HALCON
a product of MVTec

Extension Package
Programmer's Manual

HALCON 23.11 *Progress*

MVTec Software GmbH

Building Vision for Business

This manual shows how to create HALCON extension packages based on new operators written in C, Version 23.11.0.0.

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

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 normal 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 normal HALCON system. Furthermore, programming skills are required. Finally, the reader should know about his/her development 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 file)**
  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 platforms.

Symbols

The following symbol is used within the manual:

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

Contents

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, XLDs[1]) from the HALCON database of iconic objects.

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 HALCON system and the interface between the operator layer and the *iconic object database*.

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.

The new extension package example program is an Easy Extension Interface example for .NET, and shows how to easily integrate a .NET function, based on the HALCON/.NET interface, to HALCON's operator set. It can be used within HDevEngine and HDevelop. For more information, see the README.md in the Easy Extension Interface for .NET example under `%HALCONEXAMPLES%\extension_package\easy_extensions\dotnet`. Once the users have extended HALCON by a package containing their own operators, these 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 `%HALCONEXAMPLES%\extension_package\halconuser`).

Except for Linux-specific sections, file paths and environment variables are printed in the Windows convention in this manual, e.g.,

```
%HALCONEXAMPLES%\extension_package\halconuser
```

to denote the subdirectory `halconuser` containing an example package within the HALCON base directory referenced by the environment variable `HALCONEXAMPLES`. The same expression in Linux convention would look like

```
$HALCONEXAMPLES/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] e**X**tended **L**ine **D**escription; i.e., subpixel contours and polygons.

1.1.1 Using HALCON Operators in C, C++, and .NET Languages

There are two different modes of calling operators from HALCON/C: 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* (`Htuple`). Iconic objects are represented with the type `Hobject`, so that one object can contain several images, regions, or XLDs.

In contrast, the HALCON/C++ and the HALCON/.NET interface support a flexible management of several types.

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 using the generic data type `Hobject` in order to pass arguments to and from the operator. This kind of integration is also used, e.g., if you export your HDevelop program as C++ or C# code.

For further information on using HALCON in programming languages please refer to the Programmer's Guide.

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 iconic objects, e.g., image tuples, the supply procedure has to extract the single parts. An *iconic 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 figure 5.2 on page 58). Generally, a user may assume that a HALCON operator is able to handle single iconic objects as well as composed objects (i.e., `multivalue` is set to `optional`; see page 30) and that it can work on multi-channel images (i.e., `multichannel` is set to `optional`; see page 34). So, the author of a new HALCON operator should implement appropriate mechanisms for extracting the needed data from composed iconic 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. section 2.1 on page 19 and section 2.2 on page 20).

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 `Hrlregion`, channels of type `Himage`, 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. Every supply procedure returns an error code of type `Herror`. Thus, it is necessary to return an error code, which is `H_MSG_TRUE` if no error occurred.

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 figure 1.1.

The buffer contains input data as well as output data. Control parameters (containing all kind of control values like integers, floating-point values, strings or handles) are stored directly in the buffers, whereas all iconic data (including images, regions and XLDs) are stored in the database and represented by an iconic object key. Iconic data can be accessed with the help of database procedures by using these 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

- input iconic object
- output iconic object
- input control parameter
- output control parameter

that determines the name of the suitable procedures/macros (e.g.: `HGetCPar` is used to *get* an input *c*ontrol value). To identify a specific parameter within a class, its number is passed to the procedure/macro. This is the number of

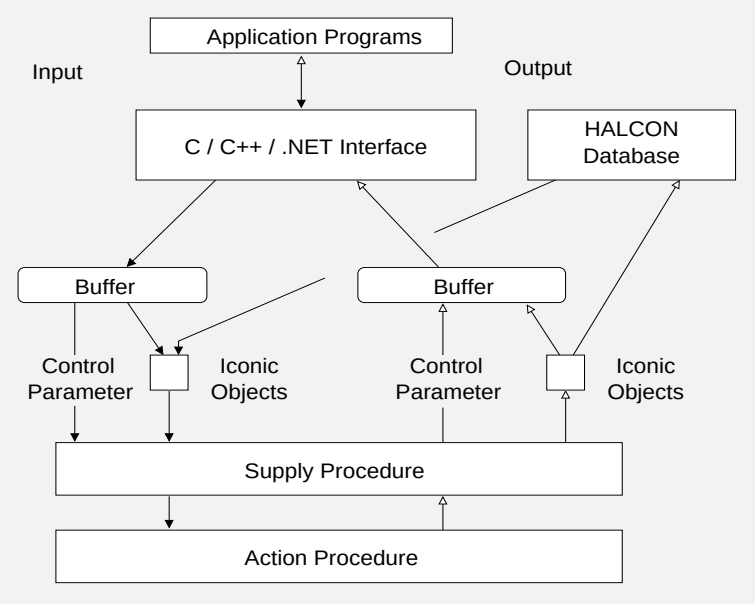

Figure 1.1: Data flow within HALCON.

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 `hcomp` (see page 16 and section 7.1 on page 97) 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,iconic object/control), about the parameter type, etc. The syntax and semantics of the definition file is explained in chapter 2 on page 19.

1.2 HALCON Packages

Similar to the standard HALCON system a programmer has to provide (at least) two libraries (DLLs in Windows, shared libraries in Linux environments) in order to extend HALCON by a new operator package (see chapter 7 on page 97).

- 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, C++, .NET). These libraries (i.e., the operator library and one of the interface libraries) have to be used in 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 on page 97 for details. In the example package these libraries are called `halconuserc`, `halconusercpp`, and `halconuserdotnet`.

The names of all these libraries are derived from the name of the package itself: A package `package` contains the libraries[2] `package`, `packagec`, `packagecpp`, and `packagedotnet`.

[2]This is true for the Windows version. For Linux, the prefix `lib` is added to the libraries, resulting in `libpackage`, `libpackagec`, and `libpackagecpp`.

1.2.1 Packages and HALCON XL

Extension packages can also be used in HALCON XL. Similar to the HALCON libraries, which exist in two versions (e.g., `halcon.dll` and `halconxl.dll`), you must provide separate versions of your package libraries to be used together with HALCON XL. The names of these libraries must have the suffix `xl`; e.g., the HALCON XL version of the example package halconuser consists of the libraries `halconuserxl.dll`, `halconusercxl.dll`, `halconusercppxl.dll`, and `halconuserxxl.dll` (Windows), `libhalconuserxl.so`, `libhalconusercxl.so`, and `libhalconusercppxl.so` (Linux), and `halconuserdotnetxl.dll` (.NET).

The HALCON XL version of the libraries is created analogously to the standard HALCON version with only small differences. See section 7.2 on page 100 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 `%HALCONEXAMPLES%\extension_package`. This directory must at least contain the following subdirectories

- `bin\%HALCONARCH%`: For Windows, this subdirectory has to contain the DLLs `package.dll`, `packagec.dll`, and `packagecpp.dll`, corresponding to the above libraries (and their HALCON XL versions). We also recommend to place compiled example programs or additional utilities of a package in this subdirectory.
- `lib\%HALCONARCH%`: In this subdirectory the libraries encapsulating the new operators (`package`) and the corresponding C and C++ language interfaces (`packagec`, `packagecpp`) reside[3], together with their HALCON XL versions. **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 `HALCONARCH` is set during the installation of HALCON; it describes the platform HALCON is running on. See the Installation Guide, section 1.4 on page 8, for details.
- `bin\dotnet`: This subdirectory has to contain the extension package's .NET assembly `packagedotnet.dll` (and its HALCON XL version, `packagedotnetxl.dll`).
- `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 providing additional subdirectories:

- `def`: In this directory the DEF file(s) with the description of the operators of a package should be placed.
- `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 `%HALCONROOT%\doc` for comparison). Especially, it is necessary to put the generated HTML files in `doc\html\reference` 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 including the path to this subdirectory in the environment variable HALCONIMAGES to allow access to these images without using an absolute file path in the programs.

[3] For Linux systems the prefix `lib` has to be added to the libraries resulting in `libpackage`, `libpackagec`, and `libpackagecpp`

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 HALCON XL 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 files (cf. chapter 2 on page 19). We recommend placing these files in the package subdirectory `def`.
2. Creation of a library (called `package`) containing the supply procedures in C and the corresponding action procedures (cf. chapters 3 to 6). This library has to be placed in the package subdirectory `lib/$HALCONARCH` (Linux) or `bin\%HALCONARCH%` (Windows).
3. Creation of a library containing the language-dependent operator interfaces (`packagec`, `packagecpp`, `packagex`, and `packagedotnet`) based on the DEF files with the help of `hcomp` (cf. section 7.1 on page 97). These libraries have to be placed in the subdirectory `lib/$HALCONARCH` (C, C++ under Linux) or `bin\%HALCONARCH%` (C, C++ under Windows) or `bin\dotnet` 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 `help` 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` of the package.
6. Generating PDF manuals with the help of `hcomp` (optional).
7. Extension of the package list in the environment variable `HALCONEXTENSIONS` by the *complete* file path of the new package, e.g.,

   ```
   %HALCONEXAMPLES%\extension_package\halconuser
   ```

   For some platforms, additional environment variables have to be modified; see section 1.3 and section 7.2.6 on page 101.
8. Linking of the generated libraries together with the HALCON library to the image analysis application (cf. chapter 7 on page 97). 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

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.,

```
%HALCONEXAMPLES%\extension_package\halconuser
```

Note that the delimiter between paths in an environment variable is a semicolon on Windows systems and a colon on Linux 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. To use the new package within HDevelop, you have to **restart** the program. For generating a stand-alone application using the package you have to link the C or C++ interface library, or reference the .NET assembly of the package in the application code, see section 7.2 on page 100.

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.

Furthermore, you must modify additional environment variables in order to use your new package (see below).

1.3.1 Additional Information for Windows

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

```
%HALCONEXAMPLES%\extension_package\halconuser\bin\%HALCONARCH
```

has to be added to the environment variable `PATH`, see section 7.2.6.1 on page 101.

Do not copy a package DLL into the Windows system directories, as it would be loaded twice in this case!

Note that you must extend the variable `PATH` also for .NET applications, even if you do not explicitly link the main package library (`package.dll`): The referenced .NET assembly automatically loads this library.

1.3.2 Additional Information for Linux

On Linux systems you have to include the package library subdirectory `lib/$HALCONARCH` 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.

For .NET applications, an alternative to setting `LD_LIBRARY_PATH` is described in the Programmer's Guide in section 12.2.3 on page 87.

1.4 An Example

The following example illustrates the steps described in section 1.2 on page 11. 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 figure 1.2; a more detailed example is provided in `%HALCONEXAMPLES%\extension_package\halconuser\source\testthreshold.c`.

```
main()
{
  Hobject   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.

To use `user_thresh` in this way within C programs[4],

- 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.

First of all one has to create a *definition file* (file extension `.def`), as illustrated in figures 1.3 and 1.4. Figure 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 HDevelop, C++, or C#, an extended version is necessary as described in section 2.2 on page 20. The minimal form of such an extended definition is illustrated in figure 1.4. The definition file specifies:

[4] To use it within C++, C#, or Visual Basic basically the same steps are necessary, see chapter 7 on page 97.

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

Figure 1.3: Defining a new operator `user_thresh`.

```
user_thresh <- CIPUserThreshold[Image:Region:Threshold:]

  short.german
    Schwellenwert-Operator.;

  short.english
    Selection of gray values by thresholding.;

  module
    foundation;

  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. `%HALCONEXAMPLES%\extension_package\halconuser\def\threshold.def`.

- the name of the operator (`user_thresh`),
- 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 iconic 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 iconic object parameter (`Image`),
- one output iconic object parameter (`Region`),
- one input control parameter (`Threshold`) of type `int` with exactly one value and
- no output control parameter.

The operator definition within the DEF file now is used by `hcomp` to generate the appropriate interface code. figure 1.5 shows the calls of `hcomp`. Note, that within this manual `halconuser` is used as name of the HALCON

```
hcomp -u -H -phalconuser threshold.def
hcomp -u -C -phalconuser threshold.def
```

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

package to be created. By calling `hcomp` with these options[5] the files `Hhalconuser.c` (HALCON interface code), `HChalconuser.c` (C-specific interface code) and `HChalconuser.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 figure 1.6.

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

/* supply procedure - always with this parameter! */
Herror CIPUserThreshold(Hproc_handle proc_handle)
{
  HGetCPar(..,&Threshold,..);              /* get control    */
                                           /* parameter      */
  HAllSegm(proc_handle,&region,&image,1,i) /* work on all    */
                                           /* input images   */
  {
    HCkP(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, this is `halconuserc` containing `HChalconuser.o`. The example CMake file `CMakeLists.txt` in the directory `%HALCONEXAMPLES%\extension_package\halconuser` can be used for generating the example package under Linux or Windows and test programs as well.

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 online help files (as they are used e.g., by HDevelop) can also be generated with the help of the HALCON compiler `hcomp`, see figure 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.

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 important especially for constructing graphical user interfaces

[5] see section 7.1 on page 97 for a complete description.

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. The HTML files have to be placed in the corresponding subdirectory `doc\html\reference` of the package.

If you now add the package path to `HALCONEXTENSIONS` and start HDevelop the user extensions, e.g. `user_thresh`, are automatically available in the specified menu (i.e., chapter). In our example this menu is called `UserExtensions` as specified in `%HALCONEXAMPLES%\extension_package\halconuser\def\threshold.def`.[6] 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 `%HALCONEXAMPLES%\extension_package\halconuser` (`*.dev`). Furthermore, the `*.dev` programs have been exported as C++ and C programs. The `*.cpp` and `*.c` files can be found in the subdirectory `source`. You can compile them using the already mentioned CMake file. These `*.cpp` and `*.c` files also serve as an example for the integration of user defined operators into the host languages C++ and C. Note that before compiling the *exported* C++ and C code containing operators from the package `halconuser`, the corresponding user extension libraries have been created. This can be done via the `CMakeLists.txt` in `%HALCONEXAMPLES%\extension_package\halconuser` which will also generate the needed user extension C++ library `halconusercpp` and C library `halconuserc`.

Please note that on some systems not all users have write 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.

[6] `UserExtensions` is integrated into the menu `Operator` of HDevelop.

Chapter 2

Operator Description (DEF File)

Note that DEF files with non-ASCII characters should be UTF-8 encoded.

The following chapters give a detailed description of the steps mentioned in chapter 1 on page 9. This is done with respect to system-defined HALCON operators as well as to the examples within the directory `%HALCONEXAMPLES%\extension_package\halconuser`. 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 parameter classes: *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 `hcomp`. It describes the operator properly and may be specified as a "one line short version" of the DEF file according to the following syntax:

```
extern_operator_name <- InternCProcedure
          [InputObjects:OutputObjects:InputCtrlParams:OuputCtrlParams];
```

Section 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

```
user_inside <- CIPUserSelect[Regions:RegionSelected:Row$i,Column$i:];
```

Figure 2.1: Short version of the definition of the operator `user_inside` (cf. `%HALCONEXAMPLES%\extension_package\halconuser\def\regionfeatures.def`).

`OuputCtrlParams`). 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 figure 2.1).

The following table shows how to code types and number of values into a character. Lower-case characters stand for *exactly one value* in the specified parameter. This corresponds to assigning `multivalue: false` in the extended version. Capital letters specify parameters that contain *one or more than one value* (`multivalue: optional`).

Type	C-type	Code	
		One Value	Unlimited Number
Integer	`long`	i	I
Floating-point Value	`double`	f	F
String	`char *`	s	S

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++ or .NET applications. To achieve this, an extended version of the operator description must be generated. This is described in the following section.

2.2 Extended Operator Description

An operator description within a 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 (`\;`).

Some slots are language dependent: For example `short.english` contains a short description of the operator in *English*, `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 on page 28 explains how to do this.

2.2.1 Header

The operator description starts with the header

```
extern_operator_name <- InternCProcedure
          [InputObjects:OutputObjects:InputCtrlParams:OuputCtrlParams]
```

Note that in contrast to the short version described in section 2.1, 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

```
/*********************  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 lowercase characters with word parts separated by underscores. This operator name is used within C programs, low-level calls[1] of the operator in C++, or within HDevelop.

The four parameter classes (input-/output iconic 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:

parameter class	keyword (cf. page 29)	parameter in the example
input iconic objects	`input_object`	OrigImage, ThresholdImage
output iconic objects	`output_object`	RegionDynThresh
input control parameters	`input_control`	Offset, LightDark
output control parameters	`output_control`	$\emptyset$

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 *Row(s)* (not *Line*!) and *Column(s)* or *Col(s)*. Parameters defining a dimension are called *Width* and *Height*.

2.2.2 short

The keyword `short` starts the short description of the operator, in our example

```
short.english
  Segment an image using a local threshold.;
```

As all slots specified by the keyword `.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 on page 35.

2.2.3 warning

With the slot `warning`, you can show a warning text in bold at the beginning of an operator description, for example

```
warning.english
  This operator is obsolete.;
```

As all slots specified by the keyword `.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 on page 35.

2.2.4 abstract

A more detailed description of the operator is given within the slot `abstract`. The description of the example is here shown as a shortened version:

[1] Within the HALCON/C++ and HALCON/.NET class hierarchy corresponding *methods* use slightly different conventions: `extern_operator_name` is transformed into `ExternOperatorName`.

```
abstract.english
  \OpRef{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
                  g_o >= g_m + Offset
  @l\[
      g_{o} \ge g_{m} + \ParRef{Offset}
  \]
  @e
  For \ParRef{LightDark} = \ValRef{'dark'} the condition is:
  @a
                  g_o <= g_m - Offset
  @l\[
  g_{o} \le g_{m} - \ParRef{Offset}
  \]
  @e
  Finally, for \ParRef{LightDark} = \ValRef{'equal'} it is:
  @a
                  g_m - Offset <= g_o <= g_m + Offset
  @l\[
  g_{m} - \ParRef{Offset} \le g_{o} \le g_{m} + \ParRef{Offset}
  \]
  @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 on page 35 for the special syntax of text.

2.2.5 module

```
module
  foundation;
```

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

2.2.6 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 HDevelop.

```
chapter.english
  Segmentation;
```

The structuring can be refined by specifying a section: `chapter[,section]`, e.g.,

```
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.

The chapters and sections created using the slot `chapter` may be supplemented by an optional chapter description as described in section 2.5 on page 38.

2.2.7 functionality

```
functionality
  image;
```

This slot denotes the class of which the operator should become a method within an object-oriented programming language like C++ or 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, xld_ext_para
```


2.2.8 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.

```
keywords.english
  threshold, gray-value threshold, dynamic threshold, local threshold;
```

2.2.9 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.

```
predecessor
  mean_image,smooth_image,gauss_image;

successor
  connection,select_shape,reduce_domain,select_gray,rank_region,
  dilation1,opening;

alternatives
  highpass,threshold,background_seg;
```

2.2.10 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,dilation1;
```

2.2.11 attention

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

[2] User-defined extensions are not inserted in the class hierarchy but considered as global methods.

```
attention.english
  If \ParRef{Offset} is chosen from @a -1 .. 1 @l$-1\dots1$@e
  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).;
```

2.2.12 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 on page 35.

```
result_state.english
  \OpRef{dyn_threshold} returns 2 (H\_MSG\_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.;
```

2.2.13 parallelization

The slot `parallelization` results in the manual entry `Execution Information`. It contains information about the timeout and parallelization characteristics of the operator. 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 HALCON uses to speed up the operator's processing when working with multi-processor or multi-core 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 and domain 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, split_domain;
  region_postprocessing: domain_concat_si_inp;
```

- `process_exclusively:  true, false, none;`
  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 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 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).

  If the subslot contains `none`, the operator is processed by HALCON independently of other operators (even exclusive ones). This value is only used internally by MVTec.

- `process_mutual: true, false;`
  This subslot of `parallelization` assigns whether 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 (but other HALCON operators can).

  An mutually 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 ('true') by a program thread, or whether it may called by any ("external") thread without problems.

  If `process_locally` is set to 'true', this signals the programmer that the corresponding operator must be used carefully within multithreaded applications. The simplest 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, HALCON processes an operator *without* parallelization regardless 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, split_partial, split_partial_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:
  - `none`: HALCON does not automatically parallelize the operator.
  - `split_tuple`: 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`: HALCON parallelizes by splitting every multichannel (input) image into several subsets of the channels; the subsets are then processed in parallel.
  - `split_domain`: HALCON parallelizes by splitting the domain of every input image into several parts; the parts are then processed in parallel.
  - `split_partial, split_partial_domain`: These methods are only used by MVTec to indicate the internal parallelization of HALCON operators.

  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 31)!

  If `method` is set to 'split_domain', the slot `region_postprocessing` (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 `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 `method`, no other string should be specified with it in order to keep the description consistent.

  If no slot `method` is filled in with an operator, it is assumed that the operator is *not* suitable for being automatically parallelized ("pessimistic" assumption).

- `region_postprocessing: none, domain_concat_si_inp,domain_concat_mult_inp;`
  This subslot of `parallelization` must (only) be set for operators that are suitable for being parallelized by splitting the domain of input objects (i.e. slot `method` contains 'split_domain'). For such operators HALCON supports a simple postprocessing of output region components: After splitting the domain of input objects and processing the operator in parallel (within the subparts of the original domain), HALCON

creates the overall output iconic objects based on the output objects of the parallel processing. At this, it supports three methods for determining the domain/region components of overall output objects. Those three methods correspond to the three possible values of `region_postprocessing`:

- `none`: no postprocessing of region/domain components. Here, HALCON directly adopts the region components of input objects and re-uses them unchanged as the region components of the corresponding overall output objects. This method is especially suitable for operators which output solely *image* objects and where the domain is left unchanged, such as filter operators (e.g. `mean_image`).
- `domain_concat_si_inp`: postprocess region components by **concat**enation for operators which have only a **si**ngle **in**put iconic parameter. Here, HALCON creates the region components of *overall* output objects by concatenating the region components of the corresponding output objects of the parallel processing. At this, the latter regions must not overlap! Therefore, `domain_concat_si_inp` can only be used for operators, for which the subslot `domain_split` (see below) is set to '0'. This method is suitable, for example, for the operator `threshold` which outputs a region containing all pixels of a certain gray value range. Here, HALCON splits the input domain into disjunctive parts, processes the threshold operator in parallel, and finally concatenates the single result regions to one overall result region.
- `domain_concat_mult_inp`: postprocess region components by **concat**enation for operators which have **mult**iple **in**put iconic parameter and which need a specific handling of the input objects. This method is quite similar to that above, i.e., HALCON also creates the region components of *overall* output objects by concatenating the region components of the corresponding output objects of the parallel processing. Again, the latter regions must not overlap! Therefore, `domain_concat_mult_inp` can only be used for operators, for which the subslot `domain_split` (see below) is set to '0'. For `domain_concat_mult_inp` HALCON uses a specific handling of input objects. Normally, HALCON splits the domains of *all* objects of all input iconic parameters in the same manner (e.g. by splitting into the first and the second half of the chords, independent from the row index of the chords). This does not mean any problem as long as all input image domains are the same — especially, if domains of corresponding objects of *different* input parameters are the same. Because then also the *split* domains will match each other. This is not the case, if we have to expect that domains of objects of different input iconic parameters may differ. Here, it may happen that split domain parts of the first parameter and that of the second parameter do not overlap because of the simple splitting mechanism which does not take into account the row index of chords. If so, and if an operator uses, for example, an intersection of the input domains of different input parameters for defining the valid domain of the processed operation, it would get an empty region — in contrast to the sequential processing of the same operator. Therefore, we must assign `domain_concat_mult_inp` for such operators. This will force HALCON to only split the domains of the objects of the *first* input iconic parameter. The domains of input objects of all other input iconic parameters are left unchanged so that they fit to the domains of the first input iconic parameter regardless of their position in the image.
  This method is suitable, for example, for the operator `add_image` which adds two images and sets the domain of the result image to the intersection of the domains of the two input images. By defining `domain_concat_mult_inp` for `add_image`, HALCON only splits the domain of the first image to add, then calculates the intersection of the split domain parts with the unchanged domain of the second image in parallel, performs the addition of gray values in parallel, and finally merges the resulting region components (domains) by concatenation into the region component of the overall output image which also contains the added gray values.

- `abort_mode: break, cancel, both;`
  This subslot of parallelization must only be used to determine the supported timeout mode. In case no timeout is supported, this slot must be omitted. For an explanation of the different modes, see the reference manual entry for `set_operator_timeout`.

Though, we recommend to specify the whole slot `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 *reentrant*, i.e. it is *not* processed under (complete or mutual) exclusion, and that it must be processed "locally". The latter also means that *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).

2.2.14 complexity

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

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

2.2.15 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.

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

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

The extension `.trias` indicates a HDevelop example. Moreover, the extensions `.c`, `.cpp`, and `.c#` 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, C++, and C# examples.

2.2.16 references

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

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

2.2.17 class Information

At the end of the DEF file, i.e., after the parameter section described in the following section, information about the classes generated for HALCON/.NET must be added, in particular the COM class and library IDs in the subslots `iid`, `clsid`, and `libid`. The example below shows the corresponding information in `threshold.def`.

```
class halconuser <- user

short.english
  Sample extension package.;

iid
  {bf6a1001-0b95-11d4-b295-00e0293dc2ac};

clsid
  {bf6a1002-0b95-11d4-b295-00e0293dc2ac};


library halconuser <- user_library

libid
  {bf6a1000-0b95-11d4-b295-00e0293dc2ac};
```

2.3 Describing the Parameters

Please note that the number of parameters is limited for an extension package operators as listed in the Installation Guide, section 1.5.6 on page 13.

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.

2.3.1 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.

Note that in order to create PDF hyperlinks referencing parameters (see section 2.4 on page 35 and section 7.1.4 on page 99) a parameter name must consist of alphanumerical characters only!

2.3.2 default_type

`default_type:` `handle, integer,real,string;`

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

2.3.3 sem_type

`sem_type:` *class[.spec]*;

This slot determines the semantic type, i.e., it 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. A list with the semantic types is given in appendix B on page 141. Some exemplary semantic types (classes) are shown below.

- **Iconic data (object parameters):**
  - `object` (any iconic object: images, regions, XLDs)
  - `image` (images)
  - `region` (regions)
  - `xld` (any XLDs: lines in e**X**tended **L**ine **D**escription)
  - `xld_cont`, `xld_poly`, `xld_para`, `xld_mod_para`, `xld_ext_para`
- **Elementary data (control parameters):**
  - `number` (unspecific)
  - `integer`, `real`, `string`
  - `channel` (channel number)
- **Handles (control parameters):**
  - `window` (HALCON graphics window)
- **Arrays (control parameters):**
  - `histogram.values` (gray value histogram)
  - `distribution.values` (distribution)
- **Geometric data (control parameters):**
  - `point.x`, `point.y` (position; (column, row) except for 2D transformations to get mathematically positive rotation)
  - `extent.x`, `extent.y` (dimensioning)
  - `angle.deg`, `angle.rad` (angle given in degrees or radians, respectively)
  - `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 2^{nd} corner)
  - `rectangle2.center.y, rectangle2.center.x` (rectangle with arbitrary orientation: center),
    `rectangle2.angle.rad` (orientation – in radians),
    `rectangle2.hwidth, rectangle2.hheigth` (half the size)
  - `ellipse.center.y, ellipse.center.x` (ellipse: center),
    `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).
  - `hom_mat2d` (2D homogeneous transformation matrix)
    `hom_mat3d` (3D homogeneous transformation matrix)
    `pose` (3D pose)

- **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` (polygonal representation)
  - `chain.begin.y, chain.begin.x, chain.code` (chain code)

- **Miscellaneous (control parameters):**
  - `filename` (name of a file)
    `filename.read` (name of an input file)
    `filename.write` (name of an output file)
  - `attribute.name` (name of a generic parameter)
    `attribute.value` (value of a generic parameter)

Please note:

- 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 modified

```
modified:          false,true;
```

This slot is used to indicate input parameters that may be modified by the operator, e.g., handles. If `modified` is set to `'true'`, a respective note is automatically created in the corresponding reference manual entry.

2.3.5 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. If a parameter may also return no values at all, `multivalue` must be set to 'true' or 'optional'.

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:

2.3.6 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 25). 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 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 `optimize_aop` 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, HALCON will take into account this parameter during its hardware check (`optimize_aop`) 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 25).

2.3.7 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;
```

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 25). 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).
Altogether, 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;
- `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'.

```
parameter
  Circularity:            output_control;
  description.english:    Roundness of the input region(s).;
  sem_type:               real;
  type_list:              real;
  default_type:           real;
  multivalue:             optional;
  assertion:              0 <= Circularity && Circularity <= 1.0;
  postprocessing:         tuple_concat;
```

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 25).

2.3.8 description

`description.english:` *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 *LaTeX/Ascii-Text* see section 2.4 on page 35.

2.3.9 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:` `handle, integer,real,string;`

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

2.3.10 default_value

`default_value:` *Default*;

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`').

2.3.11 values

`values:` *Suggested 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.

2.3.12 value_list

`value_list:` *List of 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.

2.3.13 value_min, value_max

`value_min:` *Number*;
`value_max:` *Number*;

Instead of a list an interval may be used to specify a range of 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.

Please note that the range is not used to check the passed parameter values. The slots are provided for applications that need the possible range of values to create sliders or other graphical elements to interact with parameters.

To restrict the values of a parameter, please use the slot `assertion` further below.

2.3.14 step_rec, step_min, value_function

`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`.

2.3.15 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.

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.

Group	Symbol	meaning
Unary Operators	`!`	logical NOT
	`-`	negation
Binary Operators	`&&`	logical AND
	`||`	logical OR
	`==`	equal
	`!=`	not equal
	`>`	greater
	`>=`	greater or equal
	`<`	less
	`<=`	less or equal
	`+`	sum
	`-`	difference
	`*`	multiplication
	`/`	division (DIV)
	`%`	rest of division (MOD)
Functions	`odd`	true, if odd
	`even`	true, if even
	`ld`	binary logarithm
	`length`	length
	`number`	number of values (only for `assertion`)
	`width`	(image-)width
	`height`	(image-)height

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

2.3.16 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:

- `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.

2.3.17 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.

2.3.18 multiinstance

```
multiinstance:     true,false,optional;
```

This slot is used for semantic tuples. A semantic tuple is a tuple of values that, put together, describes a single instance, e.g., a pose (consisting of seven values) or calibration parameters (consisting of several values, depending on the camera type).

The slot `multiinstance` is only used if `multivalues = true`. As for one instance multiple values need to be permitted you can use `multiinstance` in order to specify if multiple semantic tuples may be passed (`true`) or not (`false`). `optional` specifies operators that can work on both one or more semantic tuples.

2.3.19 file_ext, file_ext_descr

```
file_ext:                 List of file extensions;
file_ext_descr.english:   List of descriptions;
```

These slots are used for parameters of the semantic type `filename` (see section 2.3.3 on page 29). In `file_ext`, you can specify a list of possible file extensions. HDevelop, for example, uses this information when opening file selectors.

In `file_ext_descr.english` (and `file_ext_descr.german`), you can add descriptions for the extensions. If `file_ext` contains a list of extensions, you can either pass a single description for all of them or one for each extension. When passing a list of descriptions, you can insert a period to indicate that the previous value should be used. If a description contains a space, you must enclose it in single quotes. Note that HDevelop automatically appends the text `Files` to the descriptions.

Example 1:
```
file_ext:                 tif,tiff,gif;
file_ext_descr.english:   Image;
```
results in:
```
Image Files (*.tif; *.tiff; *.gif)
```

Example 2:
```
file_ext:                 tif,tiff,gif;
file_ext_descr.english:   'TIFF Image',.,'GIF Image';
```
results in:
```
TIFF Image Files (*.tif)
TIFF Image Files (*.tiff)
GIF Image Files (*.gif)
```

2.4 Text in DEF Files

Note that DEF files with non-ASCII characters should be UTF-8 encoded.

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 PDF reference 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 reference 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:

Quotation marks: There are two kinds of quotation marks

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

"Text": This notation must be used in all other cases, particularly when quoting a word or text segment because of its content. So `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: Underscores can be used directly for both the ASCII and the LaTeX output. The HALCON compiler `hcomp` automatically prefixes them with the necessary backslashes for the LaTeX output (`\_`).

Please note that if you want to use an underscore in a LaTeX part to create a subscript, you must encase the subscript text in curly braces ({}) even if the subscript is only a single character. For example, to create the output "S_x" you must write `$S_{x}$`. If you leave out the braces, an underscore is printed, e.g., `$S_x$` results in "S_x".

Backslash: All backslashes (\) within the text are ignored while generating the ASCII version. So the LaTeX-linefeed-symbol (\\) can be used without any problems. To embed a literal backslash in the output, use `\textbackslash{}` in the DEF file.

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_{b}`.

HALCON: The string HALCON is generated by `\Halcon`. Note that in the PDF files derived from the generated LaTeX version of the DEF files, *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 `\Halcon\` instead. So "`\Halcon XYZ`" results in "HALCON XYZ" and "`\Halcon\ XYZ`" results in "HALCON XYZ". With the ASCII version all backslashes are ignored (see above).

Names of parameters and operators: Any reference to extension package operators or their parameters must be written as `\OpRef{`*operator_name*`}`, respectively as `\ParRef{`*parameter_name*`}`. These keywords and the brackets are ignored, when generating the ASCII version so that only *operator_name* or *parameter_name* remains in the ASCII file. In the HTML and PDF documents, hyperlinks 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 *parameter_value* remains in the ASCII file.

Paragraphs: If you want to divide a longer text into paragraphs to increase the readability, all you need to do is to insert an empty line between the paragraphs:

```
This is a paragraph ...
.. which ends here.

The next paragraph starts after an empty line ...
```

This corresponds to the standard LaTeX way of separating paragraphs; in the HTML documents, the paragraphs are enclosed with the tag <P>.

Chapter references: You can add a link to a certain chapter or section using `\ChapRef{`*chapter*`}` or `\ChapRef{`*chapter*, *section*`}`, respectively. The chapter/section names must match the English names of an existing chapter definition, see section 2.2.6 on page 22.

Formatted text: You can use the following LaTeX commands to format text both in the LaTeX and HTML version:

- `\emph{text}`: results in *text* (HTML tag <E>)
- `\texttt{text}`: results in `text` (HTML tag <TT>)
- `\textbf{text}`: results in **text** (HTML tag <B>)
- `\textit{text}`: results in *text* (HTML tag <I>)

Please note that these commands cannot be nested!

Links: The following special environment adds a hyperlink to the specified URL into the HTML version of the documentation. In the LaTeX version, the link text appears as plain text.

```
\begin{External}{URL}
  Link text...
\end{External}
```

Lists: You can use the following LaTeX environments to format text in lists both in the LaTeX and HTML version:

description lists:

```
\begin{description}
  \item[title 1] text ...
  \item[title 2] text ...
\end{description}
```

lists with bullet points:

```
\begin{itemize}
  \item text ...
  \item text ...
\end{itemize}
```

enumerated lists:

```
\begin{enumerate}
  \item text ...
  \item text ...
\end{enumerate}
```

Paragraph alignment: You can use the following special environment to specify the alignment of the included text. The parameter `format` may be set to `left`, `center`, or `right`.

```
\begin{Alignment}{format}
...
\end{Alignment}
```

Tables: You can use the following special environment to format text in tabular format both in the LaTeX and HTML version. This table environment is not to be confused with the standard LaTeX `table` or `tabular` environment. Tables are centered by default. To change the alignment of a table, enclose it in the `Alignment` environment (see above).

```
\begin{Table}[style]{format}
  column & column & .. \\
  column & column & ..
\end{Table}
```

Columns are separated by `&`, and a new row is started with `\\`. Note that the last row of the table has no trailing `\\`.

The `style` specification (including the square brackets) is optional. The following values are available:

`layout`: Simple table without any borders. This is the default if no table style is specified.

`grid`: Each table cell has a border.

`table`: The first row is formatted as the table header. The header and the bottom of the table are marked with horizontal lines.

