

Use Generic Types and Native Libraries

Contents

Introduction	2
Sample 1 – Hello World!	3
Target Platform	3
Support files	4
Create method	5
Create Image object	6
Create font	11
Draw text	13
Create a window to display image	16
Show image	18
Wait for key press	20
Destroy window	21
Test	22
Sample 2 – Shape Detection	23
Use Image Box	25
Create Shape Detection Method	29
Load image from file	29
Create image object	29
Convert Image to Gray	35
Create Action List	42
Circles Detect	44
Lines Detection	51
Create a Gray object named “cannyThresholdLinking”	51
Create a Gray image named “cannyEdges”	52
Detect lines	54
Triangles and Rectangles Detection	59
Create variable triangle list	59
Create variable for rectangle list	61

Use Generic Types and Native Libraries

Create storage variable	62
Go through shapes	63
Process Each Shape.....	71
Get current contour	71
Check contour size	75
Get triangle	78
Get rectangle.....	87
Show original image.....	115
Display triangles and rectangles	117
Create a blank image	117
Go through all triangles	118
Go through all rectangles.....	122
Make action group	126
Display Circles	129
Create a blank image	130
Go through circles.....	131
Display image	134
Make action group	135
Display Lines.....	136
Create blank image	136
Go through lines.....	138
Display image	141
Make action group	142
Execute the method.....	143
Feedback	145

Introduction

To create a standalone Windows program, Limnor Studio uses Microsoft .Net Framework libraries, or called “managed libraries”. To use “unmanaged libraries”, usually created in C/C++, managed libraries

Use Generic Types and Native Libraries

are created to wrap the unmanaged libraries. For example, Emgu CV is such a managed library for Intel Open CV libraries. See <http://sourceforge.net/projects/emgucv/>

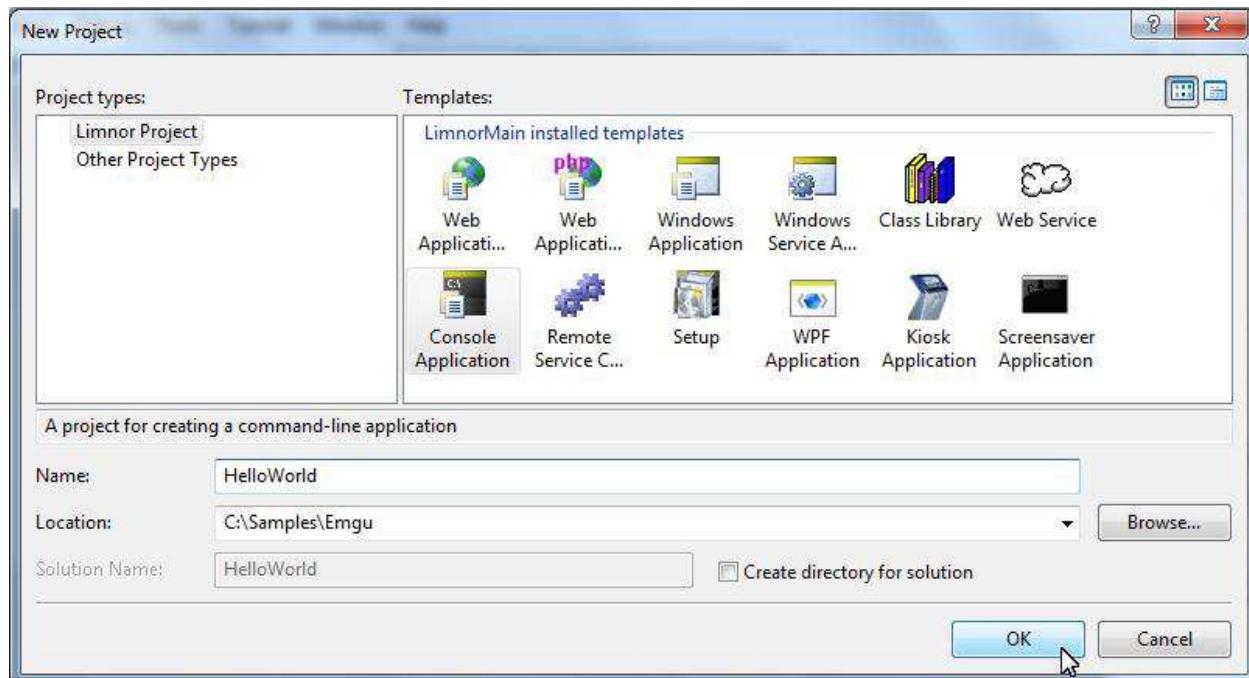
This document re-creates the samples from the Emgu CV tutorial C# samples, using Limnor Studio. It shows how to use generic types in the Emgu CV library.

It shows how to add native libraries files to the project so that those unmanaged library files can be collected into the distribution folder by Limnor Studio. It shows the use of Action Groups to make it easy to show program logic.

Sample 1 – Hello World!

This sample is based on an Emgu sample found at folder “Emgu.CV.Example\HelloWorld” under your Emgu installation folder.

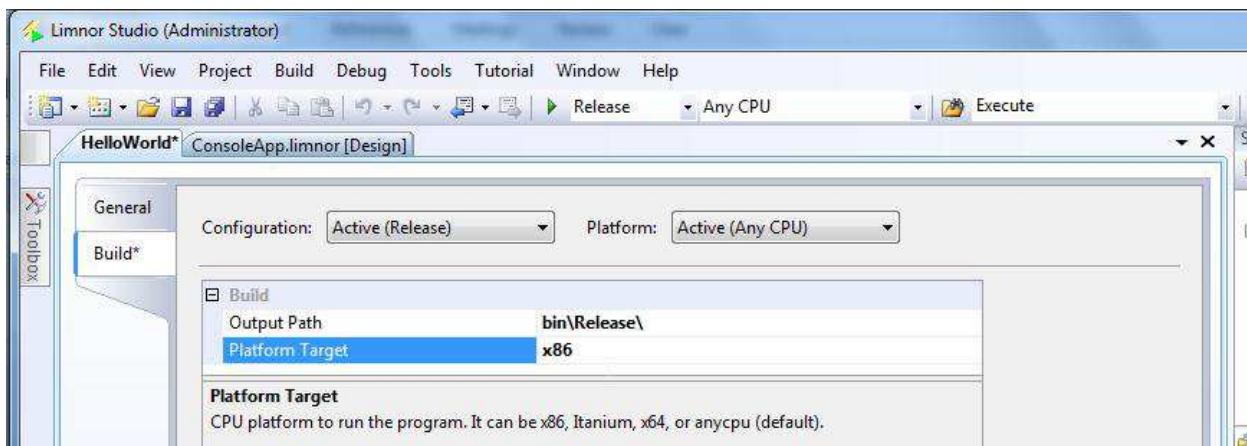
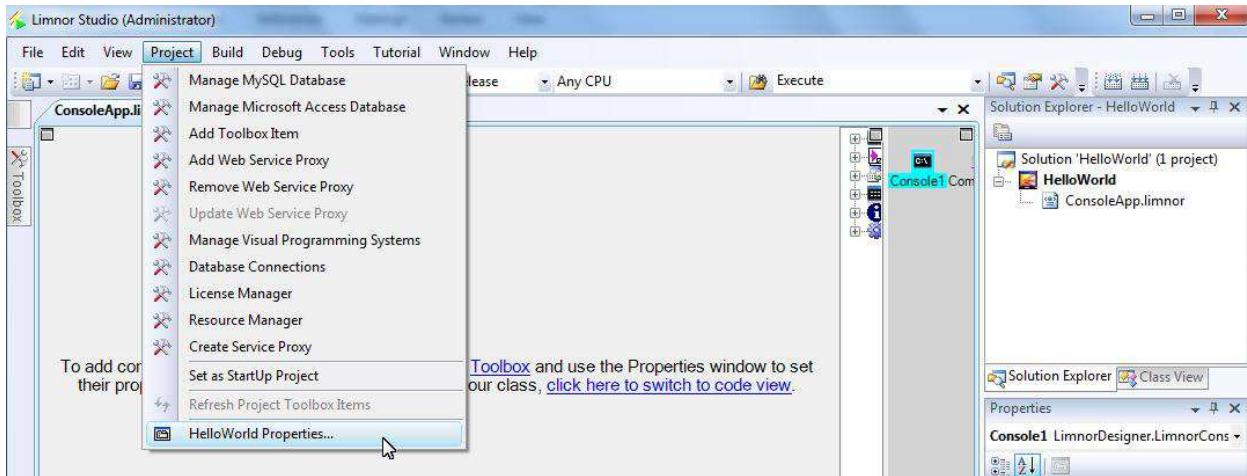
Create a console project:



Target Platform

Set the platform to x86, because we downloaded 32-bit Open CV C++ library.

Use Generic Types and Native Libraries



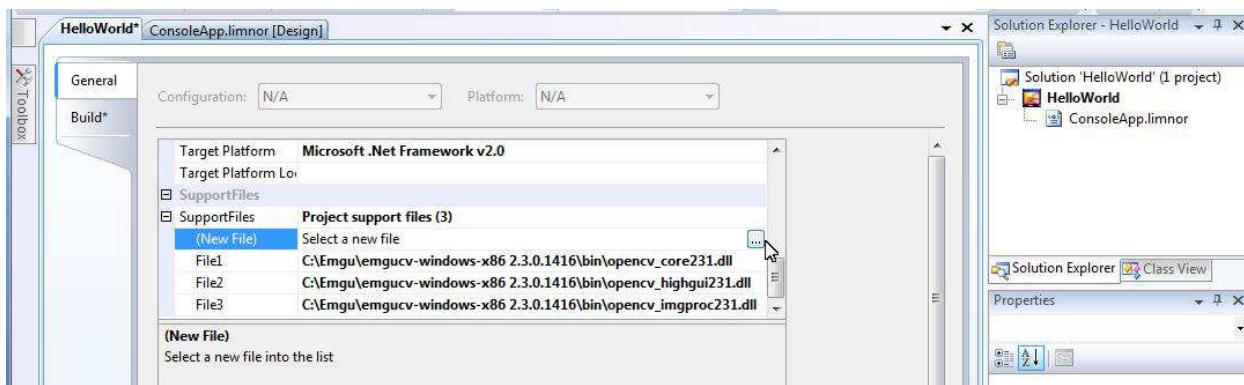
Support files

Because Emgu is a wrapper on unmanaged libraries, we need to tell Limnor Studio which support files the wrapper needs. You need to get such information from the library manufacturer or library documentation. In this sample, the following support files are needed:

opencv_core231.dll
opencv_highgui231.dll
opencv_imgproc231.dll

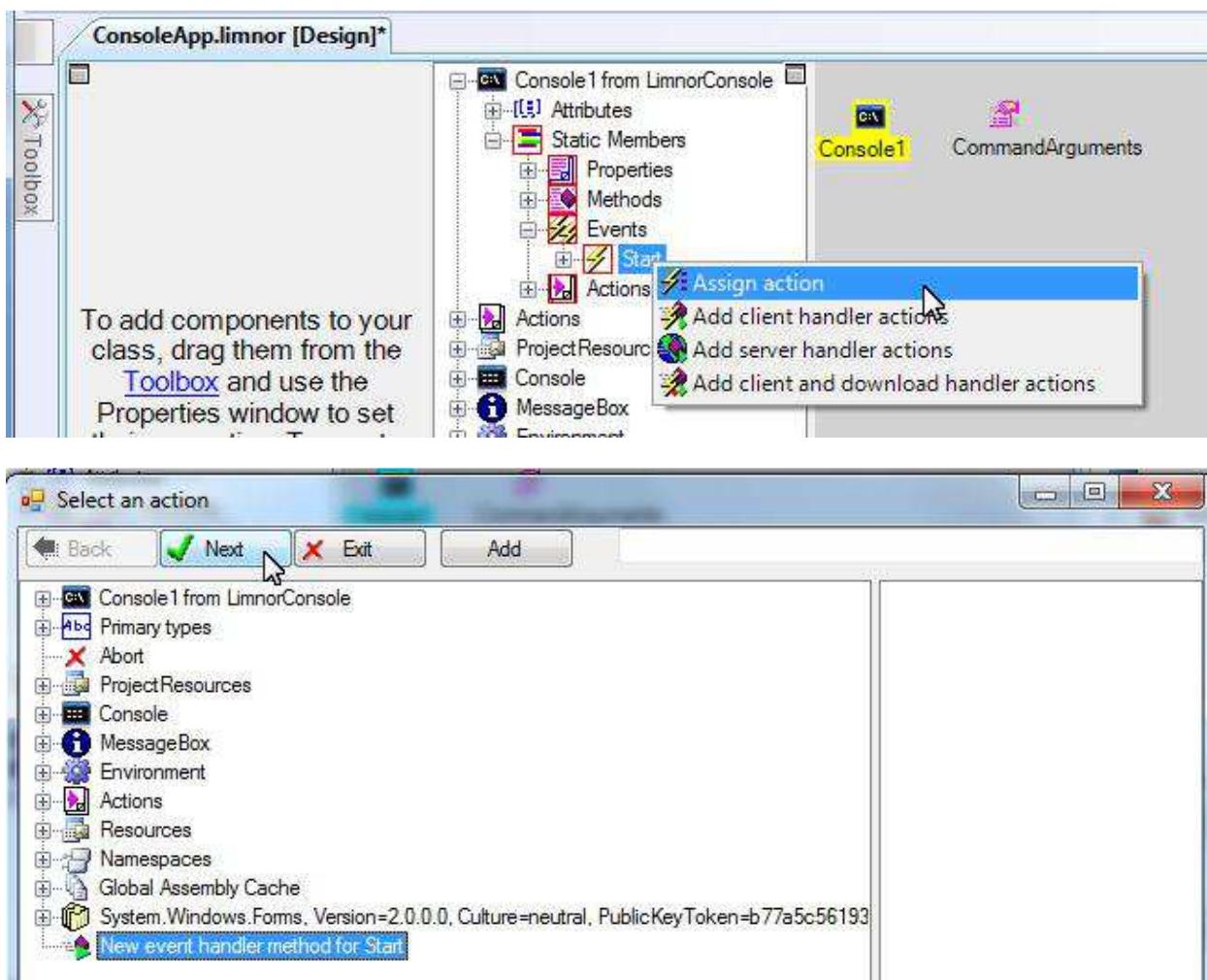
We need to add them to the support file list:

Use Generic Types and Native Libraries



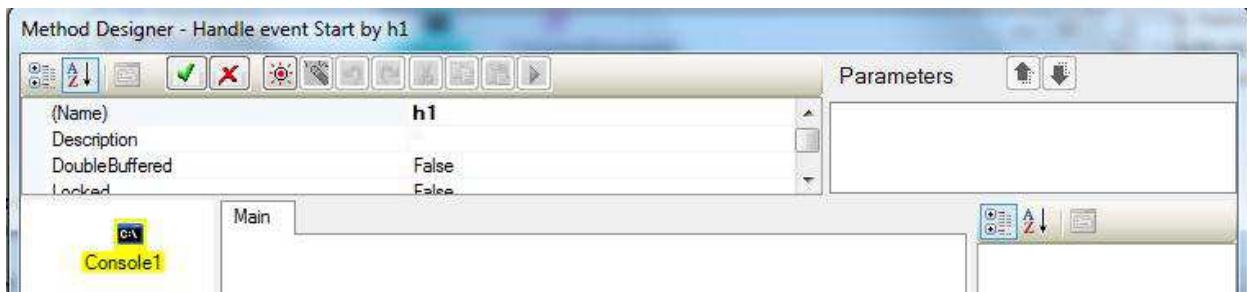
Create method

In a method, we may create variables using the classes from the Emgu CV library. For this sample, we simply create an event handler method:



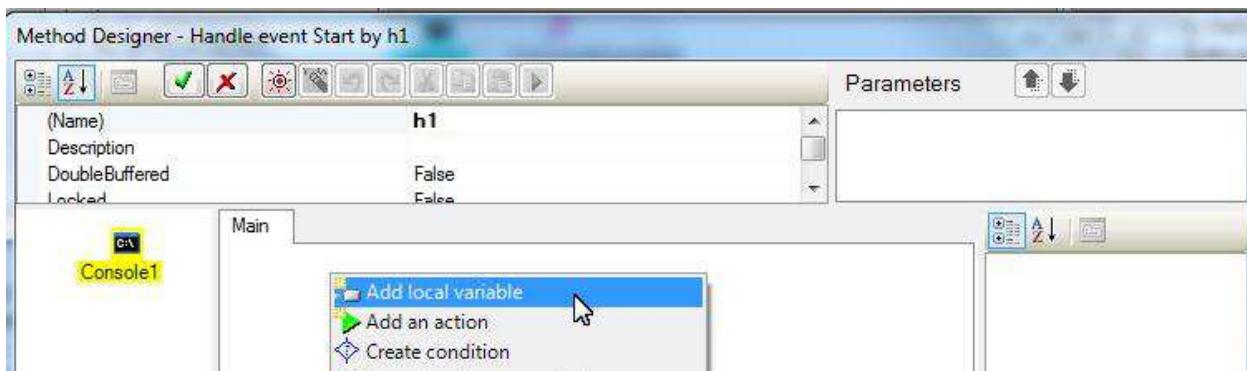
A method editor appears.

Use Generic Types and Native Libraries

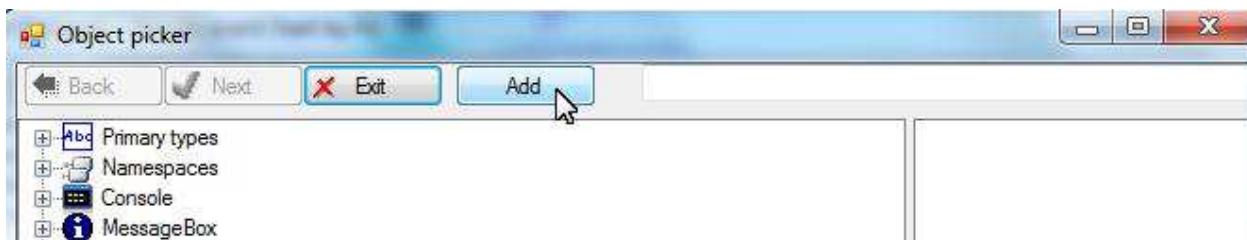


Create Image object

Emgu uses Image object to draw text on the screen. Let's create an image variable.



Click Add to add a class from an external library. In this sample, it is the Emgu library.

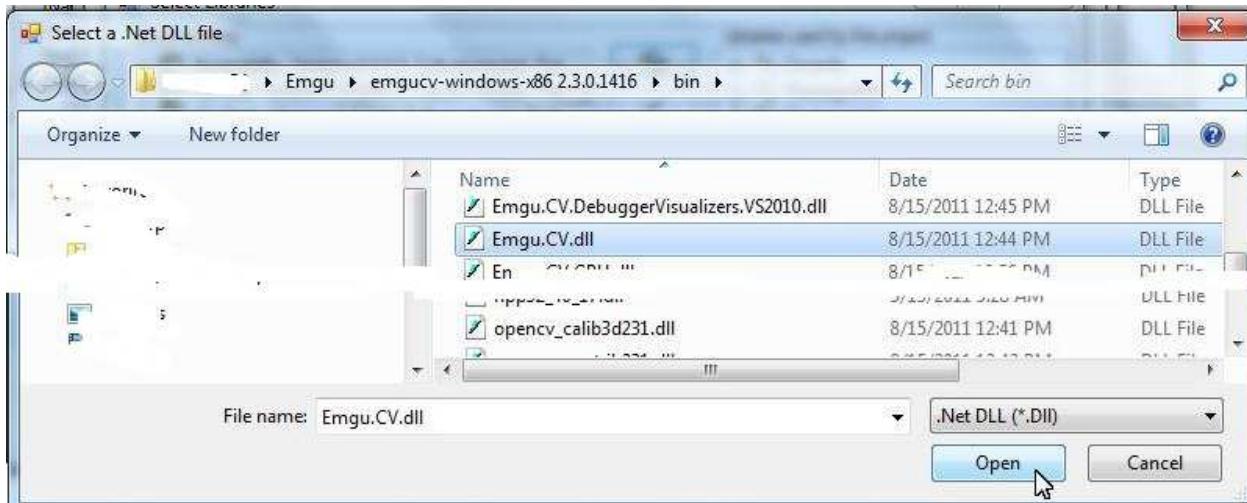


Click to load the external library:



Select Emgu.CV.dll from the bin folder under the Emgu installation folder:

Use Generic Types and Native Libraries



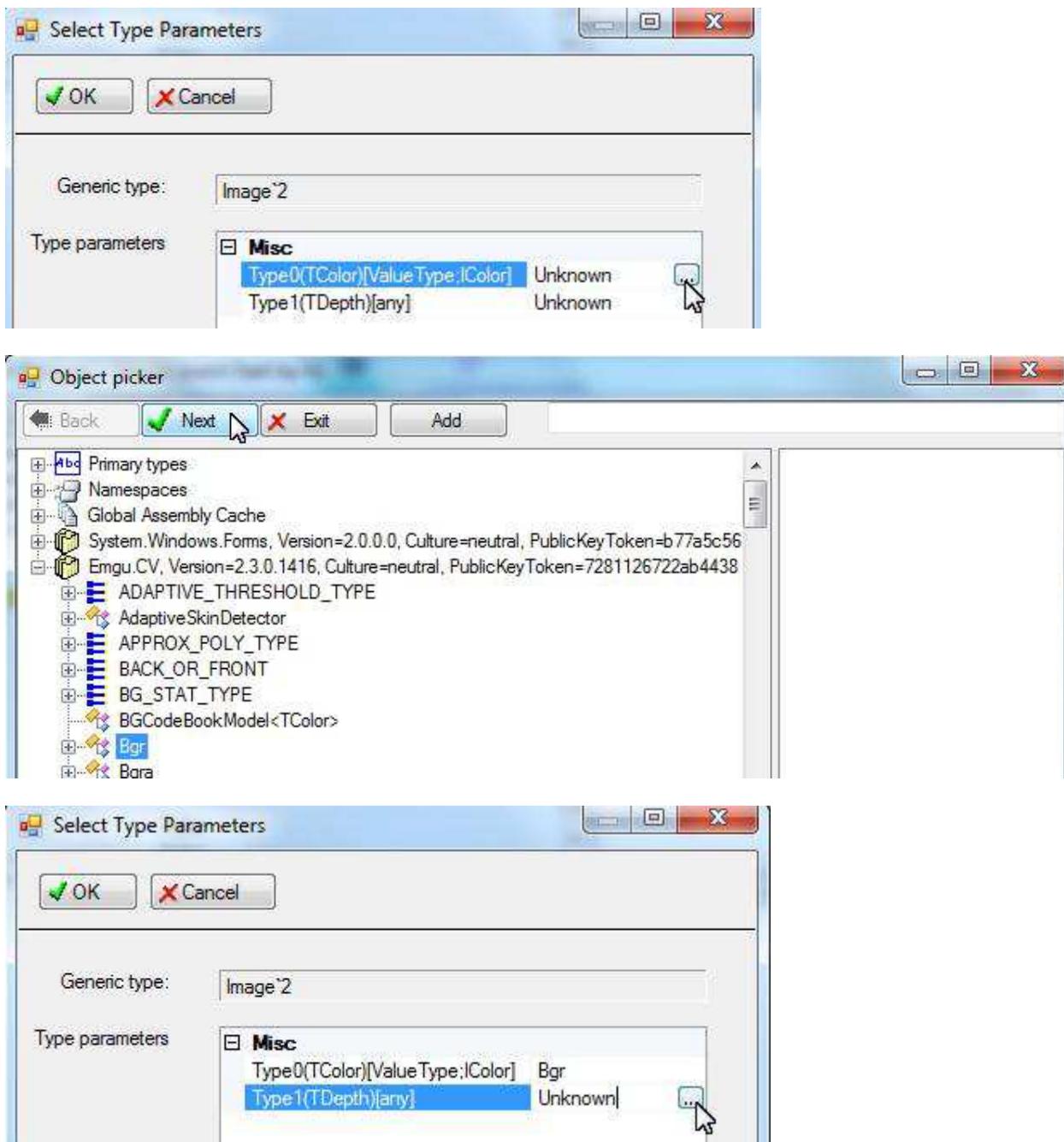
Select the Image class from the Emgu library:

The screenshots illustrate the process of selecting the `Image<TColor, TDepth>` class from the Emgu library.

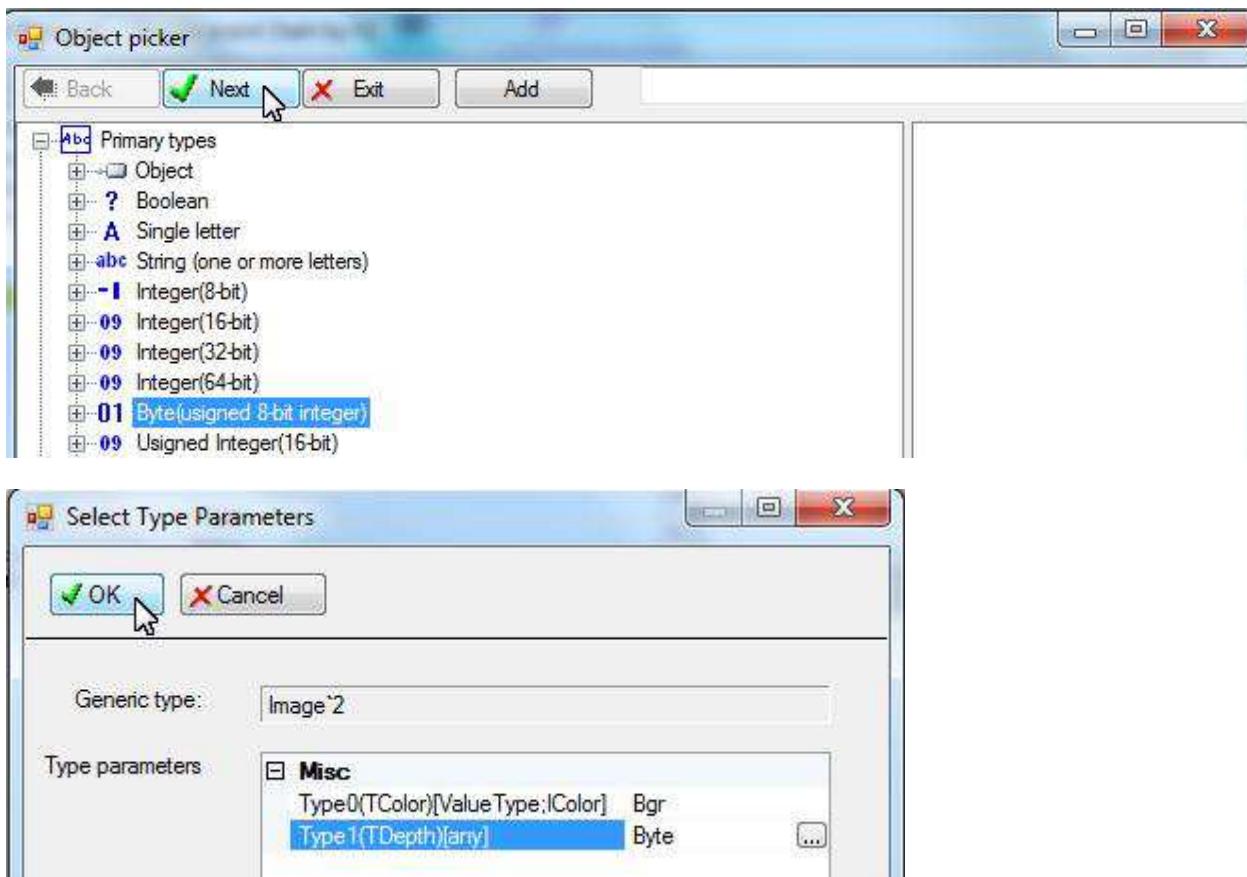
- Top Screenshot:** Shows the "Select Libraries" dialog with the "Global libraries" section expanded to show the `Emgu.CV` library. The `Image<TColor, TDepth>` class is visible under the library's namespace.
- Middle Screenshot:** Shows the "Select Libraries" dialog again, but with the "Global libraries" section expanded to show the `Image<TColor, TDepth>` class itself. A "Next" button is highlighted, indicating the next step in the selection process.
- Bottom Screenshot:** Shows the "Object picker" dialog. The left pane lists various .NET types and namespaces, including `Image<TColor, TDepth>`. The right pane is currently empty. The "Add" button is highlighted, indicating the final step to add the selected type to the project.

