






# 1 HALCON Script

## 1.1 Aufgabe

Führt HALCON-Programme aus.

## 1.2 Parameter

  **HALCON Script** 




**Parameter**

Ausführungsbedingung:

Für Inspektionsverarbeitung verwenden: ☒

Upload:  **Browse...**

### HALCON Script - Parameter

  **HALCON Script** 

**Parameter**

Ausführungsbedingung:

Für Inspektionsverarbeitung verwenden: ☒

Upload:  **Browse...**

### HALCON Script - Parameter

Parameter	Beschreibung	Datentyp
-----------	--------------	----------





Parameter	Beschreibung	Datentyp
<b>Ausführungsbedingung</b>	Bedingung, die erfüllt sein muss, damit das Tool ausgeführt wird. Es können entweder konstante Werte eingegeben werden ( <i>true</i> um das Tool immer auszuführen, <i>false</i> um es nie auszuführen) oder Verknüpfungen mit Ergebnissen vorangegangener Tools. Wenn ein Tool nicht ausgeführt wird, werden seine Ausgänge auf Standardwerte gesetzt: <ul style="list-style-type: none"><li>• Arithmetische Typen: 0</li><li>• Strings: leerer String</li><li>• Bilder: leeres Bild</li></ul>	Bool
<b>Für Inspektionsverarbeitung verwenden</b>	Bestimmt, ob sich das Ausführungsergebnis des Tools auf das Inspektionsergebnis auswirkt.	Bool
<b>Upload</b>	Eingabefeld, mit welchen HALCON-Programme und Daten ausgewählt und geladen werden können.  HALCON-Programme müssen die Datei-Endung ".hdev" haben. Die Upload-Dateigröße eines HALCON-Programms ist auf <b>1 MB</b> limitiert.	


Das Tool führt die Main-Prozedur des Programmes während der Initialisierung (sobald ein Skript geladen wird) und die lokale "HalconRun()" Prozedur (immer wenn das Tool abgearbeitet wird) aus. Aus diesem Grund darf die Main-Prozedur keine Fehler enthalten. Des Weiteren

- müssen Parameter gemäß eines vorgegebenen Namensschemas zugewiesen sein (siehe Expertenwissen),
- das Speichern und Verwenden weiterer lokaler Prozeduren im HALCON-Programm ist erlaubt und
- die Verwendung der HALCON-Verschlüsselung beim Programm und Prozeduren ist möglich.

## 1.3 Ergebnisse



































Ergebnis	Beschreibung	Datentyp	Standardwert	Toleranz einstellbar
<b>Toolverarbeitung</b>	<ul style="list-style-type: none"><li>• <b>Erfolgreich</b> : Es ist kein Fehler aufgetreten.</li><li>• <b>Fehlerhaft</b> : Es ist ein Fehler bei der Bildaufnahme aufgetreten</li></ul>			 (Fehler)
<b>Ergebnisnachricht</b>	Textuelle Beschreibung der Fehlerursache	String		 (Fehler)



Ergebnis	Beschreibung	Datentyp	Standardwert	Toleranz einstellbar
<b>Ausführungszeit [ms]</b>	Die Zeit, die allein dieses Tool zur Ausführung benötigt. Beinhaltet nicht die Zeit während der lediglich auf ein Triggersignal gewartet wird.	Double	0	 (Fehler)

## 1.4 Expertenwissen

### 1.4.1 Übersicht über die HALCON-Parameter

Typen	Iconic / Control	Postfix / Name	Input	Output (Ergebnis)	Standard Wert 1	Standard Farbe 1	Farbe	Transformati
Bild	I	<i>_Img</i>	 (Haken)	 (Haken)				 (Haken)
AOI	I	<i>_Aoi</i>	 (Haken)		 (Haken)	 (Haken)	 (Haken)	 (Haken)
Region	I	<i>_Region</i>	 (Haken)	 (Haken)		 (Haken)	 (Haken)	 (Haken)
Maske	I	<i>Mask</i>	 (Haken)			 (Haken)	 (Haken)	 (Haken)
Contour (=XLD <sup>2</sup> )	I	<i>_Contour</i>		 (Haken)		 (Haken)	 (Haken)	 (Haken)
Integer	C	<i>_Int</i>	 (Haken)	 (Haken)	 (Haken)			
MinMax Integer	C	<i>_MInt</i>	 (Haken)		 (Haken)			
Real	C	<i>_Real</i>	 (Haken)	 (Haken)	 (Haken)			
MinMax Real	C	<i>_MReal</i>	 (Haken)		 (Haken)			
String	C	<i>_String</i>	 (Haken)	 (Haken)	 (Haken)			



Typen	Iconic / Control	Postfix / Name	Input	Output (Ergebnis)	Standard Wert 1	Standard Farbe 1	Farbe	Transformati
Boolean	C	<i>_Bool</i>	✓ (Haken)	✓ (Haken)	✓ (Haken)			
Checkbox	C	<i>_MBool</i>	✓ (Haken)		✓ (Haken)			
Enum	C	<i>_Enum</i>	✓ (Haken)		✓ (Haken)			
Rechteck	C	<i>_Rect</i>	✓ (Haken)	✓ (Haken)	✓ (Haken)	✓ (Haken)	✓ (Haken)	✓ (Haken)
Virtual Image	C	<i>_VImg</i>		✓ (Haken)				
Non Transformed Image	I	<i>_NImg</i>		✓ (Haken)				
Object	I	<i>_Object</i>	( ✓ ) <sup>3</sup>	( ✓ ) <sup>3</sup>	✓ (Haken)			
Tuple	C	<i>_Tuple</i>	( ✓ ) <sup>3</sup>	( ✓ ) <sup>3</sup>	✓ (Haken)			
Communication	C	<i>_Com</i>	( ✓ ) <sup>3</sup>	( ✓ ) <sup>3</sup>				
Serialized Object	C	<i>_Serialized</i>	( ✓ ) <sup>3</sup>	( ✓ ) <sup>3</sup>				
ArrayCount	C	<i>ArrayCount</i>	( ✓ ) <sup>3</sup>					
Toolergebnis	C	<i>ToolResult</i>		( ✓ ) <sup>3</sup>				
Runmodus	C	<i>RunMode</i>	( ✓ ) <sup>3</sup>					
System Informationen	C	<i>SystemInfo</i>	( ✓ ) <sup>3</sup>					

<sup>1</sup> Standard-Werte werden in der Main-Prozedur des HALCON-Programms definiert und einmalig beim Laden des HALCON-Programms gesetzt.

<sup>2</sup> XLD steht für "eXtended Line Description" und umfasst alle *Contour* und *Polygon* basierte Daten.

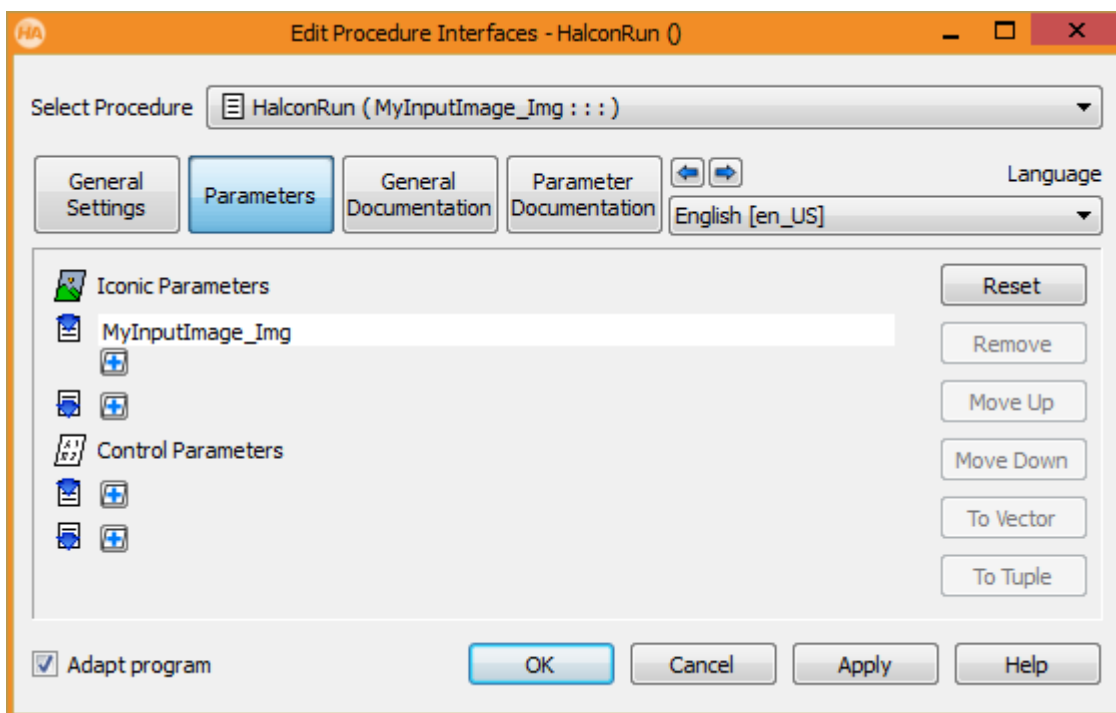
<sup>3</sup> Diese Typen werden weder als Text noch als grafisches Element dargestellt.

