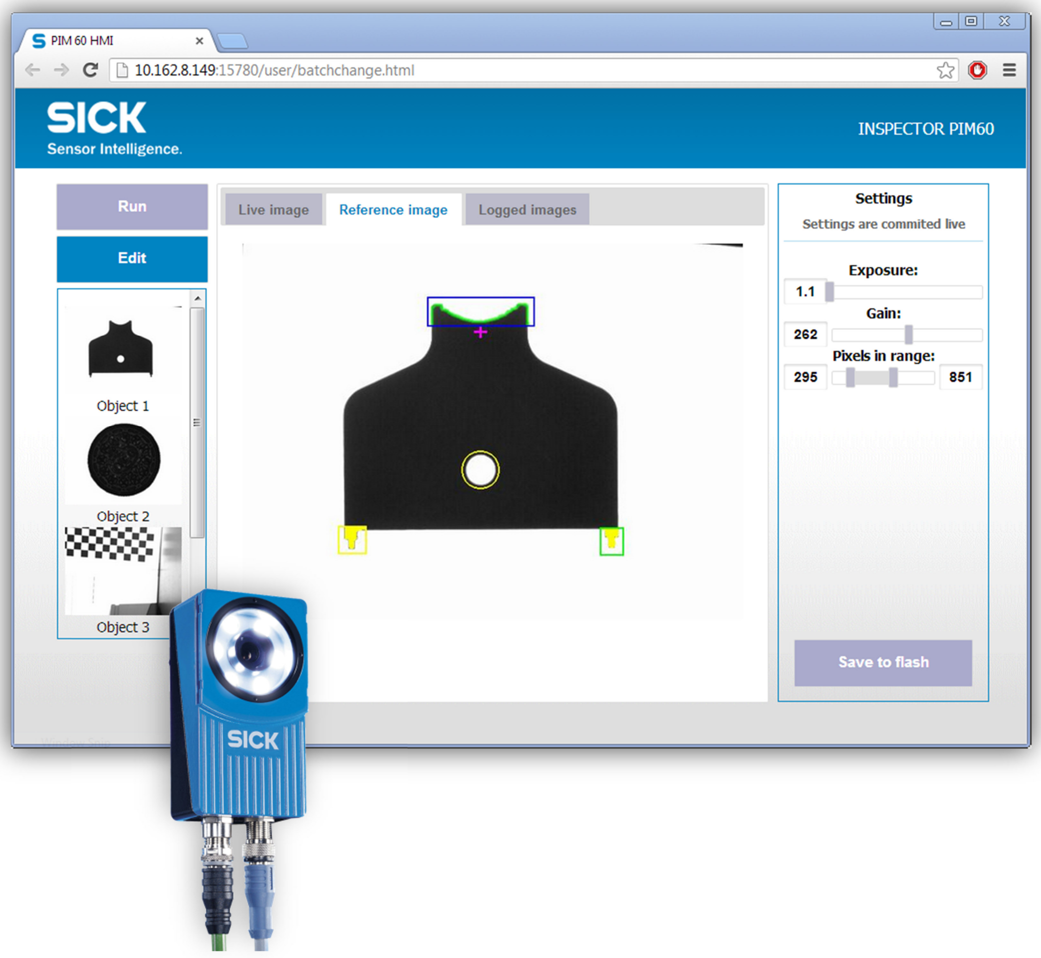


MANUAL

SICK Inspector PIM60: Custom Web Toolkit 2.0

SICK
Sensor Intelligence.

DISCLAIMER

This package is delivered "as is". SICK AG do not provide any warranty of the items whatsoever, whether express, implied, or statutory, including, but not limited to, any warranty of merchantability or fitness for a particular purpose or any warranty that the contents of the item will be error-free.

In no respect shall SICK AG incur any liability for any damages, including, but limited to, direct, indirect, special, or consequential damages arising out of, resulting from, or any way connected to the use of the item, whether or not based upon warranty, contract, tort, or otherwise; whether or not injury was sustained by persons or property or otherwise; and whether or not loss was sustained from, or arose out of, the results of, the item, or any services that may be provided by SICK AG.

Table of Contents

1 Custom Web Toolkit

The Custom Web Toolkit is a JavaScript library for building custom Human-Machine Interfaces (HMI) for the SICK PIM60 Vision Sensor in a web browser. It has an easy-to-use API and works across a multitude of browsers. The library contains graphical controls (widgets) for interactive editing of parameters of the camera configuration and viewing live imagery and log data. It's built as an extension on top of the Web API delivered with the camera. Basic knowledge of HTML, JavaScript and CSS is recommended to use this documentation, as well as some working knowledge of web development.

1.1 Workflow

A Web HMI can be seen as an operators interface. It can provide useful information, making it easy for the operator to monitor the current application and to make small adjustments if needed. An application first needs to be configured as usual with SOPAS. Only after that the HMI can be designed, based on the scenarios the operator will encounter and the identified parameters related to the application. The strong advantage of a customized HMI when done right is that it will make the end user more confident in your system. The user can recognize a familiar "look&feel" of the design in the users native language. It simplifies common tasks in the application and it also prohibits the operator from damaging the system. This toolkit will allow you to easily modify the included templates for an application, and also build your own more complex systems on top of this framework. However, using this toolkit is no guaranty of achieving a good HMI. It will only simplify the task at hand. The recommended workflow is presented in below.

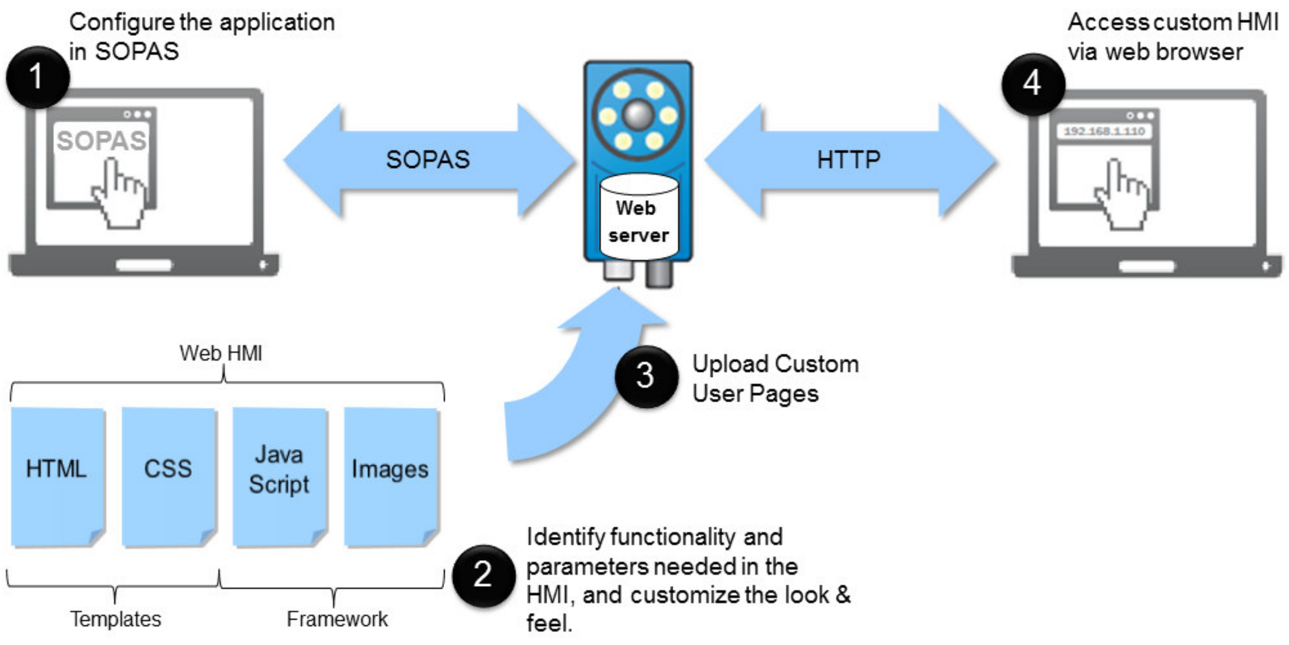

Configure the application in SOPAS
1
SOPAS
SOPAS
Web server
HTTP
Access custom HMI via web browser
4
192.168.1.110
3
Upload Custom User Pages
Web HMI
HTML
CSS
Java Script
Images
Templates
Framework
2
Identify functionality and parameters needed in the HMI, and customize the look & feel.