The `Image` class contains two generic types, `TColor` and `TDepth`. According to the original Emgu C# sample, `TColor` should be class `Bgr` from the Emgu library, and `TDepth` should be `byte`.

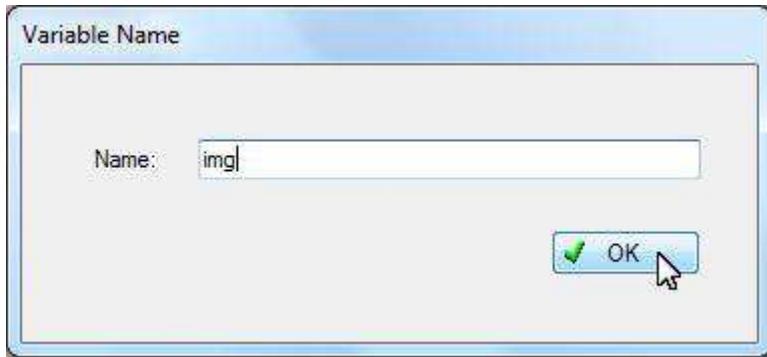
Use Generic Types and Native Libraries



Use Generic Types and Native Libraries

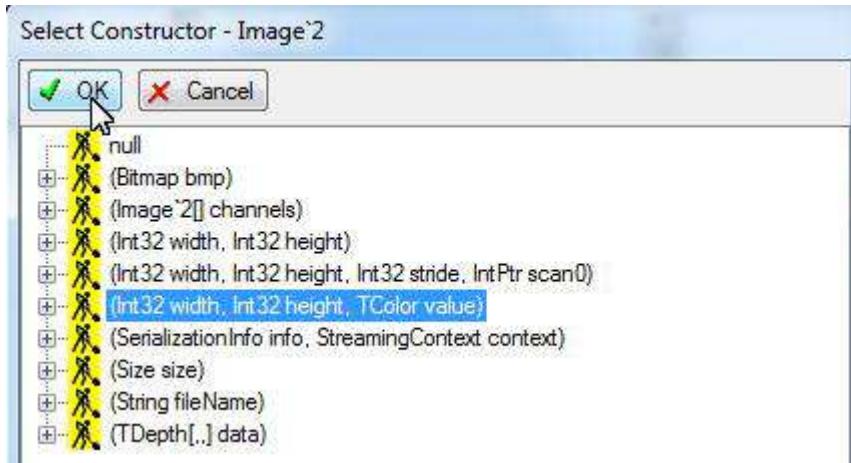


Give a name to the image variable:



Select a constructor for the variable. Choose the constructor using size and color:

Use Generic Types and Native Libraries



The image variable appears in the Variable Pane. An action appears in the Action Pane. The action constructs the variable. We need to specify the size to be (400,200) and color to blue for the action:

The screenshots show the configuration of a 'Create img' action to create an Image<2 variable.

Action Pane (Top Screenshot):

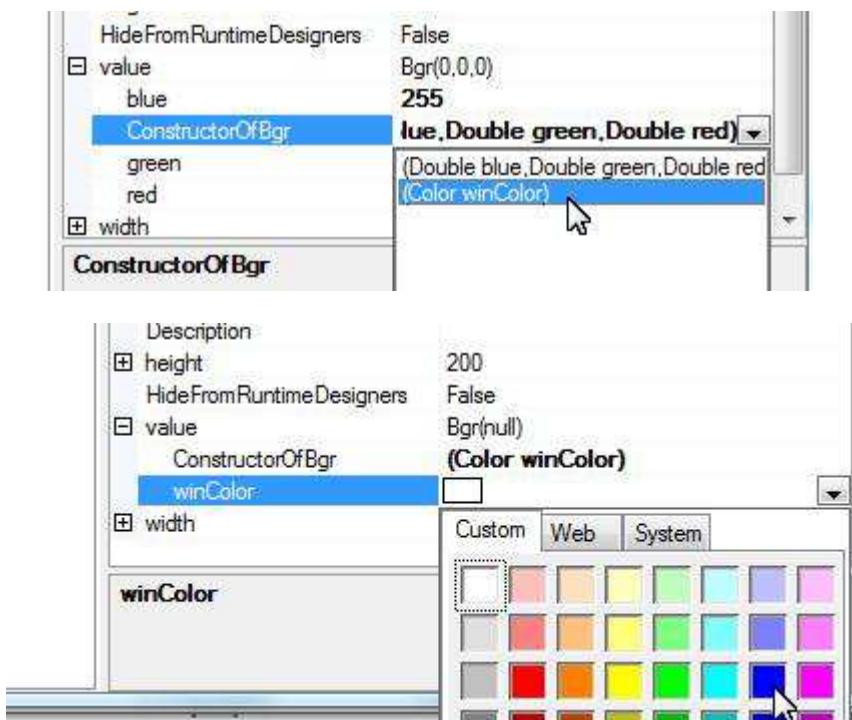
Property	Value
ActionName	Create img
(TColor)	Bgr
(TDepth)	Byte
ActionCondition	true
ActionMethod	img..ctor(Int32,Int32,TColor)Image<TColor,
BreakAfterExecute	False
BreakBeforeExecute	False
Description	
height	200
HideFromRuntimeDesigners	False
value	Bgr0
ConstructorOfBgr	Double blue,Double green,Double red
width	(Color winColor)

Variable Pane (Bottom Screenshot):

Property	Value
ActionName	Create img
(TColor)	Bgr
(TDepth)	Byte
ActionCondition	true
ActionMethod	img..ctor(Int32,Int32,TColor)Image<TColor,
BreakAfterExecute	False
BreakBeforeExecute	False
Description	
height	200
HideFromRuntimeDesigners	False
value	Bgr(0,0,0)
blue	255
ConstructorOfBgr	(Double blue,Double green,Double red)
green	0
red	0
width	400

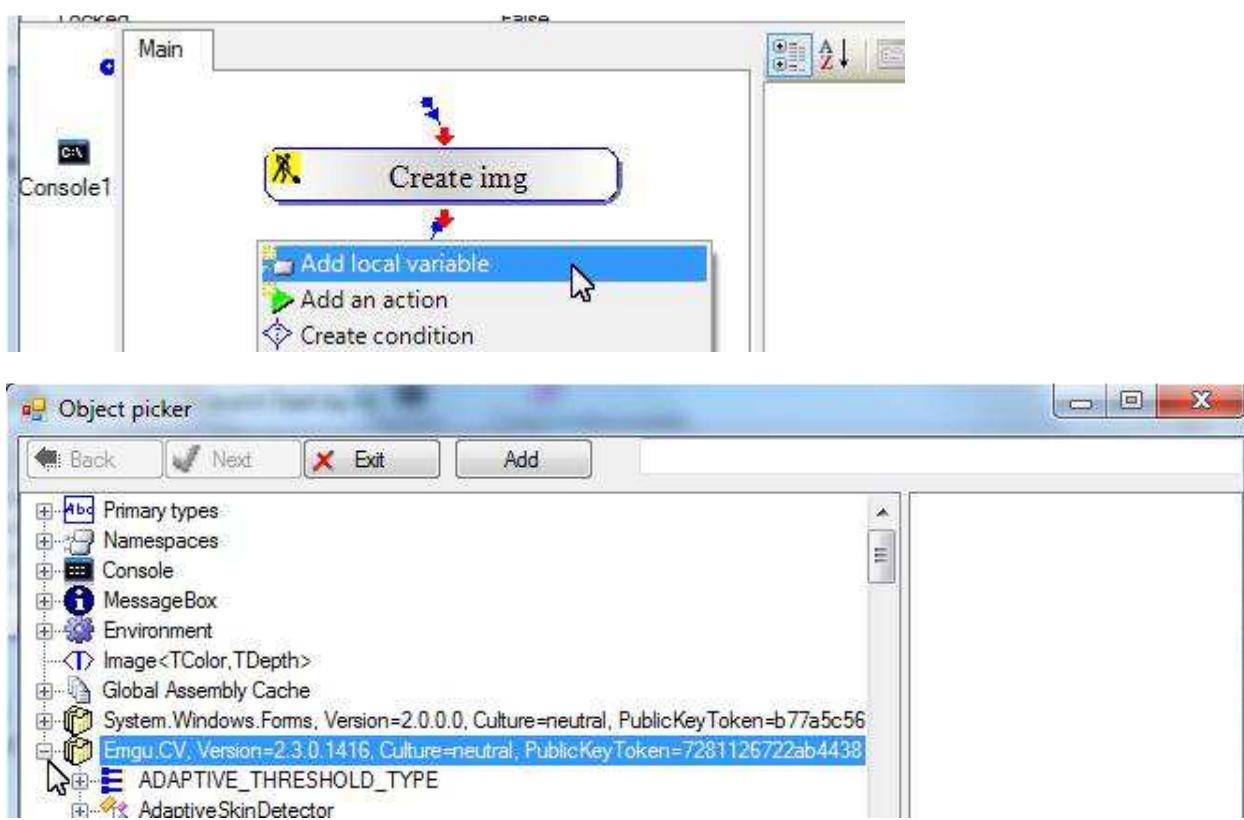
You may also specify the color visually:

Use Generic Types and Native Libraries

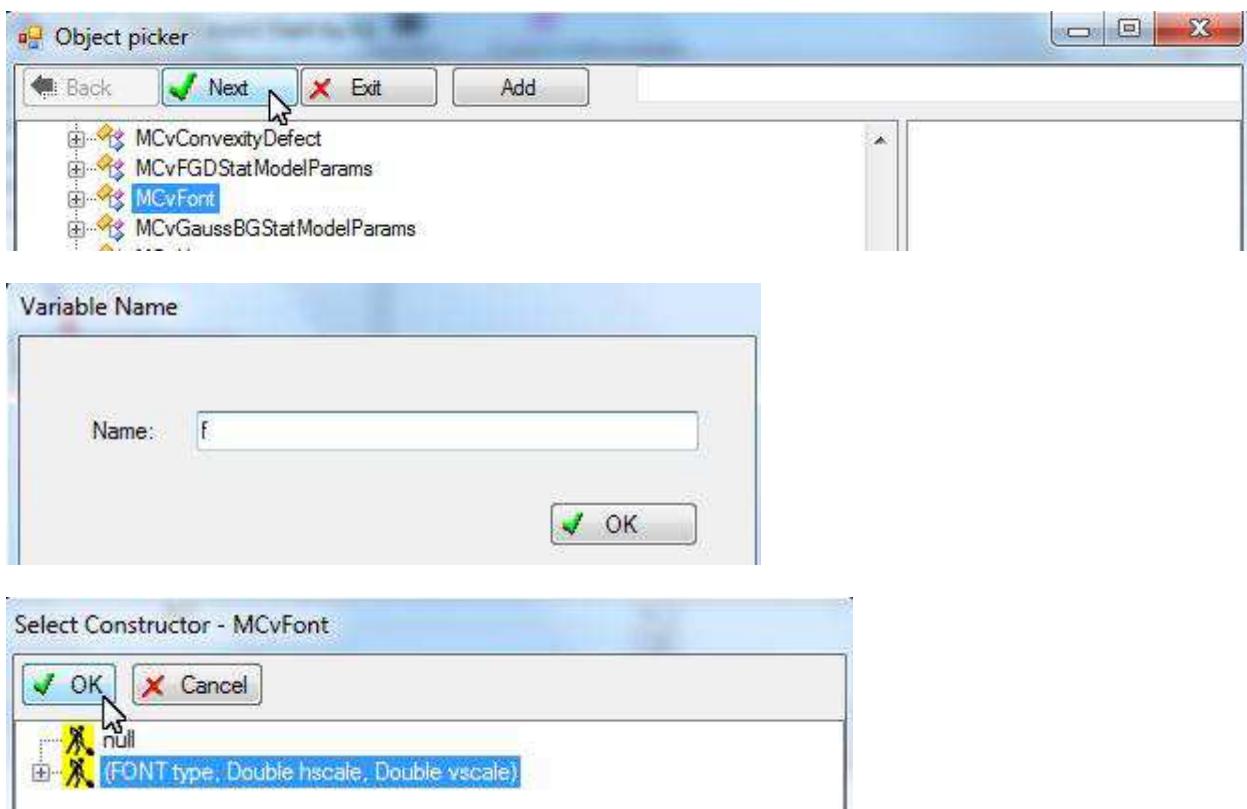


Create font

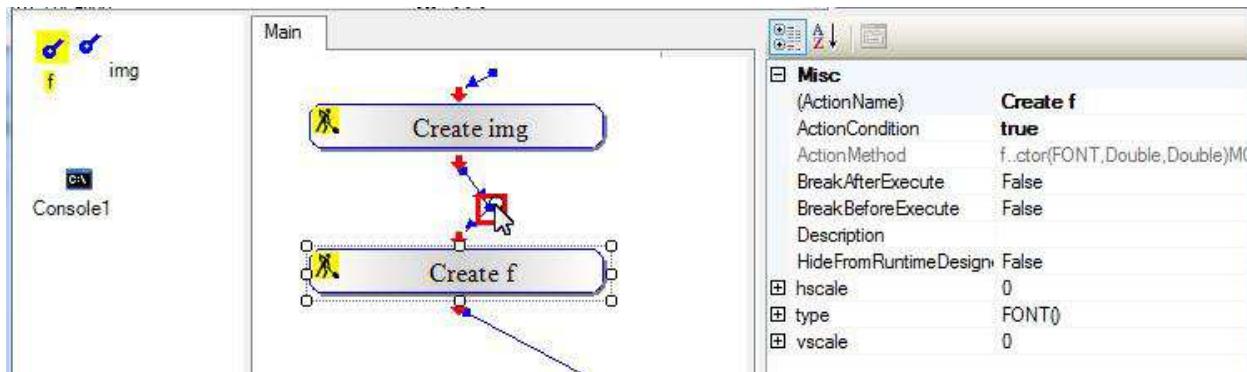
To draw text, we need to create a font.



Use Generic Types and Native Libraries

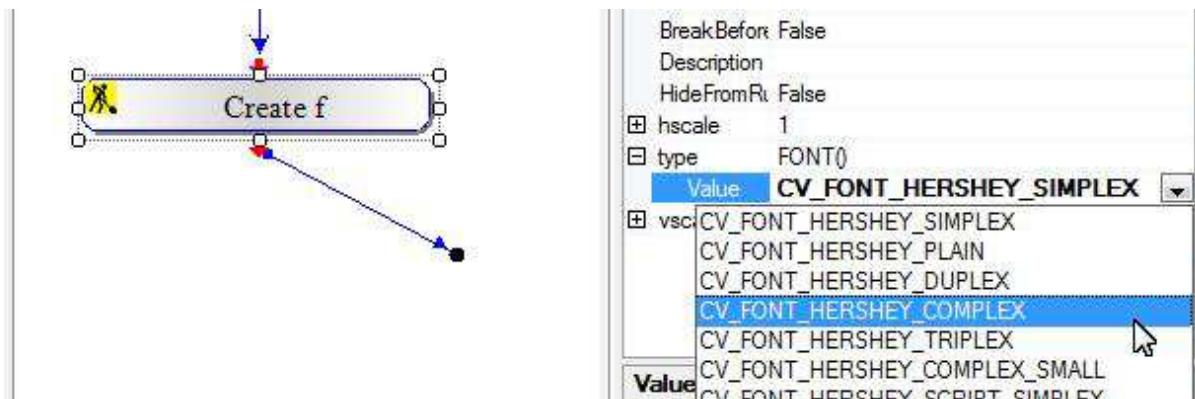


A font variable appears. A new action appears for creating the font. Link the action with the existing one:



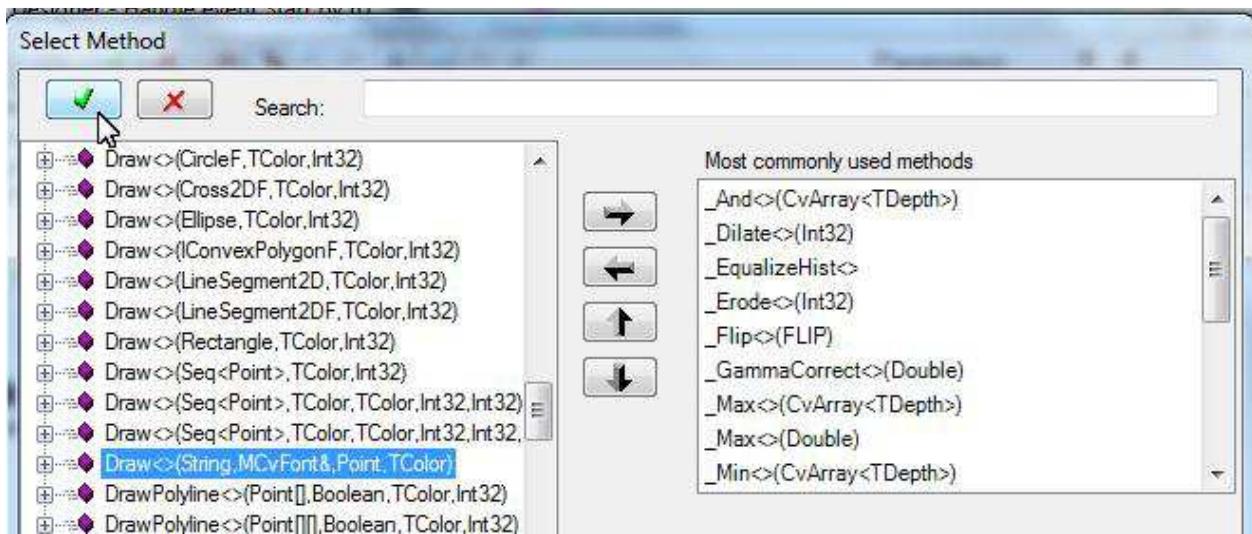
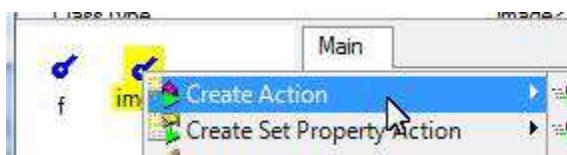
Specify the parameter values for the action:

Use Generic Types and Native Libraries



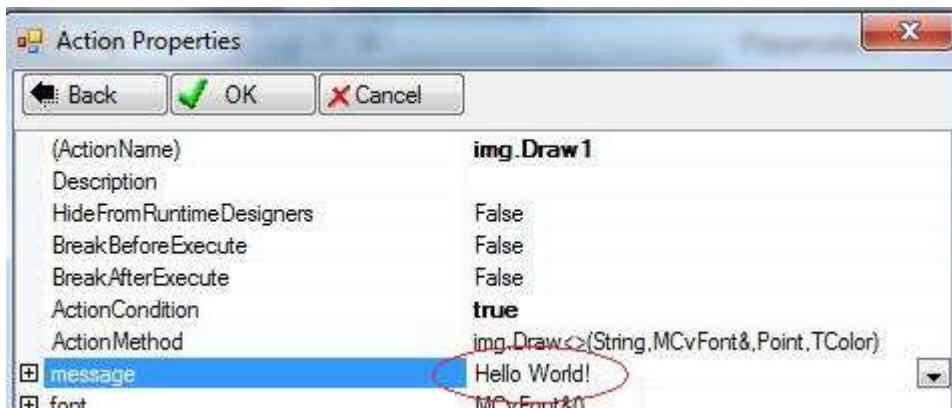
Draw text

The image object has many Draw methods. We may use one of it to create an action to draw "Hello World!" on the image.

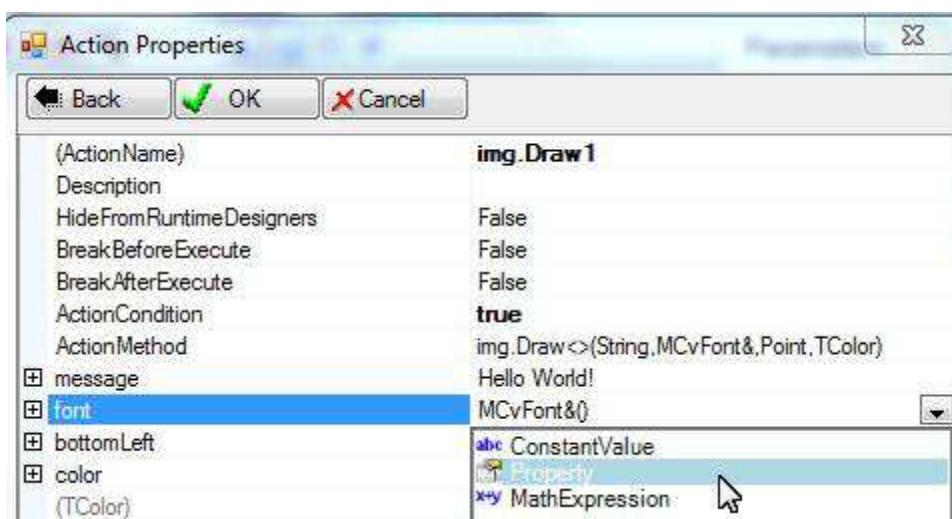


Specify the text to draw:

Use Generic Types and Native Libraries

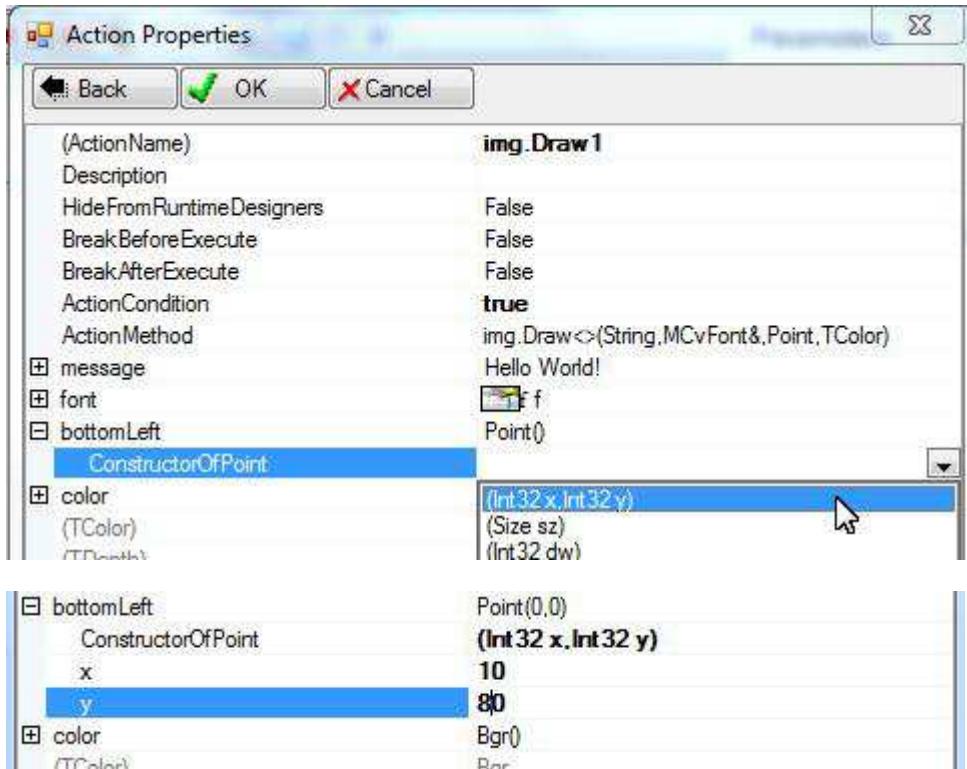


Use the font we have created:



Specify the drawing location:

Use Generic Types and Native Libraries

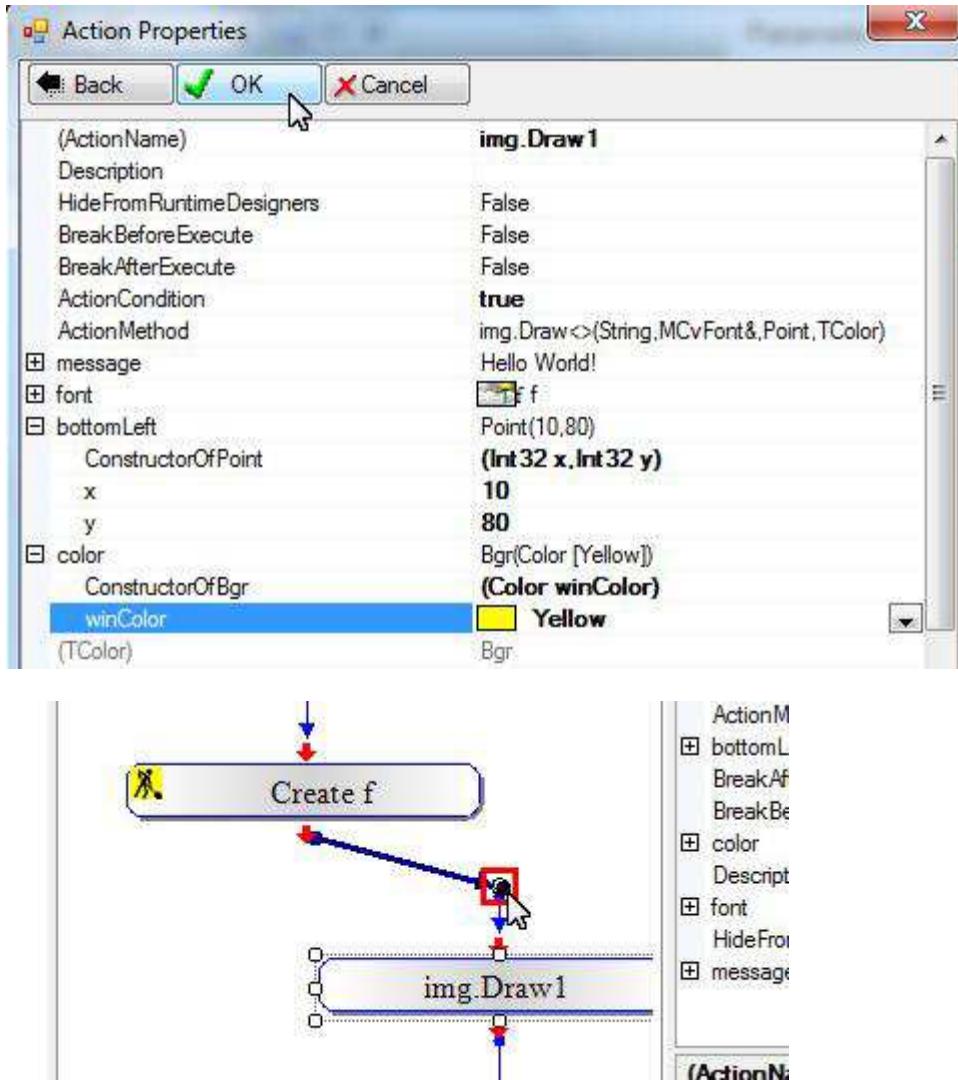


Specify the drawing color:



Click OK to finish creating this action:

Use Generic Types and Native Libraries



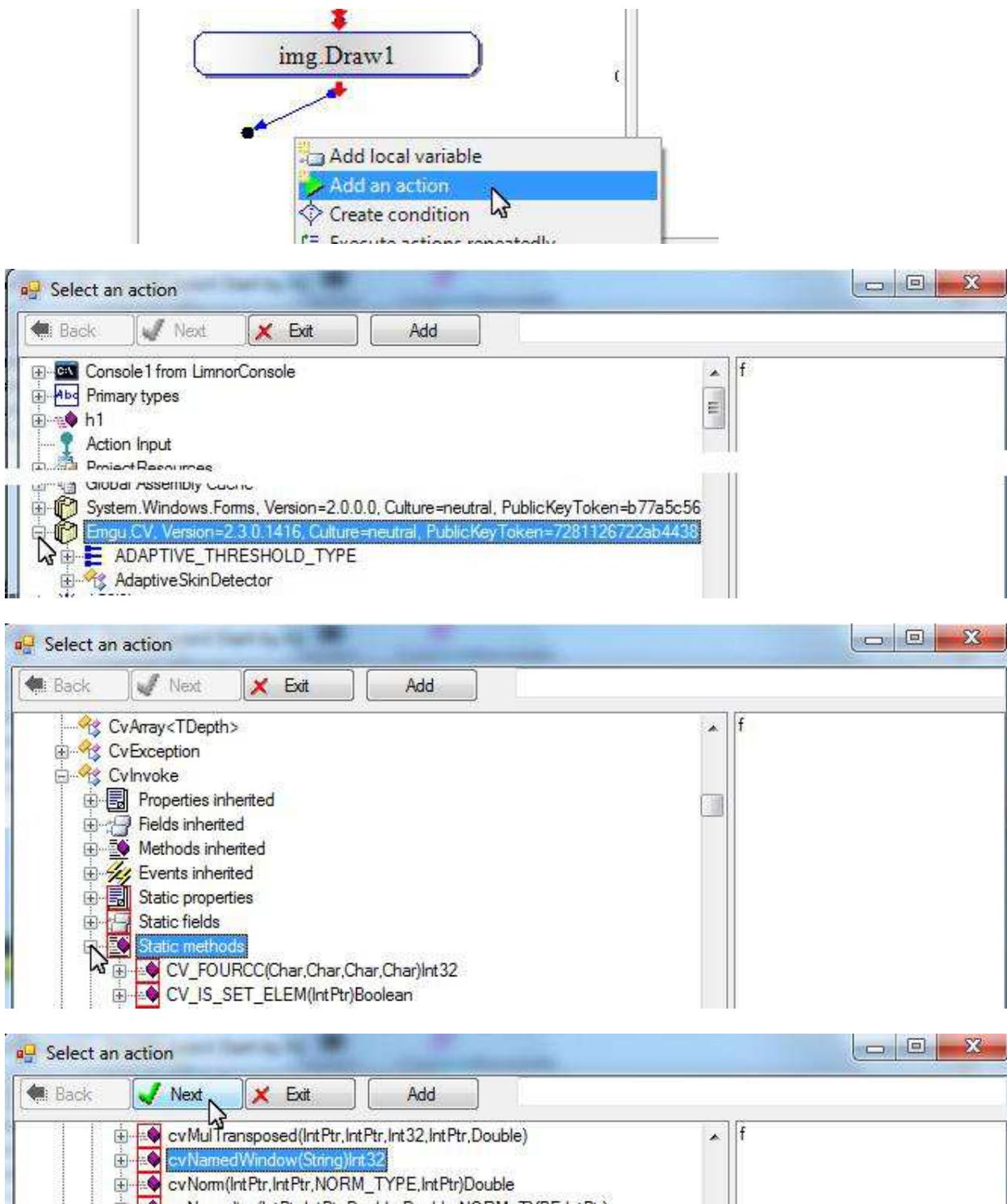
Create a window to display image

Emgu uses a class, CvInvoke, to provide many static methods. This sample uses following methods:

cvNamedWindow -- create a window
cvShowImage -- show an image on a window
cvWaitKey -- Wait for the key pressing event
cvDestroyWindow -- Destory a window

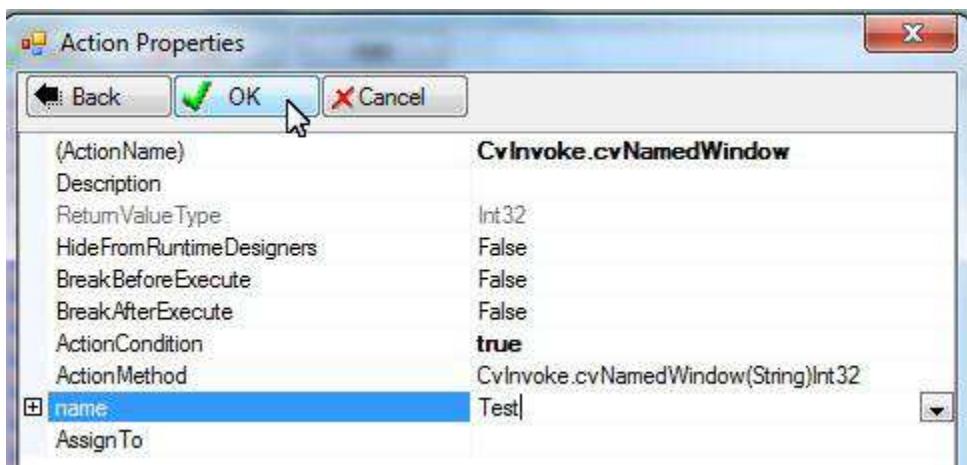
Let's create 4 actions using the above methods.

Use Generic Types and Native Libraries

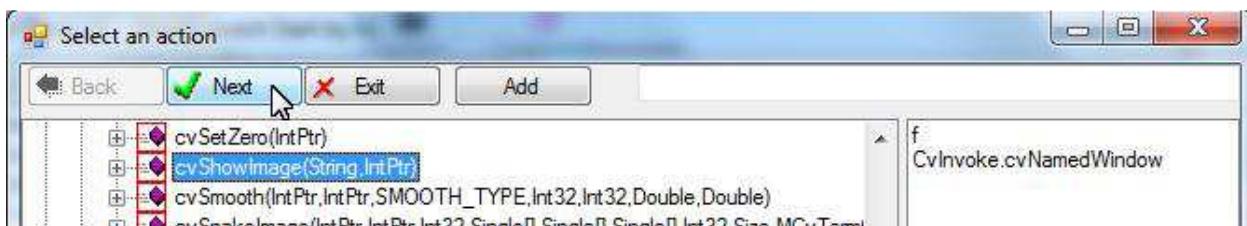
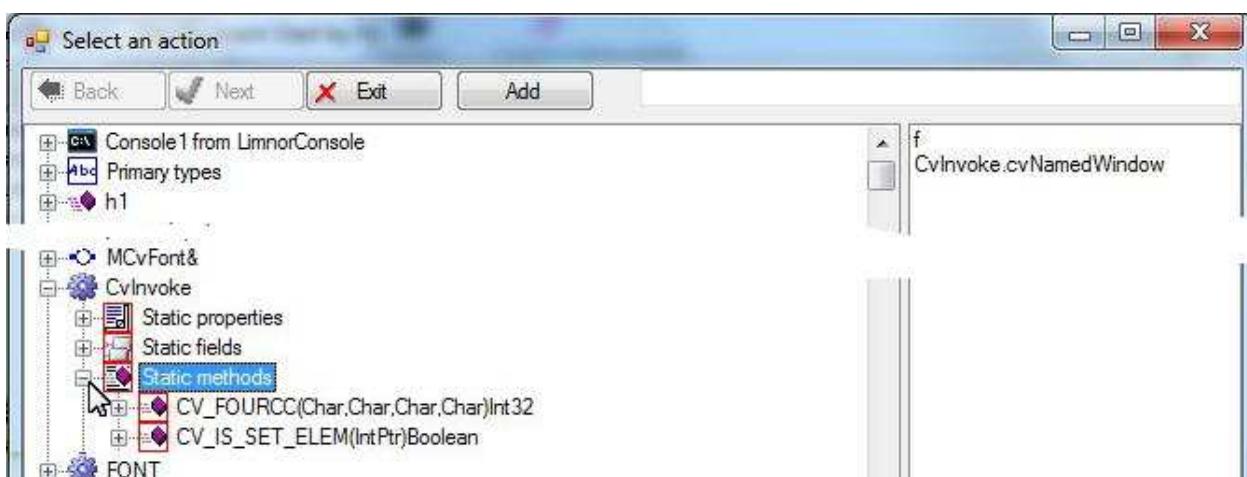
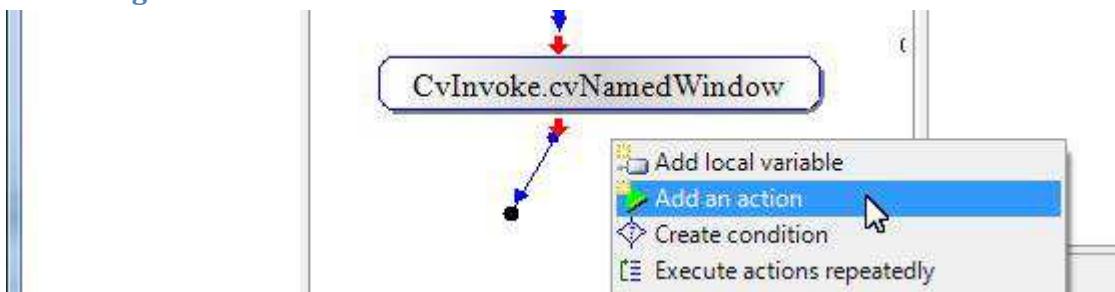


Give a name to the new window. Click OK to finish creating the action.

Use Generic Types and Native Libraries

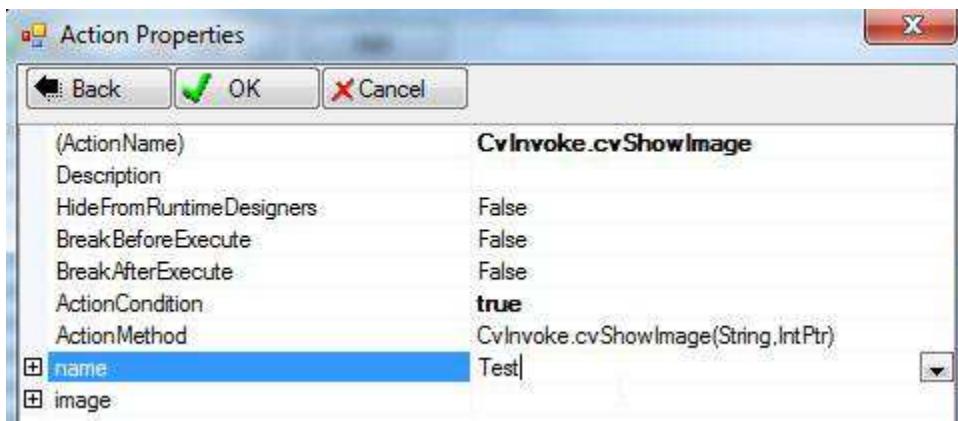


Show image

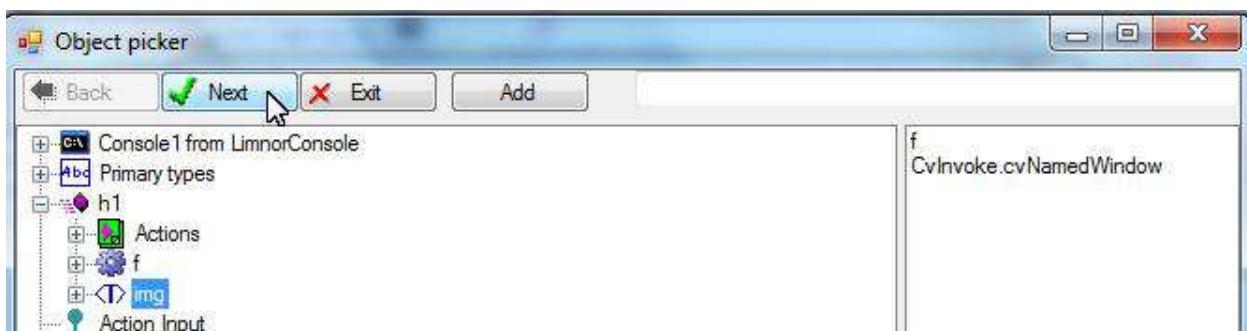
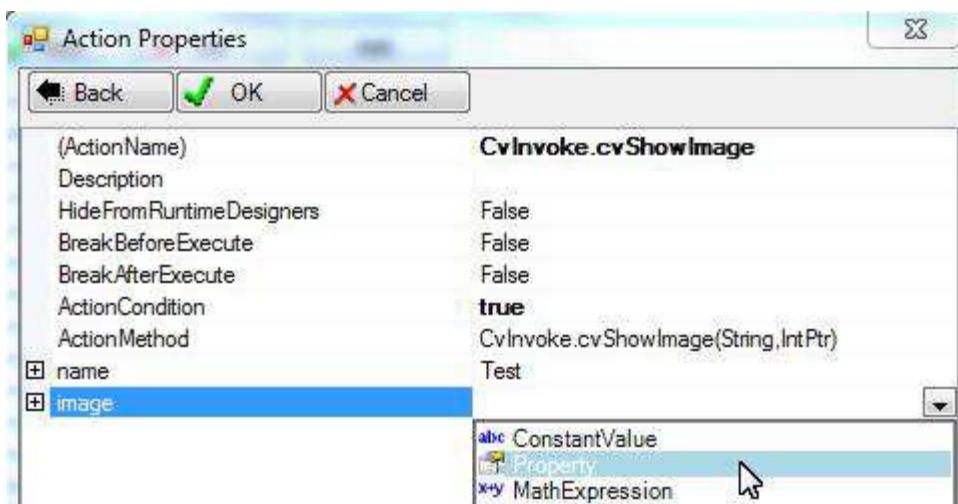


Specify the window name:

Use Generic Types and Native Libraries

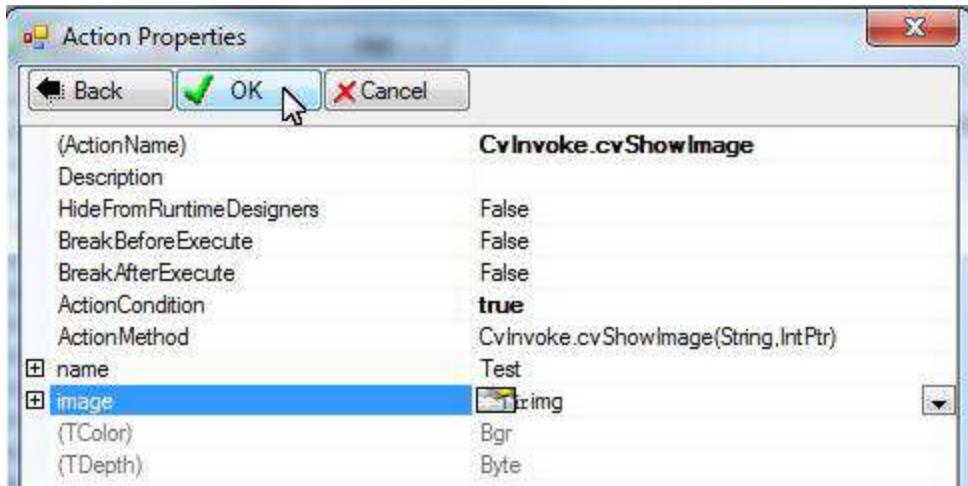


Select the image variable we have created:

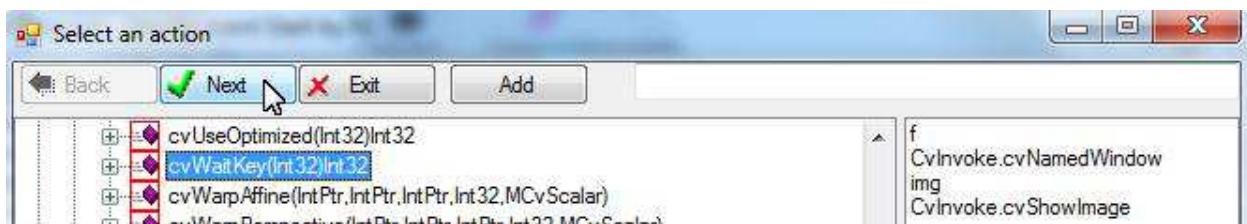
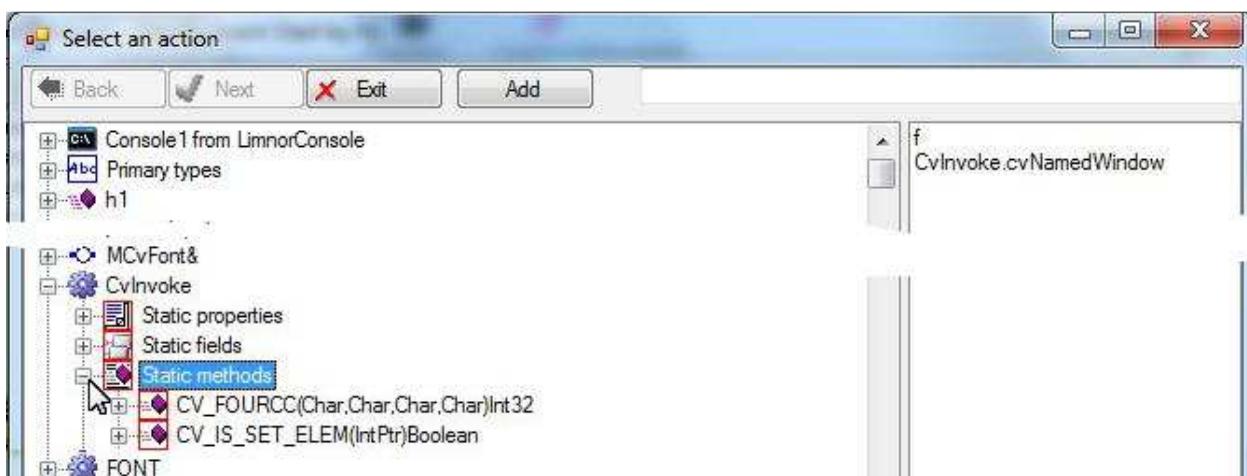
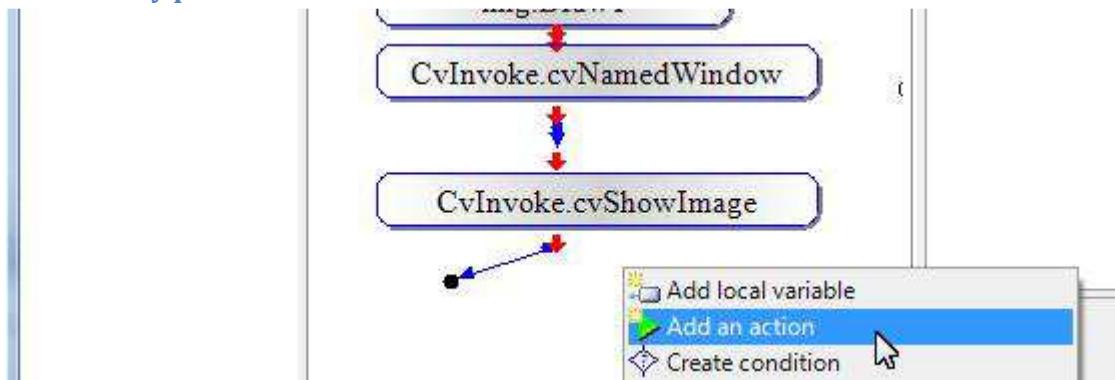


Click OK to finish creating the action:

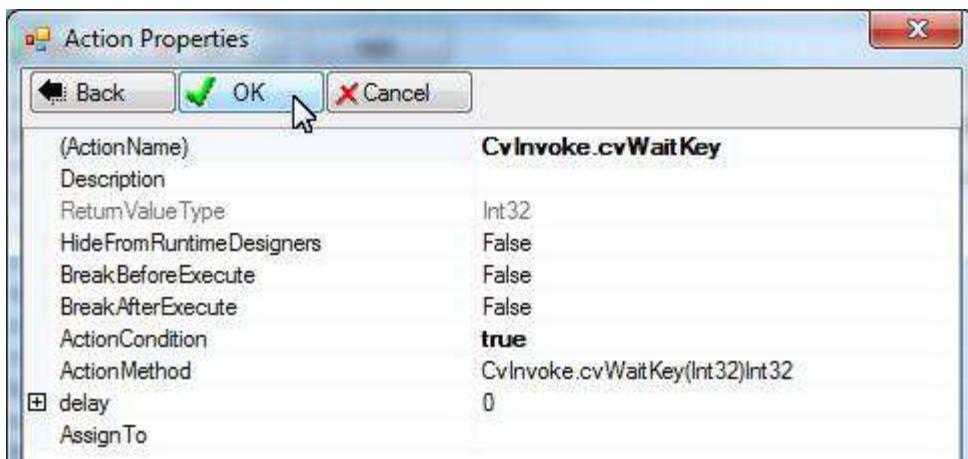
Use Generic Types and Native Libraries



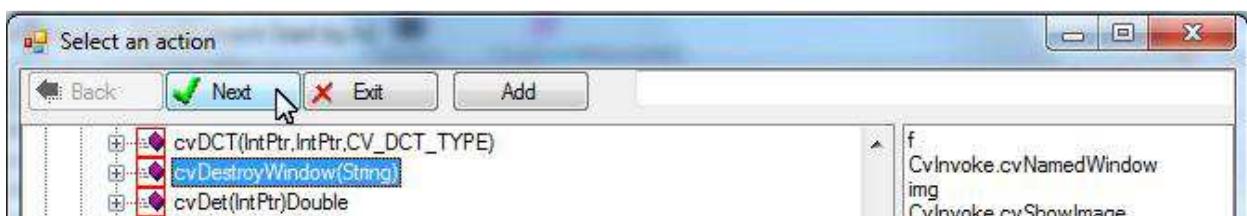
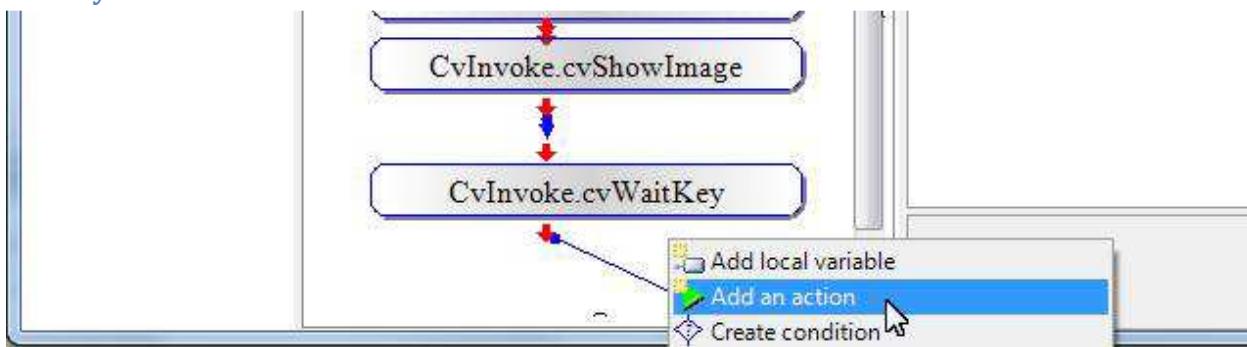
Wait for key press



Use Generic Types and Native Libraries

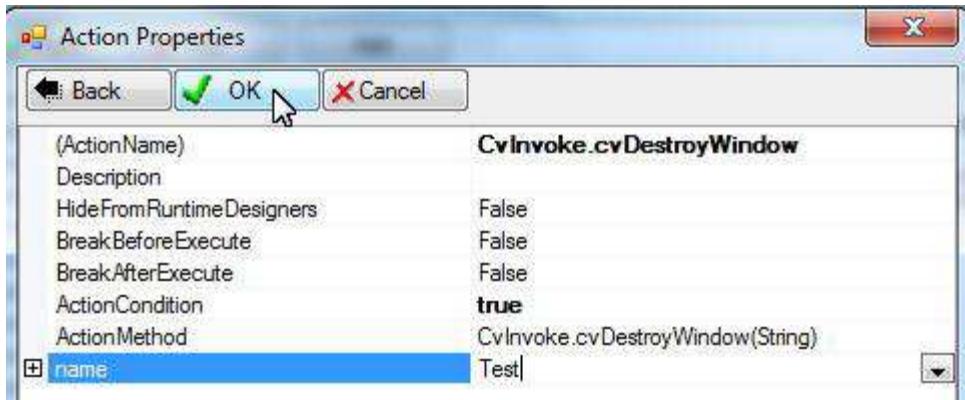


Destroy window



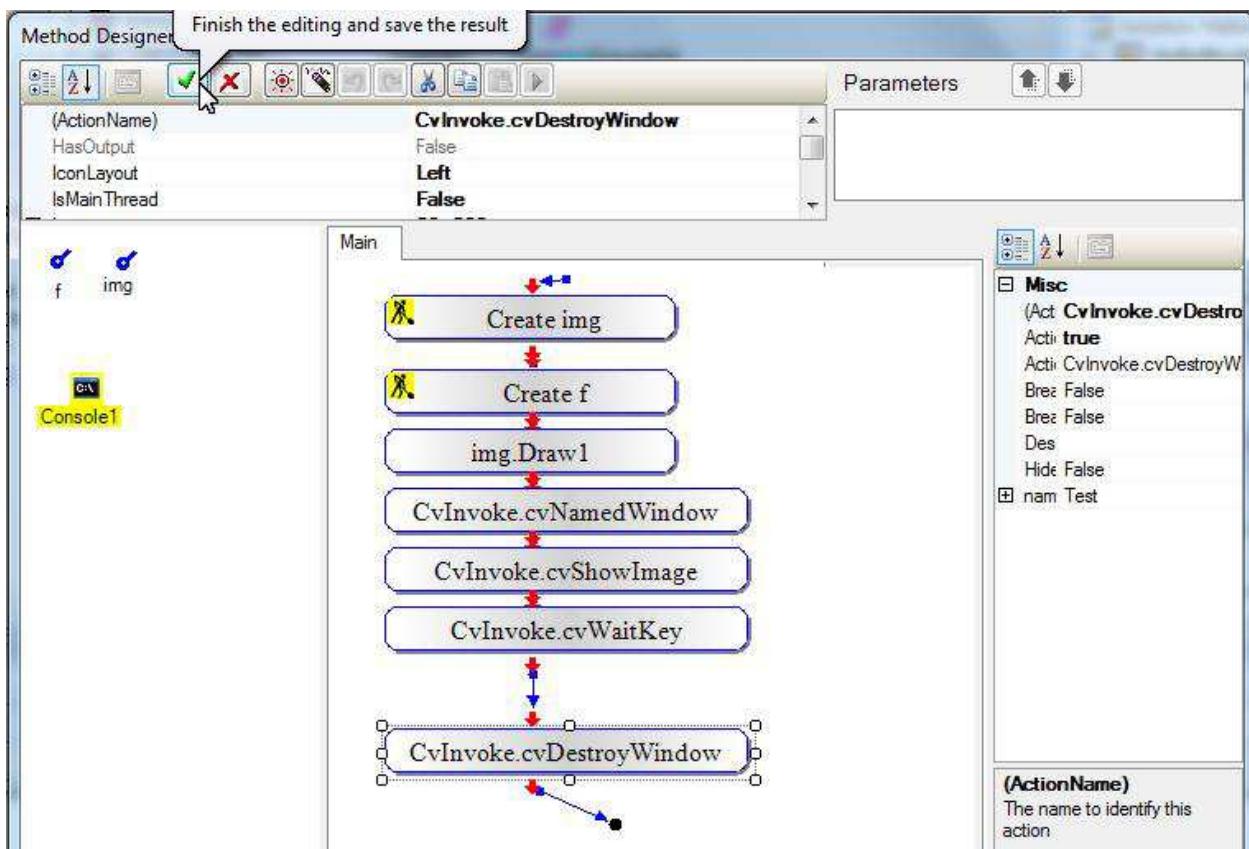
Specify the name for the window to be destroyed. Click OK.