## Regeln für HALCON-Parameter beachten

Damit Parameter richtig angewandt werden, gibt es einige Regeln zu beachten:

- Parameter in HALCON müssen nach dem Schema `<name>_<postfix>` benannt sein. Hierbei wird der Teil `<name>` im Tool als Parameter/Ergebnis zur Verfügung gestellt.  
Ausnahmen von diesem Schema sind im Folgenden beschrieben. Einige Parameter haben auch feste Namen (z.B. Mask).
- Input und Output (Parameter oder Ergebnis) werden über HALCON definiert.
- Wird am Ende des Parameters ein `_` hinzugefügt, dann wird die Richtung der Variable invertiert (siehe Invertierte Parameter).
- Die Anzahl an Parameter pro Prozedur ist durch HALCON limitiert.
  - Control: 20 (je In/Out)
  - Iconic: 9 (je In/Out)
- Parameter mit gleichen Namen (normal / invertiert) referenzieren auf dieselbe Variable.
- Input-Strings sollten nicht leer sein.




Um beispielweise ein Input-Bild zu definieren, kann in HDevelop ein Iconic input Parameter mit dem Namen **"MyInputImage\_Img"** hinzugefügt werden.



### HDevelop - MyInputImage\_Img

Dieser erscheint dann als Input-Parameter im Tool:



  **HALCON Script** 

---

**Parameter**




Ausführungsbedingung:

Für Inspektionsverarbeitung verwenden: ☒

Upload:  **Browse...**

MyInputImage:

### HALCON Script - MyInputImage

  **HALCON Script** 

---

**Parameter**

Ausführungsbedingung:

Für Inspektionsverarbeitung verwenden: ☒

Upload:  **Browse...**

MyInputImage:

### HALCON Script - MyInputImage

Beim Laden eines HALCON-Programms werden

- Parameter in der Reihenfolge, wie diese in "*HalconRun()*" definiert werden, hinzugefügt. Danach folgen Parameter aus weiteren Prozeduren ("*HalconInit()*", "*HalconFinalize()*", "Callbacks").
- Stimmen Parameter beim Laden eines anderen Programms überein, dann werden diese wiederverwendet.  
Hierbei verharren zwar die wiederverwendeten Parameter auf Ihrer Position, jedoch können neue Parameter davor, dahinter oder zwischen wiederverwendeten Parametern platziert werden.
- Wiederverwendete Parameter behalten beim Laden eines neuen Programms ihren Wert, sofern dieser nicht durch einen Standardwert in "*main()*" überschrieben wird.



## Mit HALCON-Parametern arbeiten

### Integer, Real, String, Boolean

<i>Variablen</i>	<i>Postfix</i>	<i>Beispiel</i>
		<pre>* default value in main() MyIntValue_Int := 123  * set value in HalconRun() MyIntValue_Int := 456</pre>
<ul style="list-style-type: none"><li>Booleans enthalten 1 für true und 0 für false</li></ul>		

### MinMax Integer und Real

<i>Variablen</i>	<i>Postfix</i>	<i>Beispiel</i>
	<pre>_MInt_MinMax _MReal_MinMax</pre>	<pre>* set minimum and maximum in main() Test_MInt_MinMax := [-10, 20] * default value in main() Test_MInt := 3  * set value in HalconRun() Test_MInt := 5</pre>
<ul style="list-style-type: none"><li>Im Beispiel muss <i>Test_MInt</i> vorhanden sein.</li><li>⚠ MReal und ist in der aktuellen Beta nicht enthalten</li></ul>		

### AOI

<i>Variablen</i>	<i>Postfix</i>	<i>Beispiel</i>
	<pre>_Values</pre>	<pre>* default values in main() MyAOI_Aoi_Values := [100, 200, 300, 400.567] * set default color in main() MyAOI_Aoi_Color := [255, 255, 0, 255]  * usage in HalconRun() reduce_domain(Image, MyAOI_Aoi, ImageReduced)</pre>



- AOIs werden als Region in "*HalconRun()*" übergeben.
- Default-Werte werden als Tuple [x,y,width,height] in "*main()*" gesetzt.

## Rechteck

<i>Variablen</i>	<i>Postfix</i>	<i>Beispiel</i>
Tuple [x,y,width,height,angle]	<i>_Values</i>	<pre>* default values in main() MyRectangle_Rect_Values := [100, 200, 300, 400, 55.55] * set default Color in main() MyRectangle_Rect_Color := [255, 255, 0 , 255]  * set value in HalconRun() MyRectangle_Rect := [101, 202, 303, 40 4, 66.66]</pre>
<ul style="list-style-type: none"><li>• Rechtecke können als Input, Output und invertiert verwendet werden.</li><li>• Die Angabe des Rotationswinkels ist optional, Rechtecke können mit vier oder fünf Werten gesetzt werden.</li><li>• x, y, w, h sind im Koordinatensystem der Kamerasoftware definiert.</li><li>• Der Rotationswinkel wird in Grad angegeben, Referenz für die Roation ist x,y.</li><li>• Rechtecke können <i>Transformation</i> und <i>Color</i> als zusätzliche Parameter haben.</li><li>• <i>Color</i> kann als Standardparameter gesetzt werden.</li></ul>		

## Enum

<i>Variablen</i>	<i>Postfix</i>	<i>Beispiel</i>
Tuple [index,string]	<i>_Enum</i> <i>_Enum_Values</i>	<pre>* define enum values in main() color_Enum_Values := ['red', 'green', 'blue'] * set default value in main() color_Enum := 'green'  * usage in HalconRun() color_Enum[1]</pre>





<i>Variablen</i>	<i>Postfix</i>	<i>Beispiel</i>
<ul style="list-style-type: none"><li>• <i>Enum</i> kann nur Input sein.</li><li>• Bei ungültigen Werten wird -1 als Index zurückgegeben.</li></ul>		

## Tuple

<i>Variablen</i>	<i>Postfix</i>	<i>Beispiel</i>
	<i>_Tuple</i>	<pre>test_Tuple := [123, 1.23, 'this is a string inside a tuple']</pre>
<ul style="list-style-type: none"><li>• <i>Tuple</i> können als Input, Output und invertiert verwendet werden.</li><li>• <i>Tuple</i> werden nicht im HALCON Tool angezeigt.</li><li>• <i>Tuple</i> können multiple Werte und Handles (Serialization, Model,... ) enthalten.</li></ul>		

## Object

<i>Variablen</i>	<i>Postfix</i>	<i>Beispiel</i>
	<i>_Object</i>	<pre>gen_circle (test_Object, 200, 200, 100.5)</pre>
<ul style="list-style-type: none"><li>• <i>Objects</i> können als Input, Output und invertiert verwendet werden.</li><li>• <i>Objects</i> werden nicht im HALCON Tool angezeigt.</li><li>• <i>Objects</i> können alles enthalten, was einem ikonischer Parameter zugeordnet werden kann.</li></ul>		

## Farbe

<i>Variablen</i>	<i>Postfix</i>	<i>Beispiel</i>
Tuple [R,G,B] oder [R,G,B,A]	<i>_Color</i>	<pre>* parameter MyAOI_Aoi  * default value in main() MyAOI_Aoi_Color := [255, 0, 0, 2 55]  * usage in HalconRun() MyAOI_Aoi_Color := [255, 0, 0, 2 55]</pre>



<i>Variablen</i>	<i>Postfix</i>	<i>Beispiel</i>
<ul style="list-style-type: none"><li>Für einige Parameter (<i>AOIs</i>, <i>Regionen</i>, etc.) kann ein zusätzlicher <i>Farb</i>parameter hinzugefügt werden, mit dem bei jedem Lauf die <i>Farbe</i> verändert oder ausgelesen werden kann.</li><li>In der "<i>main()</i>" Prozedur kann die Farbe ebenfalls gesetzt werden. Diese Farbe wird einmalig nach dem Laden des Scripts übernommen.</li><li>Input-<i>Regionen</i> erhalten Ihre Farbe von der verlinkten <i>Region</i>. Hier kann die <i>Farbe</i> nicht gesetzt werden.</li><li>Für Invertierte Parameter kann die <i>Farbe</i> ebenfalls ausgelesen bzw. verändert werden. Der Farbparameter muss dann auch invertiert verwendet werden (siehe invertierte Parameter).</li></ul>		

## Transformationen

<i>Variablen</i>	<i>Postfix</i>	<i>Beispiel</i>
Tuple (6D)	<i>_Trafo</i>	<pre>* parameter output_Region * transformation parameter output_Region_Trafo  * init identity trafo hom_mat2d_identity (HomMat2DIdentity) * set rotation hom_mat2d_rotate (HomMat2DIdentity, 0.78, 0, 0, output_Region_Trafo)</pre>
<ul style="list-style-type: none"><li>Für einige Parameter (z.B. <i>Bilder</i>, <i>AOIs</i>, etc.) kann ein zusätzlicher <i>Transformations</i>parameter angegeben werden, mit dem eine zugehörige Transformation gesetzt oder ausgelesen werden kann</li><li><i>Transformationen</i> werden als 6D-Tuple übergeben (siehe HALCON Dokumentation für <i>hom_mat2d_*</i>).</li><li>Wird bei einem Input Parameter der <i>Transformations</i>parameter übergeben, dann werden darin die Transformationsparameter z.B. aus dem Tool "<b>Objekt finden</b>" übergeben.</li><li>Für Invertierte Parameter kann die <i>Transformation</i> ebenfalls ausgelesen bzw. verändert werden. Der Transformationsparameter muss dann auch invertiert verwendet werden (siehe invertierte Parameter).</li></ul>		