The `format` specification defines the number of columns, their alignment, and (optionally) their width. Each occurrence of one of the letters `l`, `r`, or `c` defines a left-aligned, right-aligned or centered column, respectively. The column specifier may be followed by a number which defines the relative width of the column in percent. Note that the alignment of a column is disregarded in the LaTeX version if the width is specified. In that case the text of the corresponding column is justified.

For readability, the column specifications may be separated by commas. Extra spacing is ignored. Thus the following table specifications are equivalent:

```
\begin{Table}{l 40, c 30, r 30}

\begin{Table}{l40c30r30}
```

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

```
@a                g_o >= g_m + \ParRef{Offset}
@l\[ g_{o} \ge g_{m} + \ParRef{Offset} \]@e
```

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

```
... text text @l \aSpecialLatexCommand @e text text ...
```

In fact, the ASCII mode has two sub-modes: If `@a` is immediately followed by a carriage return, the following lines are enclosed with the HTML tag <PRE>. Thus, they appear "as they come", including all spaces and carriage returns. Otherwise, i.e., if `@a` is followed by any character (including a space!), carriage returns and spaces are not preserved in the HTML document.

2.5 Chapter Description

Chapters and sections define the structuring of HALCON operators. Sometimes, it is desirable to add an introductory text to a certain chapter or section, e.g., to explain some common concepts of the contained operators.

These chapter descriptions are defined at an arbitrary place in the DEF file outside of an operator description. They are linked to the corresponding chapter using a header that lists the English chapter name, and optionally the English section name:

```
chapter_ref chapter[,section] <- user_chapter
```

The chapter/section names must match the English names of an existing chapter definition, see section 2.2.6 on page 22.

The text of the chapter description is given within the slot `abstract`. It follows the rules of the slot `abstract` used in the operator descriptions. The syntax is explained in section 2.2.4 on page 21.

Thus, a full chapter description looks like this:

```
chapter_ref Filter,Edges <- user_chapter

abstract.english
  Text of the chapter description
  ...
  ;

abstract.german
  ...
  ;

...
```

Again, refer to section 2.4 on page 35 for the special syntax of text.

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 on page 41.

3.1 Basic Numeric Data Types

This section describes the usage of basic numeric data types. Since C made no assumption about the size of integers prior to the introduction of the C99 standard, HALCON versions 20.11 and earlier define the following types:

Type Name	C99 Type Name	Size/Range of Values
`INT1`	`int8_t`	one byte with sign
`HBYTE`	`uint8_t`	one byte, unsigned
`UINT1`	`uint8_t`	the same as `HBYTE`
`INT2`	`int16_t`	two bytes with sign (generally `short`)
`UINT2`	`uint16_t`	two bytes, unsigned
`INT4`	`int32_t`	four bytes with sign
`UINT4`	`uint32_t`	four bytes, unsigned
`INT4_8`		four (on 32 bit systems) or eight (on 64 bit systems) bytes with sign
`UINT4_8`		four (on 32 bit systems) or eight (on 64 bit systems) bytes, unsigned
`HINT`	`int`	At least two, maximal eight bytes with sign
`HUINT`	`unsigned`	At least two, maximal eight bytes, unsigned

Using these types ensures the portability of HALCON operators. If the extension package need not be compilable for HALCON 20.11 or earlier versions, the C99 data types (defined in the `stdint.h` header file) should be used where applicable. Note that C99 does not define any types that exactly match the `INT4_8` or `UINT4_8` types.

3.1.1 Attributes of Iconic Parameters

To use operators in HALCON and HALCON XL, the following types have been introduced. With a compiler define (see chapter 7 on page 97), they are mapped to the suitable basic types for the corresponding image size.

```
#if !defined(HC_LARGE_IMAGES)
typedef int32_t HIMGDIM;  /* Image dimension, e.g., width and height */
typedef int16_t HIMGCOOR; /* Image coordinates, e.g, row and column */
typedef int32_t HLINCOOR; /* Linearized image coordinates */
typedef int32_t HIMGCNT;  /* Number of pixels, e.g., area or buffer size */
typedef int32_t HITEMCNT; /* Number of elements (contours or runlengths) */
typedef float   HSUBCOOR; /* Sub-pixel precise coordinates */
typedef float   HSUBATTR; /* Sub-pixel precise attributes of contours */
#else
typedef int32_t HIMGDIM;
typedef int32_t HIMGCOOR;
typedef INT4_8  HLINCOOR;
typedef INT4_8  HIMGCNT;
typedef INT4_8  HITEMCNT;
typedef double  HSUBCOOR;
typedef double  HSUBATTR;
#endif
```

Examples for their use can be found in chapter 4 on page 49.

3.1.2 Local Variables / Temporary Results

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

Type	Size	C-Data Type
Integer	1 byte	`int` (HINT) or `unsigned` (HUINT)
	2 byte	
	3 byte	`INT4_8` or `UINT4_8`
	4 byte	
Floating-point Number		`double`

3.1.3 Procedure Parameters

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

Type	Size	C-Data Type
Integer	1 byte	`HINT`
	2 byte	
	3 byte	`INT4_8`
	4 byte	
Floating-point Number		`double`
Structure		pointer

3.1.4 Arrays

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

Type	Size	C-Data Type
Integer	1 byte	`int8_t`
	2 byte	`int16_t` or `uint16_t`
	3 byte	`int32_t` or `uint32_t`
	4 byte	
Floating-point Number	4 byte	`float`
	8 byte	`double`
Structure		pointer

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 HALCON uses multithreading to exploit multi-processor or multi-core hardware. This means, that 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

```
HAllocTmp, HFreeTmp, HFreeNTmp, HFreeUpToTmp, HFreeAllTmp,
HAllocRLTmp, HAllocRLNumTmp, HFreeRLTmp
```

Synopsis

```
#include "Halcon.h"

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

Herror HFreeTmp(        Hproc_handle proc_handle,
                        void         *pointer)

Herror HFreeUpToTmp(    Hproc_handle proc_handle,
                        void         *pointer)

Herror HFreeAllTmp(     Hproc_handle proc_handle)

Herror HAllocRLTmp(     Hproc_handle proc_handle,
                        Hrlregion    **region)

Herror HAllocRLNumTmp(  Hproc_handle proc_handle,
                        Hrlregion    **region,
                        size_t       size)

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

Figure 3.1: HALCON stack management for temporary data.

Figure 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 it's 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. 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 currently stored in the HALCON database of iconic objects (the minimal size is DEF_RL_LENGTH = 50000 chords). Using HIncrRL (section 5.6.5 on page 84), HAllocRLTmp allocates more memory. HAllocRLNumTmp allows you to determine the size of memory to be allocated by specifying the number of chords. Furthermore, HAllocRLTmp and HAllocRLNumTmp initialize the data structure Hrlregion for the new region (see section 4.2 on page 50 for a description of Hrlregion). 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

Names

```
HAllocLocal, HReallocLocal, HFreeLocal,
HAllocRLLocal, HAllocRLNumLocal, HReallocRLNumLocal, HFreeRLLocal
```

Synopsis

```
#include "Halcon.h"

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

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

Herror HFreeLocal(          Hproc_handle proc_handle,
                            void         *pointer)

Herror HAllocRLLocal(       Hproc_handle proc_handle,
                            Hrlregion    **region)

Herror HAllocRLNumLocal(    Hproc_handle proc_handle,
                            Hrlregion    **region,
                            size_t       num)

Herror HReallocRLNumLocal(  Hproc_handle proc_handle,
                            Hrlregion    *region
                            size_t       num_new,
                            Hrlregion    **new_region)

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

Figure 3.2: 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 figure 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 HReallocRLNumLocal, and HFreeRLLocal are used for handling temporary region data based on HAllocLocal. Otherwise their behavior is similar to HAllocRLTmp etc., see above. E.g., using HIncrRL (section 5.6.5 on page 84), HAllocRLLocal allocates more memory. 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.

Handle Data

Name

HAllocOutputHandle

Synopsis

```
#include "Halcon.h"

Herror HAllocOutputHandle(        Hproc_handle    proc_handle,
                                  int             par,
                                  Hphandle        elem,
                                  H_HANDLE_TYPE   HHandleInfo)
```

Figure 3.3: Memory management for handles.

The routine HAllocOutputHandle allocates one handle in the output control parameter par. HHandleInfo holds handle type information. elem is modified to point to where the new tool can be allocated.

The output control variable gains ownership of the pointer in elem. When aborting an operator with an error, the value must not be cleared inside the operator's code, but will be cleared automatically when cleaning the output control variable.

3.2.2 Permanent Data

The HALCON Extension Package Interface also provides routines for permanently allocating/deallocating memory (see figures 3.4 and 3.5). 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 database of iconic objects are allocated with this routine (see chapter 5 on page 57 and chapter 6 on page 87).

HNewImage allocates raw image data inside[1] the data structure Himage (see section 4.1 on page 49) 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 HReadGV.

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

with init_new_img of type bool and *set*

```
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 afterward, 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

[1] So HNewImage does not allocate memory for the Himage structure itself.

Names

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

Synopsis

```
#include "Halcon.h"

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

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

Herror HFree(          Hproc_handle  proc_handle,
                       void          *pointer)

Herror HNewImage(      Hproc_handle  proc_handle,
                       Himage        *image,
                       int           kind,
                       HIMGDIM       width,
                       HIMGDIM       height)

Herror HNewImagePtr(   Hproc_handle  proc_handle,
                       Himage        *image,
                       int           kind,
                       HIMGDIM       width,
                       HIMGDIM       height,
                       void          *data,
                       bool          initImg)

Herror HAllocXLDCont(  Hproc_handle  proc_handle,
                       Hcont         **cont,
                       HITEMCNT      num_points)

Herror HFreeXLDCont(   Hproc_handle  proc_handle,
                       Hcont         *cont)
```

Figure 3.4: General memory management within HALCON.

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 on page 67).

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

`HAllocXLDCont` allocates an XLD contour[2] of type `Hcont` (see section 4.3 on page 52) including memory for `num_points` contour points. Internally this routine is based on `HAlloc`. `HFreeXLDCont` is used to deallocate a contour again (including all the points *and* all attributes defined for `cont`, see page 54).

The convenience routines `HAllocRL` , `HAllocRLNum`, `HReallocRLNum`, and `HFreeRL` (see figure 3.5) are used for handling permanent region data based on `HAlloc`. Otherwise, their behavior is similar to `HAllocRLTmp` etc. E.g., using `HIncrRL` (section 5.6.5 on page 84), `HAllocRL` allocates more memory, see page 42. 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` `HAllocXLDCont` also allocates memory for the structure `Hcont` itself, not only for the data inside that structure.

Names

```
HAllocRL, HAllocRLNum, HReallocRLNum, HFreeRL
```

Synopsis

```
#include "Halcon.h"

Herror HAllocRL(      Hproc_handle proc_handle,
                      Hrlregion    **region)

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

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

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

Figure 3.5: General memory management for region data.

3.2.3 Debugging

The HALCON memory management provides debugging mechanisms. The debugging is enabled with `set_check ('memory')`, and can be switched off with `set_check('~memory')`. Every time a memory block is deallocated a couple of consistency checks are performed automatically. Moreover, the consistency of memory blocks can be checked for debugging reasons at any time using specific Extension Package Interface routines, see figure 3.6.

Names

```
HTestMem, HTestPtr, HTestAllTmp, HTestTmp
```

Synopsis

```
#include "Halcon.h"

Herror HTestMem(    void)

Herror HTestPtr(    void          *pointer)

Herror HTestAllTmp( Hproc_handle  proc_handle)

Herror HTestTmp(    Hproc_handle  proc_handle,
                    void          *pointer)
```

Figure 3.6: HALCON memory management: Check of consistency.

`HTestMem` checks all the memory allocated with `HAlloc` or `HAllocLocal`, whereas `HTestPtr` checks only the specified memory block, which must have been allocated with `HAlloc`. 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 code (`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-like systems) or within an alert box (Windows). 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. Afterward, it calls the action procedure, which performs the image processing. In some cases, it can 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 iconic 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 `Hproc_handle` 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 Extension Package Initialization

To initialize an extension package, the following optional function may be defined:

```
extern HUserExport Herror HXPkgMain(Hproc_handle)
```

If defined, HALCON will call this function upon loading the extension package. For example, this mechanism is useful to initialize synchronization objects (e.g., mutexes) that will be used in the extension package.

3.5 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`" (**i**mage **p**rocessing procedure), "`IO`" (**i**nput/**o**utput procedure), or "`DB`" (**d**ata **b**ase 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
```

3.6 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 code returned by the procedure (see next section).

3.7 Error Handling

3.7.1 Defining Own Error Codes

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

The file `%HALCONROOT%\include\HErrorDef.h` contains predefined error messages for all typical errors, see also appendix A on page 105. 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 codes that should be greater than 10 000 to avoid re-using an already allocated code or a code 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 code>) .
```

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

```
Herror HSetErrText(char *error_message)
```

can be called immediately before the operator returns the error code with `return H_ERR_*`. In this case the error message `error_message` will be displayed by the *next*[3] call of the HALCON operator `get_error_text`.

Error codes 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_TRUE  /* no error (supply procedure)              */
H_MSG_OK    /* no error (any other procedure)           */
```

All HALCON procedures return an error code. This code has to be checked after calling any HALCON procedure, e.g., by encapsulating the procedure call in the macro `HCkP`. This macro checks the return value and exits the current procedure if an error occurred (a detailed description of the macro is given in the section 5.6.1 on page 83). Please note that some of the Extension Package Interface macros described in chapter 5 on page 57 and chapter 6 on page 87 (e.g., `HGetCPar`) implicitly use the macro and therefore automatically return a proper error code.

3.7.2 Extending Error Codes

There is also a mechanism to add some extended error information about the thread's current HALCON error. The function `HSetExtendedErrorInfo`

```
Herror HSetExtendedErrorInfo(Hproc_handle proc_handle, INT4_8 error_code,
                             char const *error_message)
```

[3] Within this call the error message is reset to "No error message available ... " again to ensure that the user-defined error message is up to date.

can be called immediately before the operator returns the error code with return `return H_ERR_*`. In this case the extended error information (`error_code` and `error_message`) will be set thread-locally and can be queried by the next call of the HALCON operator `get_extended_error_info`. The set information gets cleared by both the next occurring HALCON error or the next call of `HSetExtendedErrorInfo`.

In `error_code`, a user-defined error code can be set, e.g., a 3^{rd} Party SDK error code or a `0` value (indicating that no extended error code is set).

In `error_message`, a user-defined error message can be specified, e.g., a 3^{rd} Party SDK error message or an empty string `""` (indicating that no extended error message is set).

An example of how to use `HSetExtendedErrorInfo` is provided in `$HALCONEXAMPLES/extension_package/useropencl/source/CIPOpenCLFilter.c`. An example of how to obtain the extended error information by means of `get_extended_error_info` is provided in `$HALCONEXAMPLES/extension_package/useropencl/examples/testuseropencl.hdev`.

3.8 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 `Himage` (cf. figure 4.1 on page 50).
- 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 on page 90. If an operator can only process a reduced set of types, this can be specified in the DEF file with `type_list` (cf. page 32). 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 on page 87 or `HAllObj` in section 5.3.1 on page 66). 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 on page 66).
- 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 database of iconic objects. 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 iconic 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 *channels*) can be combined to a multi-channel image. Each channel can be accessed separately, i.e. the channels are not interleaved. The structure `Himage` contains size, pixel type, and pixel data of one channel. Furthermore, a time-stamp is included. Figure 4.1 shows the corresponding type declaration.

The maximum for `width` and `height`, which describe the image size, is determined with `MAX_FORMAT` (HALCON: 32 768, HALCON XL: 1 073 741 824). 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 figure 4.2. The pixel types include basic types like `uint8_t`, `int8_t`, `int32_t`, and `float` as well as composed types. Their definitions can be seen in figure 4.3.

The pixel data is stored as a one-dimensional array (image 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 * image.width + C
```

This transformation is performed by the following macro `HLinCoor`:

```
L = HLinCoor(R,C,image.width);
```

There are two additional macros for computing rows (`HRow`) and columns (`HCol`):

```
R = HRow(L,image.width);
C = HCol(L,image.width);
```

Please see section 3.2.2 on page 43 for routines to allocate image data within `Himage`.

```
typedef union {
  uint8_t*       b;         /* 0..255 (BYTE_IMAGE)                          */
  uint8_t*       z;         /* 0..255 mod 256 (CYCLIC_IMAGE)                */
  uint8_t*       d;         /* orientation 0..180 (DIR_IMAGE)               */
  int8_t*        i;         /* -128..127 (INT1_IMAGE)                       */
  int32_t*       l;         /* 4 byte integer (LONG_IMAGE)                  */
  int64_t*       i8;        /* 8 byte integer (INT8_IMAGE)                  */
  float*         f;         /* 4 byte real (FLOAT_IMAGE)                    */
  HVFPixel*      vf;        /* vector field (VF_IMAGE)                      */
  HComplexPixel* c;         /* complex image (COMPLEX_IMAGE)                */
  HInt2Pixel     s;         /* 2 bytes with sign (INT2_IMAGE)               */
  HUInt2Pixel    u;         /* 2 bytes without sign (UINT2_IMAGE)           */
} HPixelImage;


typedef struct {
  int         kind;         /* pixel type                                   */
  HPixelImage pixel;        /* pixel data                                   */
  HIMGDIM     width;        /* image width                                  */
  HIMGDIM     height;       /* image height                                 */
  HImageFreeProc free_proc; /* function for deallocating image data         */
  bool        free;         /* free image data when deleting image          */
  /* time of creation of image                                              */
  uint16_t    msec;         /* milliseconds 0..999                          */
  uint8_t     sec;          /* seconds 0..59                                */
  uint8_t     min;          /* minutes 0.59                                 */
  uint8_t     hour;         /* 0..23                                        */
  uint8_t     day;          /* 1..31                                        */
  uint16_t    yday;         /* 1..366                                       */
  uint8_t     mon;          /* 1..12                                        */
  uint16_t    year;         /* 19xx                                         */
} Himage;
```

Figure 4.1: Data type `Himage` for images.

```
  #define BYTE_IMAGE    (int)1     /* 1 byte per pixel (0..255)      */
  #define INT4_IMAGE    (int)2     /* Type: 4 bytes per pixel (int32_t) */
  #define LONG_IMAGE    (int)2     /* 4 byte per pixel (int32_t)       */
  #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    /* -128..127                      */
  #define COMPLEX_IMAGE (int)128   /* 2 float images                 */
  #define INT2_IMAGE    (int)512   /* 2 bytes with sign              */
  #define UINT2_IMAGE   (int)1024  /* Type: 2 bytes (unsigned short) */
  #define VF_IMAGE      (int)2048  /* Two float images               */
  #define INT8_IMAGE    (int)4096  /* 8 byte per pixel (int64_t)       */
```

Figure 4.2: Definitions of pixel types.

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

```
typedef struct {
  float re;                /* real image part        */
  float im;                /* imaginary image part   */
} HComplexPixel;

typedef struct {
  int16_t* p;              /* gray values            */
  int8_t   num_bits;       /* number of used bits    */
} HInt2Pixel;

typedef struct {
  uint16_t* p;             /* gray values            */
  int8_t    num_bits;      /* number of used bits    */
} HUInt2Pixel;

typedef struct {
  float* row;              /* row direction          */
  float* col;              /* column direction       */
} HVFPixel;
```

Figure 4.3: Data types for `Himage`.

- chords are sorted in ascending order

If a region is read from the HALCON database of iconic objects, 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).

```
typedef struct {
   HIMGCOOR l;              /* line number (row) of run                */
   HIMGCOOR cb;             /* column index of beginning of run        */
   HIMGCOOR ce;             /* column index of ending of run           */
} Hrun;

typedef struct {
   bool        is_compl;    /* region is complement                    */
   HITEMCNT    num;         /* number of runs                          */
   HITEMCNT    num_max;     /* maximum number of runs                  */
   HRegFeature feature;     /* already processed features              */
   Hrun*       rl;          /* pointer on array of run lengths         */
} Hrlregion;
```

Figure 4.4: Data type `Hrlregion` for region encoding.

Figure 4.4 shows the type declaration for chords. In the structure `Hrlregion`, 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 section 3.2 on page 41). The flag `is_compl` allows an 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 figure 4.5) contains all region shape 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 `Hrlregion` is allocated with the procedure `HAllocRLTmp` or `HAllocRLNumTmp`, see section 3.2 on page 41. 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 `CB` and `CE` are very helpful, especially when processing gray values (in linear coordinates) along a chord. The programs in figure 6.5 on page 91 and figure 6.7 on page 93 illustrate how to use them. Their functionality can be seen in figure 4.6. The first parameter (`rl`) contains a pointer to the chords as it is used within `Hrlregion`. `index` specifies the index of the chord to work on and `width` contains the image width. `CB` returns the linear coordinate of the start point and `CE` that of the end point of the chord.

```
typedef struct {
  union {
    HFeatureFlags single;
    long          all;      /* if possible use 64 bits!
  }              def;
  uint8_t        shape;           /* SHAPE_*
  bool           is_convex;
  bool           is_filled;
  bool           is_connected4;
  bool           is_connected8;
  bool           is_thin;         /* one pixel thin
  double         circularity;
  double         compactness;
  double         contlength;
  double         convexity;
  double         phi;
  double         ra, rb;          /* elliptic_axis
  double         ra_, rb_;        /* elliptic_shape
  double         anisometry, bulkiness, structure_factor;
  double         m11, m20, m02, ia, ib;
  double         row, col;
  HIMGCNT        area;
  HIMGCOOR       row1, col1, row2, col2;
  double         row_rect, col_rect, phi_rect, length1, length2;
  double         row_circle, col_circle, radius;
  HIMGCOOR       min_chord, max_chord;
  HIMGCOOR       min_chord_gap, max_chord_gap;
  double         rectangularity;
} HRegFeature;
```

Figure 4.5: Data type `HRegFeature` for region shape features.

Names

CB, CE

Synopsis

```
#include "Halcon.h"

HLINCOOR CB(Hrun*    rl,
            HITEMCNT index,
            HIMGDIM  width)


HLINCOOR CE(Hrun*    rl,
            HITEMCNT index,
            HIMGDIM  width)
```

Figure 4.6: Linear coordinates from `Hrlregion`.

4.3 XLDs (`Hcont`, `Hpoly`)

XLDs (e**X**tended **L**ine **D**escriptions) are specific iconic HALCON objects to represent subpixel accurate contours and polygons. The corresponding data types `Hcont` and `Hpoly` are listed in figures 4.7 and 4.8.

HALCON contours of type `Hcont` contain an array of subpixel points along a contour. A contour might be classified concerning to topological considerations (`cont_class`). Additionally, an arbitrary number of additional contour attributes for each point along the contour can be included (`attribs`), e.g., the edge amplitude of subpixel edge points or the orientation of the local gradient. 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.

```
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,              /* end 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                  */
  HSUBATTR     *val;          /* value of the attribute (per point)     */
} Hcont_attrib;

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

typedef struct cont_type {
  HITEMCNT      num;          /* number of points along the contour     */
  HSUBCOOR*     row;          /* points / row indices (y coordinates)   */
  HSUBCOOR*     col;          /* points / column indices (x coord.)     */
  Hcont_class   cont_class;   /* contour class                          */
  int32_t       num_attrib;   /* number of additional attributes        */
  Hcont_attrib* attribs;      /* additional attributes (for each point) */
  int32_t       num_global;   /* number of additional global attributes */
  Hcont_global_attrib* global;/* additional attributes (per contour)    */
  int32_t       h;            /* auxiliary (temporary)                  */
} Hcont;
```

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

```
typedef struct lin_seg_type {
  HSUBCOOR     row,col;       /* a control point of the polygon:   */
                              /* row (y) and column (x) coordinate */
  HSUBATTR     length;        /* length of the line from the       */
                              /* current to the next point         */
  HSUBATTR     phi;           /* orientation (rad) of this line    */
  Hkey         ref;           /* database key of the underlying   */
                              /* contour                           */
  HITEMCNT     first;         /* index of the first point of the   */
                              /* underlying contour belonging to   */
                              /* current side of the polygon       */
  HITEMCNT     last;          /* index of the last contour point   */
} Hline_seg;

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

Figure 4.8: The XLD data type `Hpoly` for subpixel polygons.

Please see section 3.2.2 on page 43 for routines to allocate/deallocate contours. Two important auxiliary routines to copy contours are listed in figure 4.9 on page 54. `HCopyXLDCont` copies a complete contour, whereas `HCopyXLDContPart` 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).

Names

```
HCopyXLDCont, HCopyXLDContPart
```

Synopsis

```
#include "Halcon.h"

Herror HCopyXLDCont(     Hproc_handle proc_handle,
                         Hcont*       cont_in,
                         bool         preserve_attribs,
                         bool         preserve_global_attribs,
                         Hcont**      cont_out)

Herror HCopyXLDContPart( Hproc_handle proc_handle,
                         Hcont*       cont_in,
                         INT4_8       min_index,
                         INT4_8       max_index,
                         bool         preserve_attribs,
                         bool         preserve_global_attribs,
                         Hcont**      cont_out)
```

Figure 4.9: 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,
                            int32_t*     index)

HLookupXLDContAttrib(       Hcont*       cont,
                            char*        name,
                            int32_t*     index)

HAddXLDContGlobalAttrib(    Hproc_handle proc_handle,
                            Hcont*       cont,
                            char*        name,
                            int32_t*     indx);

HLookupXLDContGlobalAttrib( Hcont*       cont,
                            char*        name,
                            int32_t*     indx);
```

Figure 4.10: Handling contour attributes of contour pixels (within `Hcont`).

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 `HCopyXLDCont` and `HCopyXLDContPart` allocate the output contour *themselves*. So just pass a *pointer to a pointer* to `Hcont` without allocating any memory for `cont_out`. Both routines provide two additional parameters, `preserve_attribs` and `preserve_global_attribs`, to control whether the non-global and the global contour attributes should be copied.

The routines provided for handling contour attributes of contour points are summarized in figure 4.10. `HAddXLDContAttrib` is used to add a new class of contour 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: `HAddXLDContGlobalAttrib` 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 `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 `HLookupXLDContGlobalAttrib`. These routines return the index of the desired attribute within `attribs` or `global` of `cont`, or the error code `H_ERR_XLD_CAND` as result of the procedure call, if no attribute with the specified name is defined.

The XLD data type `Hpoly`, displayed in figure 4.8 on page 53, 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 database key.

4.4 Control Parameters

```
typedef union {
  INT4_8   l;     /* 4/8 byte integer */
  double   d;     /* 8 byte real */
  char*    s;     /* string */
  Hphandle H;     /* Pointer to handle */
} Hpar;

typedef struct {
  Hpar par;
  int  type;
} Hcpar;
```

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

HALCON's basic and essential view on control data are arrays. These arrays are used to pass control parameter values to supply procedures and can assume five different types, four basic or a mixed type, see section 2.3 on page 28.

Basic types are supported `INT4_8` arrays, coded as `LONG_PAR`, `double` arrays coded as `DOUBLE_PAR`, `handle` arrays coded as `HANDLE_PAR`, and string arrays, i.e., `char*` arrays, coded as `STRING_PAR`. Note that all strings passed to and from the HALCON library must be encoded in UTF-8. The legacy encoding mode (`set_system('filename_encoding','locale')`) is not supported for extension packages that use strings with non-ASCII characters.

The structure `Hcpar` enables HALCON to pass a mixed type (`MIXED_PAR`), holding any of the basic types in the union `Hpar`, encoded by the specifyer `type`. Figure 4.11 shows the corresponding definitions. Thus, it is possible to pass a native array type or to combine different types within an array of `Hcpar` values. For it, the selector `type` must be set to one of the basic type codes `LONG_PAR`, `DOUBLE_PAR`, `HANDLE_PAR` or `STRING_PAR`.

The Extension Package Interface procedures described in section 5.5 on page 72 are used to access the control parameters of HALCON operators and pass the control data arrays through the interface.

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 database.
- 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 database 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 database 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 iconic objects and writing output iconic parameters and control parameters are introduced.

Based on these low level routines a set of *convenience* routines are presented in chapter 6 on page 87. 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 figure 5.1. They form a basic interface to the HALCON database of iconic objects. Figure 5.2 illustrates how they are used to read image and region data from input iconic parameters.

5.1.1 `HGetObj`

`HGetObj` (see figure 5.1) returns the database key (type: Hkey) of an iconic object corresponding to the input object parameter with the index `par_num` of the HALCON operator.

Note that `HGetObj` automatically checks the result state of the underlying procedure with the macro HCkP (see section 5.6.1 on page 83). Thus, **you must not use** `HCkP` **on** `HGetObj` yourself.

Names

```
HGetObj, HGetComp, HGetRL, HGetImage, HGetXLD
```

Synopsis

```
#include "Halcon.h"

HGetObj(   Hproc_handle  proc_handle,
           int           par_num,
           INT4_8        obj_num
           Hkey          *obj_key)

HGetComp(  Hproc_handle  proc_handle,
           Hkey          obj_key,
           int           comp
           Hkey          *comp_key)

HGetRL(    Hproc_handle  proc_handle,
           Hkey          region_key,
           Hrlregion     *region)

HGetImage( Hproc_handle  proc_handle,
           Hkey          image_key,
           Himage        *image)

HGetXLD(   Hproc_handle  proc_handle,
           Hkey          obj_key,
           int           xld_type,
           Hpoly|Hcont   **xld)
```

Figure 5.1: Basic routines for accessing iconic input objects.

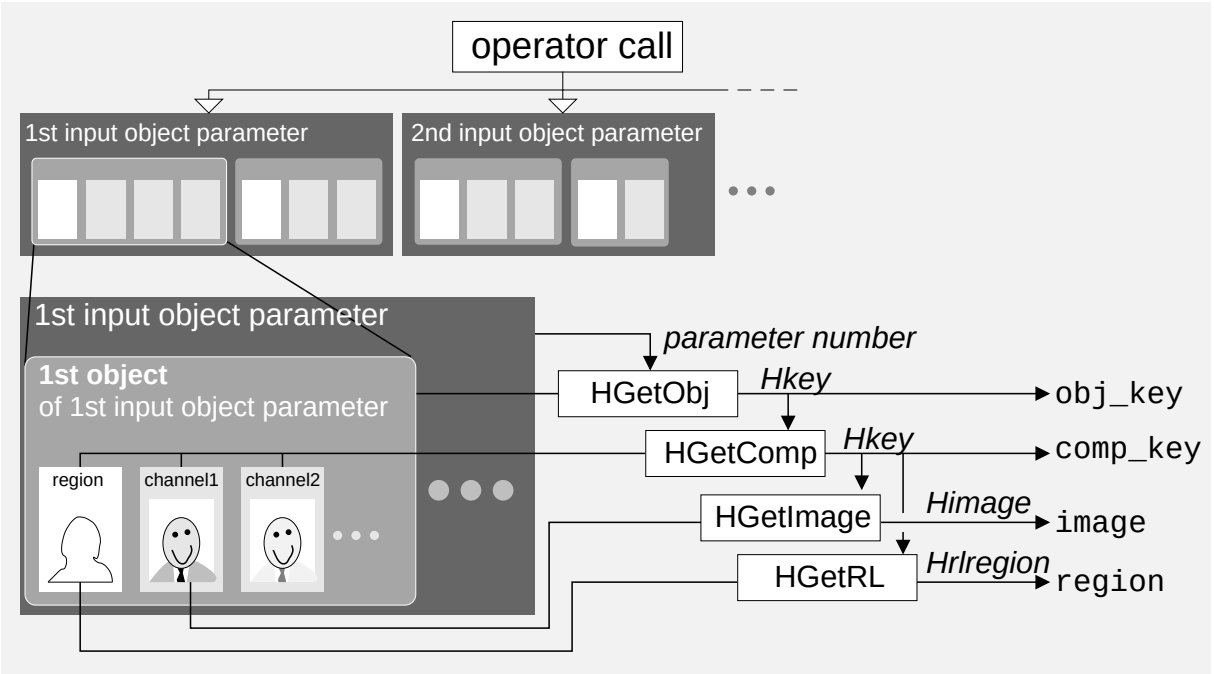

Figure 5.2: Direct access to input image objects.

The iconic object can either be a region (iconic object that contains only a region component), an image (iconic object that contains a region component and one or more image channels), or an XLD (iconic object that contains

```
int8_t   inp_pars;
int32_t  num_objs;
int      p,o;
Hkey     key;
...
/* get number of input iconic 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 iconic objects per input obj. parameter: */
   HReadGVA(proc_handle,HGnum_obj,&num_objs,p);
   for (o=1; o<=num_objs; o++) {
      HGetObj(proc_handle,p,o,&key); /* get key of iconic object   */
      /* further processing ... */
   }
}
```

Figure 5.3: Example for `HGetObj`: Database keys of all iconic objects of all input object parameters.

a contour or a polygon).

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 iconic object/control). The parameter `par_num` of `HGetObj` refers to this index, thus specifying the desired input iconic parameter. All *objects* passed within a single input iconic parameter are numbered from 1 to m, with the first iconic object in the list having the index 1. The parameter `obj_num` of `HGetObj` denotes the index of a desired iconic object within this list ($1 \leq$ `obj_num` $\leq m$).

The total number of input iconic parameters for an operator is specified in the corresponding DEF file[1], but the number of objects *within* an input iconic 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 index and `num_objs` is the desired number of iconic objects. As an alternative for reading the number of iconic objects of an input iconic parameter without the routine `HReadGVA`, you can use the macro `HGetObjNum` (see section 5.2.5 on page 65).

Note that a loop over all iconic objects passed to an operator within an input iconic object parameter is a very common task. Therefore, HALCON provides a macro for this problem: See `HAllObj` in section 5.3.1 on page 66.

5.1.2 HGetComp

`HGetComp` (see figure 5.1 on page 58) returns the database key of an image component (image matrix, i.e., channel: `image_key` or domain, i.e., area of definition: `region_key`) of an image object that is stored under the key `obj_key` in the HALCON database of iconic objects.

Note that `HGetComp` automatically checks the result state of the underlying procedure with the macro `HCkP` (see section 5.6.1 on page 83). Thus, **you must not use** `HCkP` **on** `HGetComp` yourself.

All components of an iconic object are consecutively numbered from 0 to n, with 0 denoting the region (domain) and $1 \ldots n$ 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 `HPNumOfChannels`, see figures 5.4 and 5.5:

[1] So this number is actually *known* by the programmer. However, using `HReadGV` it can be read from the operator context as well, see figure 5.3.

```
int     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(HPNumOfChannels(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`: Database keys of all regions and channels within an input image object.

Names

`HPNumOfChannels`

Synopsis

```
#include "Halcon.h"

Herror HPNumOfChannels( Hproc_handle proc_handle,
                        int          par_num,
                        INT4_8       obj_num,
                        int          *chn_num)
```

Figure 5.5: Auxiliary routine `HPNumOfChannels` .

```
HPNumOfChannels(proc_handle,par_num,obj_num,&obj_channels)
```

See also `HNumOfChannels` in section 5.6.4 on page 84 for a convenience version of this routine when dealing with the first input iconic parameter. More examples of how to use `HGetComp` can be found in figure 5.6 on page 61 and figure 5.7 on page 61.

5.1.3 `HGetRL`

`HGetRL` (see figure 5.1 on page 58) reads the runlength encoding of a region (type: `Hrlregion`, see section 4.2 on page 50) denoted by the database key `region_key` (type `Hkey`) from the HALCON database.

Note that `HGetRL` automatically checks the result state of the underlying procedure with the macro `HCkP` (see section 5.6.1 on page 83). Thus, **you must not use** `HCkP` **on** `HGetRL` yourself.

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 section 5.2.2 on page 62). However, this means that enough memory for `region` must be allocated *before* calling `HGetRL`, see section 3.2.1 on page 41.

Figures 5.6 and 5.7 show example applications of `HGetRL`.

```
Hrlregion  *region;
Hkey       obj_key;
Hkey       region_key;
...
HCkP(HAllocRLTmp(proc_handle,&region));
HGetObj(proc_handle,p,o,&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.

5.1.4 HGetImage

HGetImage (see figure 5.1 on page 58) reads the image data (type: Himage, see section 4.1 on page 49) of a specific gray value component (i.e., channel) of an image object referenced by its database key (type Hkey) in the HALCON database.