Use Generic Types and Native Libraries



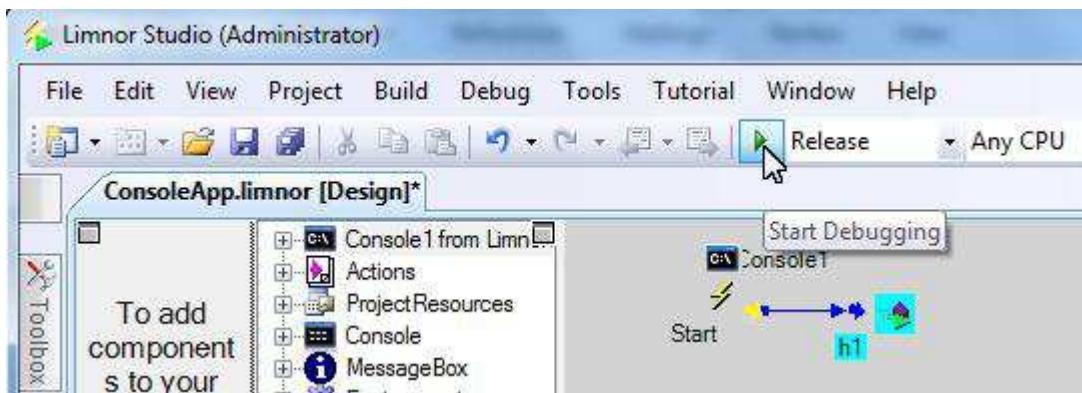
Test

We are done creating this event handler method.



We may test the project now.

Use Generic Types and Native Libraries



A window appears showing “Hello World!”

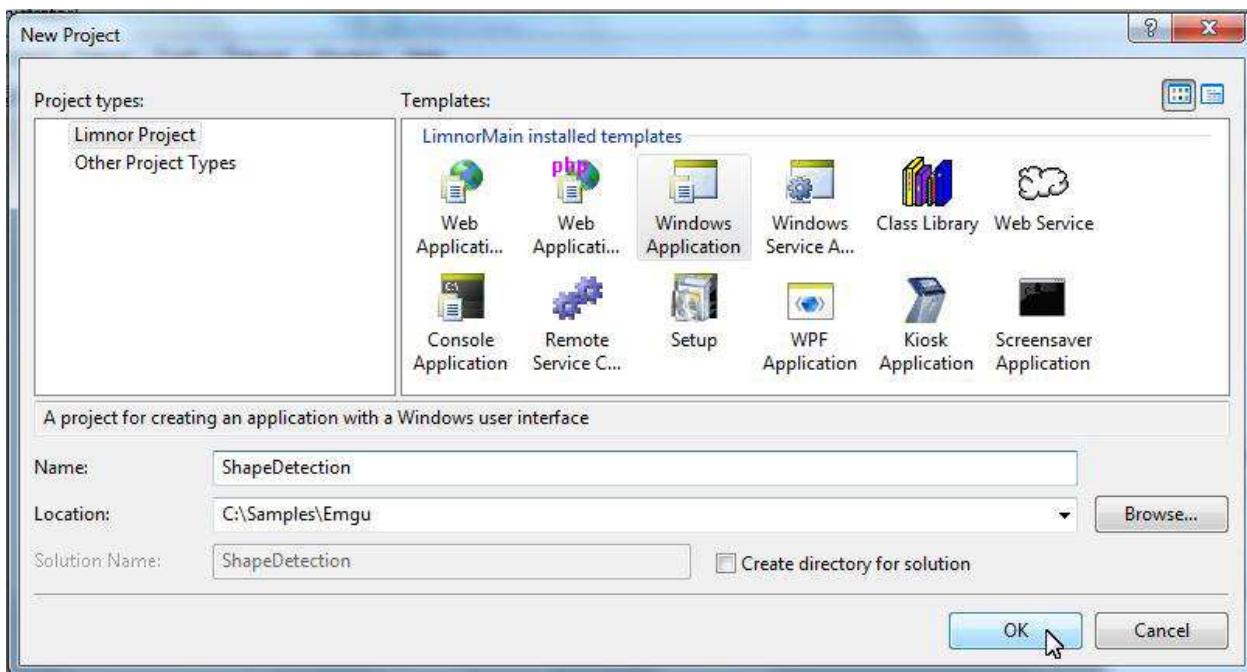


Sample 2 – Shape Detection

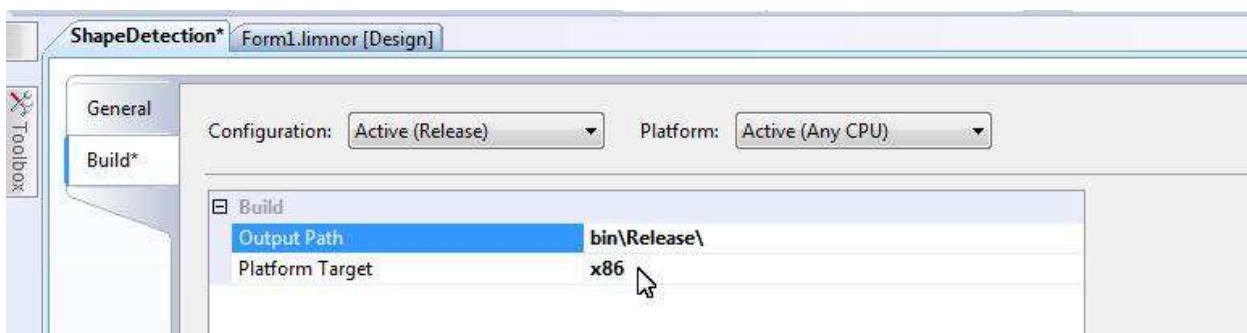
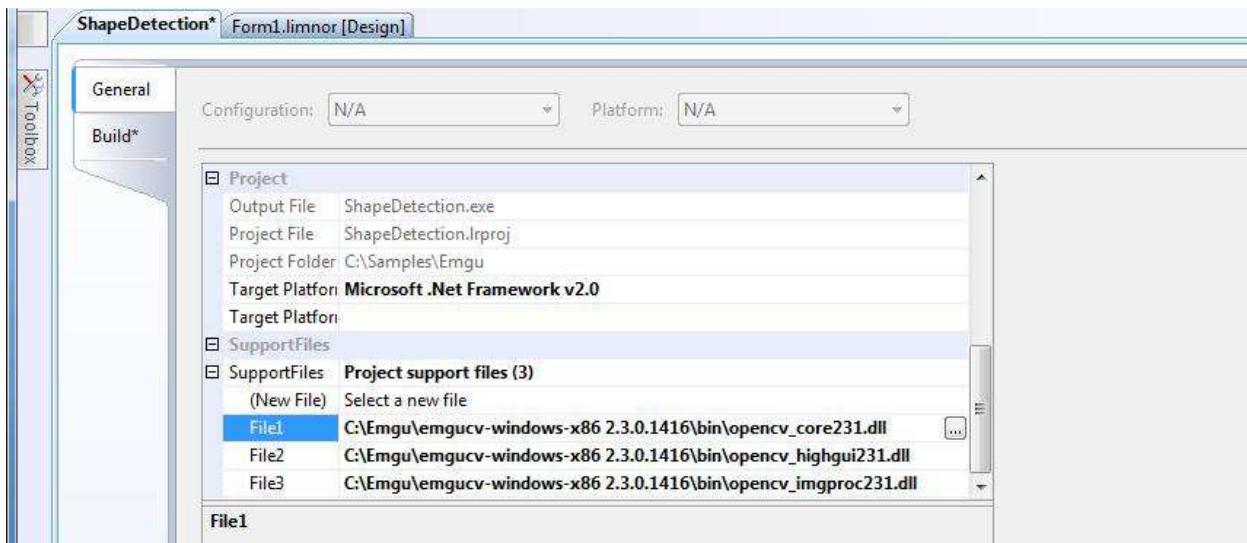
This sample is based on an Emgu sample found in folder “Emgu.CV.Example\ShapeDetection” under the Emgu installation folder.

Create a Windows Form project for this sample:

Use Generic Types and Native Libraries

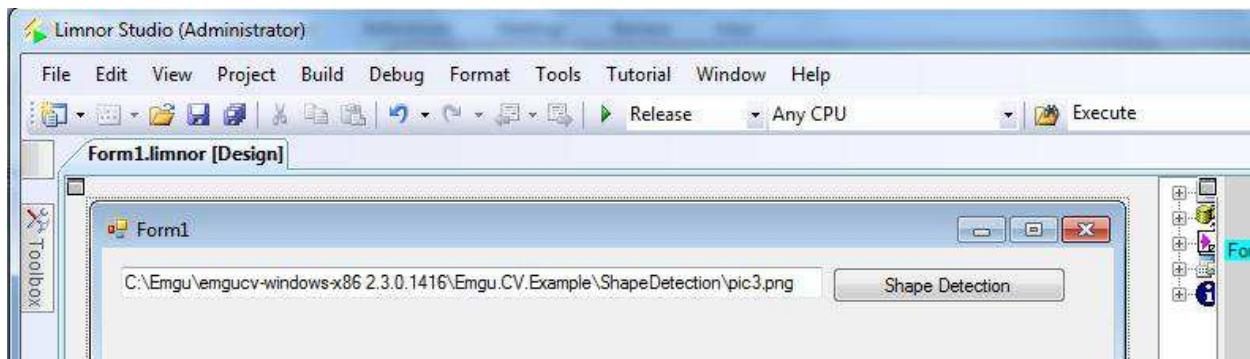


Set the target platform and add the support files to the project, see the last sample for details. Do not forget this step!



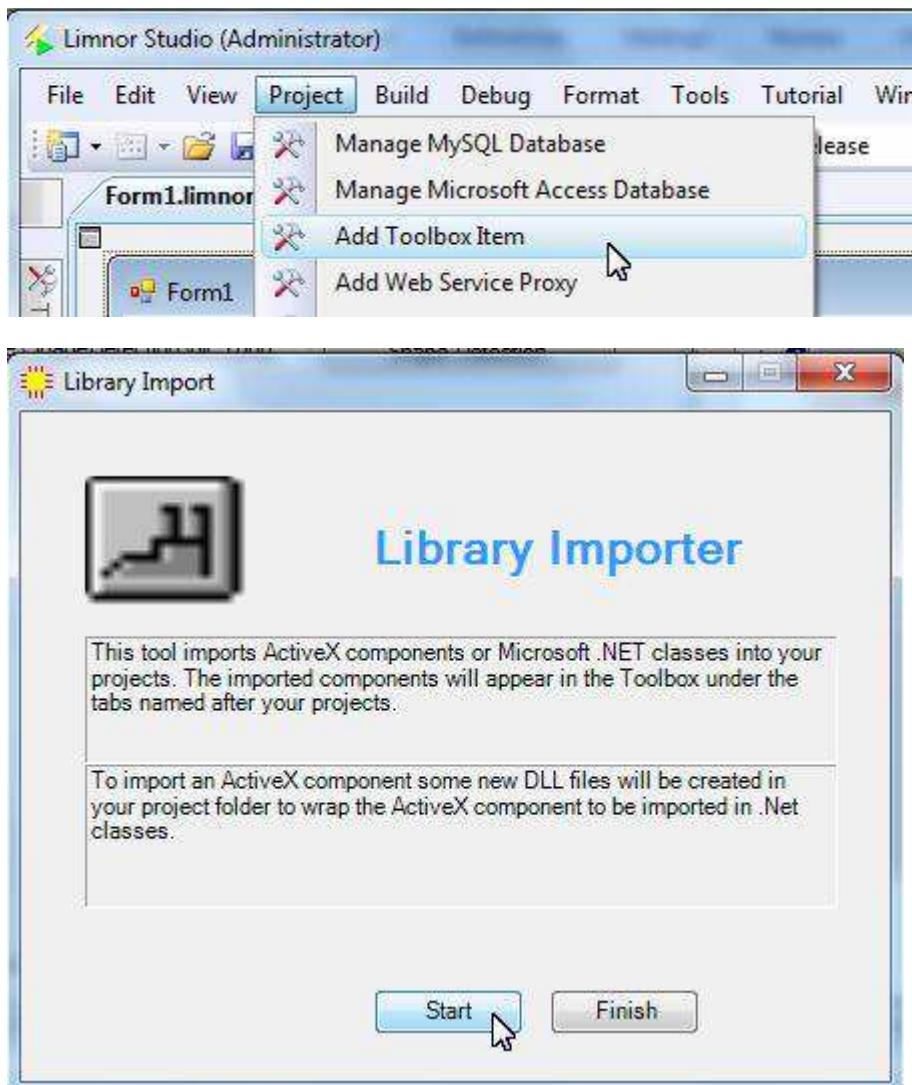
Use Generic Types and Native Libraries

Use a text box for the image file name. Use a button to start shape detection.



Use Image Box

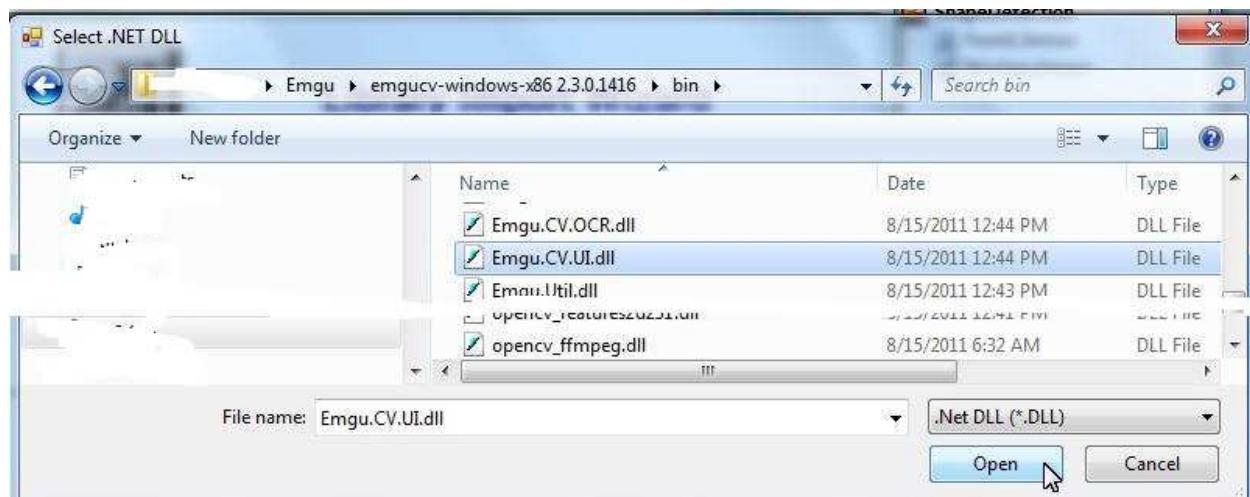
This sample uses Image Box to display images. Add Image Box to the toolbox so that we may add it to the form.



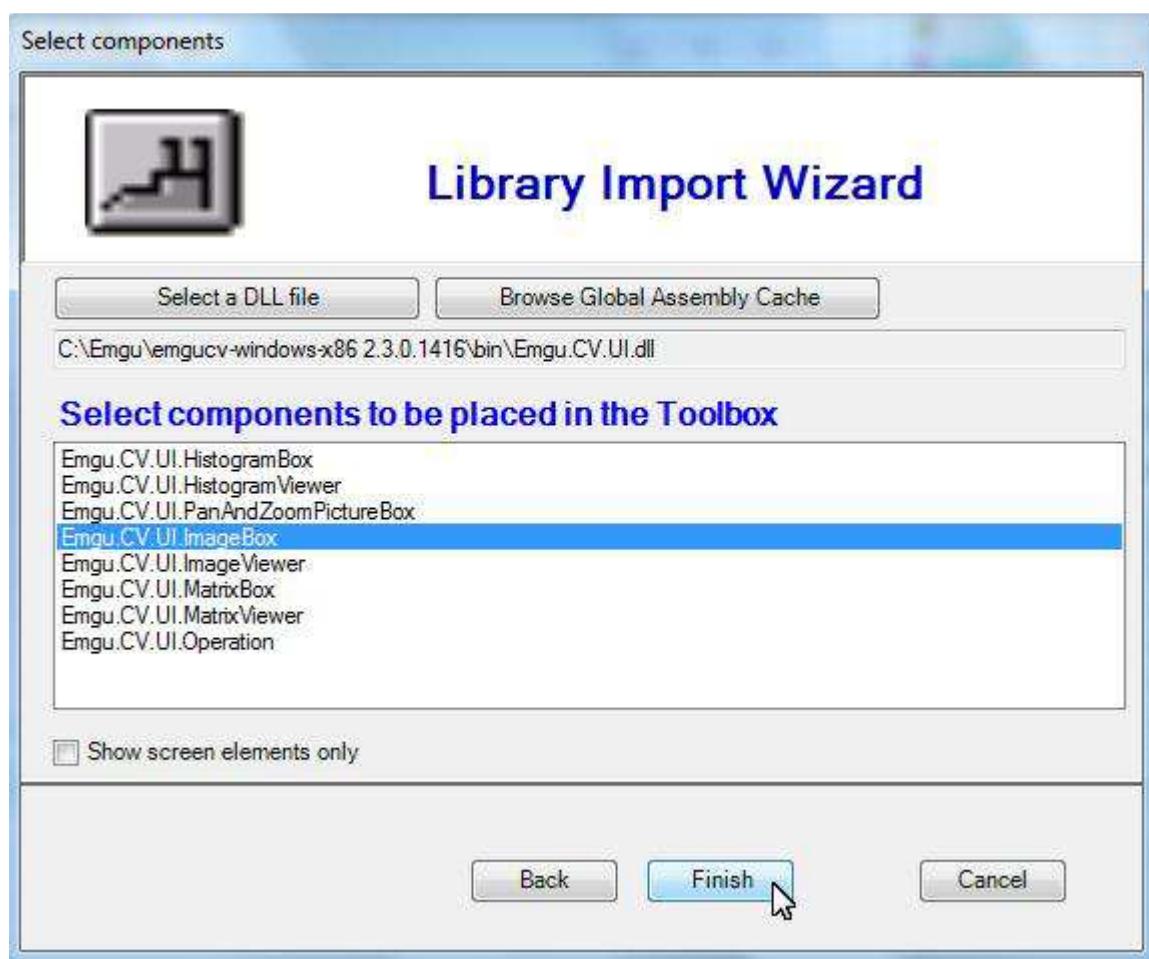


The Image Box is in Emgu.CV.UI.DLL

Use Generic Types and Native Libraries

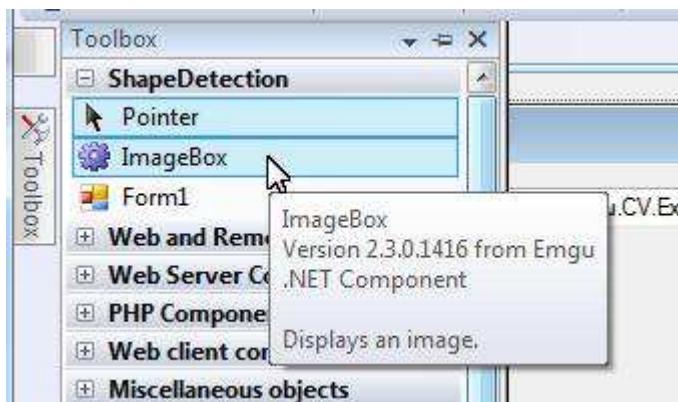


Select the Image Box, click Finish.

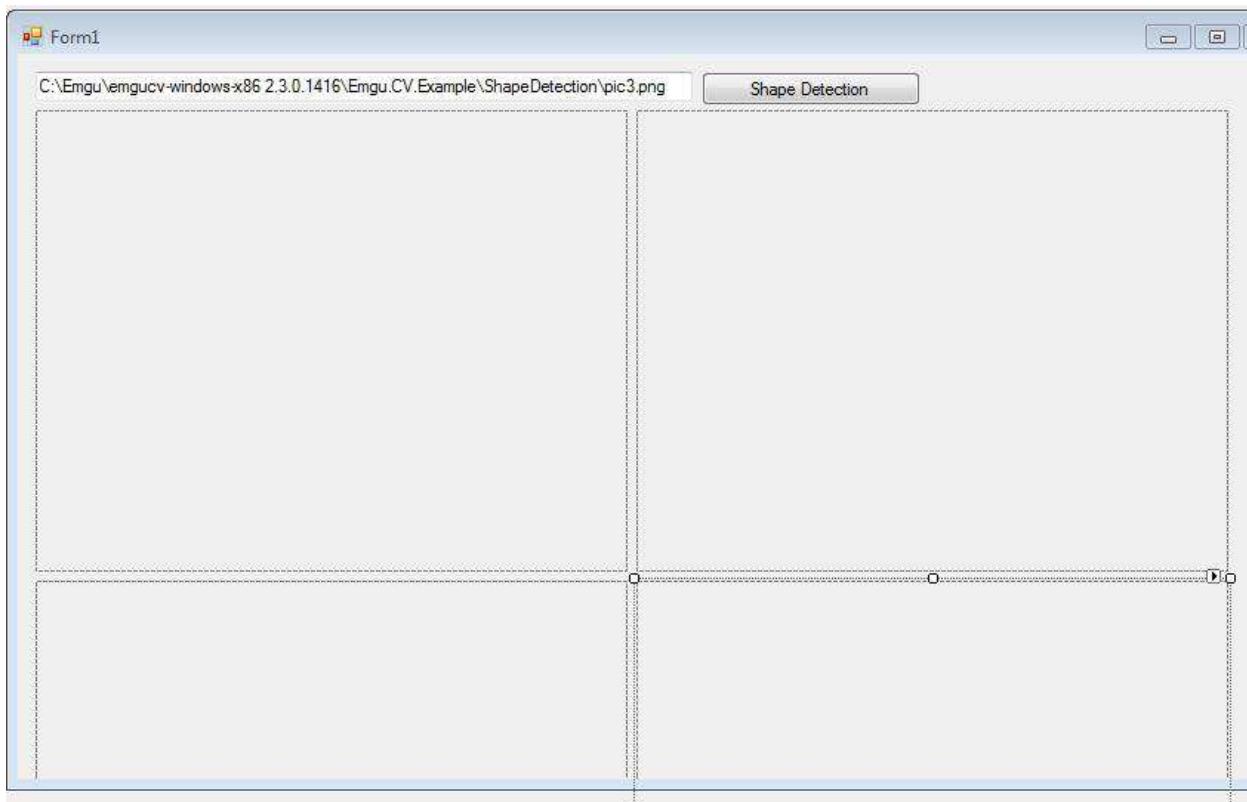


The Image Box appears in the Toolbox:

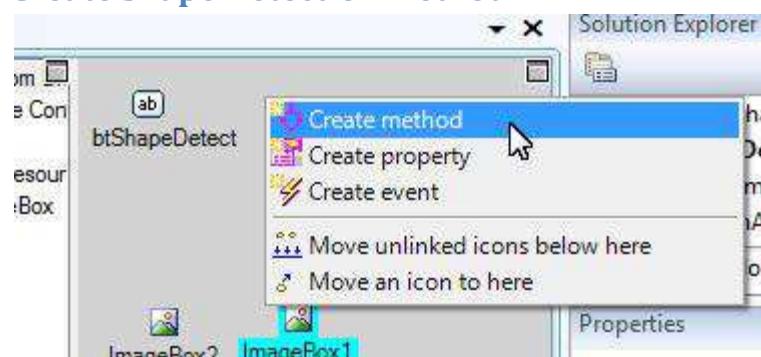
Use Generic Types and Native Libraries



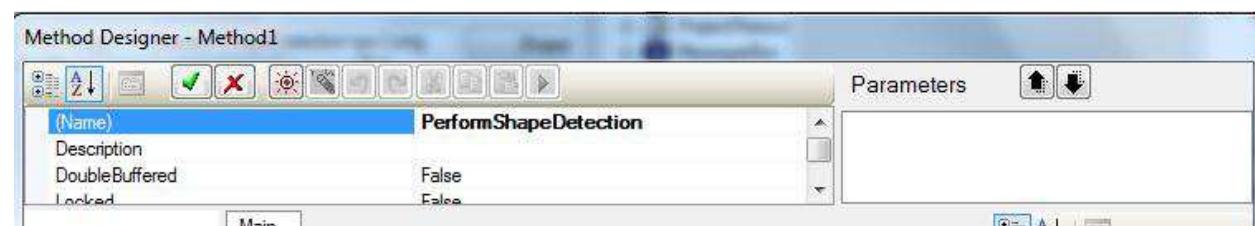
Add 4 image boxes to the form. One is for displaying the original image. The others are for displaying the detected shapes.



Create Shape Detection Method



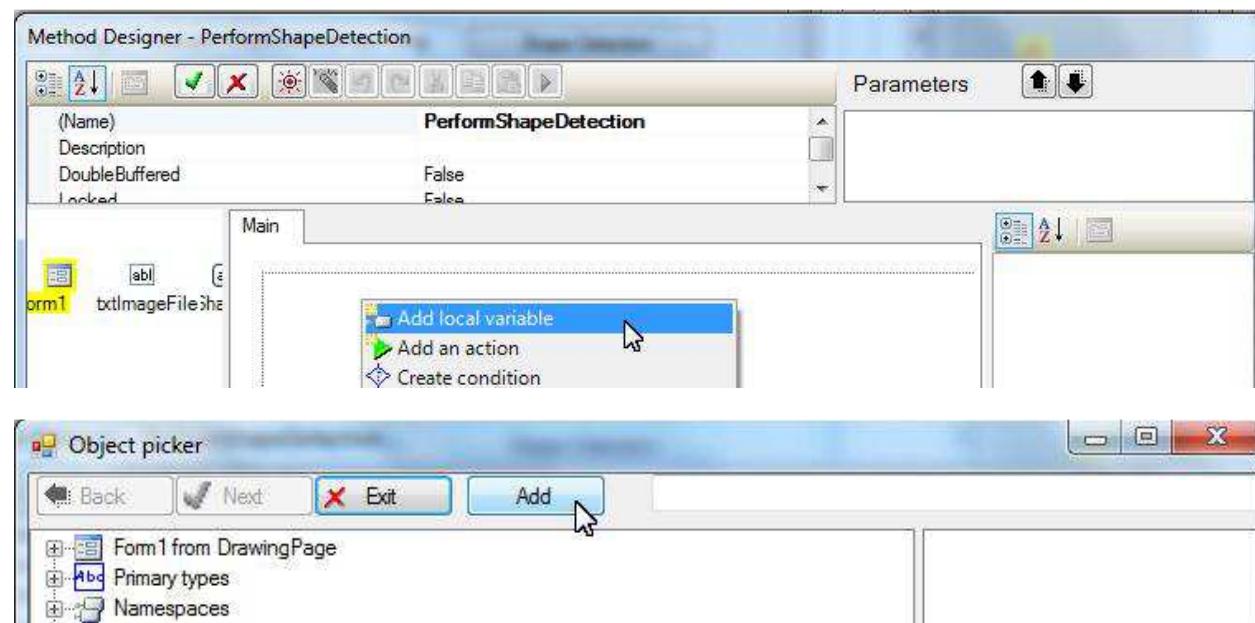
Name the method PerformShapeDetection:



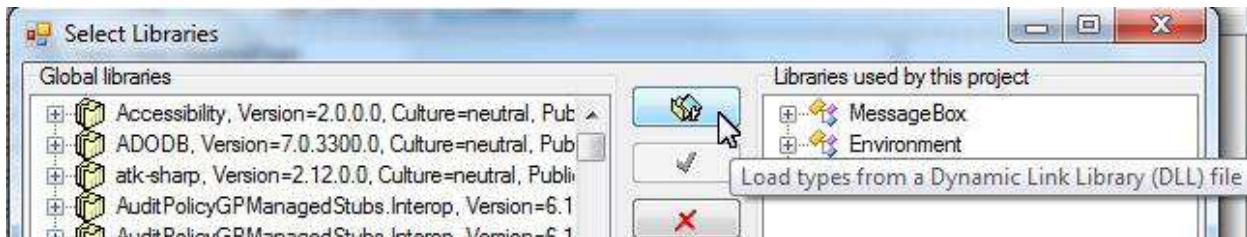
Load image from file

Create image object

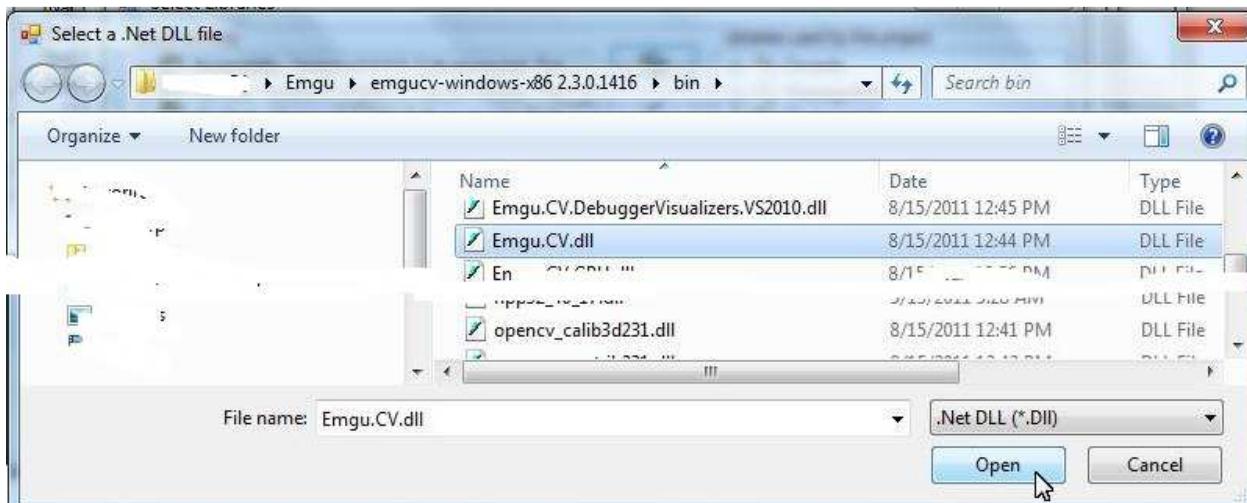
Create an image variable using the image file specified in the text box. It is a similar process as we did in the last sample, but using a different constructor.