Figure 1. The common workflow when designing Web HMIs using this Toolkit

1.2 Contents

This package contains the following components:

- Framework
- Examples and Templates
- Documentation

1.2.1 Framework Files

The framework consists of the files listed below. These are the only files needed when building a complete custom HMIs, not based on any of the templates included in the package. It is not recommended to modify these files, if you are not a skilled web developer.

Filename	Description
`ajax-loader.gif`	Animated icon shown when an action is pending
`inspector.js*`	Updated version of the original Inspector Communication library
`inspector-ui.js`	Contains all widgets
`inspector-ui.css`	Stylesheet needed for widgets
`excanvas.min.js`	Canvas emulator for Internet Explorer versions lower than 9
`flot.time.js`	JavaScript plotting library for jQuery
`jquery.float.js`	
`jquery-1.8.3.min.js`	Minified jQuery library that has been modified to support a flat folder structure and reduced number of required images
`jquery-ui-1.9.2.custom.min.js`	
`jquery-ui-1.9.2.custom.min.css`	

** Note: This is not the same file as the one delivered with the camera.*

1.2.2 Example and Template Files

The toolkit contains a number of examples with can also be used as templates for building your own HMIs. The files used by the templates are listed below.

Filename	Description
`batchchange.html`	HTML for the batch change template
`index.html`	HTML for the landing page with live image and links to the examples
`maintenance.html`	HTML for the maintenance template
`logo.png`	Company logo (SICK logo)
`supervision.html`	HTML for the supervision example
`resultplot.html`	HTML for result plotting example
`template.css`	Stylesheet
`ui-icons_0073ea_256x240.png`	UI icon collection with in different hover and disabled states
`ui-icons_454545_256x240.png`	
`ui-icons_666666_256x240.png`	
`ui-icons_ff0084_256x240.png`	
`ui-icons_ffffff_256x240.png`	

1.2.3 Documentation

In addition to this file the documentation also contains a HTML file showing all widgets.

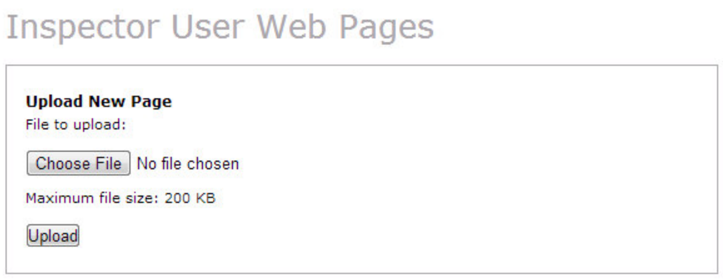
Filename	Description
`Custom Web Toolkit Manual.pdf`	This file

`SICK Inspector PIM Reference Manual.pdf`	The Web API command channel manual, included for reference

1.3 Installation

The library is designed to be uploaded to and run from the built-in web server of the Inspector. To install the library and the templates follow the instructions below. See the user manual for more details on file uploading.

1. Unzip the files in the library
2. Goto http://<ip-of-inspector>/userpages
3. Login in the box on the right side of the page using the SOPAS password. By default this is:
    a. Username: Maintenance
    b. Password: Inspector
4. Click *Choose File*, select a single file to upload and press *Upload*


Inspector User Web Pages

Upload New Page
File to upload:

Maximum file size: 200 KB

5. In a couple of seconds the file will be uploaded.
6. Repeat step 4-5.

All files in the framework folder are required for the framework to function. If you are using the toolkit for the first time we recommend you to upload all files in the template folder too, to try them all out.

If you make a change to any file and like to upload a new version you have to delete the file from the server first. Files can only be uploaded one-by-one. We recommend to keep all file dependencies on the camera and not linking to any resources on the Internet.

The Inspector requires a flat folder structure and a maximum file size of 200Kb. The total size of all files is 2Mb.

1.4 Technical background

The framework is based on the JavaScript framework jQuery (http://jquery.com/) and jQuery UI (http://jqueryui.com/) framework. Most controls are implemented as widgets in jQuery UI and are using AJAX technology to not require the page to reload for changes to be made. The communication with the camera is commenced with XMLHttpRequest messages over the web command channel described in the Inspector PIM60 Reference Manual. The auto initialization of widgets read the custom data-* attributes of the elements at page load and initializes a widget with these properties.

1.5 System Requirements

- SICK Inspector PIM60 version 2.0
- Screen resolution of client browser at least 1024x800 pixels

1.5.1 Supported Browsers

The framework has been tested on Microsoft Windows 7 in the following browsers:

- Microsoft Internet Explorer 9
- Google Chrome 26
- Mozilla Firefox 19

It should work in the following browsers:

- Microsoft Internet Explorer 6 (Move regions does not work)
- All versions of
  - Google Chrome
  - Mozilla Firefox
  - Opera
  - Safari

It is recommended to use at least version 9 of Internet Explorer with this toolkit, or preferably using Chrome, Firefox, Opera or Safari. In general Internet Explorer is really slow when it comes to running JavaScript compared to other browsers.

Different browsers render web pages a bit different, therefore it is recommended to test the the HMI with the browser that is going to be used in production.

2 Widgets

Widgets are graphical controls that display information to the user and in most cases the user is able to interact with them to change a configuration parameter. Before using any widget, make sure that the camera has been correctly configured in SOPAS. What widgets to use are highly application specific and the HMI needs to be thoughtfully designed.

2.1 Initialization

Note
All URLs on the Inspector are case sensitive.

To use the framework the following files needs to be included in your HTML header:

```
<link rel="stylesheet" href="jquery-ui-1.9.2.custom.min.css">
<link rel="stylesheet" type="text/css" href="inspector-ui.css">

<script src="jquery-1.8.3.min.js"></script>
<script src="jquery-ui-1.9.2.custom.min.js"></script>
<script src="inspector.js"></script>
<script src="inspector-ui.js"></script>
```

The order in which the famework files are included is very important and should always be in the same order as shown above.

2.2 Parameter Adjustment Controls

After including the references above, a widget to control a specific configuration parameter of a tool can be created by inserting a `<div>` tag to you HTML. The style class has to be `hmi-parameter` and a `data-id` should be specified to select which parameter to control. The id to use can be found in the PIM60 Reference Manual appendix B. To for example control the exposure add the following element to your html:

```
<div class="hmi-parameter" data-id="14"></div>
```

The widget will automatically adapt itself depending on the type of parameter being accessed.