## Kalibrierung

<i>Variablen</i>	<i>Postfix</i>	<i>Beispiel</i>
Tuple	<i>_Calib</i>	<pre>* image parameter MyInput_Img * calibration parameter name MyInput_Img_Calib</pre>



- Für Bildparameter kann ein zusätzlicher Kalibrierparameter angelegt werden, mit dem die zugehörigen Kalibrierdaten gelesen oder gesetzt werden können.
- Die Kalibrierdaten enthalten entweder
  - keinen Wert: Das Bild hat keine Kalibrierung.
  - 15 Werte: Das Bild hat eine Kalibriertransformation ("**Kamera einrichten**" Kalibrierung anwenden auf: Ergebnis oder Bild), die wie folgt angewendet werden kann:

```
* extract camera parameters
camPar := MyInput_Img_Calib[0:7]
* extract pose
pose := MyInput_Img_Calib[8:14]

* sample image coordinates in pixel
row := 123
col := 456

* transform to millimeter
image_points_to_world_plane (camPar, pose, row, col, 'mm', X_in_mm, Y_in_mm)
```

- einen Wert: Skalierungsfaktor von Pixel in Millimeter.

 nicht in der aktuellen Beta Version

## Transformation Link

<i>Variablen</i>	<i>Postfix</i>	<i>Beispiel</i>
	<i>_TrafoLink</i>	<pre>* set transformation link in main() input_aoi_Aoi_TrafoLink := 'input_image_Img'</pre>
<ul style="list-style-type: none"><li>• Die Transformation einiger Parameter kann mit der Transformation anderer Parameter verknüpft werden.</li><li>• Beispielsweise kann ein AOI mit einem Bild verknüpft werden: <code>&lt;name_of_destination_parameter&gt;_TrafoLink := '&lt;name_of_source_parameter&gt;'</code></li></ul>		


## Arrays / ArrayCount

<i>Variablen</i>	<i>Postfix</i>	<i>Beispiel</i>
	<i>_&lt;type&gt;Array</i>	<pre>* set default in main() ArrayCount := 3 ArrayCount_MinMax := [3, 8]</pre>



<i>Variablen</i>	<i>Postfix</i>	<i>Beispiel</i>
		<pre>* set value in HalconRun() Test_IntArray := [0, 1, 2, 3, 4, 5]</pre>
<ul style="list-style-type: none"><li>• <i>Arrays</i> können als Input, Output und invertiert verwendet werden.</li><li>• <i>Arrays</i> sind möglich bei <i>Int</i>, <i>Real</i>, <i>String</i>, <i>Bool</i> und <i>MInt</i>.</li><li>• <i>ArrayCount</i> legt die Anzahl der Elemente aller Arrays fest</li><li>• Für jedes <i>Array</i> werden einzelne Elemente mit dem Namen <i>&lt;name&gt; &lt;number&gt;</i> im Tool erzeugt.</li><li>• <i>ArrayCount</i> kann als Input Parameter in "<i>HalconRun()</i>" verwendet werden.</li><li>• Bei MinMax von <i>ArrayCount</i><ul style="list-style-type: none"><li>• beträgt die maximale Länge 32,</li><li>• ist <i>ArrayCount</i> unsichtbar, sobald Min und Max den gleichen Wert haben.</li></ul></li><li>• Übersteigt die Anzahl der Werte die definierte Grenze von <i>ArrayCount</i>, dann werden überzählige Werte verworfen.</li><li>• Fehlende Werte werden mit Standard-Werten aufgefüllt (Zero / leerer String).</li></ul>		

## Optionen

<i>Variablen</i>	<i>Postfix</i>	<i>Beispiel</i>
	<i>_Options</i>	<pre>* set options in main() output_int_Int_Options := ['UnitMM']</pre>
<ul style="list-style-type: none"><li>• Folgende <i>Optionen</i> können für Parameter vorgegeben werden: <i>UnitMM</i>, <i>UnitDEG</i>, <i>UnitPercent</i>, <i>UnitPixel</i>, <i>Disabled</i>, <i>UseAsResult</i>.</li><li>• Einheiten werden nur von numerischen Output Werten (einschließlich Arrays) unterstützt. Ein Parameter kann nur eine Einheit haben.</li><li>• Es können mehrere Optionen gesetzt werden, sofern diese vom jeweiligen Parametertyp unterstützt werden.</li><li>• Für Inputparameter kann die Option <i>UseAsResult</i> gesetzt werden. Inputparameter mit dieser Option werden dann auch in der Ergebnisliste dargestellt und können für den Aktionsmenu <b>Monitor</b> selektiert werden oder von nachfolgenden Tools als Input verwendet werden. Diese Option wird nicht von Bildparametern, Arrayparameter und nicht sichtbaren Parametern (z.B. Tuple) unterstützt.</li></ul> <p> <i>UseAsResult</i> ist nicht in der aktuellen Betaversion.</p>		



## Virtual Image

<i>Variablen</i>	<i>Postfix</i>	<i>Beispiel</i>
	<code>&lt;out_name&gt;_&lt;in_name&gt;_VImg</code>	<div><pre>* name of input image parameter example_input_Img  * name of virtual image parameter example_ref_example_input_VImg</pre></div>
<ul style="list-style-type: none"><li>• Hierbei muss das Input Bild mit dem Namen <code>&lt;in_name&gt;_Img</code> vorhanden sein.</li><li>• Das Verhalten ähnelt der Arbeitsweise der Tools "<b>Objekt finden</b>" und "<b>Code lesen</b>".</li></ul>		

## Virtual Image Array

<i>Variablen</i>	<i>Postfix</i>	<i>Beispiel</i>
	<code>&lt;out_name&gt;_&lt;in_name&gt;_VImgArray</code>	<div><pre>* name of input image parameter example_input_Img  * name of virtual image parameter array example_ref_example_input_VImgArray</pre></div>
<ul style="list-style-type: none"><li>• Funktioniert wie <i>Virtual Image</i> nur als Array.</li><li>• 6D <i>Transformationen</i> Tuple erfordern 6 x <i>ArrayCount</i> Werte.</li><li>• Zusätzliche <i>Transformationen</i> werden verworfen.</li><li>• Fehlende <i>Transformationen</i> werden zu Identitätsabbildung (keine Rotation und keine Translation).</li><li>• Unvollständige <i>Transformationen</i> (weniger als sechs Werte) werden ebenfalls zur Identitätsabbildung.</li></ul>		

## Non Transformed Image

<i>Variablen</i>	<i>Postfix</i>	<i>Beispiel</i>
	<code>_NImg</code>	
<ul style="list-style-type: none"><li>• Wird wie <i>Image</i> verwendet, geht aber nur als Output</li><li>• Kann als Input für das Tool "<b>Objekt finden</b>" verwendet werden.</li><li>• <i>Transformation</i> ist nicht möglich.</li></ul>		



## ReferencelImage

<i>Variablen</i>	<i>Postfix</i>	<i>Beispiel</i>
	<code>&lt;name&gt;_&lt;name_of_input_image&gt;_RImg</code>	<div><pre>* name of input image parameter example_input_Img  * name of reference image parameter example_ref_example_input_RImg</pre></div>
<ul style="list-style-type: none"><li>• <i>ReferencelImage</i> kann nur Input sein.</li><li>• Hierbei wird ein Button im Tool eingeblendet und bei Betätigung wird ein Referenzbild aufgenommen.</li></ul>		

## Invertierte Parameter

<i>Variablen</i>	<i>Postfix</i>	<i>Beispiel</i>
	—	<div><pre>* Standard Input Parameter inputInt_Int  * Output for HALCON, Input for HALCON tool inputInt_Int_</pre></div>
<ul style="list-style-type: none"><li>• Mit invertierten Parametern kann in ein Input Parameter geschrieben oder von einem Output Parameter gelesen werden (vorgesehen für Callbacks, "<i>HalconInit()</i>" und "<i>HalconFinalize()</i>" oder um einen Parameter durchzureichen).</li><li>• In der gleiche Prozedur können Parameter sowohl normal als auch invertiert sein, d.h. ein alter Wert kann einen neuen Wert erzeugen (durchreichen).</li></ul>		

## Kommunikationsparameter

<i>Variablen</i>	<i>Postfix</i>	<i>Beispiel</i>
Tuple	<code>_Com</code> <code>_Com_Format</code>	<div><pre>* set communication main() testCommunication_Com_Format := ['i', 'r', 's32']  * usage in HalconRun() testCommunication_Com := [123, 456.789, 'teststring value']</pre></div>



<i>Variablen</i>	<i>Postfix</i>	<i>Beispiel</i>
		<ul style="list-style-type: none"><li>• Ermöglicht das Senden und Empfangen von Daten ähnelt der Arbeitsweise der Tools "<b>Ergebnisse senden</b>" und "<b>Daten empfangen</b>".</li><li>• Kann nicht invertiert verwendet werden.</li><li>• Output: Wird an SPS gesendet.</li><li>• Input: Wird aus SPS gelesen.</li><li>• Möglichst Formate, die im Tuple definiert werden können:<ul style="list-style-type: none"><li>• 'i' =&gt; int32</li><li>• 'r' =&gt; real32</li><li>• 's####' =&gt; String mit Länge (zu lange Strings werden abgeschnitten, zu kurze mit '\0' aufgefüllt)</li></ul></li><li>• Enthält <code>_Com</code> im Verhältnis zu <code>_Com_Format</code> zu wenig oder zu viele Elemente, dann werden entsprechend Werte verworfen oder mit Standardwerten aufgefüllt.</li><li>• Bei Output-Werten wird eine Inspektions-ID (int32) als erster Wert hinzugefügt.</li><li>• Die Länge des Outputs darf 64kB nicht überschreiten.</li></ul>