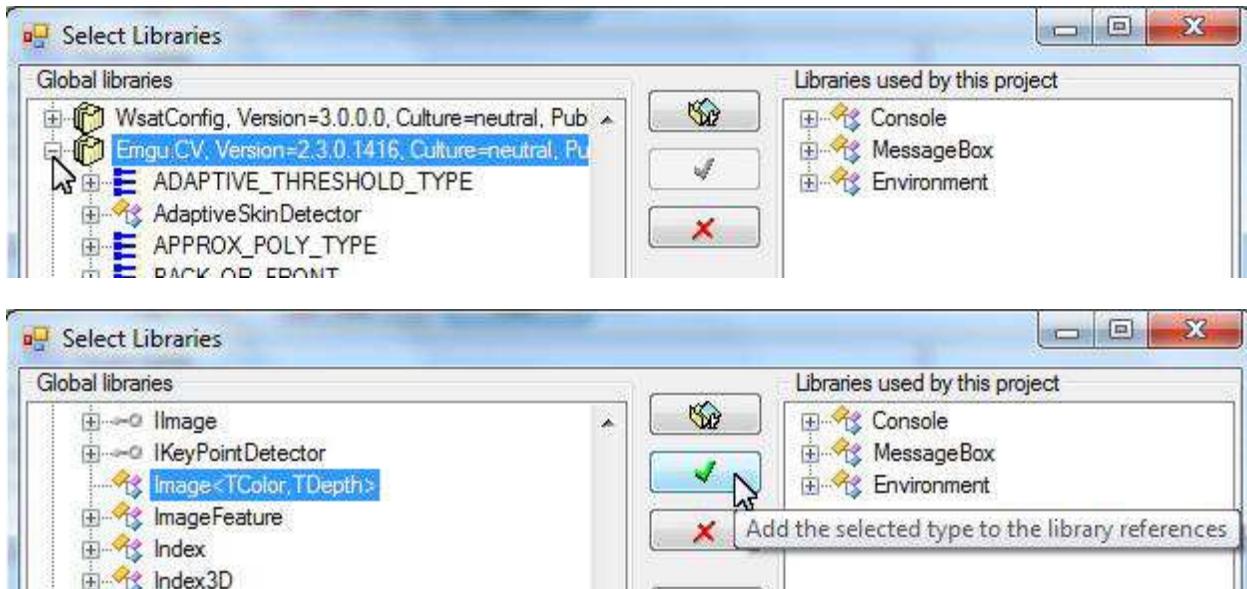
Use Generic Types and Native Libraries



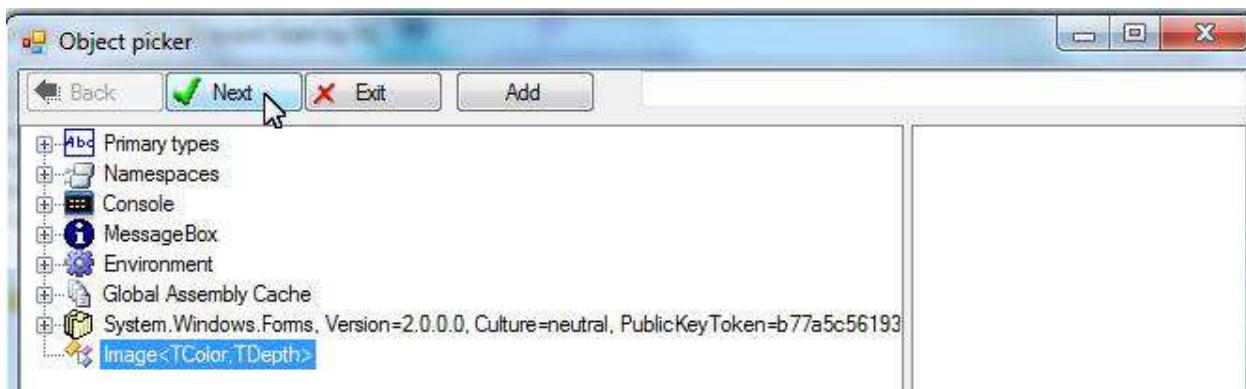
Select Emgu.CV.dll from the Emgu installation folder:



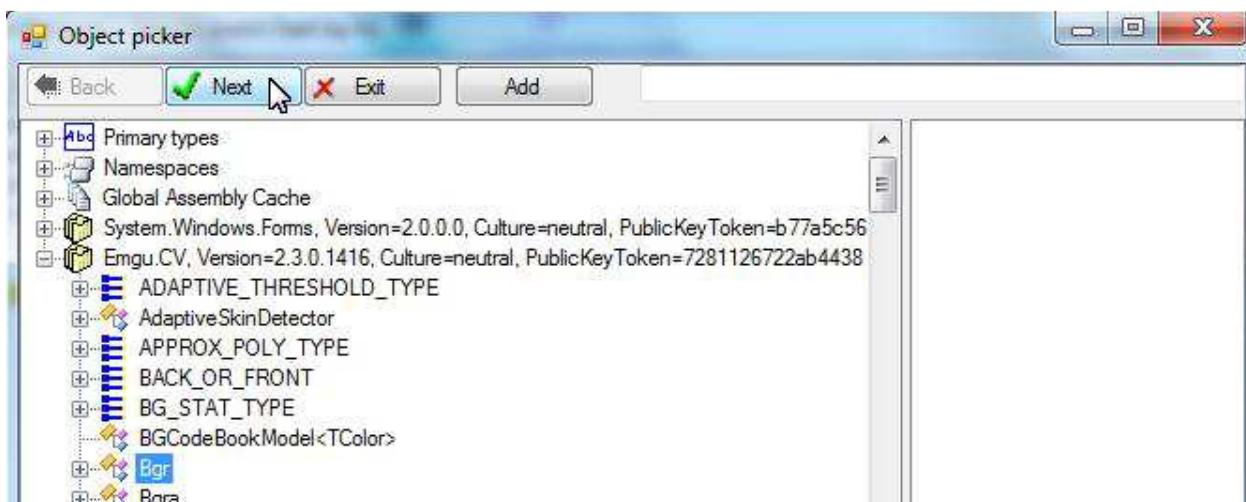
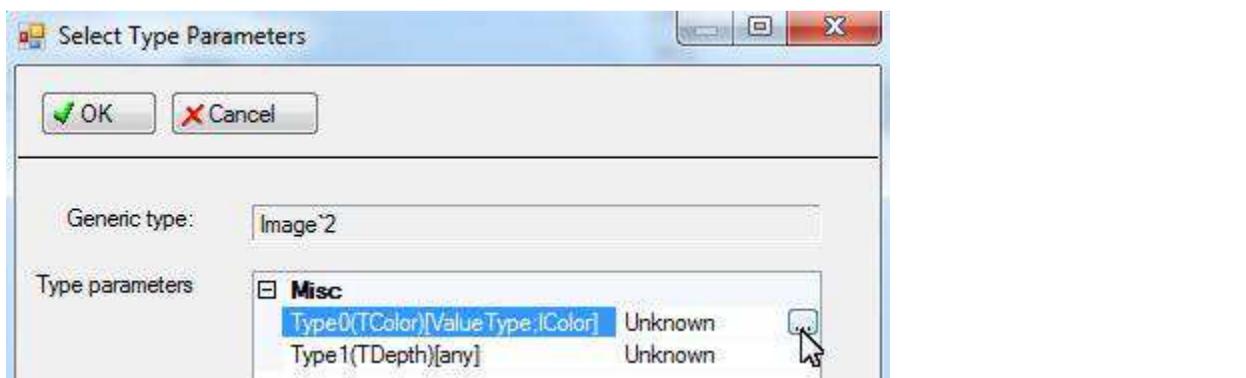
Select the Image class from the Emgu library:



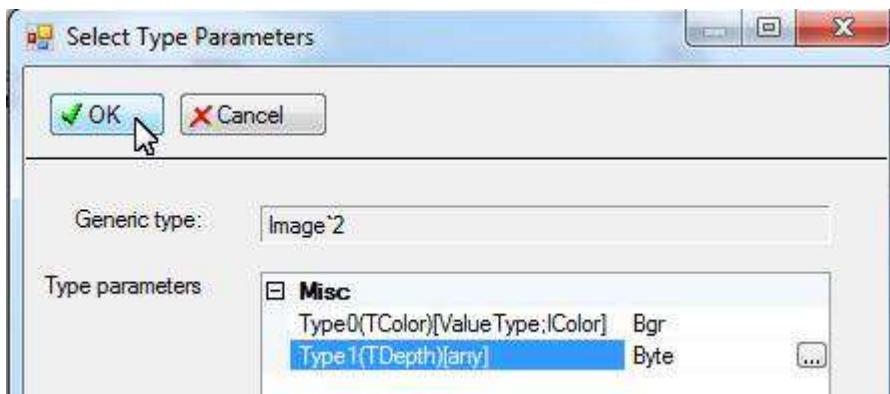
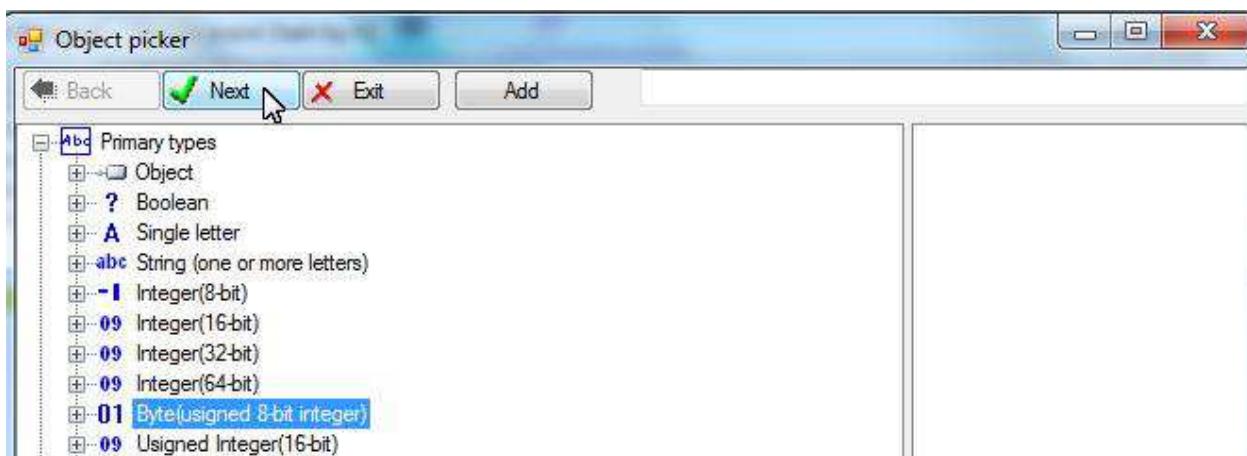
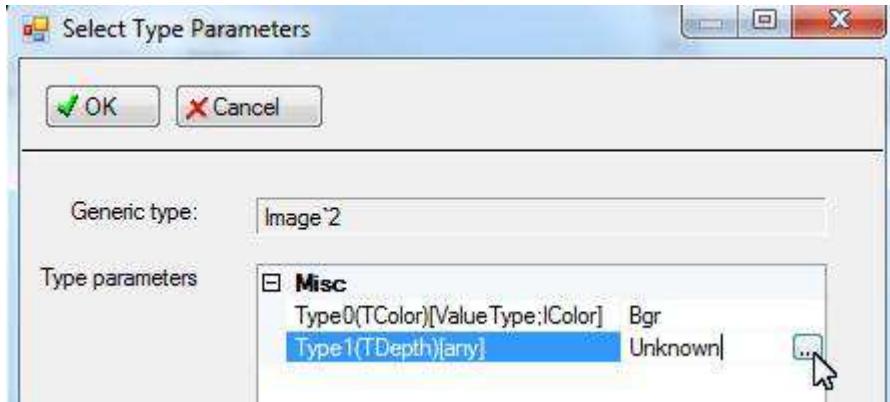
Use Generic Types and Native Libraries



The Image class contains two generic types, TColor and TDepth. According to the original Emgu sample, TColor should be Bgr from the Emgu library, and TDepth should be byte.

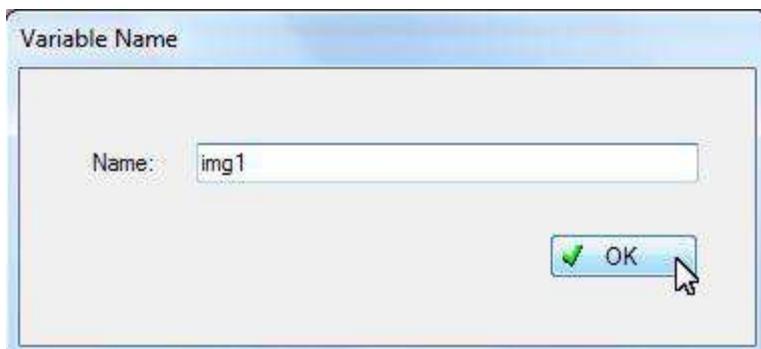


Use Generic Types and Native Libraries

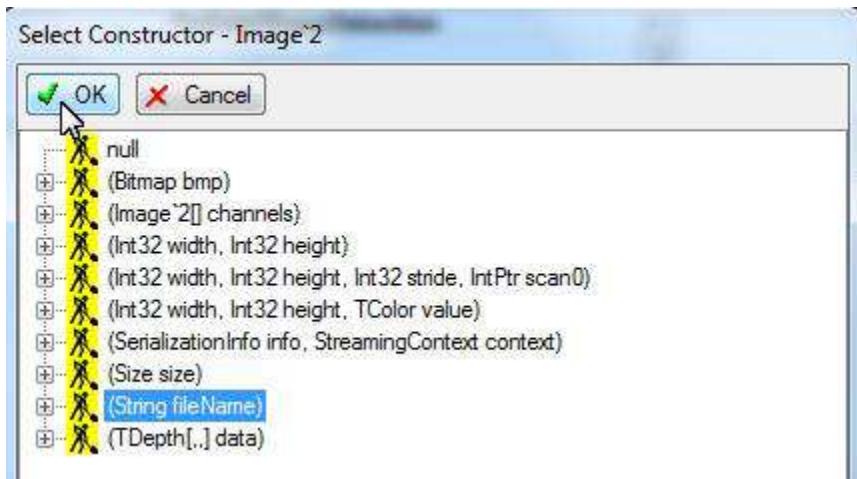


Give a name to the image variable:

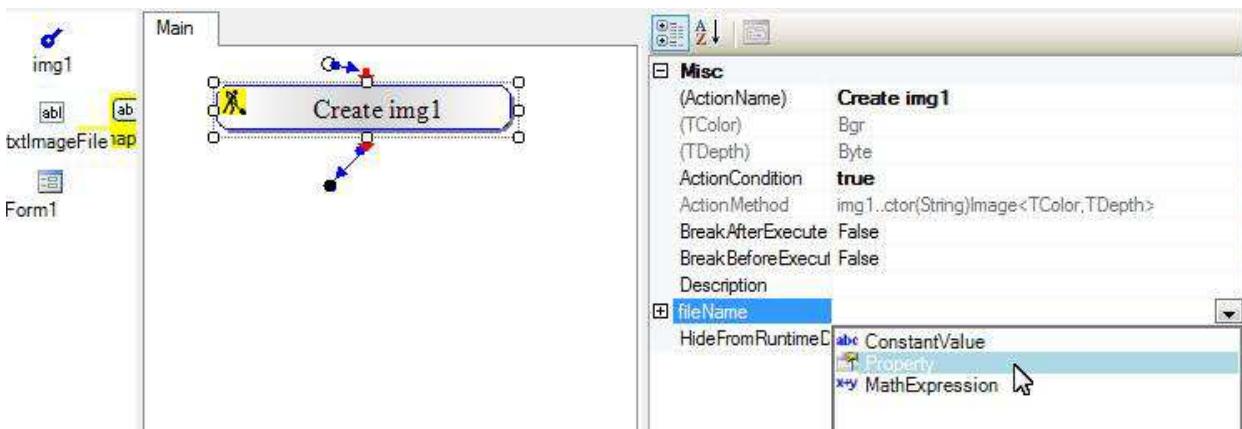
Use Generic Types and Native Libraries



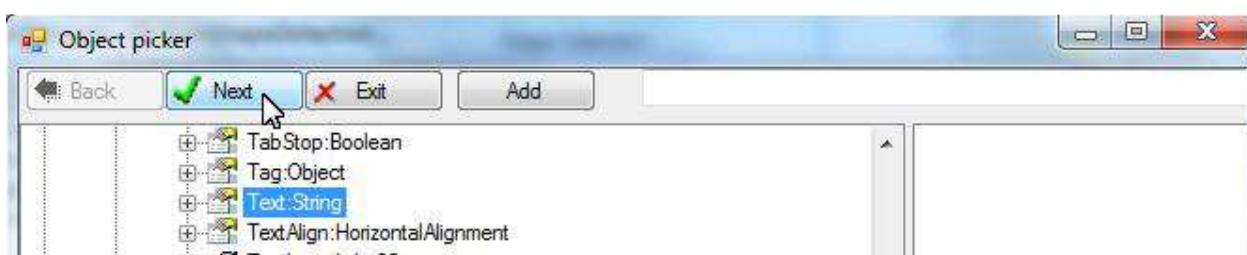
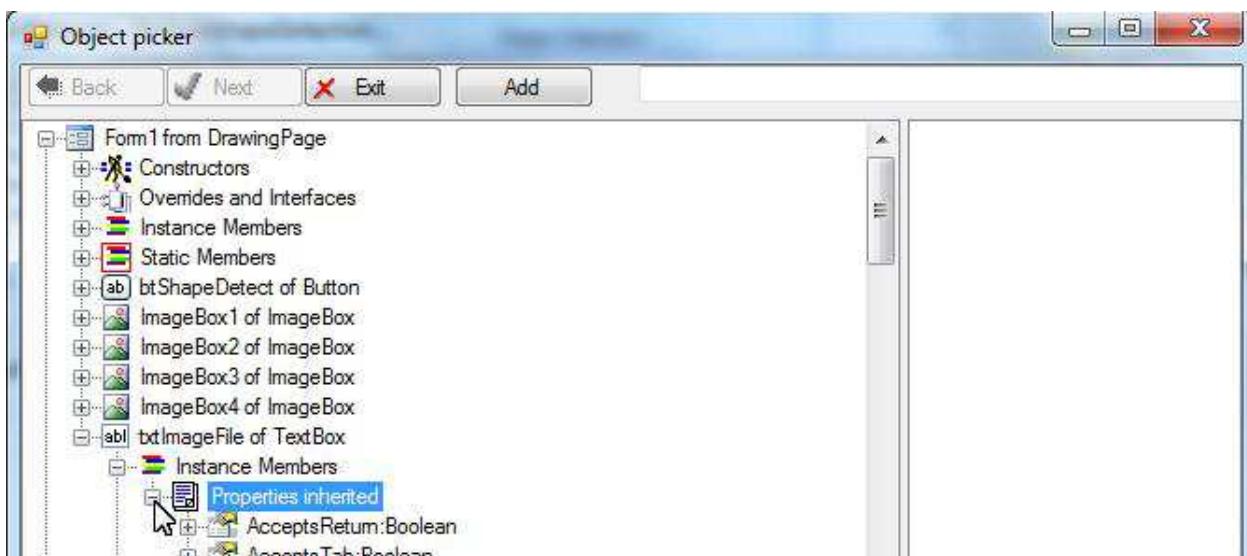
Select the constructor which takes the image file name:



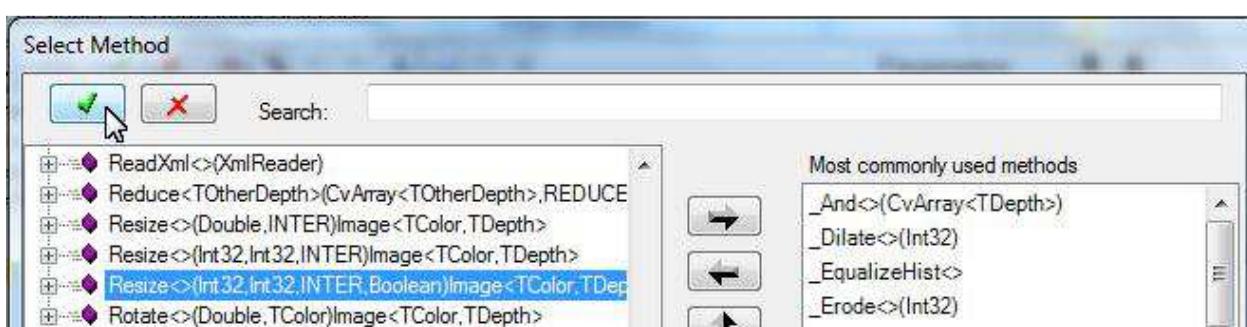
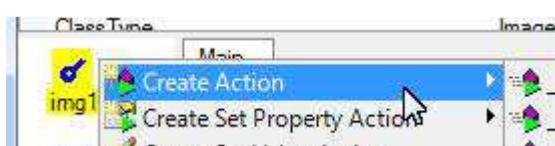
An image variable is created. An action appears. The action creates the image variable. We need to provide the image file name for the action.



Use Generic Types and Native Libraries



Resize the image to 400x400:

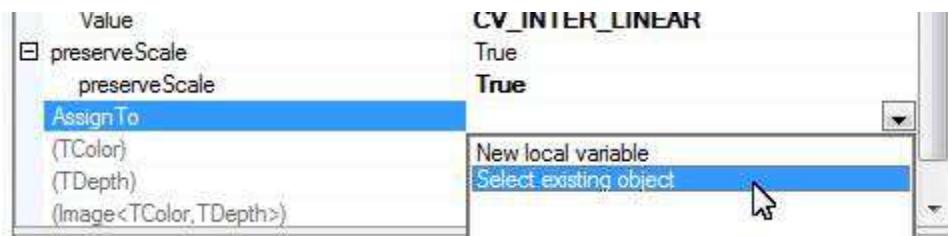


Provide the action parameters values:

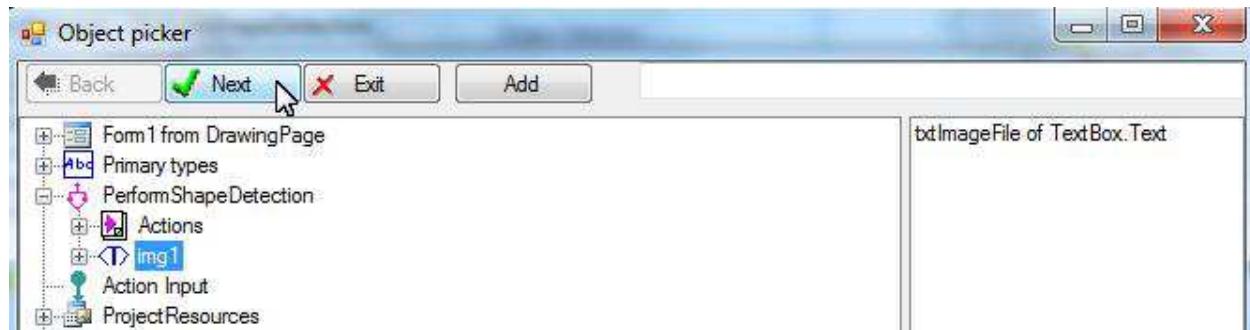


Use Generic Types and Native Libraries

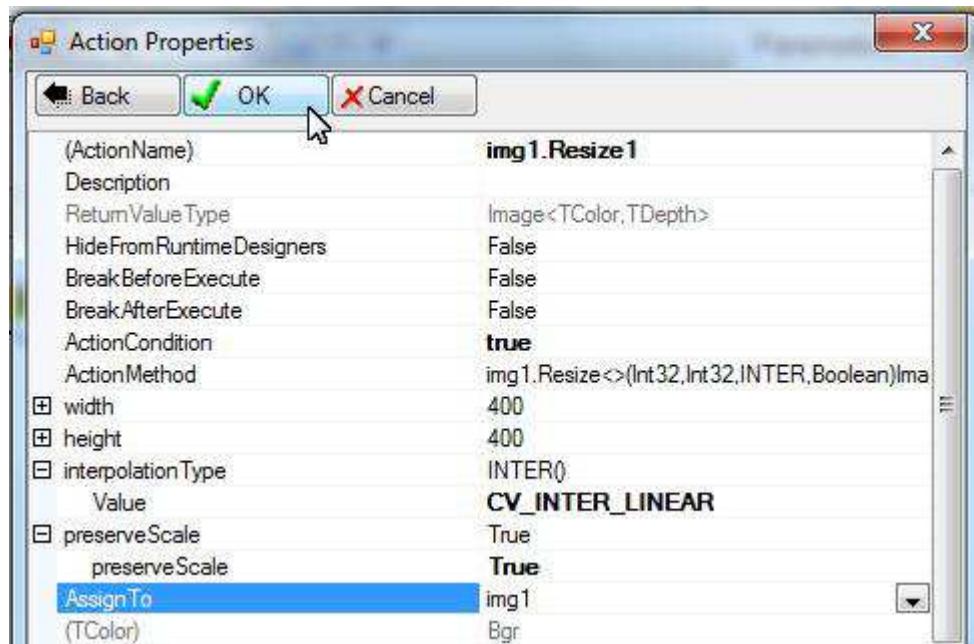
For “AssignTo”, choose “Select existing object”



Select the image variable itself:



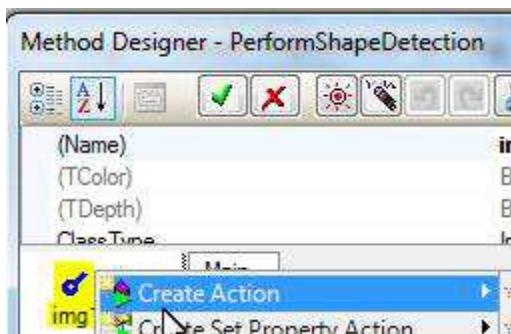
Click OK to finish creating this action:



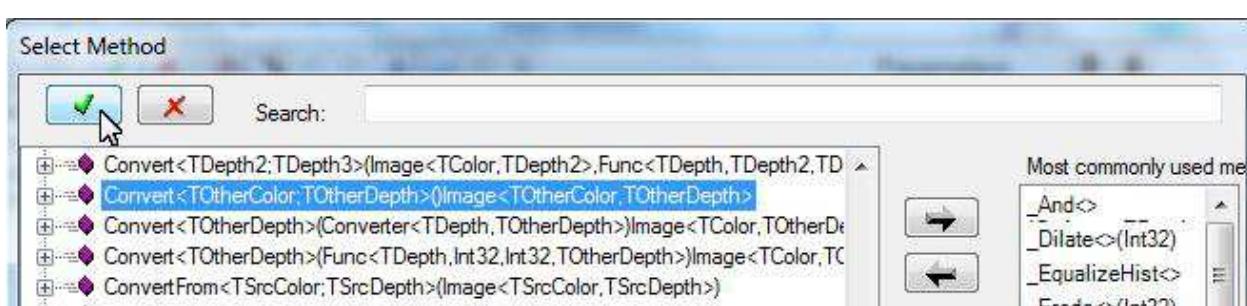
Convert Image to Gray

Shape detection needs gray image. A gray image can be generated from an existing image.

