



Sherlock 8

Software Reference Manual



Document number 405-00087-00

Software Version 8.10.0.0

Copyright© 2021 Teledyne Imaging

All Right Reserved.

All copyrights in this manual and the hardware and software described in it, are the exclusive property of Teledyne Imaging and its licensors. Claim of copyright does not imply waiver of Teledyne Imaging or its licensors other rights in the work. See the following Notice of Proprietary Rights.

NOTICE OF PROPRIETARY RIGHTS

This manual and the related hardware and software are confidential trade secrets and the property of Teledyne and its licensors. Use, examination, reproduction, copying, transfer and/or disclosure to others of all or any part of this manual and the related documentation is prohibited except with the express written consent of Teledyne.

The information in this document is subject to change without notice. Teledyne makes no representations or warranties with respect to the content of this manual and specifically disclaims any implied warranties of merchantability or fitness for a particular purpose. Teledyne assumes no responsibility for errors or omissions in this document.

Sherlock, and the Teledyne logo are trademarks of Teledyne Incorporated. All other trademarks are the property of their respective owners.

Do not attempt to reverse engineer the software. Do not attempt to use the Emulator in a production environment without obtaining a license. Your Sherlock Embedded software license is only valid for use with a BOA Pro smart camera.

Teledyne Digital Imaging US, Inc.

Information: TDI_Sales.ipd@teledyne.com

Support: TDI_Support.ipd@teledyne.com

Web: <http://www.teledynedalsa.com/visionsystems>

700 Technology Park Drive

Billerica, MA USA 01821

Tel: 1.978.670.2002 **FAX:** 1.978.670.2010

Table of Contents

Sherlock 8 Software	5
How Sherlock Works	6
Running a Program	6
Programming Sherlock.....	6
Getting Started with Sherlock 8	6
Tutorials	6
Examples	6
On-Line Help	7
ROI: Region of Interest.....	7
Extracted ROIs	7
Interpolation	7
Resulting Image Injection	8
Rectangle ROI.....	8
Annulus ROI	9
Line ROIs	10
Line.....	10
PolyLine.....	10
Arc.....	11
Composite Line ROIs	12
Rake ROI.....	12
PolyRake ROI.....	13
Spoke ROI.....	14
Rainbow ROI	14
Manipulating ROIs.....	15
Sherlock Application Windows	16
Main Window.....	16
Main Window Toolbar	17
Main Window Status Bar	17
Image Workspaces.....	18
Image Toolbar Buttons	19
Program Window	20
Program Window Toolbar	20
Add Instructions.....	21
Add Variables	21
Copy and Paste	21
System Objects Window	22

Properties Window	23
Basic Properties	24
Custom Properties	24
ROI Processing	25
OCR Training	25
Search	26
Color Training.....	28
Tolerance	28
Toolbox Window	29
Watch Window	30
Monitor Window.....	30
Reporter Window	30
Acquisition Options and Setup Menus	31
Hardware Driver Configurator	31
Sapera LT Devices Menu	32
Application Options	34
Script Editing	40
Script Editing Window Controls.....	40
JavaScript in Sherlock.....	41
Using JavaScript Objects	41
Debugging Tools.....	46
Execution Modes.....	46
Single Step Mode	46
Highlight Mode	46
Disable Execution.....	46
Instructions	46
Preprocessors and Algorithms.....	47
Breakpoints	47

Sherlock 8 Software

Sherlock 8 represents a new generation of the Sherlock™ machine vision software. All formulas, instructions, algorithms and preprocessors are now treated as instructions. All system resources and instructions are treated as objects.

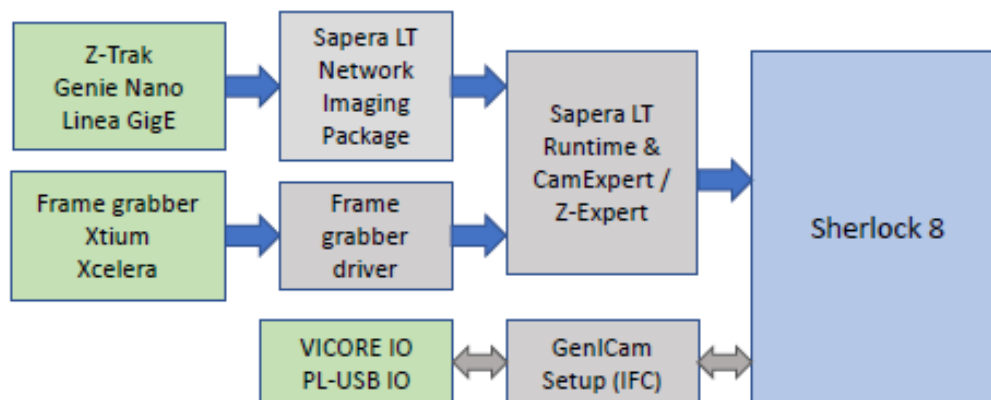
1. Everything is an “object” or program object.
2. The **System Window** exists to define and manage objects that are global or system-wide resources. Some resources are predefined (cam0, inputs, outputs, serial ports). The System Window allows you to create some objects or resources (variables, alignment objects, calibration objects, some Communication objects, etc.).
3. All objects have properties.
4. The **Properties Window** exists to display or modify an object’s properties. Not all properties can be modified. The Properties Window has two tabs: “Basic” and “Extended”. The Basic tab is a table of values. All properties that do not conform to this table format, appear in the Extended tab. The Extended tab is used to display line profile information, train OCR and Search algorithms, train target calibrations.
5. The **Program Window** exists to assign instructions to objects or resources and to define an order of execution. The Program window allows you to create instructions and subroutines and assign variables to instruction inputs or outputs. Algorithms and Preprocessors are image processing instructions that can only be created inside the scope of an ROI. ROIs can only be created inside the scope of an Image Workspace.

Sherlock is a user-programmable machine vision application that offers extensive inspection capabilities and system design flexibility. Sherlock 8 software can be installed on any PC running Microsoft Windows 7 or 10 64-bit Operating Systems.

Sherlock software can be installed alone for Demo purposes. This allows you to experiment with image processing on image files included with the software, or you can substitute your own image files.

Sherlock software requires a license to acquire images from a camera.

- Sherlock with Spera LT software, and a supported GigE source or camera: Z-Trak Profiler, Genie Nano, Linea GigE, Genie TS, or Genie camera.
- Spera frame grabber driver if using a frame grabber such as the Xtium or Xcelera with appropriate camera(s) attached.
- Sentinel is needed only if you use a USB Dongle license.
- GenICamSetup is needed if you use the PL-USB I/O, VICORE I/O, or VICORE PLC port.



Acquisition Software and Drivers

How Sherlock Works

Running a Program

Run Once: When Sherlock executes an "investigation" program once, each instruction in the "main" routine is executed in sequence, from the top of the Program window, to the bottom (including any branches or subroutine calls). After the last instruction is executed the investigation stops (the Sherlock Embedded software does not exit).

Run Continuously: When Sherlock executes an "investigation" continuously, each instruction in the "main" routine is executed in sequence, from the top of the Program window to the bottom (including any branches or subroutine calls). After the last instruction is executed, the sequence is repeated from the beginning (top) of the "main" routine. This repeats until the user halts running the investigation (the Sherlock Embedded software does not exit).

Programming Sherlock

A Sherlock program is based on the Image Workspace and Program Window. You draw one or more **ROI** ("Region Of Interest") in the Image Workspace, and add image processing to the ROI. Double-click in the ROI in the Image Workspace or double-click on the ROI instruction in the Program Window, to open a Properties Window and add processing (algorithms and preprocessors).

In general, a preprocessor changes the image data, and an algorithm extracts data or measurements from the image data. The measured results are called "readings", "output properties" or "objects". In the Program Window, you add instructions to manipulate the readings, and communicate results or decisions to the user or to peripheral equipment, through the Serial, Ethernet, Modbus or PLC interfaces.

The best way to become familiar with the Sherlock Embedded application, is by following the **Tutorial 1** available in the Sherlock8 on-line Help and in the User Documents directory.

Note: There are no "OK" "Accept" "Cancel" or "Undo" buttons because all **changes occur immediately**. In many places you must press enter or click outside the current entry field you are changing ("change the focus") to execute a value change. **Tip:** You can use Ctrl-Z to undo changes and Ctrl-Y to redo changes.

Getting Started with Sherlock 8

The Sherlock 8 Installation Manual covers installing and configuring the multiple software libraries needed to acquire images from cameras into Sherlock 8.

Sherlock 8 – image processing library and GUI interface

Sapera LT – GigE camera driver and acquisition library

GenICamSetup64 – IO Drivers for the PL-USB and VICORE and VICORE's PLC port.

Sentinel – required to read a USB dongle license. Not required for a license stored on the system disk or on a Z-Trak Profiler.

Tutorials

There are several Tutorials included with the Sherlock 8 installation. These tutorials cover Basic GUI, Configuring the Z-Trak or Genie Nano, Alignment and Calibration and training. The Tutorials are available in PDF format in the **User Documents** directory. There is a link to User Documents in the Start menu. The On-line Help contains copies of some tutorials. Tutorial 1 contains several videos. Video files are not included in a "minimal" installation.

If you are new to Sherlock, we Strongly Suggest you follow Tutorial 1 to gain familiarity with the program interface.

Examples

The Sherlock8 installation includes several examples of 1D, 2D and 3D images. There are examples of VBDotNet projects and compiled executables from these projects. These are located under the **User Data** directory link in the Start menu.

On-Line Help

The On-line Help documents all system resources and instructions. After following the Tutorial, you can access the Help for additional details and useful information.

ROI: Region of Interest

Sherlock analyzes and processes pixels in an ROI (Region Of Interest). The ROI defines a portion of the image that provides pixel data for processing.

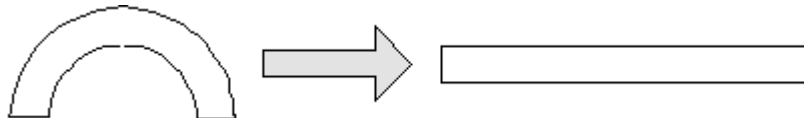
- The ROI contains the pixel data for processing. The ROI does not contain the associated graphics such as outline, labels, resize handles and graphic colors.
- The “Tracker” is the graphic component that represents an ROI in the image workspace. The Tracker is the outline, graphics, labeling, and colors associated with the ROI.

Sherlock 8 has three types of ROIs: Extracted, Line, and Composite Lines.

- Extracted ROIS are: Rectangle and Annulus.
- Line ROIS are: Point, Line, Polyline and Arc
- Composite Line ROIs are: Rake, Polyrake, Spoke and Rainbow.

Extracted ROIs

The pixels in Extracted ROIs are extracted from the image, into a rectangular buffer. If necessary, the ROI pixels are distorted to fit into a rectangle. A rotated rectangle is extracted to a rectangle buffer. An Annulus ROI is unwrapped and warped such that pixels towards the outer radius of the annulus are “squeezed together” and those on the inner radius are “stretched out” to form a rectangle of pixels. You can view the extracted image buffer, by clicking on the "Roi Image" tab at the top of the Custom view of the ROI Properties window.



You should be aware of this extraction when making linear measurements on extracted ROIs. Extracted ROIs can be quickly processed because pixel access is simple and uniform. Line and Composite ROIs are also extracted ROIs but are one-dimensional.

Note: The Annulus ROI has a `close_ends` property, which you can use to make it a full circle.

Interpolation

Either nearest-neighbor interpolation or linear interpolation can be used when extracting pixels from an input image. The type of interpolation is selected by the "interpolate" property.

Nearest-neighbor interpolation means that the pixel closest to the desired pixel position is used. This is really no interpolation.

Linear interpolation uses the four nearest pixels to estimate the pixel value at a fractional pixel location. Linear interpolation produces more accurate pixel values in extracted images but is slower than nearest-neighbor interpolation.

Interpolation is not necessary if you are using non-rotated, rectangular ROIs.

Resulting Image Injection

Each ROI can have a pipeline of preprocessors and algorithms. Preprocessors modify the pixel values. Some algorithms can also modify pixel values. At the end of the ROI's processing pipeline, the resulting image can be "injected" or re-inserted back into the Workspace Image Window for display and additional processing.

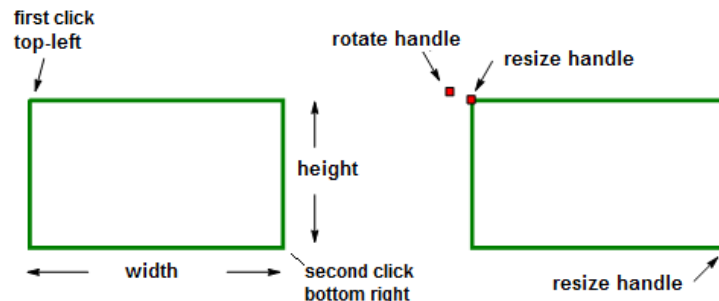
If the `modify_image` property is true, then the processed image will be injected into and modify the Image Window. This property is true by default for the Rectangle and Annulus.

The ROI's processed image is injected into the Image Window without interpolation – using nearest-neighbor selection of pixel values. This introduces errors in the modified areas of the Image Window. These errors include errors in position and pixel values. You can observe this by creating an Annulus ROI, fill it with a constant intensity value and inject it back into the Image Window (`modify_image` is true). You will see ragged edges on the "filled" Annulus. Use the Image Window's zoom in button if necessary.

The injected pixels do not all match the area defined by the Annulus ROI. As the injected pixels are all one value, errors in pixel values are not apparent. Use caution when "cascading" processors between ROIs. Processing, injecting, processing again, re-injecting; is referred to as "cascading". It is better to use the Image Extract algorithm and Image Inject preprocessor to cascade ROI processing, because these pass the extracted (and interpolated) image rather than going through insertion and another extraction (and interpolation) in the Image Window.

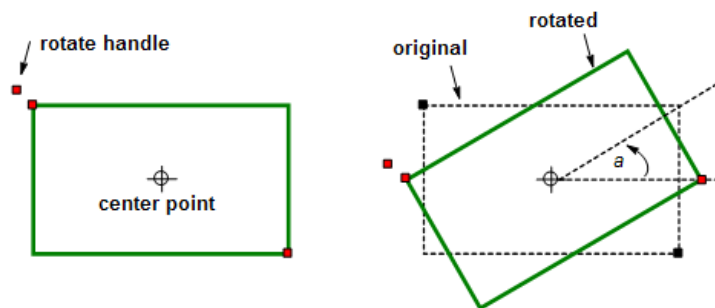
Rectangle ROI

To **draw** a Rectangle, you click at two opposite corner points. You can draw a rectangle (move your mouse) in any direction. You can drag the two corner handles to resize the ROI. You can also rotate the ROI using the Rotate handle in 2D images. Rectangles in 3D Plane (1D) images do not rotate.



- The Rectangle ROI data is **always processed** from the **top-left corner** to the **bottom-right corner**.

The Rectangle ROI is **defined** in Sherlock by the center point, width and height of the rectangle, and angle of rotation. This is independent from the direction you moved your mouse to draw the ROI. These values are available in the Rectangle Properties. You can access these values directly, in a script, or by attaching them to variables.



