds.;
sem_type: image;
type_list: byte,int2,int4,real;
multivalue: optional;

parameter
RegionDynThresh: output_object;
description.english: Segmented regions.;
sem_type: region;
multivalue: optional;

parameter
Offset: input_control;
description.english: Offset added to ThresholdImage.;
sem_type: number;
type_list: integer,real;
default_type: real;
default_value: 5.0;
values: 1.0, 3.0, 5.0, 7.0, 10.0, 20.0, 30.0;
multivalue: false;
assertion: -255 < Offset & & Offset < 255;
costs_weight: 0;

parameter
LightDark: input_control;
description.english: Extract light, dark or similar areas?;
sem_type: string;
type_list: string;
default_type: string;
default_value: light;
value_list: dark,light,equal,not_equal;
multivalue: false;
costs_weight: 0;

The minimum of information needed for every parameter is its name and type, its default type, its semantic type, the number of values allowed (multivalue), and, if the operator should be automatically parallelized, the “costs weight” of input control parameters and the “postprocessing” of output control parameters, see below.
2.3. DESCRIBING THE PARAMETERS

2.3.1 Name

parameter
  Name: input_object, output_object, input_control, output_control;

This defines the name and the class of a parameter. The parameters must be described in the same order as in their definition within the header of the operator description.

2.3.2 default_type

default_type: integer, real, string;

Only for control parameters! This slot specifies, what C-type to use in general. It must have exactly one of the three above values.

2.3.3 sem_type

sem_type: class[.spec];

This slot determines the semantics and specifies the class of data passed as parameters, when using an object-oriented language. Names of classes within the def-file are only symbolic. The mapping to the actual class names is provided by the HALCON compiler hcomp. HDevelop also uses the semantic types, e.g., in order to provide specific inspection routines (like for frame grabber handles). For complex classes it must be defined by .spec, which aspect of the class is represented by the parameter. By this, one can pass complex objects that are described by several parameters of the C-procedure as one logical parameter.

The following semantic types (classes) are available at the moment:

- **Iconic data (object parameters):**
  - object (any iconic object: images, regions, XLDs)
  - image (images)
  - region (regions)
  - xld (any XLDs: lines in eXtended Line Description)
    - xld_cont, xld_poly, xld_para, xld_mod_para, xld_ext_para.

- **Elementary data (control parameters):**
  - number (unspecific)
  - integer, real, string
  - grayval (gray value)
  - channel (channel number).

- **Handles (control parameters):**
  - frame grabber
- file
- ocr (OCR classifier)
- bg_estimation (see the HALCON operator create_bg_esti)
- class_box (general classifier)
- window.id (HALCON window).

- Arrays (control parameters):
  - histogram.values (gray value histogram)
  - distribution.values (distribution).

- Geometric data (control parameters):
  - point.x, point.y (position)
  - extent.x, extent.y (dimensioning)
  - circle.center.y, circle.center.x, circle.radius (circle)
  - line.begin.y, line.begin.x, line.end.y, line.end.x (line)
  - rectangle.origin.y, rectangle.origin.x (rectangle: upper left corner),
    rectangle.corner.y, rectangle.corner.x (lower right corner) or
    rectangle.extent.x rectangle.extent.y (expansion – as alternative to specifying the 2nd corner)
  - rectangle2.center.y, rectangle2.center.x (rectangle with arbitrary orientation: center),
    rectangle2.angle.rad (orientation – in radians),
    rectangle2.hwidth, rectangle2.hheight (half the size)
  - ellipse.center.y, ellipse.center.x (ellipse: centre),
    ellipse.angle.rad (orientation – in radians),
    ellipse.radius1, ellipse.radius2 (radii)
  - arc.center.y, arc.center.x (circle: center),
    arc.angle.rad (angle stretched by the circular arc – in radians),
    arc.begin.y, arc.begin.x (starting point of circular arc)

- Numerical region descriptions (control parameters):
  - coordinates.y, coordinates.x (coordinates)
  - contour.y, contour.x (points of a contour)
  - chord.y, chord.x1, chord.x2 (runlength code)
  - polygon.y, polygon.x (polygoneal representation)
  - chain.begin.y, chain.begin.x, chain.code (chain code).

- Miscellaneous (control parameters):
  - angle.rad or angle.deg (angle in radians or degrees)
  - filename.named (file name).
Please note:

- y-coordinates correspond to rows, x-coordinates to columns in the image.
- Please use the above semantic types for characterizing parameters whenever they are applicable.
- For control parameters characterizing a composed object like a circle etc.: Please use the specific order of parameters like indicated above.

### 2.3.4 multivalue

```
multivalue: true, false, optional;
```

This slot describes the number of values passed in a parameter: ‘true’ means that an array (tuple) of values must be passed. If ‘false’ is specified, exactly one value must be passed. ‘optional’ allows both.

All slots mentioned so far must appear in any extended operator description. Moreover, the following optional slots are recommended to provide further information about parameters:

### 2.3.5 costs_weight

```
costs_weight: an unsigned integer value;
```

This slot must only be used for input control parameters of operators which are suitable for being automatically parallelized (i.e. for operators for that the parallelization subslot method does not contain ‘none’; see page 23). Here, it assigns, whether the time for processing the operator directly depends on the content (value) of the corresponding input control parameter. If costs_weight is set to ‘0’, the duration of the operator processing does not directly depend on the content of the corresponding input control parameter. In this case, the parameter will not be taken into account when Parallel HALCON decides, whether an operator should be parallelized for a given allocation of input parameters or not. This also means, that this parameter will not be checked during the hardware training (compare the description of check_par_hw_potential in the Reference Manuals). This helps to speed up the training. If costs_weight contains a value greater than ‘0’, this assigns that there is a direct dependency between the content of the corresponding input control parameter and the computation time of the operator. In this case, Parallel HALCON will take into account this parameter during its hardware check (check_par_hw_potential) in order to determine its influence on the operator’s processing time. Note that the slot costs_weight must be filled in for every input control parameter, if the operator should be automatically parallelized, i.e. if the operator’s slot method does not contain ‘none’ (see page 23).

### 2.3.6 postprocessing

```
postprocessing: none, tuple_add, tuple_min, tuple_max, tuple_concat, channel_add, channel_min, channel_max, channel_concat, domain_add, domain_min, domain_max, domain_concat;
```

HALCON 6.0
This slot must only be used for output control parameters of operators which are suitable for being automatically parallelized (i.e. for operators for that the parallelization subslot method does not contain ‘none’; see page 23). Here, postprocessing specifies, which kind of postprocessing is used with the single result values of a parallel processed operator. If, for example, an operator is parallelized by splitting a tuple of iconic input objects and the single result values of the parallel processed operator must be put into one output control object again, the slot postprocessing should contain the keyword ‘tuple_concat’ which means “concatenate result values”. Or if an operator is parallelized by splitting the single channels of iconic input objects and the overall result value is the minimum of the single result values of the parallel processed operator, postprocessing should contain the keyword ‘channel_minimum’. Obviously, the slot postprocessing can contain several strings in order to define different postprocessing steps for different levels of parallelization (tuple, channel, and domain); of course, there should only be one string per level. If ‘none’ is specified (in this case, no other string should be specified), no postprocessing is done on every parallelization level. This means, that the overall result value of the corresponding output control parameter is directly determined by the result value of the first parallelized operator instance (e.g. the instance which worked on the first tuple element).

Alltogether, Parallel HALCON currently supports the following postprocessing steps:

- **none**: no postprocessing; adopt first value;
- **tuple_add**: overall result value is the sum of the single result values of the tuple parallelization;
- **tuple_min**: overall result value is the minimum of the single result values of the tuple parallelization;
- **tuple_max**: overall result value is the maximum of the single result values of the tuple parallelization;
- **tuple_concat**: overall result value is a tuple which contains all the result values of the tuple parallelization (“concatenation”);
- **channel_add**: overall result value is the sum of the single result values of the channel parallelization;
- **channel_min**: overall result value is the minimum of the single result values of the channel parallelization;
- **channel_max**: overall result value is the maximum of the single result values of the channel parallelization;
- **channel_concat**: overall result value is a tuple which contains all the result values of the channel parallelization (“concatenation”);
- **domain_add**: overall result value is the sum of the single result values of the parallelization on domain level;
- **domain_min**: overall result value is the minimum of the single result values of the parallelization on domain level;
- **domain_max**: overall result value is the maximum of the single result values of the parallelization on domain level;
2.3. DESCRIBING THE PARAMETERS

- **domain_concat**: overall result value is a tuple which contains all the result values of the parallelization on domain level ("concatenation");

The example below shows the definition of postprocessing for the operator circularity, which calculates the shape factor for the circularity (similarity to a circle) of input regions. If circularity is tuple parallelized the overall result is a tuple of values, of which the single elements contain the “circularity” of the single input regions. Thus the postprocessing of the corresponding output control parameter is assigned by ‘tuple_concat’.

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Circularity:</td>
<td>output_control;</td>
</tr>
<tr>
<td>description.english</td>
<td>Roundness of the input region(s).;</td>
</tr>
<tr>
<td>sem_type:</td>
<td>real;</td>
</tr>
<tr>
<td>type_list:</td>
<td>real;</td>
</tr>
<tr>
<td>default_type:</td>
<td>real;</td>
</tr>
<tr>
<td>multivalue:</td>
<td>optional;</td>
</tr>
<tr>
<td>assertion:</td>
<td>0 &lt;= Circularity &amp; Circularity &lt;= 1.0;</td>
</tr>
<tr>
<td>postprocessing:</td>
<td>tuple_concat;</td>
</tr>
</tbody>
</table>

Note that the slot postprocessing must be filled in for every output control parameter, if the operator should be automatically parallelized, i.e. if the operator’s slot method does not contain ‘none’ (see page 23).

### 2.3.7 description

**description.english:** \texttt{\LaTeX/Ascii-Text};

This slot contains a short description of the parameter. The extension .english refers to a English description. In addition to that a German description might be given in description.german. For the syntax of \texttt{\LaTeX/Ascii-Text} see section 2.4.

### 2.3.8 type_list

The slot type_list is only used in connection with object parameters with **sem_type = image** and for control parameters.

For images (**sem_type = image**):

```
type_list: any,byte,int1,int2,int4,real,cyclic,direction,complex;
```

For images this slot contains an enumeration of all supported pixel types. Note, that in many cases there will be more than one supported pixel type. Thus, type_list is a list separated by commas.

For control parameters:

```
type_list: integer,real,string;
```

For control parameters this slot contains a list of all C-types allowed for the parameter.
### 2.3.9 default_value

```plaintext
default_value: Default Value;
```

This slot defines the default value e.g., used by HDevelop to initialize the parameter in the operator window. Suggesting default values can help the programmer to find suitable values of parameters. Generally, strings are unquoted in the description (not ‘theString’, but theString); exceptions: empty string (’’) and strings with special characters (’the string’).

### 2.3.10 values

```plaintext
values: List of some typical values;
```

This slot contains a collection of possible values. This list does not have to be complete. It just suggests some typical values the user might want to try.

### 2.3.11 value_list

```plaintext
value_list: List of all values;
```

This slot contains a complete list of all allowed values. So, if a parameter can hold only a discrete number of specific values, value_list should be specified, otherwise values might be used to provide some typical values.

### 2.3.12 value_min, value_max

```plaintext
value_min: Number;
value_max: Number;
```

Instead of a list an interval may be used to specify the range of allowed values for a parameter. The interval may be unlimited in one direction. The slot value_min specifies the minimum of the allowed values, value_max the maximum.

### 2.3.13 step_rec, step_min, value_function

```plaintext
step_rec: Number;
step_min: Number;
value_function: lin, log, quad, ...;
```

If a range of values is possible for a parameter it might be a useful information to specify a suitable step width (step_rec) between distinct values to be tested. Based on this information a user interface might generate a list of suggestions for parameter values. The corresponding minimum reasonable step width can be provided by step_min. The step width can also be modified by specifying a function in the slot value_function.
2.3.14 value_number

value_number: boolean expression;

This slot contains a boolean expression that determines how the number of passed or returned values of a parameter is absolutely or relatively connected to the number of values in other parameters or some absolute values. The operators and functions listed in table 2.1 may appear within the expression.

<table>
<thead>
<tr>
<th>Group</th>
<th>Symbol</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unary Operators</td>
<td>!</td>
<td>logical NOT</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>negation</td>
</tr>
<tr>
<td>Binary Operators</td>
<td>&amp;&amp;</td>
<td>logical AND</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>==</td>
<td>equal</td>
</tr>
<tr>
<td></td>
<td>!=</td>
<td>not equal</td>
</tr>
<tr>
<td></td>
<td>&gt;</td>
<td>greater</td>
</tr>
<tr>
<td></td>
<td>&gt;=</td>
<td>greater or equal</td>
</tr>
<tr>
<td></td>
<td>&lt;</td>
<td>less</td>
</tr>
<tr>
<td></td>
<td>&lt;=</td>
<td>less or equal</td>
</tr>
<tr>
<td></td>
<td>+</td>
<td>sum</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>difference</td>
</tr>
<tr>
<td></td>
<td>*</td>
<td>multiplication</td>
</tr>
<tr>
<td></td>
<td>/</td>
<td>division (DIV)</td>
</tr>
<tr>
<td></td>
<td>%</td>
<td>rest of division (MOD)</td>
</tr>
<tr>
<td>Functions</td>
<td>odd</td>
<td>true, if odd</td>
</tr>
<tr>
<td></td>
<td>even</td>
<td>true, if even</td>
</tr>
<tr>
<td></td>
<td>3d</td>
<td>binary logarithm</td>
</tr>
<tr>
<td></td>
<td>length</td>
<td>length</td>
</tr>
<tr>
<td></td>
<td>number</td>
<td>number of values (only for assertion)</td>
</tr>
<tr>
<td></td>
<td>width</td>
<td>(image-)width</td>
</tr>
<tr>
<td></td>
<td>height</td>
<td>(image-)height</td>
</tr>
</tbody>
</table>

Table 2.1: Operators and functions used for the slots value_number and assertion.

Examples:

Param1 == Param2 && Param2 > 0

This means that the same number of values and at least one value must be passed with parameter 1 and parameter 2.

Param2 == 3

This means that exactly three values must be passed with parameter 2.

For output parameters this can be seen as an assertion. This may also refer to input parameters, e.g.: RegionOut <= RegionIn in connection with the HALCON operator select_shape expresses the fact that the number of output regions does not exceed the number of input regions.
2.3.15 assertion

assertion: boolean expression;

This slot contains a boolean expression that determines how the passed or returned values of parameters are absolutely or relatively connected to the values of other parameters or some absolute values. The same expressions may appear as described above for value number, see table 2.1.

Examples:

\begin{verbatim}
Param1 == Param2 && Param2 > 0
   This means that parameter 1 and 2 must have the same value and this value is greater than 0.

0 <= Param2 && Param2 <= 255
   This means that parameter 2 must have a value between 0 and 255.

Param1 <= number(ImageIn)
   This means that the value of parameter 1 must be less or equal the number of input images.

Param2 < width(ImageIn) && Param2 >= 0 && odd(Param2)
   This means that the value of parameter 2 must be between 0 and the width of the input image and must be an even number.
\end{verbatim}

2.3.16 multichannel

multichannel: true,false,optional;

This slot is only used in connections with image objects. It contains an assertion about the necessary or supported number of channels of an image. If set to false, the processing is done only on the first channel (all others are ignored), if set to true, a multichannel image must be passed. optional specifies operators that can work on more than one channel but can also work on only one channel as well.

2.4 Text in def-files

Within a def-file the semicolon (;) is used to separate the single slots. So, when using a semicolon within a text, it must be quoted with a backslash (\;).

def-files are not only used to create interface code, but also allow the generation of the manuals, the HTML documentation, and the help files used to access operator knowledge via HALCON operators like get_operator_info. For the online help and the HTML files an ASCII version of each text is needed, whereas the manuals are produced compiling a LaTeX-file generated from the def-files. To avoid the necessity of writing two text blocks for one operator (ASCII and LaTeX), both versions are generated by using one text. Therefore, the following conventions must be observed:
Umlauts: All German umlauts are written in the \LaTeX-typical notation: "ä, ..., "ü, "ö.

Quotation marks: There are two kinds of quotation marks

'\texttt{string}': This notation signals that the quoted text is a string as used within programming languages. So \texttt{hcomp} treats 'string' as a string-parameter.

"\texttt{Text}": This notation must be used in all other cases, particularly when quoting a word or text segment because of its content. So \texttt{hcomp} treats 'Text' as a quoted section of a text. Please do not forget or permute the quotation marks (starting with `' and ending with `').

Underscores: An underscore can be used without the necessity of prefixing a backslash as necessary in \LaTeX(\_\_) so that the text is easier to read. The backslashes are inserted automatically in the generated \LaTeX-version of the text. Note, that this mechanism is only active within the ASCII part of the text.

Backslash: All backslashes (\) within the text are ignored while generating the ASCII version. So the \LaTeX-linefeed-symbol (\) can be used without any problems.

Tilde: To use the tilde (e.g. as symbol for negation: set\_check(: :\~\{clear\} : :)), the following special notation must be used for the \LaTeX-text: set\_check(: :\~\{clear\} : :).

Formulas: Short formulas can be bracketed by dollar symbols ($) just as in \LaTeX. These symbols are ignored for the ASCII version of the text. To set an index of one character (e.g. $A_b$) \LaTeX allows to simply write $A_b$. This is not possible here, because of the special underscore handling. Therefore, any index must be written exactly as any longer index within \LaTeX: $A_{\text{b}}$.

HALCON: The string HALCON is generated by \Hacon. Note, that in the PostScript files derived from the generated \LaTeX-version of the def-files, \texttt{no space} is generated after the string HALCON. Thus, if you do not want to start the next character/string without a blank, that is concatenated to the string HALCON, you should use \Hacon\ instead. So "HALCON XYZ" results in "HALCON XYZ" and "\Hacon XYZ" results in "HALCON XYZ". With the ASCII version all backslashes are ignored (see above).

Names of parameters and operators: Any reference to HALCON parameters or operators has to be written as \ParRef{parameter\_name}, respectively as \OpRef{operator\_name}. These keywords and the brackets are ignored, when generating the ASCII version so that only \texttt{parameter\_name} or \texttt{operator\_name} remains in the ASCII file. For the HTML documentation hyper links are generated from these references.

Parameter values: A specific value for a parameter should be written as \ValRef{parameter\_value} . For example, the parameter \ParRef{LightDark} of the operator \OpRef{dyn\_threshold} can hold one of the values \ValRef{'light'}, \ValRef{'dark'}, or \ValRef{'equal'}. The keyword ValRef and the brackets are ignored, when generating the ASCII version so that only \texttt{parameter\_value} remains in the ASCII file.

Other \LaTeX\ commands: If you would like to use additional \LaTeX\ command within a text segment, two versions of this segment must be provided – one for ASCII (with @a signalling
the start of the ASCII section) and another as pure \LaTeX text (with @l starting the \LaTeX version and @e ending the special section). This technique is in particular necessary for tables and larger formulas.

Example:

\begin{verbatim}
@l \[ g_0 \geq g_m + \text{ParRef}\{\text{Offset}\} \]
@e
\end{verbatim}

This parenthesis can also be used for only one of the two text types:

\begin{verbatim}
... text text @l \text{aSpecialLatexCommand} @e text text ...
\end{verbatim}
Chapter 3

Style Guide for Programming

The following chapter contains some recommendations for implementing new HALCON operators. They aim on portable code that should be easy to understand for other HALCON programmers. One of the major topics is the memory management discussed in section 3.2.

3.1 Basic Numeric Data Types

This section describes the usage of basic numeric data types. Since C makes no assumption about the size of integers, the following types have been defined:

<table>
<thead>
<tr>
<th>Type Name</th>
<th>Size/Range of Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT1</td>
<td>one byte with sign</td>
</tr>
<tr>
<td>HBYTE</td>
<td>one byte, unsigned</td>
</tr>
<tr>
<td>UINT1</td>
<td>the same as BYTE</td>
</tr>
<tr>
<td>INT2</td>
<td>two bytes with sign</td>
</tr>
<tr>
<td>UINT2</td>
<td>two bytes, unsigned</td>
</tr>
<tr>
<td>INT4</td>
<td>four bytes with sign</td>
</tr>
<tr>
<td>UINT4</td>
<td>four bytes, unsigned</td>
</tr>
<tr>
<td>INT4_8</td>
<td>four or eight bytes</td>
</tr>
<tr>
<td>UINT4_8</td>
<td>four or eight bytes, unsigned</td>
</tr>
<tr>
<td>INT</td>
<td>At least two, maximal eight bytes with sign</td>
</tr>
<tr>
<td>UINT</td>
<td>At least two, maximal eight bytes, unsigned</td>
</tr>
</tbody>
</table>

Using these types ensures the portability of HALCON operators.

3.1.1 Local Variables / Temporary Results

Types and sizes for local variables (no arrays) should be handled as follows:
### 3.1.2 Procedure Parameters

Parameters of procedures should be of the following types within HALCON:

<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
<th>C-Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>1 byte</td>
<td>INT or UINT</td>
</tr>
<tr>
<td></td>
<td>2 byte</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 byte</td>
<td>INT4 or UINT4</td>
</tr>
<tr>
<td></td>
<td>4 byte</td>
<td></td>
</tr>
<tr>
<td>Floating-point Number</td>
<td></td>
<td>double</td>
</tr>
<tr>
<td>Structure</td>
<td></td>
<td>pointer</td>
</tr>
</tbody>
</table>

### 3.1.3 Arrays

For arrays or structures (i.e. larger sets) always the smallest possible data type should be used.

<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
<th>C-Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>1 byte</td>
<td>INT</td>
</tr>
<tr>
<td></td>
<td>2 byte</td>
<td>INT2 or UINT2</td>
</tr>
<tr>
<td></td>
<td>3 byte</td>
<td>INT4 or UINT4</td>
</tr>
<tr>
<td></td>
<td>4 byte</td>
<td></td>
</tr>
<tr>
<td>Floating-point Number</td>
<td>4 byte</td>
<td>float</td>
</tr>
<tr>
<td></td>
<td>8 byte</td>
<td>double</td>
</tr>
<tr>
<td>Structure</td>
<td></td>
<td>pointer</td>
</tr>
</tbody>
</table>

### 3.2 Memory Management

HALCON provides a sophisticated memory management not only to handle iconic objects, but also to allocate/deallocate arbitrary data. Please use the corresponding HALCON Extension Package Interface routines exclusively. Do not create/destroy memory blocks on the heap with the standard routines of the operating system (like `malloc` and `free`) since the HALCON routines provide a caching mechanism, garbage collection for temporary data, and debugging facilities.

We strongly recommend not to use *global variables* within HALCON operators. If they are not avoidable at least make them `static` within the file of usage. If even this is undesirable group them into structures to keep the number of global names small.
If you use static variables, be aware that they are shared between multiple threads. Please note, that Parallel HALCON uses multithreading to exploit multi-processor hardware (see the manual Getting Started with HALCON for more information). This means, that if you use Parallel HALCON, either by linking the corresponding libraries to your program or by using Parallel HDevelop, multiple instances of your code will actually share all the static variables! Furthermore, be aware that large static arrays consume a lot of memory – keep in mind that HALCON is a very large system enfolding hundreds of operators.

### 3.2.1 Temporary Data

#### Names

HALconTmp, HFreetmp, HFreenTmp, HFreetUpToTmp, HFreetAllTmp, 
HALlocRLTmp, HALlocRLNumTmp, HFreetRLtmp

#### Synopsis

```c
#include "Halcon.h"

Herror HALlocTmp( Hproc_handle proc_handle, 
                  void ** pointer, 
                  size_t size)

Herror HFreetmp( Hproc_handle proc_handle, 
                 void * pointer)

Herror HFreenTmp( Hproc_handle proc_handle, 
                 INT num)

Herror HFreetUpToTmp( Hproc_handle proc_handle, 
                     void * pointer)

Herror HFreetAllTmp( Hproc_handle proc_handle)

Herror HALlocRLTmp( Hproc_handle proc_handle, 
                    Hrlregion ** region)

Herror HALlocRLNumTmp( Hproc_handle proc_handle, 
                      Hrlregion ** region, 
                      size_t len)

Herror HFreetRLtmp( Hproc_handle proc_handle, 
                     Hrlregion * region)
```

Figure 3.1: HALCON stack management for temporary data.
Fig. 3.1 shows HALCON routines to allocate/deallocate temporary memory blocks. Internally, those blocks are stored within a stack. Therefore, the memory must be deallocated in reverse order of its allocation. There are two major advantages of using these routines:

- The underlying stacks are initially allocated as large blocks of memory. Thus, the memory is not fragmented and memory allocation is fast for subsequent calls.
- An automatic garbage collection deallocates all temporary data after a HALCON operator was executed. This is especially of importance in case of an error during the execution of the operator (otherwise, the data should have been deallocated anyway).

HAllocTmp is used for arbitrary data with the corresponding deallocation routine HFreeTmp. HFreeNTmp deallocates the last num blocks allocated by HAllocTmp. HFreeUpToTmp deallocates all recently allocated blocks up to (and including) the specified block. HFreeAllTmp deallocates all temporary blocks.

HAllocRLTmp and HAllocRLNumTmp are convenience routines that are based on HAllocTmp. HAllocRLTmp allocates as much memory as is necessary for the largest region actually stored in the HALCON data base (the minimal size is DEF_RL_LENGTH = 50000 chords). HAllocRLNumTmp allows to determine the size of memory to be allocated by specifying the number of chords. Furthermore, HAllocRLTmp and HAllocRLNumTmp initialize the data structure Hr1region for the new region (see section 4.2 for a description of Hr1region). Note, that since these routines are based on HAllocTmp the corresponding memory blocks are interleaved with the blocks allocated directly via HAllocTmp. This has to be considered while deallocating the corresponding memory.

**Temporary data on arbitrary heap positions**

The temporary data management routines described so far use a stack. This is of advantage concerning runtime, but it lacks flexibility if you do not want to deallocate memory in a fixed order again. Therefore, HALCON also provides routines for allocating temporary memory on an arbitrary position of the heap, see Fig. 3.2. Memory blocks allocated by HAllocLocal can be deallocated by HFreeLocal in an arbitrary order. However, similar to the stack-based temporary data management, all these blocks are automatically deallocated at the end of a HALCON operator. Again this mainly aims on preventing memory leaks in case of errors. As usual, the routine HReallocLocal is used to allocate a new memory block with modified size while preserving the data of the original memory block. The latter is deallocated. So never try to access pointer subsequent to HReallocLocal – use new pointer exclusively.