Note that HGetImage automatically checks the result state of the underlying procedure with the macro HCkP (see section 5.6.1 on page 83). Thus, **you must not use** HCkP **on** HGetImage yourself.

```
Hkey       obj_key;
Hkey       image_key;
Hkey       region_key;
Himage     image;
Hrlregion  *region;
...
HGetComp(proc_handle,obj_key,IMAGE_INDEX,&image_key);
HGetComp(proc_handle,obj_key,REGION,&region_key);
HGetImage(proc_handle,image_key,&image);
HGetRL(proc_handle,region_key,region);
switch (image.kind) {
  case BYTE_IMAGE:
    /* processing (in general this should   */
    /* be done calling an action procedure) */
    area = sum = 0;
    for (c=0; c<region->num; c++) {
      for (i=CB(region->rl,c,image.width);
           i<=CE(region->rl,c,image.width); i++, area++)
        sum += image.pixel.b[i];
    }
    average = sum / area;
    break;
  default: return(H_ERR_WIT);   /* wrong image type  */
}
```

Figure 5.7: Example for HGetImage: Average gray value of first channel.

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 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). Figure 5.7 shows an application of HGetImage.

5.1.5 HGetXLD

HGetXLD (see figure 5.1 on page 58) is used to access XLD objects in the HALCON database of iconic objects. The object of interest is specified by the database 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 database. Thus, **please avoid any write access to this data**. Otherwise you will encounter unpredictable side effects (changing other iconic objects). Figure 5.8 shows a simple example for how to use HGetXLD.

Note that HGetXLD automatically checks the result state of the underlying procedure with the macro HCkP (see section 5.6.1 on page 83). Thus, **you must not use** HCkP **on** HGetXLD yourself.

```
Hkey         obj_key;
Hcpar        num_points;
Hcont        *cont;

HGetObj(proc_handle,par_num,obj_num,&obj_key);
HGetXLD(proc_handle,obj_key,XLD_CONTOUR_ID,(void*)&cont);

/* processing (in general this should be done calling an action proc.) */
num_points.type  = LONG_PAR;
num_points.par.l = cont->num;
HPutCPar(proc_handle,1,&num_points,1);
```

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

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 figure 5.9. Basically they are combinations of the routines described in the previous section. These additional routines only require the database keys of the input iconic objects that can be extracted by using the routines HGetObj (see section 5.1.1 on page 57) or HAllObj (see section 5.3.1 on page 66). For HGetURL, even this step can be omitted.

5.2.1 HGetDRL

HGetDRL (see figure 5.9) combines HGetComp (section 5.1.2 on page 59) and HGetRL (section 5.1.3 on page 60). It reads the runlength encoding of a region (type: Hrlregion, see section 4.2 on page 50) specified by the database key obj_key of an image object. The region data is *copied* to region, which must have been allocated before with a suitable size (e.g., by using HAllocRLTmp, see section 3.2.1 on page 41). 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). Figure 5.10 shows an example application of HGetDRL.

Note that HGetDRL automatically checks the result state of the underlying procedure with the macro HCkP (see section 5.6.1 on page 83). Thus, **you must not use** HCkP **on** HGetDRL yourself.

5.2.2 HGetFDRL

In contrast to HGetDRL, only a pointer to the region data is returned by HGetFDRL (see figure 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 figure 5.11.

Note that HGetFDRL automatically checks the result state of the underlying procedure with the macro HCkP (see section 5.6.1 on page 83). Thus, **you must not use** HCkP **on** HGetFDRL yourself.

Names

```
HGetDRL, HGetFDRL, HGetURL, HGetDImage, HGetObjNum
```

Synopsis

```
#include "Halcon.h"

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

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

HGetURL(   Hproc_handle     proc_handle,
           int              par_num,
           Hrlregion        *region)

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

HGetObjNum(Hproc_handle     proc_handle,
           int              par_num,
           INT4_8           *num)
```

Figure 5.9: Additional routines for accessing input image objects.

```
Hrlregion  *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`.

5.2.3 HGetURL

`HGetURL` (see figure 5.9) reads all regions passed in the input iconic parameter with the index `par_num`, computes the union of all these regions, and returns the resulting region in `region`. The `Hrlregion` data referenced by

```
Hkey       obj_key;
Hrlregion  *region;
...
HGetFDRL(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;
```

Figure 5.11: Application of `HGetFDRL`.

region must have been allocated before with a suitable size (e.g., by using HAllocRLTmp, see section 3.2.1 on page 41, which can be increased e.g., by using HIncrRL, see section 5.6.5 on page 84). 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). Figure 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.

Note that HGetURL automatically checks the result state of the underlying procedure with the macro HCkP (see section 5.6.1 on page 83). Thus, **you must not use** HCkP **on** HGetURL yourself.
If you do not want to have the result state checked with HCkP, use HPGetURL instead. HPGetURL and HGetURL only differ regarding HCkP.

```
Hrlregion  *region;
...
HCkP(HAllocRLTmp(proc_handle,&region));
HGetURL(proc_handle,par_num,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 5.12: Application of HGetURL.

5.2.4 HGetDImage

HGetDImage (see figure 5.9 on page 63) combines HGetComp (section 5.1.2 on page 59) and HGetImage (section 5.1.4 on page 61). It reads the data of the gray value component channel of the image object specified by the database key obj_key and returns it in the Himage structure (image).

Note that HGetDImage automatically checks the result state of the underlying procedure with the macro HCkP (see section 5.6.1 on page 83). Thus, **you must not use** HCkP **on** HGetDImage yourself.

```
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.l,col.par.l,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.d = image.pixel.f[coord];
    gray.type  = DOUBLE_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 iconic object obj_key.

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. Figure 5.13 shows an application of HGetDImage.

5.2.5 HGetObjNum

HGetObjNum (see figure 5.9 on page 63) returns the number of iconic objects which are stored in the input iconic parameter denoted by its index par_num (from 1 to n). It is an alternative to the usage of HReadGVA (compare page 59). By using HGetObjNum, the example of figure 5.3 on page 59 looks as follows (see figure 5.14).

Note that HGetObjNum automatically checks the result state of the underlying procedure with the macro HCkP (see section 5.6.1 on page 83). Thus, **you must not use** HCkP **on** HGetObjNum yourself.

```
int     inp_pars;
INT4_8  num_objs;
Hkey    key;
int     p,o;
...
/* get number of input iconic 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 iconic objects per input iconic parameter: */
   HGetObjNum(proc_handle,p,&num_objs);
   for (o=1; o<=num_objs; o++) {
      HGetObj(proc_handle,p,o,&key); /* get key of iconic object   */
      /* further processing ... */
   }
}
```

Figure 5.14: Application of HGetObjNum: Access the number of iconic objects of an input iconic parameter.

5.3 Loop Macros for Accessing Single Input Objects

This section describes two macros that ease the access of *all* iconic objects and image components within an input iconic parameter, see figure 5.15. Please see also the additional loop macros in section 6.1 on page 87 that further facilitate the programming of typical supply procedure.

Names

```
HAllObj, HAllComp
```

Synopsis

```
#include "Halcon.h"

HAllObj( Hproc_handle  proc_handle,
         int           par_num,
         Hkey          &obj_key,
         INT4_8        &index)

HAllComp(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 HAllObj

HAllObj (see figure 5.15) performs a loop over all iconic objects of a specified input iconic parameter and returns their database key.

```
Hkey       obj_key;
Hrlregion  *region;
Himage     image;
INT4_8     index;
...
HAllObj(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 HAllObj: Access the domain (area of definition) and the first gray value channel of all iconic objects of the first input iconic parameter.

Within the loop, all iconic objects contained in the input iconic parameter with the index[2] par_num are accessed one by one. index is set to the index of the current iconic object to be processed and obj_key is set to the corresponding database key. Thus, using HAllObj is equivalent to programming an explicit loop over all iconic objects contained in a parameter and determining the database keys via HGetObj, see also figure 5.3 on page 59. Figure 5.16 shows an application of HAllObj.

5.3.2 HAllComp

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

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

```
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.width,image_in.height,
                  &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 HAllComp: Rotate all channels of all image objects of the first input iconic parameter.

Figure 5.17 shows an example of how to use HAllComp. Within the HAllObj loop for each input image object, an output image object is created and added to the iconic object list of the first output iconic parameter (HCrObj, see

[2] So in contrast to the routines described in section 6.1 on page 87, HAllObj allows to specify the desired parameter by its index.

section 5.4.1). After that, a loop over all gray value channels of the current image object is performed (HAllComp). Within this loop, for every gray value channel a new image matrix is created (HCrImage, see section 5.4.7 on page 70) and filled with the rotated input matrix (IPBRot90). Finally, this matrix is installed as a new gray value channel of the output image object (HDefObj, see section 5.4.5 on page 69). The corresponding region transformation (rotating the region and defining the rotated region as new domain, i.e., area of definition, of the output image object via 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 database of iconic objects and for writing output iconic parameters of a HALCON operator, see figure 5.18. Note that regions, gray value channels, and XLDs are all stored as individual iconic objects within the database. Thus, for example different image objects can *share* the same gray value channels (image matrices) or domains (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 the HALCON database of iconic objects,
- combine regions and channels to image objects,
- and add the appropriate iconic objects to the corresponding output iconic parameters.

Names

```
HCrObj, HCopyObj, HPutDRL, HPutImage
```

Synopsis

```
#include "Halcon.h"

HCrObj(          Hproc_handle  proc_handle,
                 int           par_num,
                 Hkey          *obj_key)

HCopyObj(        Hproc_handle  proc_handle,
                 Hkey          obj_key,
                 int           par_num,
                 Hkey          *obj_key)

HPutDRL(         Hproc_handle  proc_handle,
                 Hkey          obj_key,
                 Hrlregion     *region,
                 Hkey          *rl_key)

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

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

5.4.1 HCrObj

HCrObj (see figure 5.18) creates a new image object in the HALCON database of iconic objects and adds it to the list of iconic objects in the output iconic parameter with the index par_num. Note that more than one image object can be returned in the same output parameter by iterating calls of HCrObj using the same par_num: Each call appends the database key of the new iconic object at the end of the iconic object list for the parameter.

The new image object contains the following default image components:

Names

```
HDefObj, HPutDImage, HCrImage, HCrXLD
```

Synopsis

```
#include "Halcon.h"

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,
                  bool          copy,
                  Hkey          *image_key)

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

HCrXLD(           Hproc_handle  proc_handle,
                  int           par_num,
                  XLD           *xld,
                  int           xld_type,
                  Hkey          *used_xlds,
                  int           num_used_xlds,
                  DBFreeProc    free_proc,
                  Hkey          *obj_key)
```

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

- The region component specifying the area of definition, i.e., the domain of the image, is set to the empty region. This component can be changed using `HPutDRL` (see section 5.4.3 on page 69) or `HPutRect` (see section 6.2.2 on page 94).
- All channels are marked as undefined, i.e., the new image object contains no gray value components. Channels can be added to the iconic object using `HPutImage` (see section 5.4.4) and `HDefObj` (see section 5.4.5) or `HPutDImage` (see section 5.4.6 on page 70).

Figure 5.17 on page 66 and figure 5.21 on page 70 show examples how to use `HCrObj`.

Note that `HCrObj` automatically checks the result state of the underlying procedure with the macro `HCkP`, (see section 5.6.1 on page 83). Thus, **you must not use** `HCkP` **on** `HCrObj` yourself.

5.4.2 `HCopyObj`

`HCopyObj` (see figure 5.18 on page 67) creates a new *iconic* object (image, region, or XLD) that contains the same components as an already existing iconic object specified by its database key `obj_key`. The new iconic object is stored in the HALCON database of iconic objects and appended to the list of iconic objects in the output iconic parameter with the index `par_num`. The database key of the new image object is returned in `obj_key`.

Note that `HCopyObj` automatically checks the result state of the underlying procedure with the macro `HCkP` (see section 5.6.1 on page 83). Thus, **you must not use** `HCkP` **on** `HCopyObj` yourself.

```
Hkey       obj_key_in,obj_key_out,reg_key_out;
Hrlregion  *region_in;
Hrlregion  *region_out;
INT4_8     num_objs;
...
par_in  = 1;  /* index of input parameter  */
par_out = 1;  /* index of output parameter */
HReadGVA(proc_handle,HGnum_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);
}
```

Figure 5.20: Create an image object with `HCopyObj` and insert a region component.

Similar to `HCrObj`, the components of a new *image* object can be reassigned using `HDefObj` etc. `HCopyObj` is especially useful, when output iconic objects hardly differ from the corresponding input objects.

5.4.3 HPutDRL

`HPutDRL` (see figure 5.18 on page 67) stores region data encoded in the `Hrlregion` structure `region` in the HALCON database of iconic objects and returns the database key of the new iconic object. Note that `HPutDRL` actually *copies* the data itself, not only a pointer to the data. In addition to that, the region object is assigned to the image object specified by the database key `obj_key` as the region component. The latter might have been created before by using `HCrObj` or `HCopyObj`, see figures 5.20 and 5.21.

Note that `HPutDRL` automatically checks the result state of the underlying procedure with the macro `HCkP` (see section 5.6.1 on page 83). Thus, **you must not use** `HCkP` **on** `HPutDRL` yourself.

5.4.4 HPutImage

`HPutImage` (see figure 5.19 on page 68) stores the gray value channel (image matrix) `image` in the database of iconic objects and returns the corresponding database key in the parameter `obj_key`. This key can be used to insert the gray value channel (now encapsulated in a database object) as a component of an arbitrary number of image objects (see `HDefObj` in section 5.4.5). The parameter `copy` (`false` or `true`) specifies, whether the pixel data is *copied* to the HALCON database[3] or only the address of the data is passed to the database object. Figure 5.21 shows a typical example for how to use `HPutImage`.

Note that `HPutImage` automatically checks the result state of the underlying procedure with the macro `HCkP` (see section 5.6.1 on page 83). Thus, **you must not use** `HCkP` **on** `HPutImage` yourself.

5.4.5 HDefObj

`HDefObj` (see figure 5.19 on page 68) reassigns the component `comp` ($0 \ldots n$) of an image object given by its database key `comp_key`. A typical application of this routine is shown in figure 5.21: Image data is stored in a database object using `HPutImage`. Furthermore, a new image object is created by `HCrObj`. This new object does not contain any gray value channels, see section 5.4.1 on page 67. Using `HDefObj`, the previously stored image data is assigned as default gray value component 1 (`IMAGE_INDEX`) of the new image object.

Note that `HDefObj` automatically checks the result state of the underlying procedure with the macro `HCkP` (see section 5.6.1 on page 83). Thus, **you must not use** `HCkP` **on** `HDefObj` yourself.

[3] This is necessary if the original data is stored in temporary memory or if the data will be modified later on.

```
Himage    image;
Hrlregion *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);
```

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

5.4.6 HPutDImage

HPutDImage (see figure 5.19 on page 68) combines HPutImage and HDefObj. It stores the gray value channel (image matrix) image in the HALCON database of iconic objects and returns the corresponding database key in image_key. The parameter copy (false or true) is used as in HPutImage and specifies, whether the pixel data is *copied*[4] or only a pointer to the data is passed to the database. Moreover, the gray value channel is inserted into the image object obj_key as component comp ($1 \ldots n$). Figure 5.22 shows a typical application of this routine.

Note that HPutDImage automatically checks the result state of the underlying procedure with the macro HCkP (see section 5.6.1 on page 83). Thus, **you must not use** HCkP **on** HPutDImage yourself.

```
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 figure 5.19 on page 68) is used for creating output images especially in the context of filter[5] 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 domain (i.e., in their region component), but *not* in their gray values. Such objects have different database 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 on page 43). The parameters type (BYTE_IMAGE, LONG_IMAGE, FLOAT_IMAGE, etc.; see figure 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 on page 43 for details.

Furthermore, HCrImage creates a new database object encapsulating this image data and returns the corresponding database key in image_key_out (similar to HPutImage in section 5.4.4 on page 69).

Finally, HCrImage establishes a link between the original input image data (referenced by the database key image_key_in) and this new iconic object to avoid the allocation of multiple output matrices per input matrix. HCrImage checks, whether there already exists an output matrix with the specified index index (see below)

[4] This is necessary, if the pixel data was stored in temporary memory or if it will be modified later on.
[5] This is due to the fact, that filtering only modifies the gray values but not the regions of image objects.

```
Hkey       obj_key,image_key_in,image_key_out,key_out;
Hrlregion  *region;
Himage     image_in,image_out;

...
HGetSPar(proc_handle,1,LONG_PAR,&Rows,1); /* height: filter mask */
HGetSPar(proc_handle,2,LONG_PAR,&Cols,1); /* width:  filter mask */

HAllObj(proc_handle,1,obj_key,i) {         /* all input images   */
  HGetFDRL(proc_handle,obj_key,&region);   /* get region data    */
  HGetComp(proc_handle,obj_key,IMAGE1,&image_key_in);
  if (image_key_in == UNDEFINED)
    return(H_ERR_UNDI);                    /* 'undefined image'  */
  HGetImage(proc_handle,image_key_in,&image_in); /*get image data*/
  HCkP(HCrImage(proc_handle,image_key_in,1,image_in.kind,
                image_in.width,image_in.height,
                &image_key_out,&image_out));

  HCkP(IPFilterOperation(&image_in,region,Rows,Cols,&image_out));

  HCopyObj(proc_handle,obj_key,1,&key_out); /* create result obj.*/
  HDefObj(proc_handle,key_out,image_key_out,IMAGE1);
                    /* insert the result image in output object */
}
```

Figure 5.23: A typical example for using `HCrImage`: Providing an empty image matrix for a filter operation.

for a given input image `image_key_in`. If there is any, only a reference to it is returned in `image_out` and `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 iconic objects (`HAllObj`).

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, `HCrImage` must be called several times for every input component – one time for each output matrix to be created. Assign increasing indices (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.

Figure 5.23 shows an exemplary application of `HCrImage`.

Note that `HCrImage` is no macro, i.e., it does not check the result state of the underlying procedure using `HCkP` (see section 5.6.1 on page 83). Thus, **you must call `HCkP` on `HCrImage`** yourself as shown in the example.

5.4.8 HCrXLD

`HCrXLD` (see figure 5.19 on page 68) is used for creating XLD objects in the HALCON database of iconic objects. The parameter `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, `HCrXLD` expects a pointer to `Hcont` or a pointer to `Hpoly` in the parameter `xld`. In both cases, *not* the underlying XLD data but only the *pointers* to the data are copied to the database. Thus, **you must not deallocate or overwrite the XLD structures** after calling `HCrXLD`.

Note that `HCrXLD` automatically checks the result state of the underlying procedure with the macro `HCkP` (see section 5.6.1 on page 83). Thus, **you must not use `HCkP` on `HCrXLD`** yourself.

`HCrXLD` expects an appropriate deallocation routine for the XLD data in the parameter `free_proc`. For XLD contours, please use

```
(DBFreeProc) HXLDFreeContour
```

For XLD polygons allocated based on `HAlloc` use

```
Hcpar      rows[99], cols[99];
INT4_8     i,num_points;
Hkey       key_out;
Hcont      *cont;

...
HGetCPar(proc_handle,1,DOUBLE_PAR,rows,1,99,&num_points);
HGetCPar(proc_handle,2,DOUBLE_PAR,cols,1,99,&num_points);

/* create a contour (in general, this should be done */
/* calling an appropriate action procedure)          */
HCkP(HAllocXLDCont(proc_handle,&cont,num_points));
for (i=0; i < num_points; i++ {
  cont->row[i] = rows[i].par.d;
  cont->col[i] = cols[i].par.d;
}
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.

```
(DBFreeProc) HXLDFreePolygon
```

Finally, HCrXLD not only creates a new database object, but also appends this object to the output iconic parameter with index par_num. Figures 5.24 shows a typical application of HCrXLD.

5.5 Reading and Writing Control Parameters

This section describes routines for reading and writing in-/output control parameters, see figure 5.25, figure 5.29, figure 5.30, and figure 5.35. Interchanging control data with the host language is done via arrays of the data type INT4_8 for LONG_PAR, double for DOUBLE_PAR, Hphandle for HANDLE_PAR, char* for STRING_PAR, or Hcpar for MIXED_PAR array types (see section 4.4 on page 55).

Control parameters can be passed through the HALCON interface in two different ways: by reference or by value. The former is more efficient but does not allow to modify the values. It is supported by the routines HGetPElemL, HGetPElemD, HGetPElemS, and HGetPPar for reading, depending on the expected type of the input array. HGetPElem provides a general, type independent read access. In case of writing output control parameters, HPutPPar and HPutPElem are provided. In contrast, the routines HGetElemL, HGetElemD, HGetElemS, HGetCElemH1, HGetCElemH, HGetCElemHN, and HGetCPar return a copy of the input array, whereas HPutElem, HPutElemH, and HPutCPar copy an array to an output control parameter.

5.5.1 HGetPElemL, HGetPElemD, HGetPElemS

HGetPElemL, HGetPElemD, and HGetPElemS (see figure 5.25) return a reference in elem to the value array of the input control parameter at the position par $(1 \ldots n)$. According to the expected type of the array, HGetPElemL is used to get LONG_PAR arrays, HGetPElemD for DOUBLE_PAR, and HGetPElemS for STRING_PAR arrays. The behavior of the routines in case of passing array types different to the expected can be specified by the convert parameter. Following values are supported:

CONV_NONE supports no conversion and accepts only the requested element type. Otherwise, the error 'wrong type of input parameter x' (H_ERR_WIPTx) will be returned.

CONV_CAST provides conversion of LONG_PAR and DOUBLE_PAR values to the requested type by a cast. In case of conversion, the array is copied to an ad hoc allocated memory buffer. This buffer is freed automatically when the operator returns. LONG_PAR and DOUBLE_PAR value types can not be casted to type STRING_PAR and vice versa. In this case, error 'wrong type of input parameter x' (H_ERR_WIPTx) will be returned.

Names

```
HGetPElemL, HGetPElemD, HGetPElemS, HGetPPar, HGetPElem
```

Conversion Flags

```
CONV_NONE   /* no conversion; do only consider original element types */
CONV_CAST   /* <INT4_8>=(INT4_8)<double>; <double>=(double)<INT4_8>   */
CONV_IDNT   /* convert double to INT4_8, if double has no fraction    */
CONV_RND    /* <INT4_8>=(INT4_8)(<double>+0.5)                        */
```

Synopsis

```
#include "Halcon.h"

HGetPElemL(     Hproc_handle              proc_handle,
                int                       par,
                int                       convert,
                INT4_8 const*restrict     *elem,
                INT4_8                    *num)

HGetPElemD(     Hproc_handle              proc_handle,
                int                       par,
                int                       convert,
                double const*restrict     *elem,
                INT4_8                    *num)

HGetPElemS(     Hproc_handle              proc_handle,
                int                       par,
                int                       convert,
                char const* const*restrict *elem,
                INT4_8                    *num)

HGetPPar(       Hproc_handle              proc_handle,
                int                       par,
                Hcpar* restrict           *val,
                INT4_8                    *num)

HGetPElem(      Hproc_handle              proc_handle,
                int                       par,
                void const*restrict       *elem,
                INT4_8                    *num,
                int                       *type)
```

Figure 5.25: Routines for reading control input parameters by reference.

`CONV_IDNT` converts `DOUBLE_PAR` values of a `DOUBLE_PAR` or `MIXED_PAR` array to type `LONG_PAR` when passed to `HGetPElemL`. This conversion is performed only if all `double` values of the input array have no fraction, i.e., the cast is reversible. When passed to `HGetPElemD`, values of type `LONG_PAR` are casted to type `DOUBLE_PAR`. Any other conversion is not supported and in this case the error `'wrong type of input parameter x'` (`H_ERR_WIPTx`) is returned. In case of conversion, the array is copied to an ad hoc allocated memory buffer. This buffer is freed automatically when the operator returns.

`CONV_RND` provides conversion by rounding when `DOUBLE_PAR` values have to be converted to `LONG_PAR` type (`HGetPElemD`). `LONG_PAR` values are casted to `DOUBLE_PAR` if passed to `HGetPElemD`. Otherwise, the error `'wrong type of input parameter x'` (`H_ERR_WIPTx`) will be returned. In case of conversion, the array is copied to an ad hoc allocated memory buffer. This buffer is freed automatically when the operator returns.

Any conversion of input array types will be paid for by performance and memory consumption. Refer to `HGetPElem` (see section 5.5.3) if efficient support of different array types is important. The current number of parameter values is returned in `num`.

Note that HGetPElemL, HGetPElemD, and HGetPElemS automatically check the result state of the underlying procedure similarly to the macro HCkP (see section 5.6.1 on page 83). Thus, **you must not use** HCkP **on** HGetPElemL**,** HGetPElemD**, or** HGetPElemS yourself.

Figure 5.26 shows the usage of HGetPElemL andHGetPElemS together with HGetPPar (compare section 5.5.2).

5.5.2 HGetPPar

HGetPPar (see figure 5.25 on page 73) returns the reference to a MIXED_PAR array of Hcpar structures, which contain the parameter values of the input control parameter at the position par $(1 \ldots n)$. Contrary to HGetCPar (see section 5.5.7 on page 76), HGetPPar is much more efficient in terms of memory overhead, because it just returns pointers to the values instead of copying them. 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, DOUBLE_PAR, STRING_PAR) is also stored within the Hcpar structure. Therefore, the same control parameter may contain values of different control data types. In case the input array is of different type to MIXED_PAR, the array is copied and converted to an ad hoc allocated memory buffer of Hcpar structures. This buffer is freed automatically when the operator returns. The current number of parameter values is returned in num.

Note that HGetPPar automatically checks the result state of the underlying procedure similarly to the macro HCkP (see section 5.6.1 on page 83). Thus, **you must not use** HCkP **on** HGetPPar yourself.

Figure 5.26 shows the usage of HGetPPar together with HGetPElemL and HGetPElemS (compare section 5.5.1 on page 72).

```
INT4_8 const     *ctrl1;  /* LONG_PAR array                        */
char const*const *ctrl2;  /* STRING_PAR array                      */
Hcpar            *ctrl3;  /* MIXED_PAR array of Hcpar structures   */
INT4_8           num;     /* length of an input array              */

/* Read all values of parameter number 1. Say, we expect a   */
/* LONG_PAR array but also tolerate double values.           */
HGetPElemL(proc_handle,1,CONV_CAST,&ctrl1,&num);
/* now ctrl1[0] == 5; ctrl1[1] == 6;                         */
/* Read a string in parameter number 2. Say, we expect a     */
/* STRING_PAR array.                                         */
HGetPElemS(proc_handle,2,CONV_NONE,&ctrl2,&num);
/* now ctrl2[0] == "Text"                                    */
/* Read a tuple of float values in parameter number 3.       */
/* At this parameter we handle any value type                */
HGetPPar(proc_handle,3,&ctrl_val3,&num);
/* now ctrl3[0].type == DOUBLE_PAR, ctrl3[0].par.d == 1.7    */
/*     ctrl3[1].type == LONG_PAR,   ctrl3[1].par.l == 4      */
/*     ctrl3[2].type == STRING_PAR, ctrl3[2].par.s == "ctrl" */
  ...
```

Figure 5.26: Read the control parameter values of the operator call demo([5,6.8],'Text',[1.7,4,'ctrl']) using HGetPElemL, HGetPElemS, and HGetPPar.

5.5.3 HGetPElem

A general access on input control parameter without any implicite conversion is provided by the routine HGetPElem (see figure 5.25). HGetPElem returns in elem a reference to the input array of arbitrary type of the input control parameter with the index par $(1 \ldots n)$. The length of the array is returned in num. type holds the array type of the returned array and can take the values LONG_PAR, DOUBLE_PAR, STRING_PAR, and MIXED_PAR, coding arrays of type INT4_8, double, char*, and Hcpar, respectively.

Figure 5.27 shows the usage of HGetPElem.

```
const void *restrict ctrl1;
INT4_8     n1;
int        t1;

/* Read all values of parameter number 1   */
HGetPElem(proc_handle, 1, &e1, &n1, &t1);
switch( t1)
{
case LONG_PAR:
  {
    INT4_8 const *l = (INT4_8 const *)e1;
    ...
    break;
  }
case DOUBLE_PAR:
  {
    double const *d = (double const *)e1;
    ...
    break;
  }
case STRING_PAR:
  {
    char const *const *s = (char const *const *)e1;
    ...
    break;
  }
case MIXED_PAR:
  {
    Hcpar const *cpar = (Hcpar const *)e1;
    ...
    break;
  }
default:
  return H_ERR_WIPT1;
}

...
```

Figure 5.27: Read the control parameter values of the first control parameter of an operator using `HGetPElem`.

5.5.4 HGetCElemH1, HGetCElemH, HGetCElemHN

`HGetCElemH1` retrieves a single handle from an input parameter. `HGetCElemH` retrieves all handles from an input parameter. `HGetCElemHN` retrieves a given number N of handles from an input parameter. The returned array contains pointers to the data stored in the handles. It is freed automatically when the operator finishes.

5.5.5 HGetElemL, HGetElemD, HGetElemS

`HGetElemL`, `HGetElemD`, and `HGetElemS` (see figure 5.29) return a copy of the input control parameter array with the index `par` $(1 \ldots n)$. According to the expected type of the array, `HGetElemL` is used to get `LONG_PAR` arrays, `HGetElemD` for `DOUBLE_PAR`, and `HGetElemS` for `STRING_PAR` arrays. In contrast to `HGetPElemL`, `HGetPElemD`, and `HGetPElemS`, the input arrays will be allocated and the values copied in any case. Therefore, the resulting array can be modified with the drawback of memory overhead and loss of performance. The kind of memory allocation (compare section 3.2 on page 41) can be specified by the parameter `memtype`. It supports the values `HMEMglobal`, `HMEMlocal`, and `HMEMtemp`, specifying the allocation function `HAlloc`, `HAllocLocal`, or `HAllocTmp` that is used internally. The specified function is used by the routines to allocate the array, respectively. But note that this array must be freed explicitly with the according function `HFree`, `HFreeLocal` or `HFreeTmp` after usage. The behavior of the routines in case of passing array types different to the expected can be specified by the `convert` parameter according to the description in section 5.5.1 on page 72.

Names

```
HGetCElemH1, HGetCElemH, HGetCElemHN
```

Synopsis

```
#include "Halcon.h"

HGetCElemH1(    Hproc_handle            proc_handle,
                int                     par,
                H_HANDLE_TYPE const     *htype,
                Hphandle                elem)

HGetCElemH(     Hproc_handle            proc_handle,
                int                     par,
                H_HANDLE_TYPE const     *htype,
                Hphandle                elem,
                INT4_8                  *num)

HGetCElemHN(    Hproc_handle            proc_handle,
                int                     par,
                H_HANDLE_TYPE const     *htype,
                INT4_8                  num,
                Hphandle                elem)
```

Figure 5.28: Routines for reading handles by copy.

Note that HGetPPar automatically checks the result state of the underlying procedure similarly to the macro HCkP (see section 5.6.1 on page 83). Thus, **you must not use** HCkP **on** HGetPPar yourself.

5.5.6 HCopyElemL, HCopyElemD

HCopyElemL and HCopyElemD (see figure 5.30) read the parameter values of the input control parameter with the index par $(1 \ldots n)$ and write them to the array elem that must have been allocated before with a suitable size (see HGetCParNum for information on getting the actual array length). According to the routine's suffix, HCopyElemL expects LONG_PAR, HCopyElemD DOUBLE_PAR arrays. The behavior of the routines in case of passing array types different to the expected can be specified by the convert parameter according to the description in section 5.5.1 on page 72. The number of elements that have been allocated for elem must be passed to num. In return, num will hold the number of actually copied array indices.

Note that HCopyElemL and HCopyElemD automatically check the result state of the underlying procedure similarly to the macro HCkP (see section 5.6.1 on page 83). Thus, **you must not use** HCkP **on** HCopyElemL **and** HCopyElemD yourself.

5.5.7 HGetCPar

HGetCPar (see figure 5.30) reads the parameter values of the input control parameter with the index par $(1 \ldots n)$ and writes them to the array val of Hcpar structures that must have been allocated before with a suitable size.

Note that HGetCPar automatically checks the result state of the underlying procedure similarly to the macro HCkP (see section 5.6.1 on page 83). Thus, **you must not use** HCkP **on** HGetCPar yourself.

For every input value, its type (LONG_PAR, DOUBLE_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 | DOUBLE_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.

Names

```
HGetElemL, HGetElemD, HGetElemS
```

Conversion Flags

```
CONV_NONE   /* no conversion; do only consider original element types */
CONV_CAST   /* <INT4_8>=(INT4_8)<double>; <double>=(double)<INT4_8>   */
CONV_IDNT   /* convert double to INT4_8, if double has no fraction    */
CONV_RND    /* <INT4_8>=(INT4_8)(<double>+0.5)                        */
```

Synopsis

```
#include "Halcon.h"

HGetElemL(      Hproc_handle             proc_handle,
                int                      par,
                int                      convert,
                int                      memtype,
                INT4_8 *restrict         *elem,
                INT4_8                   *num)

HGetElemD(      Hproc_handle             proc_handle,
                int                      par,
                int                      convert,
                int                      memtype,
                double *restrict         *elem,
                INT4_8                   *num)

HGetElemS(      Hproc_handle             proc_handle,
                int                      par,
                int                      convert,
                int                      memtype,
                char *restrict*restrict  *elem,
                INT4_8                   *num)
```

Figure 5.29: Routines for reading control input parameters by copy.

If the specified number or types of values are violated, `HGetCPar` exits the supply procedure with an appropriate error message. Figure 5.32 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 5.5.10.

5.5.8 HGetSPar

`HGetSPar` is a simplified version of `HGetCPar`. In contrast to the latter, a *fixed* number of parameter values is read, see figure 5.33. Therefore, the routine does not return the actual number[6] of values.

Note that `HGetSPar` automatically checks the result state of the underlying procedure similarly to the macro `HCkP` (see section 5.6.1 on page 83). Thus, **you must not use** `HCkP` **on** `HGetSPar` yourself.

5.5.9 HGetCParNum

`HGetCParNum` returns the length of input control arrays of the input control parameter with the index `par` $(1 \ldots n)$. This makes it possible to allocate exactly as much memory for parameter values as needed for reading them, for example, via `HCopyElemL`, `HCopyElemD`, or `HGetCPar`. As an alternative, you can use the reference passing routines as `HGetPElem` or `HGetPPar`, or routines as `HGetElemL` that allocate memory implicitly. Figure 5.34 shows an application of `HGetCParNum`.

[6] This number is fixed. If more or less values are passed to the HALCON operator, `HGetSPar` exits the supply procedure with an error.

Names

```
HCopyElemL, HCopyElemD, HGetCPar
```

Conversion Flags

```
CONV_NONE   /* no conversion; do only consider original element types */
CONV_CAST   /* <INT4_8>=(INT4_8)<double>; <double>=(double)<INT4_8>   */
CONV_IDNT   /* convert double to INT4_8, if double has no fraction    */
CONV_RND    /* <INT4_8>=(INT4_8)(<double>+0.5)                        */
```

Synopsis

```
#include "Halcon.h"

HCopyElemL(     Hproc_handle            proc_handle,
                int                     par,
                int                     convert,
                INT4_8 *restrict        elem,
                INT4_8                  *num);

HCopyElemD(     Hproc_handle            proc_handle,
                int                     par,
                int                     convert,
                double *restrict        elem,
                INT4_8                  *num);

HGetCPar(       Hproc_handle            proc_handle,
                int                     par,
                int                     type,
                Hcpar                   *val,
                INT4_8                  min,
                INT4_8                  max,
                INT4_8                  *num)
```

Figure 5.30: Routines for reading control input parameters by copy to a preallocated array.