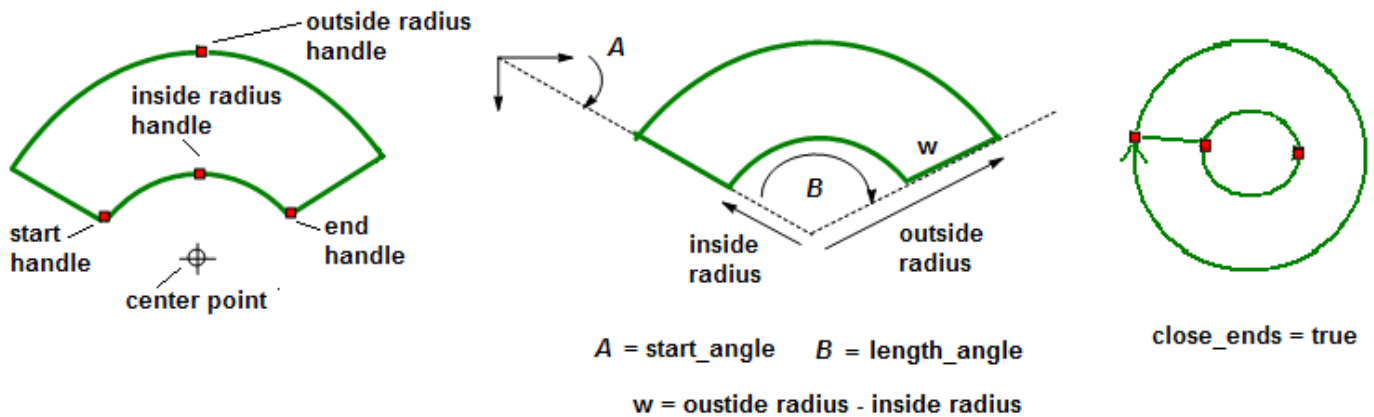
The points and angles that define an ROI are **always in image or pixel coordinates**. The ROI definitions are never given in Calibrated or World coordinates.

The Rectangle ROI is an "Extracted" area ROI. The processing result is usually re-inserted or re-injected into the Image Window.

Notice that the rectangle is always drawn parallel to the X and Y axis, and then rotated if needed. This is exactly how Sherlock represents or defines the Rectangle's position internally: a rectangle drawn parallel to the axis, then rotated about the center point. As you rotate the rectangle, the center point coordinates, width and height do not change, only the rotation angle changes. This defines how the rectangle would be drawn then rotated. This is important to remember when you move a rectangle (or Rake).

Annulus ROI

To **draw** the Annulus ROI, you click at four points in the image window. The first point is the start of the arc, the second point is the end of the arc. The third point can be on either the inside or the outside arc. The fourth point can be on either the outside or the inside arc. The point on the outside arc can also be off the end of the arc drawn in the image window (on the "extended" curve of the arc). You can drag any of the four handles to resize the Annulus ROI. You can select "close_ends" in the Property window, to make a closed circle Annulus.



The Annulus ROI is **defined** in Sherlock by the center point, inside radius, width, "start_angle" and "length_angle".

The "start_angle" is defined relative to the X,Y axis, and positive in the clockwise direction. The "length_angle" is the central angle of the Annulus (see illustration below).

The "start_angle" and center point are not changed by closing the Annulus. The "length_angle" becomes 360 degrees when closed. The "length_angle" defaults to 315 degrees if you open a closed Annulus.

These values are available in the Annulus Properties. You can access these values directly, in a script, or by attaching them to variables.

The points and angles that define an ROI are always in image or pixel coordinates. The ROI definitions are never given in Calibrated or World coordinates.

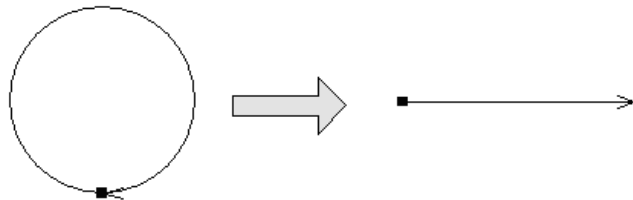
The Annulus is a "Extracted" Area ROI. The ROI area is remapped to a rectangle area for processing. The direction of processing the rectangle is from the top-left corner to the bottom-right corner. The processing result is usually re-injected into the image window.

Line ROIs

Line ROIs are small, extracted images and are used like Extracted ROIs. Line ROIs have an interpolation option, as with Extracted ROIs. Interpolation is linear or nearest neighbor. Point ROIs are treated as 1 x 1 line images and are a special type. Line and Composite ROIs are usually used only for measurements (Algorithms) that don't modify the image and injecting a single line of pixels would usually not be visible.

Note: The Arc ROI has a `close_ends` property, which you can use to create a full circle line.

Line images are put into a 1 x N (row) image. Polyline ROIs are "flattened" into one long 1 x N image. Arc ROIs are also "flattened" into a 1 x N image. The closed Arc ROI is broken where the (magenta) square and arrow point meet and unwrapped into a 1 x N image starting at the (magenta) square.



Line ROIs are usually used for measurements (algorithms) and do not need to modify the Image Window at the end of the ROI's processing chain. The Line ROIs do not have the `modify_image` property.

Algorithms and preprocessors that operate on single pixels (points) can often use the same code for Extracted, Line and Composite ROIs. When image areas are needed, as with convolution or morphological operations, then the code must be different. For example, the Erode preprocessor on a Line ROI works along the line (a 1 x n image), while the Erode preprocessor on a Rectangle ROI works on a small image area (m x n).

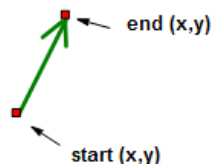
Line

To **draw** a Line ROI, you click at two places in the Image Window. The line can be drawn at any angle and any direction. The arrowhead indicates the end point, and this indicates the direction the pixel data is processed along the line (from start to end points). This is usually important to processing images.

The Line ROI is **defined** by two points, the start and end points, in the order you clicked in the image window. These values are available in the Line Properties. You can access these values directly, in a script, or by attaching them to variables.

The points and angles that define an ROI are always in image or pixel coordinates. The ROI definitions are never given in Calibrated or World coordinates.

The line is extracted, and not re-inserted or re-injected into the image window.



PolyLine

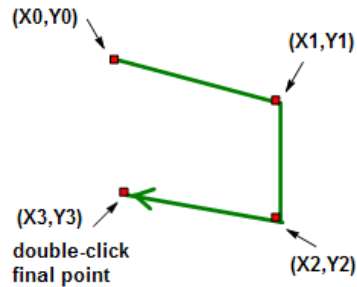
To **draw** a Polyline ROI, you click at points in the Image Window. Double-click to define the last point (fourth point in the illustration, X3,Y3) to complete the Polyline ROI. You can drag any of the corner points to change the shape of the Polyline.

The arrowhead on the last point indicates the "end point" and indicates the order of processing the pixel data points along the line (from the "start" point to the "end" point). This is usually important in processing images.

The Polyline ROI is **defined** by the click points that you used to draw the polygon, in the order that you clicked. The Polyline points are stored as an array of Points in Sherlock. These values are available in the Polyline Properties. You can access these values directly, in a script, or by attaching them to variables.

The points and angles that define an ROI are always in image or pixel coordinates. The ROI definitions are never given in Calibrated or World coordinates.

The Polyline ROI is a "Line" ROI. The line is extracted, and not re-inserted or re-injected into the image window.



Arc

To **draw** the Arc ROI, you click at three points; click on the start, the end, and the radius. You can drag the start, end, and radius handles to resize and move the Arc ROI. A closed circle ROI is the default. You can select "close_ends" in the Property window, to make an open arc or curve.

The direction you move your mouse to draw the first two points, determines the direction of the arrow head at the end of the arc, indicating the direction of processing the pixel data points in the ROI (from the "start" point to the "end" point). This direction is usually important to processing images.

The Arc ROI is **defined** in Sherlock by the center point, radius, "start_angle" and "angle-length".

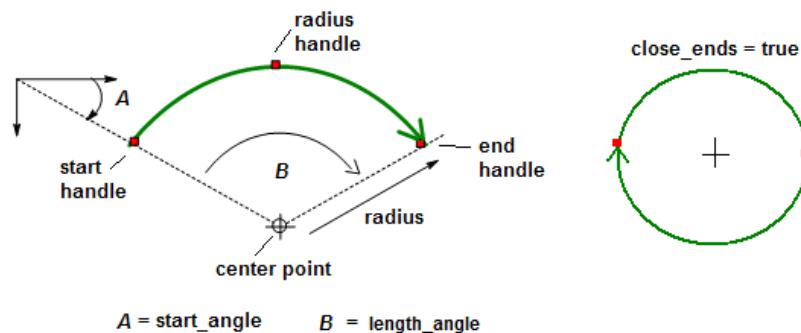
The "start_angle" is defined relative to the X,Y axis, and positive in the clockwise direction. The "angle_length" is the central angle of the Arc or Circle (see illustration below).

The "start_angle" and center point are not changed by opening or closing the Arc. The "angle_length" is 360 for the Circle. The "angle_length" defaults to 270 when you open a closed Arc.

These values are available in the Arc Properties. You can access these values directly, in a script, or by attaching them to variables.

The points and angles that define an ROI are always in image or pixel coordinates. The ROI definitions are never given in Calibrated or World coordinates.

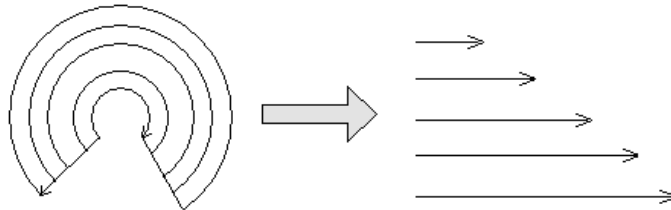
The Arc ROI is a "Line" ROI. The line is extracted, and not re-inserted or re-injected into the image window.



Composite Line ROIs

Composite ROIs are made from multiple line segments, or from multiple Line ROIs. As with Line ROIs, each Composite ROI line is extracted (with or without interpolation) into a set of 1 x N images. Line and Composite ROIs are usually used only for measurements (Algorithms) that don't modify the image and injecting a single line of pixels would usually not be visible.

Rainbow ROIs are also unwrapped or "flattened" into a 1 x N image. The closed Rainbow ROI is broken where the (magenta) square and arrow point meet and unwrapped into a 1 x N image starting at the (magenta) square. Point ROIs are treated as 1 x 1 line images and are a special type.



Note: The Spoke and Rainbow ROIs have a `close_ends` property, which you can use to close the circle, creating a full circle spoke or a full circle rainbow, also called a bull's eye.

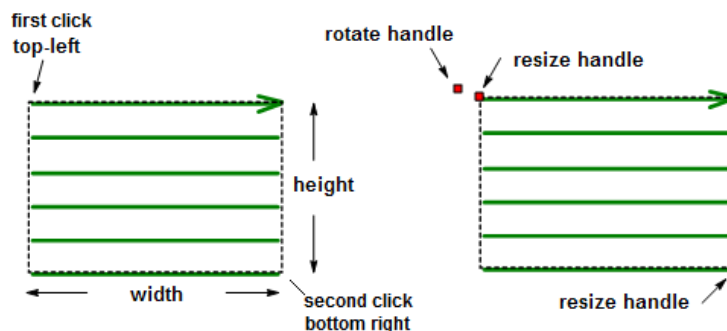
A preprocessor or algorithm is called once for each of the line segments in the Composite ROI, and get the "flattened" and optionally interpolated line segment as 1 x N image.

Composite ROIs are generally used for measurements and generally do modify the Image Window. The Line and Composite Line ROIs do not have the "modify_image" property.

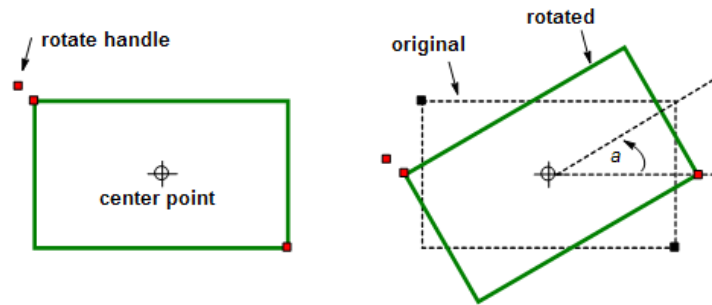
Many algorithms and some preprocessors can use the same code for Extracted, Line and Composite ROIs. Composite ROIs are processed as a series of 1 x N line images.

Rake ROI

To **draw** a Rake, you click at two opposite corner points. You can draw a Rake (move your mouse) in any direction. You can drag any of the four corner handles to resize the ROI. You can also rotate the ROI. using the Rotate handle.



The direction of the arrow on the top-most line, indicates the direction of processing the lines in the Rake ROI. This direction is usually very important to processing images. You can rotate the Rake to change the processing direction. The Rake ROI is **defined** by the center point, width, height, and rotation angle; similar to a Rectangle or bounding box. This is independent from the direction you move your mouse to draw the Rake ROI.



These values are available in the Rake Properties. You can access these values directly, in a script, or by attaching them to variables.

The points and angles that define an ROI [are always in image or pixel coordinates](#). The ROI definitions are never given in Calibrated or World coordinates.

The Rake is a "Composite Line" ROI. The lines are extracted and are not re-inserted or re-injected into the image window.

Notice that the Rake is always drawn parallel to the X and Y axis, and then rotated as needed. This is exactly how Sherlock represents or defines the Rake's position internally: a rectangle drawn parallel to the axis, then rotated about the center point. As you rotate the Rake, the center point coordinates, width and height do not change; only the rotation angle changes. This is important to remember when you move a Rake. When working with a rotated Rake, think of the center point and its relocation in the image.

PolyRake ROI

To **draw** a PolyRake ROI, you click at points in the Image Window. Double-click to define the last point (fourth point in the illustration, X3,Y3) to complete the PolyRake ROI. You can drag any of the corner points to change the shape of the PolyRake.

NOTE: The drawing algorithm is programmed to avoid crossing rake lines, as double sampling points would be incorrect. When dragging corner points, you can see the PolyRake detach and pull away from click points, to avoid crossing and double sampling.

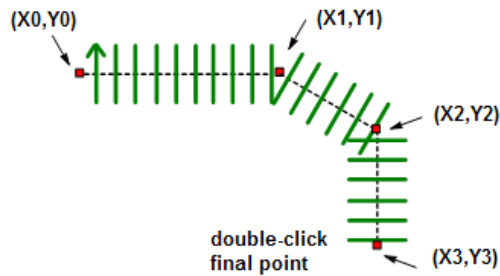
The arrowhead on the first rake line indicates the direction of processing the multiple lines of the PolyRake. This is usually important in processing images.

The PolyRake ROI is **defined** by the click points that you used to draw the polygon, in the order that you clicked. The PolyRake points are stored as an array of Points in Sherlock.

These values are available in the PolyRake Properties. You can access these values directly, in a script, or by attaching them to variables.

The points and angles that define an ROI are always in image or pixel coordinates. The ROI definitions are never given in Calibrated or World coordinates.

The Polyline ROI is a "Composite Line" ROI. The lines are extracted, and not re-inserted or re-injected into the image window.



Spoke ROI

To **draw** a Spoke ROI, you click on three points in the Image Window. Your first click is the start of the inside circle. Your second click is the end of the inside circle. Your third click is on the inside radius. The outside radius is automatically assigned. The scan direction is always from the inside radius to the outside radius; or inside-out. By default the Spoke is a full circle, but there is a "close_ends" property which makes the Spoke an open or partial spoke.

The direction of the arrow shown on one spoke line, is the scan direction, and is always from the "inside radius" to the "outside radius" or "inside-out". This indicates the direction or order of processing pixel data along the radial lines in the ROI. This direction is usually important in processing images. You can make the inside radius larger than the outside radius, which has the effect of reversing the scan direction to "outside-in".

The Spoke ROI (open or closed) is **defined** in Sherlock by a center point, inside radius, width, "start_angle" and "length_angle".

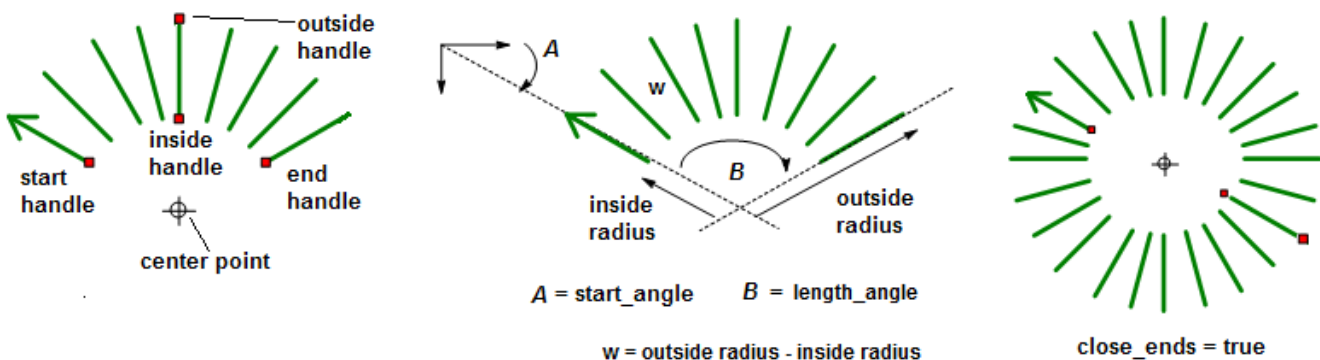
The "start_angle" is defined relative to the X,Y axis, and positive in the clockwise direction. The "length_angle" is the central angle of the Spoke (see illustration below).