The convenience routines HAllocRLLocal, HAllocRLNumLocal, HReallocRLLocal, and HReallocRLNumLocal are used for handling temporary region data based on HAllocLocal. Otherwise their behaviour is similar to HAllocRLTmp etc., see above. However, note that it is possible to change the size of regions (i.e., the number of chords) using this set of routines (HReallocRLNumLocal) since the underlying memory blocks are not allocated within the internal stacks.
3.2. MEMORY MANAGEMENT

Names

HALocLocal, HRealloclocal, HFreeLocal, HFreeAllLocal,
HALocRLLocal, HALocRLNumLocal, HReallocRLNumLocal, HFreeRLLocal

Synopsis

#include "Halcon.h"

Error HALocLocal( Hproc_handle proc_handle,
size_t size,
void **pointer)

Error HReallocLocal( Hproc_handle proc_handle,
void *pointer
size_t size,
void **new_pointer)

Error HFreeLocal( Hproc_handle proc_handle,
void *pointer)

Error HFreeAllLocal( Hproc_handle proc_handle)

Error HALocRLLocal( Hproc_handle proc_handle,
HrIregion **region)

Error HALocRLNumLocal( Hproc_handle proc_handle,
HrIregion **region,
size_t len)

Error HReallocRLNumLocal( Hproc_handle proc_handle,
HrIregion *region
size_t len,
HrIregion **new_region)

Error HFreeRLLocal( Hproc_handle proc_handle,
HrIregion *region)

Figure 3.2: Temporary data on arbitrary heap positions.
3.2.2 Permanent Data

The HALCON Extension Package Interface also provides routines for permanently allocating/deallocating memory (see Fig. 3.3 and 3.4). They work similar to the standard C-procedures malloc and free, but use a caching mechanism for small data blocks and provide additional debugging information.

Note, that in contrast to HALlocLocal memory allocated by HALloc is not deallocated automatically after the execution of a HALCON operator. Thus, this routine should be used to allocate permanent data. For example, all the iconic objects stored in the HALCON data base are allocated with this routine (see chapter 5 and 6).

HNewImage allocates raw image data inside\(^1\) the data structure Himage (see section 4.1) based on HALloc. Furthermore, image is initialized and the timestamp is set to the current time. The image matrix itself is initialized with 0 if 'init_new_image' has been set to 'true' using the HALCON operator set_system. This flag can be read inside HALCON using the Extension Package Interface call

```c
HReadGV(proc_handle, HGInitNewImage, &init_new_img);
```

with init_new_img of type HBOOL and set

```c
HWriteGV(proc_handle, HGInitNewImage, {TRUE,FALSE});
```

Thus, it is possible to buffer the global setting of this flag, set the flag to FALSE prior to HNewImage, and restore the old value afterwards, if you would like to prevent an initialization in any case.

HNewImagePtr does not allocate memory for the image data, but inserts the pointer data in the Himage structure image instead. For this routine the initialization of the image data is controlled via the parameter initImg. Note, that you will encounter program crashes during deallocation of image objects, if you insert a memory block that has not been allocated via HALloc in the underlying Himage structure (see e.g., HPutImage in section 5.4). However, in specific situations (for example while doing a frame grabber integration) it might be desirable to use existing buffers allocated by some other procedures. In that case you have to provide your own deallocation routine UserClearProcedure using DBSetClearImage for the image data when creating a HALCON image object in a supply procedure:

```c
HPutImage(proc_handle, image,FALSE,&image_key);
HckP(DBSetClearImage( proc_handle,image_key,
    (DBFreeImageProc)UserClearProcedure));
```

where DBFreeImageProc is defined as

```c
typedef void (*DBFreeImageProc)(void *mem);
```

that means, the deallocation routine has got one parameter of type void* and no return value. If you pass a NULL pointer instead of UserClearProcedure the image data will not be deallocated

\(^1\)So HNewImage does not allocate memory for the Himage structure itself.
3.2. MEMORY MANAGEMENT

Names

HAlloc, HRealloc, HFree, HNewImage, HNewImagePtr, HAllocXLDCont, HFreeXLDCont

Synopsis

#include "Halcon.h"

Error HAlloc( Hproc_handle proc_handle, size_t size, void **pointer);

Error HRealloc( Hproc_handle proc_handle, void *pointer, size_t size, void **new_pointer);

Error HFree( Hproc_handle proc_handle, void *pointer);

Error HNewImage( Hproc_handle proc_handle, Himage *image, INT kind, INT width, INT height);

Error HNewImagePtr( Hproc_handle proc_handle, Himage *image, INT kind, INT width, INT height, void *data, HB00L initImg);

Error HAllocXLDCont( Hproc_handle proc_handle, Hcont **cont, size_t num_points);

Error HFreeXLDCont( Hproc_handle proc_handle, Hcont *cont);

Figure 3.3: General memory management within HALCON.

during the deallocation of the HALCON image object containing image. Note, that this might lead to a memory leak. Note further, that inserting image data in HALCON image objects using HNewImagePtr can cause undesirable side effects: By overwriting the underlying data you
might implicitly change one or more HALCON objects. Be aware that you have to take care about such problems, if you decide not to use the standard HALCON memory management (that is HNewImage).

For both HNewImage and HNewImagePtr the pixel type of the raw data within image is specified by the parameter kind. Please see Fig. 4.2 on page 50 for the supported values.

HALlocXLDCnt allocates an XLD contour\(^2\) of type Hcont (see section 4.3) including memory for num_points contour points. Internally this routine is based on HAlloc. HFreeXLDCnt is used to deallocate a contour again (including all the points and all attributes defined for cont, see page 53).

### Names

HALlocRL, HALlocRLNum, HReallocRLNum, HFreeRL

### Synopsis

```c
#include "Halcon.h"

Error HALlocRL(Hproc_handle proc_handle, Hrlregion **region)

Error HALlocRLNum(Hproc_handle proc_handle, Hrlregion **region, size_t len)

Error HReallocRLNum(Hproc_handle proc_handle, Hrlregion *region, size_t len, Hrlregion **new_region)

Error HFreeRL(Hproc_handle proc_handle, Hrlregion *region)
```

Figure 3.4: General memory management for region data.

The convenience routines HALlocRL, HALlocRLNum, HReallocRLNum, and HFreeRL (see Fig. 3.4) are used for handling permanent region data based on HAlloc. Otherwise their behaviour is similar to HALlocRLtmp etc., see page 40. Note that it is possible to change the size of regions (i.e., the number of chords) using HReallocRLNum.

\(^2\)In contrast to HNewImage and HNewImagePtr HALlocXLDCnt also allocates memory for the structure Hcont itself, not only for the data inside that structure.
### 3.2.3 Debugging

The HALCON memory management provides debugging mechanisms. Every time a memory block is deallocated a couple of consistency checks is performed automatically. Moreover, the consistency of memory blocks can be checked for debugging reasons at any time using specific Extension Package Interface routines, see Fig. 3.5.

#### Names

- HTestMem, HTestPtr, HTestAllTmp, HTestTmp

#### Synopsis

```c
#include "Halcon.h"

Error HTestMem( void)
Error HTestPtr( void *pointer)
Error HTestAllTmp( HProc_handle proc_handle)
Error HTestTmp( HProc_handle proc_handle,
                 void *pointer)
```

Figure 3.5: HALCON memory management: Check of consistency.

HTestMem checks all the memory allocated with HALloc or HALlocLocal, whereas HTestPtr checks only the specified memory block. Similar to that HTestAllTmp checks all the memory allocated via HALlocTmp and HTestTmp checks the specified block. With any failure of the consistency check, the routines return an error number (H_ERR_ICM – inconsistent memory). Furthermore, if the global variable HDoLowError is set to TRUE, they display a description of the error either on stderr (UNIX) or within an alert box (Windows NT/2000). This variable can be set in an application with the operator set system("do_low_error","true"/"false") or used directly inside the newly written operator.

### 3.3 Structuring Programs

HALCON operators are typically implemented by at least two routines: one procedure (supply procedure) receives/unpacks all input data and checks its consistency. Afterwards it calls the action procedure that performs the image processing. In some cases it might be convenient to implement several action procedures dependent on special parameter values or pixel types of image data. Generally, these steps are performed within a loop over all input objects that may include further loops over different parameter values and over all image channels. The results are collected and finally returned to the HALCON interface to the application.
Supply procedures are of type `Herror` and have only one parameter (a handle of type `HprocHandle` for instances of HALCON operators or HALCON threads). Do not forget to return an appropriate value at the end of every supply procedure (standard value: `H_MSG_TRUE`) and every action procedure (standard value `H_MSG_OK`).

Typically, action procedures perform the image processing itself. They should return an error code of type `Herror` as well. It is convenient to implement an action procedure for each pixel type of image data.

### 3.4 Name Conventions for Procedures

To ease the interpretation of program code, HALCON introduces some conventions for procedure names. The most important ones are summarized in the following section.

The names of action procedures typically begin with “IP” (image processing procedure), “IO” (input/output procedure), or “DB” (database procedure). The rest of the name describes the task performed by the procedure (in English language, beginning with a capital letter). In case of filters an additional token might be inserted between these two parts of the name encoding the pixel type(s) of images to be processed, e.g., “B” (HBYTE), “I4” (INT4) or “F” (float).

Examples:

- `IPBLowpas`
- `IPI4Threshold`
- `IODispRegion`
- `DBGetTuple`

Basic routines performing a small task that might be used by many other routines should start with “H” (for Help). The leading “H” should be followed by a string encoding the data the routine works on, e.g., “RL” for an auxiliary routine processing region data, or “XLD” for routines working on XLDs.

Examples:

- `HRLUnion`
- `HXLDContLen`
- `HXLDContRegress`

Names of supply procedures start with “C” (this stands for Core). The rest is determined by the name of the underlying action procedure (if there is only one). If there are several action procedures with names differing only by the symbols for different pixel types, the pixel type is left out in the name of the supply procedure.

Examples:

- `CIPLowpas`
- `CIPThreshold`
- `CIOOpenWindow`

### 3.5 Input / Output

Except explicit IO routines HALCON procedures should not contain any input/output commands, especially no `print` commands. Any interaction should be done either via parameters
or via Extension Package Interface in-/output procedures. Error messages should be encoded by proper error numbers returned by the procedure (see next section).

### 3.6 Error Handling

Error messages are represented by integer constants, the error numbers of type `Error`. If an error occurs, the corresponding error number should be returned by the procedure.

The file `%HALCONROOT%\include\Hconst.h` contains predefined error messages for all typical errors, see also appendix A. But it may happen that none of them fits an error occurring in a new, user-defined operator. So the user is allowed to define new numbers that should be greater than 10,000 to avoid re-using an already allocated number or a number reserved for future extensions of the HALCON system. If the user-defined operator returns a user-defined error, the error will be reported as

```
No error message available for this error code (<error number>).
```

In order to provide user-defined errors with error messages, the function

```
H_error HSetErrText(char *error_text)
```

can be called immediately before the operator returns the error code with `return H_ERR_*`. In this case the error message `error_text` will be displayed by the `next` call of the HALCON operator `get_err_text`.

Error numbers are specified via `#define` and are called `H_ERR_XYZ`.

Examples:
```
H_ERR_WIPN1 /* wrong number of values in input control */
   /* parameter 1 */
H_ERR_WIPV3 /* wrong parameter value (control param. 3) */
H_ERR_WIPT2 /* wrong type of values (control param. 2) */
H_ERR_WION1 /* wrong number of objects in input object */
   /* parameter 1 */
H_ERR_WIT /* wrong image type */
H_MSG_FAIL /* procedure failed */
H_MSG_TRUE /* no error (supply procedure) */
H_MSG_OK /* no error (any other procedure) */
```

All HALCON procedures return an error code. So this code has to be checked after calling any HALCON procedure. This is done by using the following macro to call a procedure:

```
HchkP()
```

It checks the return value and exits the actual procedure if an error occurred (a detailed description of the macro is given in the section 5.6.1). Please note, that some of the Extension Package Interface macros described in chapter 5 and 6 (e.g. `HGetCPar`) perform similar tests automatically and return a proper error code implicitly.

---

3 Within this call the error text is reset to "No error message available ..." again to ensure that the user-defined error text is up to date.
3.7 Notes on Image Processing Operators

- HALCON images are variable in their size (format). Therefore, no constants should be used for image formats (except MAX_FORMAT for the maximum format). The actual format of an image can be extracted from the structure HiImage (cf. Fig. 4.1).

- Notice that more than one pixel type may occur, when performing gray value operations. An operator should be able to work on all of them, see also the discussion in section 6.1.3. If an operator can only process a reduced set of types, this can be specified in the def-file with type_list (cf. page 31). An operator should reject all images that it can not process by returning H_ERR_WIT – wrong image type.

- In many cases an operator might receive more than one iconic object as input, i.e. the supply procedure must contain a loop over all iconic objects (e.g., using HAllReg and HAllSegm described in section 6.1 or HAllObj in section 5.3.1). This is true for regions as well as images or XLDs. Furthermore, images may contain more than one channel. So a second loop over all channels (components) within an image is needed (see HAllComp, section 5.3.2).

- All procedures should only work within the image’s area of definition, especially when performing feature extraction, filter or segmentation tasks.
Chapter 4

HALCON Data Types

The HALCON Extension Package Interface provides special data types to handle iconic objects. All iconic objects in the application layer are represented by a key referring to the HALCON data base. The internal structure is hidden from the HALCON user. However, using the Extension Package Interface it is possible to work directly on the data structures for images, regions, and XLDs. Special macros can be used to access the internal representation of such an object. Furthermore, data structures for control parameters are provided. All data types described in the following sections are defined in the files `%HALCONROOT%/include\IPtype.h` and `%HALCONROOT%/include\HBase.h`.

4.1 Pixel Data (Himage)

Gray value images are represented by a rectangular image matrix. Several matrices (called components) can be combined to a multi-channel image. Each channel can be accessed separately, i.e. the components are not interleaved. The structure Himage contains size, pixel type, and pixel data of one component. Furthermore, a time-stamp is included. Fig. 4.1 shows the corresponding type declaration.

The maximum for width and height (MAX_FORMAT) is currently set to 32768. The origin of an image matrix is at position (0, 0). Row coordinates range from 0 to height - 1, column coordinates from 0 to width - 1. The image pointer (HPixelImage) refers to the first pixel of the matrix (index: 0,0). Different pixel types are supported. They are distinguished by a selector (kind), see Fig. 4.2. The pixel types include basic types like BYTE, INT1, INT4, and float as well as composed types. Their definitions can be seen in Fig. 4.3.

The pixel data is stored as a one-dimensional array (vector) of one of the pixel types. The indices of the array range from 0 to width*height-1. A linear coordinate L within an image is derived from the row index R and column index C as follows:

\[ L = R \times \text{image.width} + C \]

This transformation is performed by the following macro:

\[ L = \text{HLinCoor}(R,C,\text{image.width}); \]
typedef union {
    HBYTE *b;    /* 0..255 (BYTE_IMAGE) */
    HBYTE *z;    /* 0..255 mod 256 (CYCLIC_IMAGE) */
    HBYTE *d;    /* orientation 0..180 (DIR_IMAGE) */
    INT1 *i;     /* -127..126 (INT1_IMAGE) */
    INT4 *l;     /* 4 byte integer (LONG_IMAGE) */
    float *f;    /* 4 byte real (FLOAT_IMAGE) */
    HXYPixel xy; /* displacement vector field */
    HComplexPixel c; /* complex image (COMPLEX_IMAGE) */
    HLutPixel lut; /* byte with LUT (LUT_IMAGE) */
    HInt2Pixel s; /* 2 bytes with sign (INT2_IMAGE) */
} HPixelImage;

define struct {
    INT kind;       /* pixel type */
    HPixelImage pixel; /* pixel data */
    INT width;      /* image width */
    INT height;     /* image height */
    /* time of creation of image */
    UINT2 msec;     /* milliseconds 0..999 */
    UINT1 sec;      /* seconds 0..59 */
    UINT1 min;      /* minutes 0.59 */
    UINT1 hour;     /* 0..23 */
    UINT1 day;      /* 1..31 */
    UINT2 yday;     /* 1..366 */
    UINT1 mon;      /* 1..12 */
    UINT2 year;     /* starting at 1900 */
} Himage;

#define BYTE_IMAGE (INT)1 /* 1 byte per pixel (0..255) */
#define LONG_IMAGE (INT)2 /* 4 byte per pixel (INT4) */
#define FLOAT_IMAGE (INT)4 /* 4 byte per pixel (float) */
#define DIR_IMAGE (INT)8 /* edge orientation 0..180 */
#define CYCLIC_IMAGE (INT)16 /* 0..255 cyclic */
#define INT1_IMAGE (INT)32 /* -127..126 */
#define XY_IMAGE (INT)64 /* 2 byte images with sign */
#define COMPLEX_IMAGE (INT)128 /* 2 float-images */
#define LUT_IMAGE (INT)256 /* (0..255) with color table */
#define INT2_IMAGE (INT)512 /* 2 bytes with sign */

There are two additional macros for computing rows and columns:

\[
R = \text{HRow}(L, \text{image.width});
C = \text{HCol}(L, \text{image.width});
\]
4.2 Region Data (Hrlregion)

In HALCON region data is represented by a special variant of the runlength encoding – a chord encoding: For every line (chord) of a region its row index (“y coordinate”, “line number”) and the column index (“x coordinate”) of its start and end point is stored. Both the start point and the end point belong to the region.

Chord data must fulfill the following conditions:

- a chord is limited to one row
- chords may not overlap
- chords are sorted in ascending order

If a region is read from the HALCON data base, its representation fulfills all the conditions above. When writing a region, condition 2 and 3 need not to be sufficed in all cases, as the Extension Package Interface automatically modifies the data (at cost of computation time).

Fig. 4.4 shows the type declaration for chords. All chords are stored in an array of the type Hrun, where num is the current and num_max the maximum allowed number of chords (depending on the size of the region specified at its creation, see also sections 3.2). The flag is_compl allows an
typedef struct {
    INT2 l;        /* line number (row) of chord */
    INT2 cb;       /* column index of beginning of chord */
    INT2 ce;       /* column index of ending of chord */
} Hrun;

typedef struct {
    HB00L is_compl;  /* region is complement */
    INT4 num;       /* number of chords */
    INT4 num_max;   /* maximal number of chords */
    HRegFeature feature;  /* already processed features */
    Hrun   rl[RL_LENGTH];  /* array of chords */
} Hr1region;

Figure 4.4: Data type Hr1region for region encoding.

easy transformation of a region in its complement. Operators that work on regions must consider
this flag and react according to its value. The structure HRegFeature (see Fig. 4.5) contains
all features extracted so far to avoid repeating a computation. HFeatureFlags encodes, which
features already have been extracted. Do not forget to reset these flags if you modify a region.

Normally, a variable of type Hr1region is allocated with the procedure HA1locRLtmp or
HA1locRLNumTmp, see section 3.2. These routines initialize the data, especially num_max that
is needed for tests of overflow. When allocating a variable “by hand”, the programmer must
provide a suitable initialization by himself/herself.

Whereas all coordinates within runlength codes are stored as (row,column), linear coordinates
are used to address HALCON image matrices. Thus, the macros CE and CE are very helpful,
especially when processing gray values (in linear coordinates) along a chord. The programs on
page 93 and 95 illustrate how to use them. Their functionality can be seen in Fig. 4.6. The first
parameter (rl) contains a pointer to the chords as it is used within Hr1region. index specifies
the index of the chord to work on and width contains the image width. CE returns the linear
coordinate of the start point and CE that of the end point of the chord.

4.3 XLDs (Hcont, Hpoly)

XLDs (eXtended L ine D escriptions) are specific iconic HALCON objects to represent sub-
pixel accurate contours and polygons. The corresponding data types are listed in Fig. 4.7 and
Fig. 4.10.

HALCON contours of type Hcont contain an array of subpixel points along a contour. A con-
tour might be classified concerning to topological considerations (cont_class). Additionally,
an arbitrary number of additional attributes for each point along the contour can be included
(attrs), e.g., the edge amplitude of subpixel edge points or the orientation of the local gra-
dient. Finally, an arbitrary number of additional global attributes, i.e., attributes valid for the
entire contour, can be included (global), e.g., the individual parameters of a regression line to
the contour or the parameters of an ellipse segment fitted to the contour.
4.3. XLDS ('HCONT', 'HPOLY')

typedef struct {
union {
    HFeatureFlags single;    /* a bitfield */
    long all;
} def;
UINT1 shape;             /* SHAPE */
HB00L is_convex;
HB00L is_filled;
HB00L is_connected4, is_connected8;
HB00L is_thin;
float circularity;
float compactness;
float contlength;
float convexity;
float phi;
float ra, rb;
float ra_, rb_;    
float anisometry, bulkiness, structure_faktor;
float m11, m20, m02, ia, ib;
float row, col;
INT4 area;
INT2 row1,col1,row2,col2;
float row_rect, col_rect, phi_rect, length1, length2;
float row_circle, col_circle, radius;
INT2 min_chord, max_chord;
INT2 min_chord_gap, max_chord_gap;
} HRegFeature;

Figure 4.5: Data type HRegFeature for region features.

Please see section 3.2.2 for routines to allocate/deallocate contours. Two important auxiliary routines to copy contours are listed in Fig. 4.8. HCopyXLDCnt copies a contour including all attributes, whereas HCopyXLDCntPart copies only a part of the original contours specified by two indices min_index and max_index. They refer to the first and the last point along the contour to be copied (starting with index 0). For closed contours a negative value might be specified for min_index. This will lead to a “wrap-around”, that is the part of the contour to be copied starts at cont_in->num + min_index. Both HCopyXLDCnt and HCopyXLDCntPart allocate the output contour themselves. So just pass a pointer to Hcont without allocating any memory for cont_out.

The routines provided for handling attributes of contour points are summarized in Fig. 4.9.

HAddXLDCntAttrib is used to add a new class of attributes to a given contour cont. Note, that this routine allocates memory for the attribute values, but it does not set the values themselves. An arbitrary name for the attribute can be specified by the parameter name. The index of the new attribute within the attribs array in cont is returned in index and can be used to address the values by cont->attribs[index].val[xxx]. The same holds for global attributes: HAddXLDCntGlobalAttrib is used to add new global attributes, which can be accessed by cont->global[index].val. Note that these values can be accessed within the HALCON system (that is on the application layer) using the operators get_contour_attrib_xld and

HALCON 6.0
Names

CB, CE

Synopsis

```c
#include "HALCON.h"

INT4_8 CB(Hrun, *rl, INT4_8 index, INT width);
INT4_8 CE(Hrun, *rl, INT4_8 index, INT width);
```

Figure 4.6: Linear coordinates from Hrlregion.

get_contour_global_attrib_xld. See the Reference Manuals for details. Within HALCON, a specific attribute of a contour can be accessed using HLookUpXLDContAttrib, while a specific global attribute can be accessed using HLookUpXLDCGlobalAttrib. These routines return the index of the desired attribute within attribs or global of cont, or the error code H_ERR-xlrd-cand as result of the procedure call, if no attribute with the specified name is defined.

The XLD data type Hpoly, displayed in Figure 4.10, encodes subpixel accurate polygons. Basically it contains an array of control points. In many applications such polygons are derived from contours. Thus, the data structure can also hold a reference to the underlying part of a contour specified by a HALCON data base key.

### 4.4 Control Parameters (Hpar, Hcpar)

The HALCON data types Hpar and Hcpar are used within the Extension Package Interface to pass control parameters to supply procedures. Fig. 4.11 shows the corresponding definitions. Hpar encodes a single parameter value of one of the basic types integer, real or string, see section 2.3. In addition to that, Hcpar also encodes the corresponding type. Thus, it is possible to combine different types within an array of Hcpar values. The selector type can be set to LONG_PAR, FLOAT_PAR or STRING_PAR.

The Extension Package Interface procedures HGetCPar, HGetSPar, HGetPPar, HPutCPar, and HPutPPar (section 5.5) are based on Hcpar. They are used to access control parameters of HALCON operators.
typedef enum cont_class {
    cont_unknown,    /* unknown */
    cont_no_junc,    /* neither start nor end point points */
    /* are junctions */
    cont_start_junc, /* start point is a junction */
    cont_end_junc,   /* start point is a junction */
    cont_both_junc,  /* both start and end point are junctions */
    cont_closed      /* closed contour */
} Hcont_class;

typedef struct cont_attrib {
    char *name;    /* name of the attribute */
    float *val;    /* value of the attribute (per point) */
} Hcont_attrib;

typedef struct cont_global_attrib {
    char *name;    /* name of the global attribute */
    float val;     /* value of the attribute (per contour) */
} Hcont_global_attrib;

typedef struct cont_type {
    INT4 num;      /* number of points along the contour */
    float *row;    /* points / row indices (y coordinates) */
    float *col;    /* points / column indices (x coord.) */
    Hcont_class cont_class;    /* contour class */
    INT4 num_attr;  /* number of additional attributes */
    Hcont_attrib *attribs;    /* additional attributes (for each point) */
    INT4 num_global;  /* number of additional global attributes */
    Hcont_global_attrib *global;    /* additional attributes (per contour) */
    INT4 h;          /* auxiliary (temporary) */
} Hcont;

Figure 4.7: The XLD data type Hcont for subpixel contours.
Names

HCopyXLDCont, HCopyXLDContPart

Synopsis

#include "Halcon.h"

Herror HCopyXLDCont(Hproc_handle proc_handle, Hcont *cont_in, Hcont **cont_out)

Herror HCopyXLDContPart(Hproc_handle proc_handle, Hcont *cont_in, INT4_8 min_index, INT4_8 max_index, Hcont **cont_out)

Figure 4.8: Auxiliary routines for contours of type Hcont.