## Serialisierte Parameter

<i>Variablen</i>	<i>Postfix</i>	<i>Beispiel</i>
	<code>_Serialized</code>	<pre>* Serialization, parameter used as output inverted serialize_object (Region, region_Serialized_)  * Deserialization: parameter used as input deserialize_object (Region, region_Serialized)</pre>
		<ul style="list-style-type: none"><li>• Kann verwendet werden, um serialisierte HALCON-Daten im Tool zu speichern.</li><li>• Serialisierte Parameter enthalten die Referenz zur Serialisierung.</li><li>• Serialisierung und Deserialisierung werden über das HALCON-Programm gemacht, während das Speichern und Laden im Flash über das HALCON-Tool automatisch passiert.</li><li>• Bei einer als Parameter übergebene Serialisierung darf im HALCON-Programm kein Cleanup gemacht werden.</li><li>• Falls etwas nicht Deserialisiert werden kann, dann ist das Tuple leer.</li><li>• Mehrere Elemente können mit einem einzelnen Serialized-Parameter verwendet werden.</li><li>• Serialisierte Daten können mit einem Upload-Button hochgeladen werden.</li></ul>

## ToolResult

<i>Variablen</i>	<i>Postfix</i>	<i>Beispiel</i>
Tuple [string] oder [string,string]		<pre>ToolResult := ['warn', 'Could not find peak']</pre>



<i>Variablen</i>	<i>Postfix</i>	<i>Beispiel</i>
<ul style="list-style-type: none"><li>1. Variable des Tuples ist die Nachricht für " Toolverarbeitung " (mögliche Werte sind "<b>ok</b>", "<b>warn</b>", "<b>error</b>" and "<b>critical</b>").</li><li>2. Variable des Tuples ist die Nachricht für " Ergebnismnachricht ".</li></ul>		

## ToolName

<i>Variablen</i>	<i>Postfix</i>	<i>Beispiel</i>
		<pre>* set tool name in main() ToolName := 'MyFirstHalconScriptTool'</pre>
<ul style="list-style-type: none"><li>Setzt den Namen des Tools beim Laden des Scripts.</li></ul>		

## InfoString

<i>Variablen</i>	<i>Postfix</i>	<i>Beispiel</i>
		<pre>* set info string in main() InfoString := 'This is my first info string'</pre>
<ul style="list-style-type: none"><li>Setzt einen Informationstext im Input Bereich des Tools.</li></ul>		

## Skriptversion

<i>Variablen</i>	<i>Postfix</i>	<i>Beispiel</i>
Tuple		<pre>* set the script version in main() ScriptVersion := 1</pre>
<ul style="list-style-type: none"><li>Setzt die Version des HALCON-Skripts. Das HALCON-Tool prüft diese Version beim Laden eines HALCON-Programms und erzeugt einen Warndialog falls die Version aus dem Programm nicht zur Version aus dem Tool passt. Die Versionen werden ebenfalls beim Update der Kamerasoftware geprüft</li></ul>		






<i>Variablen</i>	<i>Postfix</i>	<i>Beispiel</i>
<ul style="list-style-type: none"><li>Die aktuelle Skriptversion des HALCON-Tools ist 1.</li><li>Für HALCON-Programme ohne Angabe der Skriptversion wird Version 1 angenommen.</li></ul> <p>⚠ nicht in der aktuellen Beta</p>		

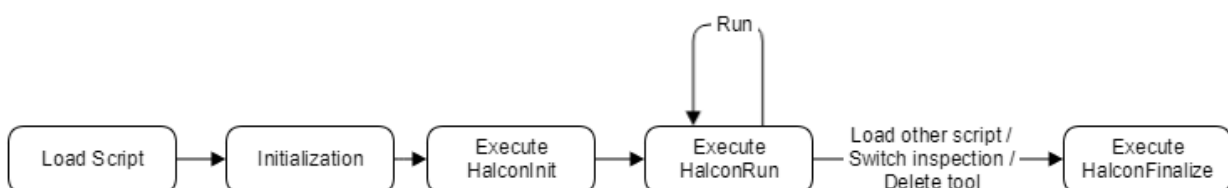
## Callback

<i>Variablen</i>	<i>Postfix</i>	<i>Beispiel</i>
Tuple [string,...]		<pre>* list names of local procedures to be used as callbacks in main() HalconCallbacks := ['Test1', 'Test2']</pre>
<ul style="list-style-type: none"><li>In der Main-Prozedur können HALCON Prozeduren hinzugefügt werden, die als Buttons im Tool einscheinen.</li><li>Die HALCON Prozeduren werden aufgerufen, wenn auf den Button geklickt wird.</li><li>Dies kann auch zum Initialisieren von Variablen genutzt werden.</li><li>Parameter-Handhabung ist wie bei "<i>HalconRun()</i>".</li><li>Parameter mit gleichen Namen referenzieren auf dieselbe Variable.</li><li>Werden im Callback Inputparameter verändert, aktualisiert sich das Tool und führt ein Durchlauf von "<i>HalconRun()</i>" aus.</li></ul>		

## Maske

<i>Variablen</i>	<i>Name</i>	<i>Beispiel</i>
	<i>Mask</i>	
<ul style="list-style-type: none"><li>Ist ein <i>Region</i> Parameter, der sich wie die <i>Maske</i> im Tool "<b>Objekt finden</b>" verhält.</li><li><i>Maske</i> kann nur Input (und invertiert) sein, ist zeichenbar und muss den Namen <b>Mask</b> haben.</li><li>Es ist nur eine <i>Maske</i> möglich.</li><li>Enthält ein HALCON-Programm einen <i>Maske</i> Parameter, dann erscheint das Wizard Icon .</li></ul>		

## Init/Finalize






<i>Variablen</i>	<i>Postfix</i>	<i>Beispiel</i>
<ul style="list-style-type: none"><li>Die HALCON Prozeduren "<i>HalconInit()</i>" und "<i>HalconFinalize()</i>" können definiert werden, um Initialisierungen beim Laden eines Inspektionsprogramms und Cleanup vor dem Entladen eines Inspektionsprogramms durchzuführen.</li><li>"<i>HalconInit()</i>" wird nach dem Start der Kamera, beim Wechsel des Inspektionsprogramms und nachdem ein neues HALCON-Programm im Konfigurationsmodus geladen wird aufgerufen.</li><li>"<i>HalconFinalize()</i>" wird vor dem Wechsel des Inspektionsprogramms und bevor ein neues HALCON-Programm im Konfigurationsmodus geladen wird aufgerufen.</li><li>Parameter-Handhabung ist wie bei "<i>HalconRun()</i>".</li><li>Parameter in Init, Finalize oder Run mit gleichen Namen referenzieren auf dieselbe Variable.</li></ul>		

## RunMode

<i>Variablen</i>	<i>Name</i>	<i>Beispiel</i>
	<i>RunMode</i>	
<ul style="list-style-type: none"><li>Gibt an, ob das HALCON-Programm im Aktionsbereich <b>Monitor</b> (Wert = 1) oder in einem anderen Modus läuft (<b>Konfiguration</b>, Initialisierung; Wert = 0).</li></ul>		

## Systeminformationen

<i>Variablen</i>	<i>Name</i>	<i>Beispiel</i>
Tuple	<i>SystemInfo</i>	
<ul style="list-style-type: none"><li>Enthält folgende Informationen zur Kamera und zur Inspektion:<ol style="list-style-type: none"><li>ID des Inspektionsprogramms</li><li>ID des Inspektionsergebnis</li><li>Zeitstempel des Inspektionsergebnisses (String)</li><li>Farbunterstützung der Kamera (1 = Farbkamera (oder PC-Version), 0 = Grauwertkamera)</li><li>Geräteinformation (String)</li><li>Information über I/O-Erweiterungen (String)</li></ol></li><li>Auch auf Grauwertkameras können über das File Device Farbbilder geladen werden.</li></ul> <p> Nicht in der aktuellen Beta enthalten</p>		

## (Modell)-Daten laden

Neben HALCON-Programmen können auch Modell-Daten wie Schriften oder Klassifizierungen geladen werden. Hierbei gilt Folgendes zu beachten:



- Jede Art von serialisierten HALCON-Daten können in einen serialisierte Parameter geladen werden, wobei der Dateiname mit dem Namen des serialisierten Parameters beginnen muss.
- Falls die serialisierte Datei mehr als ein Element enthält, dann muss vorher ein Tuple mit der Anzahl an Elementen serialisiert werden.
- Jede Dateiergung außer ".hdev" und ".zip" ist erlaubt.
- Die Daten müssen in einem serialisierten HALCON-Format zur Verfügung stehen.