```
INT4_8      l[10]; /* maximum array length of 10 */
INT4_8      num = 10;

/* read a short/int/long value in parameter 1 */
HCopyElemL(proc_handle,1,CONV_RND,l,&num);
/* now l == [2,4], num = 2      */
...
```

Figure 5.31: Read the parameter values `[1.7,4]` of control parameter `1` using `HCopyElemL`.

Note that `HGetCParNum` automatically checks the result state of the underlying procedure similarly to the macro `HCkP` (see section 5.6.1 on page 83). Thus, **you must not use** `HCkP` **on** `HGetCParNum` yourself.

5.5.10 `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 `HAllocStringMem` specifying the expected number of characters for *all* input strings (parameter `size`) at the begin of a supply procedure, see figure 5.32 and figure 5.33. It is not necessary to free this memory explicitly — this is done automatically at the end of the supply procedure.

Note that `HAllocStringMem` automatically checks the result state of the underlying procedure using `HCkP` (see section 5.6.1 on page 83). Thus, **you must not use** `HCkP` **on** `HAllocStringMem` yourself.

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

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

Figure 5.32: Read the control parameter values of the operator call `demo(5,'Text',[0.5,1.7,4.4])` using `HGetCPar`.

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

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

Figure 5.33: Read the control parameter values of the operator call `demo(5,'Text',[0.5,1.7,4.4])` using `HGetSPar`.

5.5.11 HPutPElem

`HPutPElem` (see figure 5.35 on page 81) writes a native array to the output control parameter with the index `par`. `val` holds the reference to the array of the according type, coded in `type` (`LONG_PAR` for arrays of type `INT4_8`, `DOUBLE_PAR`, for `double`, and `STRING_PAR` for `char*` arrays). The length of the array is passed in `num`. Instead of copying the array like `HPutElem` does, `HPutPElem` directly stores the pointer to the output control parameter and, thus, causes less overhead. Therefore, this array has to be allocated "permanently", i.e. it must be allocated by using `HAlloc` (see section 3.2.2 on page 43). Furthermore, the array must not be freed after passing it to `HPutPElem`. Figure 5.36 shows how to write output control data with `HPutPElem`.

Note that `HPutPElem` automatically checks the result state of the underlying procedure similarly to the macro `HCkP` (see section 5.6.1 on page 83). Thus, **you must not use** `HCkP` **on** `HPutPElem` yourself.

5.5.12 HPutPPar

`HPutPPar` (see figure 5.35 on page 81) is an alternative to `HPutCPar` and writes control data to the output control parameter with the index `par`. 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

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

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

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

Figure 5.34: 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])`.

using `HAlloc` (see section 3.2.2 on page 43). Furthermore, the array must not be freed after passing it to `HPutPPar`. Figure 5.37 shows how to write output control data with `HPutPPar`.

Note that `HPutPPar` automatically checks the result state of the underlying procedure similarly to the macro `HCkP` (see section 5.6.1 on page 83). Thus, **you must not use `HCkP` on `HPutPPar`** yourself.

As already mentioned, every data element may use a different type, because the types of the parameter values (`LONG_PAR`, `DOUBLE_PAR`, `STRING_PAR`) are stored with each `Hcpar` structure.

5.5.13 HPutElem

`HPutElem` (see figure 5.35) writes a native array to the output control parameter with the index `par` of a HALCON operator. The type of the array passed in `val` is coded by `type`, with `LONG_PAR` for arrays of type `INT4_8`, `DOUBLE_PAR`, for `double`, `STRING_PAR` for `char*` arrays. In contrast to `HPutPElem`, the values are *copied* by this routine.

5.5.14 HPutElemH

`HPutElemH` (see figure 5.35) writes `num` handle references to the output control parameter with the index `par` of a HALCON operator. `elem` points to an array that holds `num` elements. `HHandleInfo` contains information about the type of the corresponding handles. Most notably,`HHandleInfo` contains pointers to the functions that operate on the handles (clear, serialize, deserialize and signal).

Note that the output control variable gains ownership of all pointers that are in `elem`. When aborting an operator with an error, those values must not be cleared inside the operator's code, but will be cleared automatically when cleaning the output control variable.

5.5.15 HPutCPar

`HPutCPar` (see figure 5.35) writes control data to the output control parameter with the index `par` of a HALCON operator. The `num` values in the `Hcpar` structure `val` are *copied* by this routine. Because the types of the parameter

Names

```
HPutPElem, HPutPPar
HPutElemH, HPutCPar
```

Synopsis

```
#include "Halcon.h"

/* --- writing output parameters by reference */
HPutPElem(      Hproc_handle   proc_handle,
                int            par,
                void           *val,
                INT4_8         num,
                int            type)

HPutPPar(       Hproc_handle   proc_handle,
                int            par,
                Hcpar          *val,
                INT4_8         num)

/* --- writing output parameters by copy */
HPutElem(       Hproc_handle   proc_handle,
                int            parnr,
                void const     *val,
                INT4_8         num,
                int            type)

HPutElemH(      Hproc_handle   proc_handle,
                int            par,
                void const     *elem,
                INT4_8         num,
                H_HANDLE_TYPE  *HHandleInfo)

HPutCPar(       Hproc_handle   proc_handle,
                int            par,
                Hcpar const    *val,
                INT4_8         num)
```

Figure 5.35: Routines for writing control output parameters.

```
double  *d;

HCkP(HAlloc(proc_handle,sizeof(*d)*2,&d));

d[0] = 1.2;
d[1] = 4.4;
HPutPElem(proc_handle,1,d,2,DOUBLE_PAR);
```

Figure 5.36: Write the control parameter values `[1.2,4.4]` using `HPutPElem`.

values (`LONG_PAR`, `DOUBLE_PAR`, `STRING_PAR`) are stored with each `Hcpar` structure, you can use a different type for every data element, see figure 5.38.

Note that `HPutCPar` automatically checks the result state of the underlying procedure similarly to the macro `HCkP` (see section 5.6.1 on page 83). Thus, **you must not use** `HCkP` **on** `HPutCPar` yourself.

```
Hcpar   *par;

HCkP(HAlloc(proc_handle,(size_t)(sizeof(Hcpar)*2),(void**)&par));

par[0].par.l = 6;
par[0].type = LONG_PAR;
par[1].par.d = 4.4;
par[1].type = DOUBLE_PAR;
HPutPPar(proc_handle,1,par,2);
```

Figure 5.37: Write the control parameter values `[6,4.4]` using `HPutPPar`.

```
Hcpar   par[2];

par[0].par.l = 6;
par[0].type = LONG_PAR;
par[1].par.d = 4.4;
par[1].type = DOUBLE_PAR;
HPutCPar(proc_handle,1,par,2);
```

Figure 5.38: 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 figure 5.39.

Names

HCkP, HCkNoObj, HRLDecomp, HNumOfChannels, HIncrRL

Synopsis

```
#include "Halcon.h"

HCkP(             Herror          (*proc))

HCkNoObj(         Hproc_handle     proc_handle)

Herror HRLDecomp( Hrlregion const  *reg_in,
                  int              r1,
                  int              c1,
                  int              r2,
                  int              c2,
                  Hrlregion        *reg_inner,
                  Hrlregion        *reg_outer)

HNumOfChannels(   Hproc_handle     proc_handle,
                  INT4_8           obj_index,
                  int              *num_channels)

HError HIncrRL(   Hproc_handle     proc_handle)
```

Figure 5.39: Auxiliary Extension Package Interface Macros and Procedures.

5.6.1 HCkP

The macro HCkP (see figure 5.39) checks the result state of a procedure call. Most of the HALCON procedures are of the type Herror. For internal HALCON procedures, the result code H_MSG_OK is returned if no error occurred, see also section 3.7 on page 47. 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 figure 5.40: Here, the call of the procedure HRLDecomp is encapsulated within the macro. If any error occurs, the procedure exits with an appropriate error code. Otherwise the processing is continued.

Note that most of the interface routines described in this chapter internally call HCkP, i.e. they exit the current procedure returning an appropriate error code in case of an error.

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.

```
Herror   err;

/* without HCkP: */
err = HRLDecomp(reg_in,r1,c1,r2,c2,
                reg_inner,reg_outer);
if (err != H_MSG_OK) return err;


/* with HCkP: */
HCkP(HRLDecomp(reg_in,r1,c1,r2,c2,
               reg_inner,reg_outer));
```

Figure 5.40: Error handling with HCkP.

5.6.2 HCkNoObj

The macro HCkNoObj (see figure 5.39 on page 82) is used to check whether all iconic input parameters of an operator contain at least one iconic 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

```
HCkNoObj(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 figure 5.39 on page 82) 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 hlib\HRLClip.h.

The region (respectively domain or area of definition) reg_in of the image to be processed is split into two parts: reg_inner and reg_outer. Note that both new regions must have been allocated before, see section 3.2.1 on page 41.

reg_inner is the original region reg_in minus the pixels around the image border. It contains all pixels within the rectangle specified by the upper left corner[7] (r1,c1) and the lower right corner (r2,c2). These coordinates

[7] r1 and r2 denote *row* coordinates, c1 and c2 *column* coordinates.

```
Herror IPBFilter(HBYTE      *in,            /* input image           */
                 Hrlregion  *region,        /* domain (area of def.) */
                 int        width,          /* image size: width     */
                 int        height,         /* image size: height    */
                 HBYTE      *out)           /* output image          */
{
  ...
  Hrlregion *inner,*outer;

  HCkP(HAllocRLNumTmp(proc_handle,&inner,region->num));
  HCkP(HAllocRLNumTmp(proc_handle,&outer,region->num*2));
  HCkP(HRLDecomp(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] = ...
  HCkP(HFreeRLTmp(proc_handle,outer));
  HCkP(HFreeRLTmp(proc_handle,inner));
}
```

Figure 5.41: Border treatment for a 3×3 filter.

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 `reg_in`.

`reg_out` is set to all remaining pixels within `reg_in`. For those pixels an appropriate (and time consuming) border treatment has to be performed. Figure 5.41 shows an application of `HRLDecomp`.

Note that `HRLDecomp` is no macro, i.e., it does not check the result state of the underlying procedure using `HCkP`. Thus, **you must call** `HCkP` **on** `HRLDecomp` yourself as shown in the example.

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 domain (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 `reg_inner` are *defined*, if they do not belong to `reg_in`. This means, a filter might use undefined gray values along the border of the domain 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

`HNumOfChannels` (see figure 5.39 on page 82) returns for the *first* input iconic parameter of a HALCON operator the number of channels of the image object with the index `obj_index`. Thus, this routine is just a shortcut for using `HPNumOfChannels` as defined in figure 5.5 on page 60 with `par_num = 1`.

Note that `HNumOfChannels` automatically checks the result state of the underlying procedure with the macro `HCkP`. Thus, **you must not use** `HCkP` **on** `HNumOfChannels` yourself.

5.6.5 HIncrRL

`HIncrRL` (see figure 5.39 on page 82) increases the value of the system parameter *'current_runlength_number'* by 50%, not exceeding `2*(width+1)*height`, where width and height are the dimensions of the largest image. These dimensions are accessible using `get_system` with *'width'* and *'height'* (see the Reference Manuals for details). Note, `DEF_RL_LENGTH` can also be increased using `set_system`.

As a consequence the procedures `HAllocRL` (page 44), `HAllocRLTmp` (page 42), and `HAllocRLLocal` (page 43) allocate regions of increased size. Figure 5.41 shows an application of `HIncrRL`.

Note that HIncrRL is no macro, i.e., it does not check the result state of the underlying procedure using HCkP. Thus, **you must call** HCkP **on** HIncrRL yourself as shown in the example.

```
Hrlregion   *region;
Herror      err;
HCkP(HAllocRLTmp(proc_handle, &region));
do
{
  /* Use HPGetURL instead of HGetURL, to perform explicit error handling. */
  err = HPGetURL(proc_handle, 1, region);
  if (err == H_ERR_WRRLN2)
  /* Number of chords too big, increase /*
  {
    HCkP(HFreeUpToTmp(proc_handle, region));
    HCkP(HIncrRL(proc_handle));
    HCkP(HAllocRLTmp(proc_handle, &region));
  }
  else if (err != H_MSG_OK)
  {
    HCkP(HFreeRLTmp(proc_handle, region));
    return err;
  }
} while (err == H_ERR_WRRLN2);
```

Figure 5.42: Increasing memory allocated for regions.

Chapter 6

Special Routines for Typical Supply Procedures

In the previous chapter, the basic routines for handling iconic objects, iconic 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 iconic object parameters. This is true for most filter and many segmentation operators. Loop macros provide a framework for this problem, see also section 5.3 on page 65.

The following table contains an overview of all loop macros, their areas of application, and the provided image data. Their syntax is defined in figure 6.1.

Macro	Application	Image Data Provided
`HAllReg`	region shape features, region transformation, binary morphology	region data (`Hrlregion`) of input image object
`HAllSegm`	gray value features, segmentation	region and image data (`Hrlregion` and `Himage`) of input image object
`HAllFilter`	filter, image transformations	region and image data (`Hrlregion` and `Himage`) of input image object, image data structure (`Himage`) for image output object
`HAllFilter2`	filter, segmentation, arithmetic with two input images	intersection of the regions of both input image objects (`Hrlregion`), the image data of both objects, and a image data structure (`Himage`) for the image output object

All macros described in this section have in common that they implement a loop over all image objects within the **first** input iconic 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 iconic parameters. The macros provide users with the region (domain) and the 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` $(1 \ldots n)$ within the input iconic object parameter. Thus, `obj_index` can be seen as a reference parameter of the macro and therefore is notated[1] with '`&obj_index`'.

The macros return all regions as pointers to the original region data within the HALCON database. 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.

[1] Note that this is a specific notation only - do not pass pointers to the macros!

Names

```
HAllReg, HAllSegm, HAllFilter, HAllFilter2
```

Synopsis

```
#include "Halcon.h"

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

HAllSegm(    Hproc_handle    proc_handle,
             Hrlregion       **region,
             Himage          *image,
             int             max_channels,
             INT4_8          obj_index)

HAllFilter(  Hproc_handle    proc_handle,
             Hrlregion       **region,
             Himage          *image_in,
             Himage          *image_out,
             int             max_channels,
             INT4_8          obj_index)

HAllFilter2( Hproc_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.

6.1.1 HAllReg

HAllReg (see figure 6.1) implements a loop over all iconic objects of the **first** input image parameter. For every object within this parameter, it returns the region of the iconic object (domain) in region[2]. All gray value channels are ignored.

HAllReg is a combination of HAllObj (see section 5.3.1 on page 66) and HGetFDRL (see section 5.2.2 on page 62). It is typically used within feature extraction operators: A list of input regions have to be examined concerning special features. Some HALCON operators of this kind are e.g., circularity, area_center, or contlength. Moreover, it is possible to return new region(s) by using the interface macro HNewRegion (see section 6.2.1 on page 94) as it is done e.g., by erosion_rectangle1 or shape_trans.

Figure 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 iconic 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 this:

```
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 5.5.15 on page 80).

[2]Remember: HALCON image objects consist of *one* region specifying the domain (area of definition) and an arbitrary number of gray value channels containing the pixel data

```
Herror CIPCenter(Hproc_handle  proc_handle)
{
  Hcpar      *Row,*Col;
  INT4_8     num;
  double     row,col;
  INT4_8     area;
  INT4_8     i;
  Hrlregion  *region;

  /* allocate memory corresponding to the number of regions. */
  HGetObjNum(proc_handle,1,&num);
  HCkP(HAlloc(proc_handle,sizeof(*Row)*num,&Row);
  HCkP(HAlloc(proc_handle,sizeof(*Col)*num,&Col);
  /* get the regions of all input iconic objects in the first param.  */
  HAllReg(proc_handle,&region,i)
  {
    HCkP(HRLArea(proc_handle,region,&area,&row,&col));
    Row[i-1].type  = DOUBLE_PAR;       /* par. type is float   */
    Row[i-1].par.d = row;              /* resulting row        */
    Col[i-1].type  = DOUBLE_PAR;       /* par. type is float   */
    Col[i-1].par.d = col;              /* resulting column     */
  }
  HPutPPar(proc_handle,1,Row,i);       /* return result */
  HPutPPar(proc_handle,2,Col,i);       /* return result */
  return(H_MSG_TRUE);
}
```

Figure 6.2: An application of HAllReg: Compute the center of gravity for all input regions in the *first* input iconic object parameter.

6.1.2 HAllSegm

HAllSegm (see figure 6.1) extends HAllReg: Not only the region but also all gray value channels of an object within the **first** input image object parameter are accessed within a loop. Thus, HAllSegm is a combination/modification of HAllObj (see section 5.3.1 on page 66), HGetFDRL (see section 5.2.2 on page 62), and HAllComp (see section 5.3.2 on page 66). It is especially useful for segmentation operators (transition of gray value channels to regions). Some typical HALCON operators that make use of HAllSegm are threshold, regiongrowing, auto_threshold, or label_to_region.

A HALCON image object (in short a HALCON *image*) consists of *one* region (domain) 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 on page 49). If the image has only one channel, the address of a variable of the type Himage is passed to HAllSegm. For multi-channel images an array of the type Himage[max_channels] must be passed. The maximal number of channels to be accessed by HAllSegm is specified by the parameter max_channels, see figure 6.3.

```
Himage        images[3];
HAllSegm(proc_handle,region,images,3,i) {
...
Himage        image;
HAllSegm(proc_handle,region,&image,1,i) {
...
```

Figure 6.3: The parameters image and max_channels of HAllSegm.

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 on page 84). An image must contain at least *one* channel. Otherwise, HAllSegm 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[3].

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

Herror CIPSegm(Hproc_handle proc_handle)
{
  INT4_8     i;
  Hrlregion  *region, *new_region;
  Himage     image;

  HCkP(HAllocRLTmp(proc_handle,&new_region)); /* allocate memory*/
  HAllSegm(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 `HAllSegm`: A segmentation operator.

Figure 6.4 shows a framework for a supply procedure for a segmentation operator using `HAllSegm`. 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

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

The macro `HAllSegm` 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` (domain) and `image` (first gray value channel). The results of the segmentation (`new_region`) are stored via the macro `HNewRegion` (see section 6.2.1 on page 94) as new objects of the HALCON database 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 on page 41) 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.

6.1.3 `HAllFilter`

`HAllFilter` (see figure 6.1 on page 88) 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 type (of the corresponding channels),
- the same image size and
- the same area of definition (region, domain).

Thus, `HAllFilter` is a combination/modification of `HAllObj` (see section 5.3.1 on page 66), `HGetFDRL` (see section 5.2.2 on page 62), `HAllComp` (see section 5.3.2 on page 66), `HCopyObj` (see section 5.4.2 on page 68), and `HPutDImage` (see section 5.4.6 on page 70). 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_amp`, `mean_image`, `scale_image_max`, or `texture_laws`.

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

```
Herror IPScaleNew(Hrlregion         *region,
                  Himage            *image_in,
                  Himage            *image_out,
{
  INT4_8 i,l,end;
  double h;

  for (i=0; i<region->num; i++) {
    end = CE(region->rl,i,image_in->width);
    for (l=CB(region->rl,i,image_in->width); l<=end; l++) {
      HDFImage(h,image_in,l);
      HImageFD(image_out,h*mult,l);
  } }
  return(H_MSG_OK);
}

Herror CIPScaleNew(Hproc_handle proc_handle)
{
  Hcpar        mult;
  Hrlregion    *region;
  Himage       image_in,image_out;
  INT4_8       i;

  HGetSPar(proc_handle,1,DOUBLE_PAR,&mult,1);
  HAllFilter(proc_handle,&region,&image_in,&image_out,1,i) {
    HCkP(IPScaleNew(region,&image_in,mult.par.d,&image_out));
  }
  return(H_MSG_TRUE);
}
```

Figure 6.5: A typical application of `HAllFilter`: A filter operator.

Figure 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 domain, i.e., 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 on page 89. 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 database) 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 domain of the input image and writes them into the provided image components `image_out`[4]. 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 5.5.8 on page 77) is used in the example.

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

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.

[4]Note that `image_out` contains a pointer to a pixel matrix with the same size and pixel type as `image_in`.

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 figure 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 on page 49.

   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`.

Names

```
HDFImage, HImageFD
```

Synopsis

```
#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.

6.1.4 `HAllFilter2`

The macro `HAllFilter2` (see figure 6.1 on page 88) 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_image`.

`HAllFilter2` extends `HAllFiter` 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 domains (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 on page 90.

Figure 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 figure 6.5 on page 91, the macros `HDFImage` and `HImageFD` are used to read and write pixel values. This example exhibits the additional advantage that the operator can even add the gray values of two images with *different* pixel types.

6.2 Object Generation

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

```
Herror IPAddNew(Hproc_handle  proc_handle,
                Hrlregion     *region,
                Himage        *image_in1,*image_in2,*image_out)
{
  INT4_8 i,l,end;
  double h1,h2;

  for (i=0; i<region->num; i++) {
    end = CE(region->rl,i,image_in->width);
    for (l=CB(region->rl,i,image_in->width); l<=end; l++) {
      HDFImage(h1,image_in1,l);
      HDFImage(h2,image_in2,l);
      HImageFD(image_out,h1+h2,l);
  } }
  return(H_MSG_OK);
}

Herror CIPAddNew(Hproc_handle  proc_handle)
{
  Hrlregion    *region;
  Himage       image_in1,image_in2,image_out;
  INT4_8       i;

  HCkP(HAllocRLTmp(proc_handle,&region));
  HAllFilter2(proc_handle,region,&image_in1,
              &image_in2,&image_out,1,i) {
    HCkP(IPAddNew(proc_handle,region,&image_in1,
                  &image_in2,&image_out));
  }
  HCkP(HFreeRLTmp(proc_handle,region));
  return(H_MSG_TRUE);
}
```

Figure 6.7: A typical application of `HAllFilter2`.

`HNewRegion` creates a new image object in the HALCON database encapsulating the specified region and adds this object to the first output object parameter. `HPutRect` inserts a rectangular region (as domain, i.e., 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.

Names

```
HNewRegion, HPutRect, HDupObj
```

Synopsis

```
#include "Halcon.h"

HNewRegion(Hproc_handle  proc_handle,
           Hrlregion     *region)

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

HDupObj(   Hproc_handle  proc_handle,
           INT4_8        obj_index)
```

Figure 6.8: Convenience routines for creating output objects.

6.2.1 HNewRegion

HNewRegion (see figure 6.8) is used to create a new image object in the HALCON database which is also added to the first output object parameter. It contains a copy of region. It is common to call an image object *region*, when only its region component is used.

Note that HNewRegion automatically checks the result state of the underlying procedure with the macro HCkP (see section 5.6.1 on page 83). Thus, **you must not use** HCkP **on** HNewRegion yourself.

```
Herror CIPConvex(Hproc_handle  proc_handle)
{
  Hrlregion  *region;
  Hrlregion  *region_new;

  HCkP(HAllocRLTmp(proc_handle,&region_new) /* allocate memory */
  HAllReg(proc_handle,&region,i) {          /* all regions     */
    HCkP(HRLConvex2(proc_handle,region,region_new));
    /* create an output region object (first output param.)    */
    HNewRegion(proc_handle,region_new);
  }
  HCkP(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 figure 6.4 on page 90) and region transformation procedures such as the procedure CIPConvex shown in figure 6.9. CIPConvex corresponds to a hypothetical operator trans_convex which might be defined as

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

In the example, all regions within Region are provided in region by HAllReg (see section 6.1.1 on page 88) one after the other. The internal HALCON procedure HRLConvex2 computes the convex hull for each region and writes it to region_new. Furthermore, a new region object in the HALCON database is created and added to the output object parameter RegionConvex.

Note that HNewRegion was also used in the example in figure 6.4 on page 90.

6.2.2 HPutRect

HPutRect (see figure 6.8 on page 93) inserts a rectangular region in an already existing output object with database key obj_key. It is mostly used in connection with operators that create *new* image objects like in the example in figure 6.10.

Note that HPutRect automatically checks the result state of the underlying procedure with the macro HCkP (see section 5.6.1 on page 83). Thus, **you must not use** HCkP **on** HPutRect yourself.

Note that HCrObj (see section 5.4.1 on page 67) creates an object with *empty* region. In many situations the default domain (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 figure 6.8 on page 93) is a combination/simplification of HGetObj (see section 5.1.1 on page 57) and HCopyObj (see section 5.4.2 on page 68). 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

```
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(IOBGrabImage(&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.

the loop macros (HAll*) in section 6.1 on page 87. HDupObj is typically used for operators that examine objects and select them by a given criterion. Some HALCON operators of this kind are select_shape or select_gray.

Note that HDupObj automatically checks the result state of the underlying procedure with the macro HCkP (see section 5.6.1 on page 83). Thus, **you must not use** HCkP **on** HDupObj yourself.

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

```
Herror CIPUserSelect(Hproc_handle  proc_handle)
{
  Hrlregion  *region;
  Hrlregion  *region_new;
  double     length;
  Hcpar      min;
  INT4_8     i;

  HGetSPar(proc_handle,1,DOUBLE_PAR,&min,1); /* get min. length */
  HAllReg(proc_handle,&region,i) {          /* all regions     */
    HCkP(HRLContLength(proc_handle,region,&length)); /* length */
    if (length >= min.par.d)                /* length o.k. ?   */
      HDupObj(proc_handle,i);               /* duplicate object*/
  }
  return(H_MSG_TRUE);
}
```

Figure 6.11: A typical application of HDupObj: Duplicate input regions that fulfill a specific criteria (length of contour).

Since HDupObj only increases the number of references to a database object instead of copying 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 whenever possible, HDupObj should be used.

Chapter 7

Creating a New HALCON Package

This chapter explains step by step how to create a new package. Note that the description often references the environment variable HALCONARCH, which is set during the installation. For more details about this variable please refer to the Installation Guide, section 1.4 on page 8.

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 on page 19. 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 -H -ppackage MyUserOps.def
```

The parameter "-u" signals that an interface for **u**ser-defined operators should be generated. This parameter can be omitted as it is the default[1]. The option -H generates the main package interface code (Hpackage.c). 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
```

You either call hcomp in the directory of the DEF files (typically the package subdirectory def) or specify the path to the DEF files.

To use the package in a programming language, you must also generate the appropriate interface code. In the example above, the corresponding C files (HCpackage.c and HCpackage.h) are created with the option -C. Their names depend on the host language and the specified package, not on the names of the input DEF files.

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 normal HALCON system.

```
hcomp -H [-ppackage] <DEF files>
```

to create the main interface code (`Hpackage.c`), which is necessary independent whether you want to use the package in HDevelop or in a programming language.

Additionally, the commands in the following table create the needed files for the corresponding programming languages.

Language	Command	Created Files
C	`hcomp -C [-p`*package*`] <`*DEF files*`>`	`HCpackage.c` `HCpackage.h`
C++	`hcomp -P [-p`*package*`] <`*DEF files*`>`	`HCPPpackage.cpp` `HCPPpackage.h`
C#	`hcomp -N [-p`*package*`] <`*DEF files*`>`	`HDOTNETpackage.cs`

The generated source files are the basis for the interface libraries, see section 7.2 on page 100.

7.1.2 Creating the Help Files

Help files are necessary for the online access to the operator knowledge base via HALCON operators like `get_operator_info`. For example, 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 English.

```
hcomp -M -l<language> <DEF files>
```

creates files with the stem `operators_`, followed by the language (default: `en_US`), and the following extensions:

Extension	Description
`.ref`	The operator description (text, value lists, ...).
`.idx`	Index for every operator specifying the start address of its entry within the `*.ref` file.
`.key`	List of keywords (index entries) with the associated operators.
`.num`	Specification of the number of parameters per parameter class for every operator.
`.sta`	Specification of the parameter names and the chapter names for every operator.

In addition to the option `-M`

```
-c<TAGS-FILE>
```

can be set in order to insert the source file and the line number of the supply procedure of the operator into the `*.ref` file. To do this, a *TAGS-File* (usually `TAGS`) must have been generated by the `etags` command before.

7.1.3 Creating HTML Reference Files

`hcomp` can generate HTML reference files. Thus, its easy to provide an online documentation of new operators that can be accessed with an HTML browser. Use

```
hcomp -R [-p<package>] -l<lang> -a<author-info> -f<copyright-footer> <DEF files>
```

to create HTML files. Note that the created HTML files contain the information for all programming language interfaces.

With the (optional) parameters `-a` and `-f` you can customize the created manual: The author information passed with the parameter `-a` appears on the main HTML page; the string passed with the parameter `-f` appears at the bottom of every generated HTML page.

The following files are generated:

- index.html
  Home page containing the chapter structure and links to the different (sub-)chapters (toc_<chapter>_<subchapter>.html)
- index_by_name.html
  Alphabetical list of all operators with links to the corresponding operator description pages (files <operator>.html)
- index_classes.html
  List of all classes (for .NET) with links to the corresponding description pages (files <class>.html)
- toc_<chapter>_<subchapter>.html
  For every (sub-)chapter: A list of all operators within the chapter with links to the corresponding operator description pages (files <operator>.html)
- <operator>.html
  A description page for each operator
- <class>.html
  A description page for each class (for .NET)

7.1.4 Creating the PDF Reference Manuals

hcomp can also generate LaTeX files containing the complete reference manuals adapted to the different supported programming languages, i.e. the description of the operators is provided in a specific syntax. Use

```
hcomp -Rlatex:<plang> [-p<package>] -l<lang> -a<author-info> -f<copyright-footer> <DEF files>
```

to create LaTeX files. *plang* can be set to c, cpp, com, dotnet, and hdevelop (HDevelop syntax). hdevelop is default and may therefore be omitted. The package name specified with [-p] is used for the title page of the manual.

The information passed with the parameter -a appears in the PDF version of the manual (see below) in the field "Author" of the dialog describing the document. With the parameter -f you can pass a string, which then appears on the back side of the title page. A typical use of this parameter is to print a copyright notice.

The generated LaTeX files use the package halcon_reference_en_US.sty (or similar for other languages). This package resides in the directory $HALCONROOT/doc/macros (Linux notation). You must include this directory in the environment variable TEXINPUTS. If the variable is empty, set it to:

```
:$HALCONROOT/doc/macros
```

Otherwise, LaTeX will be unable to find it (typical error message: File 'halcon_reference_en_US.sty' not found.

The LaTeX file(s) can be transformed into PDF files by using pdflatex and makeindex. Note that pdflatex must be called several times to get the references right; to get a correct table of context and index the sequence of calls is pdflatex > pdflatex > makeindex > pdflatex > pdflatex.

The generated PDF contains interactive bookmarks and hyperlinks similar to the HTML version (and the author information passed with the parameter -a). Note that this functionality requires that your LaTeX distribution includes the package hyperref; if this package exists but is too old, a warning will be issued, as this can result in errors during the LaTeX process or in suboptimal PDFs.

For a printer-friendly variant of the PDF file, you can disable the interactive features by modifying the file index.tex before starting pdflatex: Replace the line (possibly with different language key)

```
\usepackage[pdflinks]{halcon_reference_en_US}
```

by

```
\usepackage{halcon_reference_en_US}
```

If you want to create both documents with and without PDF links, please use different directories or remember to delete all auxiliary LaTeX files before starting to `pdflatex` the modified `index.tex`, because the two versions are not compatible.

7.1.5 Miscellaneous

Here are some more options supported by `hcomp`:

Option	Description
`hcomp -i` *filename* `<DEF files>`	Consider only the operators listed in *filename*.
`hcomp -x` *filename* `<DEF files>`	Do *not* consider the operators listed in *filename*.

7.2 Generating HALCON Packages

Once the new operators have been described in DEF files and implemented (supply and action procedures) a couple of dynamic objects (DLLs in Windows, shared libraries in Linux environments, respectively) 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.,

```
%HALCONEXAMPLES%\extension_package\halconuser
```

Please note that the package paths in `HALCONEXTENSIONS` are separated by semicolons (Windows) or colons (Linux), see also section 1.3 on page 13.

7.2.1 Creating the Operator Libraries

The supply and action procedures must be encapsulated in a DLL (Windows), shared library (Linux). The name of this shared object must be consistent with the package name (plus file extension). For the use by HALCON XL, a second version of the library must be created, with the additional suffix `xl` to the name. Please refer to section 7.2.6 and the documentation of your programming environment for more details (also about the following sections).

Under Windows the generated DLLs must be placed in the subdirectory `bin\%HALCONARCH%`, the libraries in the subdirectory `lib\%HALCONARCH%`. Under Linux, the generated shared libraries must be placed in the subdirectory `lib/$HALCONARCH`, binaries in the subdirectory `bin/$HALCONARCH` of the package. .NET assemblies must be placed in the subdirectory `bin\dotnet`.

Independent of whether you want to use the package in HDevelop or in a programming language, you must create the main interface library `package`. This library is based on a single C file `Hpackage.c`, which is generated via

```
hcomp -H -ppackage <DEF file(s)>
```

7.2.2 Creating the C Interface

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

```
hcomp -C -ppackage <DEF file(s)>
```

from the DEF files of your new operators. Note that `hcomp` simultaneously generates the file `HCpackage.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 `packagecpp` residing in the subdirectory `lib\%HALCONARCH%` of the package. This library is based on the file `HCPPpackage.cpp` generated via

```
hcomp -P -ppackage <DEF file(s)>
```

from the DEF files of your self developed operators. Note that `hcomp` simultaneously generates the file `HCPPpackage.h` containing the C++ prototypes of your new operators. Include this file in your C++ programs using these operators.

7.2.4 Creating the .NET Interface

To access new HALCON operators inside a package within .NET programs, you must create the interface assembly `packagedotnet` residing in the subdirectory `lib\dotnet` of the package. This assembly is based on the C# source file `HDOTNETpackage.cs`, which is generated via

```
hcomp -N -ppackage <DEF file(s)>
```

from the DEF files of your new operators.

7.2.5 Creating New Applications

To create new application programs (written in C, C++, or .NET languages) based on your own HALCON operators, you must link the corresponding language interface libraries `packagec` or `packagecpp` to your objects, or reference the corresponding .NET assembly `packagedotnet`. For C or C++ applications, you furthermore need the HALCON library itself and the HALCON/C or HALCON/C++ library (as for any HALCON application). For .NET applications, you need the HALCON/.NET assembly.

7.2.6 Additional Information for Specific Platforms

The previous sections summarized the generation of HALCON extensions in general. This section contains additional information for specific platforms. The main differences concern the name handling and the generation of shared libraries / DLLs.

7.2.6.1 Generating Packages Under Windows

To activate a package, its complete path, e.g.,

`%HALCONEXAMPLES%\extension_package\halconuser`

must be included in the environment variable `HALCONEXTENSIONS`. In the Windows version the package paths in `HALCONEXTENSIONS` are separated by semicolons.

Note that DLLs must be stored in the subdirectory `bin\%HALCONARCH%` of the package, the corresponding libraries in the subdirectory `lib\%HALCONARCH%`.

- **Creating object files for the HALCON XL version**:

  By default, using `CMakeLists.txt`, the package will be built with the normal version of HALCON. If you want to build with HALCON XL, set the option `HALCON_XL` to `ON` or `1` in CMake during the configuration step. For this, use the following syntax:

  ```
  cmake -DHALCON_XL=1 %HALCONEXAMPLES%\extension_package\halconuser
  ```

- **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 `packagexl.dll` and `packagexl.lib` for HALCON XL, set the option `HALCON_XL` to `ON` or `1` in CMake during the configuration step. The corresponding object files must be linked to the HALCON XL library `halconxl.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 `HCpackage.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 `packagecxl.dll` and `packagecxl.lib` for HALCON XL, set the option `HALCON_XL` to `ON` or `1` in CMake during the configuration step. The corresponding object files must be linked to the HALCON XL version of the libraries, i.e., `packagexl.lib`, `halconcxl.lib`, and `halconxl.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 file `HCPPpackage.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 `packagecppxl.dll` and `packagecppxl.lib` for HALCON XL, set the option `HALCON_XL` to `ON` or `1` in CMake during the configuration step. The corresponding object files must be linked to the HALCON XL version of the libraries, i.e., `packagexl.lib`, `halconcppxl.lib`, and `halconxl.lib`.

- **Creating the .NET interface assembly** `packagedotnet.dll`:

  The assembly `packagedotnet.dll` providing the .NET interface to new HALCON operators is created directly from the source file `HDOTNETpackage.cs`, together with the XML file that documents the content of the assembly.

  To create the assembly `packagedotnetxl.dll` for HALCON XL, set the option `HALCON_XL` to `ON` or `1` in CMake during the configuration step.

- **Creating new applications**:

  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). For .NET applications, you reference the libraries `packagex.lib` and `halconx.lib` or the assemblies `packagedotnet.dll` and `halcondotnet.dll`, respectively.

  To create a HALCON XL version of your application, just use the HALCON XL version of the libraries (e.g., `packagecxl.lib` and `halconcxl.lib` in case of a C application).

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