Some parameters can only be controlled when the Inspector is in ‘Edit’ mode, others only in ‘Run’ mode. See the PIM60 Reference Manual for which are available in each mode. It’s good practice to hide or disable the widget when inapplicable. Read more in section 2.8.

If the configuration has several tools the index of the tool must be specified in the `data-tool` attribute to indicate which tool to modify:

```
<div class="hmi-parameter" data-id="14" data-tool="1"></div>
```

The index corresponds to the order in which the pixel counter, edge pixel counter, pattern, edge locator, circle locator, measure distance, and measure angle are listed in the **Tools** tab in the **SOPAS Single Device**. See the PIM60 Reference Manual section B.2.4 for more information. The tool index can be directly acquired by hovering over the item in this list. A tooltip will show you the tool index (called Device index in SOPAS terminology).

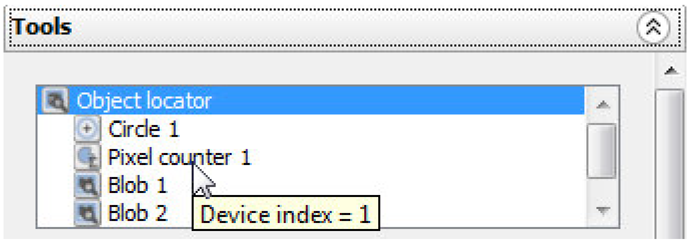


Figure 2 The Tool Index can be found in SOPAS

2.2.1 Slider

Numeric parameters will produce a slider where the user can select a value by dragging the slider or entering a value in the textbox.

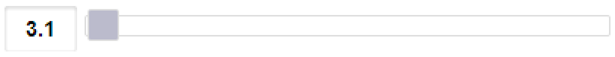


Figure 3 Example of a slider widget, here shown for Exposure parameter

The min and max of the slider will be set automatically to the limits specified in the command channel reference. To override the default settings set the `data-min` and `data-max` attributes, as shown below.

```
<div class="hmi-parameter" data-id="14" data-min="3" data-max="9"
></div>
```

2.2.2 Range slider

Numeric parameters with both a lower and upper limit will produce a range slider where the user can select values by dragging the handles of the slider or entering values in the text-boxes.

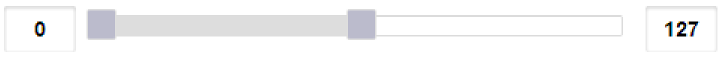


Figure 4 Example of a range slider, here shown for an intensity threshold

The min and max of the slider will be set automatically to the limits specified in the command channel reference. To override the default settings set the `data-min` and `data-max` attributes, as shown below.

```
<div class="hmi-parameter" data-id="14" data-min="3" data-max="9"
></div>
```

2.2.3 Radio

Specifying the id of a multiple choice configuration parameter in data-id will produce a radio button set widget. It will allow the user to select a value from the multiple choices.

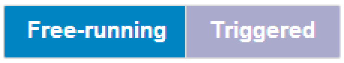


Figure 5 Example of a radio button set

Here is a code example on how to control the trigger mode:

```
<div class="hmi-parameter" data-id="16"></div>
```

2.2.4 Checkbox

A single on/off parameter will produce a checkbox widget. The label is set by the contents of the <div>-tag:

```
<div class="hmi-parameter" data-id="33">Allow Rotation</div>
```

Figure 6 Example of a checkbox

2.3 Monitor

The current value of any parameter can be displayed using the following html:

```
<span class="hmi-monitor" data-id="14"/>
```

The value is read only and cannot be changed by the user. The value will be updated every second.

2.4 Images

Live, reference and log images can be retrieved in much the same way as the controls above. Use an `<img>`-tag instead of `<div>` and use the corresponding style class of `hmi-liveimage`, `hmi-refimage` or `hmi-logimage` instead.

The size of the image can be set with standard html height and width attributes or preferably CSS width and height style attributes, for example:

```
<img class="hmi-liveimage" height="240" width="320"/>
```

The image size fetched from the camera will always be the same no matter how large the displayed image is, so showing a smaller image will not allow for a quicker update rate than a larger image. It is however possible to change the resolution of the live image which can increase the update rate, see example in 2.4.1.

2.4.1 Live image

The live image from the Inspector can be displayed by adding the following tag to your html `<img class="hmi-liveimage"/>`.

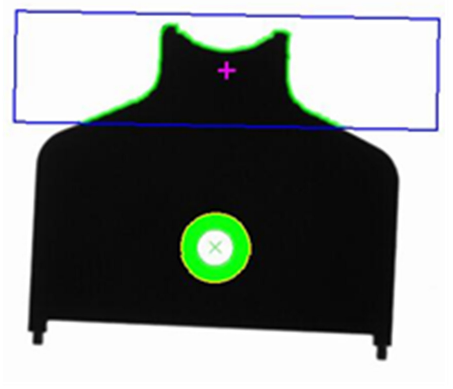
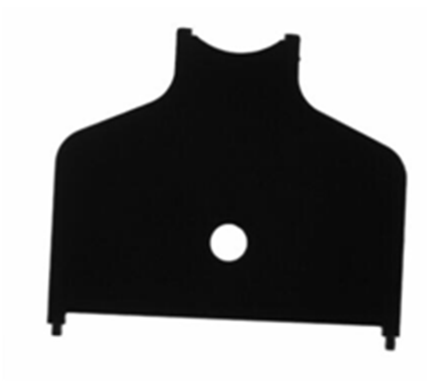
The rate of the image update can be specified in milliseconds with the `data-interval` attribute. Example: `<img class="hmi-liveimage" data-interval="600"/>`

Setting a value of less than 500ms is not recommended and will result in choppy playback and the camera not responding for extended time periods. Default value is 700ms if no other rate is specified. If multiple live images are displayed they will share the same resources and the rate might be further reduced.

To increase the update rate it is possible to downsample the image, reducing the size to 320x240, this is done with the `data-scale` attribute. Example:

```
<img class="hmi-liveimage" data-scale="2"/>
```

The overlay graphics can optionally be displayed by setting the `data-overlay` attribute to either “hide”, “show” or “simplified”.

With overlay
`<img class="hmi-liveimage" data-overlay="show"/>`

No overlay
`<img class="hmi-liveimage"/>`

Figure 7 Example of a live image with and without overlay

The update of the image will in most cases be suspended whenever the image is not visible to the operator and resumed once visible again.

2.4.2 Reference image

The current active reference image can be displayed by adding the following tag to you html: `<img class="hmi-refimage"/>`.

Reference images are always displayed with overlay graphics.