Names

HAddXLDContAttrib, HLookupXLDContAttrib, HAddXLDContGlobalAttrib, HLookupXLDContGlobalAttrib

Synopsis

#include "Halcon.h"

HAddXLDContAttrib(Hproc_handle proc_handle, Hcont *cont, char *name, INT4 *index)

HLookupXLDContAttrib(Hcont *cont, char *name, INT4 *index)

HAddXLDContGlobalAttrib(Hproc_handle proc_handle, Hcont *cont, char *name, INT4 *index);

HLookupXLDContGlobalAttrib(Hcont *cont, char *name, INT4 *index);

Figure 4.9: Handling attributes of contour pixels (within Hcont).
typedef struct lin_seg_type {
    float row, col; /* a control point of the polygon */
    /* row (y) and column (x) coordinate */
    float length; /* length of the line from the */
    /* current to the next point */
    float phi; /* orientation (rad) of this line */
    Hkey ref; /* data base key of the underlying */
    /* contour */
    INT4 first; /* index of the first point of the */
    /* underlying contour belonging to */
    /* current side of the polygon */
    INT4 last; /* index of the last contour point */
} Hline_seg;

typedef struct poly_type {
    INT4 num_line; /* number of lines */
    /* maximum number of lines (size */
    /* of the array lines) */
    Hline_seg *lines; /* control points of the polygon */
} Hpoly;

Figure 4.10: The XLD data type Hpoly for subpixel polygons.

typedef union {
    INT4 l; /* 4 byte integer */
    float f; /* 4 byte real */
    char *s; /* string */
} Hpar;

typedef struct {
    Hpar par;
    INT1 type;
} Hcpar;

Figure 4.11: Data types Hpar and Hcpar for control parameters.
Chapter 5

Handling Iconic Objects and Control Parameters

The HALCON Extension Package Interface provides a large set of procedures and macros for handling control parameters and iconic objects. It supports tasks like:

- Accessing iconic input objects in the HALCON data base.
- Accessing single iconic objects within a tuple of objects.
- Accessing specific components (regions, gray value channels) of image objects.
- Accessing XLDs.
- Creating iconic output objects in the HALCON data base based on the computed results.
- Reading input control parameters.
- Writing output control parameters.

The routines of the Extension Package Interface described in this and the following chapter especially facilitate the programming of support procedures (that is the access of the HALCON data base and all the parameter handling).

In this chapter the “low level” interface routines are introduced. They allow a straightforward access to all parameters of an operator – to iconic objects (and their components) and to control parameters. Moreover, some routines for creating objects and writing output object parameters and control parameters are introduced.

Based on these low level routines a set of convenience routines are presented in chapter 6. They are designed to further facilitate the programming of very typical support procedures.

5.1 Basic Access to Iconic Input Objects

This section introduces some basic routines for accessing iconic objects (regions, gray value channels, XLDs), see Fig. 5.1. They form a basic interface to the HALCON data base. Fig. 5.2 illustrates how they are used to read image and region data from input object parameters.
Names

HGetObj, HGetComp, HGetRL, HGetImage, HGetXLD

Synopsis

```c
#include "Halcon.h"

HGetObj( proc_handle, par_num,
       INT4 obj_num
       Hkey *obj_key)

HGetComp( proc_handle, obj_key,
          Hkey comp
          INT *comp_key)

HGetRL( proc_handle, region_key,
        Hkey *region)

HGetImage( proc_handle, image_key,
            Himg *image)

HGetXLD( proc_handle, obj_key,
         INT xld_type,
        XLD **xld)
```

Figure 5.1: Basic routines for accessing iconic input objects.

5.1.1 HGetObj

HGetObj (see Fig. 5.1) returns the data base key (type: Hkey) of an iconic object corresponding to the input object parameter number par_num of the HALCON operator.

The iconic object can either be a region (object that contains only a region component), an image object (object that contains a region component and one or more image channels), or an XLD (object that contains a contour or a polygon). For the sake of simplicity we will refer to all of them using the term object.

The parameters of a HALCON operator are numbered consecutively from 1 to n (not from 0 to n – 1) for each parameter class (input/output object/control). The parameter par_num of HGetObj refers to this number, thus specifying the desired input object parameter. All objects passed within a single input object parameter are numbered from 1 to m, with the first object in the list having the number 1. The parameter obj_num of HGetObj denotes the index of a desired object within this list (1 ≤ obj_num ≤ m).
5.1. BASIC ACCESS TO ICONIC INPUT OBJECTS

Figure 5.2: Direct access to input image objects.

```
INT2 inp_pars;
INT4 num_objs;
Hkey key;

/* get number of input object parameters: */
/* note: This number is known in general - it is specified by */
/* the operator header in the corresponding def-file of the */
/* operator */
HReadGV(proc_handle,HGinp_obj,&inp_pars);
  for (p=1; p<=inp_pars; p++) {
    /* get number of input objects per input obj. parameter: */
    HReadGVA(proc_handle,HGnum_obj,&num_objs,p);
      for (o=1; o<=num_objs; o++) {
        HGetObject(proc_handle,p,o,&key); /* get key of object */
        /* further processing ... */
      }
  }
```

Figure 5.3: Example for HGetObject: Data base keys of all iconic objects of all input object parameters.

The number of input object parameters for an operator is specified in the corresponding def-file\(^1\), but the number of objects within an input object parameter is dynamic. This value is accessible via the Extension Package Interface routine HReadGVA:

```
HReadGVA(proc_handle,HGnum_obj,&num_objs,p);
```

where \( p \) denotes the parameter number and \( \text{num\_objs} \) is the desired number of objects. There

\(^1\)So this number is actually known by the programmer. However, using HReadGV it can be read from the operator context as well, see Fig. 5.3.
is an alternative for reading the number of objects of an input object parameter, which does not need the HReadGVA routine, namely HGetObjNum (see page 67).

Note, that a loop over all objects passed to an operator within an input object parameter is a very common task. Therefore, HALCON provides a macro for this problem: See HA110bj on page 69.

### 5.1.2 HGetComp

HGetComp (see Fig. 5.1) returns the data base key of a *component* (image matrix: `image_key` or region: `region_key`) of an image object that is stored under the key `obj_key` in the HALCON data base.

```c
INT2 obj_channels;
Hkey obj_key;
Hkey region_key;
Hkey image_key;
...

/* get key of object: */
HGetObj(proc_handle,p,o,&obj_key);
/* get key of region: */
HGetComp(proc_handle,obj_key,REGION,&region_key);
/* get number of channels: */
HckP(HNNumOfChannels(proc_handle,p,o,&obj_channels));
for (i=1; i<=obj_channels; i++) {
    /* get image matrix key: */
    HGetComp(proc_handle,obj_key,i,&image_key);
    /* ...further processing */
}
```

Figure 5.4: Example for HGetComp: Data base keys of all regions and channels within an input image object.

All components of an object are consecutively numbered from 0 to \( n \), with 0 denoting the region and 1...\( n \) denoting the channels (image matrices). To get a better legibility of program code, the constants `REGION` (= 0), `IMAGE_INDEX` (= 1), `IMAGE1` (= 1), `IMAGE2` (= 2) etc. have been defined globally and may be used as parameter values.

The number of channels per image object `obj_num` of parameter `par_num` is accessible via `HNNumOfChannels`, see Fig. 5.4 and 5.5:

```c
HNNumOfChannels(proc_handle,par_num,obj_num,&obj_channels)
```

See also `HNNumOfChannels` in section 5.6.4 for a convenience version of this routine when dealing with the first input object parameter. More examples of how to use HGetComp can be found in Fig. 5.6 and 5.7.
5.1.3 HGetRL

HGetRL (see Fig. 5.1) reads the chord encoding of a region (type: Hrlregion, see section 4.2) denoted by the data base key region_key from the HALCON data base.

```c
Hrlregion  *region;
Hkey       obj_key;
Hkey       region_key;
...
HckP(HAllocRLTmp(proc_handle,&region));
HGetObj(proc_handle,p,0,&obj_key);
HGetComp(proc_handle,obj_key,REGION,&region_key);
HGetRL(proc_handle,region_key,region);
/*! processing (in general this should be done calling an action proc.) */
area = 0;
for (i=0; i<region->num; i++)
  area += region->rl[i].ce - region->rl[i].cb + 1;
HckP(HFreeRLTmp(proc_handle,region));
```

Figure 5.6: Example for HGetRL: Calculate the area of a region.

Since the region data is copied to region, it might be overwritten with new values (in contrast to HGetImage and HGetFDRL, cf. section 5.1.4 and 5.2.2). However, this means that enough memory for region must be allocated before calling HGetRL, see section 3.2.1. Fig. 5.6 and 5.7 show exemplary applications of HGetRL.

5.1.4 HGetImage

HGetImage (see Fig. 5.1) reads the image data (type: Himage, see section 4.1) of a specific gray value component (channel) of an image object referenced by its key in the HALCON data base.

The data structure Himage contains the gray values, the gray value type, and the size of the image matrix. Instead of copying the image matrix, HGetImage only returns a pointer to the
Figure 5.7: Example for HGetImage: Average gray value of first channel.

raw data in image.pixel. This is much more efficient, but means that only read access to the image matrix is recommended (otherwise you will encounter unpredictable side effects). Fig. 5.7 shows an application of HGetImage.

5.1.5 HGetXLD

HGetXLD (see Fig. 5.1) is used to access XLD objects in the HALCON data base. The object of interest is specified by the data base key obj_key. The parameter xld_type allows to specify the kind of XLD data to be accessed: It has to be set to XLD_CONTOUR_ID in case of a contour and to XLD_POLYGON_ID in case of a polygon. Corresponding to the selected type HGetXLD expects a pointer to Hcont* or a pointer to Hpoly* in the parameter xld. In both cases not the underlying data within the structures Hcont or Hpoly but only the pointers to the data are copied from the data base. Thus, please avoid any write access to this data. Otherwise you will encounter unpredictable side effects (changing other HALCON objects). Fig. 5.8 shows a simple example for how to use HGetXLD.
5.2. ADDITIONAL ROUTINES FOR ACCESSING INPUT IMAGE OBJECTS

Figure 5.8: Example for HGetXLD: Return the number of points of the contour with number obj_num within input object parameter par_num.

Names

HGetDRL, HGetFDRL, HGetURL, HGetDImage, HGetObjNum

Synopsis

#include "Halcon.h"

HGetDRL(Hproc_handle, Hkey, HrIregion, proc_handle, obj_key, *region)

HGetFDRL(Hproc_handle, Hkey, HrIregion, proc_handle, obj_key, **region)

HGetURL(Hproc_handle, INT, HrIregion, proc_handle, par_num, *region)

HGetDImage(Hproc_handle, Hkey, INT, Himage, proc_handle, obj_key, channel, *image)

HGetObjNum(Hproc_handle, INT, INT4_8, proc_handle, par_num, *num)

Figure 5.9: Additional routines for accessing input image objects.
5.2 Additional Routines for Accessing Input Image Objects

This section introduces some routines that ease the programming of supply procedures in many applications, see Fig. 5.9. Basically they are combinations of the routines described in the previous section. These additional routines only require the data base keys of the input objects that can be extracted by using the routines HGetObj (see section 5.1.1) or HALlObj (see section 5.3.1). For HGetURL even this step can be omitted.

5.2.1 HGetDRL

HGetDRL (see Fig. 5.9) combines HGetComp (section 5.1.2) and HGetRL (section 5.1.3). It reads the chord encoding of a region (type: HrIregion, see section 4.2) specified by the data base key obj_key of an image object. The region data is copied to region that must have been allocated before with a suitable size (e.g., by using HAllocRLTmp, see section 3.2.1). Therefore, this region data can be overwritten without side effects. On the other hand, the memory must be deallocated at the end of the supply procedure (e.g., by using HFreeRLTmp). Fig. 5.10 shows an example application of HGetDRL.

```c
HrIregion *region;
Hkey obj_key;
...
Hckp(HAllocRLTmp(proc_handle,&region));
HGetObj(proc_handle,par_num,obj_num,&obj_key);
HGetDRL(proc_handle,obj_key,region);
/* * processing (in general this should be done calling an action proc.) */
area = 0;
for (i=0; i<region->num; i++)
    area += region->rl[i].ce - region->rl[i].cb + 1;
Hckp(HFreeRLTmp(proc_handle,region));
```

Figure 5.10: Application of HGetDRL.

HGetDRL (see Fig. 5.9) combines HGetComp (section 5.1.2) and HGetRL (section 5.1.3). It reads the chord encoding of a region (type: HrIregion, see section 4.2) specified by the data base key obj_key of an image object. The region data is copied to region that must have been allocated before with a suitable size (e.g., by using HAllocRLTmp, see section 3.2.1). Therefore, this region data can be overwritten without side effects. On the other hand, the memory must be deallocated at the end of the supply procedure (e.g., by using HFreeRLTmp). Fig. 5.10 shows an example application of HGetDRL.

5.2.2 HGetFDRL

In contrast to HGetDRL only a pointer to the region data is returned by HGetFDRL (see Fig. 5.9). So there is no need to allocate memory for the data. On the other hand, only read access to the data is allowed. An example application of HGetFDRL can be seen in Fig. 5.11.

5.2.3 HGetURL

HGetURL (see Fig. 5.9) reads all regions passed in the input object parameter with number par_num, computes the union of all these regions, and returns the resulting region in region. The HrIregion data referenced by region must have been allocated before with a suitable size.
5.2. ADDITIONAL ROUTINES FOR ACCESSING INPUT IMAGE OBJECTS

(e.g., by using HAallocRLTmp, see section 3.2.1). Therefore, this region data can be overwritten without side effects. On the other hand, the memory must be deallocated at the end of the supply procedure (e.g., by using HFreeRLTmp). Fig. 5.12 shows an application of HGetURL. Note, that the area calculated in the example may differ from the sum of the areas of all single regions, because the single regions may overlap.

5.2.4 HGetDImage

HGetDImage (see Fig. 5.9) combines HGetComp (section 5.1.2) and HGetImage (section 5.1.4). It reads the data of the gray value component channel of the image object specified by the data base key obj_key and returns it in the Himage structure (image).

For the sake of efficiency, only a pointer to the image matrix is copied to image, instead of copying the data itself. So, only read access to the image data is recommended in order to avoid unpredictable side effects. Fig. 5.13 shows an application of HGetDImage.

5.2.5 HGetObjNum

HGetObjNum (see Fig. 5.9) returns the number of iconic objects which are stored in the input object parameter denoted by its index par_num (from 1 to n). It is an alternative to the usage of HReadGVA (compare page 61). By using HGetObjNum, the example of figure 5.3 (page 61) looks as follows (see figure 5.14).
Himage  image;
Hcpar   row,col,gray;
...
HGetSPar(proc_handle,1,LONG_PAR,&row,1);  /* row */
HGetSPar(proc_handle,2,LONG_PAR,&col,1);  /* column */
coord = HLinCoor(row.par.1,col.par.1,image.width);
HGetDImage(proc_handle,obj_key,IMAGE_INDEX,&image);
/* processing (in general this should be done calling an action proc.) */
switch (image.kind) {
   case BYTE_IMAGE:
      gray.par.l = image.pixel.b[coord];
      gray.type = LONG_PAR;
      break;
   case FLOAT_IMAGE:
      gray.par.f = image.pixel.f[coord];
      gray.type = FLOAT_PAR;
      break;
   default:
      return(H_ERR_WIT);  /* wrong image type */
}
HPutCPar(proc_handle,1,&gray,1);

Figure 5.13: Application of HGetDImage: Return the gray value at position (row,col) in the first channel of the input object obj_key.

INT2   inp_pars;
INT4_8  num_objs;
Hkey    key;
INT     p,o;
...
/* get number of input object parameters: */
/* note: This number is known in general - it is specified by */
/* the operator header in the corresponding def-file of the */
/* operator */
HReadGV(proc_handle,HGinp_obj,&inp_pars);
for (p=1; p<=inp_pars; p++) {
   /* get number of input objects per input obj. parameter: */
   HGetObjectNum(proc_handle,p,&num_objs);
   for (o=1; o<=num_objs; o++) {
      HGetObject(proc_handle,p,o,key);  /* get key of object */
      /* further processing ... */
   }
}

Figure 5.14: Application of HGetObjectNum: Get the number of iconic objects of an input object parameter.
5.3 Loop Macros for Accessing Single Input Objects

This section describes two macros that ease accessing all iconic objects and image components within an input object parameter, see Fig. 5.15. Please see also the additional loop macros in section 6.1 that further facilitate programming typical supply procedure.

**Names**

HA110bj, HA11Comp

**Synopsis**

```
#include "Halcon.h"

HA110bj(Hproc_handle proc_handle,
        INT par_num,
        Hkey &obj_key,
        INT4_8 &index)

HA11Comp(Hproc_handle proc_handle,
         Hkey obj_key,
         Hkey &image_in_key,
         Himage &image_in,
         INT4_8 &index)
```

Figure 5.15: Basic loop macros to access iconic objects. "&" denotes output parameters of the macros. This is only a special notation to make clear that these parameters are changed by the macros. So do not pass pointers to variables but the variables itself to the macro.

### 5.3.1 HA110bj

HA110bj (see Fig. 5.15) performs a loop over all iconic objects of a specified input object parameter and returns their data base keys.

```
Hkey obj_key;
Hr1region *region;
Himage image;
INT4_8 index;
...
HA110bj(proc_handle,1,obj_key,index) {
    HGetFDRL(proc_handle,obj_key,&region);
    HGetDImage(proc_handle,obj_key,IMAGE_INDEX,&image);
}
```

Figure 5.16: Application of HA110bj: Get the area of definition and the first gray value channel of all input objects of the first input object parameter.
Within the loop all iconic objects contained in the input object parameter number\(^2\) \texttt{par\_num} are accessed one by one. index is set to the number of the current iconic object to be processed and \texttt{obj\_key} is set to the corresponding data base key. Thus, using \texttt{HAllObj} is equivalent to programming an explicit loop over all objects contained in a parameter and determining the data base keys via \texttt{HGetObj}, see also Fig. 5.3. Fig. 5.16 shows an application of \texttt{HAllObj}.

### 5.3.2 \texttt{HAllComp}

\texttt{HAllComp} performs a loop over all gray value channels (components) of the image object denoted by the data base key \texttt{obj\_key}. Within the loop, \texttt{image\_in\_key} is set to the data base key of the current image object and the corresponding image data is delivered in \texttt{image\_in}. Using \texttt{HAllComp} is equivalent to programming an explicit loop over all channels of an image object using \texttt{HGetDImage}. This also implies that the raw image data, that is the image matrix itself, is \textit{not} copied. Instead, only a pointer to this data is inserted in \texttt{image\_in}. So please restrict yourself to \textit{reading} this data. Any write access to the image matrix will cause unpredictable side effects.

Note, that \texttt{HAllComp} does \textit{not} check, whether the image object contains at least one channel. Note further, that any region processing should be done outside the loop (otherwise you will process the same region again and again for each channel).

```c
Hkey obj_in;
Hkey obj_out;
Hkey image_in_key;
INT4_8 comp_index;
Himage image_in, image_out;
...
HAllObj(proc_handle, 1, obj_in, j) {
    HCrObj(proc_handle, 1, &obj_out);
    HAllComp(proc_handle, obj_in, image_in_key, image_in, comp_index) {
        HckP(HCrImage(proc_handle, image_in_key, 1, BYTE_IMAGE,
                    image_in.height, image_in.width,
                    &im_out_key, &image_out));
        HckP(IPBRot90(image_in.pixel.b, image_in.width, region_in,
                     image_out.pixel.b, width, height));
        HDefObj(proc_handle, obj_out, im_out_key, comp_index);
    }
}
```

**Figure 5.17**: Application of \texttt{HAllComp}: rotate all channels of all image objects of the first input object parameter.

Fig. 5.17 shows an example of how to use \texttt{HAllComp}. Within the \texttt{HAllObj} loop for each input image object an output image object is created and added to the object list of the first output object parameter (\texttt{HCrObj}, see section 5.4.1). After that, a loop over all gray value channels of the current image object is performed (\texttt{HAllComp}). Within this loop, for every gray value

\(^2\)So in contrast to the routines described in section 6.1, \texttt{HAllObj} allows to specify the desired parameter by its number.
channel a new image matrix is created (\texttt{HCrImage}, see section 5.4.1) and filled with the rotated input matrix (\texttt{IPBRot90}). Finally, this matrix is installed as a new gray value channel of the output image object (\texttt{HDefObj}, see section 5.4.1). The corresponding region transformation (rotating the region and defining the rotated region as new area of definition of the output image object via \texttt{HDefObj}) is omitted in this example.

### 5.4 Creating Objects and Writing Output Object Parameters

This section describes routines for creating new iconic objects in the HALCON data base and for writing output object parameters of a HALCON operator, see Fig. 5.18. Note, that regions, gray value channels, and XLDs are all stored as individual objects within the data base. Thus, for example different image objects can \textit{share} the same gray value channels (image matrices) or regions (areas of definition). Therefore, in order to return iconic objects as the result of a HALCON operator you have to

- store the computed regions, channels, and XLDs in HALCON data base objects,
- combine regions and channels to image objects,
- and add the appropriate iconic objects to the corresponding output object parameters.

#### 5.4.1 HCrObj

\texttt{HCrObj} (see Fig. 5.18) creates a new image object in the HALCON data base and adds it to the list of objects in the output object parameter with the number \texttt{par_num}. Note, that more than one image object can be returned in the same output parameter by iterating calls of \texttt{HCrObj} using the same \texttt{par_num}: Each call appends the data base key of the new object at the end of the object list for the parameter.

The new image object contains the following default components:

- The region component specifying the area of definition is set to the empty region. This component can be changed using \texttt{HPutDRL} (see section 5.4.3) or \texttt{HPutRect} (see section 6.2.2).
- All channels are marked as undefined, that is the new image object contains no gray value components. Channels can be added to the object using \texttt{HPutImage} (see section 5.4.4) and \texttt{HDefObj} (see section 5.4.5) or \texttt{HPutDImage} (see section 5.4.6).

Fig. 5.17 and 5.21 show examples for how to use \texttt{HCrObj}.
Names

HCrObj, HCopyObj, HPutDRL, HCrXLD

Synopsis

```c
#include "Halcon.h"

HCrObj( proc_handle, par_num, *obj_key)

HCopyObj( proc_handle, obj_key, par_num, *obj_key)

HPutDRL( proc_handle, obj_key, *region, *rl_key)

Herror HCrXLD( proc_handle, par_num, xld, xld_type, *used_xlds, num_used_xlds, free_proc, *obj_key)
```

Figure 5.18: Basic routines for creating new iconic objects and writing output object parameters (to be continued).

5.4.2 HCopyObj

HCopyObj (see Fig. 5.18) creates a new iconic object (image, region, or XLD) in the HALCON data base containing the same components as an already existing object specified by its data base key obj_key. The new object is appended to the list of objects in the output object parameter with the number par_num. The data base key of the new image object is returned in obj_key.

Similar to HCrObj, the components of a new image object can be reassigned using HDefObj etc. HCopyObj is especially useful, when output objects hardly differ from the corresponding input objects.
### 5.4. CREATING OBJECTS AND WRITING OUTPUT OBJECT PARAMETERS

#### Names

HPutImage, HDefObj, HPutDImage, HCrImage

#### Synopsis

```c
#include "Halcon.h"

HPutImage( Hproc_handle, proc_handle, 
           Himage, *image, 
           HB00L, copy, 
           Hkey, *obj_key)

HDefObj( Hproc_handle, proc_handle, 
         Hkey, obj_key, 
         Hkey, comp_key, 
         INT, comp)

HPutDImage( Hproc_handle, proc_handle, 
           Hkey, obj_key, 
           INT, comp, 
           Himage, *image, 
           HB00L, copy, 
           Hkey, *image_key)

Herror HCrImage( Hproc_handle, proc_handle, 
             Hkey, image_key_in, 
             INT, index, 
             INT, type, 
             INT, width,height, 
             Himage, *image_key_out,
             Himage, *image_out)
```

Figure 5.19: Basic routines for creating new iconic objects and writing output object parameters (continued).

#### 5.4.3 HPutDRL

HPutDRL (see Fig. 5.18) stores region data encoded in the Hr1region structure region in the HALCON database and returns the data base key of the new object. In addition to that, the region object is assigned to the image object specified by the data base key obj_key as the region component. The latter might have been created before by using HCrObj or HCopyObj, see Figs. 5.20 and 5.21.

#### 5.4.4 HPutImage

HPutImage (see Fig. 5.19) stores the gray value channel (image matrix) image in the data base.
CHAPTER 5. HANDLING ICONIC OBJECTS AND CONTROL PARAMETERS

Figure 5.20: Create an image object with \texttt{HCopyObj} and insert a region component.

and returns the corresponding data base key in the parameter \texttt{obj.key}. This key can be used to insert the gray value channel (now encapsulated in a data base object) as a component of an arbitrary number of image objects (see \texttt{HDefObj} in section 5.4.5). The parameter \texttt{copy} (\texttt{FALSE} or \texttt{TRUE}) specifies, whether the pixel data is \textit{copied} to the HALCON data base\footnote{This is necessary if the original data is stored in temporary memory or if the data will be modified later on.} or only the address of the data is passed to the data base object. Fig. 5.21 shows a typical example for how to use \texttt{HPutImage}.

Figure 5.21: Creating a new image object using basic Extension Package Interface routines.

5.4.5 \texttt{HDefObj}

\texttt{HDefObj} (see Fig. 5.19) reassigns the component \texttt{comp (0 \ldots n)} of an image object given by its data base key \texttt{comp.key}. A typical application of this routine is shown in Fig. 5.21: Image data is stored in a data base object using \texttt{HPutImage}. Furthermore, a new image object is created by \texttt{HCrObj}. This new object does not contain any gray value channels, see section 5.4.1.

```
Hkey obj_key_in, obj_key_out, reg_key_out;
Hr1region *region_in;
Hr1region *region_out;
INT4 num_objs;

... par_in = 1; /* number of input parameter */
par_out = 1; /* number of output parameter */
HReadGVA(proc_handle,HNum_obj,&num_objs,par_in);
for (o=1; o<=num_objs; o++) {
  HGetObj(proc_handle,par_in,o,&obj_key_in);
  HGetFDRL(proc_handle,obj_key_in,&region_in);
  /* compute some region transformation ... */
  transform(region_in,region_out);
  HCopyObj(proc_handle,obj_key_in,par_out,&obj_key_out);
  HPutDRL(proc_handle,obj_key_out,region_out,&reg_key_out);
}
```

```
Himage image;
Hr1region *region;
Hkey image_key, obj_key;

... /

/* processed image data: image */
/* processed region data: region */
par_num = 1;
HPutImage(proc_handle,&image,TRUE,&image_key);
HCrObj(proc_handle,par_num,&obj_key);
HDefObj(proc_handle,obj_key,image_key,IMAGE_INDEX);
HPutDRL(proc_handle,obj_key,region,&region_key);
```
5.4. CREATING OBJECTS AND WRITING OUTPUT OBJECT PARAMETERS

Using HDefObj the previously stored image data is assigned as default gray value component 1 (IMAGE_INDEX) of the new image object.

5.4.6 HPutDImage

HPutDImage (see Fig. 5.19) combines HPutImage and HDefObj. It stores the gray value channel (image matrix) im\text{age} in the HALCON data base and returns the corresponding data base key in im\text{age}\_key. The parameter copy (FALSE or TRUE) is used as in HPutImage and specifies, whether the pixel data is copied\(^5\) or only a pointer to the data is passed to the data base. Moreover, the gray value channel is inserted into the image object obj\_key as component comp (1...n). Fig. 5.22 shows a typical application of this routine.

```c
Hkey  obj\_key,k;
Himage  image;
Hrlregion *region;
...
/* image: processed gray value data */
/* region: region data */
HCrObj(proc\_handle,par\_num,&obj\_key);
HPutDImage(proc\_handle,obj\_key,IMAGE\_INDEX,&image,FALSE,&k);
HPutDRL(proc\_handle,obj\_key,region,&k);
```

Figure 5.22: Creating a new image object using HPutDImage and HPutDRL.