  ```
  %HALCONEXAMPLES%\extension_package\halconuser\bin\%HALCONARCH%
  ```

  has to be added to the environment variable `PATH`. This is also true for .NET applications.

  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 to 8 MB is recommended.

The directory `%HALCONEXAMPLES%\extension_package\halconuser` contains the example CMake file `CMakeLists.txt`, which can be used to create the example package halconuser and the example applications based on the package halconuser. If required, download CMake (version 3.7.1 or later) from the CMake website and install it. To build the example package halconuser using CMake, do the following:

```
mkdir build
cd build
cmake -DCMAKE_INSTALL_PREFIX=%HALCONEXAMPLES%\extension_package \
      HALCONEXAMPLES%\extension_package\halconuser
cmake --build . --target install
```

By default, the normal version of HALCON is created. If you want to build using HALCON XL, set the option `HALCON_XL` to `ON` or `1` in CMake during the configuration step.

7.2.6.2 Generating Packages Under Linux

To activate a package, its complete path, e.g.,

```
$HALCONEXAMPLES/extension_package/halconuser
```

must be included in the environment variable `HALCONEXTENSIONS`. In the Linux version, the package paths in `HALCONEXTENSIONS` are separated by colons. Please see also the comments on installing a HALCON package on a Linux system in section 1.3.2 on page 14.

- **Compiling the source code**:

  To generate shared libraries, you must produce position-independent code. For the `gcc/g++` compiler, this is done using the option `-fPIC`. For `gcc`, we also recommend the options `-march=pentium -mcpu=pentiumpro -ansi -fno-strict-prototype`.

- **Creating object files for the HALCON XL version**:

  By default, using `CMakeLists.txt`, the package will be built with the normal version of HALCON. If you want to build with HALCON XL, set the option `HALCON_XL` to `ON` or `1` in CMake during the configuration step. For this, use the following syntax:

  ```
  cmake -DHALCON_XL=1 %HALCONEXAMPLES%\extension_package\halconuser
  ```

- **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 `libpackagexl.so` for HALCON XL, set the option `HALCON_XL` to `ON` or `1` in CMake during the configuration step. The corresponding object files 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 `libpackagecxl.so` for HALCON XL, set the option `HALCON_XL` to `ON` or `1` in CMake during the configuration step. The corresponding object file must be linked to the HALCON XL version of the operator library, i.e., `libpackagexl.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 lib `libpackage.so` must be linked.

  To create the library `libpackagecppxl.so` for HALCON XL, set the option `HALCON_XL` to `ON` or `1` in CMake during the configuration step. The corresponding object files must be linked to the HALCON XL version of the operator library, i.e., `libpackage.so`.

- **Creating the .NET interface assembly** `packagedotnet.dll`:

  You can use the Mono compiler to create the assembly `packagedotnet.dll` providing the .NET interface to new HALCON operators. This assembly is based on the C# source file HDOTNETpackage.cs, see section 7.2.4 on page 101. Note that you need a successfully installed and configured version of Mono to create the assembly.

For this, run the following commands in a shell:

```
mcs HDOTNETpackage.cs /target:library /out:Hpackage.dll /doc:Hpackage.xml /warn:1 -r:halcondotnet.dll
```

To create an executable you have to reference the new created assembly `Hpackage.dll` and the HALCON .NET assembly as well.

- **Creating new applications**:

  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). For .NET applications, you reference the assemblies `packagedotnet.dll` and `halcondotnet.dll`.

  Furthermore, you must add the package library subdirectory `lib/$HALCONARCH` to the environment variable `LD_LIBRARY_PATH`.

  To create a HALCON XL version of your application, just use the HALCON XL version of the libraries (e.g., `libpackagexl.so`, `libpackagecxl.so`, `libhalconxl.so`, and `libhalconcxl.so` in case of a C application).

7.3 HALCON Directories

HALCON always presumes that the environment variable `HALCONROOT` contains the path of the HALCON installation directory. The following subdirectories are important for creating a new system:

- `include`
  HALCON include files for compiling
- `lib\%HALCONARCH%`
  HALCON libraries
- `dotnet20,dotnet35`
  HALCON/.NET assemblies for the different versions of the .NET Framework (for more information, see the Programmer's Guide, table 10.1 on page 64)
- `bin\%HALCONARCH%`
  HALCON compiler `hcomp`, under Windows also the HALCON DLLs
- `doc\macros`
  LaTeX files for generating PDF reference files or manuals

The environment variable `HALCONEXAMPLES` contains the path of the HALCON programming examples. The following subdirectory is useful for creating a new system:

- `extension_package\halconuser`
  Example for a HALCON extension package

Appendix A

HALCON Error Codes

In this section all HALCON error codes are summarized, see also section 3.7 on page 47.

Error Name	Code	Description
H_ERR_WIPT1	1201	Wrong type of control parameter 1
H_ERR_WIPT2	1202	Wrong type of control parameter 2
H_ERR_WIPT3	1203	Wrong type of control parameter 3
H_ERR_WIPT4	1204	Wrong type of control parameter 4
H_ERR_WIPT5	1205	Wrong type of control parameter 5
H_ERR_WIPT6	1206	Wrong type of control parameter 6
H_ERR_WIPT7	1207	Wrong type of control parameter 7
H_ERR_WIPT8	1208	Wrong type of control parameter 8
H_ERR_WIPT9	1209	Wrong type of control parameter 9
H_ERR_WIPT10	1210	Wrong type of control parameter 10
H_ERR_WIPT11	1211	Wrong type of control parameter 11
H_ERR_WIPT12	1212	Wrong type of control parameter 12
H_ERR_WIPT13	1213	Wrong type of control parameter 13
H_ERR_WIPT14	1214	Wrong type of control parameter 14
H_ERR_WIPT15	1215	Wrong type of control parameter 15
H_ERR_WIPT16	1216	Wrong type of control parameter 16
H_ERR_WIPT17	1217	Wrong type of control parameter 17
H_ERR_WIPT18	1218	Wrong type of control parameter 18
H_ERR_WIPT19	1219	Wrong type of control parameter 19
H_ERR_WIPT20	1220	Wrong type of control parameter 20
H_ERR_WIPV1	1301	Wrong value of control parameter 1
H_ERR_WIPV2	1302	Wrong value of control parameter 2
H_ERR_WIPV3	1303	Wrong value of control parameter 3
H_ERR_WIPV4	1304	Wrong value of control parameter 4
H_ERR_WIPV5	1305	Wrong value of control parameter 5
H_ERR_WIPV6	1306	Wrong value of control parameter 6
H_ERR_WIPV7	1307	Wrong value of control parameter 7
H_ERR_WIPV8	1308	Wrong value of control parameter 8
H_ERR_WIPV9	1309	Wrong value of control parameter 9
H_ERR_WIPV10	1310	Wrong value of control parameter 10
H_ERR_WIPV11	1311	Wrong value of control parameter 11
H_ERR_WIPV12	1312	Wrong value of control parameter 12
H_ERR_WIPV13	1313	Wrong value of control parameter 13
H_ERR_WIPV14	1314	Wrong value of control parameter 14
H_ERR_WIPV15	1315	Wrong value of control parameter 15
H_ERR_WIPV16	1316	Wrong value of control parameter 16
H_ERR_WIPV17	1317	Wrong value of control parameter 17
H_ERR_WIPV18	1318	Wrong value of control parameter 18

Error Name	Code	Description
H_ERR_WIPV19	1319	Wrong value of control parameter 19
H_ERR_WIPV20	1320	Wrong value of control parameter 20
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_WIPN17	1417	Wrong number of values of control parameter 17
H_ERR_WIPN18	1418	Wrong number of values of control parameter 18
H_ERR_WIPN19	1419	Wrong number of values of control parameter 19
H_ERR_WIPN20	1420	Wrong number of values of control parameter 20
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_OONTB	1510	Number of output objects too big
H_ERR_BREAK	20	Operator canceled by user
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_LIC_NO_LICENSE	2003	No license found
H_ERR_LIC_NO_MODULES	2005	No modules in license (invalid VENDOR_STRING)
H_ERR_LIC_NO_LIC_OPER	2006	No license for this operator
H_ERR_LIC_BADPLATFORM	2008	Vendor keys do not support this platform
H_ERR_LIC_BADVENDORKEY	2009	Bad vendor keys
H_ERR_LIC_BADSYSDATE	2021	System clock has been set back. This error can only occur when the FEATURE line contains an expiration date
H_ERR_LIC_BAD_VERSION	2022	Invalid format for version argument
H_ERR_LIC_CANTCONNECT	2024	Cannot establish a connection with a license server
H_ERR_LIC_MAXSESSIONS	2028	Maximum session limit reached
H_ERR_LIC_MAXUSERS	2029	All licenses in use
H_ERR_LIC_NO_SERVER_IN_FILE	2030	No license server specified for floating license
H_ERR_LIC_NOFEATURE	2031	Can not find feature in the license file
H_ERR_LIC_OLDVER	2033	License file does not support a version this new
H_ERR_LIC_PLATNOTLIC	2034	This platform not authorized by license - running on platform not included in PLATFORMS list

Error Name	Code	Description
H_ERR_LIC_SERVBUSY	2035	License server busy - the request should be retried (this is a rare occurrence)
H_ERR_LIC_NOCONFFILE	2036	could not find license file
H_ERR_LIC_BADFILE	2037	Invalid license file syntax
H_ERR_LIC_NOSERVER	2038	Cannot connect to a license server
H_ERR_LIC_NOTTHISHOST	2041	Invalid host
H_ERR_LIC_LONGGONE	2042	Feature has expired
H_ERR_LIC_BADDATE	2043	Invalid date format in license file
H_ERR_LIC_BADCOMM	2044	Invalid returned data from license server
H_ERR_LIC_BADHOST	2045	Cannot resolve server name
H_ERR_LIC_CANTWRITE	2047	Cannot write data to license server
H_ERR_LIC_SERVLONGGONE	2051	License server does not support this version of this feature
H_ERR_LIC_TOOMANY	2052	Request for more licenses than this feature supports
H_ERR_LIC_CANTFINDETHER	2055	Cannot find ethernet device
H_ERR_LIC_NOREADLIC	2056	Cannot read license file
H_ERR_LIC_DATE_TOOBIG	2067	Date too late for binary format
H_ERR_LIC_NOSERVRESP	2069	Server did not respond to message
H_ERR_LIC_SETSOCKFAIL	2075	setsockopt() failed
H_ERR_LIC_BADCHECKSUM	2076	Message checksum failure
H_ERR_LIC_INTERNAL_ERROR	2082	Internal licensing error
H_ERR_LIC_NOSERVCAP	2087	Server does not support this request
H_ERR_LIC_POOL	2091	This feature is available in a different license pool
H_ERR_HEN_CANCEL	21	Operator canceled
H_ERR_WOOPI	2100	Wrong index for output object parameter
H_ERR_WIOPI	2101	Wrong index for input object parameter
H_ERR_WOI	2102	Wrong index for image object (too big or too small)
H_ERR_WRCN	2103	Wrong number region/image component (see: HGetComp)
H_ERR_WRRN	2104	Wrong relation name
H_ERR_AUDI	2105	Access to undefined gray value component
H_ERR_WIWI	2106	Wrong image width
H_ERR_WIHE	2107	Wrong image height
H_ERR_ICUNDEF	2108	Undefined gray value component
H_ERR_CANCEL	22	Operator canceled by interrupt_operator
H_ERR_IDBD	2200	Inconsistent data of data base (typing)
H_ERR_WICPI	2201	Wrong index for input control parameter
H_ERR_DBDU	2202	Data of data base not defined (internal error)
H_ERR_PNTL	2203	Number of operators too big
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_ICHV	2211	Incompatible HALCON versions
H_ERR_ICOI	2212	Incompatible operator interface
H_ERR_XPKG_WXID	2220	Wrong extension package id
H_ERR_XPKG_WOID	2221	Wrong operator id
H_ERR_XPKG_WOIID	2222	Wrong operator information id
H_ERR_TIMEOUT_BREAK	23	Operator canceled by userset timeout
H_ERR_LIC_NODONGLE	2300	Dongle not attached, or can’t read dongle
H_ERR_LIC_NODONGLEDRIVER	2301	Missing Dongle Driver
H_ERR_LIC_TIMEOUT	2318	Timeout
H_ERR_LIC_INVALID_CERTIFICATE	2321	License server certificate invalid
H_ERR_LIC_INVALID_TLS_CERTIFICATE	2335	License server SSL/TLS certificate invalid
H_ERR_LIC_BAD_ACTREQ	2339	Invalid activation request received
H_ERR_LIC_NOT_ALLOWED	2345	Specified operation is not allowed
H_ERR_LIC_ACTIVATION	2348	Activation error
H_ERR_LIC_NO_CM_RUNTIME	2379	No CodeMeter Runtime installed (version 6.30 or later required)

Error Name	Code	Description
H_ERR_LIC_CM_RUNTIME_TOO_OLD	2380	Installed version of CodeMeter Runtime is too old (version 6.30 or later required)
H_ERR_LIC_WRONG_EDITION	2381	License is for wrong edition of HALCON
H_ERR_LIC_UNKNOWN_FLAGS	2382	License contains unknown FLAGS
H_ERR_LIC_PREVIEW_EXPIRED	2383	HALCON preview version expired
H_ERR_LIC_NEWVER	2384	License file does not support a version this old
H_ERR_CTPL_WTYP	2400	Wrong Hctuple element type
H_ERR_CPAR_WTYP	2401	Wrong Hcpar parameter type
H_ERR_CTPL_WIDX	2402	Wrong Hctuple index
H_ERR_WFV	2403	Unsupported file version
H_ERR_WRONG_HANDLE_TYPE	2404	Invalid handle type
H_ERR_WVTYP	2410	Wrong vector type
H_ERR_WVDIM	2411	Wrong vector dimension
H_ERR_WHDL	2450	Wrong (unknown) HALCON handle
H_ERR_WID	2451	Wrong HALCON id, no data available
H_ERR_IDOOR	2452	HALCON id out of range
H_ERR_HANDLE_NULL	2453	HALCON handle is NULL
H_ERR_HANDLE_CLEARED	2454	HALCON handle was already cleared
H_ERR_HANDLE_NOSER	2455	HALCON handle of this type cannot be serialized
H_ERR_HANDLE_CYCLES	2456	Cyclic references of handles were detected
H_ERR_WT_CTRL_EXPECTED	2460	Type mismatch: expected control value, got iconic value
H_ERR_WT_ICONIC_EXPECTED	2461	Type mismatch: expected iconic value, got control value
H_ERR_XPI_INIT_NOT_CALLED	2500	Init function of an HALCON extension that was build with the hlibxpi interface was not called (check for correct hlibxpi export)
H_ERR_XPI_NO_INIT_FOUND	2501	Init function of the required HALCON extension was not found (old interface without hlibxpi support or the hlibxpi export failed)
H_ERR_XPI_UNRES	2502	Function call in HALCON extension couldn't be resolved via hlibxpi
H_ERR_XPI_HLIB_TOO_OLD	2503	The version of the HALCON library is too old for the required HALCON extension
H_ERR_XPI_XPI_TOO_OLD	2504	The required HALCON extension uses an hlibxpi version that is no longer supported by the current HALCON version
H_ERR_XPI_MAJOR_TOO_SMALL	2505	The current HALCON version uses an hlibxpi version that is not supported by the required HALCON extension (major number too small)
H_ERR_XPI_MINOR_TOO_SMALL	2506	The current HALCON version uses an hlibxpi version that is not supported by the required HALCON extension (minor number too small)
H_ERR_XPI_INT_WRONG_MAJOR	2507	Internal error: wrong version number in hlibxpi symbol list
H_ERR_XPI_UNKNOW_HLIB_VER	2508	hlibxpi cannot connect to hlib: unknown hlib version
H_ERR_HW_WFF	2800	Wrong hardware knowledge file format
H_ERR_HW_WFV	2801	Wrong hardware knowledge file version
H_ERR_HW_RF	2802	Error while reading the hardware knowledge
H_ERR_HW_WF	2803	Error while writing the hardware knowledge
H_ERR_HW_TF	2804	Tag in hardware knowledge file not found
H_ERR_HW_CPU	2805	No CPU information in hardware knowledge file found
H_ERR_HW_AOP	2806	No aop information in hardware knowledge file found
H_ERR_HW_HVAR	2807	No aop information for this HALCON variant found
H_ERR_HW_HARCH	2808	No aop information for this HALCON architecture found
H_ERR_HW_HOP	2809	No aop information for specified Operator found
H_ERR_HW_WAOPM	2810	Unknown aop model
H_ERR_HW_WTD	2811	Wrong tag derivate in hardware knowledge file
H_ERR_HW_IE	2812	Internal error while processing hardware knowledge
H_ERR_HW_CANCEL	2813	Optimizing aop was canceled
H_ERR_GV_WA	2830	Wrong access to global variable
H_ERR_GV_NC	2831	Used global variable does not exist

Error Name	Code	Description
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	Sequential 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_UNKPARVAL	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
H_ERR_TMPFNF	2869	Error while accessing temporary file
H_ERR_MQCNCL	2890	Message queue wait operation canceled
H_ERR_MQOVL	2891	Message queue overflow
H_ERR_MQCLEAR	2892	Threads still wait on message queue while clearing it
H_ERR_M_WRFILE	2893	Invalid file format for a message
H_ERR_DICT_KEY	2894	Dictionary does not contain requested key
H_ERR_DICT_TUPLE_LENGTH	2895	Tuple in dictionary has incorrect length
H_ERR_DICT_TUPLE_TYPE	2896	Tuple in dictionary is of incorrect type
H_ERR_PTHRD_SCHED	2900	Error while forcing a context switch
H_ERR_SCHED_GAFF	2901	Error while accessing the CPU affinity
H_ERR_SCHED_SAFF	2902	Error while setting the CPU affinity
H_ERR_CO_WSO	2950	Wrong synchronization object
H_ERR_CO_WOCO	2952	Wrong thread object
H_ERR_CO_IOPNI	2953	Input Object was not initialized
H_ERR_CO_ICPNI	2954	Input control parameter is not initialized
H_ERR_CO_OOPNI	2955	Output Object parameter is not initialized
H_ERR_CO_OCPNI	2956	Output control parameter is not initialized
H_ERR_PTHRD_CR	2970	creation of pthread failed
H_ERR_PTHRD_DT	2971	pthread-detach failed
H_ERR_PTHRD_JO	2972	pthread-join failed

Error Name	Code	Description
H_ERR_PTHRD_MI	2973	Initialization of mutex variable failed
H_ERR_PTHRD_MD	2974	Deletion of mutex variable failed
H_ERR_PTHRD_ML	2975	Lock of mutex variable failed
H_ERR_PTHRD_MU	2976	Unlock of mutex variable failed
H_ERR_PTHRD_CS	2977	Failed to signal pthread condition variable
H_ERR_PTHRD_CW	2978	Failed to wait for pthread condition variable
H_ERR_PTHRD_CI	2979	Failed to init pthread condition variable
H_ERR_PTHRD_CD	2980	Failed to destroy pthread condition variable
H_ERR_PTHRD_ES	2981	Failed to signal event
H_ERR_PTHRD_EW	2982	Failed to wait for an event
H_ERR_PTHRD_EI	2983	Failed to init an event
H_ERR_PTHRD_ED	2984	Failed to destroy an event
H_ERR_PTHRD_TSDC	2985	Failed to create a tsd key
H_ERR_PTHRD_TSDS	2986	Failed to set a tsd key
H_ERR_PTHRD_TSDG	2987	Failed to get a tsd key
H_ERR_PTHRD_TSDF	2988	Failed to free a tsd key
H_ERR_PTHRD_BA	2989	Aborted waiting at a barrier
H_ERR_DCDG_FLE	2990	'Free list' is empty while scheduling
H_ERR_MSG_PNCI	2991	Communication partner not checked in
H_ERR_MSG_CSAI	2992	You can not start the communication system while running it
H_ERR_MSG_CSNI	2993	Communication partner not checked in
H_ERR_PTHRD_BI	2994	Failed to init a barrier
H_ERR_PTHRD_BW	2995	Failed to wait at a barrier
H_ERR_PTHRD_BD	2996	Failed to destroy a barrier
H_ERR_RCOIMA	3010	Region completely outside of the image domain
H_ERR_ROOIMA	3011	Region (partially) outside of the definition range of the image
H_ERR_RIEI	3012	Intersected definition range region / image empty
H_ERR_EDEF	3013	Image with empty definition range (=> no gray values)
H_ERR_IIEI	3014	No common image point of two images
H_ERR_FLTS	3015	Wrong region for image (first row < 0)
H_ERR_LLTB	3016	Wrong region for image (column in last row >= image width)
H_ERR_UENOI	3017	Number of images unequal in input parameters
H_ERR_HTS	3018	Image height too small
H_ERR_WTS	3019	Image width too small
H_ERR_CHSEG	3020	Internal error: multiple call of HRLInitSeg()
H_ERR_RLSEG1	3021	Internal error: HRLSeg() not initialized
H_ERR_WGAUSSM	3022	Wrong size of filter for Gauss
H_ERR_FSEIS	3033	Filter size exceeds image size
H_ERR_FSEVAN	3034	Filter size have to be odd
H_ERR_FSTOBIG	3035	Filter is too big
H_ERR_EMPTREG	3036	Input region is empty
H_ERR_DOM_DIFF	3037	Domains of the input images differ
H_ERR_ROWTB	3040	Row value of a coordinate > 2^30-1
H_ERR_ROWTS	3041	Row value of a coordinate < -2^30+1
H_ERR_COLTB	3042	Column value of a coordinate > 2^30-1
H_ERR_COLTS	3043	Column value of a coordinate < -2^30+1
H_ERR_WRTHR	3100	Wrong segmentation threshold
H_ERR_UNKF	3101	Unknown feature
H_ERR_UNKG	3102	Unknown gray value feature
H_ERR_EINCC	3103	Internal error in HContCut
H_ERR_EINCP1	3104	Error in HContToPol: distance of points too big
H_ERR_EINCP2	3105	Error in HContToPol: contour too long
H_ERR_TMR	3106	Too many rows (IPImageTransform)
H_ERR_SFZ	3107	Scaling factor = 0.0 (IPImageScale)
H_ERR_OOR	3108	Wrong range in transformation matrix
H_ERR_NEF	3109	Internal error in IPvvf: no element free

Error Name	Code	Description
H_ERR_NOOB	3110	Number of input objects is zero
H_ERR_EMPOB	3111	At least one input object has an empty region
H_ERR_NPOT	3112	Operation allowed for rectangular images 2**n only
H_ERR_TMEP	3113	Too many relevant points (IPHysterese)
H_ERR_LTB	3114	Number of labels in image too big
H_ERR_NNLA	3115	No labels with negative values allowed
H_ERR_WFS	3116	Wrong filter size (too small?)
H_ERR_IWDS	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_DNOC	3122	Number of channels in the input parameters are different
H_ERR_WRCFAFLT	3123	Wrong color filter array type
H_ERR_WRCFAINT	3124	Wrong color filter array interpolation
H_ERR_NO_AFFTRANS	3125	Homogeneous matrix does not represent an affine transformation
H_ERR_INPNOBDRY	3126	Inpainting region too close to the image border
H_ERR_DSIZESD	3127	Source and destination differ in size
H_ERR_TMFEAT	3128	To many Features
H_ERR_AXIS_UNDEF	3129	Reflection axis undefined
H_ERR_COWTS	3131	Coocurrence matrix: too little columns for quantization
H_ERR_COHTS	3132	Coocurrence matrix: too little rows for quantization
H_ERR_NUM_COLMN	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_NOT_SYM	3136	Matrix is not symmetric
H_ERR_NUM_COLS	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_WNOLI	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_NE_NPTS	3175	Number of suitable pixels too small to reliably estimate the noise
H_ERR_WNEE	3200	Different number of entries/exits in HContCut
H_ERR_REF	3201	Reference to the underlying contour is missing
H_ERR_XLDWT	3250	Wrong XLD type
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 operator
H_ERR_XLD_CRD	3256	Regression parameters of contours already computed
H_ERR_XLD_CRND	3257	Regression parameters of contours not yet entered! Please compute them by calling regress_cont_xld
H_ERR_DBXC	3258	Data base: XLD object has been deleted
H_ERR_DBWXID	3259	Data base: object has no XLD-ID

Error Name	Code	Description
H_ERR_XLD_WNP	3260	Internal error: wrong number of contour points allocated
H_ERR_XLD_CAND	3261	Contour attribute not defined
H_ERR_FIT_ELLIPSE	3262	Ellipse fitting failed
H_ERR_FIT_CIRCLE	3263	Circle fitting failed
H_ERR_FIT_CLIP	3264	All points classified as outliers (ClippingFactor too small or used points not similar to primitive)
H_ERR_FIT_QUADRANGLE	3265	Quadrangle fitting failed
H_ERR_INCOMPL_RECT	3266	No points found for at least one side of the rectangle
H_ERR_XLD_COI	3267	A contour point lies outside of the image
H_ERR_FIT_NOT_ENOUGH_POINTS	3274	Not enough valid points for fitting the model
H_ERR_NWF	3275	No ARC/INFO world file
H_ERR_NAIGF	3276	No ARC/INFO generate file
H_ERR_DXF_UEOF	3278	Unexpected end of file while reading DXF file
H_ERR_DXF_CRGC	3279	Cannot read DXF-group code from file
H_ERR_DXF_INAPP	3280	Inconsistent number of attributes per point in DXF file
H_ERR_DXF_INAPPN	3281	Inconsistent number of attributes and names in DXF file
H_ERR_DXF_INAPCN	3282	Inconsistent number of global attributes and names in DXF file
H_ERR_DXF_CRAPP	3283	Cannot read attributes from DXF file
H_ERR_DXF_CRAPC	3284	Cannot read global attributes from DXF file
H_ERR_DXF_CRAN	3285	Cannot read attribute names from DXF file
H_ERR_DXF_WPN	3286	Wrong generic parameter name
H_ERR_DXF_IEDT	3289	Internal DXF I/O error: Wrong data type
H_ERR_XLD_ISOL_POINT	3290	Isolated point while contour merging
H_ERR_NURBS_CCBF	3291	Constraints (MaxError/MaxDistance) cannot be fulfilled
H_ERR_NSEG	3292	No segment in contour
H_ERR_NO_ONE_P	3293	Only one or no point in template contour
H_ERR_SESF	3300	Syntax error in file for training
H_ERR_TMFE	3301	Maximum number of attributes per example exceeded
H_ERR_OPSF	3302	Not possible to open file for training
H_ERR_TMSS	3303	Too many data sets for training
H_ERR_TMSAM	3305	Too many examples for one data set for training
H_ERR_TMCLS	3306	Too many classes
H_ERR_TMBOX	3307	Maximum number of cuboids exceeded
H_ERR_OPCF	3308	Not possible to open classificator's file
H_ERR_SCLA	3309	Error while saving the classificator
H_ERR_OPF	3310	Not possible to open protocol file
H_ERR_CLEX	3311	Classificator with this name is already existent
H_ERR_TMCLA	3312	Maximum number of classificators exceeded
H_ERR_CNTL	3313	Name of classificator is too long, >= 20
H_ERR_CLNNF	3314	Classificator with this name is not existent
H_ERR_NCCLA	3315	Current classificator is not defined
H_ERR_CLASS2_ID	3316	Wrong id in classification file
H_ERR_CLASS2_VERS	3317	The version of the classifier is not supported
H_ERR_CLASS_NOSITEM	3318	Serialized item does not contain a valid classifier
H_ERR_TM_NO_CL	3319	Text model does not contain a classifier yet (use set_text_model_param)
H_ERR_KNN_CANNOT_ADD	3320	Adding new features is not possible, because the dataset has been normalized during training. Please create a new classfier and add all training data again or disable normalization during training
H_ERR_GMM_NOTRAINFILE	3330	Invalid file format for GMM training samples
H_ERR_GMM_WRTRAINVERS	3331	The version of the GMM training samples is not supported
H_ERR_GMM_WRSMPFORMAT	3332	Wrong training sample format
H_ERR_GMM_NOCLASSFILE	3333	Invalid file format for Gaussian Mixture Model (GMM)
H_ERR_GMM_WRCLASSVERS	3334	The version of the Gaussian Mixture Model (GMM) is not supported
H_ERR_GMM_TRAIN_UNKERR	3335	Internal error while training the GMM

Error Name	Code	Description
H_ERR_GMM_TRAIN_COLLAPSED	3336	Singular covariance matrix
H_ERR_GMM_TRAIN_NOSAMPLE	3337	No samples for at least one class
H_ERR_GMM_TRAIN_FEWSAMPLES	3338	Too few samples for at least one class
H_ERR_GMM_NOTTRAINED	3340	GMM has not been trained yet
H_ERR_GMM_NOTRAINDATA	3341	No training samples stored in the classifier
H_ERR_GMM_NOSITEM	3342	Serialized item does not contain a valid Gaussian Mixture Model (GMM)
H_ERR_MLP_UNKOUTFUNC	3350	Unknown output function
H_ERR_MLP_NOT01ENC	3351	Target vector not in 0-1 encoding
H_ERR_MLP_NOTRAINDATA	3352	No training samples stored in the classifier
H_ERR_MLP_NOTRAINFILE	3353	Invalid file format for MLP training samples
H_ERR_MLP_WRTRAINVERS	3354	The version of the MLP training samples is not supported
H_ERR_MLP_WRSMPFORMAT	3355	Wrong training sample format
H_ERR_MLP_NOCLASSIF	3356	MLP is not a classifier; use OutputFunction = 'softmax' in create_class_mlp
H_ERR_MLP_NOCLASSFILE	3357	Invalid file format for multilayer perceptron (MLP)
H_ERR_MLP_WRCLASSVERS	3358	The version of the multilayer perceptron (MLP) is not supported
H_ERR_WRNUMCHAN	3359	Wrong number of image channels
H_ERR_MLP_WRNUMPARAM	3360	Number of MLP parameters too large
H_ERR_MLP_NOSITEM	3361	Serialized item does not contain a valid multilayer perceptron (MLP)
H_ERR_LUT_WRNUMCHAN	3370	The number of image channels and the number of dimensions of the look-up table do not match
H_ERR_LUT_NRCHANLARGE	3371	A look-up table can be build only for a 2 or 3 channel classifier
H_ERR_LUT_CANNOTCREAT	3372	Cannot create a look-up table. Please choose a larger 'bit_depth' or select 'fast' for 'class_selection'
H_ERR_SVM_NOTRAINDATA	3380	No training samples stored in the classifier
H_ERR_SVM_NOTRAINFILE	3381	Invalid file format for SVM training samples
H_ERR_SVM_WRTRAINVERS	3382	The version of the SVM training samples is not supported
H_ERR_SVM_WRSMPFORMAT	3383	Wrong training sample format
H_ERR_SVM_NOCLASSFILE	3384	Invalid file format for support vector machine (SVM)
H_ERR_SVM_WRCLASSVERS	3385	The version of the support vector machine (SVM) is not supported
H_ERR_SVM_WRNRCLASS	3386	Wrong number of classes
H_ERR_SVM_NU_TOO_BIG	3387	Nu was chosen too big
H_ERR_SVM_TRAIN_FAIL	3388	SVM training failed
H_ERR_SVM_DO_NOT_FIT	3389	Old SVM and new SVM do not match
H_ERR_SVM_NO_TRAIN_ADD	3390	SVM contains no trained support vectors
H_ERR_SVM_KERNELNOTRBF	3391	Kernel is not an RBF kernel
H_ERR_SVM_NO_TRAIND_FOR_CLASS	3392	Train data does not contain all classes
H_ERR_SVM_NOT_TRAINED	3393	SVM not trained
H_ERR_NOT_TRAINED	3394	Classifier not trained
H_ERR_SVM_NOSITEM	3395	Serialized item does not contain a valid support vector machine (SVM)
H_ERR_ROTNR	3401	Wrong rotation number
H_ERR_GOL	3402	Wrong letter for Golay element
H_ERR_BEZ	3403	Wrong reference point
H_ERR_ITER	3404	Wrong number of iterations
H_ERR_MOSYS	3405	Morphology: system error
H_ERR_ART	3406	Wrong type of boundary
H_ERR_OBJI	3407	Morphology: wrong number of input objects
H_ERR_OBJO	3408	Morphology: wrong number of output objects
H_ERR_PARI	3409	Morphology: wrong number of input control parameter
H_ERR_PARO	3410	Morphology: wrong number of output control parameter
H_ERR_SELC	3411	Morphology: structuring element is infinite
H_ERR_WRNSE	3412	Morphology: wrong name for structuring element

Error Name	Code	Description
H_ERR_WRRLN1	3500	Wrong number of run length rows (chords): smaller than 0
H_ERR_WRRLN2	3501	Number of chords too big. Increase 'current_runlength_number' using set_system!
H_ERR_WRRLL	3502	Run length row with negative length
H_ERR_RLLTB	3503	Run length row >= image height
H_ERR_RLLTS	3504	Run length row < 0
H_ERR_RLCTB	3505	Run length column >= image width
H_ERR_RLCTS	3506	Run length column < 0
H_ERR_CHLTB	3507	For CHORD_TYPE: Number of row too big
H_ERR_CHLTS	3508	For CHORD_TYPE: Number of row too small
H_ERR_CHCTB	3509	For CHORD_TYPE: Number of column too big
H_ERR_MRLE	3510	Exceeding the maximum number of run lengths while automatical expansion
H_ERR_ICCOMPL	3511	Internal error: Region->compl neither true/false
H_ERR_RLEMAX	3512	Internal error: Region->max_num < Region->num
H_ERR_WRRLN3	3513	Internal error: number of chords too big for num_max
H_ERR_OPNOCOMPL	3514	Operator cannot be implemented for complemented regions; intersect the region with a finite region or set clip_region to true with set_system
H_ERR_WIMAW1	3520	Image width must be positive
H_ERR_WIMAW2	3521	Image width > MAX_FORMAT
H_ERR_WIMAH1	3522	Image height must be positive
H_ERR_WIMAH2	3523	Image height > MAX_FORMAT
H_ERR_WIMAW3	3524	Image width <= 0
H_ERR_WIMAH3	3525	Image height <= 0
H_ERR_TMS	3550	Too many segments
H_ERR_NO_INT8_IMAGE	3551	'int8' images are available on 64 bit systems only
H_ERR_POINT_AT_INFINITY	3600	Point at infinity cannot be converted to a Euclidean point
H_ERR_ML_NO_COVARIANCE	3601	Covariance matrix could not be determined
H_ERR_RANSAC_PRNG	3602	RANSAC algorithm didn't find enough point correspondences
H_ERR_RANSAC_TOO_DIFFERENT	3603	RANSAC algorithm didn't find enough point correspondences
H_ERR_PTI_FALLBACK	3604	Internal diagnosis: fallback method had to be used
H_ERR_PTI_TRAFO_SING	3605	Projective transformation is singular
H_ERR_PTI_MOSAIC_UNDERDET	3606	Mosaic is under-determined
H_ERR_COV_NPD	3607	Input covariance matrix is not positive definite
H_ERR_TOO_MANY_POINTS	3608	Number of input points too large
H_ERR_INPC	3620	Inconsistent number of point correspondences
H_ERR_NOPA	3621	At least one image cannot be reached from the reference image
H_ERR_IINE	3622	The image with specified index does not exist
H_ERR_NOCM	3623	Matrix is not a camera matrix
H_ERR_SKNZ	3624	Skew is not zero
H_ERR_ILFL	3625	Illegal focal length
H_ERR_KANZ	3626	Distortion is not zero
H_ERR_VARA	3627	It is not possible to determine all parameters for variable camera parameters
H_ERR_LVDE	3628	No valid implementation selected
H_ERR_KPAR	3629	Kappa can only be determined with the gold-standard method
H_ERR_IMOD	3630	Conflicting number of images and projection mode
H_ERR_PNIC	3631	Error in projection: Point not in any cube map
H_ERR_NO_SOL	3632	No solution found
H_ERR_TINZ	3633	Tilt is not zero
H_ERR_ILMD	3640	Illegal combination of estimation method and parameters to be determined
H_ERR_NOFFTOPT	3650	Invalid file format for FFT optimization data
H_ERR_WRFFTOPTVERS	3651	The version of the FFT optimization data is not supported
H_ERR_WRHALCONVERS	3652	Optimization data was created with a different HALCON variant (Sequential HALCON / Parallel HALCON)

Error Name	Code	Description
H_ERR_OPTFAIL	3653	Storing of the optimization data failed
H_ERR_FFTOPT_NOSITEM	3654	Serialized item does not contain valid FFT optimization data
H_ERR_RDS_NSC	3660	No contours suitable for self-calibration found
H_ERR_RDS_NSS	3661	No stable solution found: please change the inlier threshold or select contours manually
H_ERR_RDS_ISS	3662	Unstable solution: please choose more or different contours
H_ERR_RDS_NEC	3663	Not enough contours for calibration: please select contours manually
H_ERR_INVLD_DISP_RANGE	3690	Disparity range exceeds the maximum permitted
H_ERR_EPIINIM	3700	Epipoles are within the image domain: no rectification possible
H_ERR_EPI_FOV	3701	Fields of view of both cameras do not intersect each other
H_ERR_EPI_RECT	3702	The stereo rectification is impossible
H_ERR_BI_WT_TARGET	3710	Wrong type of parameter target_thickness
H_ERR_BI_WT_THICKNESS	3711	Wrong type of parameter thickness_tolerance
H_ERR_BI_WT_POSITION	3712	Wrong type of parameter position_tolerance
H_ERR_BI_WT_SIGMA	3713	Wrong type of parameter sigma
H_ERR_BI_WV_SIGMA	3714	Wrong value of parameter sigma
H_ERR_BI_WT_THRESH	3715	Wrong type of parameter threshold
H_ERR_BI_WV_TARGET	3716	Wrong value of parameter target_thickness
H_ERR_BI_WV_THICKNESS	3717	Wrong value of parameter thickness_tolerance
H_ERR_BI_WV_POSITION	3718	Wrong value of parameter position_tolerance
H_ERR_BI_WV_THRESH	3719	Wrong value of parameter threshold
H_ERR_BI_WT_REFINE	3720	Wrong type of parameter refinement
H_ERR_BI_WV_REFINE	3721	Wrong value of parameter refinement
H_ERR_BI_WT_RESOL	3722	Wrong type of parameter resolution
H_ERR_BI_WV_RESOL	3723	Wrong value of parameter resolution
H_ERR_BI_WT_POLARITY	3724	Wrong type of parameter polarity
H_ERR_BI_WV_POLARITY	3725	Wrong value of parameter polarity
H_ERR_SOL_EMPTY_MODEL_LIST	3751	No sheet-of-light model available
H_ERR_SOL_WNIW	3752	Wrong input image size (width)
H_ERR_SOL_WNIH	3753	Wrong input image size (height)
H_ERR_SOL_WPROF_REG	3754	The bounding-box around the profile region does not fit the domain of definition of the input image
H_ERR_SOL_CAL_NONE	3755	Calibration extend not set
H_ERR_SOL_UNDEF_DISPARITY	3756	Undefined disparity image
H_ERR_SOL_UNDEF_DISPDOMAIN	3757	Undefined domain for disparity image
H_ERR_SOL_UNDEF_CAMPAR	3758	Undefined camera parameter
H_ERR_SOL_UNDEF_LPCS	3759	Undefined pose of the light plane
H_ERR_SOL_UNDEF_CCS	3760	Undefined pose of the camera coordinate system
H_ERR_SOL_UNDEF_CCS_2_LPCS	3761	Undefined transformation from the coordinate system of the camera to the coordinate system of the light plane
H_ERR_SOL_UNDEF_MOV_POSE	3762	Undefined movement pose for xyz calibration
H_ERR_SOL_WV_SCALE	3763	Wrong value of scale parameter
H_ERR_SOL_WV_PAR_NAME	3764	Wrong parameter name
H_ERR_SOL_WT_METHOD	3765	Wrong type of parameter method
H_ERR_SOL_WT_AMBIGUITY	3766	Wrong type of parameter ambiguity
H_ERR_SOL_WT_SCORE_TYPE	3767	Wrong type of parameter score
H_ERR_SOL_WT_CALIBRATION	3768	Wrong type of parameter calibration
H_ERR_SOL_WT_NUM_PROF	3769	Wrong type of parameter number_profiles
H_ERR_SOL_WT_CAM_PAR	3770	Wrong type of element in parameter camera_parameter
H_ERR_SOL_WT_PAR_POSE	3771	Wrong type of element in pose
H_ERR_SOL_WV_METHOD	3772	Wrong value of parameter method
H_ERR_SOL_WT_THRES	3773	Wrong type of parameter min_gray
H_ERR_SOL_WV_AMBIGUITY	3774	Wrong value of parameter ambiguity
H_ERR_SOL_WV_SCORE_TYPE	3775	Wrong value of parameter score_type
H_ERR_SOL_WV_CALIBRATION	3776	Wrong value of parameter calibration
H_ERR_SOL_WV_NUM_PROF	3777	Wrong value of parameter number_profiles

Error Name	Code	Description
H_ERR_SOL_WV_CAMERA_TYPE	3778	Wrong type of camera
H_ERR_SOL_WN_CAM_PAR	3779	Wrong number of values of parameter camera_parameter
H_ERR_SOL_WN_POSE	3780	Wrong number of values of parameter pose
H_ERR_SOL_NO_TARGET_FOUND	3781	Calibration target not found
H_ERR_SOL_NO_VALID_SOL	3782	The calibration algorithm failed to find a valid solution
H_ERR_SOL_WT_CALIB_OBJECT	3783	Wrong type of parameter calibration_object
H_ERR_SOL_INVALID_CALIB_OBJECT	3784	Invalid calibration object
H_ERR_SOL_NO_CALIB_OBJECT_SET	3785	No calibration object set
H_ERR_SOL_WR_FILE_FORMAT	3786	Invalid file format for sheet-of-light model
H_ERR_SOL_WR_FILE_VERS	3787	The version of the sheet-of-light model is not supported
H_ERR_SOL_CAMPAR_UNSUPPORTED	3788	Camera type not supported by calibrate_sheet_of_light_model
H_ERR_SOL_PAR_CALIB	3790	Parameter does not match the set 'calibration'
H_ERR_SOL_WGV_DISP	3791	The gray values of the disparity image, which specify rows in the camera image, do not fit the height of the camera
H_ERR_TI_WRONGMODEL	3800	Texture inspection model has the wrong type
H_ERR_TI_NOTTRAINED	3801	Texture inspection model is not trained
H_ERR_TI_NOTRAINDATA	3802	Texture inspection model has no training data
H_ERR_TI_NOTRAINFILE	3803	Invalid file format for a texture inspection model
H_ERR_TI_WRTRAINVERS	3804	The version of the texture inspection model is not supported
H_ERR_TI_WRSMPFORMAT	3805	Invalid file format for the training samples
H_ERR_TI_WRSMPVERS	3806	The version of the training sample file is not supported
H_ERR_TI_WRIMGSIZE	3807	At least one of the images within the texture inspection model is too small
H_ERR_TI_WRSMPTEXMODEL	3808	The read samples do not match the current texture model
H_ERR_NOT_ENOUGH_IMAGES	3809	The texture model does not contain any images
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_DIMK	3900	Kalman: Dimension n,m or p has got a undefined value
H_ERR_NOFILE	3901	Kalman: File does not exist
H_ERR_FF1	3902	Kalman: Error in file (row of dimension)
H_ERR_FF2	3903	Kalman: Error in file (row of marking)
H_ERR_FF3	3904	Kalman: Error in file (value is no float)
H_ERR_NO_A	3905	Kalman: Matrix A is missing in file
H_ERR_NO_C	3906	Kalman: Matrix C is missing in file
H_ERR_NO_Q	3907	Kalman: Matrix Q is missing in file
H_ERR_NO_R	3908	Kalman: Matrix R is missing in file
H_ERR_NO_GU	3909	Kalman: G or u is missing in file
H_ERR_NOTSYMM	3910	Kalman: Covariant matrix is not symmetric
H_ERR_SINGU	3911	Kalman: Equation system is singular
H_ERR_SLM_NOT_PERSISTENT	3950	Feature or operation available only in 'persistent' mode
H_ERR_SLM_MSW_TOO_LARGE	3951	'min_stripe_width' is too large for the selected 'pattern_width' and/or 'pattern_height'
H_ERR_SLM_SSW_TOO_LARGE	3952	'single_stripe_width' is too large for the selected 'pattern_width' and/or 'pattern_height'
H_ERR_SLM_MSW_GT_SSW	3953	'min_stripe_width' has to be smaller or equal 'single_stripe_width'
H_ERR_SLM_SSW_LT_MSW	3954	'single_stripe_width' is too small for the selected 'min_stripe_width'
H_ERR_SLM_NOT_PREP	3955	The structured light model can only be decoded after calling 'gen_structured_light_pattern'
H_ERR_SLM_NO_OBJS	3956	The structured light model does not contain the queried object
H_ERR_SLM_WRVERS	3957	The version of the structured light model is not supported
H_ERR_SLM_WRFILE	3958	Invalid file format for a structured light model
H_ERR_SLM_WRONGPATTERN	3959	Parameter does not match the set 'pattern_type'
H_ERR_SLM_NOT_DECODED	3960	'defect_image' can only be computed after calling 'decode_structured_light_pattern'

Error Name	Code	Description
H_ERR_SLM_WRONGMODEL	3961	The structured light model has the wrong type
H_ERR_SLM_WNUMCAMS	3962	The camera setup model has to contain two sets of camera parameters
H_ERR_SLM_WPATTSIZE	3963	'pattern_width' and 'pattern_height' do not match the projector parameters in the set 'camera_setup_model'
H_ERR_SLM_WRONGCTYPE	3964	Camera type not supported by the structured light model
H_ERR_SLM_WRONGPTYPE	3965	Projector type not supported by the structured light model
H_ERR_SLM_NO_CSM	3966	The structured light model does not contain a camera setup model
H_ERR_SLM_NO_VERT	3967	'pattern_orientation' does not contain vertical orientation
H_ERR_SLM_NOT_DEC_REC	3968	'reconstruct_surface_structured_light' can only be called after 'decode_structured_light_pattern'
H_ERR_SLM_WCAMSIZE	3969	Size of correspondence image does not match the camera parameters in the set 'camera_setup_model'
H_ERR_DBOIT	4050	Image data management: object is an object tuple
H_ERR_DBOC	4051	Image data management: object has been deleted already
H_ERR_DBWOID	4052	Image data management: wrong object-ID
H_ERR_DBTC	4053	Image data management: object tuple has been deleted already
H_ERR_DBWTID	4054	Image data management: wrong object tuple-ID
H_ERR_DBTIO	4055	Image data management: object tuple is an object
H_ERR_DBIDNULL	4056	Image data management: object-ID is NULL (0)
H_ERR_WDBID	4057	Image data management: object-ID outside the valid range
H_ERR_DBIC	4058	Image data management: access to deleted image
H_ERR_DBWIID	4059	Image data management: access to image with wrong key
H_ERR_DBRC	4060	Image data management: access to deleted region
H_ERR_DBWRID	4061	Image data management: access to region with wrong key
H_ERR_WCHAN	4062	Image data management: wrong value for image channel
H_ERR_DBITL	4063	Image data management: index too big
H_ERR_DBIUNDEF	4064	Image data management: index not defined
H_ERR_NO_OPENCL	4100	No OpenCL available
H_ERR_OPENCL_ERROR	4101	OpenCL Error occurred
H_ERR_NO_COMPUTE_DEVICES	4102	No compute device available
H_ERR_NO_DEVICE_IMPL	4103	No device implementation for this parameter
H_ERR_OUT_OF_DEVICE_MEM	4104	Out of compute device memory
H_ERR_INVALID_SHAPE	4105	Invalid work group shape
H_ERR_INVALID_DEVICE	4106	Invalid compute device
H_ERR_CUDA_ERROR	4200	CUDA Error occurred
H_ERR_CUDNN_ERROR	4201	cuDNN Error occurred
H_ERR_CUBLAS_ERROR	4202	cuBLAS Error occurred
H_ERR_BATCH_SIZE_NOT_SUPPORTED	4203	Setting the batch size on CPUs is not supported. For more information, please read the documentation of the parameter
H_ERR_CUDA_NOT_AVAILABLE	4204	CUDA implementations are not available