The "start_angle" and center point are not changed by opening or closing the Spoke. The "length_angle" is 360 for the closed Spoke ROI. The "length_angle" defaults to 315 when you open a closed Spoke.

These values are available in the Spoke Properties. You can access these values directly, in a script, or by attaching them to variables.

The points and angles that define an ROI are always in image or pixel coordinates. The ROI definitions are never given in Calibrated or World coordinates.

The Spoke is a "Composite Line" ROI. The lines are extracted and are not re-inserted in the image window.



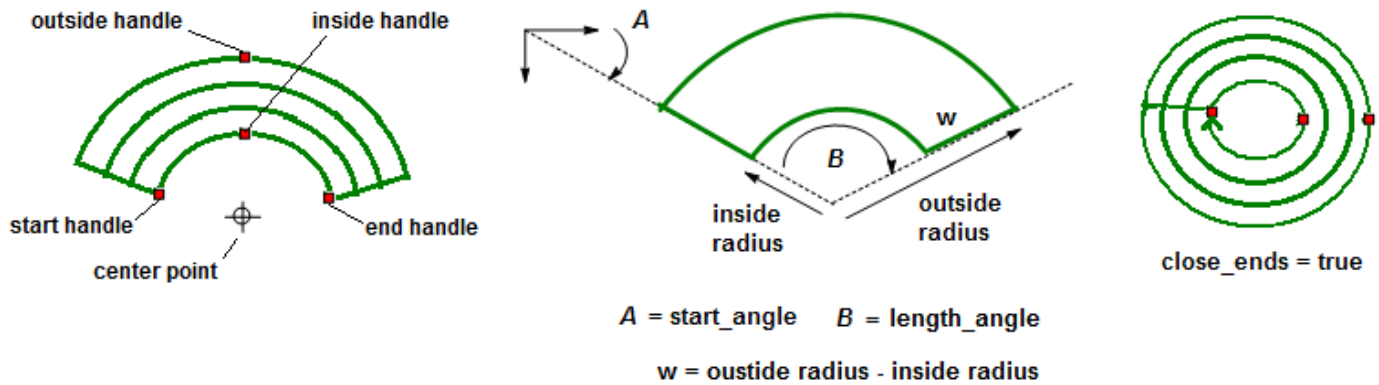
Rainbow ROI

To **draw** a rainbow ROI you click three times in the image window, click at the start, end, and a third click between them on the outside radius. You can stretch and reposition using the four handles.

The rainbow ROI is **defined** in Sherlock by the center point, start angle, inside radius, length angle, and "width" w. These values are available in the Rainbow Properties. You can access these values directly, in a script, or by attaching them to variables.

The points and angles that define an ROI are always in image or pixel coordinates. The ROI definitions are never given in Calibrated or World coordinates.

The rainbow is a Composite Line ROI. It is extracted, and not re-inserted or re-injected into the image window.



Manipulating ROIs

Once you draw an ROI of any shape, the ROI instruction is added in the Program Window, inside the program scope of the Image Workspace.

The Properties for the ROI Instruction are immediately populated with the point coordinates, size, or angle values. The values are updated when you move or resize the ROI in the Image Workspace.

You can change the values for an ROI coordinates, size, or angle directly in the Properties window, in a script, or by attaching the values to variables and changing the variable values through other instructions.

ROIs also have handles which can be used to save their image content, or pass content to another ROI or Image Workspace. The output handle is exposed in the Properties window and you can make it visible in the Program window (for dragging and dropping) by clicking on the gray bubble.

Copy and Paste

You can copy and paste ROIs in the Program Window. ROIs can only be pasted within the scope of an Image Workspace. The new instruction ROI appears directly below the highlighted instruction. The new ROI appears in the Image Workspace when you paste them into the program window.

When you copy an ROI all the settings are copied. All instructions (processing) in the ROI are copied. When you paste an ROI the names of instructions are incremented to the next unused value. The settings are not changed. "rectA" would become "rectB" unless there already exists a "rectB" in the program then it would become "rectC".

If you have changed the default settings of instructions before copying, the new copy has your changes.

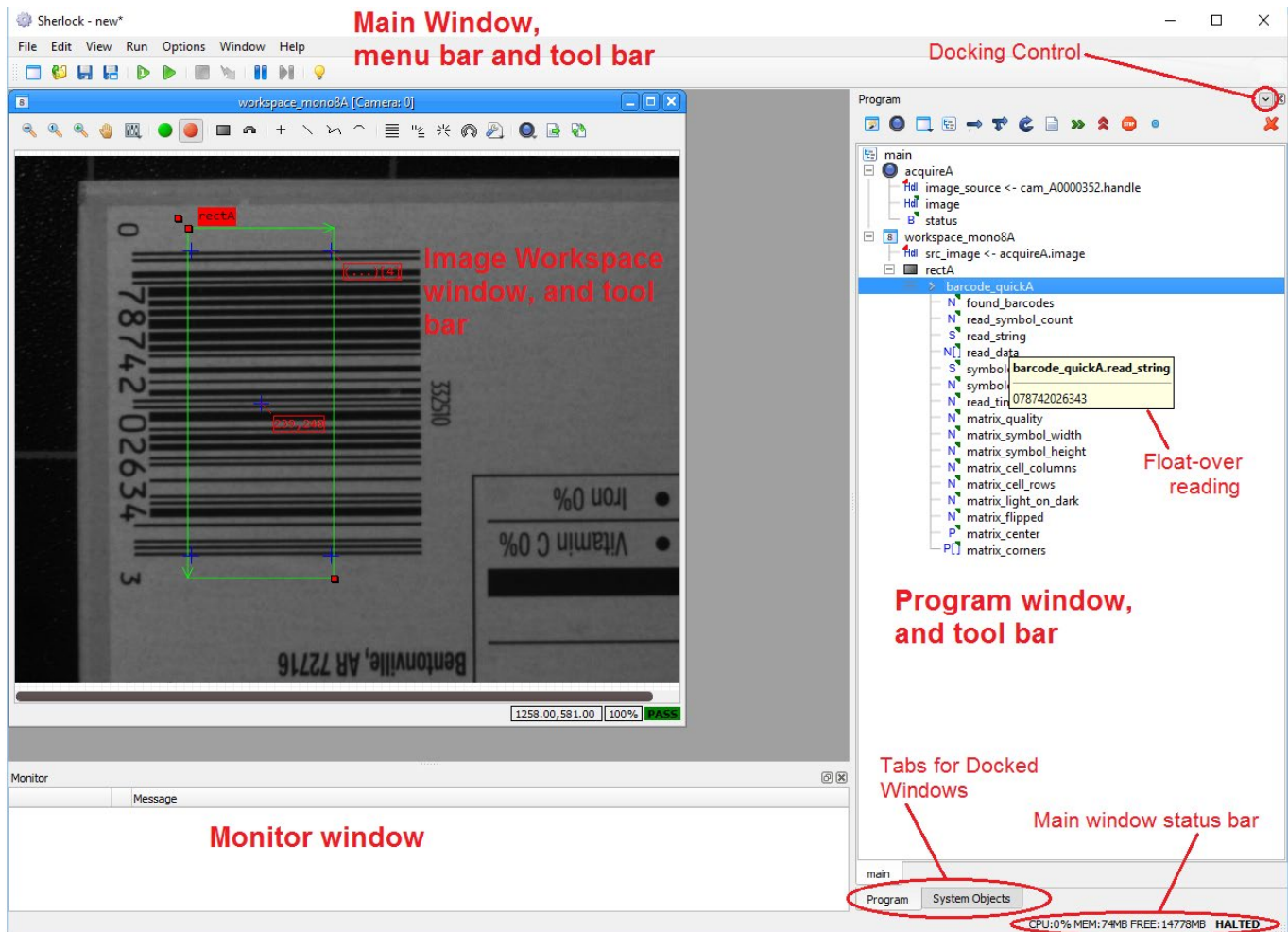
When you paste an ROI the new copy appears in the Image Workspace at the very same location as the original. The new copy is currently selected and can be dragged to a new location in the Image Workspace.

Sherlock Application Windows

Sherlock uses several windows to display features and allow the user to control them. When you exit Sherlock your window layout is saved to a file and is restored the next time you open the Client interface.

Main Window

The Sherlock "Main" window opens when you launch/start Sherlock. The Main window contains all the other window panes or views.



At the top of the Main window is the Menu bar and main Toolbar. The toolbar buttons have float-over tips that identify their actions.

The Windows® minimize button at the top right corner of the Main window, minimizes the Sherlock Main window (and all other windows or views) to the System Taskbar.

The Status bar at the bottom right corner of the main window shows the Pass/Fail status, the current run status (Stop or Run Continuous) and units for angles (Degrees or Radians).

There is a "Docking Control" button in the Program Window which toggles between enabled and disabled. Most of the "child" windows have this docking control. With Docking disabled, windows will stay separated or "float". Windows may be dragged outside the main window. With Docking enabled, you can drag and drop windows onto each other. Tabs will appear at the bottom of the stacked windows. You can resize windows. You can Dock and Un-dock all windows within the Main window, except the Image Workspace. You can drop a window into another window, to create **Tabs**, for

viewing multiple windows. You will also see docked windows shift around to allow dropping a window beside docked windows. **Tip:** The cursor, not the Title Bar, determines where an un-docked window tries to attach or dock.












If you do not like where a window has docked, you can drag it to a new position or click on the docking button to undock it.

If a window is not visible, but it is enabled in the View menu, it may be a tab inside another window. Check for Tabs at the bottom of visible panes or Views.

Your window layout is saved each time you close the Sherlock 8.

The main window is never the program focus. Usually the Image Workspace is the focus when you appear to be in the Main window. The focus only affects launching the Help file.

Main Window Toolbar

	Create a new program		Stop at the end of the current loop
	Open a saved program		Stop immediately
	Save the current program		Enable/disable Single Step execution mode
	Save the current program to a new name		Execute next single step
	Run Once		Highlight the current instruction (in single step mode)
	Run Continuously		

Main Window Status Bar

At the bottom right corner of the main window is a status bar, showing Sherlock's status.

CPU: 0% MEM USED: 47MB FREE: 14696MB CONFIG: default 0.00 ms **HALTED**

CPU	Sherlock's CPU usage, in percent
MEM USED:	Memory in use by Sherlock, in MegaBytes
FREE:	Free Memory on the system, in MegaBytes
CONFIG:	Current configuration file (default or custom)
0.00	Run time or execution time in milliseconds.
HALTED	Run State: program is halted or stopped.
RUNNING	Run State: program is running
PRELIMINARY	Preliminary or Unreleased software version

Image Workspaces

The Image Workspace contains or displays images. You can have multiple image windows open inside Sherlock. You can minimize the Image Workspace. A workspace window is created for a specific image type or pixel depth. The image type defines which ROIs, tools and instructions you can use.

- If **no cameras** are attached, you will see a mono 8-bit “Workspace” window with the “default.bmp” file displayed (image at right). This is usable for 8-bit monochrome image files. You must change the image Workspace type to use color images.
- If you have cameras but **no license**, you also get the “default.bmp” image in an 8-bit mono workspace.

If you have a license and cameras, Sherlock will automatically configure itself for the image format of your cameras.



- If a **monochrome camera** is attached, you will see a mono 8-bit Workspace or a 16-bit Workspace window.
- If a **color camera** is attached, you will see a 32-bit color Workspace window.
- If a **Z-Trak Profiler** is attached, you will see a 1D Workspace window.
- If a **Z-Trak Profiler** is attached and configured for multi-line (3D) acquisition you will see a 3D Workspace window.
- If **multiple cameras** are attached, you will see **only one** image window. You must change the Sherlock configuration for multiple cameras and create multiple acquisition instructions and Workspace instructions.

The Image Workspace cannot be docked or dropped onto another window. You can minimize, maximize, or close the image window with the standard Windows® controls. Once you have maximized the Image window, these controls appear in the top right corner, below the controls for the Main window. You can also delete an image window in the Program Window. You can right-click in the image window to select a color “pallet” to add colorization to the image. You use the Image Workspace to draw ROIs in the image, on or around features you are interested in modifying or measuring. You add image processing to the ROI either in the Image Workspace or in the Program Window. You can right-click inside an ROI to add image processing algorithms and preprocessors, copy, or delete the ROI. When you copy and paste an ROI, the new ROI appears directly on top of or at the same location as the original ROI. You can drag the new ROI to a different location. **NOTE: If you delete an Image Workspace, all ROIs and processing in that window will also be deleted.**

You can right-click on any reading displayed in the Image Workspace, to hide the reading or connect it to an instruction.

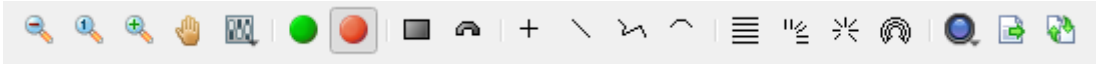
Image Toolbar Buttons

The buttons available in the Image Toolbar changes with the image type: Mono8, Mono16, Color32, 1D and 3D.

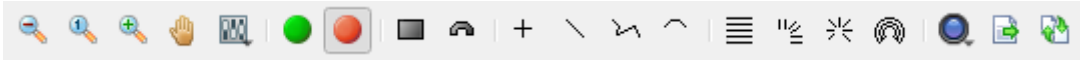
MONO8 Toolbar



MONO16 Toolbar



Color32 Toolbar



3D Profile Toolbar (1D)



3D Cloud Toolbar



	Zoom out (make image smaller)		Create Arc ROI
	Zoom reset 1:1 (original size)		Create Rake ROI
	Zoom in (make image larger)		Create a Polyrake line ROI
	Hand scroll (drag image)		Create a Spoke ROI
	Display options (drop-list menu)		Create a Rainbow ROI
	Start live image acquire. See note below.		Create a Cube ROI (3d Cloud image)
	Stop live image acquire		Create a Plane ROI (3d Cloud image)
	Create a Rectangle ROI		Create a Tool (2 ROIs and multiple algorithms)
	Create two Rectangle ROIs (3d Plane image)		Select a camera image source
	Create an Annulus ROI		Select a file image source
	Create a Point ROI		Select a directory image source
	Create a Line ROI		Save 3d Plane or 3d Cloud image with Metadata
	Create a Polyline ROI		

Note: Start Live image acquire works if the Camera and image source is in Internal Timer, or in External Trigger mode when trigger pulses are being received. If the camera and image source are in External trigger mode and there are no trigger pulses, the live image acquire does not work. Also, the camera trigger setting and image source trigger setting must agree.

Program Window

The Program window contains the "structure" of your program. Your Image windows, ROIs and instructions are all listed in the Program window. You can collapse or expand the instructions in the Program window.

You can hide and show the Program window using the View menu in the Main window menu bar.