Folgendes Beispiel zeigt, wie eine serialisierte Region erzeugt werden kann:

```
gen_circle (Circle, 500, 500, 100.5)
serialize_object (Circle, SerializedItemHandle)
open_file ('test_Serialized.bin', 'output_binary', FileHandle)
length := 1
serialize_tuple (length, SerializedItemHandle1)
fwrite_serialized_item (FileHandle, SerializedItemHandle)
close_file (FileHandle)
clear_serialized_item (SerializedItemHandle1)
clear_serialized_item (SerializedItemHandle)
```

## 1.4.2 HALCON Debug Server

Auf der Kamera kann der Debug Server der HDevEngine gestartet werden, mit dem sich HDevelop verbinden kann. Damit lässt sich in HDevelop der ScriptCode auf der Kamera debuggen. Folgende Punkte gilt es zu beachten:

- Es muss auf der Kamera ein HALCON-Tool eingefügt worden sein.
- Im HALCON-Tool sollte das zu debuggende HALCON-Programm geladen sein.
- Anschließend kann in den Systemeinstellungen in der Kategorie *Developer* der Button *HDevEngine Debug Server: **Start*** betätigt werden.
- Abschließend kann In HDevelop über das Hauptmenü *Execute Attach To Process...* eine Verbindung mit dem Debug Server auf der Kamera hergestellt werden.

Einschränkungen:

- Es kann effektiv nur mit einem HALCON-Tool debuggt werden, da HDevelop nicht mit mehreren Prozeduren mit dem gleichen Namen zurechtkommt, aber alle Programme "*HalconRun()*" verwenden.
- Es ist effektiv nicht möglich "*HalconInit()*" oder "*main()*" zu debuggen, da das zu debuggende Programm vor der Verbindung zwischen Kamera und HDevelop geladen werden muss.
- Die Version von HDevelop muss kompatibel zur HALCON-Version auf der Kamera sein. Aktuell verwendet die Kamera 13.X.

Der Debug Server kann mit dem Button *HDevEngine Debug Server: **Stop*** auf Kamera gestoppt werden.



Nicht in der aktuellen Beta enthalten

## 1.4.3 Tipps und Hinweise

### Allgemein

- Das HALCON-Tool läuft im Aktionsbereich **Konfiguration** ein wenig langsamer als im **Monitor**.
- Die Kamerasoftware verwendet ein anderes Koordinatensystem als HALCON.
- Detaillierte Fehlerbeschreibungen finden Sie in den System-Logs.
- Ändern Sie keine Input *Images*, *Regions*, etc. Dies führt zu unerwünschten Ergebnissen vor allem im Aktionsbereich **Konfiguration**.
- Globale Variablen in HALCON-Programmen sind in allen Tools global. Die Verwendung von globalen Variablen ist allerdings kein guter Programmierstil.
- Programmteile in "*main()*", die für den Betrieb auf dem PC bestimmt sind und auf der Kamera nicht funktionieren, können in einen try/catch Block eingeschlossen werden. Damit läuft das selbe Programm auf Kamera und unter HDevelop.

### Arbeiten mit Referenzen (Handles)

- Ein Cleanup von HALCON-Referenzen (*object\_model*, etc.) ist notwendig, um Speicherlecks zu verhindern. **Ausnahme:** Referenzen zu serialisierten Parametern.
- Ein Cleanup muss für jede allokierte Referenz in der Main-Prozedur gemacht werden.
- HALCON-Referenzen müssen als Tuple-Variablen gespeichert werden, falls diese als Parameter übergeben werden.

### Verwendung von Dateien

- Versuchen Sie nicht, auf Dateien im Dateisystem zuzugreifen. Falls dies dennoch nötig sein sollte, dann verwenden Sie "*../images*", dies ist der freigegebene Bildordern.
- Ein Cleanup auf Datei-Referenzen muss gemacht werden.

### Sonderfall mvBlueGEMINI



#### WARNUNG

Verwenden Sie nicht den `system_call` Operator. Dies kann dazu führen, dass die Smart Camera nicht mehr funktioniert.

Wechseln Sie nicht das aktuelle Arbeitsverzeichnis. Dies kann dazu führen, dass die Konfiguration der Kamera fehlerhaft gespeichert wird und ein Reset der gesamten Konfiguration nötig wird.

### Sonderfall SMART CAMERA



#### WARNUNG



Verwenden Sie nicht den `system_call` Operator. Dies kann dazu führen, dass die **SMART CAMERA** nicht mehr funktioniert.

Wechseln Sie nicht das aktuelle Arbeitsverzeichnis. Dies kann dazu führen, dass die Konfiguration der Kamera fehlerhaft gespeichert wird und ein Reset der gesamten Konfiguration nötig wird.






## 2 HALCON Script

### 2.1 Task

Executes HALCON programs.

### 2.2 Parameters

  **HALCON Script** 




**Parameter**

Run condition:

Use for inspection processing: ☒

Upload:  [Browse...](#)

#### HALCON Script - parameters

  **HALCON Script** 

**Parameter**

Run condition:

Use for inspection processing: ☒

Upload:  [Browse...](#)

#### HALCON Script - parameters

Parameter	Description	Data type
<b>Run condition</b>	<p>Condition that has to be met in order for the tool to run. You can use constant values (<i>true</i> to run the tool, <i>false</i> to skip it) or links to results of other tools. If the tool did not run, the outputs of this tool will be set to default values:</p> <ul style="list-style-type: none"><li>Arithmetic types: 0</li><li>Strings: empty string</li></ul>	Bool






Parameter	Description	Data type
	<ul style="list-style-type: none"><li>Images: empty images</li></ul>	
Use for inspection processing	Determines whether the tool result has an effect on the inspection result.	Bool
Upload	Input field to select and upload HALCON programs and data.  The program files must have the ending ".hdev" and the file size limit is <b>1 MB</b> .	

The tool executes the program's main procedure during initialization (when a script is loaded) and a local procedure called "HalconRun()" while executing the tool. For this reason, the execution of main must not fail. Furthermore,

- parameters must be assigned according to the naming scheme (see expert knowledge),
- it is possible to store and use other local procedures in the HALCON program, and
- it is possible to use HALCON encryption for the program and for the procedures.

## 2.3 Results

Result	Description	Data type	Default value	Tolerance adjustable
Tool processing	<ul style="list-style-type: none"><li><b>Successful</b> : No errors occurred.</li><li><b>Failed</b> : An error occurred with image acquisition</li></ul>			 (Fehler)
Results report	Text description of the cause of the error	String		 (Fehler)
Execution time [ms]	The time this single tool needs for execution. Does not comprise time that is spent solely on waiting for a trigger.	Double	0	 (Fehler)



## 2.4 Expert knowledge

### 2.4.1 Summary of the HALCON-Parameters

Types	Iconic / Control	Postfix	Input	Output (Result)	Default Value <sup>1</sup>	Default Color <sup>1</sup>	Color	Transformation	Transformation
Image	I	<i>_Img</i>	✓ (Haken)	✓ (Haken)				✓ (Haken)	
AOI	I	<i>_Aoi</i>	✓ (Haken)		✓ (Haken)	✓ (Haken)	✓ (Haken)	✓ (Haken)	✓ (Haken)
Region	I	<i>_Region</i>	✓ (Haken)	✓ (Haken)		✓ (Haken)	✓ (Haken)	✓ (Haken)	✓ (Haken)
Mask	I	<i>Mask</i>	✓ (Haken)			✓ (Haken)	✓ (Haken)	✓ (Haken)	✓ (Haken)
Contour (=XLD <sup>2</sup> )	I	<i>_Contour</i>		✓ (Haken)		✓ (Haken)	✓ (Haken)	✓ (Haken)	✓ (Haken)
Integer	C	<i>_Int</i>	✓ (Haken)	✓ (Haken)	✓ (Haken)				
MinMax Integer	C	<i>_MInt</i>	✓ (Haken)		✓ (Haken)				
Real	C	<i>_Real</i>	✓ (Haken)	✓ (Haken)	✓ (Haken)				
MinMax Real	C	<i>_MReal</i>	✓ (Haken)		✓ (Haken)				
String	C	<i>_String</i>	✓ (Haken)	✓ (Haken)	✓ (Haken)				
Boolean	C	<i>_Bool</i>	✓ (Haken)	✓ (Haken)	✓ (Haken)				
Checkbox	C	<i>_MBool</i>	✓ (Haken)		✓ (Haken)				
Enum	C	<i>_Enum</i>	✓ (Haken)		✓ (Haken)				





Types	Iconic / Control	Postfix	Input	Output (Result)	Default Value <sup>1</sup>	Default Color <sup>1</sup>	Color	Transformation	Tool for Line
Rectangle	C	<i>_Rect</i>	✓ (Haken)	✓ (Haken)	✓ (Haken)	✓ (Haken)	✓ (Haken)	✓ (Haken)	✓ (Haken)
Virtual Image	C	<i>_VImg</i>		✓ (Haken)					
Non Transformed Image	I	<i>_NImg</i>		✓ (Haken)					
Object	I	<i>_Object</i>	(✓) <sup>3</sup>	(✓) <sup>3</sup>	✓ (Haken)				
Tuple	C	<i>_Tuple</i>	(✓) <sup>3</sup>	(✓) <sup>3</sup>	✓ (Haken)				
Communication	C	<i>_Com</i>	(✓) <sup>3</sup>	(✓) <sup>3</sup>					
Serialized Object	C	<i>_Serialized</i>	(✓) <sup>3</sup>	(✓) <sup>3</sup>					
ArrayCount	C	<i>ArrayCount</i>	(✓) <sup>3</sup>						
ToolResult	C	<i>ToolResult</i>		(✓) <sup>3</sup>					
RunMode	C	<i>RunMode</i>	(✓) <sup>3</sup>						
System Information	C	<i>SystemInfo</i>	(✓) <sup>3</sup>						

<sup>1</sup> Default values can be set in the main procedure of the HALCON program and are set only once when a new script is loaded in config mode.