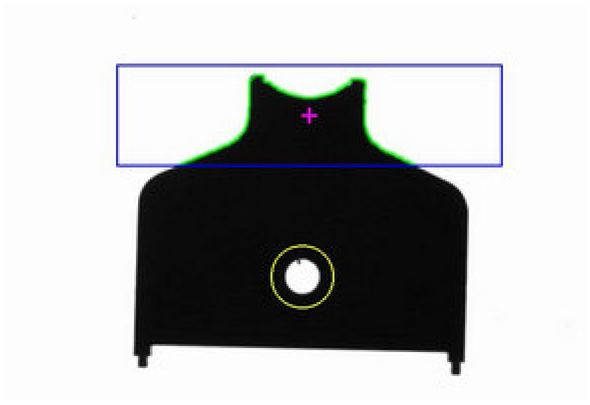


Figure 8 An example of a reference image

The image will not be automatically reloaded if the reference objects changes. The page has to be reloaded, or you will have to create a JavaScript that update the image source: `InspectorUI.RefreshReferenceImages();`

Optionally the user can be allowed to move the region of one or multiple inspections of the reference object by drag & drop. To activate this interaction set the `data-toolregion` attribute. The format of the argument is `'<tool-index>:<parameter-id>'` where `<tool-index>` is the index of the tool (see section 2.2 Parameter Adjustment Controls) and `<parameter-id>` is the usual id of the move command in the command channel. Multiple regions can be provided separated by a comma (,). For example to move the first and second pixel counter and the object locator use:

```
<img class="hmi-refimage" data-toolregion="0:86, 1:86, 38" />
```

Unfortunately there is no way for the widget to know what tool is at what index so the parameter id must also be provided. Following list summarize the available combinations:

Tool	Possible Id
Object Locator	38
Blob	58
Polygon	72
Pixel counter Edge pixel counter Pattern Edge Circle tools	86

For more complete example see the **batch change** example.

Only regions of the active reference object can be moved and the Inspector must be in Edit mode.

2.4.3 Reference image list

A list with all reference images in the configuration can be displayed by adding the following tag to your html:
`<div class="hmi-refimagelist"/>`.

Optionally the user can be allowed to click the depicted reference object to switch to that reference object. To activate this interaction set `data-selectable="true"` for example:
`<img class="hmi-refimagelist" data-selectable="true"/>`

By default the list is shown vertically with a vertical scrollbar if needed. To make the list horizontal set `data-orientation="horizontal"`, a horizontal scrollbar will be shown if needed. Here is an example:
`<img class="hmi-refimagelist" data-orientation="horizontal"/>`

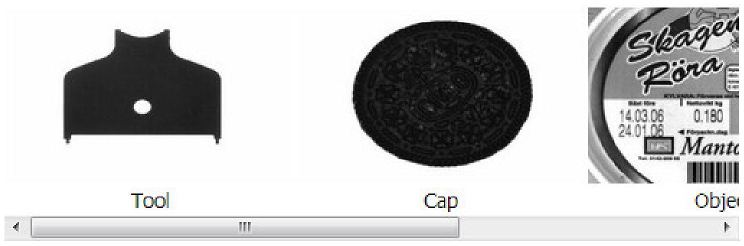


Figure 9 An example of a horizontal reference image list.

It is recommended to set the height and width of the list area in the stylesheet, otherwise the scrollbars might not appear. See #refImageBanks in batchchange.html for an example.

The images will not be automatically reloaded if the reference objects changes from another source than the webpage itself. The page has to be reloaded, or you will have to create a JavaScript that update the images:
`InspectorUI. RefreshReferenceImageList();`

Because of a bug in the web API the name and image of reference objects can become out of sync when objects are added or removed from the list of reference objects in SOPAS. It is therefore recommended to have an incrementally ordered list without any jumps or missing numbers. Unfortunatly there is no easy way of sorting the images, but with copy and remove commands it is possible with some extra effort.

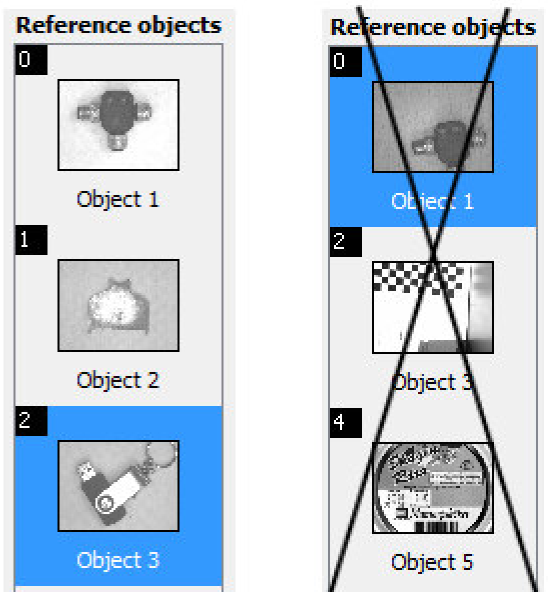


Figure 10 Good and bad reference object list.

Example
For an example on how to use this widget to list all reference images and allow the operator to change reference object, see the Batch Change example.

2.4.4 Log image

The latest log image can be displayed by adding the following tag to your html
`<img class="hmi-logimage"/>`.

To get another log image use the data-index attribute to specify the index of the log image to get. The index is zero-indexed so the first log image is number 0. For example to get log image number 4 use:
`<img class="hmi-logimage" data-index="3"/>`.

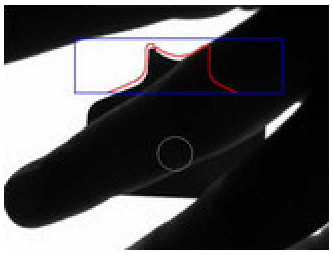


Figure 11 An example of a log image

The image will not be automatically reloaded when the log images change. The page has to be reloaded, or can call the `RefreshLogImages` function, see chapter 3.

During the loading of log images the image collection will be locked on the camera and no new images will be added to the collection. Try to minimize the number of log images based on the application to minimize the time the log image collection is locked.

2.4.5 Log image list

The latest log images can be displayed as a list by adding the following tag to your html `<div class="hmi-logimagelist" />`.

By default all 30 images will be loaded. This will slow down the camera and might take some time to load. To reduce the load an image range can be specified using the `data-index` attribute. For example to load log image 2 to 8 use:

```
<img class="hmi-logimage" data-index="2-8"/>
```

By default the list is shown vertically with a vertical scrollbar if needed. To make the list horizontal set `data-orientation="horizontal"`, a horizontal scrollbar will be shown if needed. Here is an example:

```
 <img class="hmi-logimagelist" data-orientation="horizontal"/>
```

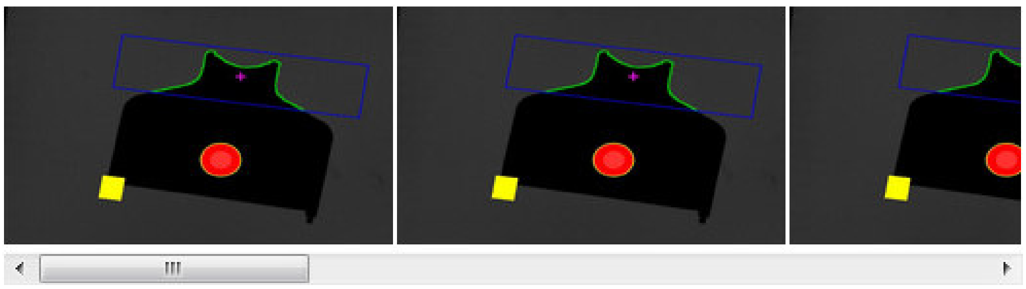


Figure 12 An example of a log image list.

The images will not be automatically reloaded if the reference objects changes. The page has to be reloaded, or you will have to create a JavaScript that update the images:

```
InspectorUI. RefreshLogImageList();
```

2.5 Result string

The latest result string can be retrieved by adding the following element to your html:

```
<div class="hmi-resultstring"></div>
```

The string will be automatically updated at the same rate as the live image. But, there is no guaranty that the image and result string are in sync. The contents of the result string can be set using the “Ethernet Result Output” dialog in SOPAS. The string is displayed as plain text and you can use CSS-styling to format the display.