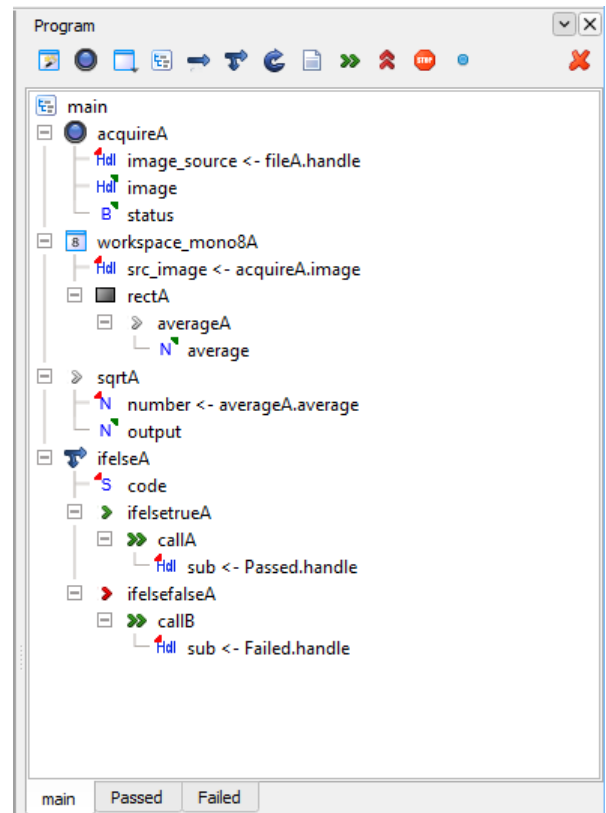
You can add new instructions by right-clicking on instructions in the Program window, using the toolbar, drag instructions from the Toolbox window if open. You can also copy and paste instructions in the program window.

You can have more than one image Workspace window in the Program window. You can have more than one ROI in an image Workspace window.

You can add multiple Preprocessors and Algorithms in an ROI. You can add system or geometric Instruction types in an Image or ROI. Other instructions must be outside the "context" of the image Workspace window.














The Program window always has a "main" subroutine which cannot be renamed or deleted. You can create and rename additional subroutines. There are tabs at the bottom of the Program Window, for the main routine and subroutines.

The Sherlock program does not restrict or pre-define the order of instructions.



Program Window Toolbar

The toolbar creates or deletes "system" instructions in the Program window.

	Create Acquire & Workspace Wizard		Create a Script instruction
	Create an Acquire instruction		Create a Call instruction, to call a subroutine
	Create a Workspace instruction		Create a Return instruction, to return to the calling routine
	Create a Subroutine instruction		Create a Stop instruction
	Create an IF instruction		Create a Comment instruction
	Create an IF-ELSE instruction		Delete the selected instruction
	Create a WHILE instruction		

Add Instructions

You add subroutines and instructions to the Program Window by one of three ways:

- a. Drag and drop instructions from the Instructions Window if it is open. New instructions appear **below** instruction the new instruction is "dropped on".
- b. Click on a button in the Program window toolbar. A new instruction appears **below** the highlighted item.
- c. Right-click on any instruction in the Program Window to get a pop-up menu of the instructions. New instructions appear **below** the highlighted item. You can also temporarily disable an instruction in the pop-up menu.

The pop-up menus are context sensitive. You do not get the Instructions popup if you right-click on an instruction's input or output.

- You can right-click on an instruction's input or output to connect it to a variable or another program resource.
- You can right-click on an output (or reading) to hide the reading in the Image Workspace (where an algorithm may have several outputs you are not interested in).
- You can drag outputs from the Program Window into the Watch Window.

Add Variables

Variables are created in the System window. You can attach variables to inputs or outputs of instructions.

- You can drag and drop a variable from the System window onto an instruction Input or Output.
- You can right-click on an instruction Input or Output to assign a variable.

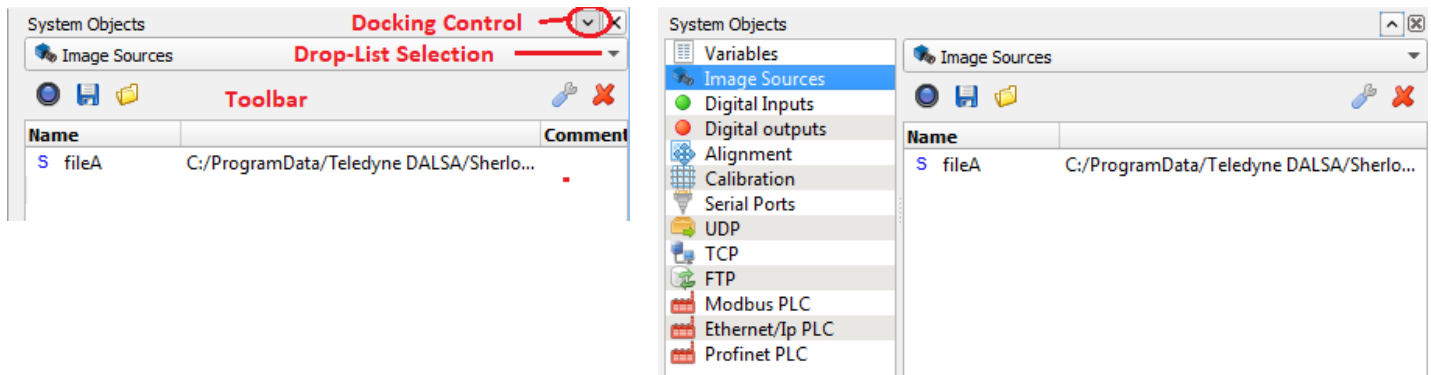
Copy and Paste

- You can copy and paste instructions in the Program Window.
- You can copy and paste values in the Properties window and into existing objects the System Objects window, only if the value is the appropriate data type.
- You cannot copy and paste objects in the System Objects window (variables, calibration, etc.).
- You cannot copy and paste in the Image Workspace.
- When you copy an instruction you also copy all instructions within the scope or "the children of" the instruction. *Examples:* Copying an ROI also copies the preprocessors and algorithms contained in it. Copying an image workspace also copies the ROIs contained and any instructions contained in the scope of Workspace. Copying a subroutine copies all program elements or instructions contained in it.
- When you copy instructions or program elements, you are also copying the settings or values and the connections or instruction inputs and outputs. *Example:* Create an ROI and add some processing. Change settings in the processing instructions. Copy and paste the ROI. The new processing instructions have the same setting changes you made.
- A pasted instruction appears directly below the highlighted instruction, at the same level (after children if any are present). You can use drag-and-drop to change the order of instructions.
- When you paste an instruction all instruction names are incremented to the next unused value. *Examples:* nameA becomes nameB unless there is already a nameB, then it would be nameC. rectA becomes rectB (if there is no rectB already).
- You must click on a routine name (main or any subroutine) to paste a routine.
- When you copy and paste an ROI, the new ROI is placed on top of the first at exactly the same coordinates. The new ROI is currently selected. You can drag it to a new location. You can change the current selection in the Program Window.
- When you paste an instruction the settings and connections are not changed (except when a resource does not exist; see the next bullet).

- When you paste instructions from one program into another, resources are not created automatically. *Example:* In the first program, there are 3 variables: numberA, numberB numberC. You copy a add_numberA instruction with inputs assigned to numberA and numberB and output assigned to numberC. A second program has numberA and numberB created but no numberC. You paste the add_numberA instruction into the second program. Connections to numberA and numberB are kept but the output assignment to numberC is lost. There is no output assignment in the instruction. The resource (number variable) is not created by pasting the instruction.
- It is *very important* that you verify or change the values or connections in pasted instructions.

System Objects Window

You can create and delete system resources in the System Objects window. The window can be vertically divided into two panes however the left pane may be hidden. You can select object types in the left pane (if visible) or use the drop-list at the top of the right pane. You can drag the vertical separator to the left, to hide the left pane and use the drop-list only. You can show the left pane by dragging the vertical separator right.



There is a search field at the bottom of the System Objects window. You can search for text in the current list of **objects**. This may be most useful in the variables display to search the list by variable name.

Variables	Create or Delete variables (11 types).
Image Sources	Create or Delete Image Sources (3 types).
Digital Inputs	See the attached hardware inputs, Create or Delete Digital Inputs.
Digital Outputs	See the attached hardware outputs, Create or Delete Digital Outputs.
Alignment	Create or Delete an Alignment object.
Calibration	Create or Delete a Calibration object (3 types).
Serial Ports	Create or Delete Serial Port objects.
UDP	Create or Delete a UDP communication object.
TCP	Create or Delete TCP communication objects (2 types).
FTP	Create or Delete an FTP communication object.
Modbus PLC	Create or Delete a Modbus communication object (2 types).
Ethernet/Ip PLC	Create or Delete Ethernet communication objects (2 types).
Profinet PLC	Create or Delete Profinet communication objects (2 types).

The Toolbar changes with the selection or category of objects.

Properties Window

The Properties window displays an object's properties. Not all properties can be modified, some are read-only.

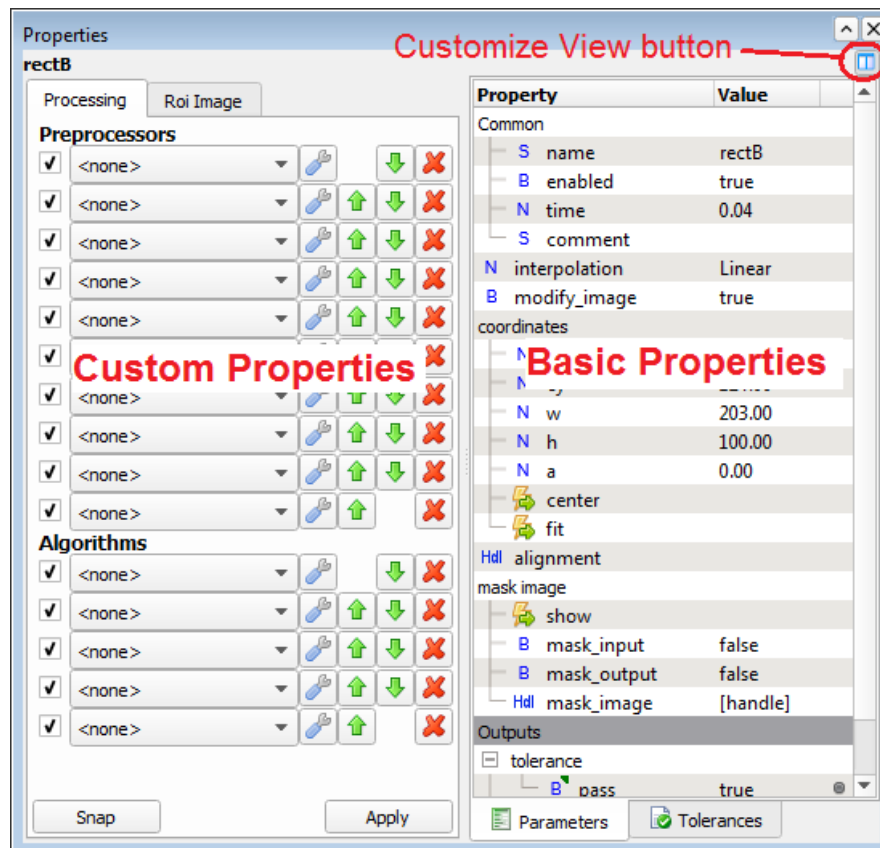
Note: Handles, Instruction Inputs and Instruction Outputs do not have properties.

You can show or hide the Properties window in the Main window's View menu. You have the option of leaving the Property window open or closing it. If the Property window is closed, double-clicking on an object or instruction will open the Properties window.

The size of the Properties window sometimes changes with the instruction or object selected. This is caused by a Custom Properties.

The Properties Window has two views: "Basic" and "Custom". The Basic view is a table of values. All properties that do not conform to this table format appear in the Custom view. The Properties window has a "Customize view" button that toggles between "split" and "basic" view. In "basic" view the basic and custom properties are tabs at the bottom of the properties window.

In "custom" view the Properties window is split into two panes, displaying the custom properties on the left and the basic properties on the right. If you select an instruction or object that does not have custom properties, the left pane disappears and only the basic properties are displayed.

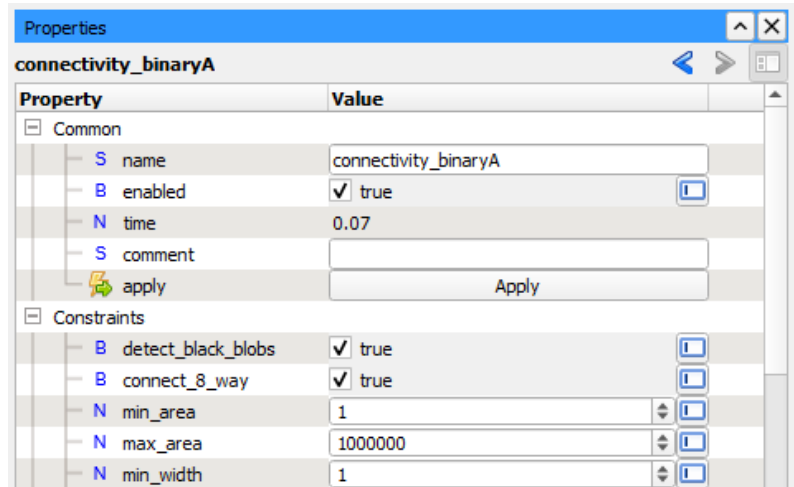


Basic Properties

The Basic view or tab has two columns: Property and Value.

The **Property** column shows the name of each property. To the left of the name is a graphic that represents the data type.

The **Value** column shows the current value of the property. The Value column has several different controls for entering data or values. The controls change with the accepted entries.



Most data types match those shown in the Variables window: Boolean, Number, String, and arrays of these data types. Other types are specific to data types defined in the structure of the Sherlock environment, such as Points, Lines, and handles to Images, Routines, hardware and peripherals.

Most editable values have a field mode button at the far right. This button shows the current entry mode. Click on the field control button to toggle between the two modes:



Entry field - enter (type) numbers or strings in this field or use other controls (number wheel) for value entries.



Link - attach the property value to a variable or program object. The Value field changes to a drop list.

Note: There are no “OK” “Accept” “Cancel” or “Undo” buttons because all changes occur immediately. In many places you must press enter or click outside the current entry field you are changing (“change the focus”) to execute a value change. You can use Ctrl-Z to undo changes and Ctrl-Y to redo changes.

- Gray bubble – the output is not exposed in the Program Window. Click on the bubble to expose this output.
- Blue bubble – the output is exposed in the Program Window. You can drag and drop it into other instructions. Click on the bubble to hide this output in the Program Window.


Custom Properties

The Custom tab exists to display or modify object properties that do not fit the table presentation of the Basic tab. The custom tab changes with the selected object:

- For ROIs, the Custom pane or tab allows you to easily manage processing.
- For OCR algorithms, the Custom pane or tab allows you to Train characters.
- For Search algorithms the Custom pane or tab allows you to train a Search pattern.
- For a few Color algorithms the custom pane or tab shows a Color Training menu.
- For a few Algorithms, the Custom pane or tab displays a plot of intensity and edge strength.


ROI Processing


The Processing tab allows you to select preprocessors and algorithms. This tab allows you to easily substitute and experiment with processing. You can change the order of processing and temporarily disable a processor.

 The drop-lists select preprocessors and algorithms. Only the processors that apply to the current ROI or image type are displayed.

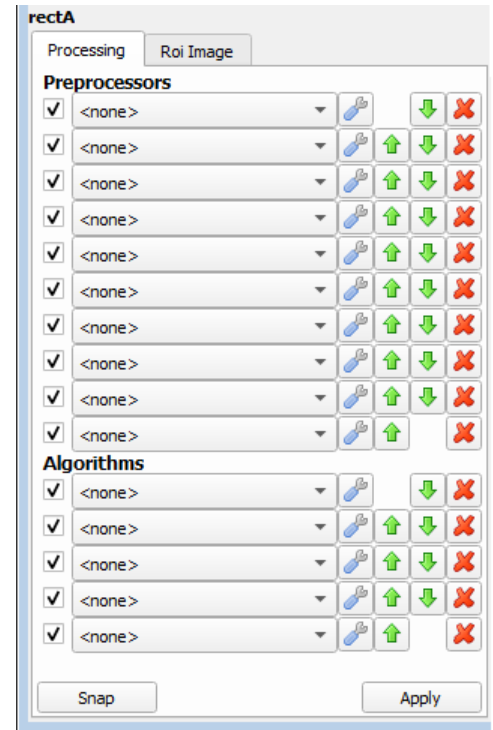
Clear a checkbox to temporarily disable a processor without deleting it. This way your changes to settings are not lost.