<sup>2</sup> XLD stands for "eXtended Line Description" and comprises all Contour and Polygon based data.

<sup>3</sup> These types are displayed neither as text nor as graphical element.

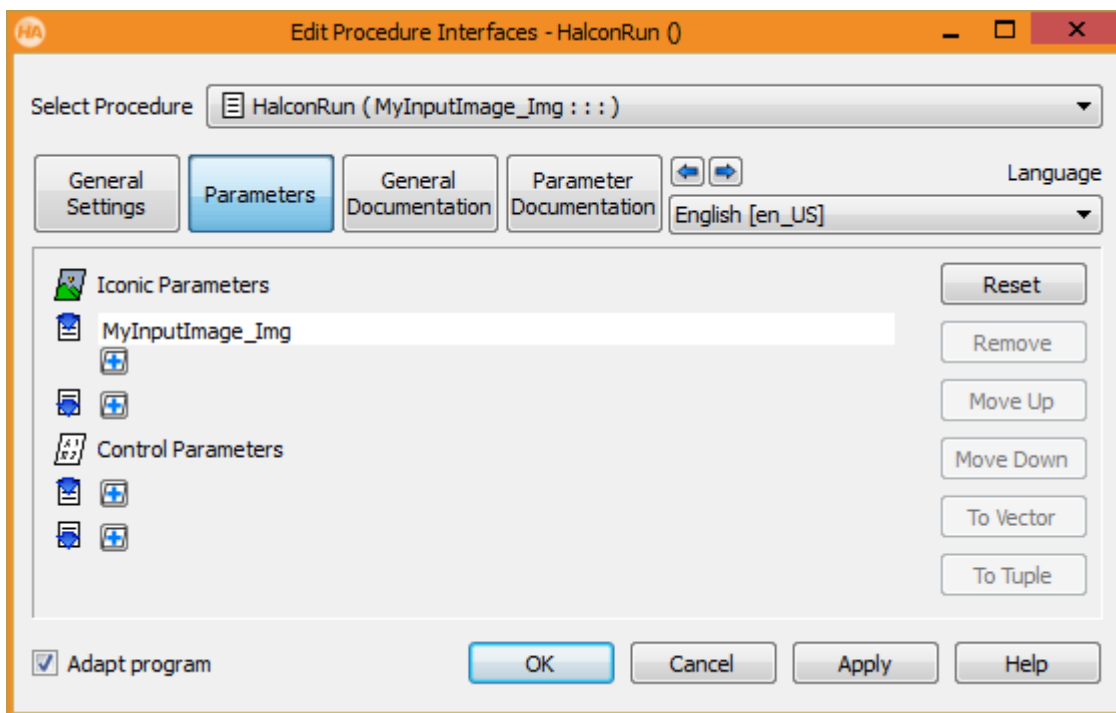
## Rules for HALCON parameters

In order to work with the parameters properly, you have to follow some rules:

- Parameters in Halcon must be named like *<name>\_<postfix>*. The *<name>* part is used as parameter /result name in the tool.  
Exceptions of this scheme are described in the following. Some parameters have fixed names (e.g. Mask).
- Input or output (parameter or result) is defined in HALCON.
- Adding an *\_* at the end of the name inverts the direction of a variable (see inverted parameters)

- Parameter count per procedure is limited by HALCON.
  - Control: 20 (In/Out each)
  - Iconic: 9 (In/Out each)
- Parameters with the same name (normal / inverted) are the same variable
- Input strings should not be empty.

To define an input image, for example, add an iconic input parameter with the name **"MyInputImage\_Img"** in HDevelop:



### HDevelop - MyInputImage\_Img

This will generate an input image parameter:

#### Parameter

Run condition:

Use for inspection processing: ☒

Upload:

MyInputImage:

### HALCON Script - MyInputImage



**HALCON Script**

---

**Parameter**

---

Run condition:

Use for inspection processing: ☒

Upload:

MyInputImage:

### HALCON Script - MyInputImage

While loading a HALCON program,

- parameters are added in the order they are defined inside the parameter settings of the "*HalconRun()*" procedure. This is followed by parameters from further procedures ("*HalconInit()*", "*HalconFinalize()*", "*Callbacks*").
- When loading another script, parameters that match are reused.  
Reused parameters stay at their position, new parameters can be added before, after or between reused parameters, but reused parameters can't switch positions.  
Reused parameters keep their value while a new program is loaded, except there are overwritten in the *main()* by a default value.

## Working with HALCON parameters

### Integer, Real, String, Boolean

Variables	Postfix	Example
		<pre>* default value in main() MyIntValue_Int := 123  * set value in HalconRun() MyIntValue_Int := 456</pre>
<ul style="list-style-type: none"><li>Booleans contain 1 for true and 0 for false</li></ul>		



## MinMax Integer and Real

Variables	Postfix	Example
	<code>_MInt_MinMax</code> <code>_MReal_MinMax</code>	<pre>* set minimum and maximum in main() Test_MInt_MinMax := [-10, 20] * default value in main() Test_MInt := 3  * set value in HalconRun() Test_MInt := 5</pre>

- In the example, `Test_MInt` must be available.



`_MReal` is not in the current Beta version

## AOI

Variables	Postfix	Example
	<code>_Values</code>	<pre>* default values in main() MyAOI_Aoi_Values := [100, 200, 300, 400.567] * set default color in main() MyAOI_Aoi_Color := [255, 255, 0, 255]  * usage in HalconRun() reduce_domain(Image, MyAOI_Aoi, ImageReduced)</pre>

- AOIs are passed to "*HalconRun()*" as regions.
- Default values are set as *Tuple* [x,y,width,height] in "*main()*".

## Rectangle

Variables	Postfix	Example
<code>Tuple [x,y,width,height,angle]</code>	<code>_Values</code>	<pre>* default values in main() MyRectangle_Rect_Values := [100, 200, 300, 400, 55.55] * set default Color in main() MyRectangle_Rect_Color := [255, 255, 0 , 255]</pre>



Variables	Postfix	Example
		<pre>* set value in HalconRun() MyRectangle_Rect := [101, 202, 303, 40 4, 66.66]</pre>
<ul style="list-style-type: none"><li>• <i>Rectangles</i> can be used as input, output and inverted.</li><li>• The pecification of an angle is optional, so <i>Rectangles</i> can be set with either four or five values.</li><li>• x, y, w, h are defined in the camera's coordinate system.</li><li>• The angle is specified in degree, rotation is around (x,y).</li><li>• <i>Rectangles</i> can have <i>Transformation</i> and <i>Color</i> as additional parameters.</li><li>• A <i>Color</i> can be set as default parameter.</li></ul>		

## Enum

Variables	Postfix	Example
Tuple [index,string]	<i>_Enum</i> <i>_Enum_Values</i>	<pre>* define enum values in main() color_Enum_Values := ['red', 'green', 'blue'] * set default value in main() color_Enum := 'green'  * usage in HalconRun() color_Enum[1]</pre>
<ul style="list-style-type: none"><li>• <i>Enum</i> parameters can only be input.</li><li>• For invalid values a -1 is passed as index.</li></ul>		

## Tuple

Variables	Postfix	Example
	<i>_Tuple</i>	<pre>test_Tuple := [123, 1.23, 'this is a string inside a tuple']</pre>
<ul style="list-style-type: none"><li>• <i>Tuple</i> can be used as input, output and inverted.</li><li>• <i>Tuple</i> are not displayed in the HALCON Tool.</li><li>• <i>Tuple</i> can hold multiple values and handles (serialization handles, model handles, ...).</li></ul>		



## Object

Variables	Postfix	Example
	<code>_Object</code>	<pre>gen_circle (test_Object, 200, 200, 100.5)</pre>
<ul style="list-style-type: none"><li>• <i>Objects</i> can be used as input, output and inverted.</li><li>• <i>Objects</i> are not displayed in the HALCON Tool.</li><li>• <i>Objects</i> can hold anything that can be assigned to an iconic variable (<i>Images</i>, <i>Regions</i>, ...).</li></ul>		

## Color

Variables	Postfix	Example
Tuple [R,G,B] oder [R,G,B,A]	<code>_Color</code>	<pre>* parameter MyAOI_Aoi  * default value in main() MyAOI_Aoi_Color := [255, 0, 0, 255]  * usage in HalconRun() MyAOI_Aoi_Color := [255, 0, 0, 255]</pre>
<ul style="list-style-type: none"><li>• For some parameters (<i>AOIs</i>, <i>Regions</i>, <i>etc.</i>), you can specify an additional <i>Color</i> parameter which you can use to change or the read the <i>Color</i> with every run .</li><li>• In the "<i>main()</i>" procedure, you can also add a <i>Color</i>, which will be applied once after loading the script.</li><li>• Input <i>Regions</i> get their color from the linked <i>Region</i>. In this case, the <i>Color</i> cannot be set.</li><li>• For inverted parameters you can also read and change the <i>Color</i>. The <i>Color</i> parameter must be inverted, too (look at inverted parameters).</li></ul>		

## Transformation

Variables	Postfix	Example
Tuple (6D)	<code>_Trafo</code>	<pre>* parameter output_Region * transformation parameter output_Region_Trafo</pre>