2.5.1 JSON formatted result string

To increase the ease of use when it comes to getting specific values out of the result string it is possible to send the result string JSON (JavaScript Object Notation) formatted. In the “Ethernet Result Output” dialog in SOPAS it is possible to generate an example JSON formatted result string. This string can afterwards be modified with the desired contents.

In SOPAS the formatting string could for example look like:

```
{"MESSAGE": {
    "IMAGE_NUMBER":"<IMAGE_NUMBER/>",<SPACE/>
    "Pixel_counter_1": {
    <PIXEL_COUNTER name="Pixel counter 1">
        "DECISION":"<DECISION/>",<SPACE/>
        "PIXELS":"<PIXELS/>"
    </PIXEL_COUNTER>
    }
}}
```

This would result in the following string:

```
{"MESSAGE":{"IMAGE_NUMBER":"1234",
"Pixel_counter_1":{"DECISION":"1", "PIXELS":"5678"}}}
```

This string can then be converted into a JavaScript object with use of jQuery or the browsers native JSON parser, example:

```
var result = $.parseJSON(resultString);
```

Specific results can then be easily accessed simply by writing:

```
var pixels = result[MESSAGE][Pixel_counter_1][PIXELS];
```

2.6 Statistics

The same statistics that are shown in SOPAS can be retrieved by adding the following element to your html:

```
<div class="hmi-statistics"></div>
```

Use the `data-interval` attribute to change the refresh interval from the default value of 2000 milliseconds. All statistics widgets share the same interval (the smallest value is used), this way only a single request is needed to the camera for each update.

As default the widget will display all available statistics. To only get a specific value add the attribute `data-item`, example:

```
<div class="hmi-statistics"
data-item="Number_of_acquired_images"></div>
```

Statistical values that can be of interest are listed below:

Statistic	**Comment**
Calculated_frame_period	Corresponds to the frame rate shown in SOPAS.
Calculated_minimum_delay_time	Corresponds to the minimum delay time shown in SOPAS.
Number_of_acquired_images	Number of acquired images since boot
Number_of_frames_in_which_no_match_was_found	Not located
Number_of_frames_with_blob_tool_timeout	Number of times the blob tool has bailed out of the analysis
Number_of_frames_with_match_and_all_pass	All passed
Number_of_frames_with_match_and_any_fail	Located but detail failed
Number_of_frames_with_no_match_and_any_fail	Not located and any detailed failed

Number_of_ignored_trigger_pulses	Number of trigger pulses the device has ignored due to being busy
Number_of_inspections	Total number of inspections in reference bank
Number_of_overflow_trigger_pulses	Number of trigger pulses due to overflow

2.7 Actions

Actions can be performed on the Inspector by adding the following tag to your html

```
<button class="hmi-action" data-id="1">Save to Flash</button>
```

The data-id attribute specify which action to run. Supported values are listed below. Some values allow for an optional attribute to be specified as well.

ID	Description	Additional optional attributes (=default)
1	Save settings to Flash	-
2	Re-Teach current reference object	-
3	Perform calibration	data-squareside = 6
4	Remove calibration	-
5	Apply IP settings	data-usedhcp = 0
6	Restart	-

The displayed widget will look like below:

Figure 13 An example of an action button

2.8 Mode selection

To allow the user to switch camera mode add the following tags to your html

```
<button class="hmi-mode" data-mode="0">Run</button>
<button class="hmi-mode" data-mode="1">Edit</button>
```

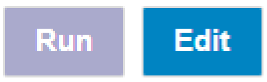


Figure 14 An example of two mode switch buttons

The current mode will be highlighted when the page loads or the user switch mode.

After the mode has changed it is recommended to update the user interface to disable or hide widgets based on the new state. Do not do this in the eventhandler for the button click. Instead add an eventhandler for the modeChange event. This way it will also be run when the mode is switched from another location (for example from within SOPAS). The `InspectorUI.modeChange` event (see section 3.1.1) can be used for this, see the **batch change** example on how to use this event.

2.9 Result plot

This toolkit makes use of the JavaScript plotting library Flot to visualize results from the Inspector over time. Currently the widget is quite limited in its default behavior, but the user have the possibility to freely configure it through the option `plotoptions` which controls how the plot should look and behave, see http://www.flotcharts.org/ for more information.

The result plot works with a JSON formatted Ethernet Result Output string. This result string currently must at least contain the image number as well as one inspection, see 2.5.1 for an example.

The default plot will have a list in which the user can select which result to visualize as well as a button to clear the plot of all data. The plot will show up to 100 data samples after which the oldest sample will be removed from view.

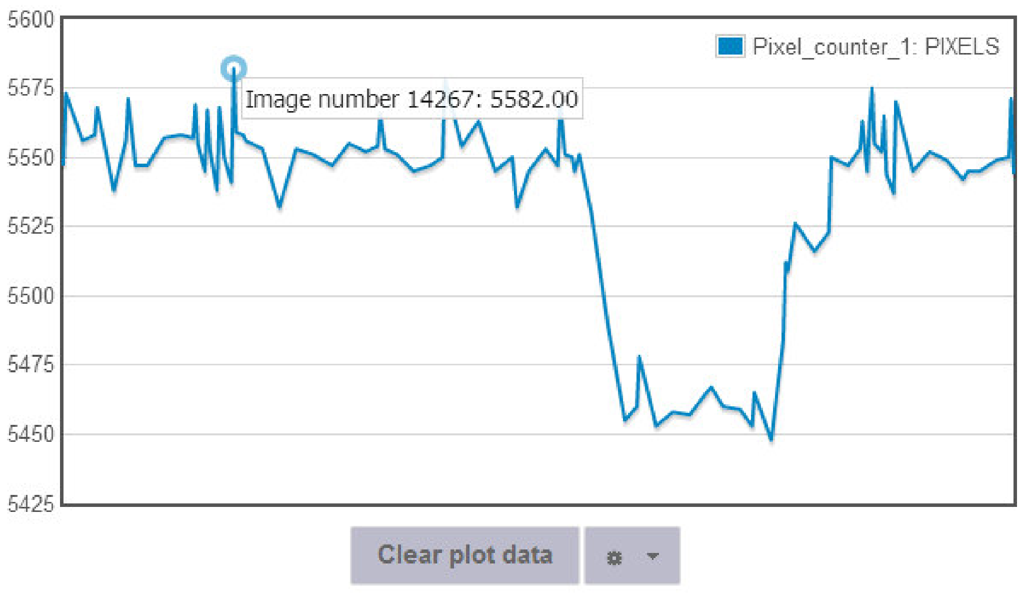


Figure 15 Result plot

The following tag is used to add the result plot:
`<div class="hmi-resultplot"></div>`

The result selection menu and plot clearing button are enabled by default and can be disabled by setting the attributes `data-showclearbutton` and `data-showmenu` to `false`.

By default all tools in the Ethernet Result Output string will be added to the result selection list. The tools in this list can be specified by using the attribute `data-toollist` with a CSV list containing the names of the tools, e.g. `"Pixel_counter_1,Edge_1,Blob_1"`.

To select a default result the attribute `data-selectedvalue` is used. The tool and its result are specified as a JSON key value pair, e.g. `{"Pixel_counter_1":"PIXELS"}`.