5.4.7 HCrImage

HCrImage (see Fig. 5.19) is used for creating output images especially in the context of filter\(^6\) operations. As you know, HALCON image objects can share the underlying image matrices (gray value channels). Thus, input image objects of a filter operation might only differ in their area of definition (that is in their region component), but not in their gray values. Such objects have different data base keys and different regions, but contain references to the same gray value channels. Obviously, it is very desirable to propagate this relation to the resulting output image objects as well.

HCrImage allocates memory for the image matrix within the Himage structure image\_out (similar to HNewImage in section 3.2.2). The parameters type (BYTE\_IMAGE, LONG\_IMAGE, FLOAT\_IMAGE etc.; see Fig. 4.2 on page 50), width and height allow to specify the pixel type and the size of the new matrix. Note, that new matrices are by default initialized with 0. However, this can be suppressed from outside or inside of the operator, see the discussion of HNewImage in section 3.2.2 for details.

Furthermore, HCrImage creates a new data base object encapsulating this image data and returns the corresponding data base key in image\_key\_out (similar to HPutImage in section 5.4.4).

Finally, HCrImage establishes a link between the original input image data (referenced by the data base key image\_key\_in) and this new object to avoid the allocation of multiple output

\(^5\)This is necessary, if the pixel data was stored in temporary memory or if it will be modified later on.

\(^6\)This is due to the fact, that filter only modify the gray values but not the regions of image objects.
Figure 5.23: A typical example for using \texttt{HCrImage}: Providing an empty image matrix for a filter operation.

matrices per input matrix. \texttt{HCrImage} checks, whether there already exists an output matrix with the specified index index (see below) for a given input image \texttt{image\_key\_in}. If there is any, only a reference to it is returned in \texttt{image\_out} and \texttt{image\_key\_out} instead of creating a new matrix. Without this mechanism, a new matrix would be created again and again within the loop over all input objects (\texttt{HAL10obj}).

The parameter index is only of importance, if more than one output matrix per input matrix is needed: This is necessary for example, if an edge filter is performed in x- and y-direction independently. In this case \texttt{HCrImage} must be called several times for every input component – one time for each output matrix to be created. Assign increasing numbers (1, 2, 3, ...) to the parameter index to indicate a new output matrix. If there is only one matrix to create for the output image set index to 1.

Fig. 5.23 shows an exemplary application of \texttt{HCrImage}.

\subsection*{5.4.8 \texttt{HCrXLD}}

\texttt{HCrXLD} (see Fig. 5.18) is used for creating XLD objects in the HALCON data base. The parameter \texttt{xld\_type} allows to specify the kind of XLD to be created: It has to be set to XLD\_CONTOUR\_ID in case of a contour and to XLD\_POLYGON\_ID in case of a polygon. Corresponding to the selected type \texttt{HCrXLD} expects a pointer to \texttt{Hcont} or a pointer to \texttt{Hpoly} in the
5.4. CREATING OBJECTS AND WRITING OUTPUT OBJECT PARAMETERS

parameter xld. In both cases not the underlying XLD data but only the pointers to the data are copied to the data base. Thus, you must not deallocate or overwrite the XLD structures after calling HCrXLD.

```c
Hcpar rows[99], cols[99];
INT4_8 i,num_points;
Hkey key_out;
Hcont *cont;
...
HGetCPar(proc_handle,1,FLOAT_PAR,rows,1,99,&num_points);
HGetCPar(proc_handle,2,FLOAT_PAR-cols,1,99,&num_points);

/* create a contour (in general, this should be done */
/* calling an appropriate action procedure) */
HCrP(HAllocXLDCont(proc_handle,cont,num_points));
for (i=0; i < num_points; i++) {
    cont->row[i] = rows[i].par.f;
    cont->col[i] = cols[i].par.f;
}
cont->num = num_points;
HCrXLD(proc_id,1,cont,XLD_CONTOUR_ID,NULL,0,
      (DBFreeProc)HXLDFreeContour,&key_out);
```

Figure 5.24: A typical example for using HCrXLD: Create an XLD contour from a list of points.

```c
Hkey obj_key,key_out;
Hcont *cont;
Hpoly *poly;
...
HAllocObj(proc_handle,1,obj_key,i) {
    HGetXLD(proc_id,key,XLD_CONTOUR_ID,(void*)&cont);
    HCrP(HAlloc(proc_handle,(size_t)sizeof(Hpoly),&poly));
    /* approximate cont by poly */
    HCrP(HXLDPolygon(proc_handle,cont,...,&poly));
    HCrXLD(proc_handle,1,poly,XLD_POLYGON_ID,&obj_key,1,
           (DBFreeProc)HXLDFreePolygon,&key_out);
}
```

Figure 5.25: Another typical example for using HCrXLD: Create XLD polygons based on XLD contours.

XLD polygons contain a reference to the underlying contours (if any), see the definition of Hpoly in Fig. 4.10, section 4.3. Such a dependency between XLDs can be specified in HCrXLD by passing an array of data base keys in used_xlds and the corresponding number of keys in num_used_xlds. For contours these parameters typically are set to NULL and 0.

HCrXLD expects an appropriate deallocation routine for the XLD data in the parameter free_proc. Please use

HALCON 6.0
for XLD contours and

for XLD polygons allocated based on  HAlloc.
Finally,  HCrXLD not only creates a new database object, but also appends this object to the output object parameter with number  par_num. Fig. 5.24 and 5.25 show two typical applications of  HCrXLD.

5.5 Reading and Writing Control Parameters

This section describes routines for reading and writing in-/output control parameters, see Fig. 5.26. Interchanging control data with the host language is done via the data type  Hcpar (see section 4.4).

5.5.1  HGetCPar

HGetCPar (see Fig. 5.26) reads the parameter values of the input control parameter number  par_num (1...n) and writes them to the array  val of  Hcpar structures, that must has been allocated before with a suitable size (note, that ).

Note, that for every input value, its type (  LONG_PAR,  FLOAT_PAR,  STRING_PAR ) is also stored within the  Hcpar structure. Thus, it is possible to pass different control data types within one control parameter of a HALCON operator. The current number of values is returned in  num.

HGetCPar allows to restrict the parameter access in two respects:

- The expected type can be specified by  type. This also includes type combinations such as  LONG_PAR  |  FLOAT_PAR.
- The expected number of values can be specified by the interval (  min,  max ), whereas  max should not exceed the number of allocated  Hcpar elements within the array  val.

If the specified number or types of values are violated,  HGetCPar exits the supply procedure with an appropriate error message. Fig. 5.27 shows an application of  HGetCPar.

Note, that in case of string parameters, memory has to be allocated for  val[i].par.s. The easiest way to do this is to use  HAllocStringMem specifying the expected number of characters for all input strings, see section 5.5.7.

5.5.2  HGetSPar

HGetSPar (see Fig. 5.26) is a simplified version of  HGetCPar. In contrast to the latter a fixed number of parameter values is read, see Fig. 5.28. Therefore, the routine does not return the actual number\(^7\) of values.

---

\(^7\)This number is fixed. If more or less values are passed to the HALCON operator,  HGetSPar exits the supply procedure.
Names

HGetCPar, HGetSPar, HGetPPar, HGetCParNum, HPutCPar, HPutPPar, HAllocStringMem

Synopsis

#include "Halcon.h"

HGetCPar( Hproc_handle proc_handle, INT par_num, INT type, Hcpar *val, INT4_8 min,max, INT4_8 *num)

HGetSPar( Hproc_handle proc_handle, INT par_num, INT type, Hcpar *val, INT4_8 num)

HGetPPar( Hproc_handle proc_handle, INT par_num, Hcpar **val, INT4_8 *num)

HGetCParNum( Hproc_handle proc_handle, INT par_num, INT4_8 *num)

HPutCPar( Hproc_handle proc_handle, INT par_num, Hcpar *val, INT4_8 num)

HPutPPar( Hproc_handle proc_handle, INT par_num, Hcpar *val, INT4_8 num)

HAllocStringMem(Hproc_handle proc_handle, size_t size)

Figure 5.26: Routines for handling control parameters.

5.5.3 HGetCParNum

HGetCParNum (see Fig. 5.26) returns the number of single control values of the input control procedure with an error.
Figure 5.27: Read the control parameter values of the operator call demo(5, ’Text’, [0.5, 1.7, 4.4]) using HGetCPar.

```c
Hcpar   par1,par2;
Hcpar   par3[10];

/* allocate memory for strings */
HAllocStringMem(proc_handle,1000);
/* read a short/int/long value in parameter number 1 */
HGetCPar(proc_handle,1,LONG_PAR,&par1,1,1,&num); /* now par1.par.1 == 5 */
/* read a string in parameter number 2 */
HGetCPar(proc_handle,2,STRING_PAR,&par2,1,1,&num);
/* now par2.par.s == ”Text” */
/* read a tuple of float values in parameter number 3 */
/* (max. 10 values) */
HGetCPar(proc_handle,3,FLOAT_PAR,par3,1,10,&num);
for (i=0; i<num; i++) {
   /* par3[i].par.f == 0.5,1.7,4.4 */
   ...
```

Figure 5.28: Read the control parameter values of the operator call demo(5, ’Text’, [0.5, 1.7, 4.4]) using HGetSPar.

```c
Hcpar   par1,par2;
Hcpar   par3[3];

/* allocate memory for strings */
HAllocStringMem(proc_handle,1000);
/* read a short/int/long value in parameter number 1 */
HGetSPar(proc_handle,1,LONG_PAR,&par1,1);
/* now par1.par.1 == 5 */
/* read a string in parameter number 2 */
HGetSPar(proc_handle,2,STRING_PAR,&par2,1);
/* now par2.par.s == ”Text” */
/* read a tuple of 3 float values in parameter number 3 */
HGetSPar(proc_handle,3,FLOAT_PAR,par3,3);
for (i=0; i<num; i++) {
   /* par3[i].par.f == 0.5,1.7,4.4 */
   ...
```

5.5.4 HGetPPar

HGetPPar (see Fig. 5.26) returns the pointer to an array of Hcpar structures, which contain the parameter number par_num (1...n). This makes it possible to allocate exactly as much memory for parameter values as needed for reading them, for example, via HGetSPar (note that there is an alternative routine HGetPPar, which needs no memory to be allocated).
5.5. READING AND WRITING CONTROL PARAMETERS

```c
Hcpar  par1, par2;
Hcpar  *par3;
INT4_8  num;

/* allocate memory for strings */
HAllocStringMem(proc_handle,1000);
/* read a short/int/long value in parameter number 1 */
HGetSPar(proc_handle,1,LONG_PAR,&par1,1);
/* now par1.par.1 == 5 */
/* read a string in parameter number 2 */
HGetSPar(proc_handle,2,STRING_PAR,&par2,1);
/* now par2.par.s == "Text" */
/* read a tuple of float values in parameter number 3 */
/* first, get the number of values */
HGetCParNum(proc_handle,3,&num);
/* now num == 3 */
/* second, allocate memory for the values and read them */
HChP(HAllocTmp(proc_handle,(void**)&par3,(size_t)(num*sizeof(Hcpar))));

HGetSPar(proc_handle,3,FLOAT_PAR,par3,num);
for (i=0; i<num; i++) {
    /* par3[i].par.f == 0.5, 1.7, 4.4 */
    ...
}
```

Figure 5.29: Use HGetCParNum to get the number of control values before reading the control parameter values of the operator call demo(5,'Text',[0.5,1.7,4.4]).

parameter values of the input control parameter number par.num (1 ... n). It is an alternative to HGetCPar (see page 78). However, HGetPPar is much more efficient in terms of memory overhead, because it just returns pointers to the values instead of copying them. Thus, the caller needs not to allocate any memory before calling HGetPPar. On the other hand, the parameter values may only be read and must not be modified. Note, that for every parameter value, its type (LONG_PAR, FLOAT_PAR, STRING_PAR) is also stored within the Hcpar structure. This is necessary, because the same control parameter may contain values of different control data types. The current number of parameter values is returned in num.

Figure 5.30 shows the usage of HGetPPar based on the example which was also used for HGetCPar (compare figure 5.27 on page 80).

5.5.5 HPutCPar

HPutCPar (see Fig. 5.26) writes control data to the output control parameter number par.num of a HALCON operator. The num values in the Hcpar structure val are copied by this routine.

Note, that the types of the parameter values (LONG_PAR, FLOAT_PAR, STRING_PAR) are stored with each Hcpar structure. This allows to use a different type for every data element, see Fig. 5.31.
5.5.6 HPutPPar

HPutPPar (see Fig. 5.26) is an alternative to HPutCPar and writes control data to the output control parameter number num. However, instead of copying the num values like HPutCPar does, HPutPPar directly stores the pointer to the array of Hcpar structures (val). Thus, it causes less overhead. As HPutPPar directly uses the passed Hcpar array without copying, this array has to be allocated “permanently”, i.e. it must be allocated by using HALloc (see section 3.2.2 on page 42). Furthermore, the array must not be given free after passing it to HPutPPar. Figure 5.32 shows how to write output control data with HPutPPar.

Note again, that the types of the parameter values (LONG_PAR, FLOAT_PAR, STRING_PAR) are stored with each Hcpar structure, so that every data element may use a different type.

5.5.7 HALlocStringMem

In case of string parameters, additional memory for the parameter values has to be allocated, because the Hcpar structure only contains a pointer to char. The easiest way to do this is to use

```
Hcpar *ctrl_val1; /* array of Hcpar structures */
Hcpar *ctrl_val2;
Hcpar *ctrl_val3;

/* read all values of parameter number 1 */
HGetPPar(proc_handle,1,&ctrl_val1,&num);
/* now ctrl_val1[0].par.1 == 5 */
/* read a string in parameter number 2 */
HGetPPar(proc_handle,2,&ctrl_val2,&num);
/* now ctrl_val2[0].par.s == "Text" */
/* read a tuple of float values in parameter number 3 */
/* (may be any number of values) */
HGetPPar(proc_handle,3,&ctrl_val3,&num);
for (i=0; i<num; i++) {
    /* ctrl_val3[i].par.f == 0.5,1.7,4.4 */
    ...
}
```

Figure 5.30: Read the control parameter values of the operator call demo(5, ‘Text’, [0.5, 1.7, 4.4]) using HGetPPar.

```
Hcpar par[2];

par[0].par.1 = 6;
par[0].type = LONG_PAR;
par[1].par.f = 4.4;
par[1].type = FLOAT_PAR;
HPutCPar(proc_handle,1,par,2);
```

Figure 5.31: Write the control parameter values [6, 4.4] using HPutCPar.
5.6 Auxiliary Extension Package Interface Macros and Procedures

In this section a couple of auxiliary routines are described that facilitate the programming of supply or action procedures, see Fig. 5.33.

5.6.1 HckP

The macro HckP (see Fig. 5.33) checks the result state of a procedure call. Most of the HALCON procedures are of the type Herror\textsuperscript{8}. For internal HALCON procedures the result code HMSG_OK is returned if no error occurred, see also section 3.6. HckP exits the current procedure if the encapsulated procedure returns any other error code. Moreover, this error code is returned to the caller of the current procedure.

The macro is used to make source code more compact, but still safe with respect to error handling, see Fig. 5.34: Here, the call of the procedure HRLDecodep is encapsulated within the macro. If any error occurs, the procedure exits with an appropriate error number. Otherwise the processing is continued.

\textbf{Be aware, that all memory permanently allocated within a procedure will not be deallocated in case of an error} if you exit the procedure using HckP. This will lead to memory leaks.

\textsuperscript{8}All interface routines called in the examples in this chapter without being encapsulated by HckP have a functionality comparable to HckP, i.e. they exit the current procedure returning an appropriate error number in case of an error.
Names

HCKP, HCKNoObj, HRLDecomp, HNumChannels

Synopsis

#include "Halcon.h"

HCKP() Hproc_handle proc_handle,
       Herror (*proc())

HCKNoObj() Hproc_handle proc_handle)

Herror HRLDecomp() Hproc_handle proc_handle,
       Hregion *reg_in,
       INT r1, r1, r2, r2,
       Hregion *reg_inner,
       Hregion *reg_outer)

HNumChannels() Hproc_handle proc_handle,
       INT4_8 obj_index,
       INT *num_channels)

Figure 5.33: Auxiliary Extension Package Interface Macros and Procedures.

Figure 5.34: Error handling with HCKP.

5.6.2 HCKNoObj

The macro HCKNoObj (see Fig. 5.33) is used to check whether all iconic input parameters of an operator contain at least one object. Otherwise the supply procedure is exited with a return code depending on the current setting of

'no_object_result'

accessible by the HALCON operators set_system and get_system (see the Reference Manuals for details). The default setting for no_object_result is H_MSG_TRUE, that means HCKNoObj
reports “no error” in case of empty iconic input parameters. Note, that in this case all output parameters are also empty.

We recommend to call

\[
\text{HClNoObj}(\text{proc\_handle});
\]

at the beginning of the supply procedure for every operator with iconic input objects to guarantee the existence of iconic data to be processed within the operator.

### 5.6.3 HRLDecomp

HRLDecomp (see Fig. 5.33) is an auxiliary procedure to ease border treatment within the action procedures of filter operators based on filter masks. It’s prototype is included in \text{hlib/HRLClip.h}.

```c
Herror IPBFilter(Hproc\_handle proc\_handle, /* proc\_handle */
                  HBYTE *in,    // input image */
                  Hr\_region *region,  // area of def. */
                  INT width, height,  // image size */
                  HBYTE *out)   // output image */
{
    ...
    Hr\_region *inner, *outer;
    HClP(HAllocRLNumTmp(proc\_handle,&inner,region->num));
    HClP(HAllocRLNumTmp(proc\_handle,&outer,region->num*2));
    HClP(HRLDecomp(proc\_handle,region,1,1,
                   height-2,width-2,&inner,&outer));
    /* filtering without border treatment */
    for (i=0; i<inner->num; i++)
        for (k=CB(inner->rl,i,width); k<CE(inner->rl,i,width); k++)
            out[k] = ...;
    /* filtering with border treatment */
    for (i=0; i<outer->num; i++)
        for (k=CB(outer->rl,i,width); k<CE(outer->rl,i,width); k++)
            out[k] = ...;
    HClP(HFreeRLTmp(proc\_handle,outer));
    HClP(HFreeRLTmp(proc\_handle,inner));
}
```

**Figure 5.35:** Border treatment for a $3 \times 3$ filter.

The region (respectively area of definition) \text{reg\_in} of the image to be processed is split into two parts: \text{reg\_inner} and \text{reg\_outer}. Note, that both new regions must have been allocated before, see section 3.2.1.

\text{reg\_inner} is the original region \text{reg\_in} minus the pixels around the image border. It contains all pixels within the rectangle specified by the upper left corner\(^9\) ($r_{1,c1}$) and the lower right

\(^9\) $r_1$ and $r_2$ denote row coordinates, $c_1$ and $c_2$ column coordinates.
corner \((r_2,c_2)\). These coordinates should be selected that way, that the filter mask is completely within the image when placed on any pixel inside of the rectangle. Thus, no border treatment is necessary within \(\text{reg\_in}\).

\(\text{reg\_out}\) is set to all remaining pixels within \(\text{reg\_in}\). For those pixels an appropriate (and time consuming) border treatment has to be done. Fig. 5.35 shows an application of \(\text{HRLDecomp}\).

Note, that the distinction of “inner” and “outer” area only refers to the image border, but not to the border of the regions specifying the area of definition. So, it is only ensured that no memory access to positions outside the image matrix can happen, but there is no guarantee that all pixel values covered by the filter mask when placed on pixels within \(\text{reg\_inner}\) are defined, if they do not belong to \(\text{reg\_in}\). This means, a filter might use undefined gray values along the border of the area of definition of the input images. The HALCON user should keep that in mind, when applying a sequence of filters to restricted regions of interest.

### 5.6.4 HNumOfChannels

\(\text{HNumOfChannels}\) returns for the first input object parameter of a HALCON operator the number of channels of the image object with number \(\text{obj\_index}\). Thus, this routine is just a shortcut for using \(\text{HPNumOfChannels}\) as defined in Fig. 5.5 with \(\text{par\_num} = 1\).
Chapter 6

Special Routines for Typical Supply Procedures

In the previous chapter the basic routines for handling iconic objects, object parameters, and control parameters of a HALCON operator have been introduced. Based on these this chapter describes a set of convenience routines that facilitate the programming of supply procedures in typical situations.

6.1 Loop Macros

A frequently needed task is to process all image objects (images or regions) passed to an operator by one or two input object parameters. This is true for example for most filter and many segmentation operators. Loop macros provide a framework for this problem, see also section 5.3.

All macros described in this section have in common that they implement a loop over all image objects within the first input object parameter. HAllFilter2 also includes a parallel loop over all objects within the second input object parameter. It is assumed that the operator does not have any more input object parameters. The macros provide users with the region and pixel data of each image object so that they can work on the data within the loop. Some also create output image objects.

The macros return the index of the current object obj_index (1...n) within the input object parameter. Thus, obj_index can be seen as a reference parameter of the macro and therefore is notated with `&obj_index`. The following table contains an overview of all loop macros, their areas of application, and the provided image data. Their syntax is defined in Fig. 6.1.

1Note, that this is a specific notation only - do not pass pointers to the macros!
<table>
<thead>
<tr>
<th>Macro</th>
<th>Application</th>
<th>Image Data Provided</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAllReg</td>
<td>region features, region transformation, binary morphology</td>
<td>region data (Hr1region) of input image object</td>
</tr>
<tr>
<td>HAllSegm</td>
<td>gray value features, segmentation</td>
<td>region and image data (Hr1region and Himage) of input image object</td>
</tr>
<tr>
<td>HAllFilter</td>
<td>filter, image transformations</td>
<td>region and image data (Hr1region and Himage) of input image object, image data structure (Himage) for image output object</td>
</tr>
<tr>
<td>HAllFilter2</td>
<td>filter, segmentation, arithmetics with two input images</td>
<td>intersection of the regions of both input image objects (Hr1region), the image data of both objects, and a image data structure (Himage) for the image output object</td>
</tr>
</tbody>
</table>

The macros return all regions as pointers to the original region data within the HALCON data base. So the programmer is only allowed to read them. The only exception is again HAllFilter2 that computes the intersection of the region data of both input images and stores it in a new region.

### 6.1.1 HAllReg

HAllReg (see Fig. 6.1) implements a loop over all objects of the first input image object parameter. For every object within this parameter it returns the region of the object in region\(^2\). All gray value channels are ignored.

HAllReg is a combination of HAllObj (see section 5.3.1) and HGetFDRL (see section 5.2.2). It is typically used within feature extraction operators: A list of input regions has to be examined concerning special features. Some HALCON operators of this kind are e.g., circularity, area_center or cont_length. Moreover, it is possible to return new region(s) by using the interface macro HNewRegion (see page 95) as it is done e.g., by select_shape or shape_trans.

Fig. 6.2 illustrates the application of HAllReg showing a complete supply procedure for a hypothetical operator center that computes the center of gravity of all input regions. The operator has one input object parameter that exclusively contains regions and two output control parameters for returning the results as tuples of floating-point numbers (the coordinates of the centers of gravity of all regions). The corresponding def-file (short version) might look like

```c
center <- CIPCenter[Regions::Rows$F,Columns$F];
```

center may be called with one or more regions as input. HAllReg implements a loop over all of them, sets the loop index \(i\) to the current index of the region \((1 \ldots n)\), and passes a pointer to the region to the action procedure that performs the center of gravity. In our case the action procedure is the internal HALCON procedure HRLArea. The result values are written into two arrays of the type Hcpar and returned by HPutCPar (see section 5.5.5).

---

\(^2\)Remember: HALCON image objects consist of one region specifying the area of definition and an arbitrary number of gray value channels containing the pixel data
Names

HAllReg, HAllSegm, HAllFilter, HAllFilter2

Synopsis

#include "Halcon.h"

HAllReg( 
    proc_handle proc_handle, 
    Hrlregion **region, 
    INT4_8 &obj_index)

HAllSegm( 
    proc_handle proc_handle, 
    Hrlregion **region, 
    Himage *image, 
    INT max_channels, 
    INT4_8 &obj_index)

HAllFilter( 
    proc_handle proc_handle, 
    Hrlregion **region, 
    Himage *image_in, 
    Himage *image_out, 
    INT max_channels, 
    INT4_8 &obj_index)

HAllFilter2( 
    proc_handle proc_handle, 
    Hrlregion *region, 
    Himage *image_in1, 
    Himage *image_in2, 
    Himage *image_out, 
    INT max_channels, 
    INT4_8 &obj_index)

Figure 6.1: Convenience loop macros to access iconic objects. "&" denotes output parameters of the macros. This is only a special notation to make clear that these parameters are changed by the macros. So do not pass pointers to variables but the variables itself to the macro.

Note, that the Hcpar arrays Row and Col are allocated with a fixed size on the stack using MAX_OBJ_PER_PAR. This define equals the maximum allowed number of objects per input object parameter of a HALCON operator. Keep in mind that this number is quite large (100 000 in the current version). Thus, it might be better to allocate the arrays dynamically using HAllocTmat (see section 3.2.1) with only as much elements as needed (see HReadGVA(...) on page 61).

6.1.2 HAllSegm

HAllSegm (see Fig. 6.1) extends HAllReg: Not only the region but also all gray value chan-
Figure 6.2: An application of HAL1Reg: Compute the center of gravity for all input regions in the first input object parameter.

nals of an object within the first input image object parameter are accessed within a loop. Thus, HAL1Segm is a combination/modification of HAL1Obj (see section 5.3.1), HGetFDRL (see section 5.2.2), and HAL1Comp (see section 5.3.2). It is especially useful for segmentation operators (transition of gray value channels to regions). Some typical HALCON operators that make use of HAL1Segm are threshold, regiongrowing, auto_threshold1, class_ndim1 or label_to_region.

A HALCON image object (in short a HALCON image) consists of one region that specifies its area of definition and of one or more channels (gray value components) containing the pixel data. All channels of an image are of the same size, but may have different pixel types. The data contained in a channel (pixel type, the image matrix etc.) is stored in a structure of type Himage (see section 4.1). If the image has only one channel, the address of a variable of the type Himage is passed to HAL1Segm. For multichannel images an array of the type Himage [max_channels] must be passed to it. The maximal number of channels to be accessed by HAL1Segm is specified by the parameter max_channels, see Fig. 6.3.