 The arrows change the order of processing.

 The Properties button changes the Properties window to the Preprocessor or Algorithm.

 Use the navigation arrows or click in the Program window, to return to the previous properties display.

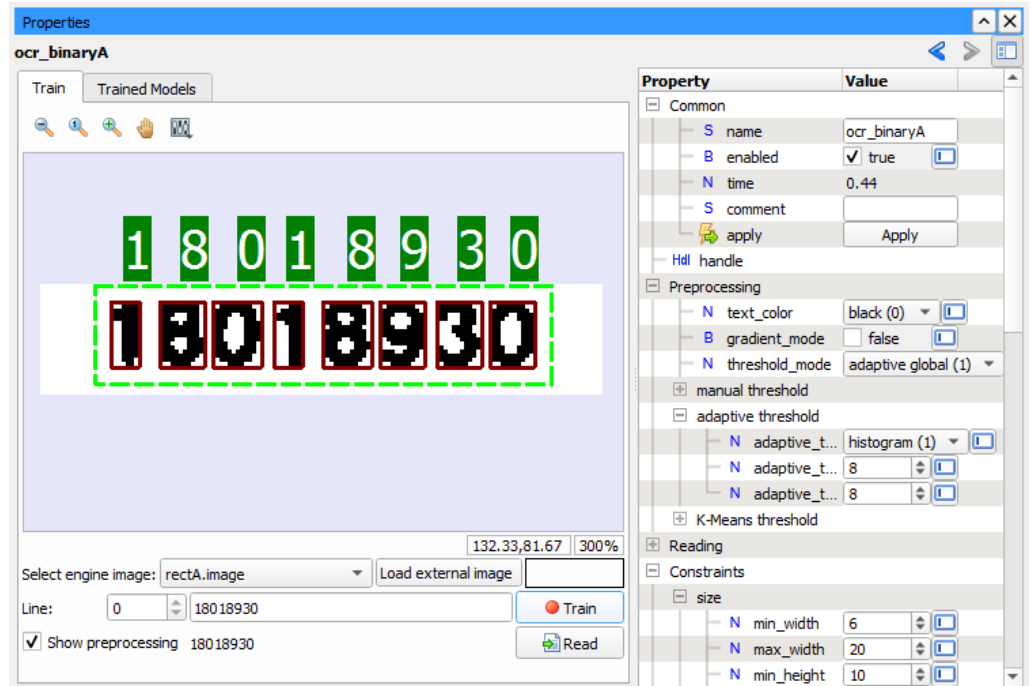
You can click on an instruction in the program window, including preprocessors and algorithms to change the Property window contents.



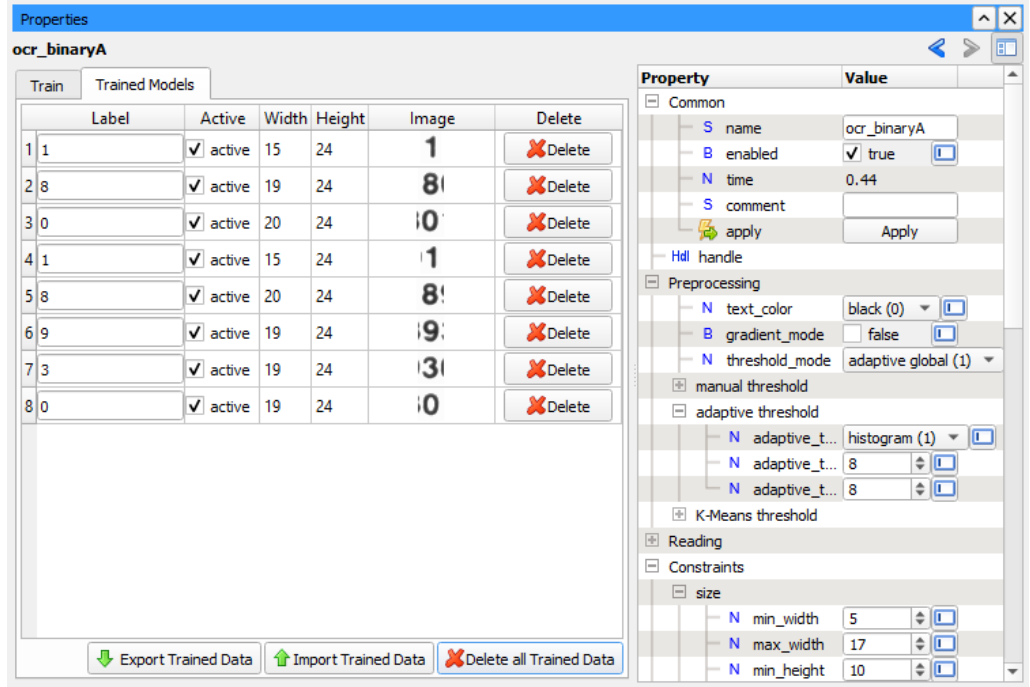
OCR Training

OCR Training – appears when you click on any OCR algorithm in the Program Window. There are two tabs at the top of the Custom pane for “Train” and “Trained”.

The **Train** tab shows the ROI image, and has buttons at the top for zoom, drag, and display options. In `ocr_grayscale` (above left) there is a rectangle to position over each single character to train. In `ocr_binary` (above right) the characters are separated from the background, and rectangles appear over the characters automatically. You can adjust the threshold settings (in the Properties Window) to adjust the separation.



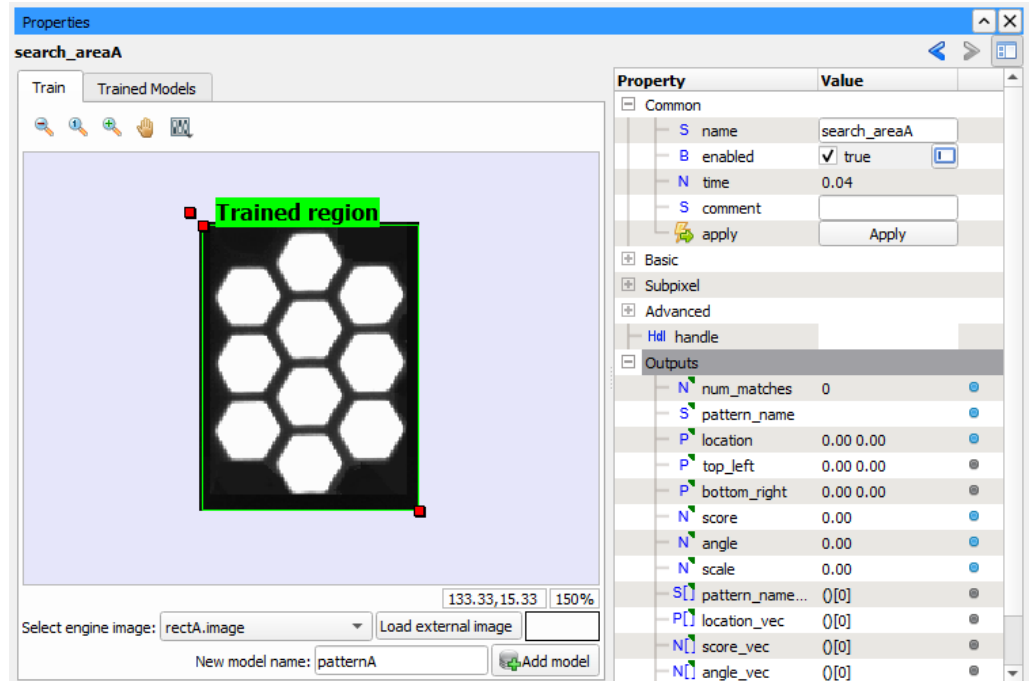
The **Trained Models** tab lists all characters trained in this ROI. You can select characters and use the buttons at the bottom of the panel to enable, disable, rename and remove (delete) trained characters.



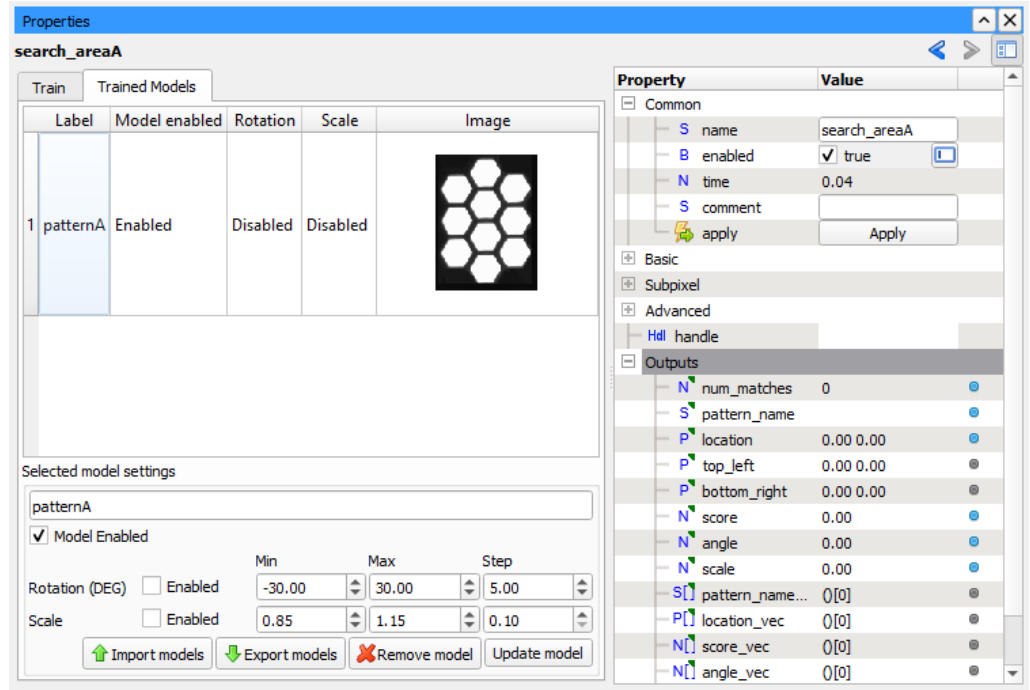
Search

Search Training – appears when you click on any Search algorithm in the Program Window. The **Search Area** has two tabs at the top of the Custom pane for Train, and Trained Models.

The **Train tab** shows the ROI image and a small rectangle to outline the actual feature for the model to be trained. Buttons at top for zoom, drag, and display options.

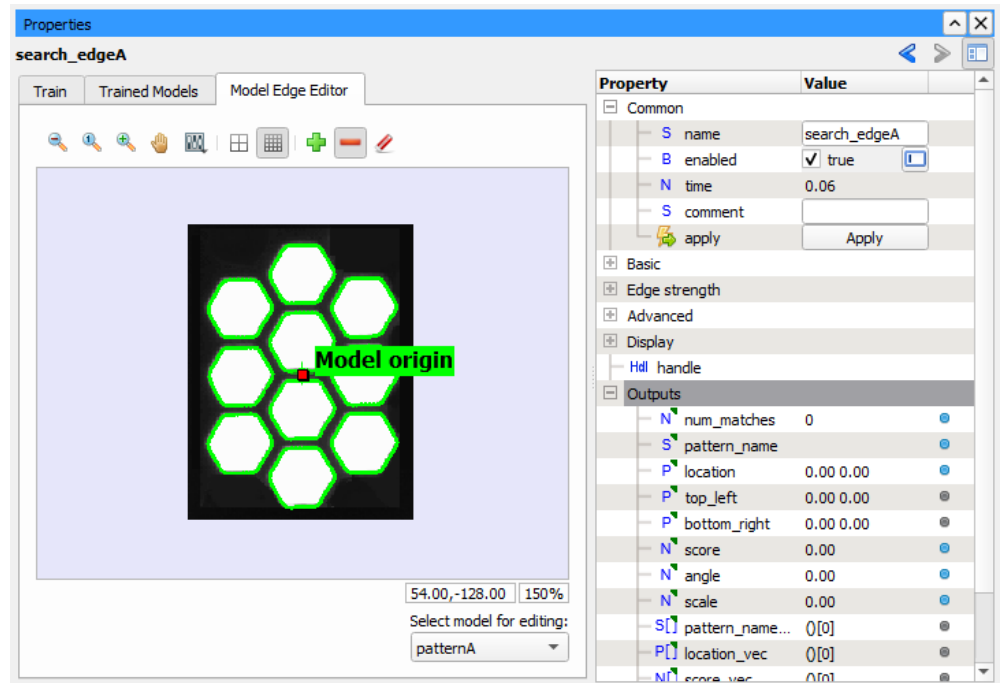





Trained Models tab shows the models trained for this ROI, and status. You can select a model and use the buttons to enable, disable, rename and remove (delete) a model.



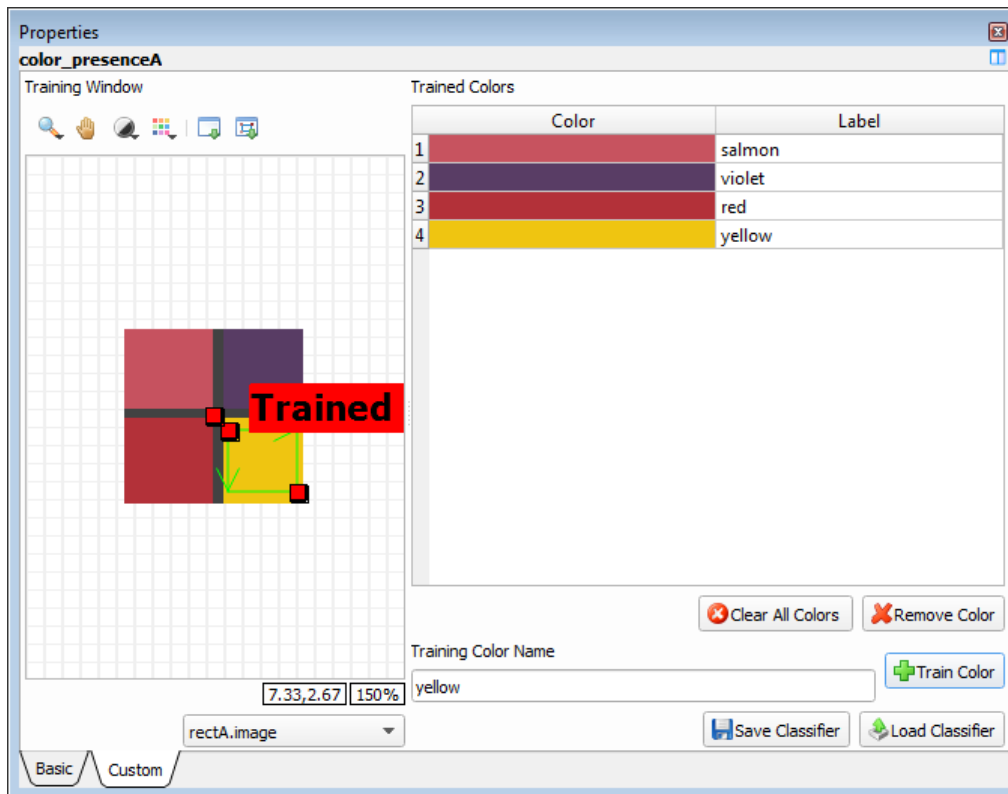
The **Search Edge** has three tabs at the top, for Train, Trained Models, and Model Edge Editor. The Train and Trained Models are the same as for Search Area. The Model Edge Editor is unique.

The **Model Edge Editor** tab allows you to delete some edges from the trained model, such as changing or extraneous features.



	Select the Coarse Edge Model for editing.		Remove a complete Edge from the model.
	Select the Fine Edge Model for editing.		Paint or Mask out part of an Edge in the model.
	Restore all edges removed (restore original).		