Note that the inner quotes need to be escaped as `&quot;`, i.e. `data-selectedvalue="{&quot;Pixel_counter_1&quot;:&quot;PIXELS&quot;}"`

The result plot will store its contents in the browser using Web storage. This makes it possible to save up to 5 MB of data. The result plot will store the results of all inspections by default, unless otherwise specified by the `data-toollist` attribute. Note this storage limitation will be reached faster the more tool results are available.

After any changes to the result string or available results the plot needs to be cleared of all previous stored data.

3 Advanced

This section assumes that you have a good knowledge of web programming and especially JavaScript.

The `InspectorUI` object has a couple of events and functions to help the programmer with some common operations.

3.1 Events

All events have the optional argument `deviceip`, this parameter is only needed if multi-camera HMIs are built.

3.1.1 modeChange

This event is trigged when the user switches camera mode between Run and Edit and at page load. It is triggered when the mode is acknowledged, not when the mode is command-ed. Use this event to update the HMI accordingly, for example to hide or show widgets or update control values. The event contains the new mode as an argument. The following is a usage example:

```
$(InspectorUI).on( "modeChange" , function(e, newMode, deviceip){
    alert(newMode);
});
```

For a more complete example see the batch change example.

3.1.2 refObjChange

This event is trigged when the user switches the active reference object or reteaches the object or at page load. It is triggered when the object change is acknowledged, not when it is commanded. Use this event to update the HMI accordingly, for example to update control values. The event contains the new reference object as an argument. The following is a usage example:

```
$(InspectorUI).on( "refObjChange" , function(e, index, deviceip){
    alert(index);
});
```

For a more complete example see the batch change example.

3.1.3 parameterChange

This event is trigged when the user changes a parameter setting through a widget. It is triggered when the change is acknowledged, not when it is commanded. Use this event to update the HMI accordingly, for example update or hide other widgets. The following is a usage example:

```
$(InspectorUI).on( "parameterChange" , function(e, id, val,
deviceip){
    alert("Parameter " + id + " is now " + val);
});
```

3.2 Functions

All functions below are called through the InspectorUI object, example:
`InspectorUI.RefreshReferenceImages();`

3.2.1 RefreshReferenceImages()

Refresh all reference images

3.2.2 RefreshReferenceImageList()

Refresh the reference image list.

3.2.3 RefreshLogImageList()

Refresh all images in a log image list.

3.2.4 RefreshControls()

Refresh the value of all sliders, radio buttons and checkboxes.

3.2.5 AddDevices(objs)

Adds a list of Inspector API wrapper instances to the `InspectorUI` object, used in multi-camera setups, see section 3.3.

3.2.6 AddDevicesIP(ips)

Adds a list of IP-addresses of devices to the `InspectorUI` object, used in multi-camera setups, see section 3.3.

3.2.7 Other helper functions

There are also other helper functions included in the toolkit. These are located in the inspector-ui.js file and often used in the templates. The functions are called in a similar fashion as exemplified above. See the code for details.

All of the helper functions have an optional parameter `deviceip`, this parameter is only needed if multi-camera HMIs are built.

Calibrate(mm, deviceip)
Starts the calibration sequence with the provided square side length of the calibration object provided in mm.

RemoveCalibration(deviceip)
Removes the calibration

RestartDevice(deviceip)
Restarts the camera. The page will have to be manually reloaded when camera has restarted.

SaveToFlash(deviceip)
Will save current configuration to flash so that it is retained when the camera is restarted.

Reteach(deviceip)
Will re-teach the active reference object.

SetReferenceObject(index, deviceip)
Will set the active reference object to the provided index.

GetIP(ip1, ip2, ip3, ip4, deviceip)
Will get the IP-address of the camera and write the octet values to the input textboxes provided as ip1-4.

SetIP(ip1, ip2, ip3, ip4, deviceip)
Will set the IP-address of the camera to the octet values provided. You will have to call ApplyIP() for the settings to apply. (See maintenance example)

GetNetmask(nm1, nm2, nm3, nm4, deviceip)
Will get the IP-netmask of the camera and write the octet values to the input textboxes provided as nm1-4.

SetNetmask(nm1, nm2, nm3, nm4, deviceip)
Will set the IP-netmask of the camera to the octet values provided. You will have to call ApplyIP() for the settings to apply. (See maintenance example)

GetGateway(gw1, gw2, gw3, gw4, deviceip)
Will get the IP-gateway of the camera and write the octet values to the input textboxes provided as gw1-4.

SetGateway(gw1, gw2, gw3, gw4, deviceip)
Will set the IP-gateway of the camera to the octet values provided. You will have to call ApplyIP() for the settings to apply. (See maintenance example)

ApplyIP(usedhcp, deviceip)
Will apply the network settings set with the commands above. The camera will need to be restarted in order for the IP-settings to take effect, and the page needs to be manually reloaded after this operation. (See maintenance example)

SetMode(mode, callback, deviceip)
Will set the mode (Run or Edit) of the camera. When the value is acknowledged the callback will be run.

When using any of these commands the Inspector will proceed through a sequence of mode changes and other commands. The HMI designer should make sure that no other operations can be commanded during this time.

3.3 Multi-camera HMI

It is possible to control multiple Inspectors from a single page. This can be useful when the user wants to have an overview over several Inspectors on a single page. There are two steps that are needed to setup a multi-camera page. First the `InspectorUI` object needs to know about all the devices that should be used. Second each widget needs to know which device it belongs to. The initialization is preferably done in the HTML header in a JavaScript before the DOM structure has been constructed.

```
...
<script type="text/javascript">
    // Place init code here
   $(function() {
        // Additional operations here, e.g. setting device mode.
   });
</script>
...
```

To specify which cameras to use choose one of the following methods.
Adding devices by IP-address:

```
InspectorUI.AddDevicesIP(["192.168.0.100", "192.168.0.101"]);
```

Adding Inspector API wrapper instances:

```
var inspectorLeft = new Inspector("192.168.0.100");
var inspectorRight = new Inspector("192.168.0.101");
InspectorUI.AddDevices([inspectorLeft, inspectorRight]);
```

This second method can be useful if there is a need to access the API wrapper directly to perform specific tasks.

To specify which device a widget should be connected to, add the `data-cameraip` attribute, example:

```
<img id="leftimg" class="hmi-liveimage" data-
cameraip="192.168.0.100"/>
<img id="rightimg" class="hmi-liveimage" data-
cameraip="192.168.0.101"/>
```

The IP-addresses above are used to refeer to the cameras. The cameras should preferably use a known static IP and not a DHCP server.

3.4 Widget specifications

The following section contains the specifications for every widget included in the toolkit. This includes the available options that can be set, as well as public functions.