Figure 6.3: The parameters image and max_channels of HAL1Segm.

The parameter max_channels only specifies the maximum number of channels. If an image
contains less channels, the corresponding Himage elements in the image array are undefined. The actual number of channels can be accessed e.g., using HNumOfChannels (see section 5.6.4). An image must contain at least one channel. Otherwise, HAL1Segm returns an error. If the number of channels exceeds max_channels, the remaining channels are ignored. Furthermore, the size of all channels is checked for equality.

```
Error IPSegm(Hproc_handle proc_handle, /* HALCON proc. handle */
    Hrlregion *region,    /* area of definition */
    Himage  *image,     /* input image */
    Hrlregion *out)  /* segmentation result */
{
    /* compute region "out" from "region" and "image" */
    return(H.MSG_OK);
}

HUserExport Error CIPSegm(Hproc_handle proc_handle)
{
    INT4_8 i;
    Hrlregion *region, *new_region;
    Himage  image;
    HCKP(HAllocRLTmp(proc_handle,&new_region)); /* allocate memory*/
    HAL1Segm(proc_handle,&region,&image,1,i) {    /* all inp. images*/
        HCKP(IPSegm(region,&image,new_region));
        HNewRegion(proc_handle,new_region); /* allocate new */
    }
    /* region */
    HCKP(HFreeRLTmp(proc_handle,new_region));    /* free memory */
    return(H.MSG_TRUE);
}
```

**Figure 6.4: A typical application of HAL1Segm: A segmentation operator.**

Fig. 6.4 shows a framework for a supply procedure for a segmentation operator using HAL1Segm. The corresponding hypothetical operator segm has one input object parameter for input images containing at least one channel. The def-file (short version) might look like

```c
segm <- CIPSegm[Image:Region::];
```

The macro HAL1Segm used in CIPSegm performs a loop over all image objects within the first input object parameter Image of segm. With every pass of the loop it returns the image data of the current input object in the variables region (area of definition) and image (first gray value channel). The results of the segmentation (new_region) are stored via the macro HNewRegion (see section 6.2.1, page 95) as new objects of the HALCON data base and returned in the (first) output object parameter (Region). Note, that this simple operator has no control parameters.

A new region is allocated for the action procedure via HAllocRLTmp (see section 3.2.1) before entering the loop. This region is used as temporary memory for the computation result and must be deallocated (via HFreeRLTmp) before exiting the supply procedure.

---

3 Remember: The sizes must be equal, whereas the pixel types of different channels may vary
6.1.3 HAllFilter

HAllFilter (see Fig. 6.1) further extends the loop macros introduced so far: For every input image object a new output image object is created (and added to the first output object parameter) with

- the same number of channels (components),
- the same pixel types (of the corresponding channels),
- the same image size and
- the same area of definition (region).

Thus, HAllFilter is a combination/modification of HAllObj (see section 5.3.1), HGetFDRL (see section 5.2.2), HAllComp (see section 5.3.2), HCopyObj (see section 5.4.2), and HPutDImage (see section 5.4.6). It has been designed for filter operators that typically create a modified result image for every input image. The region remains unmodified. Typical examples for this functionality are the HALCON operators sobel, meanimage, fft, scale_max or laws_byte.

Fig. 6.5 shows a typical application of HAllFilter – a hypothetical operator scale_new that multiplies all gray values of the input objects within the first input object parameter with a constant. The corresponding def-file(short version) might look like

```
scale_new <- CIPScaleNew[Image:ImageScaled:Mult$f;];
```

The operator has one input object parameter (Image) that contains one or more input image objects (each consisting of one or more gray value channels and one region as area of definition). With every pass of the loop an image object is accessed and its components are transferred to region and image_in, see also HAllSegm in section 6.1.2. These components are passed to the action procedure IPScaleNew. In this example only the first channel of each input object is used.

HAllFilter also creates output objects (in the HALCON data base) to return the modified data. Furthermore, these objects are added to the list of objects for the first output object parameter (here ImageScaled). Their underlying image matrices are accessible via image_out. Those are also passed to the action procedure IPScaleNew within the loop. IPScaleNew calculates the new gray values for all pixels within the area of definition of the input image and writes them into the provided image components image_out. The regions of the image objects remain unmodified so that all pixels of output objects lying outside the region are undefined.

Note, that HAllFilter creates all necessary image matrices, combines them with the input regions to new image objects, and returns those in the first output object parameter. So the programmer of the supply procedure needs not to bother about the handling of image objects etc. To access the input control parameter, HGetSPar (see section 78) is used in the example.

The structure image_in may contain different pixel types. There are several ways to handle this within an operator:

---

4Note, that image_out contains a pointer to a pixel matrix with the same size and pixel type as image_in.
1. The simplest method is to implement the operator just for the most common pixel type `BYTE_IMAGE` and return an error message (`H_ERR_WIT`) for any other pixel type.

2. Another way is to provide several action procedures – one for every pixel type – and call the appropriate procedure via `switch(image_in.kind)`.

3. The third (generic) method makes use of the macros `HDFImage` and `HImageFD`, see Fig. 6.6. They encapsulate the access to gray values by buffering them in a `double` variable.

   By writing pixel values via `HImageFD`, the `double` value `val` is converted into the current pixel type of `image_in` (and therefore may be clipped) and stored in the pixel specified by the linear coordinate `lin_coord`, see section 4.1.

   The other way around, the specified pixel value is converted to `double` and returned in `val` when using `HDFImage` to read image data.

The third method has been used in our example, as it allows a very compact source code. But this variant naturally shows drawbacks in terms of computation time: Two type conversions have to be computed with every pixel access and all pixel arithmetic has to be done in `double`.  

Figure 6.5: A typical application of HAllFilter: A filter operator.
6.1.4 HAllFilter2

The macro HAllFilter2 (see Fig. 6.1) is a variation of HAllFilter introduced in the previous section. It facilitates the implementation of filters with two input object parameters. Examples for this technique are the HALCON operators add_image, mult_image, bit_and, and max_2_images.

HAllFilter2 extends HAllFilter so that with every image object of the first input object parameter also the corresponding image object (with the same object index obj_index) of the second input object parameter is provided. Moreover, it checks the image sizes of both image objects for equality. If the sizes are equal, the images are provided in the loop variables image_in1 and image_in2, otherwise an error is returned.

Moreover, HAllFilter2 computes a new input region (region) by intersecting the areas of definition of both input images. Note, that memory for this region must have been allocated before (e.g. with HAllocRLTmp). Furthermore, region must be deallocated at the end of the supply procedure (HFreeRLTmp).

Output image objects are created and returned in the first output object parameter in the same way as described for HAllFilter in section 6.1.3.

Fig. 6.7 shows a typical application of HAllFilter2 – the implementation of an operator that adds the gray values of two input images. As in procedure IPScaleNew in Fig. 6.5, the macros HDFImage and HImageFD are used to read and write pixel values. In this example, that exhibits the additional advantage that the operator can even add the gray values of two images with different pixel types.

---

**Names**

HDFImage, HImageFD

**Synopsis**

```c
#include "Halcon.h"

HDFImage( double &val,
           Himage *image,
           INT4_8 lin_coord)

HImageFD( Himage *image,
          double val,
          INT4_8 lin_coord)
```

Figure 6.6: Auxiliary macros for generic access to pixel data. "&" denotes an output parameter of HDFImage. This is only a special notation to make clear that this parameter is *changed* by the macro. So do *not* pass a pointer to double but the double variable itself to the macro.
6.2 Object Generation

This section describes the convenience routines HNewRegion, HPutRect, and HDupObj for creating new image objects, see Fig. 6.8.

HNewRegion creates a new image object in the HALCON data base encapsulating the specified region and adds this object to the first output object parameter. HPutRect inserts a rectangular region (as area of definition) into an already existing output image object. Finally, HDupObj adds an iconic input object to the object list of the first output object parameter.

6.2.1 HNewRegion

HNewRegion (see Fig. 6.8) is used to create a new image object in the HALCON data base which is also added to the first output object parameter. Only the region component (area of
Names

HNewRegion, HPutRect, HDupObj

Synopsis

#include "Halcon.h"

HNewRegion(Hproc_handle proc_handle,
            Hrlregion *region)

HPutRect( Hproc_handle proc_handle,
          Hkey obj_key,
          INT width,height)

HDupObj( Hproc_handle proc_handle,
        INT4_8 obj_index)

Figure 6.8: Convenience routines for creating output objects.

definition) of the new object is defined: It contains a copy of region. It is common to call an
image object region, when only its region component is used.

HUserExport Error CIPConvex(Hproc_handle proc_handle)
{
   Hrlregion *region;
   Hrlregion *region_new;

   HChkP(HAllocRLtmp(proc_handle,&region_new) /* allocate memory */
   HAllReg(proc_handle,&region,i) { /* all regions */
      HChkP(HRLConvex2(proc_handle,region,region_new));
      /* create an output region object (first output param.) */
      HNewRegion(proc_handle,region_new);
   }
   HChkP(HFreeRLtmp(proc_handle,region_new)); /* free memory */
   return(H_MSG_TRUE);
}

Figure 6.9: A typical application of HNewRegion: The region transformation “convex hull”.

Typically, HNewRegion is used within segmentation procedures (see Fig. 6.4) and region trans-
forming procedures such as the procedure CIPConvex shown in Fig. 6.9. CIPConvex corre-
sponds to a hypothetical operator trans_convex which might be defined like

trans_convex <- CIPConvex[Region:RegionConvex::];

In the example all regions within Region are provided in region by HAllReg (see section
6.1.1) one after the other. The HALCON procedure HRLConvex2 computes the convex hull for
6.2. OBJECT GENERATION

each region and writes it to region_new. Furthermore, a new region object in the HALCON
data base is created and added to the output object parameter RegionConvex.

Note, that HNewRegion was also used in the example in Fig. 6.4 on page 91.

6.2.2 HPutRect

HPutRect (see Fig. 6.8) inserts a rectangular region in an already existing output object with
data base key obj_key. Mostly it is used in connection with operators that create new image
objects like in the example in Fig. 6.10.

```c
HUserExport Herror CIOGrabImage(Hproc_handle proc_handle)
{
    Hkey obj_key, image_key;
    Himage image;

    HCrObj(proc_handle, 1, &obj_key);
    HckP(HNewImage(proc_handle, &image, BYTE_IMAGE, 640, 480));
    /* grab an NTSC image 640 x 480 pixel, 8 bit */
    HckP(10BGrabImage(&image));
    HPutDImage(proc_handle, obj_key, 1, &image, FALSE, &image_key);
    HPutRect(proc_id, obj_key, image.width, image.height);
    return(H_MSG_TRUE);
}
```

Figure 6.10: A typical application of HPutRect: Grabbing an image.

Note, that HCrObj (see section 5.4.1) creates an object with empty region. In many situations
the default area of definition for a new image will be the full image domain which is a rectangle.
In this situation the use of HPutRect is very convenient.

6.2.3 HDupObj

HDupObj (see Fig. 6.8) is a combination/simplification of HGetObj (see section 5.1.1) and
HCopyObj (see section 5.4.2). It duplicates iconic input objects (images, regions, XLD) in
order to pass them directly to the first output object parameter. The objects to be duplicated are
specified by their index obj_index within the first input object parameter. Note, that this index
corresponds to obj_index as defined for the loop macros (HALl*) in section 6.1. HDupObj is
typically used for operators that examine objects and select them by a given criteria. Some
HALCON operators of this kind are select_shape, has_shape, or select_gray.

Fig. 6.11 shows an application of HDupObj: CIPUserSelect computes the length of the contour
each input region (in parameter 1). All regions with a contour of at least the minimal length
min are passed to the output.

Since HDupObj only increases the number of references to a data base object instead of copy-
ning it physically, its memory costs and computational costs are neglectable. In contrast, e.g.,
HNewRegion and HALlFilter always allocate new memory for the output objects. So when-
ever possible, HDupObj should be used.
Figure 6.11: A typical application of \texttt{HDupObj}: Duplicate input regions that fulfill a specific criteria (length of contour).
Chapter 7

Creating a New HALCON Package

7.1 The HALCON Compiler hcomp

The HALCON compiler hcomp is the most important tool for creating a new HALCON package. It automatically generates the interface code for the desired host language, the help files, and the documentation files. hcomp processes def-files that contain descriptions of all operators as a basic resource, see chapter 2. The HALCON compiler uses the same mechanism for generating the original system operators and user-defined operators.

The syntax of the hcomp command is as follows:

```
hcomp <options> <files>
```

Example:

```
hcomp -u -C -ppackage MyUserOps.def
```

The parameter “-u” signals that an interface for user-defined operators should be generated. This parameter can be omitted as it is the default\(^1\). One or more definition files that are separated by blanks must be specified as files argument. The extension .def may be omitted:

```
hcomp -C -ppackage MyUserOps1 MyUserOps2 MyUserOps3
```

To generate the appropriate interface code, the correct option must specify the desired host language (for C as host language: -C). Then, hcomp generates a couple of source files (in the example above: HCpackage.c and HCpackage.h) for the package package. Their names depend on the host language and the specified package, not on the names of the input def-file(s).

7.1.1 Selection of the Host Language

One call of hcomp can always create only one interface, i.e, the options described as follows can only be switched on under mutual exclusion. Use

\(^1\)Note, that hcomp also is used by MVTec to generate the interface code for the standard HALCON system.
to create interface code for C. \texttt{hcomp -C -ppackage} creates the files \texttt{HCPackage.c} (interface code) and \texttt{HCPackage.h} (prototypes of operators).

To create interface code for C++ use

\texttt{hcomp -D -ppackage} \texttt{def-files}

creating \texttt{HCPPackage.cpp}, \texttt{HCPPackageGlobal.cpp}, and \texttt{HCPPackage.h} (prototypes) providing the C++ interface to the new operators.

The generated source files are the basis for the interface libraries, see section 7.2.

### 7.1.2 Creating the help-files

These files are necessary for the online access to the operator knowledge base via HALCON operators like \texttt{get\_operator\_info}. For example, \texttt{HDevelop} uses their information to build up the menu tree 'Operator' (chapter and section structuring) and 'Suggestions' (alternatives, cross references, predecessor, successor, and keywords). The help files are generated for the language selected with the (optional) -l option. The default language is \texttt{en}.English.

\texttt{hcomp -M [-l\texttt{language}] def-files}

creates the following files:

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{language.hlp}</td>
<td>The operator description (text, value lists, ...)</td>
</tr>
<tr>
<td>\texttt{language.idx}</td>
<td>Index for every operator specifying the start address of its entry within the \texttt{.*\texttt{.hlp}} file.</td>
</tr>
<tr>
<td>\texttt{language.key}</td>
<td>List of keywords with the associated operators.</td>
</tr>
<tr>
<td>\texttt{language.num}</td>
<td>Specification of the number of parameters per parameter class for every operator.</td>
</tr>
<tr>
<td>\texttt{language.sta}</td>
<td>Specification of the parameter names and the chapter names for every operator.</td>
</tr>
</tbody>
</table>

In addition to the option -M

\texttt{-cTAGS-FILE}

can be set in order to insert the source file and the line number of the supply procedure of the operator into the \texttt{.*\texttt{.hlp}} file. To do this, a \texttt{TAGS-File} (usually \texttt{TAGS}) must have been generated by the \texttt{etags} command before.
7.1.3 Creating the Manuals

\texttt{hcomp} can generate \LaTeX{} files containing the complete reference manuals (full text or short version) adapted to the different supported programming languages, i.e. the description of the operators is provided in a specific syntax. Use

\begin{verbatim}
  hcomp -R \textit{planguage:specification} [-l \textit{language}] \textit{def-files}
\end{verbatim}

to create \LaTeX{} files with \textit{planguage} and \textit{specification} as follows:

\textit{planguage} can be set to \texttt{c}, \texttt{c++}, and \texttt{trias} (corresponds basically to the \texttt{HDevelop-Syntax}). \texttt{trias} is default and may therefore be omitted. The \textit{specification} determines the extent of the manual by specifying the additional operator information that will be inserted into the manual besides the operator header. Supported \textit{specifications} are:

\begin{tabular}{|c|p{0.5\textwidth}|}
\hline
\textbf{Specification} & \textbf{Description} \\
\hline
\texttt{all} & All available information is presented in the manual. \\
\texttt{sh [ort]} & Only the short descriptions are inserted. \\
\texttt{so1} & A file is generated for each operator containing its complete description. \\
\hline
\end{tabular}

By specifying \texttt{so1} a file \texttt{operator.tex} is created for every operator. Using any of the other specifications the file \texttt{HReference.tex} is generated.

The generated \LaTeX{} files use the package \texttt{halcon-reference.sty}. This package resides in the directory \texttt{HALCONROOT/doc/macros} (UNIX notation). You must include this directory in the environment variable \texttt{TEXINPUTS}, otherwise \LaTeX{} will be unable to find it (typical error message: File ‘halcon_reference.sty’ not found.

The \LaTeX{} file(s) can be transformed to PostScript files by using \texttt{latex}, \texttt{makeindex}, and \texttt{dvips}. Note, that \texttt{latex} must be called several times to get the references right.

7.1.4 Creating HTML Files

\texttt{hcomp} can also generate HTML files. Thus, its easy to provide an online documentation of new operators that can be accessed using an HTML browser. Use

\begin{verbatim}
  hcomp -L \textit{planguage:specification} [-l \textit{language}] \textit{def-files}
\end{verbatim}

to create HTML files. All things said above about the issues \textit{planguage} and \textit{specification} hold also for this section — with the exception that \texttt{so1} is not supported. The following files are generated:
## 7.1.5 Miscellaneous

Here are some more options supported by `hcomp`:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>hcomp -i filename</code></td>
<td>Consider only the operators listed in <code>filename</code>.</td>
</tr>
<tr>
<td><code>hcomp -x filename</code></td>
<td>Do not consider the operators listed in <code>filename</code>.</td>
</tr>
<tr>
<td><code>hcomp -m def-files</code></td>
<td>Generate a list of operators within the <code>def-files</code> sorted by <code>modules</code>.</td>
</tr>
</tbody>
</table>

### 7.2 Generating HALCON Packages

Once the new operators have been described in `def-file(s)` and implemented (supply and action procedures) a couple of dynamic objects (DLLs in Windows NT an Windows 2000, shared libraries in UNIX environments) must be created. **Never change the name of a package or the corresponding names of the libraries/DLLs contained in a package.** These names are encoded within the libraries. If you change the names this information will not match any longer. Thus, the loader of the operating system will fail to open the dynamic libraries. If you want to rename a package, you must create the libraries/DLLs again.

To activate a package, its complete path must be added to the environment variable `HALCONEXTENSIONS`, e.g.,

```bash
%HALCONROOT%\examples\extension_package\halconuser
```

Please note, that the package paths in `HALCONEXTENSIONS` are separated by semicolons (Windows NT/ 2000) or colons (UNIX), see also section 1.3.

### 7.2.1 Creating the Operator Libraries

The supply and action procedures must be encapsulated in a DLL or a shared library. The name of this shared object has to be identical with the package name (plus file extension). For the use
by Parallel HALCON, a second version of the library must be created, with the additional prefix `par` to the name. Please refer to section 7.2.5 and the documentation of your programming environment for more details (also about the following sections).

Under Windows NT or Windows 2000 the generated DLLs must be placed in the subdirectory `bin\%ARCHITECTURE%`, the libraries in the subdirectory `lib\%ARCHITECTURE%`. Under UNIX, the generated shared libraries must be placed in the subdirectory `lib\%ARCHITECTURE%`, binaries in the subdirectory `bin\%ARCHITECTURE%` of the package.

### 7.2.2 Creating the C Interface

To access new HALCON operators inside a package within C programs or from `HDevelop` you must create a C interface library `package.c` residing in the subdirectory `lib\%ARCHITECTURE%` of the package. This library is based on a single C file `HPackage.c` which is generated via

```
hcomp -C -ppackage def-file(s)
```

from the `def-file(s)` of your new operators. Note, that `hcomp` simultaneously generates the file `HPackage.h` containing the C prototypes of your new operators. Include this file in your C programs using these operators.

### 7.2.3 Creating the C++ Interface

To access new HALCON operators inside a package within C++ programs you must create a C++ interface library `package.cpp` residing in the subdirectory `lib\%ARCHITECTURE%` of the package. This library is based on the files `HCPPackage.cpp` and `HCPPackageGlobal.cpp` generated via

```
hcomp -D -ppackage def-file(s)
```

from the `def-file(s)` of your self developed operators. Note, that `hcomp` simultaneously generates the file `HCPPackage.h` containing the C++ prototypes of your new operators. Include this file in your C++ programs using these operators.

### 7.2.4 Creating New Applications

In order to create new application programs (written in C or C++) based on your own HALCON operators you must link the corresponding language interface libraries `package.c` or `package.cpp` to your objects. Furthermore, you will need the HALCON library itself and the HALCON/C or HALCON/C++ library (as for any HALCON application).

### 7.2.5 Additional Information for Specific Architectures

The previous sections summarized the generation of HALCON extensions in general. This section contains additional information for specific architectures. The main differences concern the name handling and the generation of shared libraries / DLLs.
7.2.5.1 Generating Packages Under Windows NT or Windows 2000

In order to activate a package, its complete path, e.g.

```
%HALCONROOT%\examples\extension-package\halconuser
```
must be included in the environment variable HALCONEXTENSIONS. In the Windows NT / 2000 version the package paths in HALCONEXTENSIONS are separated by semicolons.

Note, that DLLs must be stored in the subdirectory `bin\i586-nt4` of the package, the corresponding libraries in the subdirectory `lib\i586-nt4`.

- **Exporting the supply procedures:**
  The supply procedures (like `CIPUserThreshold()` in `cipuserthreshold.c`) must be explicitly exported by the operator DLL `package.dll` containing your new operators. This is done by the macro `HUserExport`, see e.g., Fig. 1.6.

- **Creating object files for the Parallel HALCON version:**
  If you want to create a Parallel HALCON version of your package, a second set of object files must be created from the source code files, using the compiler define `/DH_PARALLEL`. We recommend to place these object files into a separate directory to prevent confusing them with the object files created for the standard HALCON version.

- **Creating the operator DLL** `package.dll` and the library `package.lib`:
  To create the operator DLL `package.dll` containing new HALCON operators and the corresponding library `package.lib`, the object files containing the corresponding supply and action procedures and the HALCON library `halcon.lib` must be linked.

  To create the libraries `parpackage.dll` and `parpackage.lib` for Parallel HALCON, the corresponding object files compiled with the define `/DH_PARALLEL` must be linked to the Parallel HALCON library `parhalcon.lib` instead of `halcon.lib`.

- **Creating the C interface DLL** `packagec.dll` and the library `packagec.lib`:
  To create the libraries `packagec.dll` and `packagec.lib` which provide the C interface to new HALCON operators, the object file `Hpackage.obj`, the new operator library `package.lib`, the HALCON C-library `halconc.lib`, and the HALCON library `halcon.lib` must be linked.

  To create the libraries `parpackagec.dll` and `parpackagec.lib` for Parallel HALCON, the corresponding object file compiled with the define `/DH_PARALLEL` must be linked to the Parallel HALCON version of the libraries, i.e., `parpackage.lib`, `parhalconc.lib`, and `parhalcon.lib`.

- **Creating the C++ interface DLL** `packagecpp.dll` and the library `packagecpp.lib`:
  To create the DLL `packagecpp.dll` providing the C++ interface to new HALCON operators and the corresponding library `packagecpp.lib`, the object files `HCPPpackage.obj` and `HCPPpackageGlobal.obj`, the new operator library `package.lib`, the HALCON C++ library `halconcpp.lib` and the HALCON library `halcon.lib` must be linked.

  To create the libraries `parpackagecpp.dll` and `parpackagecpp.lib` for Parallel HALCON, the corresponding object files compiled with the define `/DH_PARALLEL` must be linked to the Parallel HALCON version of the libraries, i.e., `parpackage.lib`, `parhalconcpp.lib`, and `parhalcon.lib`. 
7.2. GENERATING HALCON PACKAGES

- Creating new applications:
  In order to create new application programs (written in C or C++) you must link packagec.lib or packagecpp.lib to your objects. Furthermore, you will need halconc.lib or halconcpp.lib (as for any HALCON application).
  To create a Parallel HALCON version of your application, just link the object files to the Parallel HALCON version of the libraries (e.g., parpackagec.lib and parhalconc.lib in case of a C application). You can use the same object files as for creating the standard HALCON application.
  To be able to link the package DLL to your application program, the complete DLL file path of the new package, e.g.,
  
  \%HALCONROOT\%\examples\extension_package\halconuser\bin\i586-nt4

  has to be added to the environment variable PATH.
  
  **Do not copy a package DLL into the Windows system directories**, as it would be loaded twice in this case!
  
  If you encounter program crashes, **check the stack size** allocated by your application. A stack size of 6 - 8 MB is recommended.

The directory \%HALCONROOT\%\examples\extension_package\halconuser contains two example makefiles: makefile.nt creates the example package halconuser, both for standard HALCON and for Parallel HALCON. makefile_testprog.nt creates the example applications based on the package halconuser, again for both HALCON versions. You start the making process by calling

```
 nmake -f makefile.nt
```

and

```
 nmake -f makefile_testprog.nt
```

Both makefiles include the file make.i586-nt4 which sets architecture-dependent variables.

7.2.5.2 Generating Packages Under Linux

In order to activate a package, its complete path, e.g.

```bash
$HALCONROOT/examples/extension_package/halconuser
```

must be included in the environment variable HALCONEXTENSIONS. In the UNIX version the package paths in HALCONEXTENSIONS are separated by colons. Please see also the comments on installing a HALCON package on a Linux system on page 8.

- Compiling the source code:
  In order to generate shared libraries you must produce position independent code. For the gcc/g++ compiler (version gcc 2.95) this is done using the option `-fPIC`. For the gcc we also recommend the options `-march=pentium -mcpu=pentiumpro -ansi -fno-strict-prototype.`
Creating object files for the Parallel HALCON version:
If you want to create a Parallel HALCON version of your package, a second set of object files must be created from the source code files, using the compiler define `DH_PARALLEL'. We recommend to place these object files into a separate directory to prevent confusing them with the object files created for the standard HALCON version.

Creating the operator library `libpackage.so':
To create the operator library `libpackage.so' containing new HALCON operators, the object files containing the corresponding supply and action procedures must be linked. The linkage of shared libraries is done with `ld -shared'. Specify the desired name `libpackage.so' and location of the library with the `ld' option `-o'.
To create the library `libparpackage.so' for Parallel HALCON, the corresponding object files compiled with the define `DH_PARALLEL' must be linked.