H_ERR_CUDNN_UNSUPPORTED_VERSION	4205	Unsupported version of cuDNN
H_ERR_CUDNN_FEATURE_NOT_SUPPORTED	4206	Current configuration requests a feature not supported by cuDNN. Use different input size and/or the batch size!
H_ERR_CUDA_DRIVER_VERSION	4207	Your graphics driver does not support the required CUDA version. Please update your graphics driver!
H_ERR_TRAINING_UNSUPPORTED	4301	Training is unsupported with the selected runtime.
H_ERR_CPU_INFERENCE_NOT_AVAILABLE	4302	CPU based inference is not supported on this platform
H_ERR_DNNL_ERROR	4303	Error occurred in DNNL library
H_ERR_HAI2_ERROR	4320	Error occurred in an AI Accelerator Interface
H_ERR_HAI2_INVALID_PARAM	4321	Invalid parameter for AI Accelerator Interface
H_ERR_ACL_ERROR	4400	Error occurred in ACL library
H_ERR_VISUALIZATION	4500	Internal visualization error
H_ERR_COLOR_TYPE_UNEXP	4501	Unexpected color type
H_ERR_NUM_COLOR_EXCEEDED	4502	Number of color settings exceeded
H_ERR_WSCN	5100	Wrong (logical) window number

Error Name	Code	Description
H_ERR_DSCO	5101	Error while opening the window
H_ERR_WWC	5102	Wrong window coordinates
H_ERR_NWA	5103	It is not possible to open another window
H_ERR_DNA	5104	Device or operator not available
H_ERR_UCOL	5105	Unknown color
H_ERR_NWO	5106	No window has been opened for desired action
H_ERR_WFM	5107	Wrong filling mode for regions (fill or margin)
H_ERR_WGV	5108	Wrong gray value (0..255)
H_ERR_WPV	5109	Wrong pixel value (use value of get_pixel(P) only)
H_ERR_WLW	5110	Wrong line width (see: query_line_width(Min,Max))
H_ERR_WCUR	5111	Wrong name of cursor
H_ERR_WLUT	5112	Wrong color table (see: query_lut(Name)
H_ERR_WDM	5113	Wrong representation mode (see: query_insert(Mode))
H_ERR_WRCO	5114	Wrong representation color (see: query_color(List))
H_ERR_WRDM	5115	Wrong dither matrix (binary image representation)
H_ERR_WRIT	5116	Wrong image transformation (name or image size)
H_ERR_IPIT	5117	Unsuitable image type for image transformation
H_ERR_WRZS	5118	Wrong zooming factor for image transformation
H_ERR_WRDS	5119	Wrong representation mode
H_ERR_WRDV	5120	Wrong code of device
H_ERR_WWINF	5121	Wrong number for father window
H_ERR_WDEXT	5122	Wrong window size
H_ERR_WWT	5123	Wrong window type
H_ERR_WND	5124	No current window has been set
H_ERR_WRGB	5125	Wrong color combination or range (RGB)
H_ERR_WPNS	5126	Wrong number of pixels set
H_ERR_WCM	5127	Wrong value for comprise (object or image)
H_ERR_FNA	5128	set_fix with 1/4 image levels and static not valid
H_ERR_LNFS	5129	set_lut not valid in child windows
H_ERR_LOFL	5130	Number of concurrent used color tables is too big
H_ERR_WIDT	5131	Wrong device for window dump
H_ERR_WWDS	5132	Wrong window size for window dump
H_ERR_NDVS	5133	System variable DISPLAY (setenv) not defined
H_ERR_WBW	5134	Wrong thickness for window margin
H_ERR_WDVS	5135	System variable DISPLAY has been set wrong (<host>:0.0)
H_ERR_TMF	5136	Too many fonts loaded
H_ERR_WFN	5137	Wrong font name
H_ERR_WCP	5138	No valid cursor position
H_ERR_NTW	5139	Window is not a textual window
H_ERR_NPW	5140	Window is not a image window
H_ERR_STL	5141	String too long or too high
H_ERR_NSS	5142	Too little space in the window rightwards
H_ERR_NMS	5143	Window is not suitable for the mouse
H_ERR_DWNA	5144	Here Windows on a equal machine is permitted only
H_ERR_WOM	5145	Wrong mode while opening a window
H_ERR_WWM	5146	Wrong window mode for operation
H_ERR_LUTF	5147	Operation not possible with fixed pixel
H_ERR_LUTN8	5148	Color tables for 8 image levels only
H_ERR_WTCM	5149	Wrong mode for pseudo real colors
H_ERR_WIFTL	5150	Wrong pixel value for LUT
H_ERR_WSOI	5151	Wrong image size for pseudo real colors
H_ERR_HRLUT	5152	Error in procedure HRLUT
H_ERR_WPFSL	5153	Wrong number of entries in color table for set_lut
H_ERR_WPVS	5154	Wrong values for image area
H_ERR_WLPN	5155	Wrong line pattern
H_ERR_WLPL	5156	Wrong number of parameters for line pattern

Error Name	Code	Description
H_ERR_WNOC	5157	Wrong number of colors
H_ERR_WPST	5158	Wrong value for mode of area creation (0,1,2)
H_ERR_SWNA	5159	Spy window is not set (set_spy)
H_ERR_NSFO	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_LUTO	5170	LUT of this display is full
H_ERR_WCC	5171	Internal error: wrong color code
H_ERR_WWATTRT	5172	Wrong type for window attribute
H_ERR_WWATTRN	5173	Wrong name for window attribute
H_ERR_WRSPART	5174	Negative height of area (or 0)
H_ERR_WCSPART	5175	Negative width of area (or 0)
H_ERR_WNCV	5176	Window not completely visible
H_ERR_FONT_NA	5177	Font not allowed for this operation
H_ERR_WDIFFTH	5178	Operation not possible (window was created in different thread)
H_ERR_DEPTH_NOT_STORED	5179	Depth was not stored with window
H_ERR_CHA3	5180	Internal error: only RGB-Mode
H_ERR_NMWA	5181	No more (image-)windows available
H_ERR_INDEX_NOT_STORED	5182	Object index was not stored with window
H_ERR_PRIM_NO_POINTS	5183	Operator does not support primitives without point coordinates
H_ERR_REMOTE_DESKTOP_SIZE	5184	Maximum image size for Windows Remote Desktop exceeded
H_ERR_NOGL	5185	No OpenGL support available
H_ERR_NODEPTH	5186	No depth information available
H_ERR_OGL_ERROR	5187	OpenGL error occurred
H_ERR_UNSUPPORTED_FBO	5188	Required framebuffer object is unsupported
H_ERR_OGL_HSR_NOT_SUPPORTED	5189	OpenGL accelerated hidden surface removal not supported on this machine
H_ERR_WP_IWP	5190	Invalid window parameter
H_ERR_WP_IWPV	5191	Invalid value for window parameter
H_ERR_UMOD	5192	Unknown mode
H_ERR_ATTIMG	5193	No image has been previously attached to window
H_ERR_OBJ_ATTACHED	5194	The drawing object has already been attached to another window
H_ERR_NVG_WM	5195	Invalid value of navigation mode
H_ERR_FINTERN	5196	Internal file error
H_ERR_FS	5197	Error while file synchronization
H_ERR_FISR	5198	Insufficient rights on file
H_ERR_BFD	5199	Bad file descriptor
H_ERR_FNF	5200	File not found
H_ERR_DWI	5201	Error while writing image data (sufficient memory?)
H_ERR_DWID	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_DRID1	5205	Error while reading image descriptor: file too small
H_ERR_DIMMAT	5206	Image matrices are different
H_ERR_HNF	5207	Help file not found (setenv HALCONROOT <HALCON-Homedirectory>)
H_ERR_XNF	5208	Help index not found (setenv HALCONROOT <HALCON-Homedirectory>)

Error Name	Code	Description
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
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_WFMO	5216	Wrong file mode
H_ERR_WPTY	5217	Wrong type for pixel (e.g. byte)
H_ERR_WIW	5218	Wrong image width (too big?)
H_ERR_WIH	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_WDPI	5222	Wrong value for resolution (dpi)
H_ERR_WNOW	5223	Wrong output image size (width)
H_ERR_WNOH	5224	Wrong output image size (height)
H_ERR_WNFP	5225	Wrong number of parameter values: format description
H_ERR_WPNA	5226	Wrong parameter name for operator
H_ERR_WSNA	5227	Wrong slot name for parameter
H_ERR_NPCF	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_NFWKEF	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 not a TIFF file
H_ERR_WFF	5238	Wrong file format
H_ERR_NOGPPROC	5239	gnuplot could not be started
H_ERR_NOGPFILE	5240	Output file for gnuplot could not be opened
H_ERR_NOGPOUT	5241	Not a valid gnuplot output stream
H_ERR_NOPNM	5242	No PNM format
H_ERR_ICODB	5243	Inconsistent or old help file ($HALCONROOT/help)
H_ERR_INVAL_FILE_ENC	5244	Invalid text file encoding
H_ERR_FNO	5245	File not open
H_ERR_NO_FILES	5246	No files in use so far (none opened)
H_ERR_NORFILE	5247	Invalid file format for regions
H_ERR_RDTB	5248	Error while reading region data: Format of region too big
H_ERR_BINFILE_ENC	5249	Setting the encoding is not supported for binary files
H_ERR_EDRF	5250	Error while reading from file
H_ERR_SNO	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_CNGSA	5255	Could not get serial port attributes
H_ERR_CNSSA	5256	Could not set serial port attributes
H_ERR_WRSBR	5257	Wrong baud rate for serial connection
H_ERR_WRSDB	5258	Wrong number of data bits for serial connection
H_ERR_WRSFC	5259	Wrong flow control for serial connection
H_ERR_CNFS	5260	Could not flush serial port
H_ERR_EDWS	5261	Error during write to serial port
H_ERR_EDRS	5262	Error during read from serial port

Error Name	Code	Description
H_ERR_REG_NOSITEM	5270	Serialized item does not contain valid regions
H_ERR_REG_WRVERS	5271	The version of the regions is not supported
H_ERR_IMG_NOSITEM	5272	Serialized item does not contain valid images
H_ERR_IMG_WRVERS	5273	The version of the images is not supported
H_ERR_XLD_NOSITEM	5274	Serialized item does not contain valid XLD objects
H_ERR_XLD_WRVERS	5275	The version of the XLD objects is not supported
H_ERR_OBJ_NOSITEM	5276	Serialized item does not contain valid objects
H_ERR_OBJ_WRVERS	5277	The version of the objects is not supported
H_ERR_OBJ_UNEXPECTED	5279	Unexpected object detected
H_ERR_FNOTF	5280	File has not been opened in text format
H_ERR_FNOBF	5281	File has not been opened in binary file format
H_ERR_DIRCR	5282	Cannot create directory
H_ERR_DIRRM	5283	Cannot remove directory
H_ERR_GETCWD	5284	Cannot determine current directory
H_ERR_SETCWD	5285	Cannot set current directory
H_ERR_XINIT	5286	Missing call to XInitThreads: multithreading support for X11 required
H_ERR_NFS	5300	No image acquisition device opened
H_ERR_FGWC	5301	Image acquisition: wrong color depth
H_ERR_FGWD	5302	Image acquisition: wrong device
H_ERR_FGVF	5303	Image acquisition: determination of video format not possible
H_ERR_FGNV	5304	Image acquisition: no video signal
H_ERR_UFG	5305	Unknown image acquisition device
H_ERR_FGF	5306	Image acquisition: failed grabbing of an image
H_ERR_FGWR	5307	Image acquisition: wrong resolution chosen
H_ERR_FGWP	5308	Image acquisition: wrong image part chosen
H_ERR_FGWPR	5309	Image acquisition: wrong pixel ratio chosen
H_ERR_FGWH	5310	Image acquisition: handle not valid
H_ERR_FGCL	5311	Image acquisition: instance not valid (already closed?)
H_ERR_FGNI	5312	Image acquisition: device cannot be initialized
H_ERR_FGET	5313	Image acquisition: external triggering not supported
H_ERR_FGLI	5314	Image acquisition: wrong camera input line (multiplex)
H_ERR_FGCS	5315	Image acquisition: wrong color space
H_ERR_FGPT	5316	Image acquisition: wrong port
H_ERR_FGCT	5317	Image acquisition: wrong camera type
H_ERR_FGTM	5318	Image acquisition: maximum number of acquisition device classes exceeded
H_ERR_FGDV	5319	Image acquisition: device busy
H_ERR_FGASYNC	5320	Image acquisition: asynchronous grab not supported
H_ERR_FGPARAM	5321	Image acquisition: unsupported parameter
H_ERR_FGTIMEOUT	5322	Image acquisition: timeout
H_ERR_FGGAIN	5323	Image acquisition: invalid gain
H_ERR_FGFIELD	5324	Image acquisition: invalid field
H_ERR_FGPART	5325	Image acquisition: invalid parameter type
H_ERR_FGPARV	5326	Image acquisition: invalid parameter value
H_ERR_FGFNS	5327	Image acquisition: function not supported
H_ERR_FGIVERS	5328	Image acquisition: incompatible interface version
H_ERR_FGSETPAR	5329	Image acquisition: could not set parameter value
H_ERR_FGGETPAR	5330	Image acquisition: could not query parameter setting
H_ERR_FGPARNA	5331	Image acquisition: parameter not available in current configuration
H_ERR_FGCLOSE	5332	Image acquisition: device could not be closed properly
H_ERR_FGCAMFILE	5333	Image acquisition: camera configuration file could not be opened
H_ERR_FGCALLBACK	5334	Image acquisition: callback type not supported
H_ERR_FGDEVLOST	5335	Image acquisition: device lost
H_ERR_FGABORTED	5336	Image acquisition: aborted

Error Name	Code	Description
H_ERR_IOTIMEOUT	5350	IO: timeout
H_ERR_IOIVERS	5351	IO: incompatible interface version
H_ERR_IOWH	5352	IO: handle not valid
H_ERR_IODBUSY	5353	IO: device busy
H_ERR_IOIAR	5354	IO: insufficient user rights
H_ERR_IONF	5355	IO: device or channel not found
H_ERR_IOPART	5356	IO: invalid parameter type
H_ERR_IOPARV	5357	IO: invalid parameter value
H_ERR_IOPARNUM	5358	IO: invalid number of parameters
H_ERR_IOPARAM	5359	IO: unsupported parameter
H_ERR_IOPARNA	5360	IO: parameter not available in current configuration
H_ERR_IOFNS	5361	IO: function not supported
H_ERR_IOME	5362	IO: maximum number of IO devices or channels exceeded
H_ERR_IODNA	5363	IO: driver of IO device not available
H_ERR_IOABORTED	5364	IO: aborted
H_ERR_IODATT	5365	IO: invalid data type
H_ERR_IODEVLOST	5366	IO: device lost
H_ERR_IOSETPAR	5367	IO: could not set parameter value
H_ERR_IOGETPAR	5368	IO: could not query parameter value
H_ERR_IOCLOSE	5369	IO: device could not be closed properly
H_ERR_JXR_UNSUPPORTED_FORMAT	5400	Image type is not supported
H_ERR_JXR_INVALID_PIXEL_FORMAT	5401	Invalid pixel format
H_ERR_JXR_INTERNAL_ERROR	5402	Internal JPEG-XR error
H_ERR_JXR_FORMAT_SYNTAX_ERROR	5403	Invalid format string
H_ERR_JXR_TOO_MANY_CHANNELS	5404	Maximum number of channels exceeded
H_ERR_JXR_EC_ERROR	5405	Unspecified error in JPEG-XR library
H_ERR_JXR_EC_BADMAGIC	5406	Bad magic number in JPEG-XR library
H_ERR_JXR_EC_FEATURE_NOT_IMPLEMENTED	5407	Feature not implemented in JPEG-XR library
H_ERR_JXR_EC_IO	5408	File read/write error in JPEG-XR library
H_ERR_JXR_EC_BADFORMAT	5409	Invalid file format in JPEG-XR library
H_ERR_LIB_FILE_CLOSE	5500	Error while closing the image file
H_ERR_LIB_FILE_OPEN	5501	Error while opening the image file
H_ERR_LIB_UNEXPECTED_EOF	5502	Premature end of the image file
H_ERR_IDTL	5503	Image dimensions too large for this file format
H_ERR_ITLHV	5504	Image too large for this HALCON version
H_ERR_TMIO	5505	Too many iconic objects for this file format
H_ERR_FILE_FORMAT_UNSUPPORTED	5506	File format is unsupported
H_ERR_INCONSISTENT_DIMENSIONS	5507	All channels must have equal dimensions
H_ERR_PCX_NO_PCX_FILE	5510	File is no PCX-File
H_ERR_PCX_UNKNOWN_ENCODING	5511	PCX: unknown encoding
H_ERR_PCX_MORE_THAN_4_PLANES	5512	PCX: More than 4 image plains
H_ERR_PCX_COLORMAP_SIGNATURE	5513	PCX: Wrong magic in color table
H_ERR_PCX_REPEAT_COUNT_SPANS	5514	PCX: Wrong number of bytes in span
H_ERR_PCX_TOO_MUCH_BITS_PIXEL	5515	PCX: Wrong number of bits/pixels
H_ERR_PCX_PACKED_PIXELS	5516	PCX: Wrong number of plains
H_ERR_GIF_NO_GIF_PICTURE	5520	File is no GIF-File
H_ERR_GIF_BAD_VERSION	5521	GIF: Wrong version (not 87a/89a)
H_ERR_GIF_SCREEN_DESCRIPTOR	5522	GIF: Wrong descriptor
H_ERR_GIF_COLORMAP	5523	GIF: Wrong color table
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_TABL_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

Error Name	Code	Description
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_TOO_MANY_COLORS	5552	TIFF: Too many colors
H_ERR_TIF_BAD_PHOTOMETRIC	5553	TIFF: Wrong photometric interpretation
H_ERR_TIF_PHOTOMETRIC_DEPTH	5554	TIFF: Wrong photometric depth
H_ERR_TIF_NO_REGION	5555	TIFF: Image is no binary file
H_ERR_TIF_UNSUPPORTED_FORMAT	5556	TIFF: Image format not supported by HALCON
H_ERR_TIF_BAD_SPECIFICATION	5557	TIFF: Wrong specification of the TIFF file format
H_ERR_TIF_FILE_CORRUPT	5558	TIFF: TIFF file is corrupt
H_ERR_TIF_TAG_UNDEFINED	5559	TIFF: A required TIFF tag is missing in the TIFF 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_BMP_UNKNOWN_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 implemented 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_PNG_NO_PNG_FILE	5580	PNG: File is not a PNG file
H_ERR_PNG_UNKNOWN_INTERLACE_TYPE	5581	PNG: Unknown interlace type
H_ERR_PNG_UNSUPPORTED_COLOR_TYPE	5582	PNG: Unsupported color type
H_ERR_PNG_NO_REGION	5583	PNG: Image is no binary file
H_ERR_PNG_SIZE_TOO_BIG	5584	PNG: Maximal image size exceeded
H_ERR_TIF_TAG_ACCESS	5587	TIFF: Error accessing a TIFF tag
H_ERR_TIF_TAG_DATATYPE	5588	TIFF: Wrong datatype for specified TIFF tag
H_ERR_TIF_TAG_UNSUPPORTED	5589	TIFF: Requested TIFF tag not supported by HALCON
H_ERR_JP2_CORRUPT	5590	JPEG-2000: File corrupt
H_ERR_JP2_PREC_TOO_HIGH	5591	JPEG-2000: Image has more than 28 significant bits
H_ERR_JP2_ENCODING_ERROR	5592	JPEG-2000: Error while encoding
H_ERR_JP2_SIZE_TOO_BIG	5593	JPEG-2000: Maximal image size exceeded
H_ERR_HOBJ_NOT_ONLY_IMAGES	5599	HOBJ: File does not contain only images
H_ERR_SOCKET_BLOCK	5600	Socket can not be set to block
H_ERR_SOCKET_UNBLOCK	5601	Socket can not be set to unblock
H_ERR_SOCKET_NO_CPAR	5602	Received data is no tuple

Error Name	Code	Description
H_ERR_SOCKET_NO_IMAGE	5603	Received data is no image
H_ERR_SOCKET_NO_RL	5604	Received data is no region
H_ERR_SOCKET_NO_XLD	5605	Received data is no xld object
H_ERR_SOCKET_READ_DATA_FAILED	5606	Error while reading from socket
H_ERR_SOCKET_WRITE_DATA_FAILED	5607	Error while writing to socket
H_ERR_SOCKET_WRONG_BYTE_NUMBER	5608	Illegal number of bytes with get_rl
H_ERR_SOCKET_BUFFER_OVERFLOW	5609	Buffer overflow in read_data
H_ERR_SOCKET_CANT_ASSIGN_FD	5610	Socket can not be created
H_ERR_SOCKET_CANT_BIND	5611	Bind on socket failed
H_ERR_SOCKET_CANT_GET_PORTNUMBER	5612	Socket information is not available
H_ERR_SOCKET_CANT_LISTEN	5613	Socket cannot listen for incoming connections
H_ERR_SOCKET_CANT_ACCEPT	5614	Connection could not be accepted
H_ERR_SOCKET_CANT_CONNECT	5615	Connection request failed
H_ERR_SOCKET_GETHOSTBYNAME	5616	Hostname could not be resolved
H_ERR_SOCKET_ILLEGAL_TUPLE_TYPE	5618	Unknown tuple type on socket
H_ERR_SOCKET_TIMEOUT	5619	Timeout occurred on socket
H_ERR_SOCKET_NA	5620	No more sockets available
H_ERR_SOCKET_NI	5621	Socket is not initialized
H_ERR_SOCKET_OOR	5622	Invalid socket
H_ERR_SOCKET_IS	5623	Socket is NULL
H_ERR_SOCKET_DATA_TOO_LARGE	5624	Received data type is too large
H_ERR_SOCKET_WRONG_TYPE	5625	Wrong socket protocol
H_ERR_SOCKET_NO_PACKED_DATA	5626	Received data does not contain packed data
H_ERR_SOCKET_PARAM_FAILED	5627	Error when handling the parameter
H_ERR_SOCKET_FORMAT_MISMATCH	5628	Format specification does not match the data
H_ERR_SOCKET_INVALID_FORMAT	5629	Invalid format specification
H_ERR_SOCKET_NO_SERIALIZED_ITEM	5630	Received data is no serialized item
H_ERR_SOCKET_TLS_CONTEXT	5631	Could not initialize TLS library
H_ERR_SOCKET_TLS_CERT_KEY	5632	Invalid TLS Certificate or private Key
H_ERR_SOCKET_TLS_HANDSHAKE	5633	TLS handshake failed
H_ERR_XLD_DATA_TOO_LARGE	5678	XLD object data can only be read by HALCON XL
H_ERR_ARCINFO_TOO_MANY_XLDS	5700	Too many contours/polygons for this file format
H_ERR_QUAT_WRONG_VERSION	5750	The version of the quaternion is not supported
H_ERR_QUAT_NOSITEM	5751	Serialized item does not contain a valid quaternion
H_ERR_HOM_MAT2D_WRONG_VERSION	5752	The version of the homogeneous matrix is not supported
H_ERR_HOM_MAT2D_NOSITEM	5753	Serialized item does not contain a valid homogeneous matrix
H_ERR_HOM_MAT3D_WRONG_VERSION	5754	The version of the homogeneous 3D matrix is not supported
H_ERR_HOM_MAT3D_NOSITEM	5755	Serialized item does not contain a valid homogeneous 3D matrix
H_ERR_TUPLE_WRONG_VERSION	5756	The version of the tuple is not supported
H_ERR_TUPLE_NOSITEM	5757	Serialized item does not contain a valid tuple
H_ERR_TUPLE_DTLFTHV	5758	Number too big for a string to number conversion (overflow)
H_ERR_POSE_WRONG_VERSION	5759	The version of the camera parameters (pose) is not supported
H_ERR_POSE_NOSITEM	5760	Serialized item does not contain valid camera parameters (pose)
H_ERR_CAM_PAR_WRONG_VERSION	5761	The version of the internal camera parameters is not supported
H_ERR_CAM_PAR_NOSITEM	5762	Serialized item does not contain valid internal camera parameters
H_ERR_DUAL_QUAT_WRONG_VERSION	5763	The version of the dual quaternion is not supported
H_ERR_DUAL_QUAT_NOSITEM	5764	Serialized item does not contain a valid dual quaternion
H_ERR_NP	6000	Access to undefined memory area
H_ERR_MEM	6001	Not enough memory available
H_ERR_ICM	6002	Memory partition on heap has been overwritten
H_ERR_WMS	6003	HAlloc: 0 bytes requested
H_ERR_NOTMP	6004	Tmp-memory management: Call freeing memory although nothing had been allocated
H_ERR_TMPNULL	6005	Tmp-memory management: Null pointer while freeing

Error Name	Code	Description
H_ERR_CNFMEM	6006	Tmp-memory management: could not find memory element
H_ERR_WMT	6007	Memory management: wrong memory type allocated
H_ERR_MEM_VID	6021	Not enough video memory available
H_ERR_IAD	6040	System parameter for memory-allocation inconsistent
H_ERR_NRA	6041	No memory block allocated at last
H_ERR_INVALID_ALIGN	6042	Invalid alignment
H_ERR_NULL_PTR	6043	Function was given a NULL pointer as input
H_ERR_CP_FAILED	6500	Process creation failed
H_ERR_WOCPI	7000	Wrong index for output control parameter
H_ERR_WOCPVN	7001	Wrong number of values: output control parameter (see: HPut*Par)
H_ERR_WOCPT	7002	Wrong type: output control parameter (see: HPut*Par)
H_ERR_WKT	7003	Wrong data type for object key (input objects)
H_ERR_IOOR	7004	Range for integer had been passed
H_ERR_IHV	7005	Inconsistent HALCON version
H_ERR_NISS	7006	Not enough memory for strings allocated
H_ERR_PROC_NULL	7007	Internal error: Proc is NULL
H_ERR_UNKN	7105	Unknown symbolic object key (input objects)
H_ERR_WOON	7200	Wrong number of output object parameter
H_ERR_OTSE	7400	System error: output type <string> expected
H_ERR_OTLE	7401	System error: output type <long> expected
H_ERR_OTFE	7402	System error: output type <float> expected
H_ERR_OPINP	7403	Object parameter is a zero pointer ('_' not allowed)
H_ERR_TWC	7404	Tuple had been deleted; values are not valid any more
H_ERR_CNN_DATA	7701	CNN: Internal data error
H_ERR_CNN_MEM	7702	CNN: Invalid memory type
H_ERR_CNN_IO_INVALID	7703	CNN: Invalid data serialization
H_ERR_CNN_IMPL_NOT_AVAILABLE	7704	CNN: Implementation not available
H_ERR_CNN_NUM_INPUTS_INVALID	7705	Wrong number of input data
H_ERR_CNN_IMPL_INVALID	7706	CNN: Invalid implementation type selected
H_ERR_CNN_TRAINING_NOT_SUP	7707	CNN: Training is not supported by the current environment
H_ERR_CNN_GPU_REQUIRED	7708	For this operation a GPU with certain minimal hardware specifications is required (See installation guide)
H_ERR_CNN_CUDA_LIBS_MISSING	7709	For this operation a suitable GPU and the corresponding CUDA libraries need to be available (See installation guide)
H_ERR_OCR_CNN_RE	7710	OCR File: Error while reading classifier
H_ERR_OCR_CNN_WGPN	7711	Wrong generic parameter name
H_ERR_OCR_CNN_EXCLUSIV_PARAM	7712	One of the generic parameters returns several values and has to be used exclusively
H_ERR_CNN_WGPN	7713	Wrong generic parameter name
H_ERR_CNN_INVALID_LABELS	7714	Invalid label
H_ERR_OCR_CNN_FILE_WRONG_VERSION	7715	OCR File: Wrong file version
H_ERR_CNN_MULTIPLE_CLASSES	7716	Invalid classes: At least one class occurs twice
H_ERR_CNN_CUBLAS_LIBS_MISSING	7717	For this operation the cuBLAS library needs to be available (See installation guide)
H_ERR_CNN_CUDNN_LIBS_MISSING	7718	For this operation the cuDNN library needs to be available (See installation guide)
H_ERR_OCR_FNF_FIND_TEXT_SUPPORT	7719	File 'find_text_support.hotc' not found. Please place this file in the ocr subdirectory of the root directory of the HALCON installation or in the current working directory
H_ERR_CNN_TRAINING_FAILED	7720	Training step failed. This might be caused by unsuitable hyperparameters
H_ERR_CNN_NO_PRETRAINED_WEIGHTS	7721	Weights of the pretrained network have been overwritten and cannot be restored
H_ERR_CNN_INVALID_INPUT_SIZE	7722	One or more input dimensions are too small for the network
H_ERR_CNN_RESULT_NOT_AVAILABLE	7723	The requested results are not available for this network
H_ERR_CNN_INVALID_INPUT_DEPTH	7724	The number of channels of the input image must be either 1 or 3

Error Name	Code	Description
H_ERR_CNN_DEPTH_NOT_AVAILABLE	7725	The number of channels of the input image cannot be set
H_ERR_CNN_INVALID_BATCH_SIZE	7726	Batch size smaller than subbatch size
H_ERR_CNN_INVALID_PARAM_SPEC	7727	Invalid parameter specification
H_ERR_CNN_EXCEEDS_MAX_MEM	7728	Internal memory management limit exceeded
H_ERR_CNN_BATCH_SIZE_OVERFLOW	7729	The required memory exceeds internal limits. This can be caused by very high values of batch size
H_ERR_CNN_INVALID_IMAGE_SIZE	7730	The size of input image for the model is invalid. Width and height must be divisible through the level of the last backbone layer
H_ERR_CNN_INVALID_LAYER_PARAM_VALUE	7731	The parameter value is invalid for this graph node.
H_ERR_CNN_INVALID_LAYER_PARAM_NUM	7732	The number of parameters is invalid for this graph node.
H_ERR_CNN_INVALID_LAYER_PARAM_TYPE	7733	The parameter type is invalid for this graph node.
H_ERR_CNN_NUM_OUTPUTS_INVALID	7734	Wrong number of output data
H_ERR_CNN_INVALID_SHAPE	7735	CNN: Invalid shape of input
H_ERR_CNN_INVALID_INPUT_DATA	7736	CNN: Invalid input data
H_ERR_CNN_CUDNN_CTC_LOSS_BUGGY	7737	CNN: For variable input lengths the ctc loss layer only computes correct gradients if the used cuDNN version is >= 7.6.3. Please upgrade cuDNN or do not use variable input lengths.
H_ERR_CNN_INVALID_PADDING	7738	CNN: Invalid padding
H_ERR_CNN_IO_INVALID_LAYER_TYPE	7740	CNN: Invalid layer type serialization
H_ERR_CNN_INFERENCE_FAILED	7741	CNN: Inference failed. This might be caused by bad weights created by unsuitable hyperparameters during the training
H_ERR_CNN_RUNTIME_FAILED	7742	CNN: Runtime unsupported on this machine
H_ERR_GRAPH_INTERNAL	7751	Graph: Internal error
H_ERR_GRAPH_IO_INVALID	7752	Graph: Invalid data serialization
H_ERR_GRAPH_INVALID_INDEX	7753	Graph: Invalid index
H_ERR_CNNGRAPH_INTERNAL	7760	CNN graph: Internal error
H_ERR_CNNGRAPH_IO_INVALID	7761	CNN graph: Invalid data serialization
H_ERR_CNNGRAPH_LAYER_INVALID	7762	CNN graph: Invalid layer specification
H_ERR_CNNGRAPH_NOINIT	7763	CNN graph: Graph not initialized properly
H_ERR_CNNGRAPH_INVALID_MEM	7764	CNN graph: Invalid memory type
H_ERR_CNNGRAPH_INVALID_NUML	7765	CNN graph: Invalid number of layers
H_ERR_CNNGRAPH_INVALID_IDX	7766	CNN graph: Invalid index
H_ERR_CNNGRAPH_SPEC_STATUS	7767	CNN graph: Invalid specification state
H_ERR_CNNGRAPH_NOCHANGE	7768	CNN graph: Graph is not allowed to be changed after initialization
H_ERR_CNNGRAPH_PREPROC	7769	CNN graph: Missing preprocessing
H_ERR_CNNGRAPH_DEGREE	7770	CNN graph: Invalid vertex degree
H_ERR_CNNGRAPH_OUTSHAPE	7771	CNN graph: Invalid output shape
H_ERR_CNNGRAPH_SPEC	7772	CNN graph: Invalid specification
H_ERR_CNNGRAPH_DEF	7773	CNN graph: Invalid graph definition
H_ERR_CNNGRAPH_NO_CLASS_CHANGE	7774	CNN-Graph: Architecture not suitable for the adaption of the number of output classes
H_ERR_CNNGRAPH_NO_IMAGE_RESIZE	7775	CNN-Graph: Architecture not suitable for the adaption of the image size
H_ERR_CNNGRAPH_AUX_INDEX_OOB	7776	CNN-Graph: Auxiliary output index out of bounds.
H_ERR_CNNGRAPH_AUX_SPEC	7777	CNN-Graph: Invalid graph definition. Probably the auxiliary outputs of a layer have not been connected with corresponding aux selection layers (SelectAux) or at least one aux output has not been specified during model creation (create_dl_model call).
H_ERR_CNNGRAPH_LAYER_UNSUPPORTED	7778	CNN-Graph: At least one layer is unsupported for the selected runtime or operation
H_ERR_DL_INTERNAL	7779	DL: Internal Error
H_ERR_DL_FILE_READ	7780	DL: Error while reading file
H_ERR_DL_FILE_WRITE	7781	DL: Error while writing file
H_ERR_DL_FILE_WRONG_VERSION	7782	DL: Wrong file version
H_ERR_DL_INPUTS_MISSING	7783	DL: Inputs missing in input dict
H_ERR_DL_INPUT_WRONG_BS	7784	DL: Inputs have incorrect batch size

Error Name	Code	Description
H_ERR_DL_INVALID_NAME	7785	DL: Invalid layer name
H_ERR_DL_DUPLICATE_NAME	7786	DL: Duplicate layer name
H_ERR_DL_INVALID_OUTPUT	7787	DL: Invalid output layer
H_ERR_DL_PARAM_NOT_AVAILABLE	7788	DL: Parameter is not available for this model
H_ERR_DL_INPUT_WRONG_LENGTH	7789	DL: Tuple inputs have incorrect length
H_ERR_DL_INPUT_WRONG_TYPE	7790	DL: Tuple inputs have incorrect type
H_ERR_DL_INPUT_WRONG_VALUES	7791	DL: Some inputs have incorrect values
H_ERR_DL_CLASS_IDS_NOT_UNIQUE	7792	DL: Some class ids are not unique
H_ERR_DL_CLASS_IDS_INVALID	7793	DL: Some class ids have incorrect values.The expected value range is [0, 65534]
H_ERR_DL_CLASS_IDS_INVALID_CONV	7794	DL: Input data of class id conversion is invalid
H_ERR_DL_TYPE_ALREADY_DEFINED	7795	DL: Type is already defined so it cannot be changed
H_ERR_DL_NO_INFERENCE_INPUTS	7796	DL: Cannot identify inference inputs
H_ERR_DL_CLASS_IDS_INVALID_OVERLAP	7797	DL: Class ids must not overlap with ignore class ids
H_ERR_DL_WRONG_OUTPUT_LAYER_NUM	7798	DL: Tuple of output layers has incorrect length
H_ERR_DL_WRONG_BS_MULTIPLIER	7799	DL: Batch size multiplier needs to be greater than 0
H_ERR_DL_INPUT_WRONG_BS_WITH_MULTIPLIER	7800	DL: Inputs have incorrect batch size. The number of needed inputs is defined by batch_size * batch_size_multiplier
H_ERR_DL_READ_ONNX	7801	Error occurred while reading an ONNX model
H_ERR_DL_CLASS_IDS_MISSING	7802	DL: Model has no class ids
H_ERR_DL_WRITE_ONNX	7803	Error occurred while writing an ONNX model
H_ERR_DL_ONNX_LOADER	7804	Libprotobuf for ONNX could not be loaded
H_ERR_DL_FPN_SCALES	7810	DL: min_level or max_level not compatible with backbone
H_ERR_DL_FPN_INVALID_BACKBONE	7811	DL: Backbone unusable for FPN creation
H_ERR_DL_FPN_INVALID_FEATURE_MAP_SIZE	7812	DL: Image size not compatible with backbone
H_ERR_DL_FPN_INVALID_LEVELS	7813	DL: Invalid FPN-levels.
H_ERR_DL_ANCHOR	7820	DL: Internal error with anchors
H_ERR_DL_DETECTOR_INVALID_PARAM	7821	DL: Invalid detector parameter
H_ERR_DL_DETECTOR_INVALID_PARAM_VALUE	7822	DL: Invalid detector parameter value
H_ERR_DL_DETECTOR_INVALID_DOCKING_LAYER	7823	DL: Invalid backbone docking layer names
H_ERR_DL_DETECTOR_INVALID_INSTANCE_TYPE	7824	DL: Invalid instance type
H_ERR_DL_NODE_MISSING_PARAM_NAME	7830	DL-Node: Missing generic parameter 'name'. Please specify a layer name.
H_ERR_DL_NODE_GENPARAM_NAME_NOT_ALLOWED	7831	DL-Node: No generic parameter 'name' allowed for this node.
H_ERR_DL_NODE_INVALID_SPEC	7832	DL-Node: Invalid layer specification.
H_ERR_DL_NODE_DUPLICATE_EDGE	7833	DL-Node: There can only be one direct connection between two layers.
H_ERR_DL_SOLVER_INVALID_TYPE	7840	DL: Invalid solver type.
H_ERR_DL_SOLVER_INVALID_UPDATE_FORMULA	7841	DL: Invalid solver update formula.
H_ERR_DL_HEATMAP_UNSUPPORTED_RUNTIME	7850	DL: The heatmap is unsupported with the selected runtime
H_ERR_DL_HEATMAP_UNSUPPORTED_MODEL_TYPE	7851	DL: The heatmap is not available for the selected model type
H_ERR_DL_HEATMAP_UNSUPPORTED_METHOD	7852	DL: The selected heatmap method is not supported
H_ERR_DL_HEATMAP_WRONG_TARGET_CLASS_ID	7853	DL: The selected target class is not available. Please choose a valid class id.
H_ERR_DL_GCAD_NETWORK_NOT_AVAILABLE	7870	DL: GC anomaly detection model network is not available.
H_ERR_DL_ANOMALY_MODEL_INTERNAL	7880	DL: Anomaly detection model internal error.
H_ERR_DL_ANOMALY_MODEL_UNTRAINED	7881	DL: Anomaly detection model is not yet trained.
H_ERR_DL_ANOMALY_MODEL_TRAINING_FAILED	7882	DL: Training of anomaly detection model failed.
H_ERR_DL_ANOMALY_MODEL_PARAM_TRAINED	7883	DL: Unable to set parameter in a trained anomaly detection model.
H_ERR_DL_ANOMALY_MODEL_RESIZE	7884	DL: Changing the model input size failed for the specified parameter value.
H_ERR_DL_ANOMALY_MODEL_DEPTH	7885	DL: Model input depth is not supported.
H_ERR_DL_ANOMALY_MODEL_INPUT_DOMAIN	7886	DL: The domain of the input image is empty.
H_ERR_DEEP_OCR_MODEL_INTERNAL	7890	Deep OCR internal error.
H_ERR_DEEP_OCR_MODEL_INVALID_ALPHABET	7891	Alphabet cannot contain items with a string length greater than one.

Error Name	Code	Description
H_ERR_DEEP_OCR_MODEL_INVALID_ALPHABET_IDX	7892	Out of bounds indexing into alphabet.
H_ERR_DEEP_OCR_MODEL_INVALID_MODEL_TYPE	7893	The type of the given Deep OCR model component is not allowed.
H_ERR_DEEP_OCR_MODEL_NOT_AVAILABLE	7894	The Deep OCR model component is not available.
H_ERR_DEEP_OCR_MODEL_INVALID_ALPHABET_MAPPING_NO_ALPHABET	7895	It is not possible to specify a mapping because there is no internal alphabet specified.
H_ERR_DEEP_OCR_MODEL_INVALID_ALPHABET_MAPPING_IDX	7896	Out of bounds index into internal alphabet given in mapping.
H_ERR_DEEP_OCR_MODEL_INVALID_ALPHABET_MAPPING_LEN	7897	The length of the mapping and the length of the internal alphabet need to be the same.
H_ERR_DEEP_OCR_MODEL_FILE_NOT_FOUND	7898	The Deep OCR model file could not be found. Deep OCR requires the installation of Deep Learning. Please make sure it is installed correctly (see Installation Guide).
H_ERR_DEEP_OCR_MODEL_UNKNOWN_CHAR	7899	Some character is not part of the internal alphabet.
H_ERR_DEEP_OCR_MODEL_INVALID_WORD_LENGTH	7900	The given word length is invalid.
H_ERR_DEEP_OCR_MODEL_ALPHABET_NOT_UNIQUE	7901	The given alphabet is not a unique list of characters
H_ERR_DL_MODEL_APPLY_NO_DEF_OUTPUTS	7910	DL: The model architecture does not allow to specify default outputs
H_ERR_DL_MODEL_UNSUPPORTED_GENPARAM	7911	DL: Unsupported generic parameter
H_ERR_DL_MODEL_OPERATOR_UNSUPPORTED	7912	DL: Operator does not support the model type
H_ERR_DL_MODEL_RUNTIME	7913	DL: Requested runtime cannot be set
H_ERR_DL_MODEL_UNSUPPORTED_GENVALUE	7914	DL: Unsupported generic parameter value
H_ERR_DL_MODEL_INVALID_NUM_SAMPLES	7915	DL: Invalid number of samples
H_ERR_DL_MODEL_CONVERTED_PARAM	7916	DL: Parameter unsupported for converted model
H_ERR_DL_MODEL_CONVERTED_UNSUPPORTED	7917	DL: Unsupported operation on converted model
H_ERR_DEEP_COUNTING_NOT_PREPARED	7940	The Deep Counting model is not prepared for applying on images
H_ERR_DEEP_COUNTING_UNSUPPORTED_BACKBONE	7941	The type of the backbone for Deep Counting model must be of the model type counting
H_ERR_DEEP_COUNTING_PREPARE_UNSUPPORTED	7942	The usage of preparing a Deep Counting model is not supported in this context
H_ERR_DEEP_COUNTING_NO_BACKBONE	7943	Deep Counting model has no backbone
H_ERR_DL_DEVICE_UNSUPPORTED_PRECISION	7960	DL: Unsupported device precision
H_ERR_DL_PRUNING_WRONG_DATA	7980	DL: Given pruning data and model are not compatible
H_ERR_DL_PRUNING_UNSUPPORTED_BY_CNN	7981	DL: The model architecture does not support pruning
H_ERR_DL_MODULE_NOT_LOADED	7990	DL: The DL module (libhalcondl.so/halcondl.dll) could not be loaded
H_ERR_WPRN	8000	Unknown operator name
H_ERR_RCNA	8001	register_comp_used is not activated (see set_system)
H_ERR_WPC	8002	Unknown operator class
H_ERR_ORMF	8101	convol/mask: error while opening the file
H_ERR_EOFRMF	8102	convol/mask: premature end of file
H_ERR_CVTRMF	8103	convol/mask: conversion error
H_ERR_LCNRMF	8104	convol/mask: wrong row-/column number
H_ERR_WCOVRMF	8105	convol/mask: mask size overflow
H_ERR_NEOFRMF	8106	convol/mask: too many elements entered
H_ERR_WRRA	8107	convol: wrong margin type
H_ERR_MCNO	8108	convol: no mask object has got empty region
H_ERR_WFO	8110	convol: Weight factor is 0
H_ERR_NWC	8111	convol: inconsistent number of weights
H_ERR_WRRV	8112	rank: wrong rank value
H_ERR_ROVFL	8113	convol/rank: error while handling margin
H_ERR_EWPMF	8114	Error while parsing filter mask file
H_ERR_WNUMM	8120	Wrong number of coefficients for convolution (sigma too big?)
H_ERR_WBEDN	8200	No valid ID for data set
H_ERR_NBEDA	8201	No data set active (set_bg_esti)
H_ERR_BEDNAU	8202	ID already used for data set (is not possible)
H_ERR_NBEDC	8204	No data set created (create_bg_esti)
H_ERR_NTM	8205	Not possible to pass an object list

Error Name	Code	Description
H_ERR_WISBE	8206	Image has other size than the background image in data set
H_ERR_UDNSSBE	8207	Up-date-region is bigger than background image
H_ERR_SNBETS	8208	Number of statistic data sets is too small
H_ERR_WAMBE	8209	Wrong value for adapt mode
H_ERR_WFMBE	8210	Wrong value for frame mode
H_ERR_PE_NPCTS	8250	Number of point correspondences too small
H_ERR_PE_INVMET	8251	Invalid pose estimation method
H_ERR_OCR_MEM1	8300	Maximum number of fonts exceeded
H_ERR_OCR_WID	8301	Wrong ID (Number) for font
H_ERR_OCR1	8302	OCR internal error: wrong ID
H_ERR_OCR_NNI	8303	OCR not initialized: no font was read in
H_ERR_OCR_NAI	8304	No font activated
H_ERR_OCR_WTP	8305	OCR internal error: wrong threshold in angle determination
H_ERR_OCR_WF	8306	OCR internal error: wrong attribute
H_ERR_OCR_READ	8307	The version of the OCR classifier is not supported
H_ERR_OCR_NODES	8308	OCR File: inconsistent number of nodes
H_ERR_OCR_EOF	8309	OCR File: File too short
H_ERR_OCR_INC1	8310	OCR: internal error 1
H_ERR_OCR_INC2	8311	OCR: internal error 2
H_ERR_WOCRTYPE	8312	Wrong type of OCR tool (no 'box' or 'net')
H_ERR_OCR_TRF	8313	The version of the OCR training characters is not supported
H_ERR_TRF_ITL	8314	Image too large for training file
H_ERR_TRF_RTL	8315	Region too large for training file
H_ERR_TRF_PT	8316	Protected training file
H_ERR_TRF_WPW	8317	Wrong password for protected training file
H_ERR_OCR_NOSITEM	8318	Serialized item does not contain a valid OCR classifier
H_ERR_TRF_CON_EIO	8319	OCR training file concatenation failed: identical input and output files
H_ERR_OCR_MLP_NOCLASSFILE	8320	Invalid file format for MLP classifier
H_ERR_OCR_MLP_WRCLASSVERS	8321	The version of the MLP classifier is not supported
H_ERR_OCR_MLP_NOSITEM	8322	Serialized item does not contain a valid MLP classifier
H_ERR_OCR_SVM_NOCLASSFILE	8330	Invalid file format for SVM classifier
H_ERR_OCR_SVM_WRCLASSVERS	8331	The version of the SVM classifier is not supported
H_ERR_OCR_SVM_NOSITEM	8332	Serialized item does not contain a valid SVM classifier
H_ERR_OCR_KNN_NOCLASSFILE	8333	Invalid file format for k-NN classifier
H_ERR_OCR_KNN_NOSITEM	8334	Serialized item does not contain a valid k-NN classifier
H_ERR_OCR_CNN_NOCLASSFILE	8335	Invalid file format for CNN classifier
H_ERR_OCR_CNN_WRCLASSVERS	8336	The version of the CNN classifier is not supported
H_ERR_OCR_CNN_NOSITEM	8337	Serialized item does not contain a valid CNN classifier
H_ERR_OCR_RESULT_NOT_AVAILABLE	8338	The result is not available for the selected text segmentation method
H_ERR_OCV_NI	8350	OCV system not initialized
H_ERR_WOCVTYPE	8351	The version of the OCV tool is not supported