To set an option after initialization has been performed use the following syntax:

```
$(".class").widgetname("option", "optionname", value);
```

Example:

```
$(".hmi-liveimage").liveimage("option", "interval", 1000);
```

Public methods are called in a similar fashion, example:

```
$(".hmi-statistics").statistics("statistics",
"Calculated_frame_period");
```

3.4.1 Action button

Widget name: actionbutton

Options	Type	Description
cameraip	String	IP address for the device
id	Integer	Command channel identifier
squareside	Integer	Calibration chessboard square side length

3.4.2 Checkbox

Widget name: checkbox

Options	Type	Description
cameraip	String	IP address for the device
id	Integer	Command channel identifier
tool	Integer	Tool device index

3.4.3 Live image

Widget name: liveimage

Options	Type	Description
cameraip	String	IP address for the device
interval	Integer	Refresh interval of the live image in milliseconds
overlay	String	Overlay mode: “hide”, “show” and “simple”
scale	Integer	Downsample factor for image, 1 or 2

3.4.4 Log image

Widget name: logimage

Options	Type	Description
cameraip	String	IP address for the device
index	Integer	Log image index, 0 to 30
overlay	String	Overlay mode: “hide”, “show” and “simple”

3.4.5 Log image list

Widget name: logimagelist

Options	Type	Description
cameraip	String	IP address for the device
index	Integer	Log image index, 0 to 30
orientation	String	Widget orientation, “vertical” or “horizontal”
overlay	String	Overlay mode: “hide”, “show” and “simple”

3.4.6 Mode

Widget name: mode

Options	Type	Description
cameraip	String	IP address for the device
mode	Integer	Set initial mode

3.4.7 Monitor

Widget name: monitor

Options	Type	Description
cameraip	String	IP address for the device
id	Integer	Command channel identifier to monitor
multiplier	Integer	Float parameters needs to be set using integer values, see Reference manual

3.4.8 Radio

Widget name: radio

Options	Type	Description
cameraip	String	IP address for the device
id	Integer	Command channel identifier
labels	String array	Text labels for the radio buttons
tool	Integer	Tool device index

3.4.9 Reference image

Widget name: refimage

Options	Type	Description
cameraip	String	IP address for the device
toolregion	Object array	Enable region movement, see 2.4.2. Object is defined as { tool: <device-index>, id: <move-command-identifier> }

3.4.10 Reference image list

Widget name: refimagelist

Options	Type	Description
cameraip	String	IP address for the device
orientation	String	Widget orientation, “vertical” or “horizontal”

3.4.11 Result string

Widget name: resultstring

Options	Type	Description
cameraip	String	IP address for the device
interval	Integer	Refresh interval of the result string in milliseconds

3.4.12 Result plot

Widget name: resultplot

Options	Type	Description
cameraip	String	IP address for the device
color	String	Set the color of the plot, overridden by plotoptions
plotlength	Integer	Maximum number of data points shown in plot
plotoptions	JSON	Options to control the layout and values of the plot, see Flot for more information. (http://www.flotcharts.org/) NOTE! This option cannot be set through the use of `data-` attribute.
showclearbutton	Boolean	Show or hide the button for clearing the plot
showmenu	Boolean	Show or hide the menu for selecting result
selectedvalue	JSON	Specifies the result that should be plotted, e.g. {"Pixel_counter_1":"PIXELS"}
toollist	CSV String	Specify which results should be listed in the result menu, e.g. "Pixel_counter_1,Edge_1".

3.4.13 Slider

Widget name: slidertext

Options	Type	Description
cameraip	String	Device IP-address
id	Integer	Command channel identifier
max	Integer	Upper threshold
min	Integer	Lower threshold
multiplier	Integer	Float parameters needs to be set using integer values, see Reference manual
range	Boolean	Regular or range slider
tool	Integer	Tool device index
unit	Integer	Unit to set the value in, i.e. pixels (= 0) or mm (= 1)

3.4.14 Statistics

Widget name: statistics

Options	Type	Description
cameraip	String	IP address for the device
id	Integer	Command channel identifier

Public functions

Function name	Parameters	Description

refresh	-	Manually request a statistics update
statistics	name	Returns the latest requested statistic specified by ‘name’. This assumes that the statistic has been re-quested at least once before. If ‘name’ is not specified a HTML formatted string containing all statistics will be returned.

4 Templates and Examples

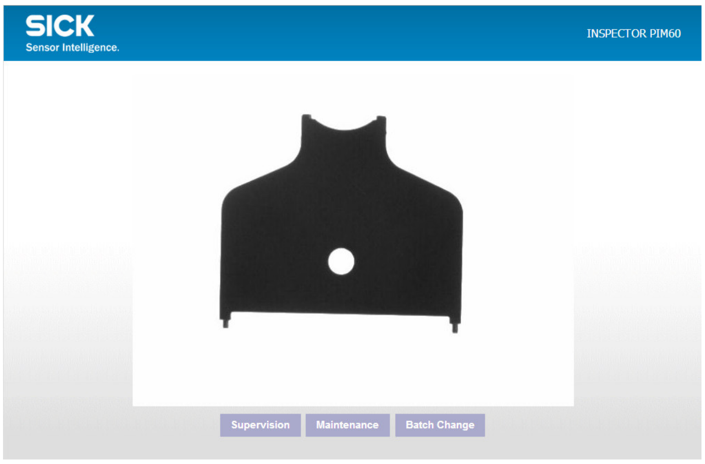
There are a couple of pages included in the toolkit, intended as both examples and templates to be customized for a particular need:

Index	An empty boilerplate html file showing the live image and links to the other pages.
Supervision	A simple HMI where the user is presented with the live image, reference image, result string and latest log images. No changes to the Inspectors configuration are done.
Maintenance	A scenario where the user is to exchange the camera to a new one. This illustrates how to perform calibration, set IP settings and up/downloading a complete configuration.
Batch Change	A more advanced scenario where the reference object is switched and some configuration parameters are set. It also shows how to move an inspection’s region in the reference image. This example illustrates three possible for ways of doing a batch change, normally only one is used.
Result Plotting	A page that contains both statistics for the current reference bank as well as a widget that plots a result over time (Image number). There is a button to clear the plot as well as a menu where it is possible to select which result to visualize.
Multi Camera	A page which demonstrates the possibility to control two Inspectors from a single web page.

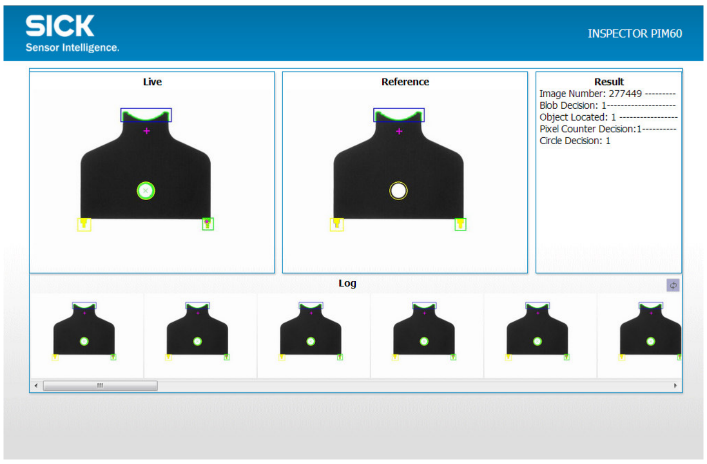
Note

To try out these examples: Upload all files to the Inspector and run them from the internal web server. The templates are made for a screen width of 1024 pixels wide.