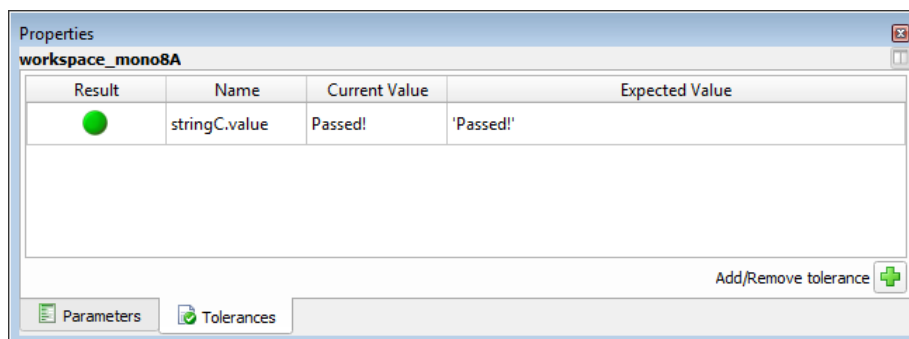
Color Training



Color Training – if you have a color camera or color image, appears when you create a Trained Colors object in the System Window and then click on the Trained Colors A object. The Configuration window changes to a Color Training window. You can enter a label (name) and train either a single point (mouse click in the image) or all samples within a rectangle. This creates a list of trained color samples. When you run the investigation, the list of samples is expanded into a 10M lookup table. **NOTE:** The first time you run an investigation after training or loading a saved program, the run time includes time taken to expand the color data into a lookup table. After the first run, the time is much shorter.

Tolerance

You can assign a tolerance test to any ROI or to any Image Workspace.



Use the Add button to open a pop-up menu of readings that you can select for a tolerance test.

Use the Expected Value field to set the maximum and minimum values for a Number, or set required value for a Boolean or a String.

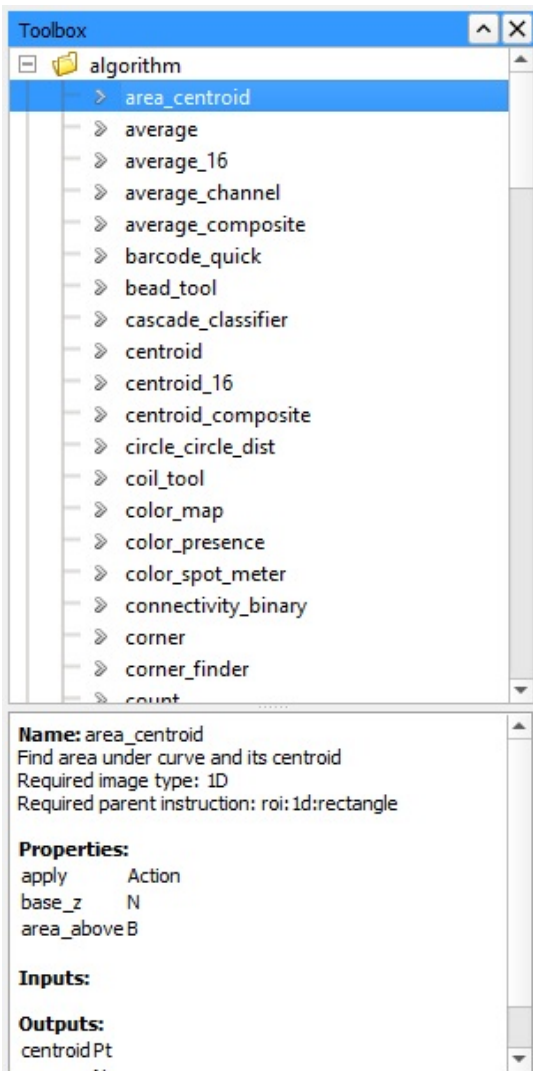
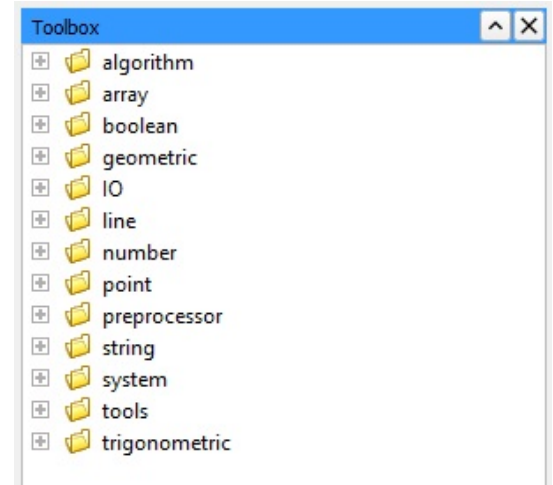
Select/highlight a value in the table and use the Remove button to delete the Tolerance test.

Toolbox Window

The Toolbox window lists all the instructions that can be added to a Sherlock program. You can show and hide the Toolbox window using the View menu in the Main window's Menu bar. If you click on an instruction, the lower pane gives a short-hand help.

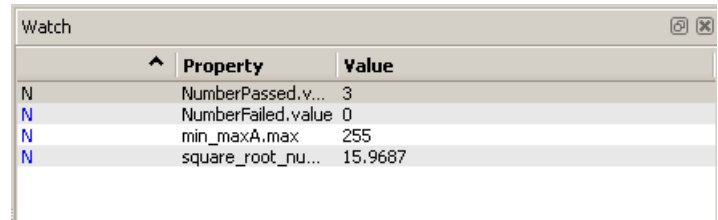
The Toolbox window default has all groups collapsed as shown to the right. You can expand individual groups to select instructions as shown below.

- You can drag instructions from the Toolbox window into the Program window.
- You can also right-click in the Program Window to insert the instructions.



Watch Window

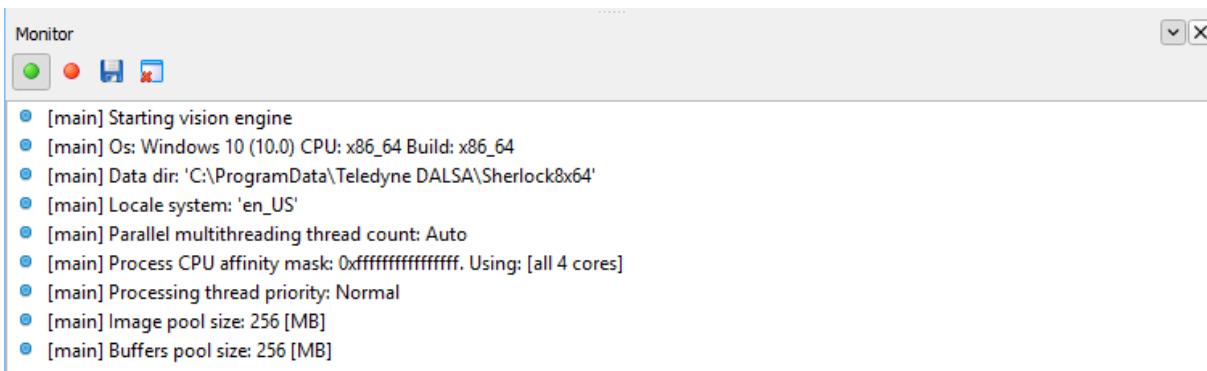
The Watch Window displays live data values as the inspection program executes. Drag outputs into the Watch Window, from the Program Window. Drag variables into the Watch Window, from the Program Window or from the System Window.



Tip: The Sherlock window layout is re-saved each time you close Sherlock 8. If the Watch Window was not selected open when you close Sherlock 8, it will not be open the next time you open Sherlock. Having objects displayed in the Watch window does not force it to open.

Monitor Window

The Monitor Window displays error messages, warnings, and instruction time information. The Monitor Window will pop open if it is already displayed. If the Monitor is docked and a tab, it does not force itself in front of other windows. You can change the level of errors and messages in the Program Options menu (page 34). The blue circles shown in this image indicate normal status messages. These occur when you launch Sherlock. Red circles will indicate errors.



As you add instructions to your program you may get messages that items are not specified. This is because a required argument or handle is empty when you create the new instruction. Use the Clear button in the toolbar to clear the Monitor window of all messages.

You can send messages to the Monitor window and to the Sherlock Log file Sh8Log.txt with the IO:System instruction log_message.

Reporter Window

The Reporter window displays messages that come from the reporter_print instruction used in a Sherlock program.

Numbers, values or variables can be evaluated for display, and the display precision can be set. For example; "The value is [numberA.value.3]" -- The numeric value of numberA will be displayed, with a precision of 3.



Tip: The IO:reporter instruction reporter_print sends a string to the Reporter window only. The IO:system instruction log_message instruction sends a string to the Monitor window and to the Sherlock 8 log file Sh8Log.txt.

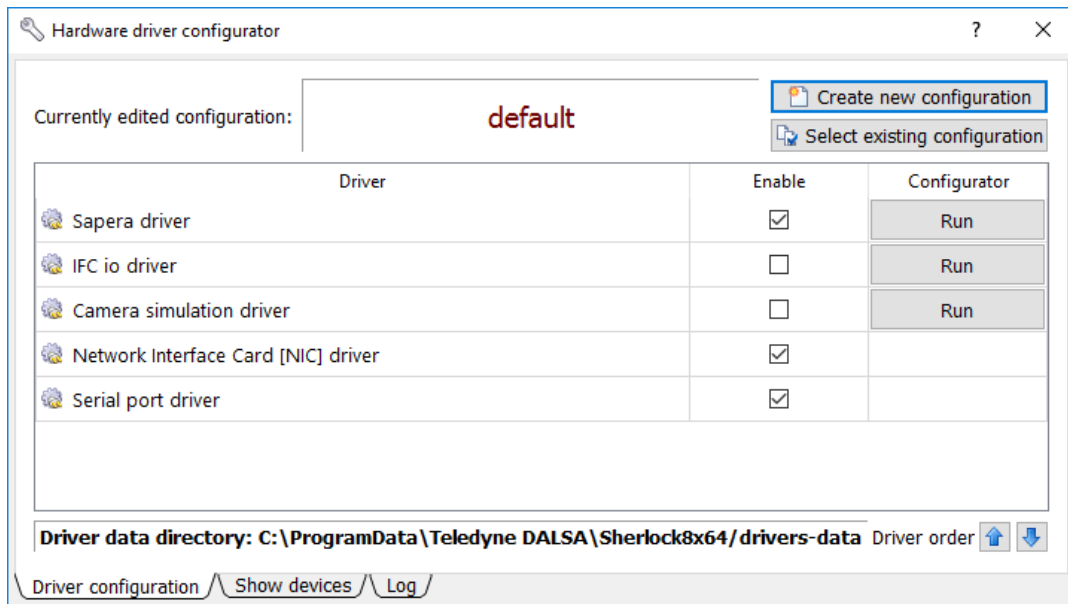
Tip: The Sherlock window layout is re-saved each time you close Sherlock 8. If the Reporter Window was not selected open when you close Sherlock 8, it will not be open the next time you open Sherlock. Writing messages to the Reporter window does not force it to open.

Acquisition Options and Setup Menus

In the Main window Menu bar, the Options menu selection “Configure Drivers” opens the Hardware driver configurator menu shown below.

Hardware Driver Configurator

The configuration is interactive, showing changes immediately. When you reopen Sherlock 8 the configuration reflects the default settings and the currently connected GigE devices. Opening Sherlock 8 or loading a saved program uses the current or default configuration.



Use the checkboxes to enable or disable drivers.

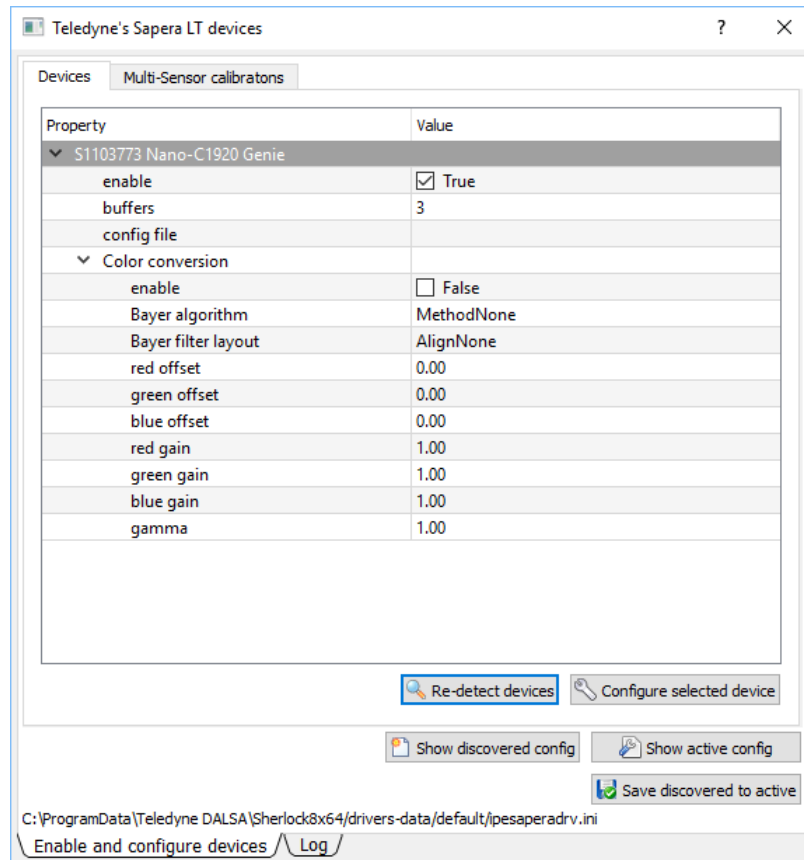
- Sopera driver – for GigE devices such as Genie Nano cameras or the Z-Trak Profiler. The Run button opens the GenICam Devices menu.
- IFC io driver – for the PL-USB IO Module or the VICORE built-in IO and VICORE dedicated PLC Port. The Run button opens a text file ifcdriver. You can edit this file to point to a PL-USB or VICORE IO configuration file created by GenICamSetup Configuration Tool. **Note:** You must run Sherlock with **Administrator Privileges** to load the IFC Driver and access I/O hardware such as the PL-USB or VICORE internal IO.
- Camera simulation driver – a software simulation of camera image sources. Creates a static grayscale image of dimensions you define in a text file. The Run button opens the text file where you can specify multiple “cameras” and their image sizes.
- Network Interface Card (NIC) driver – network ports on this PC or system.
- Serial port driver – serial ports on this system.

You can use the buttons at the top to create a new configuration or select an existing saved configuration. Clicking “Create new configuration” copies the “Factory” default settings to a new name you supply and uses the currently connected GigE devices. You can specify a saved configuration file by launching Sherlock from a command line.

You can change the order of the drivers using the arrow keys in the bottom right of this menu. Click on a driver and move it up or down in the list. This affects the index numbers assigned to your inputs and outputs. For example, in the image above the Sopera driver comes before the IFC driver. This means the individual camera IO are assigned to the lowest index numbers, followed by any IO associated with IFC, such as the PL-USB or VICORE.

Sapera LT Devices Menu

The Sapera LT devices menu shows all currently connected cameras or frame grabbers. There are two ways to reach the Sapera LT devices menu. From the Main window Options menu by selecting “Sapera LT Devices” or from the Hardware driver configurator menu by clicking the “Run” button beside Sapera driver.



Re-Detect Devices – rescan for newly connected devices, usually cameras or IO.

Configure selected device – Launches CamExpert or CamExpertX to configure settings in the hardware. If you have a Z-Trak profiler, Z-Expert opens. If you have Genie Nano or Linea GigE cameras, CamExpert opens. If you have a Teledyne frame grabber, CamExpert opens.