Use Generic Types and Native Libraries

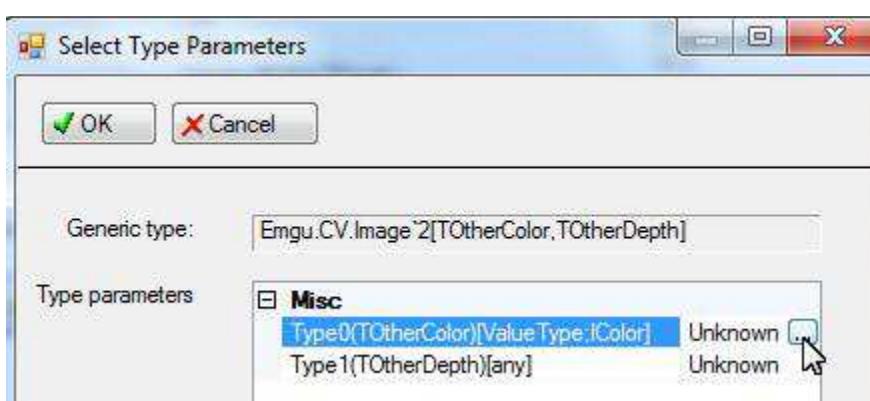


The Method Designer interface shows a node named 'Create Action' selected in the list.

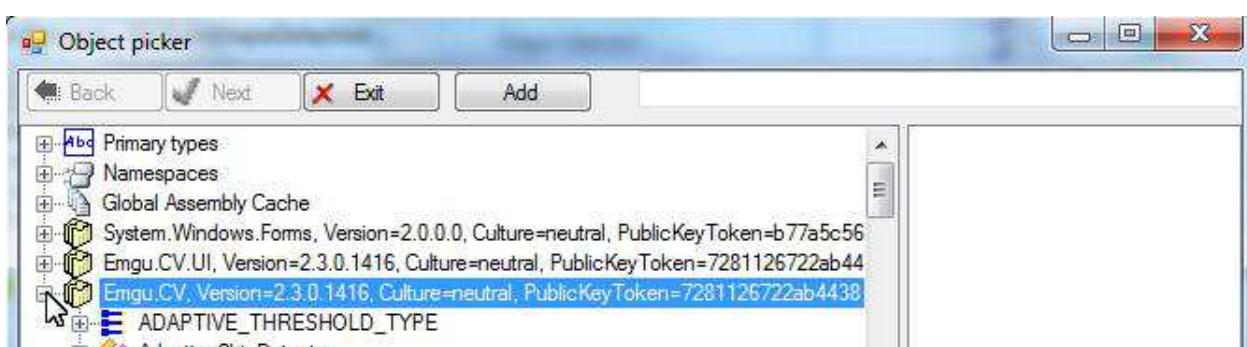


The Select Method dialog displays a list of methods under the category 'Convert'. One method, 'Convert<TOtherColor, TOtherDepth>(Image<TColor, TDepth2>, Func<TDepth, TDepth2, TDepth3>) Image<TOtherColor, TOtherDepth>', is highlighted.

Select Gray as the color:

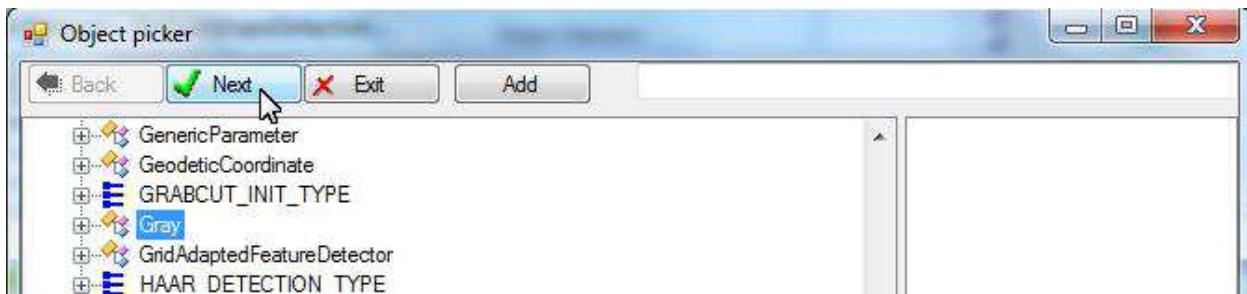


The Select Type Parameters dialog shows the generic type 'Emgu.CV.Image<2[TOtherColor, TOtherDepth]>' and its type parameters. The 'Misc' section lists 'Type0(TOtherColor)[Value Type : IColor]' and 'Type1(TOtherDepth)[any]'. The 'Type1' entry has a dropdown menu open, with 'Unknown' selected.

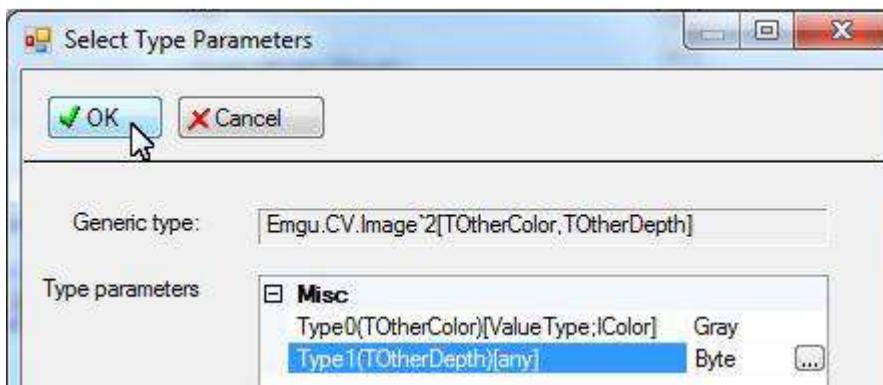
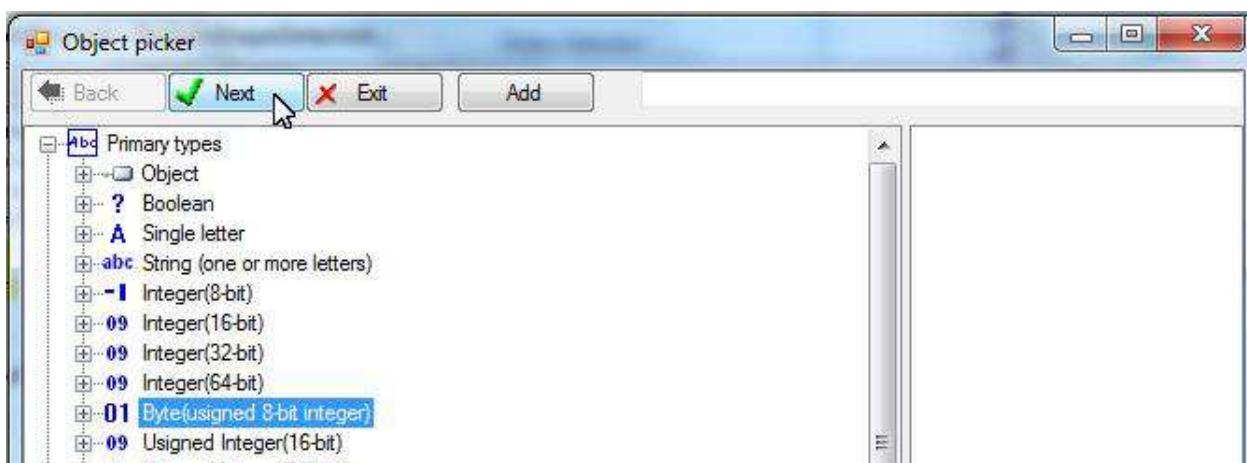
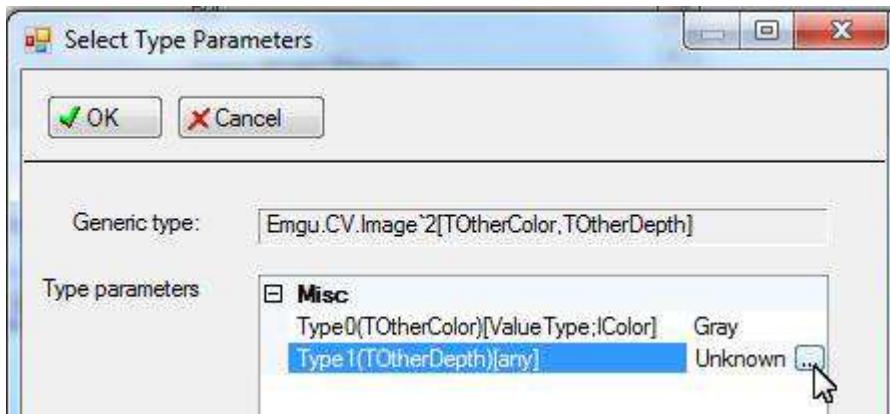


The Object picker dialog shows a tree view of assemblies. The 'Emgu.CV' assembly is selected, revealing its internal types, including 'ADAPTIVE_THRESHOLD_TYPE'.

Use Generic Types and Native Libraries

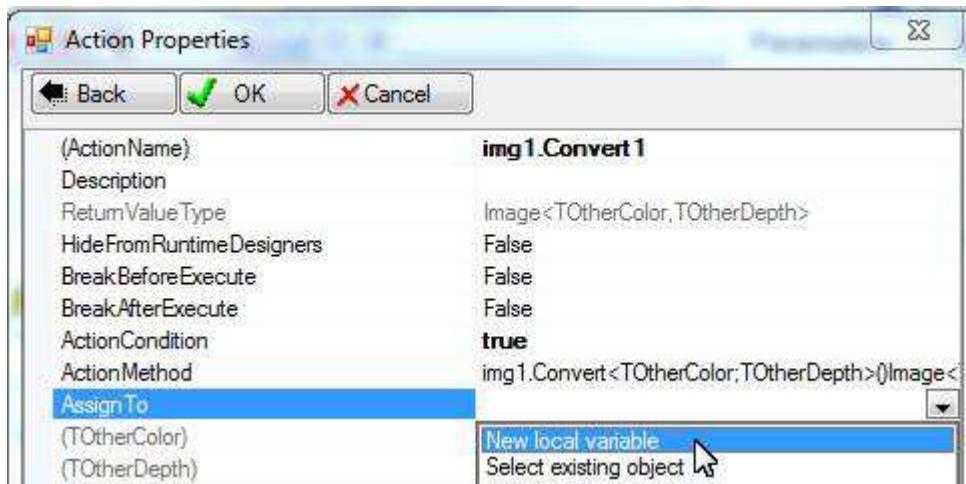


"TOtherDepth" is Byte:

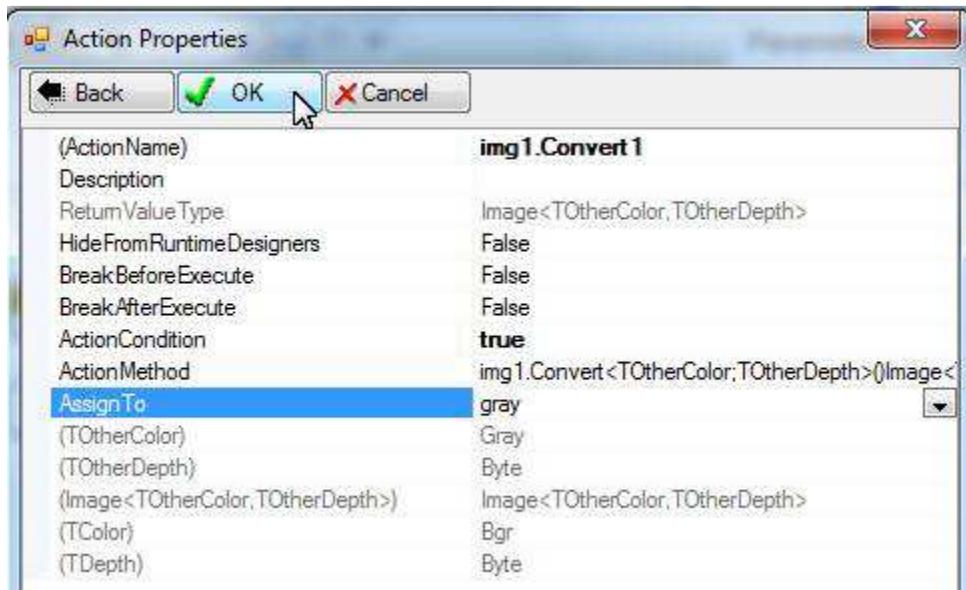


Use Generic Types and Native Libraries

For “AssignTo”, select “New local variable”:

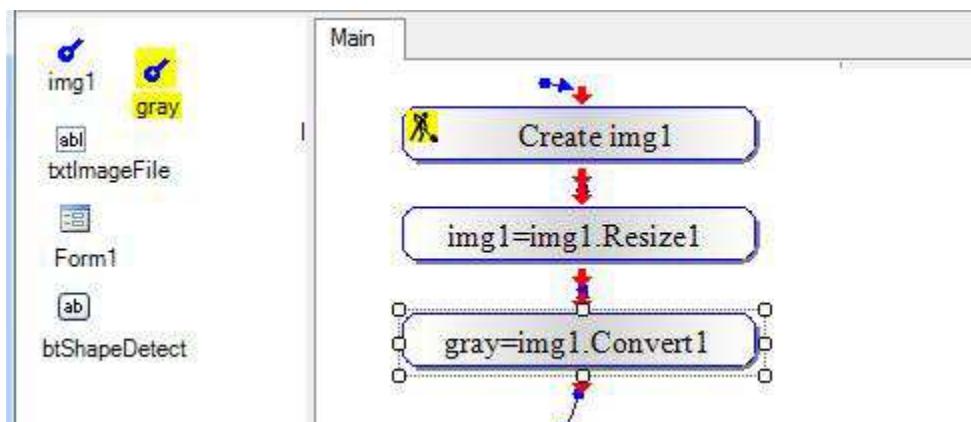


Click OK:

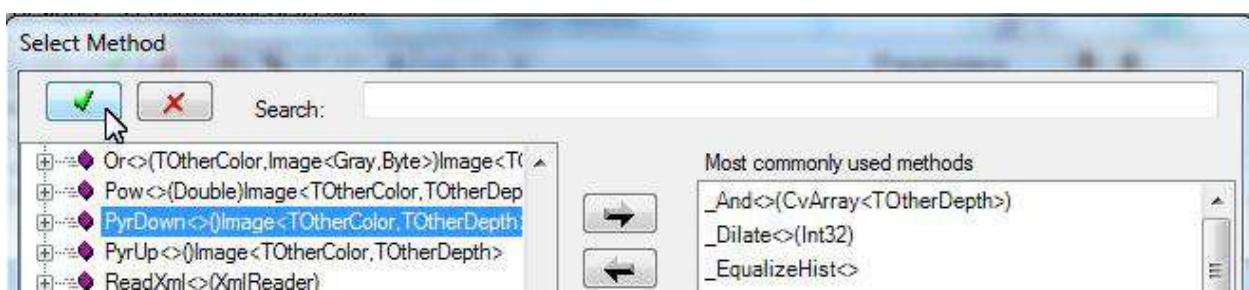
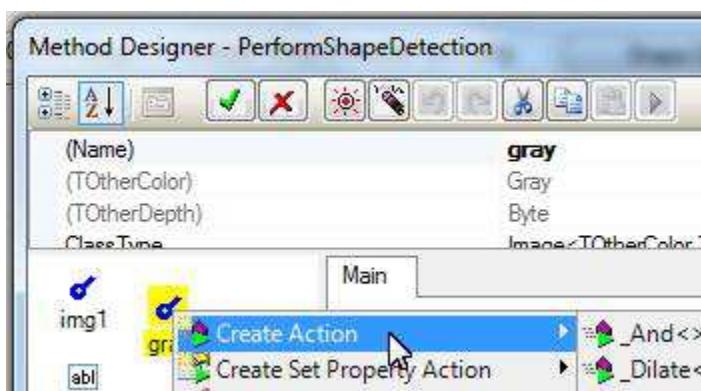


Use Generic Types and Native Libraries

A variable named “gray” appears in the Variable Pane. An action for creating “gray” appears in the Action Pane.

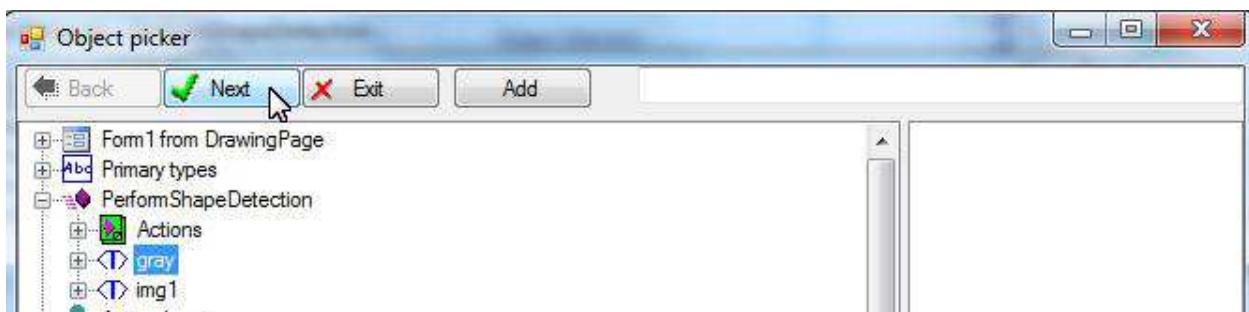
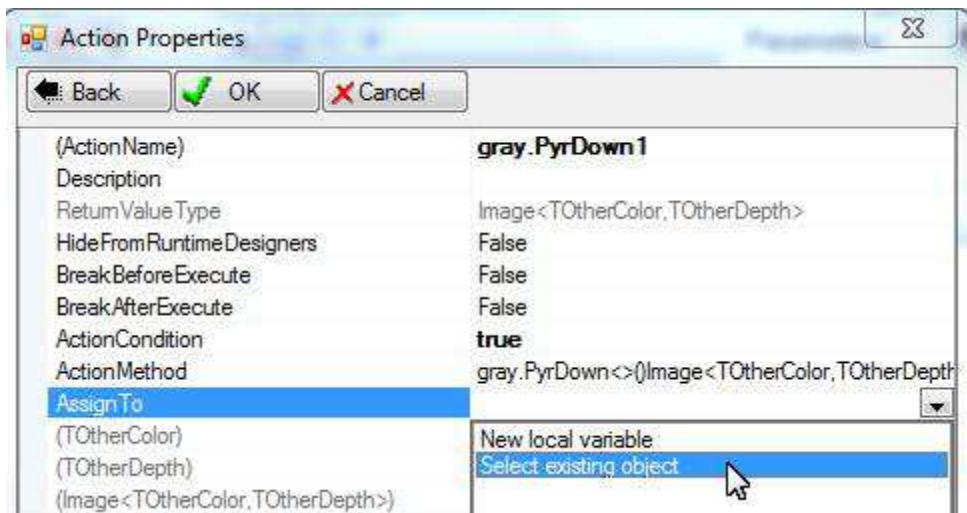


Use PyrDown and PyrUp to create two actions to filter out the noise.

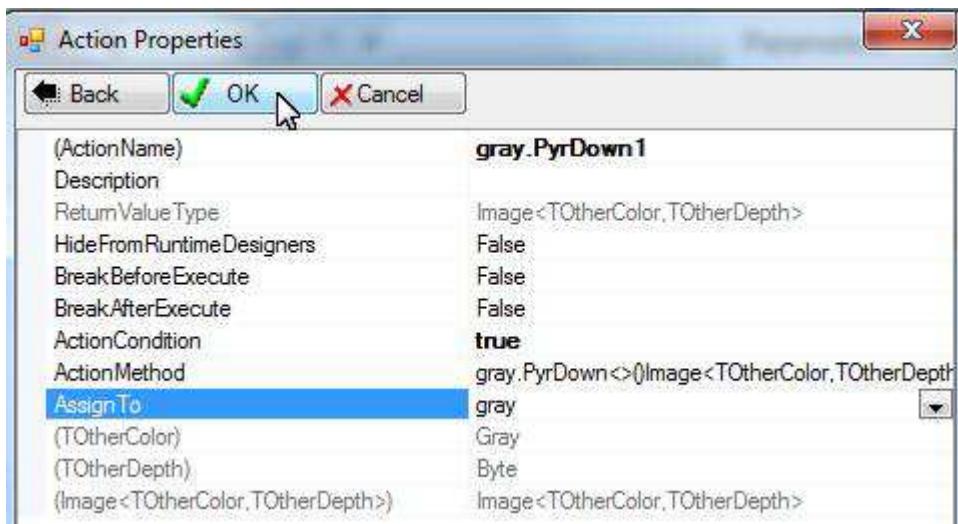


For “AssignTo”, use “Select existing object” and select variable “gray”:

Use Generic Types and Native Libraries

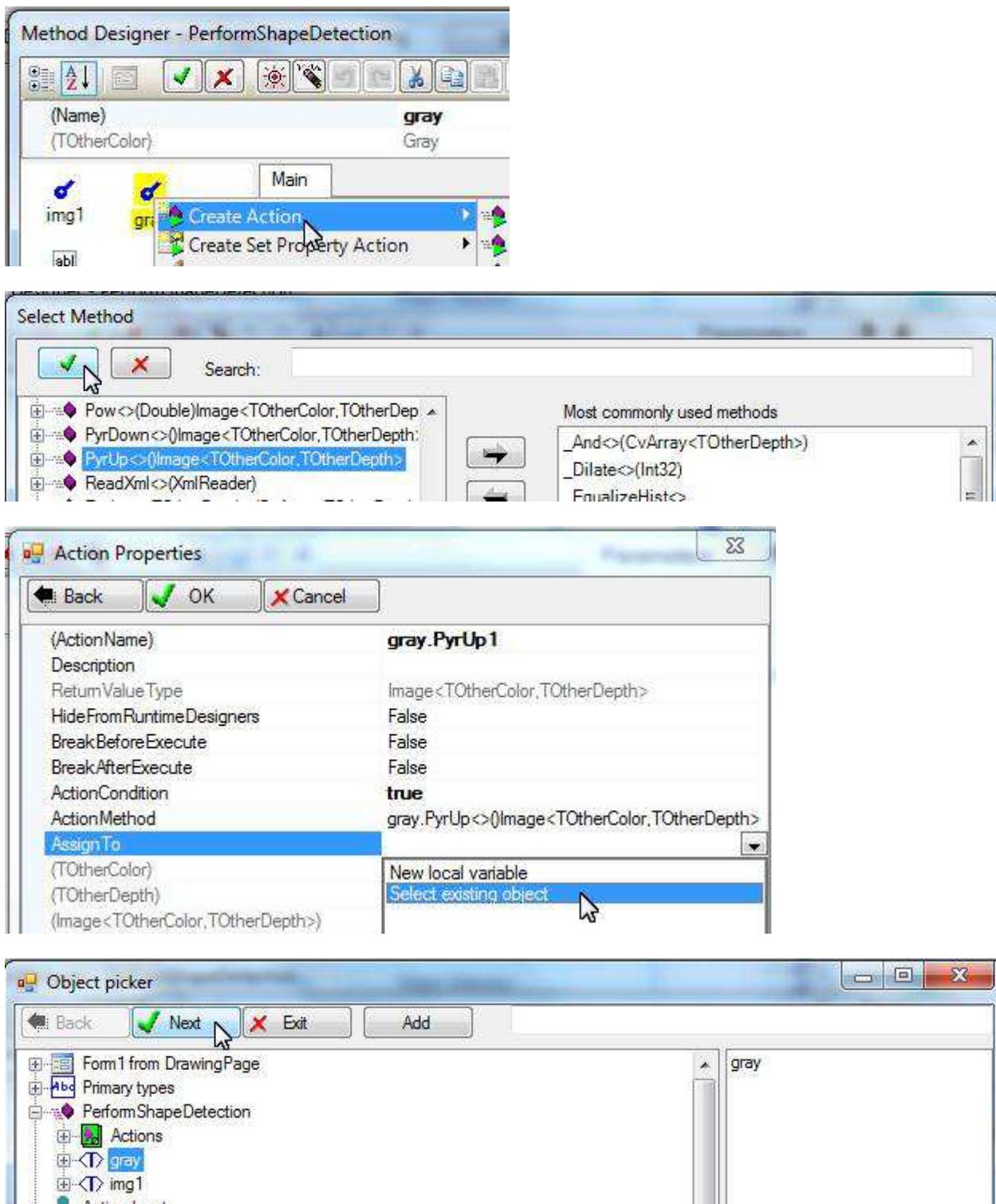


Click OK to finish creating this action:

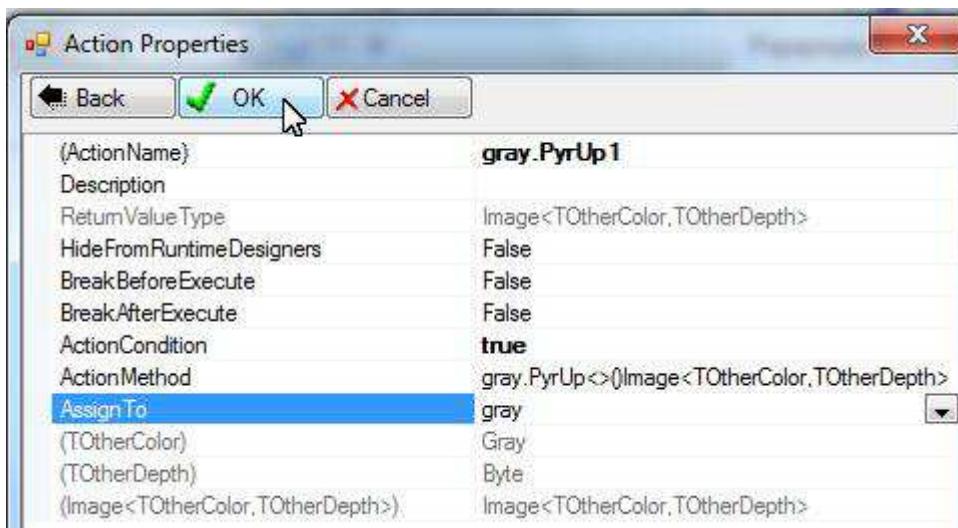


Create PyrUp action:

Use Generic Types and Native Libraries



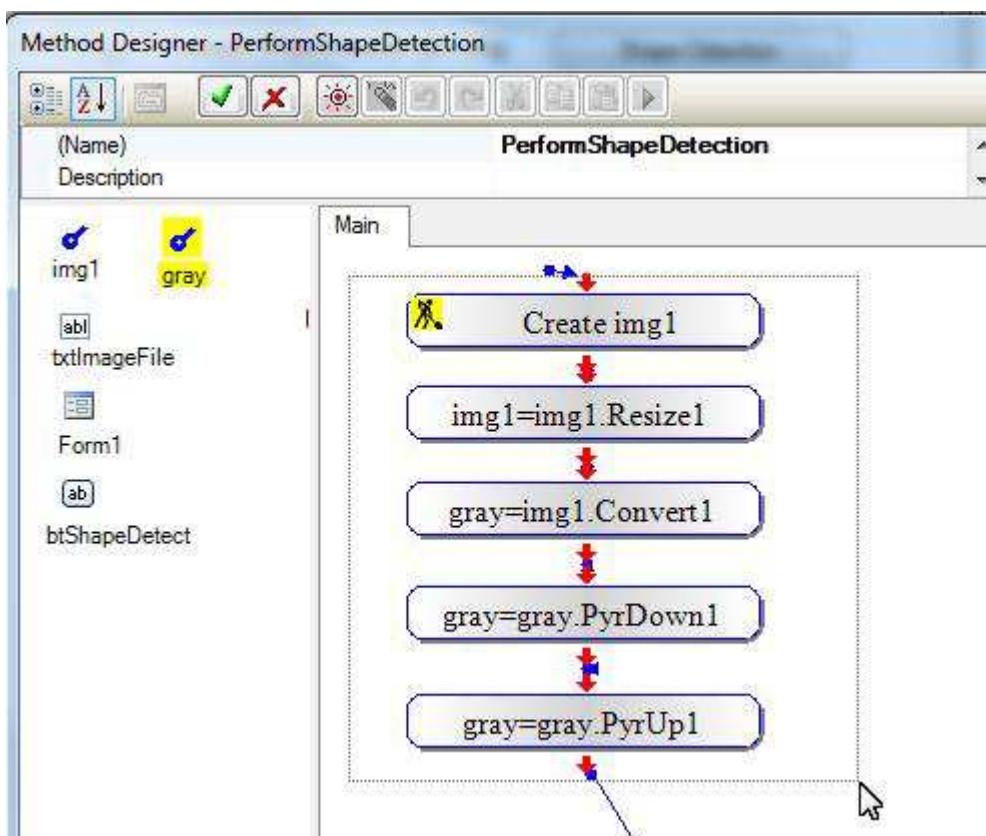
Use Generic Types and Native Libraries



Create Action List

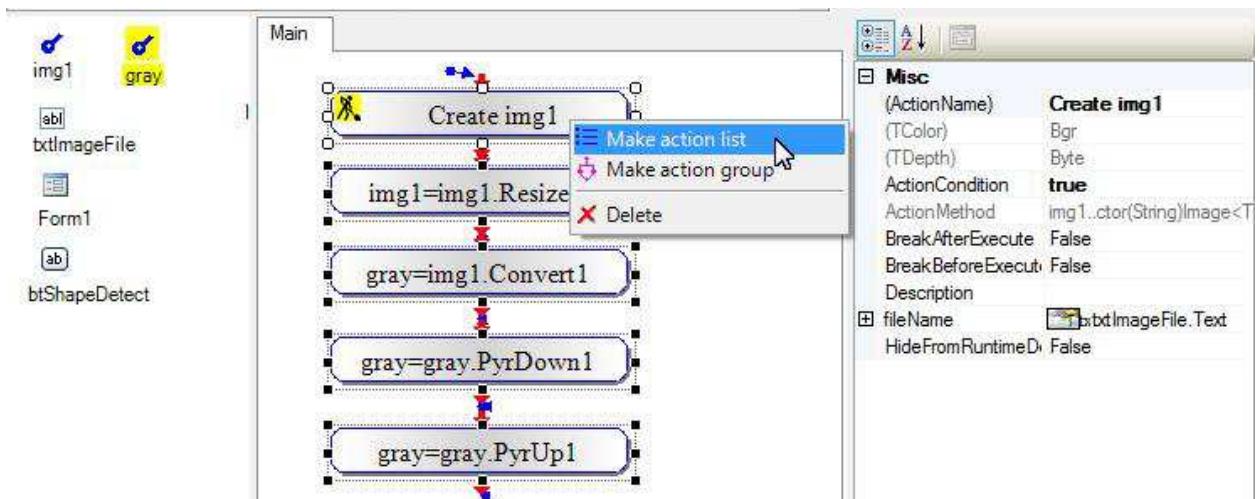
By now the Action Pane is pretty much full. We may create an action list to include all actions.

Select all the actions:

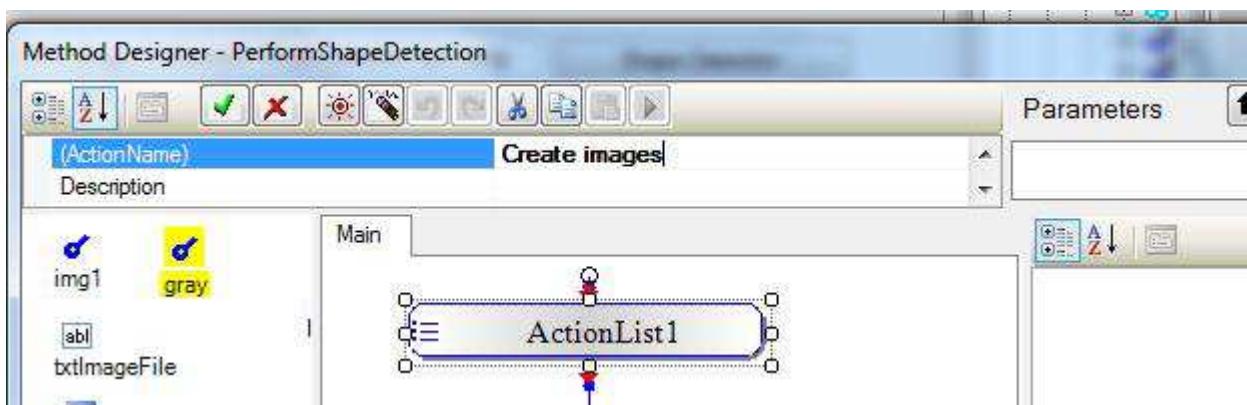


Right-click selected actions; choose "Make action list":

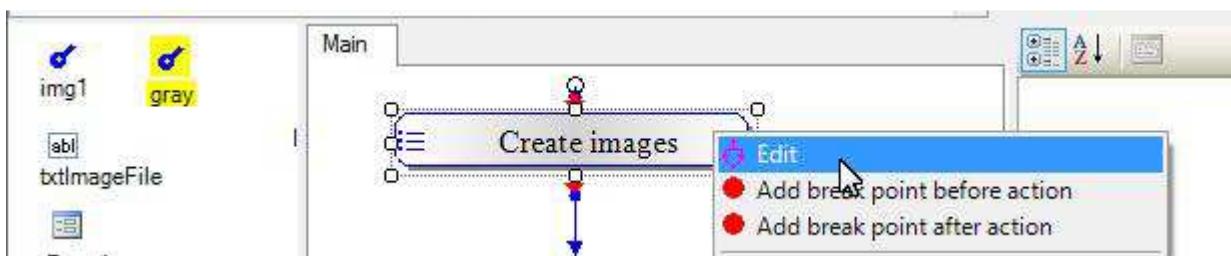
Use Generic Types and Native Libraries



An action list is created. We may rename it to “Create images”:

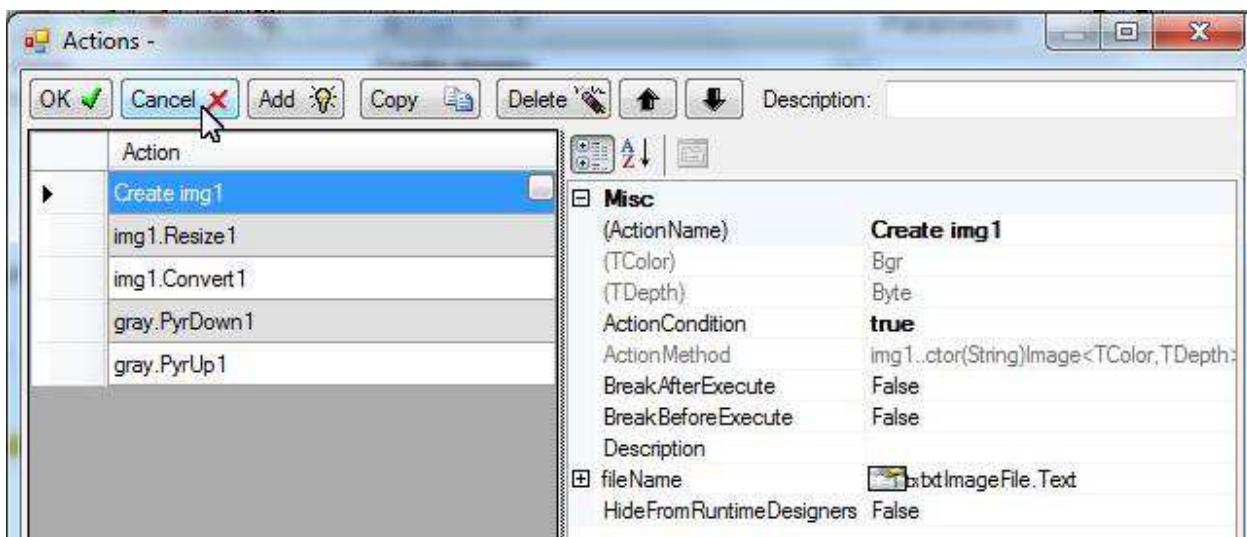


To modify it, right-click it and choose “Edit”:



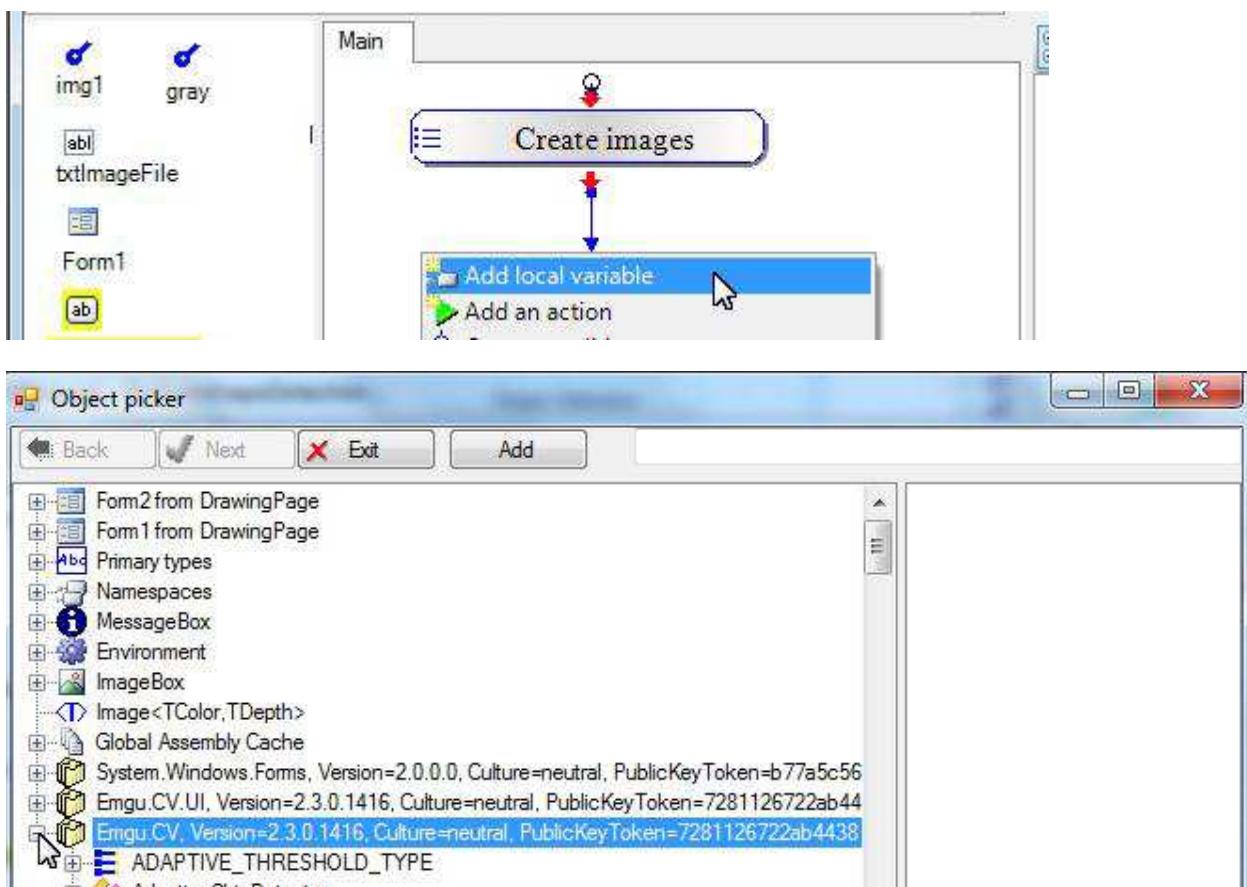
A large number of actions can be included. You may change the action order and modify each action.

Use Generic Types and Native Libraries

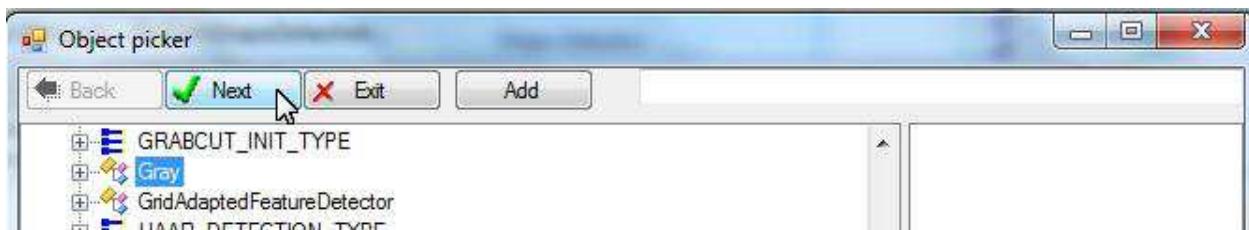


Circles Detect

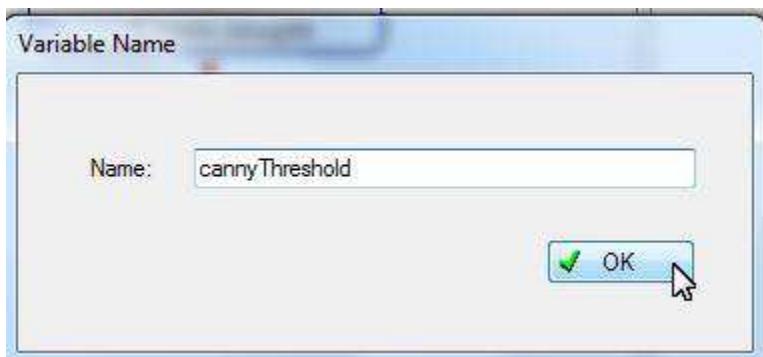
A gray image's HoughCircles method is used to detect circles. This method needs two Gray objects. Let's create two Gray objects first.



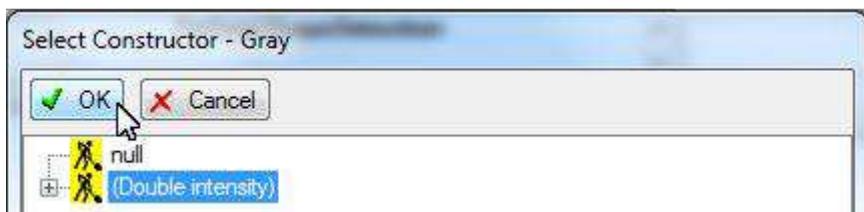
Use Generic Types and Native Libraries



Give a variable name:

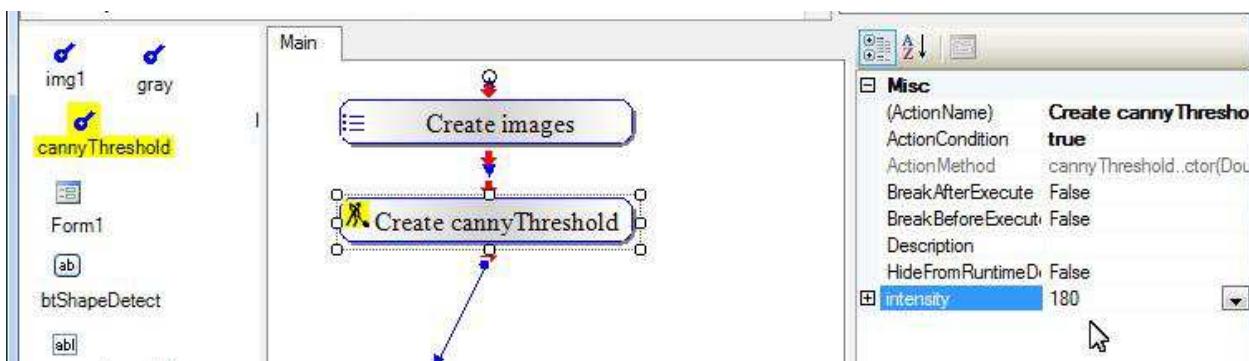


Select the constructor:



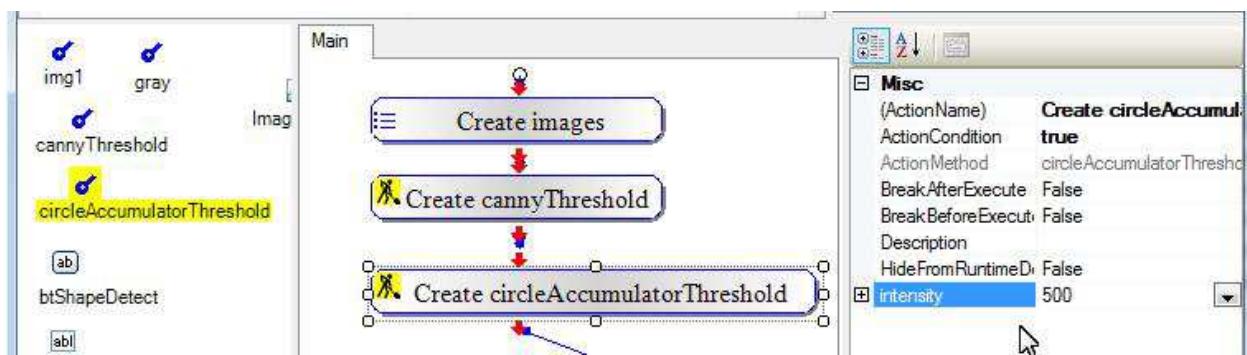
The variable "cannyThreshold" appears in the Variable Pane. An action appears in the Action Pane for creating the variable.

According to the tutorial from Emgu, 180 is used for "cannyThreshold":

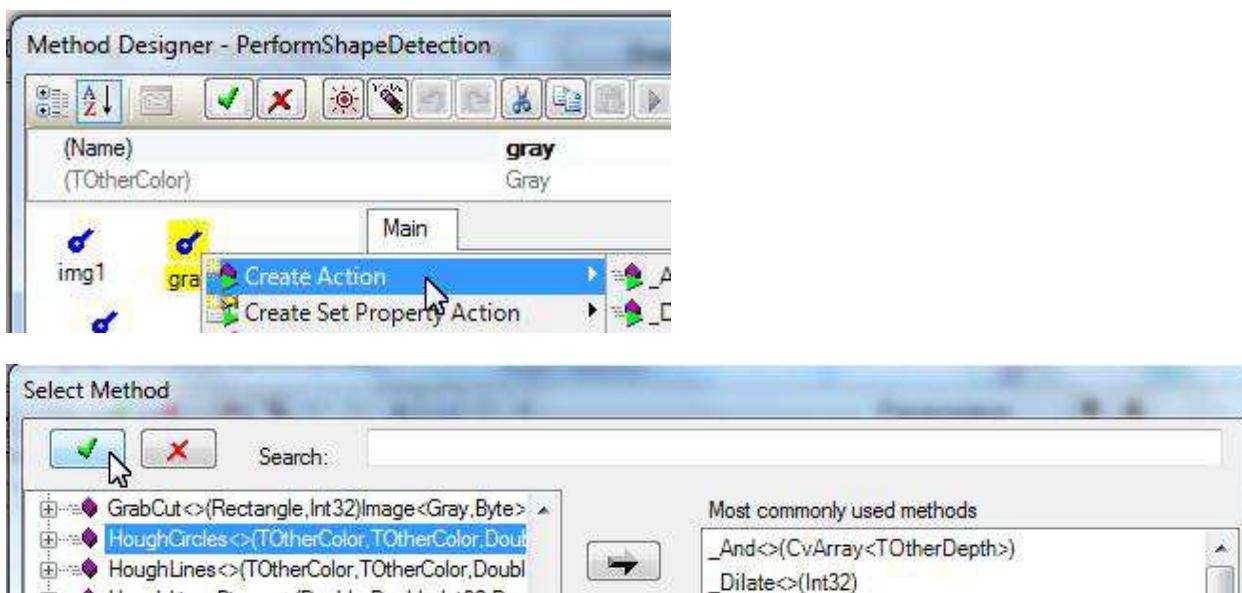


Create another Gray object with value 500:

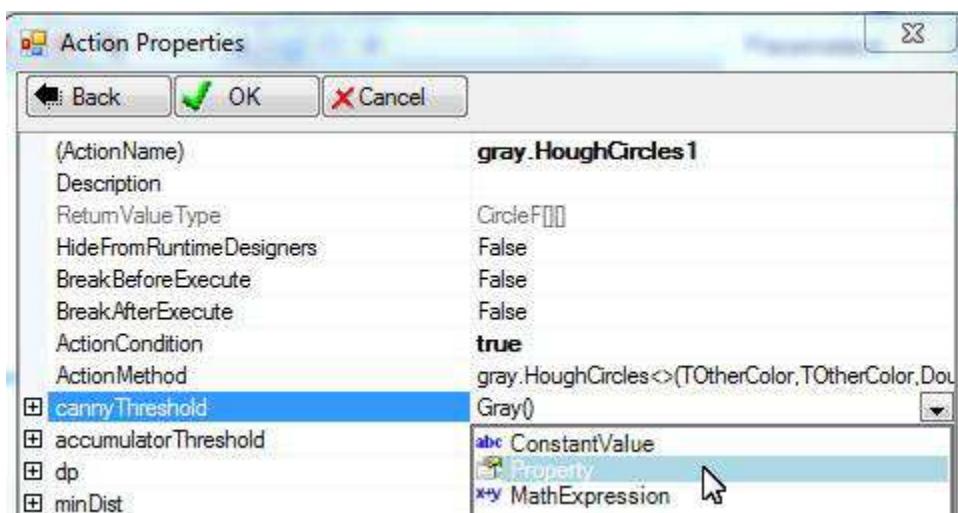
Use Generic Types and Native Libraries



Now we may use variable "gray" to create HoughCircles action:



Assign to two Gray objects to the action:



Use Generic Types and Native Libraries

The figure consists of three vertically stacked windows from a software interface:

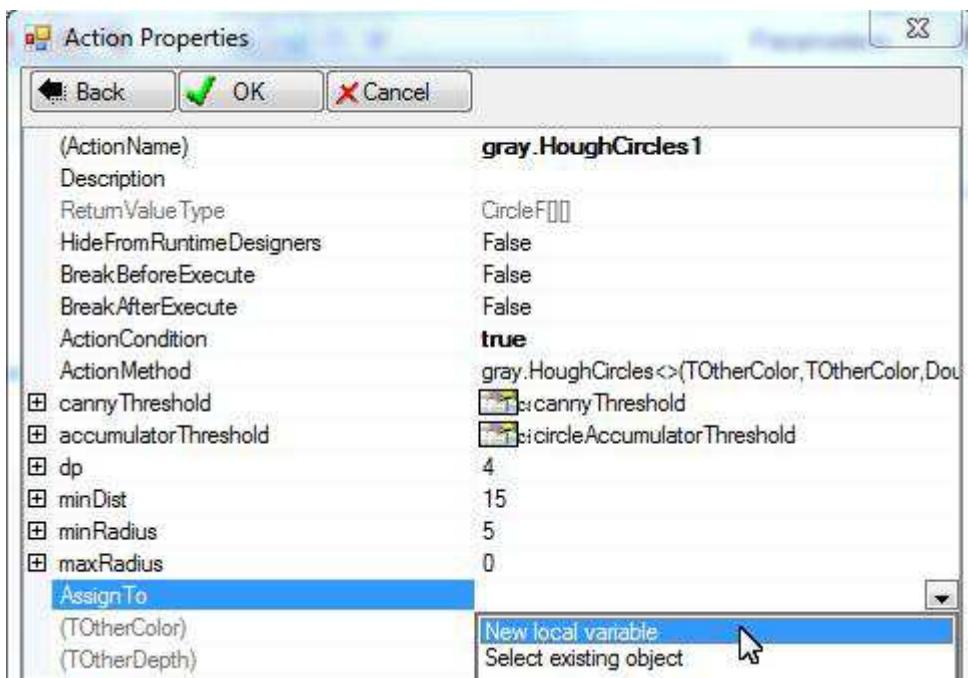
- Object picker (Top Window):** Shows a tree view of objects. The "PerformShapeDetection" node has two children: "cannyThreshold" and "circleAccumulatorThreshold". The "cannyThreshold" node is selected. The status bar at the bottom right shows "gray cannyThreshold".
- Action Properties (Middle Window):** A dialog for the "cannyThreshold" action. It contains the following properties:

(ActionName)	gray.HoughCircles1
Description	
ReturnValueType	CircleF[]
HideFromRuntimeDesigners	False
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	true
ActionMethod	gray.HoughCircles<(TOtherColor, TOtherColor, Dou
cannyThreshold	Gray()
accumulatorThreshold	Gray()
dp	abc ConstantValue
minDist	Property
minRadius	MathExpression
- Object picker (Bottom Window):** Similar to the top one, showing the "PerformShapeDetection" node with "cannyThreshold" and "circleAccumulatorThreshold" children. The "cannyThreshold" node is selected, and the status bar shows "gray cannyThreshold".

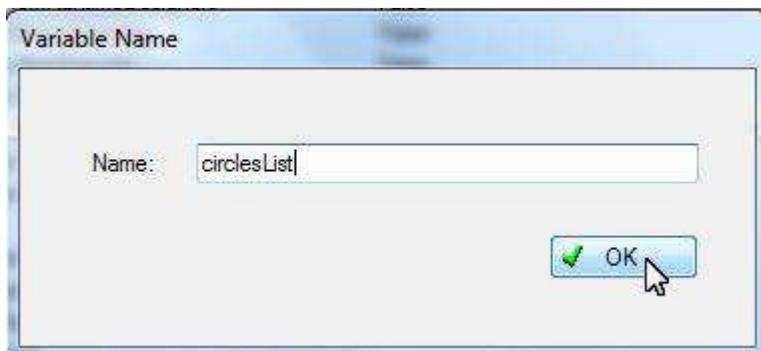
Set other parameter values according to the Emgu tutorial.