SICK Inspector PIM60 Vision Sensor

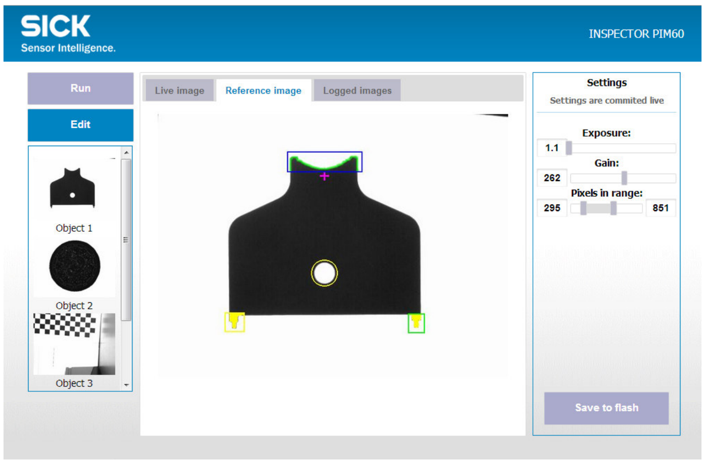


index.html



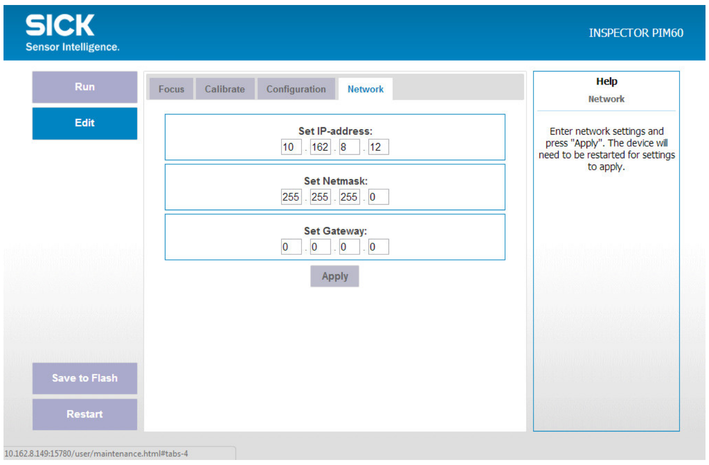
Result
Image Number: 277449 ---------
Blob Decision: 1--------------------
Object Located: 1 ----------------
Pixel Counter Decision:1----------
Circle Decision: 1

supervision.html



Settings
Settings are commited live
Exposure: 1.1
Gain: 262
Pixels in range: 295 851

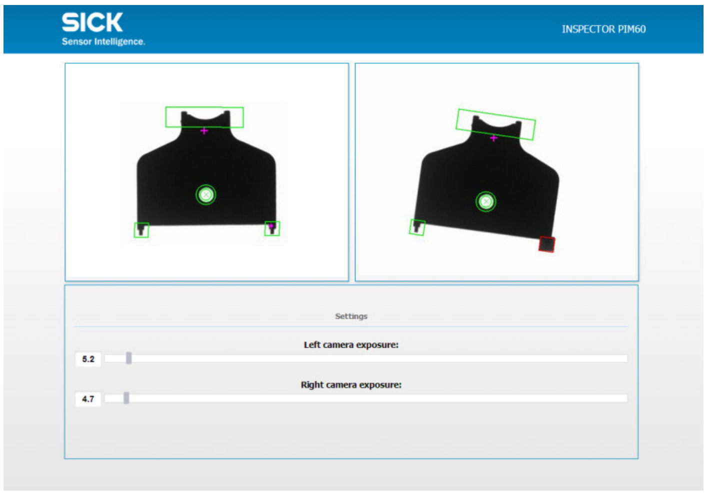
batchchange.html



Set IP-address: 10 . 162 . 8 . 12
Set Netmask: 255 . 255 . 255 . 0
Set Gateway: 0 . 0 . 0 . 0

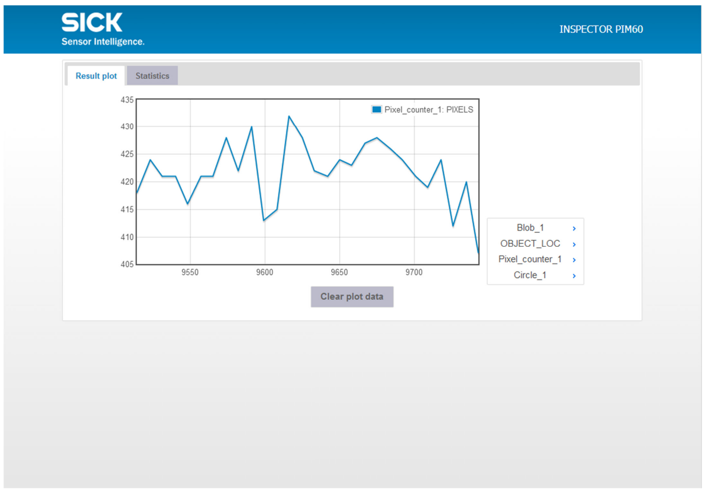
Help
Network
Enter network settings and press "Apply". The device will need to be restarted for settings to apply.

maintenance.html



Settings
Left camera exposure: 5.2
Right camera exposure: 4.7

multicamera.html



resultplot.html

4.1 Customization

The templates can easily be customized for a specific look and feel and a specific application. For production quality this should be done by a professional web designer. But for demonstration purposes just follow the simple steps below.

Change the logo

1. Login to the user web pages (see installation chapter)
2. Delete /user/logo.png from the Inspector.
3. Save the new logo as logo.png.
   (The logo can often be downloaded from the customer's website)
4. Upload the new logo.png
5. Reload the page to see the result

The logo will automatically scale the fit the header, but for bandwidth reasons the image should preferably be scaled to 55 pixels high. Remember to keep the aspect ratio.

Change the color scheme

1. Open the template.css file for editing in Notepad etc. (Do not use Word)
2. Locate the line beginning with `.backgroundColor`, it should be near the top. This is the 'style rule' for backgrounds.
3. Find the line reading: background-color: #0084c2 !important;
4. Change the hex-color to the new color. You can get the hex color value from the customer's website or use a service like www.colorpicker.com.
5. Save the file
6. Login to the user web pages (see installation chapter)
7. Delete /user/ template.css from the Inspector.
8. Upload the edited template.css file.

The other colors template can be edited the same way in the template.css file. See the follow table for reference.

Style rule	Description
`body`	The background color of the whole page.
`.backgroundColor`	The background color for example the header.
`.borderColor`	The color and width of the borders of boxes.
`.ui-state-default`	The color widgets in default mode. (Not in any of the other states below)
`.ui-state-active,` `.active`	The color of widgets in active mode. (For example active camera mode)
`.ui-state-hover`	The color of widgets in mouse-over / hover mode.
`.ui-state-disabled`	The color of disabled widgets. For technical reasons this does not apply to all controls.

There are more style rules in the template.css file for more advanced users.

4.2 FAQ

I am not getting any live image. What is wrong?

SOPAS have higher priority than the web interface inside the camera. Disconnecting from the camera in SOPAS will give you the live image back.

I set a high rate of the live image widget, but the image still only updates at a couple of Hertz. What is the problem?

The live image widget of this toolkit allows you to select a rate. This will be the maximum rate, since the control will not ask for a new image when an old request is still pending. If this is circumvented you could end up with a lot of pending requests, leading to a known bug of the camera – it will probably stop responding.

Can I change the size of tool inspection regions?

No, this is unfortunately not supported by the command channel which the framework is based on. You are only able to move the regions.

Can I host the pages on an external server?

No, the framework is designed to run from the internal webserver of the PIM60. Advanced users can however copy code from the framework to use in their own implementation.

4.3 Support

Support for this toolkit is given at the vision support forum (http://visionsupport.sick.com). If you find any issues or bugs please report them at the same forum.