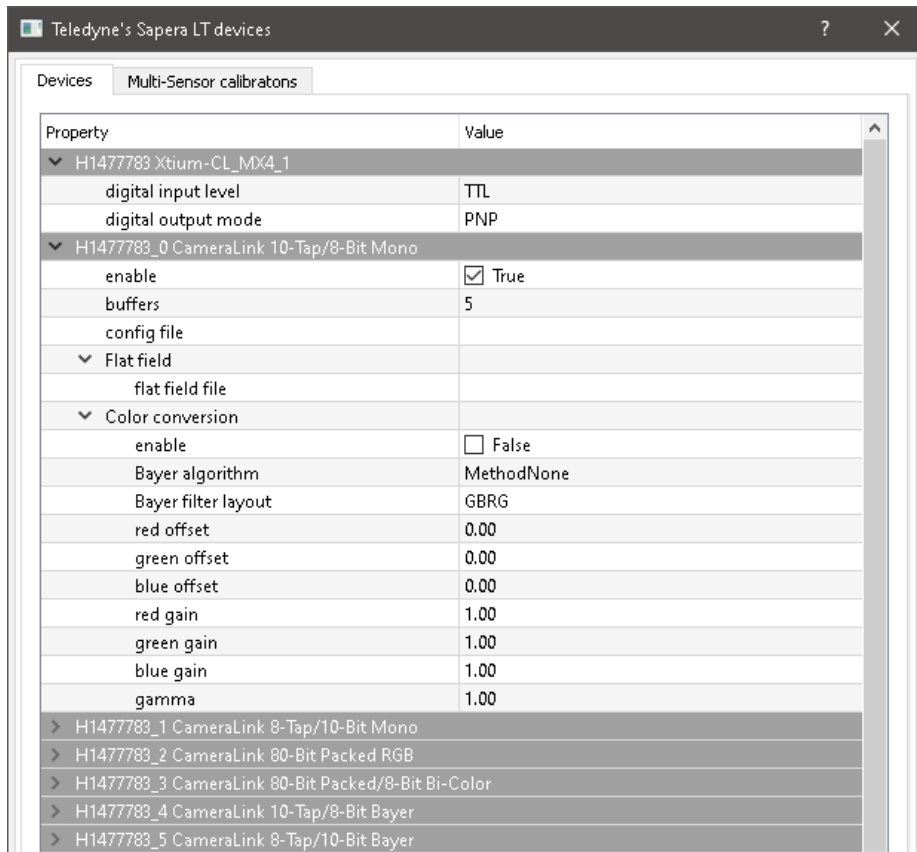
Show discovered config – shows what devices (cameras) are currently attached to the system.

Show active config – shows the current loaded default configuration. This may be blank if you have not saved a configuration since Sherlock was installed. Sherlock will still access all the attached cameras.

Save discovered to active – Saves the “discovered” list (actually attached devices) over the “active” (default saved) configuration. Sherlock 8 will look for the devices (cameras) in the saved list.

Note: Sherlock will attempt to use all supported devices attached unless they are disabled in this list and in the default configuration file.

The image below shows the Sapera LT devices menu with an Xtium-CL CameraLink frame grabber installed. You must expand one of the different camera data mode sections and enable it. In this example the 10-tab 8-bit mode has been enabled. Only one data mode can be enabled at one time.



Application Options

This menu allows you to configure GUI and program options in Sherlock 8. When you click "OK" a file "engine.ini" is saved in the User Data directory. There is a help page, and some float-over Tips in this menu.

Property	Value
▼ Engine	
Log messages to a file '[data dir]\log...	<input type="checkbox"/> False
▼ Performance	
Program end sleep [ms]	-1
Faster startup but only SYSTEM and ...	<input type="checkbox"/> False
Display stores image copy	<input checked="" type="checkbox"/> True
Temporary images pool max memor...	256.00
Optimized buffers pool max memor...	256.00
GUI update effort	Average
Message log level	Error

Engine

Log messages to a file '[data dir]\log.txt' [Bool] – Enable or disable saving a log file. The default is False, no log saved. Change to True to save a log file in the User Data directory. The "Message log level" controls the type or detail of messages saved in the log. At runtime you should disable saving a log file if possible.

Performance

Program end sleep [ms] – adds a small amount of "sleep" time to the end of the program execution loop, after the End instruction. This is used in Run Continuously mode to allow other processes access to the CPU.

Faster startup but only SYSTEM and DONGLE based license detection [Bool] – if True, license detection stops after a dongle or system license is detected. A license stored on a GigE device such as the Z-Trak will not be detected. If False, license detection will continue searching all attached GigE cameras for licenses even after a dongle or system license is detected.

Display stores image copy [Bool] - If True, the Image Windows has a separate copy of the image. This consumes more memory. If False the image Window does not have a separate copy. This consumes less memory but you will see processing changes occur "on the fly" and you may see flashing, tiling or tearing effects while the program runs.

Temporary images pool max memory size [MB] [Double] - This allows you to control the amount of memory used for a temporary images.

Optimized buffers pool max memory size [MB] [Double] - This allows you to control the amount of memory used for buffers.

GUI update effort [Enum] - This allows you to control the update priority of the GUI display.

Message log level [Enum] - controls the type of messages in the Monitor Window and in the Log file (if enabled). Selecting a level includes all messages of a higher level or nature. Trace is used for investigating problems with a program. Trace includes more verbose information on Sherlock's processes. At runtime you should use the lowest level of messages and disable saving a log file if possible.

CPU	
Number of CPU cores dedicated ...	-1
▶ Worker threads allowed CPU cores	Core0 Core1 Core2 Core3 Core4 Core5 C...
▶ GUI thread allowed CPU cores	Core0 Core1 Core2 Core3 Core4 Core5 C...
Processing thread priority	Normal

CPU

Number of CPU cores dedicated to image processing (-1=auto) - Controls the number of system CPUs that remain dedicated to Sherlock. The -1 value allows the system to automatically manage CPU allocation. Automatic control may be the best selection for a multi-core system. Dedicating too many cores to Sherlock can interfere with your Windows operating system's efficiency. **Note:** You must close and restart Sherlock after changing this value.

Worker threads allowed CPU cores [Boolean array]- Allows control over which cores are used for image processing. Double-click on the value field to display a drop-down of Boolean values for the individual cores.

GUI thread allowed CPU cores [Boolean array]- Allows control over which cores are used for updating the user interface (GUI). Double-click on the value field to display a drop-down of Boolean values for the individual cores.

Processing thread priority - Controls the priority of image processing threads in the CPU management.

Application	
Reload last program after application start	<input checked="" type="checkbox"/> True
Run program after application start	<input type="checkbox"/> False
Autosave time in minutes (-1: disable)	20
Style	Fusion
Style sheet	sherlock.qss
Maximum visible monitor widget lines	1000
Maximum visible reporter widget lines	1000
Create window on program new	<input checked="" type="checkbox"/> True
Enable monitor on startup	<input checked="" type="checkbox"/> True
Floating point display precision	2
Angles' unit of measure	Radians

Application

Reload last program after application start - If true, the last program is reloaded when you open Sherlock. Default value true.

Run program after application start - If true, Sherlock will start running continuously after loading the last program. If no program is loaded, Sherlock will not run.

Autosave time in minutes (-1:disable) - Sherlock will save the current program after the number of minutes in this field. If the field is -1 Sherlock will not automatically save.

Style [Enum] - select a predefined style for the look of Sherlock 8. The Value field is a drop-list.

Style sheet [String] - allows for future support of custom defined styles. The Value field is a string entry.

Pane titlebar button size [Integer] – allows you to increase or decrease the size of the control buttons (Close, Docking) in the child windows. This may be useful when using a high resolution setting on a smaller display monitor or laptop.

Maximum visible monitor widget lines – maximum number of lines displayed.







Maximum visible reporter widget lines – maximum number of lines displayed.

create window on program new [Bool] - enable or disable automatic creation of the image Workspace window when you create a new program from the File menu or Main toolbar.

Enable monitor on startup - forces the monitor window to enabled or live when Sherlock opens.

Floating point precision [Integer] - number of decimal places displayed in the GUI. Value field is a number entry. This affects how numbers are reported or displayed but not the internal precision of calculations.

Angles' unit of measure [Enum] - select Radians or Degrees. The Value field is a drop-list.

Display	
Tracker	
pass	
Color	 [50, 205, 50] (255)
Label color	 [255, 215, 0] (255)
fail	
Color	 [220, 20, 60] (255)
Label color	 [250, 128, 114] (255)
Label font	 [Terminal, 12]
Label transparent for not selected	<input type="checkbox"/> False
Handle color	 [255, 0, 0] (255)
Handle size	6
Selected Tracker	

Display

In the **display color** settings, the four numbers are Red, Green, Blue and Alpha. The Alpha value sets the transparency level. You can expand the entry to get four separate number entry fields. You can enter numbers or use the "Browse" button to open the Windows Color Selector.

Display:Tracker

The "Tracker" is the outline or graphic in the image Workspace, related to the ROI. These options are for display of **not-selected** or not-active ROIs in the image Workspace.

Display:Tracker:Pass

color [Integer array] - outline color of a passing un-selected ROI Tracker. Array format: [Red, Green, Blue] Transparency.

label color [Integer array] - color of the label for an un-selected passing ROI.

Display:Tracker:Fail

color [Integer array] - outline color of a un-selected failing ROI Tracker. Array format: [Red, Green, Blue] Transparency.

label color [Integer array] - color of the label for an un-selected failing ROI Tracker.

label font [Handle] - font style used to label the Tracker. Value field lets you Browse and Select a font on your PC.

label transparent for not selected [Bool] - makes the label transparent when the Tracker is not selected. *Default value:* false (not transparent).

handle color [Integer array] - color of the square re-size and rotate handles for the ROI Tracker.

handle size [Integer] - the size in pixels of the ROI Tracker handles.

Selected Tracker	
pass	
Color	[0, 255, 0] (255)
Label color	[255, 255, 0] (255)
fail	
Color	[255, 69, 0] (255)
Label color	[255, 0, 255] (255)
Font	A [Terminal, 12]
Selected extra info color	[0, 100, 0] (255)
Draw extra sizing info	<input checked="" type="checkbox"/> True
Markers	

Display:Selected Tracker

These options are for the **selected** ROI in the image Workspace.

Display:Selected Tracker:Pass

color [Integer array] - outline color of a selected passing ROI Tracker. Array format: [Red, Green, Blue] Transparency.

label color [Integer array] - color of the label for a selected passing ROI Tracker.

Display:Selected Tracker:Fail

color [Integer array] - outline color of a selected failing ROI Tracker. Array format: [Red, Green, Blue] Transparency.

label color [Integer array] - color of the label for a selected failing ROI Tracker.

font [Handle] - font style used to label the ROI Tracker. Value field lets you Browse and Select a font on your PC.

selected extra info color - [Integer array] - Color of supplemental information when you resize.

draw extra sizing info [Bool] - enables the "extra info" display.

Markers	
Label font	A [Terminal, 12]
Label text color	[255, 0, 0] (255)
String font	A [Terminal, 12]
String color	[255, 255, 0] (255)
String back color	[0, 0, 255] (255)
String background	<input type="checkbox"/> False
Show name in label	<input type="checkbox"/> False
Show value in label	<input checked="" type="checkbox"/> True
Point color	[0, 0, 255] (255)
Point array color	[0, 0, 255] (255)
Line color	[255, 0, 0] (255)
Line array color	[255, 0, 0] (255)
Point size	15
Alignment color	[255, 255, 0] (255)
Alignment size	24

Display:Markers

"Markers" are the extra information displayed in the image Workspace, such as alignment points, labels, readings, result or reading points and lines drawn in the image Workspace.

label font [Handle] - font style used in labels. Value field lets you Browse and Select a font on your PC.

label text color [Integer array] - display color of the text in a label.

string font [Handle] - font style used in display text. Value field lets you Browse and Select a font on your PC.

string color [Integer array] - display color of strings.

string back color [Integer array] - color used to fill a string background.

string background [Bool] enables or disables filling a string background.

show name in label [Bool] includes the instruction name in the label.

show value in label [Bool] includes the reading in the label.

point color [Integer array] - color of individual readings, results or construction points.

point array color [Integer array] - color of array readings, results or construction points.




line color [Integer array] - color of single reading, result, or construction lines.

line array color [Integer array] - color of array reading, result, or construction lines.

point size [Integer] size in pixels of reading points.

alignment color [Integer array] - color of the alignment anchors.

alignment size [Integer] - size of the alignment anchors in pixels.

Profile 1D	
▶ Color	 [255, 0, 0] (255)
Size	30.00
▶ Invalid color	 [255, 255, 0] (255)
Invalid size	30.00
▶ Major axis color	 [255, 255, 255] (50)
▶ Minor axis color	[255, 255, 255] (30)

Display: Profile 1D

Options specific to the 1D "image" data display.

Color [Integer array] - color of the laser scan line display or good data points.

size [Integer] - size of the laser scan line display.

invalid color [Integer array] - color of invalid point positions in the laser scan line display.

invalid size [Integer] size of the invalid scan line points.

major axis color [Integer array] - color of the larger unit grid lines. The two grids are overlaid. 10 minor grid lines fit between major grid lines. The default value for Alpha is usually low (50) for a light, non-distracting grid. You can increase the Alpha value to strengthen the grid in the image.

minor axis color [Integer array] - color of smaller unit grid lines. The two grids are overlaid. 10 minor grid lines fit between major grid lines. The default value for Alpha is usually low (30) for a very light, non-distracting grid. You can increase the Alpha value to strengthen the grid in the image.

TCP/IP Command Line Interface	
Enable	<input type="checkbox"/> False
Address	0.0.0.0
Port	5050

TCP/IP Command Line Interface

enable [Bool] - enable using the command line interface to TCP/IP.

address [Double] - IP Address of the TCP device.

port [Integer] - port number of the TCP device.

▼ Program	
▼ Images	
Save images	<input checked="" type="checkbox"/> True
Save indented	<input checked="" type="checkbox"/> True
▼ Dir	
Programs	C:\ProgramData\Teledyne DALSA\Sherlock8x64/programs
Images	C:\ProgramData\Teledyne DALSA\Sherlock8x64/images
▼ OpenGL	
Driver mode for 3D plot rendering	UseDesktopOpenGL

Program

Program:Images

save images [Bool] - enable automatically saving Workspace images to image files.

save indented [Bool] – create subdirectories.

Program:Dir

programs [String] - full path to the default directory for saved programs.

images [String] - full path to the default directory for saved images.

OpenGL

Driver mode [Enum] - select the OpenGL (Open Graphics Library) driver to use. Changing the driver may improve rendering of 2D and 3D images, such as the 2.5D or 3D Profiler surface. The performance depends upon your PC resources for graphics. Options are: UseDesktopOpenGL, UseOpenGLES, UseSoftwareOpenGL.

Script Editing

Sherlock 8 incorporates a reduced instruction set of the JavaScript editor. Many web functions that do not apply to Sherlock have been eliminated.

Tip: A very good resource for JavaScript is available on-line at www.w3schools.com/js. Remember that not all functionality of JavaScript is available in Sherlock.

The JavaScript editing window appears when you click beside "code" in the Properties window, or when you double-click on the "code" input in the Program window. The instructions: script, if, if-else, and while all use the JavaScript edit window. The JavaScript editing window is the same for all script entry.

The larger field is for script entry. You can type any statements you wish. The bottom field is for messages and will display errors when you click "Check Syntax" or it will display the result when you click "Evaluate".

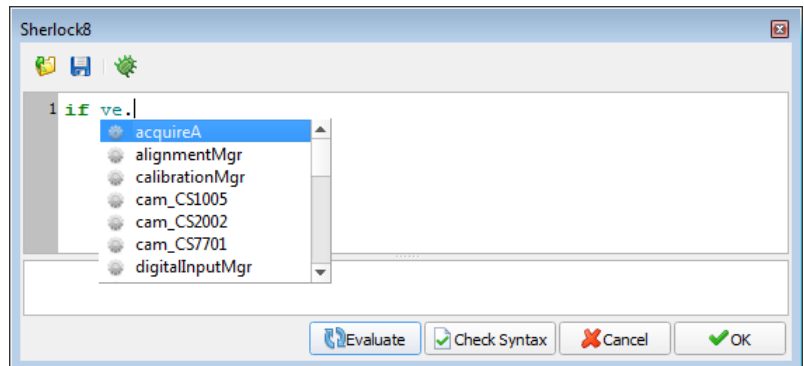
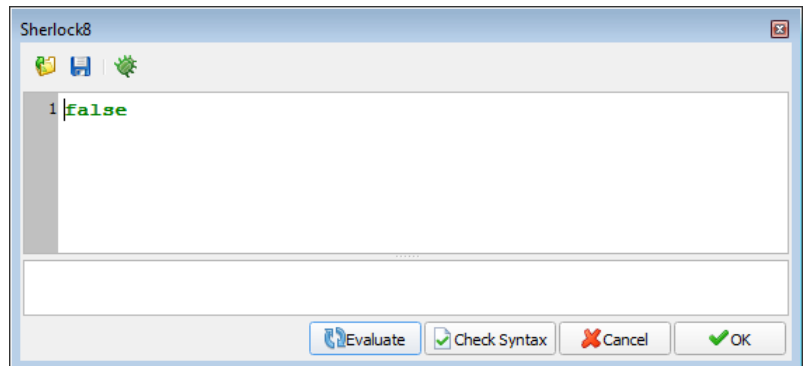
NOTE: You must click the "OK" button to add the script to your program.