Creating the C interface library `libpackagec.so':
To create the C interface library `libpackagec.so' providing the interface to new HALCON operators, the object file `HCPackage.o' and the new operator library `libpackage.so' must be linked.
To create the library `libparpackagec.so' for Parallel HALCON, the corresponding object file compiled with the define `DH_PARALLEL' must be linked to the Parallel HALCON version of the operator library, i.e., `libparpackage.so'.

Creating the C++ interface library `libpackagecpp.so':
To create the C++ interface library `libpackagecpp.so' providing the interface to new HALCON operators, the object files `HCPPackage.o' and `HCPPackageGlobal.o', and the new operator library `libpackage.so' must be linked.
To create the library `libparpackagecpp.so' for Parallel HALCON, the corresponding object files compiled with the define `DH_PARALLEL' must be linked to the Parallel HALCON version of the operator library, i.e., `libparpackage.so'.

Creating new applications:
In order to create new application programs (written in C or C++) you must link `libpackage.so' and `libpackagec.so' or `libpackagecpp.so' to your objects (besides `libhalcon.so' and `libhalconc.so' or `libhalconc.cpp.so' as for any HALCON application). Furthermore, you must add the package library subdirectory `lib/ia64-linux2.2' to the environment variable `SLIB_PATH', otherwise the loader will fail to access the libraries.
To create a Parallel HALCON version of your application, just link the object files to the Parallel HALCON version of the libraries (e.g., `libparpackage.so', `libparpackagec.so', `libparhalcon.so', and `libparhalconc.so' in case of a C application). You can use the same object files as for creating the standard HALCON application.

The directory `$HALCONROOT/examples/extension_package/halconuser' contains two example makefiles: `makefile' creates the example package halconuser, both for standard HALCON and for Parallel HALCON. `makefile_testprog' creates the example applications based on the package halconuser, again for both HALCON versions. Both makefiles include the file `make.i586-linux2.2' which sets architecture-dependent variables.
7.2.5.3 Generating Packages Under Solaris

In order to activate a package, its complete path, e.g. 

```
$HALCONROOT/examples/extension_package/halconuser
```

must be included in the environment variable `HALCONEXTENSIONS`. In the UNIX version the package paths in `HALCONEXTENSIONS` are separated by colons. Please see also the comments on installing a HALCON package on a Solaris system on page 8.

- **Compiling the source code:**
  
  In order to generate shared libraries you must produce position independent code. For the `cc/CC` compiler this is done using the option `-KPIC`.

- **Creating object files for the Parallel HALCON version:**
  
  If you want to create a Parallel HALCON version of your package, a second set of object files must be created from the source code files, using the compiler define `-DH_PARALLEL`. We recommend to place these object files into a separate directory to prevent confusing them with the object files created for the standard HALCON version.

- **Creating the operator library `libpackage.so`:**
  
  To create the operator library `libpackage.so` containing new HALCON operators, the object files containing the corresponding supply and action procedures must be linked. The linkage of shared libraries is done with `ld -g`. Specify the desired name `libpackage.so` and location of the library with the `ld` option `-o`.

  To create the library `libparpackage.so` for Parallel HALCON, the corresponding object files compiled with the define `-DH_PARALLEL` must be linked.

- **Creating the C interface library `libpackagec.so`:**
  
  To create the C interface library `libpackagec.so` providing the interface to new HALCON operators, the object file `HCpackage.o` and the new operator library `libpackage.so` must be linked.

  To create the library `libparpackagec.so` for Parallel HALCON, the corresponding object file compiled with the define `-DH_PARALLEL` must be linked to the Parallel HALCON version of the operator library, i.e., `libparpackage.so`.

- **Creating the C++ interface library `libpackagecpp.so`:**
  
  To create the C++ interface library `libpackagecpp.so` providing the interface to new HALCON operators, the object files `HCPPpackage.o` and `HCPPpackageGlobal.o`, and the new operator library `libpackage.so` must be linked.

  To create the library `libparpackagecpp.so` for Parallel HALCON, the corresponding object files compiled with the define `-DH_PARALLEL` must be linked to the Parallel HALCON version of the operator library, i.e., `libparpackage.so`.

- **Creating new applications:**

  In order to create new application programs (written in C or C++) you must link `libpackage.so` and `libpackagec.so` or `libpackagecpp.so` to your objects (besides `libhalcon.so` and `libhalconc.so` or `libhalconcpp.so` as for any HALCON application). Furthermore, you must add the package library subdirectory `lib/sparc-sun-solaris7` to the environment variable `LD_LIBRARY_PATH`, otherwise the loader will fail to access the libraries.
To create a Parallel HALCON version of your application, just link the object files to the Parallel HALCON version of the libraries (e.g., libparpackage.so, libparpackagec.so, libparhalcon.so, and libparhalconc.so in case of a C application). You can use the same object files as for creating the standard HALCON application.

The directory $HALCONROOT/examples/extension_package/halconuser contains two example makefiles: makefile creates the example package halconuser, both for standard HALCON and for Parallel HALCON. makefile_testprog creates the example applications based on the package halconuser, again for both HALCON versions. Both makefiles include the file make.sparc-sun-solaris7 which sets architecture-dependent variables.

### 7.2.5.4 Generating Packages Under IRIX

In order to activate a package, its complete path, e.g.

```
$HALCONROOT/examples/extension_package/halconuser
```

must be included in the environment variable $HALCONEXTENSIONS. In the UNIX version the package paths in $HALCONEXTENSIONS are separated by colons. Please see also the comments on installing a HALCON package on a IRIX system on page 8.

- **Creating the operator library** libpackage.so:
  
  To create the operator library libpackage.so containing new HALCON operators the object files containing the corresponding supply and action procedures must be linked. The linkage of shared libraries is done with

  ```
  ld -check_registry /usr/lib.so_locations -shared
  ```

  Specify the desired name libpackage.so and location of the library with the ld option -0. To allow the location of such a library to be resolved by the operating system during runtime (using $LD_LIBRARY_PATH) also specify -soname libpackage.so.

- **Creating the C interface library** libpackagec.so:
  
  To create the C interface library libpackagec.so providing the interface to new HALCON operators, the object file HCPackage.o and the new operator library libpackage.so must be linked.

- **Creating the C++ interface library** libpackagecpp.so:
  
  To create the C++ interface library libpackagecpp.so providing the interface to new HALCON operators, the object files HCPPackage.o and HCPPackageGlobal.o, and the new operator lib libpackage.so must be linked.

- **Creating new applications**:
  
  In order to create new application programs (written in C or C++) you must link libpackage.so and libpackagec.so or libpackagecpp.so to your objects (besides libhalcon.so and libhalconc.so or libhalconcpp.so as for any HALCON application). Furthermore, you must add the package library subdirectory

\[\text{Of course this has to be done for libpackagec.so (-soname libpackagec.so) and libpackagecpp.so (-soname libpackagecpp.so) as well.}\]
lib/mips-sgi-irix6.5 to the environment variable LD_LIBRARY_PATH, otherwise the loader will fail to access the libraries.

The directory $HALCONROOT/examples/extension_package/halconuser contains two example makefiles: makefile creates the example package halconuser for standard HALCON. makefile_testprog creates the example applications based on the package halconuser. Both makefiles include the file make.mips-sgi-irix6.5 which sets architecture-dependent variables.

7.2.5.5 Generating Packages Under HP-UX

In order to activate a package, its complete path, e.g. $HALCONROOT/examples/extension_package/halconuser must be included in the environment variable HALCONEXTENSIONS. In the UNIX version the package paths in HALCONEXTENSIONS are separated by colons. Please see also the comments on installing a HALCON package on a HP-UX system on page 8.

- **Compiling the source code:**
  In order to generate shared libraries you must produce position independent code. For the cc/CC compiler this is done using the option +Z.

- **Creating the operator library** libpackage.sl:
  To create the operator library libpackage.sl containing new HALCON operators, the object files containing the corresponding supply and action procedures must be linked. The linkage of shared libraries is done with ld -b. Specify the desired name libpackage.sl and location of the library with the ld option -o. To allow the location of such a library to be resolved by the operating system during runtime (using SHLIB_PATH) also specify the ld option +s.

- **Creating the C interface library** libpackagec.sl:
  To create the C interface library libpackagec.sl providing the interface to new HALCON operators, the object file Hcpackage.o and the new operator library libpackagec.sl must be linked.

- **Creating the C++ interface library** libpackagexcpp.sl:
  To create the C++ interface library libpackagexcpp.sl providing the interface to new HALCON operators, the object files HCPPackage.o and HCPPackageGlobal.o, and the new operator library libpackagexcpp.sl must be linked.

- **Creating new applications:**
  In order to create new application programs (written in C or C++) you must link libpackagec.sl and libpackagexcppc.sl or libpackagexcppcpp.sl to your objects (besides libhalconc.so and libhalconcppc.so or libhalconcppcppc.sl as for any HALCON application). Furthermore, you must add the package library subdirectory lib/hppa1.1-hp-hpux10 to the environment variable SHLIB_PATH, otherwise the loader will fail to access the libraries.

3 Of course this has to be done for libpackagec.sl and libpackagexcppc.sl as well.
If you use `cc` or `CC` for linking specify the option `-Wl,+s` in order to allow the usage of the environment variable `SHLIB_PATH` for resolving the location of shared libraries by the loader.

The directory `$HALCONROOT/examples/extension_package/halconuser` contains two example makefiles: `makefile` creates the example package `halconuser` for standard HALCON. `makefile_testprogs` creates the example applications based on the package `halconuser`. Both makefiles include the file `make.hppa1.1-hp-hpuxi10` which sets architecture-dependent variables.

### 7.2.5.6 Generating Packages Under DIGITAL UNIX (Tru64 UNIX)

In order to activate a package, its complete path, e.g.

```
$HALCONROOT/examples/extension_package/halconuser
```

must be included in the environment variable `HALCONEXTENSIONS`. In the UNIX version the package paths in `HALCONEXTENSIONS` are separated by colons. Please see also the comments on installing a HALCON package on a DIGITAL UNIX (now called Tru64 UNIX) system on page 8.

- **Creating the operator library** `libpackage.so`:
  To create the operator library `libpackage.so` containing new HALCON operators, the object files containing the corresponding supply and action procedures must be linked. The linkage of shared libraries is done with

  ```
  ld -check_registry /usr/shlib/so_locations -shared -all
  ```

  Specify the desired name `libpackage.so` and location of the library with the `ld` option `-o`. In order to allow a checking of referenced symbols also include

  ```
  -none -lx11 -lc -lm -lhalcon
  ```

  at the **end** of the `ld` command (see the example makefile).

- **Creating the C interface library** `libpackagec.so`:
  To create the C interface library `libpackagec.so` providing the interface to new HALCON operators, the object file `Hpackage.o` and the new operator library `libpackagec.so` must be linked. In order to allow a checking of referenced symbols also include

  ```
  -none -lx11 -lc -lm -lhalcon -lhalconc
  ```

  at the **end** of the `ld` command (see the example makefile).

- **Creating the C++ interface library** `libpackagecpp.so`:
  To create the C++ interface library `libpackagecpp.so` providing the interface to new HALCON operators, the object files `HCPPpackage.o` and `HCPPpackageGlobal.o`, and the new operator lib `libpackagec.so` must be linked. In order to allow a checking of referenced symbols also include

  ```
  -none -lx11 -lcxx -lm -lhalcon -lhalconcpp
  ```
7.3 HALCON DIRECTORIES

at the end of the ld command (see the example makefile).

- Creating new applications:
  In order to create new application programs (written in C or C++) you must link
  libpackage.so and libpackagec.so or libpackagecpp.so to your objects (be-
  sides libhalcon.so and libhalconc.so or libhalconcpp.so as for any HAL-
  CON application). Furthermore, you must add the package library subdirectory
  lib/alpha-dec-osf4.0 to the environment variable SHLIB_PATH, otherwise the loader
  will fail to access the libraries.

The directory $HALCONROOT/examples/extension_package/halconuser contains two ex-
ample makefiles: makefile creates the example package halconuser for standard HALCON.
makefile_testprog creates the example applications based on the package halconuser. Both
makefiles include the file make.alpha-dec-osf4.0 which sets architecture-dependent vari-
ables.

7.3 HALCON Directories

HALCON always presumes that the environment variable HALCONROOT contains the path of the
HALCON homedirectory. Starting at this homedirectory the following directories are important
for creating a new system:

examples\extension_package\halconuser: Example for a HALCON extension package.
include: HALCON include files for compiling.
lib\%ARCHITECTURE%: HALCON libraries.
bin\%ARCHITECTURE%: HALCON compiler hcomp, under Windows NT/ 2000 also the HAL-
CON DLLs.
doc\macros: LATEX files for generating PostScript reference files or manuals.
Appendix A

HALCON Error Codes

In this section all HALCON error codes are summarized, see also section 3.6.

<table>
<thead>
<tr>
<th>Error Name</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>H_ERR_WIPT1</td>
<td>1201</td>
<td>Wrong type of control parameter: 1</td>
</tr>
<tr>
<td>H_ERR_WIPT2</td>
<td>1202</td>
<td>Wrong type of control parameter: 2</td>
</tr>
<tr>
<td>H_ERR_WIPT3</td>
<td>1203</td>
<td>Wrong type of control parameter: 3</td>
</tr>
<tr>
<td>H_ERR_WIPT4</td>
<td>1204</td>
<td>Wrong type of control parameter: 4</td>
</tr>
<tr>
<td>H_ERR_WIPT5</td>
<td>1205</td>
<td>Wrong type of control parameter: 5</td>
</tr>
<tr>
<td>H_ERR_WIPT6</td>
<td>1206</td>
<td>Wrong type of control parameter: 6</td>
</tr>
<tr>
<td>H_ERR_WIPT7</td>
<td>1207</td>
<td>Wrong type of control parameter: 7</td>
</tr>
<tr>
<td>H_ERR_WIPT8</td>
<td>1208</td>
<td>Wrong type of control parameter: 8</td>
</tr>
<tr>
<td>H_ERR_WIPT9</td>
<td>1209</td>
<td>Wrong type of control parameter: 9</td>
</tr>
<tr>
<td>H_ERR_WIPT10</td>
<td>1210</td>
<td>Wrong type of control parameter: 10</td>
</tr>
<tr>
<td>H_ERR_WIPT11</td>
<td>1211</td>
<td>Wrong type of control parameter: 11</td>
</tr>
<tr>
<td>H_ERR_WIPT12</td>
<td>1212</td>
<td>Wrong type of control parameter: 12</td>
</tr>
<tr>
<td>H_ERR_WIPT13</td>
<td>1213</td>
<td>Wrong type of control parameter: 13</td>
</tr>
<tr>
<td>H_ERR_WIPT14</td>
<td>1214</td>
<td>Wrong type of control parameter: 14</td>
</tr>
<tr>
<td>H_ERR_WIPT15</td>
<td>1215</td>
<td>Wrong type of control parameter: 15</td>
</tr>
<tr>
<td>H_ERR_WIPT16</td>
<td>1216</td>
<td>Wrong type of control parameter: 16</td>
</tr>
<tr>
<td>H_ERR_WIPV1</td>
<td>1301</td>
<td>Wrong value of control parameter: 1</td>
</tr>
<tr>
<td>H_ERR_WIPV2</td>
<td>1302</td>
<td>Wrong value of control parameter: 2</td>
</tr>
<tr>
<td>H_ERR_WIPV3</td>
<td>1303</td>
<td>Wrong value of control parameter: 3</td>
</tr>
<tr>
<td>H_ERR_WIPV4</td>
<td>1304</td>
<td>Wrong value of control parameter: 4</td>
</tr>
<tr>
<td>H_ERR_WIPV5</td>
<td>1305</td>
<td>Wrong value of control parameter: 5</td>
</tr>
<tr>
<td>H_ERR_WIPV6</td>
<td>1306</td>
<td>Wrong value of control parameter: 6</td>
</tr>
<tr>
<td>H_ERR_WIPV7</td>
<td>1307</td>
<td>Wrong value of control parameter: 7</td>
</tr>
<tr>
<td>H_ERR_WIPV8</td>
<td>1308</td>
<td>Wrong value of control parameter: 8</td>
</tr>
<tr>
<td>H_ERR_WIPV9</td>
<td>1309</td>
<td>Wrong value of control parameter: 9</td>
</tr>
<tr>
<td>H_ERR_WIPV10</td>
<td>1310</td>
<td>Wrong value of control parameter: 10</td>
</tr>
<tr>
<td>H_ERR_WIPV11</td>
<td>1311</td>
<td>Wrong value of control parameter: 11</td>
</tr>
<tr>
<td>H_ERR_WIPV12</td>
<td>1312</td>
<td>Wrong value of control parameter: 12</td>
</tr>
<tr>
<td>H_ERR_WIPV13</td>
<td>1313</td>
<td>Wrong value of control parameter: 13</td>
</tr>
<tr>
<td>H_ERR_WIPV14</td>
<td>1314</td>
<td>Wrong value of control parameter: 14</td>
</tr>
<tr>
<td>H_ERR_WIPV15</td>
<td>1315</td>
<td>Wrong value of control parameter: 15</td>
</tr>
</tbody>
</table>
APPENDIX A. HALCON ERROR CODES

H_ERR_WIPV16  1316  Wrong value of control parameter: 16
H_ERR_WCOMP   1350  Wrong value of component (see reset_obj_db())
H_ERR_WGCOMP   1351  Wrong value of gray value component (see reset_obj_db())
H_ERR_WIPN1    1401  Wrong number of values of control parameter: 1
H_ERR_WIPN2    1402  Wrong number of values of control parameter: 2
H_ERR_WIPN3    1403  Wrong number of values of control parameter: 3
H_ERR_WIPN4    1404  Wrong number of values of control parameter: 4
H_ERR_WIPN5    1405  Wrong number of values of control parameter: 5
H_ERR_WIPN6    1406  Wrong number of values of control parameter: 6
H_ERR_WIPN7    1407  Wrong number of values of control parameter: 7
H_ERR_WIPN8    1408  Wrong number of values of control parameter: 8
H_ERR_WIPN9    1409  Wrong number of values of control parameter: 9
H_ERR_WIPN10   1410  Wrong number of values of control parameter: 10
H_ERR_WIPN11   1411  Wrong number of values of control parameter: 11
H_ERR_WIPN12   1412  Wrong number of values of control parameter: 12
H_ERR_WIPN13   1413  Wrong number of values of control parameter: 13
H_ERR_WIPN14   1414  Wrong number of values of control parameter: 14
H_ERR_WIPN15   1415  Wrong number of values of control parameter: 15
H_ERR_WIPN16   1416  Wrong number of values of control parameter: 16
H_ERR_IONTB    1500  Number of input objects too big
H_ERR_WION1    1501  Wrong number of values of object parameter: 1
H_ERR_WION2    1502  Wrong number of values of object parameter: 2
H_ERR_WION3    1503  Wrong number of values of object parameter: 3
H_ERR_WION4    1504  Wrong number of values of object parameter: 4
H_ERR_WION5    1505  Wrong number of values of object parameter: 5
H_ERR_WION6    1506  Wrong number of values of object parameter: 6
H_ERR_WION7    1507  Wrong number of values of object parameter: 7
H_ERR_WION8    1508  Wrong number of values of object parameter: 8
H_ERR_WION9    1509  Wrong number of values of object parameter: 9
H_ERR_QONTB    1510  Number of output objects too big
H_ERR_WNP      2000  Wrong specification of parameter (error in file: xxx.def)
H_ERR_HONI     2001  Initialize Halcon: reset_obj_db(Width,Height,Components)
H_ERR_WRKNN    2002  Used number of symbolic object names too big
H_ERR_NO_LICENSE  2003  No license found
H_ERR_NO_LS_SERVER  2004  Lost connection to license server
H_ERR_NO_MODULES  2005  No modules in license (no VENDOR_STRING)
H_ERR_NO_LIC_OPER 2006  No license for this operator
H_ERR_FLEX_LM_BAD_TZ  2007  Time zone offset from GMT is > 24 hours
H_ERR_FLEX_LM_BADPLATFORM  2008  Vendor keys do not support this platform
H_ERR_LM_BADKEYDATA  2009  Bad vendor keys
H_ERR_FLEX_LM_BADVENDORDATA  2010  Unknown vendor key type
H_ERR_FLEX_LM_CANTMALLOC  2011  malloc() call failed
H_ERR_FLEX_LM_EXPIREDKEYS  2012  Vendor keys have expired
H_ERR_FLEX_LM_FUNCNOTALAILABLE  2013  Second call to lc_init() (multiple jobs), and vendor keys do not support multiple jobs
H_ERR_FLEX_LM_NOKEYDATA  2014  Vendor key data not supplied
 HALCON 6.0
<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2055</td>
<td>Cannot find ethernet device</td>
</tr>
<tr>
<td>2056</td>
<td>Cannot read license file</td>
</tr>
<tr>
<td>2057</td>
<td>Feature not yet available (wrong time/date set?)</td>
</tr>
<tr>
<td>2059</td>
<td>Clock difference too large between client and server</td>
</tr>
<tr>
<td>2060</td>
<td>Feature database corrupted in daemon</td>
</tr>
<tr>
<td>2061</td>
<td>Duplicate selection mismatch for this feature</td>
</tr>
<tr>
<td>2062</td>
<td>User/host on EXCLUDE list for feature</td>
</tr>
<tr>
<td>2063</td>
<td>User/host not on INCLUDE list for feature</td>
</tr>
<tr>
<td>2064</td>
<td>Feature was never checked out</td>
</tr>
<tr>
<td>2065</td>
<td>Invalid FLEXlm key data supplied</td>
</tr>
<tr>
<td>2066</td>
<td>Clock setting check not available in daemon</td>
</tr>
<tr>
<td>2067</td>
<td>Date too late for binary format</td>
</tr>
<tr>
<td>2068</td>
<td>FLEXlm not initialized</td>
</tr>
<tr>
<td>2069</td>
<td>Server did not respond to message</td>
</tr>
<tr>
<td>2070</td>
<td>Request rejected by vendor-defined filter</td>
</tr>
<tr>
<td>2071</td>
<td>No FEATURESET line present in license file</td>
</tr>
<tr>
<td>2072</td>
<td>Incorrect FEATURESET line in license file</td>
</tr>
<tr>
<td>2073</td>
<td>Cannot compute FEATURESET line</td>
</tr>
<tr>
<td>2074</td>
<td>socket() call failed</td>
</tr>
<tr>
<td>2075</td>
<td>setsockopt() failed</td>
</tr>
<tr>
<td>2076</td>
<td>Message checksum failure</td>
</tr>
<tr>
<td>2077</td>
<td>Cannot read license file from server</td>
</tr>
<tr>
<td>2078</td>
<td>Not a license administrator</td>
</tr>
<tr>
<td>2079</td>
<td>Inremove request too soon</td>
</tr>
<tr>
<td>2080</td>
<td>Attempt to read beyond the end of LF path</td>
</tr>
<tr>
<td>2081</td>
<td>SYS$SETIMR call failed</td>
</tr>
<tr>
<td>2082</td>
<td>Internal FLEXlm Error - Please report to Globetrotter Software</td>
</tr>
<tr>
<td>2083</td>
<td>FLEXadmin API functions not available</td>
</tr>
<tr>
<td>2084</td>
<td>Invalid PACKAGE line in license file</td>
</tr>
<tr>
<td>2085</td>
<td>Server FLEXlm version older than client’s</td>
</tr>
<tr>
<td>2086</td>
<td>Incorrect number of USERS/HOSTS INCLUDED in options file – see server log</td>
</tr>
<tr>
<td>2087</td>
<td>Server doesn’t support this request</td>
</tr>
<tr>
<td>2088</td>
<td>This license object already in use</td>
</tr>
<tr>
<td>2100</td>
<td>Wrong index for output object parameter</td>
</tr>
<tr>
<td>2101</td>
<td>Wrong index for input object parameter</td>
</tr>
<tr>
<td>2102</td>
<td>Wrong index for image object (too big or too small)</td>
</tr>
<tr>
<td>2103</td>
<td>Wrong number region/image component (see: HGetComp)</td>
</tr>
<tr>
<td>2104</td>
<td>Wrong relation name</td>
</tr>
<tr>
<td>2105</td>
<td>Access to undefined gray value component</td>
</tr>
<tr>
<td>2106</td>
<td>Wrong image width</td>
</tr>
<tr>
<td>2107</td>
<td>Wrong image height</td>
</tr>
<tr>
<td>2108</td>
<td>Undefined gray value component</td>
</tr>
<tr>
<td>2200</td>
<td>Inconsistent data of data base (typing)</td>
</tr>
<tr>
<td>2201</td>
<td>Wrong index for input control parameter</td>
</tr>
</tbody>
</table>
H_ERR_DBDU 2202 Data of data base not defined (internal error)
H_ERR_FNTL 2203 Number of operators too big
H_ERR_WUNDI 2204 System error in CORE2: wrong setting of H_UNDEFINED
H_ERR_UEXTNI 2205 User extension not properly installed
H_ERR_NPTL 2206 Number of packages too large
H_ERR_NSP 2207 No such package installed
H_ERR_GV_WA 2830 Wrong access to global variable
H_ERR_GV_NC 2831 Used global variable does not exist
H_ERR_GV_NG 2832 Used global variable not accessible via GLOBAL_ID
H_ERR_HM_NT 2835 Halcon server to terminate is still working on a job
H_ERR_HM_NA 2837 No such HALCON software agent
H_ERR_AG_CN 2838 Hardware check for parallelization not possible on a single-processor machine
H_ERR_AG_NC 2839 (Seq.) HALCON does not support parallel hardware check (use Parallel HALCON instead)
H_ERR_AG_IN 2840 Initialization of agent failed
H_ERR_AG_NT 2841 Termination of agent failed
H_ERR_AG_HW 2842 Inconsistent hardware description file
H_ERR_AG_II 2843 Inconsistent agent information file
H_ERR_AG_IK 2844 Inconsistent agent knowledge file
H_ERR_AG_WV 2845 The file with the parallelization information does not match to the currently HALCON version/revision
H_ERR_AG_WH 2846 The file with the parallelization information does not match to the currently used machine
H_ERR_AG_KC 2847 Inconsistent knowledge base of HALCON software agent
H_ERR_AG_CT 2848 Unknown communication type
H_ERR_AG_MT 2849 Unknown message type for HALCON software agent
H_ERR_AG_WK 2850 Error while saving the parallelization knowledge
H_ERR_AG_WW 2851 Wrong type of work information
H_ERR_AG_WA 2852 Wrong type of application information
H_ERR_AG_WE 2853 Wrong type of experience information
H_ERR_AG_NU 2854 Unknown name of HALCON software agent
H_ERR_AG_NE 2855 Unknown name and communication address of HALCON software agent
H_ERR_AG_RR 2856 cpu representative (HALCON software agent) not reachable
H_ERR_AG_Cr 2857 cpu refuses work
H_ERR_AG_RN 2858 Description of scheduling resource not found
H_ERR_AG_TILT 2859 Not accessible function of HALCON software agent
H_ERR_WRT 2860 Wrong type: HALCON scheduling resource
H_ERR_WRS 2861 Wrong state: HALCON scheduling resource
H_ERR_UNKPT 2862 Unknown parameter type: HALCON scheduling resource
H_ERR_UNKPVAL 2863 Unknown parameter value: HALCON scheduling resource
H_ERR_CTRL_WPP 2864 Wrong post processing of control parameter
H_ERR_GETTI 2867 Error while trying to get time (time query)
H_ERR_GETCPUNUM 2868 Error while trying to get the number of processors
APPENDIX A. HALCON ERROR CODES