Variables	Postfix	Example
		<pre>* init identity trafo hom_mat2d_identity (HomMat2DIdentity) * set rotation hom_mat2d_rotate (HomMat2DIdentity, 0.78, 0, 0, output_Region_Trafo)</pre>
<ul style="list-style-type: none"><li>For some parameters (e.g. <i>Images</i>, <i>AOIs</i>, etc.), you can specify additional <i>Transformation</i> parameters, which you can use to change or the read the related <i>Transformation</i>.</li><li><i>Transformations</i> are passed as 6D Tuple (look at HALCON documentation for <i>hom_mat2d_*</i>).</li><li>When passing a <i>Transformation</i> parameter to an input parameter, in it you will pass the <i>Transformation</i> parameters from, for example, the <b>"Find object"</b> tool.</li><li>For inverted parameters you can also read and change the <i>Transformation</i>. The <i>Transformation</i> parameter must be inverted, too (look at inverted parameters).</li></ul>		

## Calibration

Variables	Postfix	Example
Tuple	_Calib	<pre>* image parameter MyInput_Img * calibration parameter name MyInput_Img_Calib</pre>
<ul style="list-style-type: none"><li>For <i>Image</i> parameters, you can specify additional <i>Calibration</i> parameters, which you can use to change or the read the related <i>Calibration</i>.</li><li><i>Calibration</i> data contain either<ul style="list-style-type: none"><li>no value: The image has no <i>Calibration</i>.</li><li>15 values: The image has <i>Calibration</i> transformation (<b>"Set up camera"</b> use calibration for: result or image), which can be used as follows:<pre>* extract camera parameters camPar := MyInput_Img_Calib[0:7] * extract pose pose := MyInput_Img_Calib[8:14]  * sample image coordinates in pixel row := 123 col := 456  * transform to millimeter image_points_to_world_plane (camPar, pose, row, col, 'mm', X_in_mm, Y_in_mm)</pre></li></ul></li><li>one value: scaling factor of pixel in millimeter. (<b>"Set up camera"</b> use calibration for: image)</li></ul>		



Variables	Postfix	Example
 not in the current Beta version		

## Transformation Link

Variables	Postfix	Example
	<code>_TrafoLink</code>	<pre>* set transformation in main() input_aoi_Aoi_TrafoLink := 'input_image_Img'</pre>
<ul style="list-style-type: none"><li>• <i>Transformations</i> of some parameters can be linked to other parameters.</li><li>• For example an AOI can be linked to an image: <code>&lt;name_of_destination_parameter&gt;_TrafoLink := &lt;name_of_source_parameter&gt;'</code></li></ul>		


## Array / ArrayCount

Variables	Postfix	Example
	<code>_&lt;type&gt;Array</code>	<pre>* set default in main() ArrayCount := 3 ArrayCount_MinMax := [3, 8]  * set value in HalconRun() Test_IntArray := [0, 1, 2, 3, 4, 5]</pre>
<ul style="list-style-type: none"><li>• <i>Arrays</i> can be input or output and used as inverted parameter.</li><li>• <i>Arrays</i> are possible in combination with <i>Int</i>, <i>Real</i>, <i>String</i>, <i>Bool</i>, and <i>MInt</i>.</li><li>• <i>ArrayCount</i> sets the number of elements of all <i>Arrays</i>.</li><li>• For each <i>Array</i> single elements with <code>&lt;name&gt; &lt;number&gt;</code> are created in the tool.<ul style="list-style-type: none"><li>• <i>ArrayCount</i> can be used as input parameter in "<i>HalconRun()</i>".</li></ul></li><li>• MinMax of <i>ArrayCount</i><ul style="list-style-type: none"><li>• has a maximum length of 32, and</li><li>• <i>ArrayCount</i> is invisible, as soon as Min and Max are equal.</li></ul></li><li>• Additional values are discarded when the number of values is higher than the number specified in <i>ArrayCount</i>.</li><li>• Missing values are filled with default values (zero / empty string).</li></ul>		





## Options

<i>Variables</i>	<i>Postfix</i>	<i>Example</i>
	<code>_Options</code>	<pre>* set options in main() output_int_Int_Options := ['UnitMM']</pre>
<ul style="list-style-type: none"><li>It is possible to set the following <i>Options</i>: <i>UnitMM</i>, <i>UnitDEG</i>, <i>UnitPercent</i>, <i>UnitPixel</i>, <i>Invisible</i>, <i>Disabled</i>, <i>UseAsResult</i>.</li><li>Only numerical output values (including arrays) support units. A parameter can only have on unit.</li><li>Several options can be set, as long as they are supported by the respective parameter type.</li><li>For Input parameters it is possible to set the option <i>UseAsResult</i>. Input parameter with this option will be displayed in the result list and can be selected in the action menu <b>Monitoring</b> or used as Input in subsequent tools. This option is not supported by <i>Image</i> parameters, <i>Array</i> parameters and invisible parameters (e.g. <i>Tuples</i>).</li></ul> <p> UseAsResult not in the current Beta version.</p>		

## Virtual Image

<i>Variables</i>	<i>Postfix</i>	<i>Example</i>
	<code>&lt;out_name&gt;_&lt;in_name&gt;_VImg</code>	<pre>* name of input image parameter example_input_Img  * name of virtual image parameter example_ref_example_input_VImg</pre>
<ul style="list-style-type: none"><li>The input image with the name <code>&lt;in_name&gt;_Img</code> must be existent.</li><li>This resembles the behavior of the tools "<b>Find object</b>" and "<b>Read code</b>".</li></ul>		

## Virtual Image Array

<i>Variables</i>	<i>Postfix</i>	<i>Example</i>
	<code>&lt;out_name&gt;_&lt;in_name&gt;_VImgArray</code>	<pre>* name of input image parameter example_input_Img  * name of virtual image parameter array</pre>



<i>Variables</i>	<i>Postfix</i>	<i>Example</i>
		<code>example_ref_example_input_VImgArray</code>
<ul style="list-style-type: none"><li>• Works like <i>Virtual Image</i>, but as an array.</li><li>• 6D <i>Transformation</i> tuples requires 6 x <i>ArrayCount</i> values.</li><li>• Additional <i>Transformations</i> are discarded.</li><li>• Missing <i>Transformations</i> are set to identity transformation (no rotation and no translation).</li><li>• Incomplete <i>Transformations</i> (less than six values) are set to identity transformation, too.</li></ul>		

## Non Transformed Image

<i>Variables</i>	<i>Postfix</i>	<i>Example</i>
	<code>_NImg</code>	
<ul style="list-style-type: none"><li>• Is used like normal <i>Image</i>, but as output only.</li><li>• Can be used as input for "Find object" tool.</li><li>• Can't have a <i>Transformation</i>.</li></ul>		

## ReferenceImage

<i>Variables</i>	<i>Postfix</i>	<i>Example</i>
	<code>&lt;name&gt;_&lt;name_of_input_image&gt;_RImg</code>	<div><p>* name of input image parameter example_input_Img</p><p>* name of reference image parameter example_ref_example_input_RImg</p></div>
<ul style="list-style-type: none"><li>• <i>ReferenceImage</i> can only be input.</li><li>• A button is displayed in the tool and if you click this button, a new reference image will be taken.</li></ul>		

## Inverted Parameters

<i>Variables</i>	<i>Postfix</i>	<i>Example</i>
	—	<div><p>* Standard Input Parameter inputInt_Int</p></div>



<i>Variables</i>	<i>Postfix</i>	<i>Example</i>
		<pre>* Output for HALCON, Input for HALCON tool inputInt_Int_</pre>
<ul style="list-style-type: none"><li>• With inverted parameters you can write to an input parameter or read an output parameter (intended for Callbacks, "<i>HalconInit()</i>", and "<i>HalconFinalize()</i>" or to build a pass-through parameter).</li><li>• A parameter can be used as normal and inverted in the same procedure, this way a parameter can be used as "pass-through" parameter, the old value can be used to generate a new value.</li></ul>		

## Communication Parameters

<i>Variables</i>	<i>Postfix</i>	<i>Example</i>
Tuple	<i>_Com</i> <i>_Com_Format</i>	<pre>* set communication main() testCommunication_Com_Format := ['i', 'r', 's32']  * usage in HalconRun() testCommunication_Com := [123, 456.789, 'teststringv alue']</pre>
<ul style="list-style-type: none"><li>• It is possible to send and receive data via communication plugin similar to the "<b>Send results</b>" and "<b>Receive data</b>" tools.</li><li>• Can't be used inverted.</li><li>• Output: Is sent via communication plugin to PLC.</li><li>• Input: Current input from communication plugin (from PLC) is read.</li><li>• Format tuple in main():<ul style="list-style-type: none"><li>• 'i' =&gt; int32</li><li>• 'r' =&gt; real32</li><li>• 's####' =&gt; string with length (string is cut at length or filled with '\0')</li></ul></li><li>• If the parameter tuple has more/less items then the format tuple, values are discarded or filled with default values.</li><li>• For output values an int32 is added as first item.</li><li>• The resulting output length must not exceed about 64kB.</li></ul>		

## Serialized Parameters

<i>Variables</i>	<i>Postfix</i>	<i>Example</i>
	<i>_Serialized</i>	<pre>* Serialization, parameter used as output inverted</pre>