The code entry window has "intellisense" support. When you type **ve.** (it must be




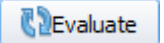
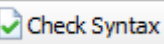
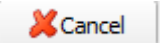
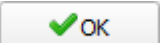
lower case and followed by a period) a popup will appear. You can click on any item in the popup to add to your script statement. If you type another period, another popup appears.

Important: "ve" is a special variable that links the Sherlock program environment to the JavaScript environment. But "VE" and "Ve" are not special variables. The intellisense editor will not correct your capitalization.

Note: "numberA" is a variable name. "numberA.value" is the value of the variable.



Script Editing Window Controls

	Open a saved script file and paste the contents into the entry field. Note: This will overwrite the entry field!
	Save all statements in the script field to a file
	Open a JavaScript reference text window.
	Evaluate the script entry field (all statements).
	Check syntax in the script entry field.
	Close the edit window without saving.
	Save the script field in Sherlock and close the edit window.

JavaScript in Sherlock

JavaScript is a scripting language developed by Netscape to enable Web authors to build interactive sites. Although it shares many of the features and structures of the full Java language, it was developed independently. **JavaScript is not Java.**

Sherlock 8 incorporates a reduced instruction set of the JavaScript editor. Many web functions that do not apply to Sherlock have been eliminated.

A JavaScript code module can be used to perform tasks that "native" Sherlock code either cannot perform easily or cannot perform at all.

The JavaScript language includes many common programming constructs, such as: **For, While, Do/While, If, If/Else, Switch**, etc.

IMPORTANT: "ve" is a special variable that links the Sherlock program environment to the JavaScript environment. But "Ve" "vE" and "VE" are not special variables. They will not connect to the Sherlock program environment. The editor will not correct your capitalization.

The following special variables are defined in Sherlock 8:

- ve - "vision engine" represents the Sherlock engine object.
- vs - "vision service" represents Sherlock services, such as logs, etc.
- vt - "vision types" represents Sherlock related types.
- vp - "vision processing" represents the library of processing related calls.

A very good resource for JavaScript is available on-line at www.w3schools.com/js. Remember that not all functionality of JavaScript is available in Sherlock.

Using JavaScript Objects

JavaScript is an object-oriented language. A JavaScript object encapsulates members and methods for different types of operations.

JavaScript Variables

JavaScript variables do not usually have to be declared. A variable's type and size are determined at runtime based on context.

When you assign a Sherlock array variable to a JavaScript variable, the type and array size of the JavaScript variable are automatically set.

Because variables are dynamic in JavaScript, there is no type checking when passing values to Sherlock. Assigning the wrong type of JavaScript variable to a Sherlock variable will not generate a syntax error but will generate a runtime error.

A JavaScript variable's type will change if you assign it different types of data. (But this is not good coding practice!)

All JavaScript variables are local to the module in which they appear. They are destroyed when the code module is exited; their values are not maintained between calls to the module.

JavaScript variable names are case sensitive. *jsNumber* is not the same variable as *JSnumber*.

JavaScript cannot recognize variables with embedded spaces. The JavaScript engine will generate an error at runtime. (The engine will generate the same error in the JavaScript Code Edit Window if you click the **Check Syntax** button.)

Modifying Arrays from JavaScript

How you access a Sherlock array from within a JavaScript module determines the interaction between the two.

By reference: The JavaScript number array JSarrayNum is a reference to the Sherlock number array ve.arrayNum; they access the same memory.

```
JSarrayNum = ve.arrayNum.value
for (index=0; index<JSarrayNum.length; index++){
    JSarrayNum[index] = JSarrayNum[index] + 2
}
```

ve.arrayNum has been modified; all of its entries have been incremented by 2.

By value: The JavaScript number array JSarrayNum is a copy of the array Sherlock number array ve.arrayNum; they do not access the same memory.

```
var JSarrayNum = new Array()
for (index=0; index<ve.arrayNum.value.length; index++){
    JSarrayNum[index] = ve.arrayNum.value[index]
}
for (index=0; index<JSarrayNum.length; index++){
    JSarrayNum[index] = JSarrayNum[index] + 2
}
```

ve.arrayNum has NOT been modified; it contains its original data

Accessing Sherlock Data Types in JavaScript

You can read from or write to Sherlock data types in JavaScript.

```
ve.point0.value = vt.Point(6,7)
//
jsCentroid0 = vt.Point // Not absolutely necessary, Without this line, jsCentroid0 will be set to type vt.Point in the
next line
jsCentroid0 = ve.connectivity_binaryA.centroid0
ve.sh8Centroid0 = jsCentroid0
//
ve.pt0x.value = jsCentroid0.x
ve.pt0y.value = jsCentroid0.y
//
//
jsCentroids = vt.PointVector() // Not absolutely necessary. Without this line, jsCentroids will be set to type
vt.PointVector in the next line
jsCentroids = ve.connectivity_binaryA.centroid
ve.sh8Centroid1.value = jsCentroids1
//
ve.pt1x.value = jsCentroids1.x
ve.pt1y.value = jsCentroids1.y
```

Sherlock Plugin Calls

Only limited set of plugins is supported; (calibration, alignment, digital IO).

An object "arg" can be used to set any property of the plugin before the call.

This plugin call (to display_message) has no outputs:

```
var arg = new Array
arg["message"] = "hello"
ve.display_message(arg)
```

Here is a plugin call (to get_dig_io) returning values:

```
arg = new Array
arg["channel"] = 1
var ret
ret = ve.get_dig_io(arg)
if(ret["state"] == true){
//instructions for true
}else{
//instructions for false
}
```

Plugins taking infinite inputs (variance_number) take array of values instead:

```
arg = new Array
ret = new Array
arg["number"] = new Array(1,2,3,1,1,1,1,1,1,1)
ret = ve.variance_number(arg)
ve.varC.value = ret["variance"]
```

JavaScript Operators

Comparisons

- == Equal
- != Not equal
- < Less than
- <= Less than or equal
- > Greater than
- >= Greater than or equal

Arithmetic Operations

- + addition
- subtraction
- * multiplication
- / floating point division (12 /5 = 2.4)
- % modulus division. Returns the integer remainder of dividing the two operands (12%5 = 2)
- (Two minus signs, no space) Unary decrement. Subtracts one (1) from its operand. The operator can be prefix or postfix.

as prefix: `x = 3 ; y = --x` sets `x` to 2 and `y` to 2. The prefix decrements the value of `x` first, then assigns the value of `x` to `y`.

as postfix : `x = 3 ; y = x--` sets `y` to 3 and `x` to 2. The postfix assigns the value of `x` to `y` first, then decrements `x`.

++ Unary increment. Adds one (1) to its operand. The operator can be prefix or postfix.

as prefix : `x = 17 ; y = ++x` sets `x` to 18 and `y` to 18. The prefix increments the value of `x` first, then assigns the value of `x` to `y`.

as postfix : `x = 17 ; y = x++` sets `y` to 17 and `x` to 18. The postfix assigns the value of `x` to `y` first, then increments `x`.

Logical (Boolean) Operations

&& (Logical AND) `expr1 && expr2` Returns true if both operands are true; otherwise, returns false.

|| (Logical OR) `expr1 || expr2` Returns true if either or both operands are true; if both are false, returns false.

! (Logical NOT) `!expr` Returns false if its single operand is true; otherwise, returns true.

Built-In Objects in JavaScript

JavaScript, like all languages, has intrinsic or "built-in" objects and methods. Here are some of the most commonly used objects and methods. You can find definitions, more information, and examples on the World Wide Web.

All Objects, their Properties, and Methods are case sensitive.

Number Object

The JavaScript Number object has the following Read-Only properties.

`MIN_VALUE` - returns the minimum or smallest number JavaScript can handle.

`MAX_VALUE` - returns the maximum or largest number JavaScript can handle.

`POSITIVE_INFINITY`

`NEGATIVE_INFINITY`

The number object has the following methods.

`toExponential`, `toFixed`, `toPrecision`, `toString`, `valueOf`

Boolean Object

The JavaScript Boolean object can be instantiated using the **new** keyword but is most often a variable set to a true or false value.

The JavaScript Boolean object has a number of values that qualify as a value equal to false (0, -0, null, or "" [an empty string]), undefined (NaN), and false. All other Boolean values qualify as a value equal to true.

The boolean object (and all other built-in objects) has the following methods, but they are rarely used.

`toString`, `valueOf`

String Object

The JavaScript String object is a wrapper for text values. The String object does not need to be instantiated to be used. For example, you can set a variable to a string, and then all of the String object properties or methods will become available to that variable.

The string object has one Read-Only property.

`length`

The string object has the following methods.

`charAt`, `charCodeAt`, `concat`, `fromCharCode`, `indexOf`, `lastIndexOf`, `match`, `replace`, `search`, `slice`, `split`, `substr`, `substring`, `toLowerCase`, `toUpperCase`, `toString`, `valueOf`

Date Object

The JavaScript Date object provides a way to work with dates and times. You can instantiate it in a number of different ways depending on the desired results. You can instantiate it with or without arguments.

The JavaScript Date object has the following methods.

getDate, getDay, getFullYear, getHours, getMilliseconds, getMinutes, getMonth, getSeconds, getTime, getTimezoneOffset,
setDate, setFullYear, setHours, setMilliseconds, setMinutes, setMonth, setSeconds, setTime,
getUTCDate, getUTCDay, getUTCFullYear, getUTCHours, getUTCMilliseconds, getUTCMinutes, getUTCMonth, getUTCSeconds
setUTCDate, setUTCFullYear, setUTCHours, setUTCMilliseconds, setUTCMinutes, setUTCMonth, setUTCSeconds
toDate, toString, toLocaleDateString, toLocaleTimeString, toLocaleString, toTimeString, toUTCString, valueOf

Array Object

The JavaScript Array object has one Read-Only property.

length

The JavaScript Array object has the following methods.

pop, push, shift, unshift, sort, reverse, slice, splice, toString, valueOf

The sort() function sorts arrays of numbers (N[]), strings (S[]) and Booleans (B[]). It cannot sort arrays of points (P[]) or lines (L[]). The sort order can be either alphabetic or numeric, ascending or descending. The default is alphabetic and ascending. When numbers are sorted alphabetically and ascending, '40' comes before '5'.

This function changes the original array. You cannot directly sort an algorithm or instruction's output array; you must save it to an array variable, and sort the array variable.

Example

Engine.numAreas is a number array variable (N[]). This sorts the array alphabetically, in ascending order:

```
ve.numAreas.value.sort()
```

To perform a numeric sort, you must pass a function as a parameter. The function specifies whether the numbers are to be sorted in ascending or descending order, numerically. This sorts the array numerically, in descending order:

```
ve.numAreas.value.sort(function(a,b){return b-a})
```

The (a,b) is standard JavaScript notation. b-a is descending order (b through a), a-b is ascending order (a through b).

Math Object

The JavaScript Math object is used to perform mathematical functions. It is not instantiated. You use the Math object and call properties and methods from it without any instantiation, for example, [Math.random](#).

The JavaScript Math object has the following Read-Only properties.

E, LN2, LN10, LOG2E, LOG10E, PI, SQRT1_2, SQRT2

The JavaScript Math object has the following methods.

abs, acos, asin, atan, atan2, cos, exp, log, pow, sin, sqrt, tan, ceil, floor, round, max, min, random, toString, valueOf

Debugging Tools

The debugging tools available fall into the following categories:

- Execution Modes change how Sherlock runs, without making instruction changes to the program.
- Disable Execution causes any one instruction to be skipped. Every instruction has an "enable" property. By changing the property to "false" the instruction is not executed.
- Breakpoints cause execution to pause at each breakpoint instruction.






NOTE: Remember to turn off all debugging modes and re-enable all execution when finished!

Execution Modes

Single Step Mode




Run one instruction and pause. Each instruction is highlighted in single step mode.

Run one instruction and pause. Each instruction is highlighted in single step mode.

1. Click the "single step execution mode" button  in the Main toolbar, or select "single step execution" in the Run menu. - The program does not start yet.
2. Click the "Run once" button  in the Main toolbar, or select "run once" in the Run menu. The program execution starts, and pauses after the first instruction.
3. Click the "execute single instruction" button  in the Main toolbar or select "execute single step" in the Run menu. Each time you click the button, the program executes one instruction and pauses.
4. Click the "Abort" button  at any time, to stop execution immediately.
5. Click the "single step execution mode" button  again to turn off single-step execution mode.

Highlight Mode

Each instruction is highlighted when executed. The program runs slower during highlight executed instruction mode.


1. Click the "highlight executed instructions" button  in the Main toolbar.
2. Click the "Run once" button  or "Run continuously" button . The program starts to run, and each instruction is highlighted as it is executed.
3. Click the "Abort" button at any time, to stop execution immediately.
4. Click the "highlight executed instructions" button to turn off this mode.

Disable Execution

Instructions

All instructions have an "enable" property. You can temporarily disable any instruction without deleting it. Any individual instruction can be skipped during program execution including ROIs and their processing. You can disable multiple instructions. Many instructions can be disabled by right-clicking on the instruction in the Program window and clearing the Enable checkbox that appears below the Breakpoint checkbox.

1. Right-click on an instruction in the Program window and clear the "Enabled" checkbox at the bottom of the popup menu, or
2. In the Properties window, click on the value field beside "enabled".
3. Click a second time to clear the check box. The value changes to "false".

4. A red circle with a white bar appears to the left of the disabled instruction, on the instruction icon.
5. Click the "Run once" button . The program executes but skips the disabled instructions.
6. To re-enable an instruction, right-click on the disabled instruction in the Program window.
7. In the pop-up, check the "Enable" box at the bottom of the list, or in the Properties window click in the value field beside "enabled" and check the box to change the value to true again.

NOTE: Remember to re-enable all instructions when finished debugging. The enabled/disabled is a property value change and it will be saved in the program. Re-enable all instructions before saving the final program, unless you want the instructions to be disabled.

Preprocessors and Algorithms







An ROI's preprocessors and algorithms can be temporarily disabled, without deleting or changing their settings.


1. Click on an ROI instruction in the Program Window.
2. In the Properties window, click the Custom tab. The Custom tab does not appear if you have enabled "custom window display" which shows the Basic and Custom properties side-by-side.
3. On the left side beside each Preprocessor and Algorithm is a checkbox. Clear a checkbox to temporarily disable executing that preprocessor or algorithm.

NOTE: Remember to re-enable all processors when finished debugging. This is a property value change and it will be saved in the program file.

Breakpoints

A breakpoint causes the program to pause when the instruction is reached. You can have multiple breakpoints in a program.

1. Right-click on an instruction in the Program window.
2. At the bottom of the instructions popup click the box beside "breakpoint".  In the Program window, a red pause symbol appears beside the instruction name, on the instruction icon.
3. Click the "Run once"  or "Run continuously"  button. The program executes until the breakpoint is reached.
4. Click the "execute single instruction" button  to execute one instruction at a time.
5. Click the "single step execution mode" button  to continue execution to the next breakpoint.
6. Click the "Abort" button  to stop execution immediately.
7. To remove a breakpoint, right-click on the instruction. At the bottom of the instruction popup, clear the checkbox

beside "breakpoint". 

NOTE: The breakpoint is a flag set on an instruction. This flag will be **saved** in the program if it is still set. You must remember to clear all breakpoints before saving the final program.