H_ERR_OCV_WNAME	8353	Wrong name for an OCV object
H_ERR_OCV_II	8354	Training has already been applied
H_ERR_OCV_NOTTR	8355	No training has been applied to the character
H_ERR_OCV_NOSITEM	8356	Serialized item does not contain a valid OCV tool
H_ERR_WLENGTH	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	Illegal distance of function points
H_ERR_NOT_MONOTONIC	8374	Function is not monotonic
H_ERR_WFUNCTION	8375	Wrong function type
H_ERR_SAME_XVAL_CONV	8376	The function contains multiple points with different x-values, that match the same value after the conversion from double to float.
H_ERR_GRID_CONNECT_POINTS	8390	The input points could not be arranged in a regular grid

Error Name	Code	Description
H_ERR_GRID_GEN_MAP	8391	Error while creating the output map
H_ERR_GRID_AUTO_ROT	8392	Auto rotation failed
H_ERR_CAL_NO_COMM_PAR	8393	No common camera parameters
H_ERR_CAL_NEGVY	8394	Vy must be > 0
H_ERR_CAL_IDENTICAL_FP	8395	Same finder pattern found multiple times
H_ERR_CAL_LSCPNA	8396	Function not available for line scan cameras with perspective lenses
H_ERR_CAL_MARK_SEGM	8397	Segmentation of the calibration marks failed
H_ERR_CAL_CONT_EXT	8398	Extraction of the mark contours failed
H_ERR_CAL_NO_FP	8399	No finder pattern could be found
H_ERR_CAL_LCALP	8400	You have to indicate at least 3 calibration points
H_ERR_CAL_INCONSISTENT_FP	8401	Inconsistent finder pattern positions (image mirrored?)
H_ERR_CAL_NCPF	8402	No calibration table found
H_ERR_CAL_RECPF	8403	Error while reading calibration table description file
H_ERR_CAL_LTMTH	8404	Minimum threshold while searching for ellipses
H_ERR_CAL_FRCP	8405	Read error / format error in calibration table description file
H_ERR_CAL_PROJ	8406	Point cannot be projected
H_ERR_CAL_UNPRO	8407	Error in inverse projection
H_ERR_CAL_RICPF	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_FOPOS	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 normalizing the vector
H_ERR_CAL_NPLAN	8417	Fitting of calibration target failed
H_ERR_CAL_NNMAR	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_NOELL	8421	Non elliptic contour
H_ERR_CAL_WPARV	8422	Wrong parameter value slvand()
H_ERR_CAL_WFRES	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_CAL_NPILS	8429	Point could not be projected into linescan image
H_ERR_CAL_DIACM	8430	Diameter of calibration marks could not be determined
H_ERR_CAL_ORICP	8431	Orientation of calibration plate could not be determined
H_ERR_CAL_CPNII	8432	Calibration plate does not lie completely inside the image
H_ERR_CAL_WNCME	8433	Wrong number of calibration marks extracted
H_ERR_CAL_UNKPG	8434	Unknown name of parameter group
H_ERR_CAL_NEGFL	8435	Focal length must be non-negative
H_ERR_CAL_TELNA	8436	Function not available for cameras with telecentric lenses
H_ERR_CAL_LSCNA	8437	Function not available for line scan cameras
H_ERR_CAL_ELLDP	8438	Ellipse is degenerated to a point
H_ERR_CAL_NOMF	8439	No orientation mark found
H_ERR_CAL_NCONV	8440	Camera calibration did not converge
H_ERR_CAL_HYPNA	8441	Function not available for cameras with hypercentric lenses
H_ERR_CAL_DISTORT	8442	Point cannot be distorted
H_ERR_CAL_WREDGFILT	8443	Wrong edge filter for the mark extraction

Error Name	Code	Description
H_ERR_CAL_NEGPS	8444	Pixel size must be positive
H_ERR_CAL_NEGTS	8445	Tilt must be within the range of 0° and 90°
H_ERR_CAL_NEGRS	8446	Rot must be within the range of 0° and 360°
H_ERR_CAL_INVCAMPAR	8447	Invalid camera parameters
H_ERR_CAL_ILLFL	8448	Focal length must be positive
H_ERR_CAL_ILLMAG	8449	Magnification must be positive
H_ERR_CAL_ILLIPD	8450	Illegal image plane distance
H_ERR_CM_NOT_OPTIMIZED	8451	Model not optimized yet - no results can be queried
H_ERR_CM_NOT_POSTPROCC	8452	Model not postprocessed yet - no auxiliary results can be queried
H_ERR_CM_NOT_INTERCONN	8453	Calibration setup: fields of view do not intersect
H_ERR_CM_CAMPAR_MISMCH	8454	Camera type and camera parameters incompatible
H_ERR_CM_CAMTYP_MISMCH	8455	Calibration setup: incompatible camera types
H_ERR_CM_CAMTYP_UNSUPD	8456	Camera type not supported
H_ERR_CM_INVALD_CAMIDX	8457	Invalid camera index
H_ERR_CM_INVALD_DESCID	8458	Invalid calibration object index
H_ERR_CM_INVALD_COBJID	8459	Invalid calibration object pose index
H_ERR_CM_UNDEFINED_CAM	8460	Undefined camera
H_ERR_CM_REPEATD_INDEX	8461	Indices: ambiguous observation index
H_ERR_CM_UNDEFI_CADESC	8462	Undefined calibration object
H_ERR_CM_NO_DESCR_FILE	8463	Invalid file format for calibration data model
H_ERR_CM_WR_DESCR_VERS	8464	The version of the calibration data model is not supported
H_ERR_CM_ZERO_MOTION	8465	Zero-motion in line scan camera parameters
H_ERR_CM_MULTICAM_UNSP	8466	Calibration setup: multiple cameras and/or calibration objects not supported for camera type
H_ERR_CM_INCMPLTE_DATA	8467	Incomplete observation data
H_ERR_CSM_NO_DESCR_FIL	8468	Invalid file format for camera setup model
H_ERR_CSM_WR_DESCR_VER	8469	The version of the camera setup model is not supported
H_ERR_CM_CALTAB_NOT_AV	8470	Full HALCON calibration plate description required
H_ERR_CM_INVAL_OBSERID	8471	Invalid observation index
H_ERR_CSM_NOSITEM	8472	Serialized item does not contain a valid camera setup model
H_ERR_CM_NOSITEM	8473	Serialized item does not contain a valid calibration data model
H_ERR_CM_INV_TOOLPOSID	8474	Invalid tool pose index
H_ERR_CM_UNDEFINED_TOO	8475	Undefined tool pose
H_ERR_CM_INVLD_MODL_TY	8476	Feature or operation not supported for current calibration data model type
H_ERR_CSM_UNINIT_CAM	8477	The camera setup model contains an uninitialized camera
H_ERR_CM_NO_VALID_SOL	8478	The hand-eye algorithm failed to find a valid solution
H_ERR_CM_INVAL_OBS_POSE	8479	Invalid observation pose: set_calib_data_observ_pose can only be called with calibration data models that have been configured for hand-eye-calibration; please use find_calib_object or set_calib_data_observ_points
H_ERR_CM_TOO_FEW_POSES	8480	Not enough calibration object poses: For the hand-eye-calibration at least three calibration object poses are necessary
H_ERR_CM_UNDEF_CAM_TYP	8481	Undefined camera type
H_ERR_SM_INVLD_IMG_PAIRS_DISP_VAL	8482	Number of image pairs does not correspond to the number of disparity values
H_ERR_SM_INVLD_DISP_VAL	8483	Wrong value of stereo model parameter 'min_disparity' or 'max_disparity'
H_ERR_SM_NO_IM_PAIR	8484	No camera pair set by set_stereo_model_image_pairs
H_ERR_SM_NO_VIS_COLOR	8485	No reconstructed point is visible for coloring
H_ERR_SM_NO_RECONSTRUCT	8486	No camera pair yields reconstructed points (please check parameters of disparity method or bounding box)
H_ERR_SM_INVLD_BB_PARTITION	8487	Partitioning of bounding box is too fine (please adapt the parameter 'resolution' or the bounding box)
H_ERR_SM_INVLD_DISP_RANGE	8488	Disparity range as calculated by the bounding box and the parameter 'rectif_sub_sampling' exceeds the maximum permitted

Error Name	Code	Description
H_ERR_SM_INVLD_BIN_PAR	8489	The parameter to be set is not supported by the chosen disparity method
H_ERR_SM_INVLD_MODL_TY	8490	Feature or operation not supported for current stereo model type
H_ERR_SM_NOT_PERSISTEN	8491	Feature or operation available only in 'persistent' mode
H_ERR_SM_INVLD_BOU_BOX	8492	Invalid bounding box
H_ERR_SR_INVLD_IMG_SIZ	8493	Image sizes must be identical with the corresponding camera parameters from the camera setup
H_ERR_SR_BBOX_BHND_CAM	8494	Bounding box lies partially or completely behind (for hypercentric lenses: in front of) the base line of at least one camera pair
H_ERR_CAL_AMBIGUOUS	8495	Ambiguous calibration: Please, recalibrate with improved input data!
H_ERR_CAL_PCPND	8496	Pose of calibration plate could not be determined!
H_ERR_CAL_FAILED	8497	Calibration failed: Please check your input data and calibrate again!
H_ERR_CAL_MISSING_DATA	8498	No observation data supplied!
H_ERR_CAL_FEWER_FOUR	8499	The calibration object has to be seen at least once by every camera, if less than four cameras are used
H_ERR_NOAP	8500	Invalid file format for template
H_ERR_WPFV	8501	The version of the template is not supported
H_ERR_MATCH_MODE	8502	Error during changing the file mode (t/b)
H_ERR_MATCH_OOR	8503	Inconsistent match file: coordinates out of range
H_ERR_NOTAP	8505	The image(s) is not a pyramid (wrong zooming factor?)
H_ERR_NGTPTS	8506	Number of template points too small
H_ERR_PDTL	8507	Template data can only be read by HALCON XL
H_ERR_NCC_NOSITEM	8508	Serialized item does not contain a valid NCC model
H_ERR_MATCH_NOSITEM	8509	Serialized item does not contain a valid template
H_ERR_NTPTS	8510	Number of shape model points too small
H_ERR_CGSMM	8511	Gray-value-based and color-based shape models cannot be searched simultaneously
H_ERR_SMTL	8512	Shape model data can only be read by HALCON XL
H_ERR_SMNXLD	8513	Shape model was not created from XLDs
H_ERR_SM_NOSITEM	8514	Serialized item does not contain a valid shape model
H_ERR_SM_CL_CONT	8515	Shape model contour too near to clutter region
H_ERR_SM_NO_CLUT	8516	Shape model does not contain clutter parameters
H_ERR_SM_SAME_CL	8517	Activation status for the usage of clutter parameters is not identical for all shape models
H_ERR_SM_WRONG_CLCO	8518	Shape model has an invalid clutter contrast
H_ERR_FIND_BOX_UNSUP_GENPARAM	8520	Box Finder: Unsupported generic parameter
H_ERR_COMP_DRT	8530	Initial components have different region types
H_ERR_COMP_SAMF	8531	Solution of ambiguous matches failed
H_ERR_IGF_NC	8532	Computation of the incomplete gamma function not converged
H_ERR_MSA_TMN	8533	Too many nodes while computing the minimum spanning arborescence
H_ERR_CTTL	8534	Component training data can only be read by HALCON XL
H_ERR_CMTL	8535	Component model data can only be read by HALCON XL
H_ERR_COMP_NOSITEM	8536	Serialized item does not contain a valid component model
H_ERR_TRAIN_COMP_NOSITEM	8537	Serialized item does not contain a valid component training result
H_ERR_VARIATION_WS	8540	Size of the training image and the variation model differ
H_ERR_VARIATION_PREP	8541	Variation model has not been prepared for segmentation
H_ERR_VARIATION_WRMD	8542	Invalid variation model training mode
H_ERR_VARIATION_NOVF	8543	Invalid file format for variation model
H_ERR_VARIATION_WVFV	8544	The version of the variation model is not supported
H_ERR_VARIATION_TRDC	8545	Training data has already been cleared
H_ERR_VARIATION_NOSITEM	8546	Serialized item does not contain a valid variation model
H_ERR_MEASURE_NA	8550	No more measure objects available

Error Name	Code	Description
H_ERR_MEASURE_NI	8551	Measure object is not initialized
H_ERR_MEASURE_OOR	8552	Invalid measure object
H_ERR_MEASURE_IS	8553	Measure object is NULL
H_ERR_MEASURE_WS	8554	Measure object has wrong image size
H_ERR_MEASURE_NO_MODEL_FILE	8555	Invalid file format for measure object
H_ERR_MEASURE_WRONG_VERSION	8556	The version of the measure object is not supported
H_ERR_MEASURE_TL	8557	Measure object data can only be read by HALCON XL
H_ERR_MEASURE_NOSITEM	8558	Serialized item does not contain a valid measure object
H_ERR_METROLOGY_MODEL_NI	8570	Metrology model is not initialized
H_ERR_METROLOGY_OBJECT_INVALID	8572	Invalid metrology object
H_ERR_METROLOGY_FIT_NOT_ENOUGH_MEASURES	8573	Not enough valid measures for fitting the metrology object
H_ERR_METROLOGY_NO_MODEL_FILE	8575	Invalid file format for metrology model
H_ERR_METROLOGY_WRONG_VERSION	8576	The version of the metrology model is not supported
H_ERR_METROLOGY_NO_FUZZY_FUNC	8577	Fuzzy function is not set
H_ERR_METROLOGY_NOSITEM	8578	Serialized item does not contain a valid metrology model
H_ERR_METROLOGY_UNDEF_CAMPAR	8579	Camera parameters are not set
H_ERR_METROLOGY_UNDEF_POSE	8580	Pose of the measurement plane is not set
H_ERR_METROLOGY_SET_MODE	8581	Mode of metrology model cannot be set since an object has already been added
H_ERR_METROLOGY_OP_NOT_ALLOWED	8582	If the pose of the metrology object has been set several times, the operator is not longer allowed
H_ERR_METROLOGY_MULTI_POSE_CAM_PAR	8583	All objects of a metrology model must have the same world pose and camera parameters
H_ERR_METROLOGY_WRONG_INPUT_MODE	8584	The input type does not correspond to the input type set for the metrology model
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_COMPONENT_NOT_INSTALLED	8603	Interface library not available. Check www.mvtec.com/download for additional interfaces
H_ERR_EAD_CAL_NII	8650	Not enough information for radiometric calibration
H_ERR_WGSMFV	8670	The version of the shape model result is not supported
H_ERR_GSM_INVALID_RES_SCALE	8671	Restrict scale parameter outside the trained range
H_ERR_GSM_INVALID_ANGLE	8672	Angle parameter outside the trained range
H_ERR_GSM_NEEDS_TRAINING	8673	Shape model needs training
H_ERR_GSM_CONTRAST_HYS	8674	contrast_high cannot be smaller than contrast_low
H_ERR_GSM_CONTRAST_MIN_CONTRAST	8675	Neither contrast_low nor contrast_high can be smaller than min_contrast
H_ERR_GSM_ISO_SCALE_PAIR	8676	iso_scale_max cannot be smaller than iso_scale_min
H_ERR_GSM_ANISO_SCALE_ROW	8677	scale_row_max cannot be smaller than scale_row_min
H_ERR_GSM_ANISO_SCALE_COLUMN	8678	scale_column_max cannot be smaller than scale_column_min
H_ERR_GSM_ISO_NOT_SET	8679	Isotropic scaling not set
H_ERR_GSM_ANISO_NOT_SET	8680	Anisotropic scaling not set
H_ERR_GSM_INVALID_METRIC_XLD	8681	No edge direction available to change shape matching metric
H_ERR_GSM_SAME_IDENTIFIER	8682	Shape models with the same identifier cannot be searched simultaneously
H_ERR_BAR_WNOM	8701	Wrong number of modules
H_ERR_BAR_WNOE	8702	Wrong number of elements
H_ERR_BAR_UNCHAR	8703	Unknown character (for this code)
H_ERR_BAR_WRONGDESCR	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_WRONGCODE	8708	Wrong type of bar code
H_ERR_BAR_INTERNAL	8709	Internal error in bar code reader
H_ERR_BAR_NO_DECODED_SCANLINE	8710	Bar code candidate does not contain a decoded scanline
H_ERR_BC_EMPTY_MODEL_LIST	8721	List of bar code models is empty
H_ERR_BC_TRAIN_ONLY_SINGLE	8722	Training cannot be done for multiple bar code types

Error Name	Code	Description
H_ERR_BC_GET_SPECIFIC	8723	Cannot get bar code type specific parameter with get_bar_code_param. Use get_bar_code_param_specific
H_ERR_BC_GET_OBJ_MULTI	8724	Cannot get this object for multiple bar code types. Try again with single bar code type
H_ERR_BC_WR_FILE_FORMAT	8725	Invalid file format for bar code model
H_ERR_BC_WR_FILE_VERS	8726	The version of the bar code model is not supported
H_ERR_BC_NOT_PERSISTANT	8727	Feature or operation available only in 'persistent' mode
H_ERR_BC_GRAY_OUT_OF_RANGE	8728	Incorrect index of scanline's gray values
H_ERR_NO_PERSISTENT_OP_CALL	8729	Neither find_bar_code nor decode_bar_code_rectanlge2 have been called in persistent mode on this model
H_ERR_BC_ZOOMED_ABORTED	8730	The zoomed barcode algorithm has been aborted
H_ERR_BC_ZOOMED_INVALID_INPUT	8731	zoomed barcode: Invalid input data.
H_ERR_BC_XCORR_INVALID_INPUT	8740	Barcode: Invalid inputs for NCC
H_ERR_BC_XCORR_TOO_MANY_BAD_ROWS	8741	Barcode: Too many invalid rows during NCC
H_ERR_BC_XCORR_NO_CORRELATION	8742	Barcode: No correlation found during NCC
H_ERR_INVALID_SYNTAX_DICTIONARY	8743	Invalid GS1 Application Identifier Syntax Dictionary
H_ERR_BAR2D_UNKNOWN_TYPE	8800	Specified code type is not supported
H_ERR_BAR2D_WRONG_FOREGROUND	8801	Wrong foreground specified
H_ERR_BAR2D_WRONG_SIZE	8802	Wrong matrix size specified
H_ERR_BAR2D_WRONG_SHAPE	8803	Wrong symbol shape specified
H_ERR_BAR2D_WRONG_PARAM_NAME	8804	Wrong generic parameter name
H_ERR_BAR2D_WRONG_PARAM_VAL	8805	Wrong generic parameter value
H_ERR_BAR2D_WRONG_MODE	8806	Wrong symbol printing mode
H_ERR_BAR2D_SYMBOL_ON_BORDER	8807	Symbol region too near to image border
H_ERR_BAR2D_MODULE_CONT_NUM	8808	No rectangular modul boundaries found
H_ERR_BAR2D_SYMBOL_FINDER	8809	Couldn't identify symbol finder
H_ERR_BAR2D_SYMBOL_DIMENSION	8810	Symbol region with wrong dimension
H_ERR_BAR2D_CLASSIF_FAILED	8811	Classification failed
H_ERR_BAR2D_DECODING_FAILED	8812	Decoding failed
H_ERR_BAR2D_DECODING_READER	8813	Reader programming not supported
H_ERR_DC2D_GENERAL	8820	General 2d data code error
H_ERR_DC2D_BROKEN_SIGN	8821	Corrupt signature of 2d data code handle
H_ERR_DC2D_INVALID_HANDLE	8822	Invalid 2d data code handle
H_ERR_DC2D_EMPTY_MODEL_LIST	8823	List of 2d data code models is empty
H_ERR_DC2D_NOT_INITIALIZED	8824	Access to uninitialized (or not persistent) internal data
H_ERR_DC2D_INVALID_CANDIDATE	8825	Invalid 'Candidate' parameter
H_ERR_DC2D_INDEX_PARNUM	8826	It's not possible to return more than one parameter for several candidates
H_ERR_DC2D_EXCLUSIV_PARAM	8827	One of the parameters returns several values and has to be used exclusively for a single candidate
H_ERR_DC2D_DEF_SET_NOT_FIRST	8828	Parameter for default settings must be the first in the parameter list
H_ERR_DC2D_INTERNAL_UNEXPECTED	8829	Unexpected 2d data code error
H_ERR_DC2D_WRONG_PARAM_VALUE	8830	Invalid parameter value
H_ERR_DC2D_WRONG_PARAM_NAME	8831	Unknown parameter name
H_ERR_DC2D_WRONG_POLARITY	8832	Invalid value for 'polarity'
H_ERR_DC2D_WRONG_SYMBOL_SHAPE	8833	Invalid value for 'symbol_shape'
H_ERR_DC2D_WRONG_SYMBOL_SIZE	8834	Invalid symbol size
H_ERR_DC2D_WRONG_MODULE_SIZE	8835	Invalid module size
H_ERR_DC2D_WRONG_MODULE_SHAPE	8836	Invalid value for 'module_shape'
H_ERR_DC2D_WRONG_ORIENTATION	8837	Invalid value for 'orientation'
H_ERR_DC2D_WRONG_CONTRAST	8838	Invalid value for 'contrast_min'
H_ERR_DC2D_WRONG_MEAS_THRESH	8839	Invalid value for 'measure_thresh'
H_ERR_DC2D_WRONG_ALT_MEAS_RED	8840	Invalid value for 'alt_measure_red'
H_ERR_DC2D_WRONG_SLANT	8841	Invalid value for 'slant_max'
H_ERR_DC2D_WRONG_L_DIST	8842	Invalid value for 'L_dist_max'
H_ERR_DC2D_WRONG_L_LENGTH	8843	Invalid value for 'L_length_min'

Error Name	Code	Description
H_ERR_DC2D_WRONG_GAP	8844	Invalid module gap
H_ERR_DC2D_WRONG_DEF_SET	8845	Invalid value for 'default_parameters'
H_ERR_DC2D_WRONG_TEXTURED	8846	Invalid value for 'back_texture'
H_ERR_DC2D_WRONG_MIRRORED	8847	Invalid value for 'mirrored'
H_ERR_DC2D_WRONG_CLASSIFICATOR	8848	Invalid value for 'classificator'
H_ERR_DC2D_WRONG_PERSISTENCE	8849	Invalid value for 'persistence'
H_ERR_DC2D_WRONG_MODEL_TYPE	8850	Invalid model type
H_ERR_DC2D_WRONG_MOD_ROI_PART	8851	Invalid value for 'module_roi_part'
H_ERR_DC2D_WRONG_FP_TOLERANCE	8852	Invalid value for 'finder_pattern_tolerance'
H_ERR_DC2D_WRONG_MOD_ASPECT	8853	Invalid value for 'mod_aspect_max'
H_ERR_DC2D_WRONG_SM_ROBUSTNESS	8854	Invalid value for 'small_modules_robustness'
H_ERR_DC2D_WRONG_CONTRAST_TOL	8855	Invalid 'contrast_tolerance'
H_ERR_DC2D_WRONG_AP_TOLERANCE	8856	Invalid value for 'alternating_pattern_tolerance'
H_ERR_DC2D_READ_HEAD_FORMAT	8860	Invalid header in 2d data code model file
H_ERR_DC2D_READ_HEAD_SIGN	8861	Invalid code signature in 2d data code model file
H_ERR_DC2D_READ_LINE_FORMAT	8862	Corrupted line in 2d data code model file
H_ERR_DC2D_WRONG_MODULE_ASPECT	8863	Invalid module aspect ratio
H_ERR_DC2D_WRONG_LAYER_NUM	8864	Invalid layer num
H_ERR_DCD_READ_WRONG_VERSION	8865	Wrong data code model file version
H_ERR_DC2D_NOSITEM	8866	Serialized item does not contain a valid 2D data code model
H_ERR_DC2D_WR_FILE_FORMAT	8867	Invalid file format for data code model
H_ERR_SM3D_WRONG_PARAM_NAME	8900	Unknown parameter name
H_ERR_SM3D_WRONG_NUM_LEVELS	8901	Invalid value for 'num_levels'
H_ERR_SM3D_WRONG_OPTIMIZATION	8902	Invalid value for 'optimization'
H_ERR_SM3D_WRONG_METRIC	8903	Invalid value for 'metric'
H_ERR_SM3D_WRONG_MIN_FACE_ANGLE	8904	Invalid value for 'min_face_angle'
H_ERR_SM3D_WRONG_MIN_SIZE	8905	Invalid value for 'min_size'
H_ERR_SM3D_WRONG_MODEL_TOLERANCE	8906	Invalid value for 'model_tolerance'
H_ERR_SM3D_WRONG_FAST_POSE_REF	8907	Invalid value for 'fast_pose_refinement'
H_ERR_SM3D_WRONG_LOWEST_MODEL_LEVEL	8908	Invalid value for 'lowest_model_level'
H_ERR_SM3D_WRONG_PART_SIZE	8909	Invalid value for 'part_size'
H_ERR_SM3D_PROJECTION_TOO_LARGE	8910	The projected model is too large (the value for DistMin or the image size in CamParam is too small)
H_ERR_SM3D_WRONG_OPENGL_ACCURACY	8911	Invalid value for 'opengl_accuracy'
H_ERR_SM3D_WRONG_RECOMPUTE_SCORE	8913	Invalid value for 'recompute_score'
H_ERR_SM3D_WRONG_LON_MIN	8920	Invalid value for 'longitude_min'
H_ERR_SM3D_WRONG_LON_MAX	8921	Invalid value for 'longitude_max'
H_ERR_SM3D_WRONG_LAT_MIN	8922	Invalid value for 'latitude_min'
H_ERR_SM3D_WRONG_LAT_MAX	8923	Invalid value for 'latitude_max'
H_ERR_SM3D_WRONG_ROL_MIN	8924	Invalid value for 'cam_roll_min'
H_ERR_SM3D_WRONG_ROL_MAX	8925	Invalid value for 'cam_roll_max'
H_ERR_SM3D_WRONG_DIST_MIN	8926	Invalid value for 'dist_min'
H_ERR_SM3D_WRONG_DIST_MAX	8927	Invalid value for 'dist_max'
H_ERR_SM3D_WRONG_NUM_MATCHES	8928	Invalid value for 'num_matches'
H_ERR_SM3D_WRONG_MAX_OVERLAP	8929	Invalid value for 'max_overlap'
H_ERR_SM3D_WRONG_POSE_REFINEMENT	8930	Invalid value for 'pose_refinement'
H_ERR_SM3D_WRONG_COV_POSE_MODE	8931	Invalid value for 'cov_pose_mode'
H_ERR_SM3D_WRONG_OUTLIER_SUP	8932	Invalid value for 'outlier_suppression'
H_ERR_SM3D_WRONG_BORDER_MODEL	8933	Invalid value for 'border_model'
H_ERR_SM3D_UNDEFINED_POSE	8940	Pose is not well-defined
H_ERR_SM3D_NO_SM3D_FILE	8941	Invalid file format for 3D shape model
H_ERR_SM3D_WRONG_FILE_VERSION	8942	The version of the 3D shape model is not supported
H_ERR_SM3D_MTL	8943	3D shape model data can only be read by HALCON XL
H_ERR_SM3D_NO_OM3D_FACES	8944	3D object model does not contain any faces
H_ERR_SM3D_NOSITEM	8945	Serialized item does not contain a valid 3D shape model
H_ERR_SM3D_WRONG_UNION_ADJACENT_CONTOURS	8946	Invalid value for 'union_adjacent_contours'

Error Name	Code	Description
H_ERR_DESCR_NODESCRFILE	8960	Invalid file format for descriptor model
H_ERR_DESCR_WRDESCRVERS	8961	The version of the descriptor model is not supported
H_ERR_DM_WRONG_NUM_CIRC_RADIUS	8962	Invalid value for 'radius'
H_ERR_DM_WRONG_NUM_CHECK_NEIGH	8963	Invalid value for 'check_neighbor'
H_ERR_DM_WRONG_NUM_MIN_CHECK_NEIGH	8964	Invalid value for 'min_check_neighbor_diff'
H_ERR_DM_WRONG_NUM_MIN_SCORE	8965	Invalid value for 'min_score'
H_ERR_DM_WRONG_NUM_SIGMAGRAD	8966	Invalid value for 'sigma_grad'
H_ERR_DM_WRONG_NUM_SIGMAINT	8967	Invalid value for 'sigma_smooth'
H_ERR_DM_WRONG_NUM_ALPHA	8968	Invalid value for 'alpha'
H_ERR_DM_WRONG_NUM_THRESHOLD	8969	Invalid value for 'threshold'
H_ERR_DM_WRONG_NUM_DEPTH	8970	Invalid value for 'depth'
H_ERR_DM_WRONG_NUM_TREES	8971	Invalid value for 'number_trees'
H_ERR_DM_WRONG_NUM_MIN_SCORE_DESCR	8972	Invalid value for 'min_score_descr'
H_ERR_DM_WRONG_NUM_PATCH_SIZE	8973	Invalid value for 'patch_size'
H_ERR_DM_WRONG_TILT	8974	Invalid value for 'tilt'
H_ERR_DM_WRONG_PAR_GUIDE	8975	Invalid value for 'guided_matching'
H_ERR_DM_WRONG_PAR_SUBPIX	8976	Invalid value for 'subpix'
H_ERR_DM_TOO_FEW_POINTS	8977	Too few feature points can be found
H_ERR_DM_WRONG_NUM_MINROT	8978	Invalid value for 'min_rot'
H_ERR_DM_WRONG_NUM_MAXROT	8979	Invalid value for 'max_rot'
H_ERR_DM_WRONG_NUM_MINSCALE	8980	Invalid value for 'min_scale'
H_ERR_DM_WRONG_NUM_MAXSCALE	8981	Invalid value for 'max_scale'
H_ERR_DM_WRONG_NUM_MASKSIZEGRD	8982	Invalid value for 'mask_size_grd'
H_ERR_DM_WRONG_NUM_MASKSIZESMOOTH	8983	Invalid value for 'mask_size_smooth'
H_ERR_BROKEN_MODEL	8984	Model broken
H_ERR_DM_WRONG_DESCR_TYPE	8985	Invalid value for 'descriptor_type'
H_ERR_DM_WRONG_PAR_MATCHER	8986	Invalid value for 'matcher'
H_ERR_DM_TOO_MANY_CLASSES	8987	Too many point classes - model storing in a file is not possible
H_ERR_DESCR_NOSITEM	8988	Serialized item does not contain a valid descriptor model
H_ERR_NOT_IMPL	9000	Function not implemented on this machine
H_ERR_WIT	9001	Image to process has wrong gray value type
H_ERR_WIC	9002	Wrong image component (see: get_system(obj_images,H))
H_ERR_UNDI	9003	Undefined gray values
H_ERR_WIS	9004	Wrong image format for operation (too big or too small)
H_ERR_WCN	9005	Wrong number of image components for image output
H_ERR_STRTL	9006	String is too long (max. 1024 characters)
H_ERR_WITFO	9007	Wrong pixel type for this operation
H_ERR_NIIT	9008	Operation not realized yet for this pixel type
H_ERR_NOCIMA	9009	Image is no color image with three channels
H_ERR_DEMO_NOFG	9010	Image acquisition devices are not supported in the demo version
H_ERR_DEMO_NOPA	9011	Packages are not supported in the demo version
H_ERR_IEUNKV	9020	Internal error: Unknown value
H_ERR_WPFO	9021	Wrong parameter for this operation
H_ERR_IDTS	9022	Image domain too small
H_ERR_CNCLDRW	9023	Draw operator has been canceled
H_ERR_REGEX_MATCH	9024	Error during regular expression matching
H_ERR_STUD_OPNA	9050	Operator is not available in this restricted version of HALCON
H_ERR_STUD_PANA	9051	Packages are not available in this restricted version of HALCON
H_ERR_STUD_FGNA	9052	The selected image acquisition interface is not available in this restricted version of HALCON
H_ERR_NDPA	9053	No data points available
H_ERR_WR_OBJ_TYPE	9054	Type of the object is not supported
H_ERR_OP_DISABLED	9055	Operator is disabled
H_ERR_TMU	9100	Too many unknown variables in linear equation

Error Name	Code	Description
H_ERR_NUS	9101	No (unique) solution for the linear equation
H_ERR_NEE	9102	Too little equations in linear equation
H_ERR_PDDL	9150	Points do not define a line
H_ERR_MNI	9200	Matrix is not invertible
H_ERR_SVD_CNVRG	9201	Singular value decomposition did not converge
H_ERR_SVD_FEWROW	9202	Matrix has too few rows for singular value partition
H_ERR_TQLI_CNVRG	9203	Eigenvalue computation did not converge
H_ERR_JACOBI_CNVRG	9204	Eigenvalue computation did not converge
H_ERR_MATRIX_SING	9205	Matrix is singular
H_ERR_MATCH_CNVRG	9206	Function matching did not converge
H_ERR_MAT_UNDEF	9207	Input matrix undefined
H_ERR_MAT_WDIM	9208	Input matrix with wrong dimension
H_ERR_MAT_NSQR	9209	Input matrix is not quadratic
H_ERR_MAT_FAIL	9210	Matrix operation failed
H_ERR_MAT_NPD	9211	Matrix is not positive definite
H_ERR_MAT_DBZ	9212	One element of the matrix is zero: Division by zero
H_ERR_MAT_NUT	9213	Matrix is not an upper triangular matrix
H_ERR_MAT_NLT	9214	Matrix is not a lower triangular matrix
H_ERR_MAT_NEG	9215	One element of the matrix is negative
H_ERR_MAT_UNCHAR	9216	Matrix file: Invalid character
H_ERR_MAT_NOT_COMPLETE	9217	Matrix file: Matrix incomplete
H_ERR_MAT_READ	9218	Invalid file format for matrix
H_ERR_MAT_COMPLEX	9219	Resulting matrix has complex values
H_ERR_WMATEXP	9220	Wrong value in matrix of exponents
H_ERR_MAT_WRONG_VERSION	9221	The version of the matrix is not supported
H_ERR_MAT_NOSITEM	9222	Serialized item does not contain a valid matrix
H_ERR_WNODE	9230	Internal error: wrong Node
H_ERR_CMP_INCONSISTENT	9231	Inconsistent red black tree
H_ERR_LAPACK_PAR	9250	Internal error: Wrong LAPACK parameter
H_ERR_STRI_NPNT	9260	Number of points too small for spherical triangulation
H_ERR_STRI_COLL	9261	First three points are collinear in spherical triangulation
H_ERR_STRI_IDPNT	9262	Spherical triangulation contains identical input points
H_ERR_STRI_NALLOC	9263	Internal error: array not allocated large enough for spherical triangulation
H_ERR_STRI_DEGEN	9264	Spherical Voronoi diagram contains degenerate triangle
H_ERR_STRI_ITRI	9265	Internal error: inconsistent spherical triangulation
H_ERR_STRI_SELFINT	9266	Spherical Voronoi diagram contains self-intersecting polygon
H_ERR_STRI_INCONS	9267	Internal error: inconsistent spherical polygon data
H_ERR_STRI_AMBINT	9268	Internal error: Ambiguous great circle arc intersection
H_ERR_STRI_AMBARC	9269	Internal error: Ambiguous great circle arc
H_ERR_STRI_ILLPAR	9270	Internal error: Illegal parameter
H_ERR_TRI_NPNT	9280	Not enough points for planar triangular meshing
H_ERR_TRI_COLL	9281	The first three points of the triangular meshing are collinear
H_ERR_TRI_IDPNT	9282	Planar triangular meshing contains identical input points
H_ERR_TRI_IDPNTIN	9283	Invalid points for planar triangular meshing
H_ERR_TRI_NALLOC	9284	Internal error: allocated array too small for planar triangular meshing
H_ERR_TRI_ITRI	9285	Internal error: planar triangular meshing inconsistent
H_ERR_TRI_OUTR	9286	Node index outside triangulation range
H_ERR_TRI_LOCINC	9290	Local inconsistencies in all valid neighborhoods (parameters only allow few valid neighborhoods or point cloud not subsampled)
H_ERR_WSPVP	9300	Eye point and reference point coincide
H_ERR_DQ_ZERO_NORM	9310	Real part of the dual quaternion has length 0
H_ERR_TIMEOUT	9400	Timeout occurred
H_ERR_WRONG_TIMEOUT	9401	Invalid value for timeout

Error Name	Code	Description
H_ERR_TIMEOUT_AFTER_SBM_CLEAR	9402	Timeout occurred after cached transformations have been freed (internal error)
H_ERR_DEFORM_WRONG_NUM_CLUSTER	9450	Invalid value for 'part_size'
H_ERR_DEFORM_WRONG_NUM_MIN_SIZE	9451	Invalid value for 'min_size'
H_ERR_DEFORM_WRONG_NUM_LSQ	9452	Invalid number of least-squares iterations
H_ERR_DEFORM_WRONG_ANGLE_STEP	9453	Invalid value for 'angle_step'
H_ERR_DEFORM_WRONG_SCALE_R_STEP	9454	Invalid value for 'scale_r_step'
H_ERR_DEFORM_WRONG_SCALE_C_STEP	9455	Invalid value for 'scale_c_step'
H_ERR_DEFORM_WRONG_MAX_ANGLE	9456	Invalid value for 'max_angle_distortion'
H_ERR_DEFORM_WRONG_MAX_ANISO	9457	Invalid value for 'max_aniso_scale_distortion'
H_ERR_DEFORM_WRONG_MIN_SIZE	9458	Invalid value for 'min_size'
H_ERR_DEFORM_WRONG_COV_POSE_MODE	9459	Invalid value for 'cov_pose_mode'
H_ERR_DEFORM_NO_CALIBRATION_INFO	9460	Model contains no calibration information
H_ERR_DEFORM_WRONG_PARAM_NAME	9461	Generic parameter name does not exist
H_ERR_DEFORM_IMAGE_TO_CAMERA_DIFF	9462	Provided camera parameters have different resolution than image
H_ERR_DEFORM_NO_MODEL_IN_FILE	9463	Invalid file format for deformable model
H_ERR_DEFORM_WRONG_VERSION	9464	The version of the deformable model is not supported
H_ERR_DEFORM_WRONG_SMOOTH_DEFORM	9465	Invalid 'deformation_smoothness'
H_ERR_DEFORM_WRONG_EXPAND_BORDER	9466	Invalid 'expand_border'
H_ERR_DEFORM_ORIGIN_OUTSIDE_TEMPLATE	9467	Model origin outside of axis-aligned bounding rectangle of template region
H_ERR_DEFORM_NOSITEM	9468	Serialized item does not contain a valid deformable model
H_ERR_VIEW_ESTIM_FAIL	9499	Estimation of viewpose failed
H_ERR_SFM_NO_POINTS	9500	3D Object Model has no points
H_ERR_SFM_NO_FACES	9501	3D Object Model has no faces
H_ERR_SFM_NO_NORMALS	9502	3D Object Model has no normals
H_ERR_SFM_NO_VISIBILITY	9503	3D surface model not trained for calculating view-based score
H_ERR_SFM_NO_3D_EDGES	9504	3D surface model not trained for edge-supported matching
H_ERR_SFM_NO_SFM_FILE	9506	Invalid file format for 3D surface model
H_ERR_SFM_WRONG_FILE_VERSION	9507	The version of the 3D surface model is not supported
H_ERR_SFM_NOSITEM	9508	Serialized item does not contain a valid 3D surface model
H_ERR_SFM_TOO_MANY_SYMMS	9509	Given symmetry poses generate too many symmetric poses
H_ERR_OM3D_INVALID_FILE	9510	Invalid 3D file
H_ERR_OM3D_INVALID_MODEL	9511	Invalid 3D object model
H_ERR_OM3D_UNKNOWN_FILE_TYPE	9512	Unknown file type
H_ERR_OM3D_WRONG_FILE_VERSION	9513	The version of the 3D object model is not supported
H_ERR_OM3D_MISSING_ATTRIB	9514	Required attribute missing in 3D object model
H_ERR_OM3D_MISSING_ATTRIB_V_COORD	9515	Required points missing in 3D object model
H_ERR_OM3D_MISSING_ATTRIB_V_NORMALS	9516	Required normals missing in 3D object model
H_ERR_OM3D_MISSING_ATTRIB_F_TRIANGLES	9517	Required triangulation missing in 3D object model
H_ERR_OM3D_MISSING_ATTRIB_F_LINES	9518	Required polylines missing in 3D object model
H_ERR_OM3D_MISSING_ATTRIB_F_TRINEIGB	9519	Required triangle neighborhood missing in 3D object model
H_ERR_OM3D_MISSING_ATTRIB_F_POLYGONS	9520	Required polygons missing in 3D object model
H_ERR_OM3D_MISSING_ATTRIB_V_2DMAP	9521	Required 2D mapping missing in 3D object model
H_ERR_OM3D_MISSING_ATTRIB_O_PRIMITIVE	9522	Required primitive missing in 3D object model
H_ERR_OM3D_MISSING_ATTRIB_SHAPE_MODEL	9523	Required 3D shape model missing in 3D object model
H_ERR_OM3D_MISSING_ATTRIB_EXTENDED	9524	Required extended attribute missing in 3D object model
H_ERR_OM3D_NOSITEM	9525	Serialized item does not contain a valid 3D object model
H_ERR_OM3D_MISSING_O_PRIMITIVE_EXTENSION	9526	Primitive in 3D object model has no extended data
H_ERR_OM3D_CONTAIN_ATTRIB_F_TRIANGLES	9527	Operation invalid, 3D object model already contains triangles
H_ERR_OM3D_CONTAIN_ATTRIB_F_LINES	9528	Operation invalid, 3D object model already contains lines
H_ERR_OM3D_CONTAIN_ATTRIB_F_POLYGONS	9529	Operation invalid, 3D object model already contains faces or polygons
H_ERR_OM3D_ISOLATED_OBJECT	9530	For at least one input 3D object model no neighbor with sufficient surface overlap is available
H_ERR_OM3D_SET_ALL_COORD	9531	All components of points must be set at once

Error Name	Code	Description
H_ERR_OM3D_SET_ALL_NORMALS	9532	All components of normals must be set at once
H_ERR_OM3D_NUM_NOT_FIT_COORD	9533	Number of values does not correspond to number of already existing points
H_ERR_OM3D_NUM_NOT_FIT_NORMALS	9534	Number of values does not correspond to number of already existing normals
H_ERR_OM3D_NUM_NOT_FIT_TRIANGLES	9535	Number of values does not correspond to already existing triangulation
H_ERR_OM3D_NUM_NOT_FIT_POLYGONS	9536	Number of values does not correspond to length of already existing polygons
H_ERR_OM3D_NUM_NOT_FIT_LINES	9537	Number of values does not correspond to length of already existing polylines
H_ERR_OM3D_NUM_NOT_FIT_2DMAP	9538	Number of values does not correspond to already existing 2D mapping
H_ERR_OM3D_NUM_NOT_FIT_EXTENDED	9539	Number of values does not correspond to already existing extended attribute
H_ERR_OM3D_FACE_INTENSITY_WITH_POINTS	9540	Per-face intensity is used with attribute 'points'
H_ERR_OM3D_ATTRIBUTE_NOT_SUPPORTED	9541	At least one attribute is not supported
H_ERR_OM3D_NOT_IN_BB	9542	No point within bounding box
H_ERR_DIF_TOO_SMALL	9543	distance_in_front is smaller than the 'Resolution'
H_ERR_MINTH_TOO_SMALL	9544	The minimum thickness is smaller than the surface tolerance
H_ERR_OM3D_WRONG_DIMENSION	9545	Input width or height does not match the number of points in 3D object model
H_ERR_OM3D_MISSING_DIMENSION	9546	Image width or height must be set to compute the XYZ-mapping
H_ERR_SF_OM3D_TRIANGLES_NOT_SUITABLE	9550	Triangles of the 3D object model are not suitable for this operator
H_ERR_SF_OM3D_FEW_POINTS	9551	Too few suitable 3D points in the 3D object model
H_ERR_NO_SERIALIZED_ITEM	9580	Invalid file format for serialized items
H_ERR_END_OF_FILE	9581	Serialized item: premature end of file
H_ERR_SID_WRONG_RESIZE_METHOD	9600	Invalid value for 'image_resize_method'
H_ERR_SID_WRONG_RESIZE_VALUE	9601	Invalid value for 'image_resize_value'
H_ERR_SID_WRONG_RATING_METHOD	9602	Invalid value for 'rating_method'
H_ERR_SID_NO_IMAGE_INFO_TYPE	9603	At least one type of image information must be added
H_ERR_SID_MODEL_NO_COLOR	9604	Sample identifier does not contain color information
H_ERR_SID_MODEL_NO_TEXTURE	9605	Sample identifier does not contain texture information
H_ERR_SID_NO_IMAGE_INFO	9606	Sample image does not contain enough information
H_ERR_SID_NO_UNPREPARED_DATA	9607	Sample identifier does not contain unprepared data (use add_sample_identifier_preparation_data)
H_ERR_SID_MODEL_NOT_PREPARED	9608	Sample identifier has not been prepared yet (use prepare_sample_identifier)
H_ERR_SID_NO_UNTRAINED_DATA	9609	Sample identifier does not contain untrained data (use add_sample_identifier_training_data)
H_ERR_SID_MODEL_NOT_TRAINED	9610	Sample identifier has not been trained yet (use train_sample_identifier)
H_ERR_SID_NO_RESULT_DATA	9611	Sample identifier does not contain result data
H_ERR_SID_NUM_TRAIN_OBJ	9612	Sample identifier must contain at least two training objects (use add_sample_identifier_training_data)
H_ERR_FINI_USR_THREADS	9700	More than one user thread still uses HALCON resources during finalization
H_ERR_NO_ENCRYPTED_ITEM	9800	Invalid file format for encrypted items
H_ERR_WRONG_PASSWORD	9801	Wrong password
H_ERR_ENCRYPT_FAILED	9802	Encryption failed
H_ERR_DECRYPT_FAILED	9803	Decryption failed

Appendix B

HALCON Semantic Types

In the following, the HALCON semantic types are listed. For more information about semantic types, see section 2.3.3 on page 29.

angle
any
arc
attribute
barcode
barrier
bead_inspection_model
bg_estimation
calib_data
camera_setup_model
campar
chain
channel
chord
circle
circle_arc
circle_sector
class_box
class_gmm
class_knn
class_lut
class_mlp
class_svm
class_train_data
color_trans_lut
component_model
component_training
compute_device
condition
contour
coordinates
datacode_2d
deep_counting
deep_ocr
deformable_model
deformable_surface_matching_result
deformable_surface_model
descriptor_model
dev_tool
dict
distribution
dl_classifier
dl_classifier_result
dl_classifier_train_result
dl_device
dl_layer
dl_model
dl_pruning
drawing_object
dual_quaternion
ellipse
ellipse_arc
ellipse_sector
encrypted_item
event
exception
extent
feature_set
file
filename
framegrabber
function_1d
generic_shape_model_result
gnuplot
grayval
handle
hesseline
histogram
hom_mat2d
hom_mat3d
iline
image
info
integer
io_channel
io_device
lexicon
line
matrix
measure
memory_block
message
message_queue
metrology_model
mutex
ncc_model
number
object
object_model_3d
ocr_box
ocr_cnn
ocr_knn
ocr_mlp
ocr_svm
ocv
operator_set
point
point3d
pointer
polygon
pose
proc_name
quaternion
real
rectangle
rectangle2
region
sample_identifier
scanner
scattered_data_interpolator
scene_3d
serial
serialized_item
shape_model
shape_model_3d
sheet_of_light_model
socket
stereo_model
string
structured_light_model
surface_matching_result
surface_model
system
template
text_model
text_result
texture_inspection_model
texture_inspection_result
thread_id
tuple
untyped_object
user
variation_model
window
xld
xld_cont
xld_dist_trans
xld_ext_para
xld_mod_para
xld_para
xld_poly

Index