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>H_ERR_TMPFN</td>
<td>Error while accessing temporary file</td>
</tr>
<tr>
<td>H_ERRPTHRD_CR</td>
<td>creation of pthread failed</td>
</tr>
<tr>
<td>H_ERRPTHRD_DT</td>
<td>pthread-detach failed</td>
</tr>
<tr>
<td>H_ERRPTHRD_J0</td>
<td>pthread-join failed</td>
</tr>
<tr>
<td>H_ERRPTHRD_MI</td>
<td>initialization of mutex variable failed</td>
</tr>
<tr>
<td>H_ERRPTHRD_MD</td>
<td>deletion of mutex variable failed</td>
</tr>
<tr>
<td>H_ERRPTHRD_ML</td>
<td>Lock of mutex variable failed</td>
</tr>
<tr>
<td>H_ERRPTHRD_MU</td>
<td>unlock of mutex variable failed</td>
</tr>
<tr>
<td>H_ERRPTHRD_CS</td>
<td>failed to signal pthread condition variable</td>
</tr>
<tr>
<td>H_ERRPTHRD_CW</td>
<td>failed to wait for pthread condition variable</td>
</tr>
<tr>
<td>H_ERRPTHRED_CI</td>
<td>failed to init pthread condition variable</td>
</tr>
<tr>
<td>H_ERR_DCDG_FILE</td>
<td>Tried to leave 'critical section' without entering it before</td>
</tr>
<tr>
<td>H_ERR_M_SGCN</td>
<td>failed to give free a handle via CloseHandle</td>
</tr>
<tr>
<td>H_ERR_M_SCSAI</td>
<td>'Free list' is empty while scheduling</td>
</tr>
<tr>
<td>H_ERR_M_SCSN</td>
<td>Communication partner not checked in</td>
</tr>
<tr>
<td>H_ERR_M_SCSNI</td>
<td>you can not start the communication system while running it</td>
</tr>
<tr>
<td>H_ERR_R_IEI</td>
<td>Region (partially) outside of the definition range of the image</td>
</tr>
<tr>
<td>H_ERR_EDEF</td>
<td>Intersected definition range region / image empty</td>
</tr>
<tr>
<td>H_ERR_T_IEI</td>
<td>Image with empty definition range (=&gt; no gray values)</td>
</tr>
<tr>
<td>H_ERR_FLTS</td>
<td>No common image point of two images</td>
</tr>
<tr>
<td>H_ERR_LNLTB</td>
<td>Wrong region for image (first row &lt; 0)</td>
</tr>
<tr>
<td>H_ERR_LNLTB</td>
<td>Wrong region for image (column in last row &gt;= image width)</td>
</tr>
<tr>
<td>H_ERR_UENUI</td>
<td>Number of images unequal in input parameters</td>
</tr>
<tr>
<td>H_ERR_HTS</td>
<td>Image height too small</td>
</tr>
<tr>
<td>H_ERR_VTS</td>
<td>Image width too small</td>
</tr>
<tr>
<td>H_ERR_CHSSEG</td>
<td>Internal error: multiple call of HRLInitSeg()</td>
</tr>
<tr>
<td>H_ERR_HSEG1</td>
<td>Internal error: HRLSeg() not initialized</td>
</tr>
<tr>
<td>H_ERR_VGAUSSM</td>
<td>Wrong size of filter for Gauss</td>
</tr>
<tr>
<td>H_ERR_FSEIS</td>
<td>Filter size exceeds image size</td>
</tr>
<tr>
<td>H_ERR_R0FTB</td>
<td>Row value of a coordinate &gt; 2^16</td>
</tr>
<tr>
<td>H_ERR_R0FTS</td>
<td>Row value of a coordinate &lt; -2^16</td>
</tr>
<tr>
<td>H_ERR_COLTB</td>
<td>Column value of a coordinate &gt; 2^16</td>
</tr>
<tr>
<td>H_ERR_COLTS</td>
<td>Column value of a coordinate &lt; -2^16</td>
</tr>
<tr>
<td>H_ERR_VTHTR</td>
<td>Wrong segmentation threshold</td>
</tr>
<tr>
<td>H_ERR_UNKF</td>
<td>Unknown attribute of a region</td>
</tr>
<tr>
<td>H_ERR_UNKG</td>
<td>Unknown attribute of a gray value</td>
</tr>
<tr>
<td>H_ERR_EINCC</td>
<td>Internal error in HContCut</td>
</tr>
<tr>
<td>H_ERR_EINCP</td>
<td>Error in HContToPol: distance of points too big</td>
</tr>
<tr>
<td>H_ERR_EINCP2</td>
<td>Error in HContToPol: contour too long</td>
</tr>
<tr>
<td>H_ERR_TMR</td>
<td>Too many rows (IPImageTransform)</td>
</tr>
<tr>
<td>H_ERR_SFZ</td>
<td>Scaling factor = 0.0 (IPImageScale)</td>
</tr>
<tr>
<td>H_ERR_QOR</td>
<td>Wrong range in transformation matrix</td>
</tr>
<tr>
<td>H_ERR_NEF</td>
<td>Internal error in IPvvf: no element free</td>
</tr>
<tr>
<td>H_ERR_NQDB</td>
<td>Number of input objects is zero</td>
</tr>
</tbody>
</table>
119

H_ERR_EMPOb 3111 At least one input object has got an empty region
H_ERR_NFOT 3112 Operation allowed for rectangular images 2**n only
H_ERR_TMPH 3113 Too many relevant points (IPHysteresis)
H_ERR_LTB 3114 Number of labels in image too big
H_ERR_HNLA 3115 No labels with negative values allowed
H_ERR_WFS 3116 Wrong filter size (too small ?)
H_ERR_LDS 3117 Images with different image size
H_ERR_IWTL 3118 Target image too wide or too far on the right
H_ERR_IWTS 3119 Target image too narrow or too far on the left
H_ERR_IHTL 3120 Target image too high or too far down
H_ERR_IHTS 3121 Target image too low or too far up
H_ERR_DNOQ 3122 Number of channels in the input parameters are different
H_ERR_COrts 3131 Coocurrence Matrix: too little columns for quantisation
H_ERR_Cogts 3132 Coocurrence Matrix: too little rows for quantisation
H_ERR_NUM_Column 3133 Wrong number of columns
H_ERR_NUM_LINES 3134 Wrong number of rows
H_ERR_OVL 3135 Number has too many digits
H_ERR_NUM_SYM 3136 Matrix is not symmetric
H_ERR_NUM_CODL 3137 Matrix is too big
H_ERR_SYNTAX 3138 Wrong structure of file
H_ERR_MISSING 3139 Lesser than 2 matrices
H_ERR_COOC_MEM 3140 Not enough memory
H_ERR_NO_FILE 3141 Can not read the file
H_ERR_FILE_WR 3142 Can not open file for writing
H_ERR_NUM_LUCOLS 3143 Too many lookup table colors
H_ERR_WGQLI 3145 Too many Hough points (lines)
H_ERR_DITS 3146 Target image has got wrong height (not big enough)
H_ERR_WINTM 3147 Wrong interpolation mode
H_ERR_THICK_NK 3148 Region not compact or not connected
H_ERR_WIND3 3170 Wrong filter index for filter size 3
H_ERR_WIND5 3171 Wrong filter index for filter size 5
H_ERR_WIND7 3172 Wrong filter index for filter size 7
H_ERR_WLAWSS 3173 Wrong filter size; only 3/5/7
H_ERR_WNEE 3200 Different number of entries/exits in HContCut
H_ERR_XLDWT 3250 Wrong XLD type
H_ERR_XLD_CTL 3251 Internal error: contour too long for representation
H_ERR_XLD_RPF 3252 Internal error: border point is set to FG
H_ERR_XLD_MCL 3253 Internal error: maximum contour length exceeded
H_ERR_XLD_MCN 3254 Internal error: maximum number of contours exceeded
H_ERR_XLD_CTS 3255 Contour too short for fetch_angleXld
H_ERR_XLD_CRD 3256 Regression parameters of contours already computed
H_ERR_XLD_CRD 3257 Regression parameters of contours not yet entered!
      Please compute them by calling regress_angleXld
H_ERR_DBXCC 3258 Data base: XLD object has been deleted
H_ERR_DBMXID 3259 Data base: object has no XLD-ID
H_ERR_XLD_MINP 3260 Internal error: wrong number of contour points allocated
H_ERR_XLD_CAND 3261 Contour attribute not defined

HALCON 6.0
<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>H_ERR_FIT_ELLIPSE</td>
<td>3262 Ellipse fitting failed</td>
</tr>
<tr>
<td>H_ERR_FIT_CIRCLE</td>
<td>3263 Circle fitting failed</td>
</tr>
<tr>
<td>H_ERR_NWF</td>
<td>3275 No ARC/INFO world file</td>
</tr>
<tr>
<td>H_ERR_NAIGF</td>
<td>3276 No ARC/INFO generate file</td>
</tr>
<tr>
<td>H_ERR_XLDS_ISOL_POINT</td>
<td>3290 Isolated point while contour merging</td>
</tr>
<tr>
<td>H_ERR_SESF</td>
<td>3300 Syntax error in file for training</td>
</tr>
<tr>
<td>H_ERR_TMFE</td>
<td>3301 Maximum number of attributes per example exceeded</td>
</tr>
<tr>
<td>H_ERR_OPSF</td>
<td>3302 Not possible to open file for training</td>
</tr>
<tr>
<td>H_ERR_TMSS</td>
<td>3303 Too many data sets for training</td>
</tr>
<tr>
<td>H_ERR_WSKY</td>
<td>3304 Wrong key for data for training</td>
</tr>
<tr>
<td>H_ERR_TMSAM</td>
<td>3305 Too many examples for one data set for training</td>
</tr>
<tr>
<td>H_ERR_TMCLS</td>
<td>3306 Too many classes</td>
</tr>
<tr>
<td>H_ERR_TMBOX</td>
<td>3307 Maximum number of cuboids exceeded</td>
</tr>
<tr>
<td>H_ERR_OPCF</td>
<td>3308 Not possible to open classifikator’s file</td>
</tr>
<tr>
<td>H_ERR_SCLA</td>
<td>3309 Error while saving the classifikator</td>
</tr>
<tr>
<td>H_ERR_OPF</td>
<td>3310 Not possible to open protocol file</td>
</tr>
<tr>
<td>H_ERR_CLEX</td>
<td>3311 Classifikator with this name is already existent</td>
</tr>
<tr>
<td>H_ERR_TMCLA</td>
<td>3312 Maximum number of classifikators exceeded</td>
</tr>
<tr>
<td>H_ERR_CNTL</td>
<td>3313 Name of classifikator is too long, &gt;= 20</td>
</tr>
<tr>
<td>H_ERR_CLMNCF</td>
<td>3314 Classifikator with this name is not existent</td>
</tr>
<tr>
<td>H_ERR_NOCLA</td>
<td>3315 Current classifikator is not defined</td>
</tr>
<tr>
<td>H_ERR_CLASS2_ID</td>
<td>3316 Wrong id in classification file</td>
</tr>
<tr>
<td>H_ERR_CLASS2_VERS</td>
<td>3317 Wrong version number in classification file</td>
</tr>
<tr>
<td>H_ERR_ROTNR</td>
<td>3401 Wrong rotation number</td>
</tr>
<tr>
<td>H_ERR_GOL</td>
<td>3402 Wrong letter for Golay element</td>
</tr>
<tr>
<td>H_ERR_BEZ</td>
<td>3403 Wrong reference point</td>
</tr>
<tr>
<td>H_ERR_ITER</td>
<td>3404 Wrong number of iterations</td>
</tr>
<tr>
<td>H_ERR_MOSYS</td>
<td>3405 Morphology: system error</td>
</tr>
<tr>
<td>H_ERR_ART</td>
<td>3406 Wrong type of boundary</td>
</tr>
<tr>
<td>H_ERR_OBJI</td>
<td>3407 Morphology: wrong number of input objects</td>
</tr>
<tr>
<td>H_ERR_OBJO</td>
<td>3408 Morphology: wrong number of output objects</td>
</tr>
<tr>
<td>H_ERR_PARI</td>
<td>3409 Morphology: wrong number of input control parameter</td>
</tr>
<tr>
<td>H_ERR_PARO</td>
<td>3410 Morphology: wrong number of output control parameter</td>
</tr>
<tr>
<td>H_ERR_SELCE</td>
<td>3411 Morphology: structuring element is infinite</td>
</tr>
<tr>
<td>H_ERR_WRESE</td>
<td>3412 Morphology: wrong name for structuring element</td>
</tr>
<tr>
<td>H_ERR_WRRLNI1</td>
<td>3500 Wrong number of run length rows (chords): smaller than 0</td>
</tr>
<tr>
<td>H_ERR_WRRRL</td>
<td>3502 Run length row with negative length</td>
</tr>
<tr>
<td>H_ERR_RLLTB</td>
<td>3503 Run length row &gt;= image height</td>
</tr>
<tr>
<td>H_ERR_RLLTS</td>
<td>3504 Run length row &lt; 0</td>
</tr>
<tr>
<td>H_ERR_RLCTB</td>
<td>3505 Run length column &gt;= image width</td>
</tr>
<tr>
<td>H_ERR_RLCSTS</td>
<td>3506 Run length column &lt; 0</td>
</tr>
<tr>
<td>H_ERR_CHLTB</td>
<td>3507 For CHORD_TYPE: Number of row too big</td>
</tr>
<tr>
<td>H_ERR_CHLTS</td>
<td>3508 For CHORD_TYPE: Number of row too small</td>
</tr>
<tr>
<td>H_ERR_CHCTB</td>
<td>3509 For CHORD_TYPE: Number of column too big</td>
</tr>
<tr>
<td>H_ERR_MRELE</td>
<td>3510 Exceeding the maximum number of run lengths while automatical expansion</td>
</tr>
</tbody>
</table>
H_ERR_ICOMPL 3511 Internal error: Region->compl neither TRUE/FALSE
H_ERR_RLEMAX 3512 Internal error: Region->max_num < Region->num
H_ERR_WRLIN3 3513 Internal error: number of chords too big for num_max
H_ERR_WIMA1 3520 Image width < 0
H_ERR_WIMA2 3521 Image width >= MAX_FORMAT
H_ERR_WIMA1H 3522 Image height < 0
H_ERR_WIMA2H 3523 Image height >= MAX_FORMAT
H_ERR_WIMAH2 3524 Image width <= 0
H_ERR_WIMAH3 3525 Image height <= 0
H_ERR_TMS 3550 Too many segments
H_ERR_SING 3850 The light source positions are linearly dependent
H_ERR_FEWIM 3851 No sufficient image indication
H_ERR_ZBR_NOS 3852 Internal error: Function has equal signs in HZBrent
H_ERR_NFILE 3900 Kalman: Dimension n,m or p has got a undefined value
H_ERR_FF1 3901 Kalman: File does not exist
H_ERR_FF2 3902 Kalman: Error in file (row of dimension)
H_ERR_FF3 3903 Kalman: Error in file (row of marking)
H_ERR_NO_A 3904 Kalman: Error in file (value is no float)
H_ERR_NO_C 3905 Kalman: Matrix A is missing in file
H_ERR_NO_Q 3906 Kalman: Matrix C is missing in file
H_ERR_NO_R 3907 Kalman: Matrix Q is missing in file
H_ERR_NO_Gu 3908 Kalman: Matrix R is missing in file
H_ERR_NOTSYMM 3909 Kalman: G or u is missing in file
H_ERR_SINGU 3910 Kalman: Covariant matrix is not symmetric
H_ERR_DB0IT 4050 Image data management: object is a object tupel
H_ERR_DB0C 4051 Image data management: object has been deleted already
H_ERR_DB0UID 4052 Image data management: wrong object-ID
H_ERR_DB0TC 4053 Image data management: object tupel has been deleted already
H_ERR_DBWTID 4054 Image data management: wrong object tupel-ID
H_ERR_DB1IT 4055 Image data management: object tupel is a object
H_ERR_DB1DNULL 4056 Image data management: object-ID is NULL (0)
H_ERR_WDBID 4057 Image data management: object-ID outside the valid range
H_ERR_DB1C 4058 Image data management: access to deleted image
H_ERR_DB1IID 4059 Image data management: access to image with wrong key
H_ERR_DB1RC 4060 Image data management: access to deleted region
H_ERR_DB1RUID 4061 Image data management: access to region with wrong key
H_ERR_WCHAN 4062 Image data management: wrong value for image channel
H_ERR_DB1ITL 4063 Image data management: index too big
H_ERR_DB1UNDEF 4064 Image data management: index not defined
H_ERR_DB1ICT 4065 Internal error image data management: wrong clear type
H_ERR_WSCL 5100 Wrong (logical) window number
H_ERR_DSCL0 5101 Error while opening the window
H_ERR_WECL 5102 Wrong window coordinates
H_ERR_NWA 5103 It is not possible to open another window
<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>H_ERR_DNA</td>
<td>5104 Device resp. operator not available</td>
</tr>
<tr>
<td>H_ERR_UCOL</td>
<td>5105 Unknown color</td>
</tr>
<tr>
<td>H_ERR_NMW</td>
<td>5106 No window has been opened for desired action</td>
</tr>
<tr>
<td>H_ERR_VFM</td>
<td>5107 Wrong filling mode for regions (fill or margin)</td>
</tr>
<tr>
<td>H_ERR_WGV</td>
<td>5108 Wrong gray value (0..255)</td>
</tr>
<tr>
<td>H_ERR_WPV</td>
<td>5109 Wrong pixel value (use value of get pixel(P) only)</td>
</tr>
<tr>
<td>H_ERR_KLV</td>
<td>5110 Wrong line width (see: query line_width(Min,Max))</td>
</tr>
<tr>
<td>H_ERR_CUR</td>
<td>5111 Wrong name of cursor (see: query mshape(Name))</td>
</tr>
<tr>
<td>H_ERR_VLT</td>
<td>5112 Wrong color table (see: query_Lut(Name))</td>
</tr>
<tr>
<td>H_ERR_WDM</td>
<td>5113 Wrong representation mode (see: query_insert(Mode))</td>
</tr>
<tr>
<td>H_ERR_VRCO</td>
<td>5114 Wrong representation color (see: query_color(List))</td>
</tr>
<tr>
<td>H_ERR_WRD</td>
<td>5115 Wrong dither matrix (binary image representation)</td>
</tr>
<tr>
<td>H_ERR_WIT</td>
<td>5116 Wrong image transformation (name or image size)</td>
</tr>
<tr>
<td>H_ERR_WRZ</td>
<td>5117 Unsuitable image type for image transformation</td>
</tr>
<tr>
<td>H_ERR_WRD</td>
<td>5118 Wrong zooming factor for image transformation</td>
</tr>
<tr>
<td>H_ERR_WDV</td>
<td>5119 Wrong representation mode</td>
</tr>
<tr>
<td>H_ERR_WINF</td>
<td>5120 Wrong code of device</td>
</tr>
<tr>
<td>H_ERR_WDEXT</td>
<td>5121 Wrong number for father window</td>
</tr>
<tr>
<td>H_ERR_WDT</td>
<td>5122 Wrong window size</td>
</tr>
<tr>
<td>H_ERR_WTT</td>
<td>5123 Wrong window type</td>
</tr>
<tr>
<td>H_ERR_WND</td>
<td>5124 No current window has been set</td>
</tr>
<tr>
<td>H_ERR_WRGB</td>
<td>5125 Wrong color combination or range (RGB)</td>
</tr>
<tr>
<td>H_ERR_WPS</td>
<td>5126 Wrong number of pixels set</td>
</tr>
<tr>
<td>H_ERR_WCM</td>
<td>5127 Wrong value for comprise (object or image)</td>
</tr>
<tr>
<td>H_ERR_FNA</td>
<td>5128 set_fix with 1/4 image levels and static not valid</td>
</tr>
<tr>
<td>H_ERR_LNFS</td>
<td>5129 set_lut not valid in child windows</td>
</tr>
<tr>
<td>H_ERR_LOFL</td>
<td>5130 Number of concurrent used color tables is too big</td>
</tr>
<tr>
<td>H_ERR_WDIT</td>
<td>5131 Wrong device for window dump</td>
</tr>
<tr>
<td>H_ERR_WDS</td>
<td>5132 Wrong window size for window dump</td>
</tr>
<tr>
<td>H_ERR_NDVS</td>
<td>5133 System variable DISPLAY (setenv) not defined</td>
</tr>
<tr>
<td>H_ERR_WBW</td>
<td>5134 Wrong thickness for window margin</td>
</tr>
<tr>
<td>H_ERR_WDV</td>
<td>5135 System variable DISPLAY has been set wrong (&lt;host&gt;:0.0)</td>
</tr>
<tr>
<td>H_ERR_TMF</td>
<td>5136 Too many fonts loaded</td>
</tr>
<tr>
<td>H_ERR_WFN</td>
<td>5137 Wrong font name</td>
</tr>
<tr>
<td>H_ERR_WCP</td>
<td>5138 No valid cursor position</td>
</tr>
<tr>
<td>H_ERR_NTW</td>
<td>5139 Window is not a textual window</td>
</tr>
<tr>
<td>H_ERR_NPW</td>
<td>5140 Window is not a image window</td>
</tr>
<tr>
<td>H_ERR_STL</td>
<td>5141 String too long or too high</td>
</tr>
<tr>
<td>H_ERR_NSS</td>
<td>5142 Too little space in the window rightwards</td>
</tr>
<tr>
<td>H_ERR_RMS</td>
<td>5143 Window is not suitable for the mouse</td>
</tr>
<tr>
<td>H_ERR_DWNA</td>
<td>5144 Here Windows on a equal machine is permitted only</td>
</tr>
<tr>
<td>H_ERR_WDM</td>
<td>5145 Wrong mode while opening a window</td>
</tr>
<tr>
<td>H_ERR_WWM</td>
<td>5146 Wrong window mode for operation</td>
</tr>
<tr>
<td>H_ERR_LUTC</td>
<td>5147 Operation not possible with fixed pixel</td>
</tr>
<tr>
<td>H_ERR_LUTN8</td>
<td>5148 Color tables for 8 image levels only</td>
</tr>
<tr>
<td>H_ERR_WTCM</td>
<td>5149 Wrong mode for pseudo real colors</td>
</tr>
</tbody>
</table>
HALCON 6.0

H_ERR_W1FTL 5150 Wrong pixel value for LUT
H_ERR_WSGI 5151 Wrong image size for pseudo real colors
H_ERR_HRULT 5152 Error in procedure HRLUT
H_ERR_WPFSL 5153 Wrong number of entries in color table for set_lut
H_ERR_WPFVS 5154 Wrong values for image area
H_ERR_WLPN 5155 Wrong line pattern
H_ERR_WLPFL 5156 Wrong number of parameters for line pattern
H_ERR_WP0C 5157 Wrong number of colors
H_ERR_WPST 5158 Wrong value for mode of area creation (0,1,2)
H_ERR_SWNNA 5159 Spy window is not set (set_spy)
H_ERR_NSF0 5160 No file for spy has been set (set_spy)
H_ERR_WSPN 5161 Wrong parameter output depth (set_spy)
H_ERR_WIFFD 5162 Wrong window size for window dump
H_ERR_WLUTF 5163 Wrong color table: wrong file name or query_lut()
H_ERR_WLUTE 5164 Wrong color table: empty string ?
H_ERR_WLUTD 5165 Using this hardware set_lut(’default’) is allowed only
H_ERR_CNDP 5166 Error while calling online help
H_ERR_LNPR 5167 Row can not be projected
H_ERR_NFSC 5168 Operation is unsuitable using a computer with fixed color table
H_ERR_NACD 5169 Computer represents gray scales only (no colors)
H_ERR_LUT0 5170 LUT of this display is full
H_ERR_WOC 5171 Internal error: wrong color code
H_ERR_WWATTRT 5172 Wrong type for window attribute
H_ERR_WWATRNN 5173 Wrong name for window attribute
H_ERR_WHSPAR 5174 negativ height of area (or 0)
H_ERR_WCSPART 5175 negativ width of area (or 0)
H_ERR_WNCV 5176 Window not completely visible
H_ERR_CHA3 5180 Internal error: only RGB-Mode
H_ERR_NMWA 5181 No more (image-)windows available
H_ERR_FNF 5200 File not found
H_ERR_DWI 5201 Error while writing image data (sufficient memory ?)
H_ERR_DWD 5202 Error while writing image descriptor (sufficient memory ?)
H_ERR_DRI1 5203 Error while reading image data (format of image too small ?)
H_ERR_DRI2 5204 Error while reading image data (format of image too big ?)
H_ERR_DRI1D 5205 Error while reading image descriptor: file too small
H_ERR_LIS 5206 Inconsistent image format
H_ERR_HNF 5207 Help file not found (setenv HALCONROOT <Halcon-Homedirectory> )
H_ERR_XNF 5208 Help index not found (setenv HALCONROOT <Halcon-Homedirectory> )
H_ERR_CNCSI 5209 File <standard_input> can not be closed
H_ERR_CNCSO 5210 <standard_output/error> can not be closed
H_ERR_CNCF 5211 File can not be closed
H_ERR_EDWF 5212 Error while writing to file
APPENDIX A. HALCON ERROR CODES