<i>Variables</i>	<i>Postfix</i>	<i>Example</i>
		<pre>serialize_object (Region, region_Serialized_)  * Deserialization: parameter used as input deserialize_object (Region, region_Serialized_)</pre>
<ul style="list-style-type: none"><li>• Can be used to store any serialized HALCON data in the tool.</li><li>• Serialized parameters are control variables that contain serialization handles.</li><li>• The serialization/deserialization is done in the HALCON script, the actual saving and loading to flash is done automatically by the tool.</li><li>• When a serialized handle is passed as parameter it must not be cleaned in HALCON script.</li><li>• When something can't be deserialized the tuple is empty.</li><li>• Multiple items can be used with a single serialized parameter.</li><li>• It is possible to upload serialized data via the Upload button.</li></ul>		

## ToolResult

<i>Variables</i>	<i>Postfix</i>	<i>Example</i>
Tuple [string] or [string,string]		<pre>ToolResult := ['warn', 'Could not find peak']</pre>
<ul style="list-style-type: none"><li>• The first string sets the tool processing (valid values are "<b>ok</b>", "<b>warn</b>", "<b>error</b>", and "<b>critical</b>").</li><li>• The second string is optional and sets the result message.</li></ul>		

## ToolName


<i>Variables</i>	<i>Postfix</i>	<i>Example</i>
		<pre>* set tool name in main() ToolName := 'MyFirstHalconScriptTool'</pre>
<ul style="list-style-type: none"><li>• Sets the tool name when loading a script.</li></ul>		



## InfoString

<i>Variables</i>	<i>Postfix</i>	<i>Example</i>
		<pre>* set info string in main() InfoString := 'This is my first info string'</pre>
<ul style="list-style-type: none"><li>• Sets the an information string in the input area of the tool.</li></ul>		

## ScriptVersion

<i>Variables</i>	<i>Postfix</i>	<i>Example</i>
Tuple		<pre>* set the script version in main() ScriptVersion := 1</pre>
<ul style="list-style-type: none"><li>• Sets the version of the HALCON script. The HALCON tool checks this version while loading the HALCON program and creates a warning, if the version of the program does not match with the version of the tool. Versions will also be check during an update of the camera software.</li><li>• The current script version of the HALCON tool is 1.</li><li>• 1 will be default for HALCON programs, where the script version is not stated.</li></ul> <p> not in the current Beta version.</p>		


## Callback

<i>Variables</i>	<i>Postfix</i>	<i>Example</i>
Tuple [string,...]		<pre>* list names of local procedures to be used as callbacks in main() HalconCallbacks := ['Test1', 'Test2']</pre>
<ul style="list-style-type: none"><li>• Adds HALCON procedures which appear as Buttons in the tool.</li><li>• These procedures are called when the button is pressed.</li><li>• Can also be used to initialize variables.</li><li>• Parameters of these procedures are named the same way they are named in "<i>HalconRun()</i>".</li><li>• When a parameter has the same name in a different procedure, it refers to the same variable.</li></ul>		

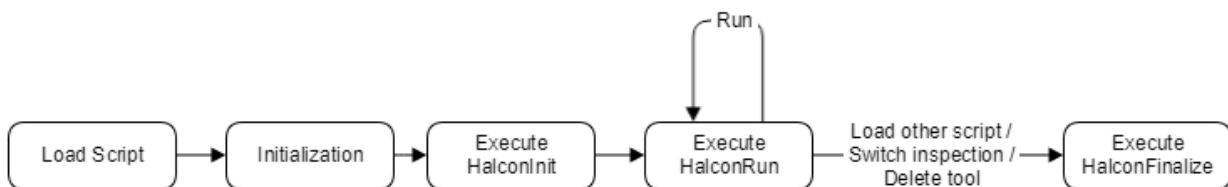


<i>Variables</i>	<i>Postfix</i>	<i>Example</i>
<ul style="list-style-type: none"><li>If input parameters are changed in the Callback, the tool will be updated and "<i>HalconRun()</i>" will be run once.</li></ul>		

## Mask

<i>Variables</i>	<i>Postfix</i>	<i>Example</i>
<ul style="list-style-type: none"><li>Is a <i>Region</i> parameter, that is handled as mask like in the "<b>Find object</b>" tool.</li><li><i>Mask</i> can be input (or used inverted), is drawable, and the name must be <b>Mask</b>.</li><li>Only one <i>Mask</i> is possible.</li><li>When a HALCON script with a mask parameter is loaded, the wizard icon appears .</li></ul>		

## Init/Finalize



<i>Variables</i>	<i>Postfix</i>	<i>Example</i>
<ul style="list-style-type: none"><li>You can define HALCON procedures "<i>HalconInit()</i>" and "<i>HalconFinalize()</i>" to make initializations as soon as an inspection program is loaded and cleanups when a program is unloaded.</li><li>"<i>HalconInit()</i>" will be called after starting of the camera, when changing an inspection program, and after a new inspection program was loaded in the action menu <b>Configuration</b>.</li><li>"<i>HalconFinalize()</i>" will be called before changing an inspection program and before a new inspection program was loaded in the action menu <b>Configuration</b>.</li><li>Parameters of init and finalize are named the same way they are named in "<i>HalconRun()</i>".</li><li>When a variable has the same name in init, finalize or run, it refers to the same variable.</li></ul>		


## RunMode

<i>Variables</i>	<i>Name</i>	<i>Example</i>
	<i>RunMode</i>	



<i>Variables</i>	<i>Name</i>	<i>Example</i>
<ul style="list-style-type: none"><li>Indicates the script is running in <b>Monitoring</b> (value = 1) or in other modes (Configuration, Initialization; value = 0).</li></ul>		

## System Information

<i>Variables</i>	<i>Name</i>	<i>Example</i>
Tuple	<i>SystemInfo</i>	
<ul style="list-style-type: none"><li>Contains following information about the camera and the inspection:<ol style="list-style-type: none"><li>ID of the inspection program</li><li>ID of the inspection result</li><li>Timestamp of the inspection result (string)</li><li>Color support of the camera (1 = color camera (or PC version), 0 = gray scale camera)</li><li>Device information (string)</li><li>Information about I/O extensions (string)</li></ol></li><li>It is possible to load color images via the File Device on gray scale camera, too.</li></ul> <p> not in the current Beta version.</p>		

## Loading (model) data

Beside HALCON programs, it is also possible to upload model data like fonts or classifications. Please note the following:

- Any kind of serialized HALCON data can be loaded in a serialized parameter.
- The data file name must start with the name of the serialized parameter.
- If the serialized file contains more then one item, a tuple with the numer of items must be serialized before the other items.
- File names can have any extension, except ".hdev" and ".zip".
- The data must be in the serialized HALCON format.

Example, how to generate a serialized region:

```
gen_circle (Circle, 500, 500, 100.5)
serialize_object (Circle, SerializedItemHandle)
open_file ('test_Serialized.bin', 'output_binary', FileHandle)
length := 1
serialize_tuple (length, SerializedItemHandle1)
fwrite_serialized_item (FileHandle, SerializedItemHandle)
close_file (FileHandle)
clear_serialized_item (SerializedItemHandle1)
clear_serialized_item (SerializedItemHandle)
```



## 2.4.2 HALCON Debug Server

You can start the debug server of HDevEngine on the camera, HDevelop can connect with. Thus you can debug the script code with HDevelop on the camera. The following must be considered:

- The HALCON tool must be added to your inspection program.
- In the HALCON tool, the HALCON program you want to debug, must be loaded.
- Afterwards, you can click on the button *HDevEngine Debug Server: **Start*** in the System settings category *Developer*.
- Finally, you can connect to the debug server on the camera via the main menu *Execute Attach To Process...* in HDevelop.

Limitations:

- You can only debug one HALCON tool, given that HDevelop cannot deal with more than one procedures with the same name (but all programs use "HalconRun()").
- It is not possible to debug "*HalconInit()*" or "*main()*", given that the program, you want to debug, must be loaded before the connection between camera and HDevelop.
- The version of HDevelop must be compliant to the HALCON version of the camera. The camera uses 13.X currently.

You can stop the debug server using the button *HDevEngine Debug Server: **Stop*** on the camera.

⚠ not in the current Beta version.

## 2.4.3 Tips and tricks

### General

- The tool runs a bit slower in action menu **Configuration** than in **Monitoring**.
- The camera software uses a different coordinate system as HALCON.
- You can find detailed error descriptions in the system log.
- Do not modify input *Images*, *Regions*, etc. This will lead to undesired results, especially in the action menu **Configuration**.
- Global variables in HALCON programs are global in all tools. However, the usage of global variables is not a good programming style.
- Part of programs in "*main()*", which are thought for working on a PC and not on the camera, can be put in a try/catch block. Thus, the same program will run on the camera and under HDevelop.

### Working with Handles

- You have to cleanup halcon handles (*object\_model*, etc.) otherwise this will cause memory leaks.  
**Exception:** serialization handles for serialized parameters.
- You have to cleanup any handles that are allocated in the main procedure.
- HALCON handles must be stored in tuple variables when passed as parameters.



## Usage of files

- Try not to access files at all, use “*../images*” if you have to. This is the shared images directory.
- You have to cleanup file handles.

## Special case mvBlueGEMINI



### WARNING

Do not use the `system_call` operator, you can easily make your Smart Camera unusable.

Do not change the current working directory. This can cause the configuration of the camera to contain errors and a reset of the complete configuration can be necessary.

## Special case SMART CAMERA



### WARNING

Do not use the `system_call` operator, you can easily make your **SMART CAMERA** unusable.

Do not change the current working directory. This can cause the configuration of the camera to contain errors and a reset of the complete configuration can be necessary.