H_ERR_NFA  5213  Exceeding of maximum number of files
H_ERR_WFIN  5214  Wrong file name
H_ERR_CNOF  5215  Error while opening the file
H_ERR_VFM0  5216  Wrong file mode
H_ERR_WPTY  5217  Wrong type for pixel (e.g. byte)
H_ERR_WIV  5218  Wrong image width (too big ?)
H_ERR_VIH  5219  Wrong image height (too big ?)
H_ERR_FTS1  5220  File already exhausted before reading an image
H_ERR_FTS2  5221  File exhausted before terminating the image
H_ERR_WDFI  5222  Wrong value for resolution (dpi)
H_ERR_WNOW  5223  Wrong output image size (width)
H_ERR_WNIH  5224  Wrong output image size (height)
H_ERR_WNF  5225  Wrong number of parameter values: format description
H_ERR_WPNA  5226  Wrong parameter name for operator
H_ERR_WSHA  5227  Wrong slot name for parameter
H_ERR_NFCF  5228  Operator class is missing in help file
H_ERR_WHIF  5229  Wrong or inconsistent help/*.idx or help/*.sta
H_ERR_HINF  5230  File help/*.idx not found (setenv HALCONROOT <Halcon-Homedirectory>)
H_ERR_HSNF  5231  File help/*.sta not found (setenv HALCONROOT <Halcon-Homedirectory>)
H_ERR_ICSF  5232  Inconsistent file help/*.sta
H_ERR_EFNF  5233  No explication file (.exp) found
H_ERR_NFWKSF  5234  No file found in known graphic format
H_ERR_WIFT  5235  Wrong graphic format
H_ERR_ICNF  5236  Inconsistent file halcon.num
H_ERR_WTIFF  5237  File with extension ‘.tif’ is no Tiff-file
H_ERR_WFF  5238  Wrong file format
H_ERR_HOPPROC  5239  gnuplot could not be started
H_ERR_HOPFILE  5240  Output file for gnuplot could not be opened
H_ERR_HOPOUT  5241  Not a valid gnuplot output stream
H_ERR_HOPNM  5242  No PNM format
H_ERR_TCODEB  5243  Inconsistent or old help file (SHALCONROOT/help)
H_ERR_WFID  5244  Wrong file handle
H_ERR_FNO  5245  File not open
H_ERR_NO_FILES  5246  No files in use so far (none opened)
H_ERR_WSID  5250  Invalid handle for a serial connection
H_ERR_SN0  5251  Serial port not open
H_ERR_NSA  5252  No serial port available
H_ERR_CNOS  5253  Could not open serial port
H_ERR_CNCS  5254  Could not close serial port
H_ERR-CNOSA  5255  Could not get serial port attributes
H_ERR-CNSSA  5256  Could not set serial port attributes
H_ERR_WRSRB  5257  Wrong baud rate for serial connection
H_ERR_WRSDB  5258  Wrong number of data bits for serial connection
H_ERR_WRSRFC  5259  Wrong flow control for serial connection
H_ERR_CNFS  5260  Could not flush serial port
HALCON 6.0
APPENDIX A. HALCON ERROR CODES

H_ERR_GIF_READ_ERROR_EOF  5524  GIF: Premature end of file
H_ERR_GIF_NOT_ENOUGH_IMAGES  5525  GIF: Wrong number of images ‘;’
H_ERR_GIF_ERROR_ON_EXTENSION  5526  GIF: Wrong image extension ‘!’
H_ERR_GIF_LEFT_TOP_WIDTH  5527  GIF: Wrong left top width
H_ERR_GIF_CIRCULAR_TABLE_ENTRY  5528  GIF: Cyclic index of table
H_ERR_GIF_BAD_IMAGE_DATA  5529  GIF: Wrong image data
H_ERR_SUN_RASTERFILE_TYPE  5530  File is no Sun-Raster-File
H_ERR_SUN_RASTERFILE_HEADER  5531  SUN-Raster: Wrong header
H_ERR_SUN_COLS  5532  SUN-Raster: Wrong image width
H_ERR_SUN_ROWS  5533  SUN-Raster: Wrong image height
H_ERR_SUN_COLORMAP  5534  SUN-Raster: Wrong color map
H_ERR_SUN_RASTERFILE_IMAGE  5535  SUN-Raster: Wrong image data
H_ERR_SUN_IMPOSSIBLE_DATA  5536  SUN-Raster: Wrong type of pixel
H_ERR_XWD_IMPOSSIBLE_DATA  5540  XWD: Wrong type of pixel
H_ERR_XWD_VISUAL_CLASS  5541  XWD: Wrong visual class
H_ERR_XWD_X10_HEADER  5542  XWD: Wrong X10 header
H_ERR_XWD_X11_HEADER  5543  XWD: Wrong X11 header
H_ERR_XWD_X10_COLORMAP  5544  XWD: Wrong X10 colormap
H_ERR_XWD_X11_COLORMAP  5545  XWD: Wrong X11 colormap
H_ERR_XWD_X11_PIXMAP  5546  XWD: Wrong pixmap
H_ERR_XWD_UNKNOWN_VERSION  5547  XWD: unknown version
H_ERR_XWD_READING_IMAGE  5548  XWD: Error while reading an image
H_ERR_TIF_BAD_INPUTDATA  5550  TIFF: Error while reading a file
H_ERR_TIF_COLORMAP  5551  TIFF: Wrong colormap
H_ERR_TIF_BAD_CHANNEL_COLOR  5552  TIFF: Wrong number of color channels
H_ERR_TIF_TOO_MANY.COLORS  5553  TIFF: Too many colors
H_ERR_TIF_BAD_PHOTOMETRIC  5554  TIFF: Wrong photometric interpretation
H_ERR_TIF_PHOTOMETRIC_DEPTH  5555  TIFF: Wrong photometric depth
H_ERR_TIF_PHOTOMETRIC_MASK  5556  TIFF: Wrong photometric mask
H_ERR_TIF_SAMPLE_TOO_LARGE  5557  TIFF: Number of samples too large
H_ERR_TIF_NO_REGION  5558  TIFF: Image is no binary file
H_ERR_BMP_NO_BMP_PICTURE  5560  File is no BMP-File
H_ERR_BMP_READ_ERROR_EOF  5561  BMP: Premature end of file
H_ERR_BMP_INCOMPLETE_HEADER  5562  BMP: Incomplete header
H_ERR_BMP_UNKNOWN_FORMAT  5563  BMP: Unknown bitmap format
H_ERR_BMPUNKNOWN_COMPRESSION  5564  BMP: Unknown compression format
H_ERR_BMP_COLORMAP  5565  BMP: Wrong color table
H_ERR_BMP_WRITE_ERROR  5566  BMP: Write error on output
H_ERR_BMP_NO_REGION  5567  BMP: File does not contain a binary image
H_ERR_JPG_COMP_NUM  5570  JPEG: wrong number of components in image
H_ERR_JPGLIB_UNKNOWN  5571  JPEG: unknown error from libjpeg
H_ERR_JPGLIB_NOTIMPL  5572  JPEG: no implementet feature in libjpeg
H_ERR_JPGLIB_FILE  5573  JPEG: file access error in libjpeg
H_ERR_JPGLIB_TMPFILE  5574  JPEG: tmp file access error in libjpeg
H_ERR_JPGLIB_MEMORY  5575  JPEG: memory error in libjpeg
H_ERR_JPGLIB_INFORMAT  5576  JPEG: error in input image
H_ERR_SOCKET_BLOCK  5600  Socket can not be set to block
<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>H_ERR_SOCKET_UNBLOCK</td>
<td>5601  Socket can not be set to unblock</td>
</tr>
<tr>
<td>H_ERR_SOCKET_NO_CP</td>
<td>5602   Received data is no tuple</td>
</tr>
<tr>
<td>H_ERR_SOCKET_NO_IMAGE</td>
<td>5603  Received data is no image</td>
</tr>
<tr>
<td>H_ERR_SOCKET_NO_RL</td>
<td>5604   Received data is no region</td>
</tr>
<tr>
<td>H_ERR_SOCKET_READ_DATA FAILED</td>
<td>5605  Received data is no xld object</td>
</tr>
<tr>
<td>H_ERR_SOCKET_WRITE_DATA FAILED</td>
<td>5606  Error while reading from socket</td>
</tr>
<tr>
<td>H_ERR_SOCKET_WRONG_BYTE_NUMBER</td>
<td>5607  Error while writing to socket</td>
</tr>
<tr>
<td>H_ERR_SOCKET_BUFFER_OVERFLOW</td>
<td>5608  Illegal number of bytes with get_rl</td>
</tr>
<tr>
<td>H_ERR_SOCKET_CANT_ASSIGN_FD</td>
<td>5609   Buffer overflow in read_data</td>
</tr>
<tr>
<td>H_ERR_SOCKET_CANT_BIND</td>
<td>5610   Socket can not be created</td>
</tr>
<tr>
<td>H_ERROCKET_CANT_GET_PORTNUMBER</td>
<td>5611   Bind on socket failed</td>
</tr>
<tr>
<td>H_ERR_SOCKET_CANT_LISTEN</td>
<td>5612   Socket information is not available</td>
</tr>
<tr>
<td>H_ERR_SOCKET_CANT_ACCEPT</td>
<td>5613   Socket cannot listen for incoming connections</td>
</tr>
<tr>
<td>H_ERR_SOCKET_CANT_CONNECT</td>
<td>5614   Connection could not be accepted</td>
</tr>
<tr>
<td>H_ERR_SOCKET_GETHOSTBYNAME</td>
<td>5615   Connection request failed</td>
</tr>
<tr>
<td>H_ERR_SOCKET_RECV_IS_NEG</td>
<td>5616   Hostname could not be resolved</td>
</tr>
<tr>
<td>H_ERR_SOCKET_ILLEGAL_TUPLE_TYPE</td>
<td>5617   No data on socket</td>
</tr>
<tr>
<td>H_ERR_LINK_WRITE_DATA_FAILED</td>
<td>5618   Unknown tuple type on socket</td>
</tr>
<tr>
<td>H_ERR_LINK_READ_DATA_FAILED</td>
<td>5620   Error while writing to virtual (EPX-) connection.</td>
</tr>
<tr>
<td>H_ERR_LINK_NO_RL</td>
<td>5622   Received data is no Hrlregion</td>
</tr>
<tr>
<td>H_ERR_LINK_NO_IMAGE</td>
<td>5623   Received data is no Himage</td>
</tr>
<tr>
<td>H_ERR_NP</td>
<td>6000   Access to undefined memory area</td>
</tr>
<tr>
<td>H_ERR_MEM</td>
<td>6001   not enough memory available</td>
</tr>
<tr>
<td>H_ERR_ICM</td>
<td>6002   Memory partition on heap has been overwritten</td>
</tr>
<tr>
<td>H_ERR_WMS</td>
<td>6003   HALloc: 0 bytes requested</td>
</tr>
<tr>
<td>H_ERR_NOTMP</td>
<td>6004   Tmp-memory management: Call freeing memory although nothing had been allocated</td>
</tr>
<tr>
<td>H_ERR_TMPNULL</td>
<td>6005   Tmp-memory management: Null pointer while freeing</td>
</tr>
<tr>
<td>H_ERR_CNFMEM</td>
<td>6006   Tmp-memory management: could not find memory element</td>
</tr>
<tr>
<td>H_ERR_IAD</td>
<td>6040   System parameter for memory-allocation inconsistent</td>
</tr>
<tr>
<td>H_ERR_NRA</td>
<td>6041   No memory block allocated at last</td>
</tr>
<tr>
<td>H_ERR_WOCPI</td>
<td>7000   Wrong index for output control parameter</td>
</tr>
<tr>
<td>H_ERR_WOCVPN</td>
<td>7001   Wrong number of values: output control parameter (see: HPut*Par)</td>
</tr>
<tr>
<td>H_ERR_WOCPT</td>
<td>7002   Wrong type: output control parameter (see: HPut*Par)</td>
</tr>
<tr>
<td>H_ERR_WKT</td>
<td>7003   Wrong data type for object key (input objects)</td>
</tr>
<tr>
<td>H_ERR_IODR</td>
<td>7004   Range for object key had been passed</td>
</tr>
<tr>
<td>H_ERR_IHV</td>
<td>7005   Inconsistent Halcon version</td>
</tr>
<tr>
<td>H_ERR_NISS</td>
<td>7006   Not enough memory for strings allocated</td>
</tr>
<tr>
<td>H_ERR_PROC_NULL</td>
<td>7007   Internal error: Proc is NULL</td>
</tr>
<tr>
<td>H_ERR_WLST</td>
<td>7100   Wrong list structure using input objects</td>
</tr>
<tr>
<td>H_ERR_WIOP</td>
<td>7101   Wrong input object parameter (not bound)</td>
</tr>
<tr>
<td>H_ERR_WICP</td>
<td>7102   Wrong input control parameter (not bound)</td>
</tr>
<tr>
<td>H_ERR_WODP</td>
<td>7103   Wrong output object parameter (already bound)</td>
</tr>
<tr>
<td>H_ERR_WOCP</td>
<td>7104   Wrong output control parameter (already bound)</td>
</tr>
<tr>
<td>H_ERR_UNKN</td>
<td>7105   Unknown symbolic object key (input objects)</td>
</tr>
<tr>
<td>Code</td>
<td>Error Description</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>H_ERR_WOODN</td>
<td>Wrong number of output object parameter</td>
</tr>
<tr>
<td>H_ERR_WOODIP</td>
<td>Wrong number of input parameter</td>
</tr>
<tr>
<td>H_ERR_OTSE</td>
<td>System error: output type &lt;string&gt; expected</td>
</tr>
<tr>
<td>H_ERR_OTLE</td>
<td>System error: output type &lt;long&gt; expected</td>
</tr>
<tr>
<td>H_ERR_OPIBP</td>
<td>System error: output type &lt;float&gt; expected</td>
</tr>
<tr>
<td>H_ERR_CRC</td>
<td>Object parameter is a zero pointer (&quot;.&quot; not allowed)</td>
</tr>
<tr>
<td>H_ERR_CTP</td>
<td>Tupel had been deleted; values are not valid any more</td>
</tr>
<tr>
<td>H_ERR_TNOME</td>
<td>CPP-interface internal error: wrong object mode</td>
</tr>
<tr>
<td>H_ERR_TNPC</td>
<td>Wrong number of regions (&gt;1) for type HRegion</td>
</tr>
<tr>
<td>H_ERR_TNOM</td>
<td>Wrong number of images (&gt;1) for type HImage</td>
</tr>
<tr>
<td>H_ERR_TNQAL</td>
<td>Tupel with undefined values</td>
</tr>
<tr>
<td>H_ERR_TPCS</td>
<td>No contact to RPC server</td>
</tr>
<tr>
<td>H_ERR_TPC</td>
<td>Error in remote procedure call</td>
</tr>
<tr>
<td>H_ERR_SWI_NOLIST</td>
<td>Parameter value is neither a list nor a atom</td>
</tr>
<tr>
<td>H_ERR_WFRN</td>
<td>Unknown operator name</td>
</tr>
<tr>
<td>H_ERR_RDNA</td>
<td>register used is not activated (see set system)</td>
</tr>
<tr>
<td>H_ERR_RFC</td>
<td>Unknown operator class</td>
</tr>
<tr>
<td>H_ERR_RRMF</td>
<td>convol/Maske: error while opening the file</td>
</tr>
<tr>
<td>H_ERR_EOFRMF</td>
<td>convol/Maske: premature end of file</td>
</tr>
<tr>
<td>H_ERR_CVTRMF</td>
<td>convol/Maske: conversion error</td>
</tr>
<tr>
<td>H_ERR_LCRMF</td>
<td>convol/Maske: wrong row-/column number</td>
</tr>
<tr>
<td>H_ERR_WCOVRMF</td>
<td>convol/Maske: mask size overflow</td>
</tr>
<tr>
<td>H_ERR_NEGFRMF</td>
<td>convol/Maske: too many elements entered</td>
</tr>
<tr>
<td>H_ERR_WRA</td>
<td>convol: wrong margin type</td>
</tr>
<tr>
<td>H_ERR_MCN0</td>
<td>convol: no mask object has got empty region</td>
</tr>
<tr>
<td>H_ERR_FCN0</td>
<td>convol: no filter object has got empty region</td>
</tr>
<tr>
<td>H_ERR_WF0</td>
<td>convol: Weight factor is 0</td>
</tr>
<tr>
<td>H_ERR_WIC</td>
<td>convol: inconsistent number of weights</td>
</tr>
<tr>
<td>H_ERR_WRRV</td>
<td>rank: wrong rank value</td>
</tr>
<tr>
<td>H_ERR_ROVFL</td>
<td>convol/rank: error while handling margin</td>
</tr>
<tr>
<td>H_ERR_WNUMM</td>
<td>Wrong number of coefficients for convolution (sigma too big?)</td>
</tr>
<tr>
<td>H_ERR_WBEDN</td>
<td>No valid ID for data set</td>
</tr>
<tr>
<td>H_ERR_NBEDA</td>
<td>No data set active (set bg esti)</td>
</tr>
<tr>
<td>H_ERR_BEDNAU</td>
<td>ID already used for data set (is not possible)</td>
</tr>
<tr>
<td>H_ERR_BEDNTB</td>
<td>Maximum number of data sets exceeded</td>
</tr>
<tr>
<td>H_ERR_NBEDC</td>
<td>No data set created (create bg esti)</td>
</tr>
<tr>
<td>H_ERR_NTM</td>
<td>Not possible to pass an object list</td>
</tr>
<tr>
<td>H_ERR_WISBE</td>
<td>Image has other size than the background image in data set</td>
</tr>
<tr>
<td>H_ERR_UDNISSEBE</td>
<td>Up-date-region is bigger than background image</td>
</tr>
<tr>
<td>H_ERR_SNBETS</td>
<td>Number of statistic data sets is too small</td>
</tr>
<tr>
<td>H_ERR_WAMBE</td>
<td>Wrong value for adapt mode</td>
</tr>
<tr>
<td>H_ERR_WFMBE</td>
<td>Wrong value for frame mode</td>
</tr>
<tr>
<td>H_ERR_OCREM1</td>
<td>Maximum number of fonts exceeded</td>
</tr>
<tr>
<td>H_ERR_OCRES1</td>
<td>Wrong ID (Number) for font</td>
</tr>
<tr>
<td>H_ERR_OCRES1</td>
<td>OCR internal error: wrong ID</td>
</tr>
<tr>
<td>H_ERR_OCRESNN</td>
<td>OCR not initialised: no font was read in</td>
</tr>
</tbody>
</table>
H_ERR_OCRA_NAI  8304  No font activated
H_ERR_OCRA_WTP  8305  OCR internal error: wrong threshold in angle
determination
H_ERR_OCRA_VF  8306  OCR internal error: wrong attribute
H_ERR_OCRA_READ  8307  OCR: Error during reading file
H_ERR_OCRA.Nodes  8308  OCR File: inconsistent number of nodes
H_ERR_OCRA_EOF  8309  OCR File: File too short
H_ERR_OCRA_INC1  8310  OCR: internal error 1
H_ERR_OCRA_INC2  8311  OCR: internal error 2
H_ERR_WOCRYPE  8312  Wrong type of OCR tool (no 'box' or 'net')
H_ERR_OCR_NI  8350  OCV system not initialized
H_ERR_WOCRVTYP  8351  Wrong type of OCV features
H_ERR_OCV_NODEID  8352  No more free OCV handles available
H_ERR_OCV_PLAYER  8353  Wrong name for an OCV object
H_ERR_OCV_NI  8354  Training has already been applied
H_ERR_OCV_NOTTR  8355  No training has been applied to the character
H_ERR_WLENTH  8370  Wrong number of function points
H_ERR_NO_FUNCTION  8371  List of values is not a function
H_ERR_NOT_ASCENDING  8372  Wrong ordering of values (not ascending)
H_ERR_ILLEGAL_DIST  8373  Illega distance of function points
H_ERR_CAL_ICALP  8400  You have to indicate at least 3 calibration points
H_ERR_CAL_CPTB  8401  Calibration table is too big
H_ERR_CAL_NCIPF  8402  No calibration table found
H_ERR_CAL_RECIPF  8403  Error while reading calibration table description file
H_ERR_CAL_LTMTTH  8404  Minimum threshold while searching for ellipses
H_ERR_CAL_FRCP  8405  Read error / format error in calibration table description
file
H_ERR_CAL_PRDJ  8406  Error in projection: s_x = 0 or s_y = 0 or z = 0
H_ERR_CAL_UNPR0  8407  Error in inverse projection
H_ERR_CAL_R1CPF  8408  Not possible to open camera parameter file
H_ERR_CAL_FICP1  8409  Format error in file: no colon
H_ERR_CAL_FICP2  8410  Format error in file: 2. colon is missing
H_ERR_CAL_FICP3  8411  Format error in file: semicolon is missing
H_ERR_CAL_REPOS  8412  Not possible to open camera parameter (pose) file
H_ERR_CAL_FOP0S  8413  Format error in camera parameter (pose) file
H_ERR_CAL_OCPDF  8414  Not possible to open calibration target description file
H_ERR_CAL_OCPPS  8415  Not possible to open postscript file of calibration target
H_ERR_CAL_EVECn  8416  Error while norming the vector
H_ERR_CAL_NPLAN  8417  Fitting of calibration target failed
H_ERR_CAL_NMMAR  8418  No next mark found
H_ERR_CAL_NNEQU  8419  Normal equation system is not solvable
H_ERR_CAL_QETHM  8420  Average quadratic error is too big for 3D position of mark
H_ERR_CAL_NDELL  8421  Non elliptic contour
H_ERR_CAL_WAARV  8422  Wrong parameter value slvand()
H_ERR_CAL_WFRS  8423  Wrong function results slvand()
H_ERR_CAL_ECPDI  8424  Distance of marks in calibration target description file is
not possible
H_ERR_CAL_WEFLA  8425  Specified flag for degree of freedom not valid
H_ERR_CAL_NOMER  8426 Minimum error did not fall below
H_ERR_CAL_WPTYP  8427 Wrong type in Pose (rotation / translation)
H_ERR_CAL_WIMSZ  8428 Image size does not match the measurement in camera parameters
H_ERR_NOAJP     8500 Not a pattern file
H_ERR_WPFV      8501 wrong pattern file version
H_ERR_MATCH_MODE 8502 Error during changing the file mode (t/b)
H_ERR_MATCH_0DR  8503 Inconsistent match file: coordinates out of range
H_ERR_MATCH_BORDER 8504 Pattern too near the image border
H_ERR_NOTAP      8505 The image(s) is not a pyramid (wrong zooming factor?)
H_ERR_MEASURE_NA 8550 No more measure objects available
H_ERR_MEASURE_NI 8551 Measure object is not initialized
H_ERR_MEASURE_OS 8552 Invalid measure object
H_ERR_MEASURE_WS 8553 Measure object is NULL
H_ERR_DLOPEN     8600 Dynamic library could not be opened
H_ERR_DLCLOSE    8601 Dynamic library could not be closed
H_ERR_DLLOOKUP   8602 Symbol not found in dynamic library
H_ERR_BAR_UNKNOWN 8700 Unknown bar code
H_ERR_BAR_WNOM   8701 Wrong number of modules
H_ERR_BAR_WNDE   8702 Wrong number of elements
H_ERR_BAR_UNCHAR 8703 Unknown character (for this code)
H_ERR_BAR_WNGDESCR 8705 wrong name for attribute in barcode descriptor
H_ERR_BAR_EL_LENGTH 8706 Wrong thickness of element
H_ERR_BAR_NO_REG 8707 No region found
H_ERR_BAR_UNCHAR01 8720 The 1st character is unknown (for this code)
H_ERR_BAR_UNCHAR02 8721 The 2nd character is unknown (for this code)
H_ERR_BAR_UNCHAR03 8722 The 3rd character is unknown (for this code)
H_ERR_BAR_UNCHAR04 8723 The 4th character is unknown (for this code)
H_ERR_BAR_UNCHAR05 8724 The 5th character is unknown (for this code)
H_ERR_BAR_UNCHAR06 8725 The 6th character is unknown (for this code)
H_ERR_BAR_UNCHAR07 8726 The 7th character is unknown (for this code)
H_ERR_BAR_UNCHAR08 8727 The 8th character is unknown (for this code)
H_ERR_BAR_UNCHAR09 8728 The 9th character is unknown (for this code)
H_ERR_BAR_UNCHAR10 8729 The 10th character is unknown (for this code)
H_ERR_BAR_UNCHAR11 8730 The 11th character is unknown (for this code)
H_ERR_BAR_UNCHAR12 8731 The 12th character is unknown (for this code)
H_ERR_BAR_UNCHAR13 8732 The 13th character is unknown (for this code)
H_ERR_BAR_UNCHAR14 8733 The 14th character is unknown (for this code)
H_ERR_BAR_UNCHAR15 8734 The 15th character is unknown (for this code)
H_ERR_BAR_UNCHAR16 8735 The 16th character is unknown (for this code)
H_ERR_BAR_UNCHAR17 8736 The 17th character is unknown (for this code)
H_ERR_BAR_UNCHAR18 8737 The 18th character is unknown (for this code)
H_ERR_BAR_UNCHAR19 8738 The 19th character is unknown (for this code)
H_ERR_BAR_UNCHAR20 8739 The 20th character is unknown (for this code)
H_ERR_BAR2D_UNKNOWN_TYPE 8800 Specified code type is not supported
131

8801  Wrong foreground specified
8802  Wrong matrix size specified
8803  Wrong symbol shape specified
8804  Wrong generic parameter name
8805  Wrong generic parameter value
8806  Wrong symbol printing mode
8807  Symbol region too near to image border
8808  No rectangular modul boundings found
8809  Couldn’t identify symbol finder
8810  Symbol region with wrong dimension
8811  Classification failed
8812  Decoding failed
8813  Reader programing not supported
9000  Function not implemented on this machine
9001  Image to process has wrong gray value type
9002  Wrong image component (see: get_system(obj_images,H))
9003  Undefined gray values
9004  Wrong image format for operation (too big or too small)
9005  Wrong number of image components for image output
9006  String is too long (max. 255 characters)
9007  Wrong pixel type for this operation
9008  Operation not realized yet for this pixel type
9009  Image is no color image with three channels
9010  Frame grabbers are not supported in the demo version
9011  Packages are not supported in the demo version
9050  Operator is not available in the student version of HALCON
9051  Packages are not available in the student version of HALCON
9052  The selected frame grabber is not available in the student version of HALCON
9100  Too many unknown variables in linear equation
9101  No (unique) solution for the linear equation
9102  Too little equations in linear equation
9200  No inversion of matrix possible
9201  Singular value decomposition did not converge
9202  Matrix has too little rows for singular value partition
9203  Eigenvalue computation did not converge
9204  Eigenvalue computation did not converge
9205  Matrix is singular
9206  Function matching did not converge
9207  Input matrix undefined
9208  Input matrix with wrong dimension
9209  Input matrix is not quadratic
9210  Matrix operation failed
9211  Matrix is not positive definite
9300  Eye point and reference point coincide