For “AssignTo”, select “New local variable”:

Use Generic Types and Native Libraries

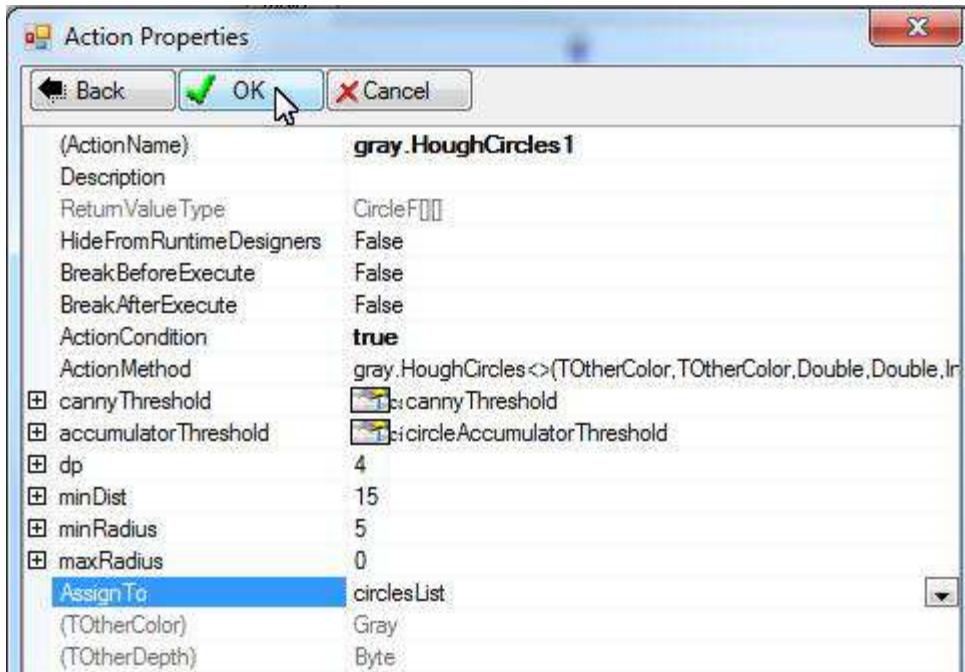


Give a new variable name:

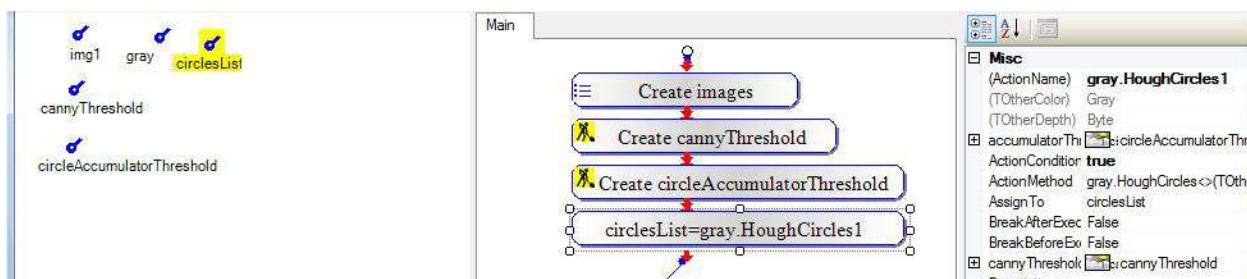


Click OK to finish creating this action:

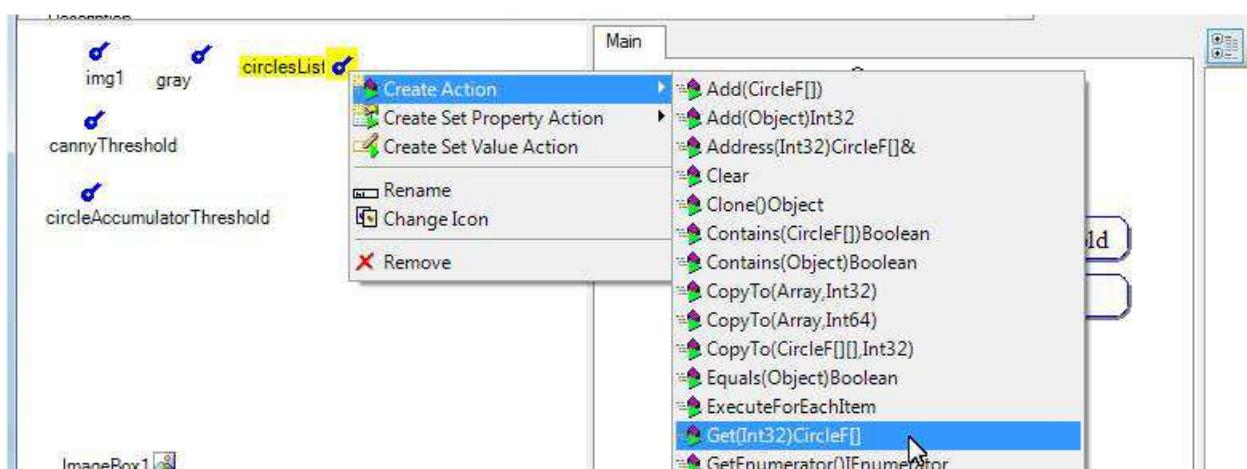
Use Generic Types and Native Libraries



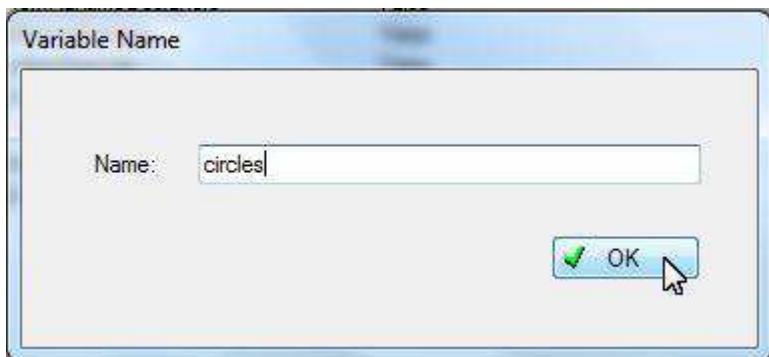
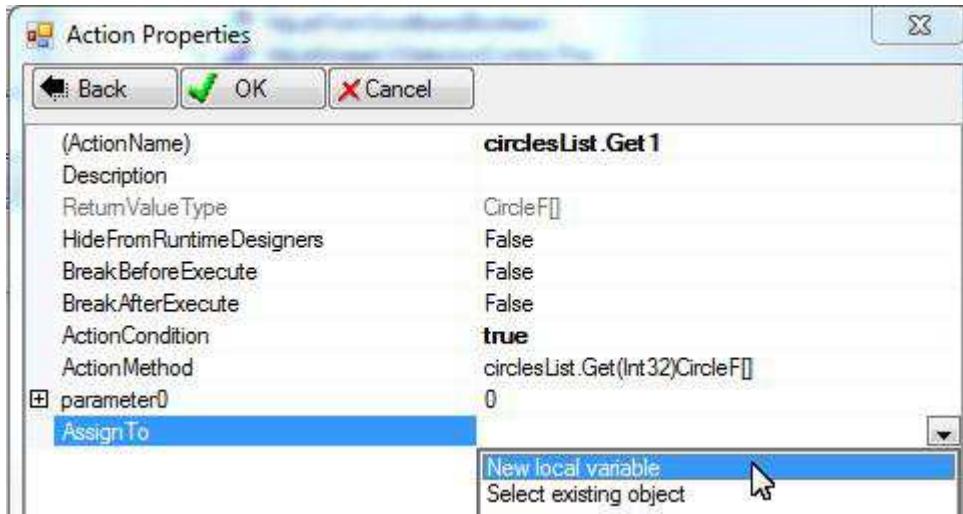
A new variable, "circlesList", appears in the Variable Pane. A new action creating "circlesList" appears in the Action Pane.



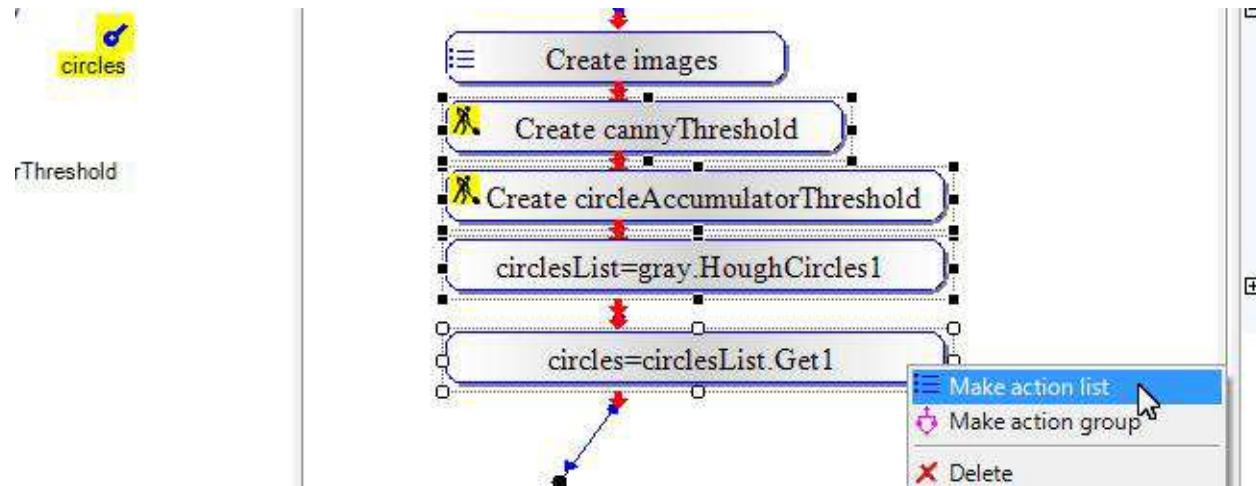
Get the first circle array from "circlesList":



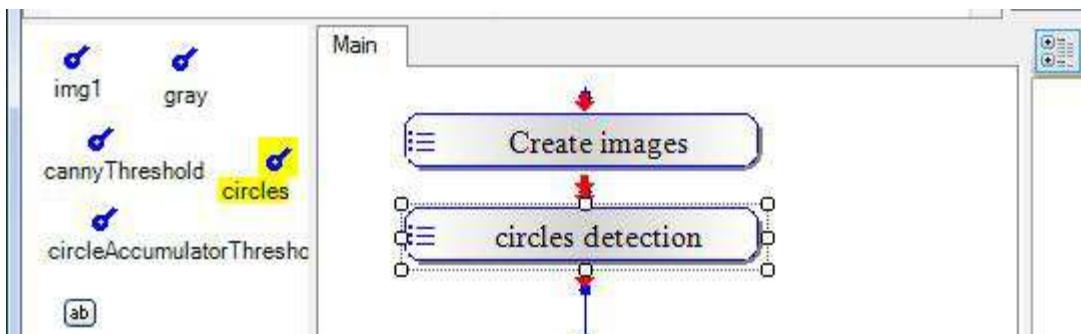
Use Generic Types and Native Libraries



We may group the above 4 action into a “circles detection” action list to make room for the editor:

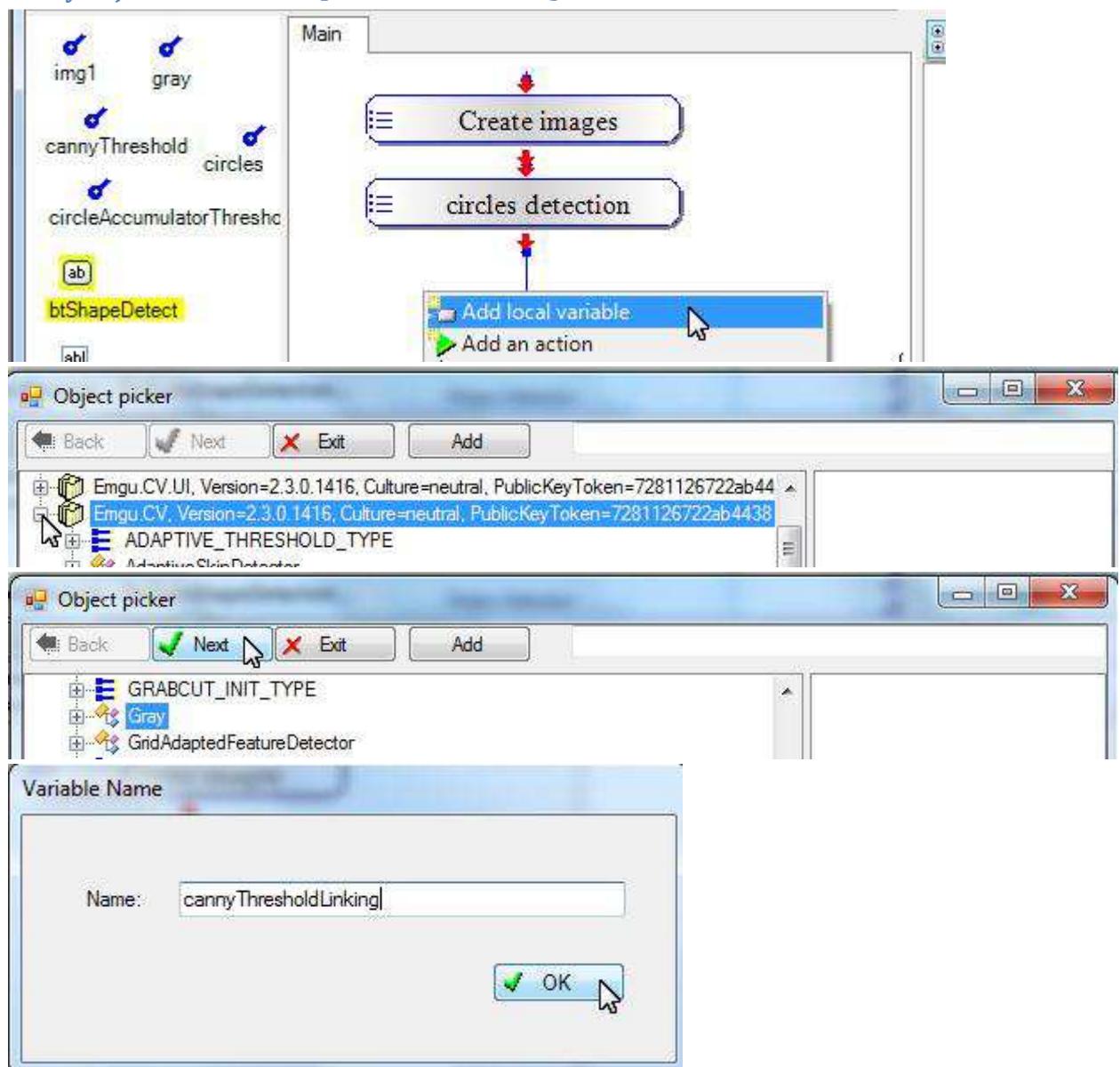


Use Generic Types and Native Libraries

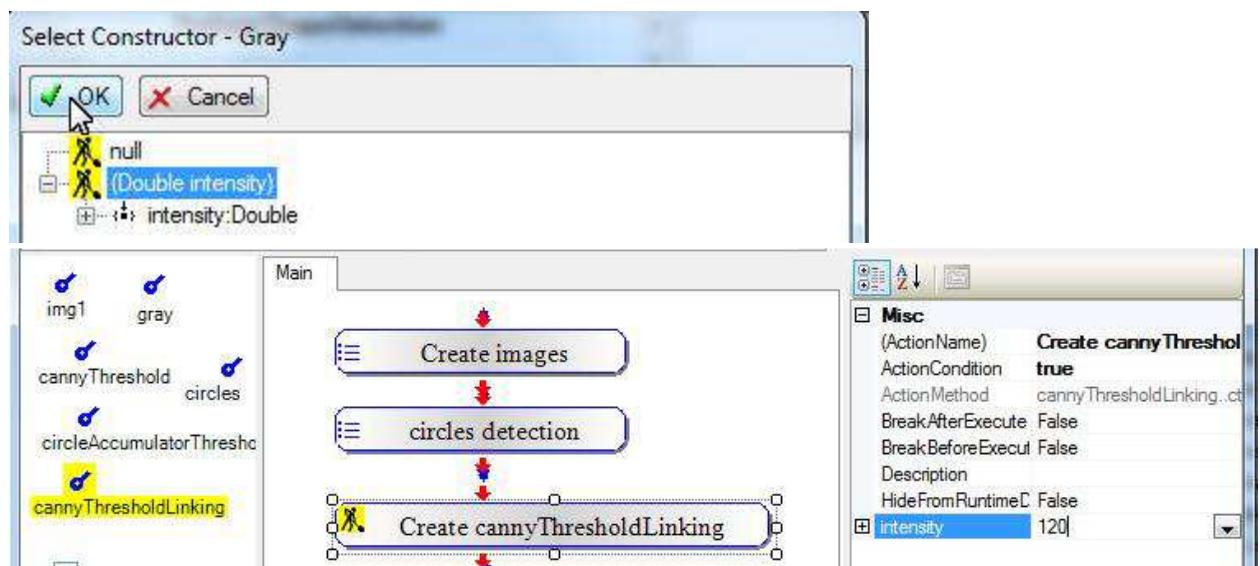


Lines Detection

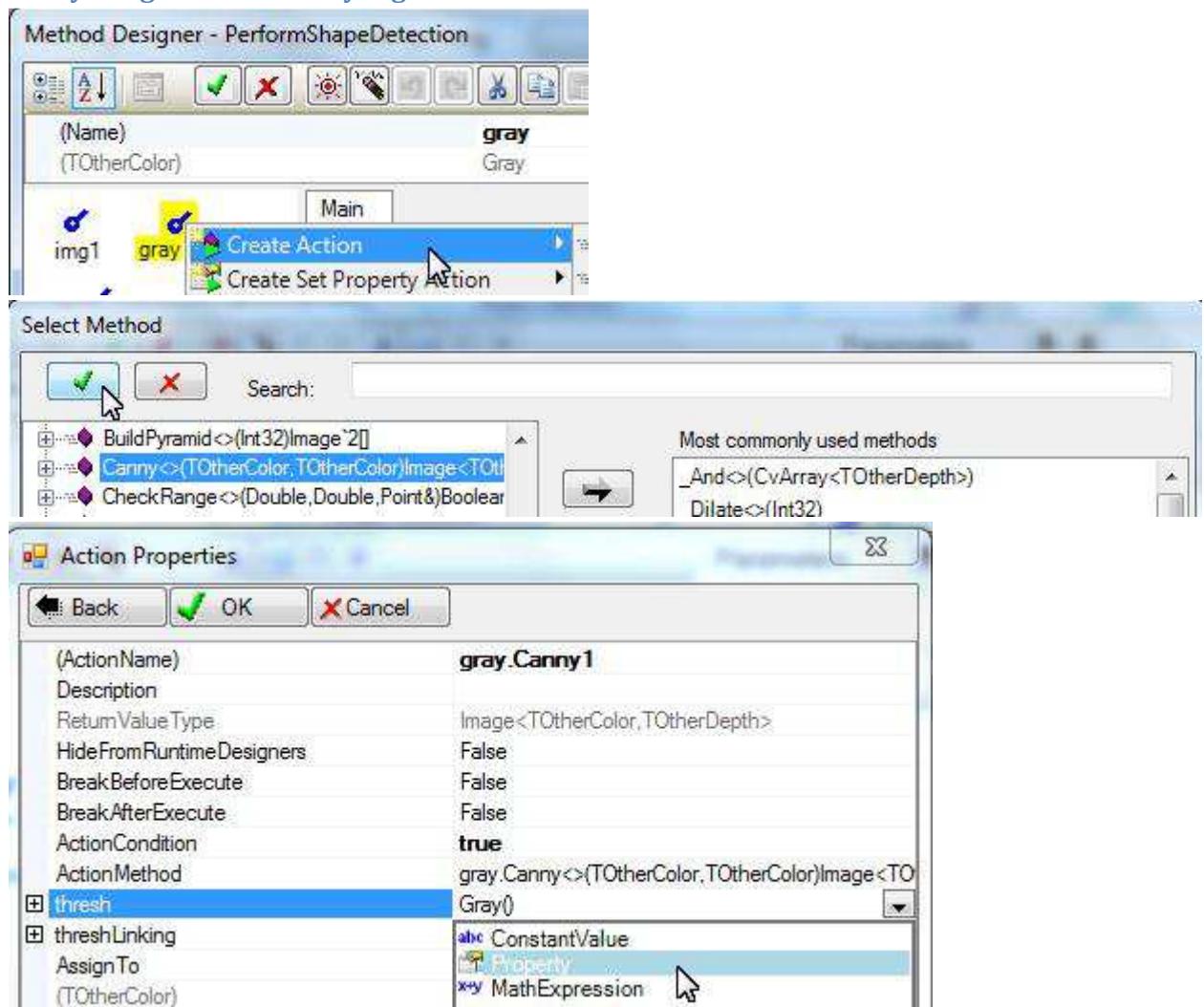
Create a Gray object named “cannyThresholdLinking”



Use Generic Types and Native Libraries



Create a Gray image named "cannyEdges".



Use Generic Types and Native Libraries

The screenshot shows two instances of the "Object picker" and "Action Properties" dialog boxes, likely from a software development environment.

Object picker (Top):

- Buttons: Back, Next (highlighted), Exit, Add.
- Tree View:
 - Form1 from DrawingPage
 - Primary types
 - PerformShapeDetection
 - Actions
 - cannyThreshold (selected)
 - cannyThresholdLinking
- Properties pane: gray, cannyThreshold, circleAccumulatorThreshold.

Action Properties (Top):

- Buttons: Back, OK (highlighted), Cancel.
- Properties:
 - (ActionName) gray.Canny1
 - Description
 - ReturnValueType Image<TOtherColor,TOtherDepth>
 - HideFromRuntimeDesigners False
 - BreakBeforeExecute False
 - BreakAfterExecute False
 - ActionCondition true
 - ActionMethod gray.Canny<>(TOtherColor,TOtherColor)Image<TOtherColor,TOtherDepth>
 - thresh cannyThreshold
 - threshLinking Gray()
 - AssignTo
 - ConstantValue
 - Property (selected)
 - MathExpression

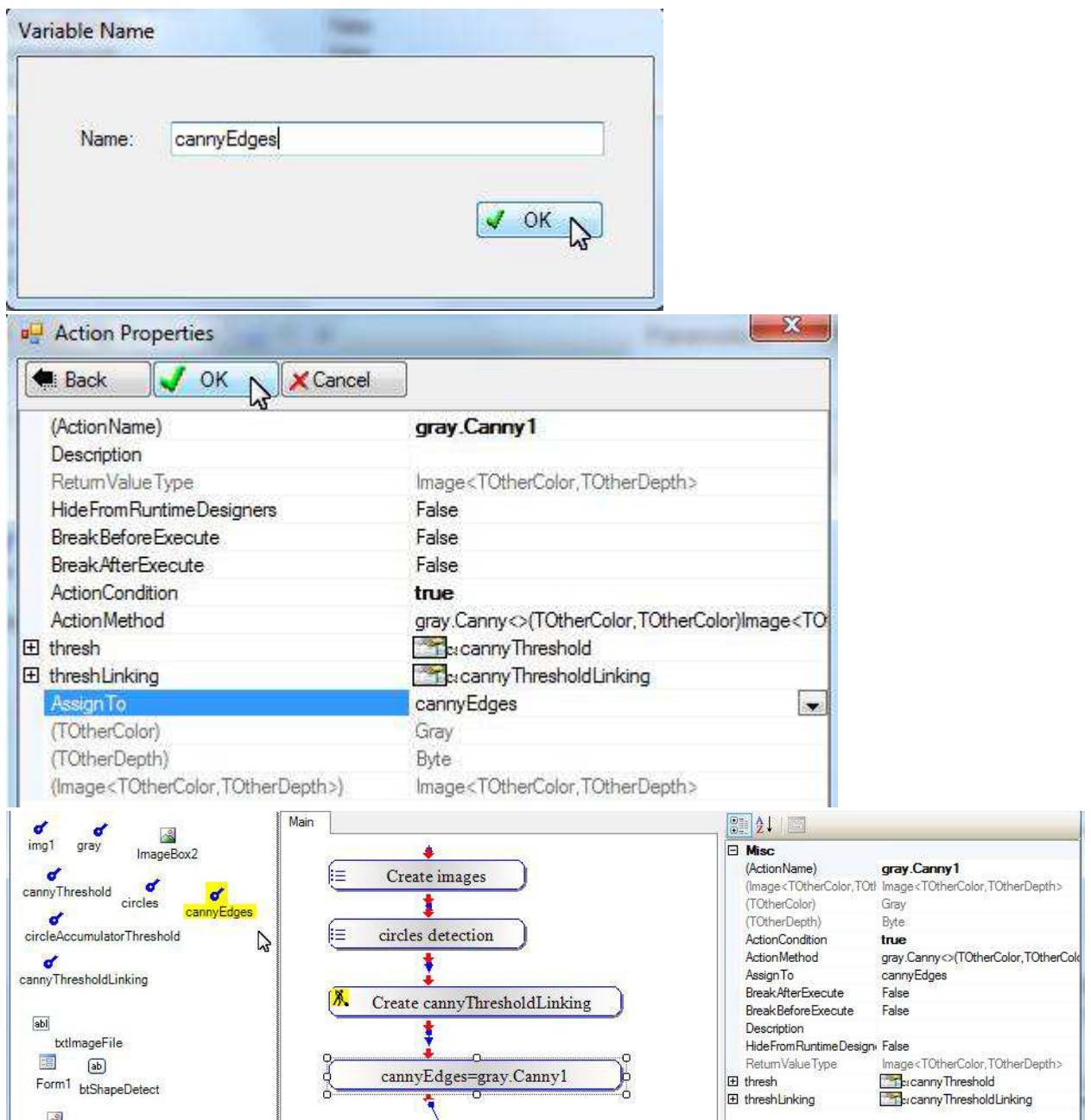
Object picker (Bottom):

- Buttons: Back, Next (highlighted), Exit, Add.
- Tree View:
 - Form1 from DrawingPage
 - Primary types
 - PerformShapeDetection
 - Actions
 - cannyThreshold
 - cannyThresholdLinking (selected)
 - circleAccumulatorThreshold
- Properties pane: gray, cannyThreshold, circleAccumulatorThreshold.

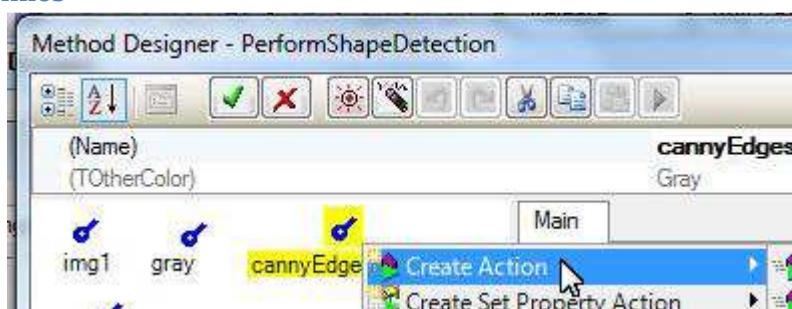
Action Properties (Bottom):

- Buttons: Back, OK (highlighted), Cancel.
- Properties:
 - (ActionName) gray.Canny1
 - Description
 - ReturnValueType Image<TOtherColor,TOtherDepth>
 - HideFromRuntimeDesigners False
 - BreakBeforeExecute False
 - BreakAfterExecute False
 - ActionCondition true
 - ActionMethod gray.Canny<>(TOtherColor,TOtherColor)Image<TOtherColor,TOtherDepth>
 - thresh cannyThreshold
 - threshLinking cannyThresholdLinking
 - AssignTo
 - New local variable (highlighted)
 - Select existing object

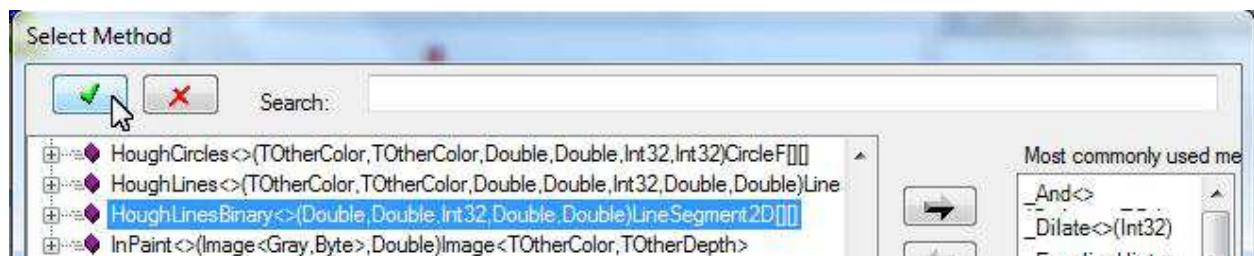
Use Generic Types and Native Libraries



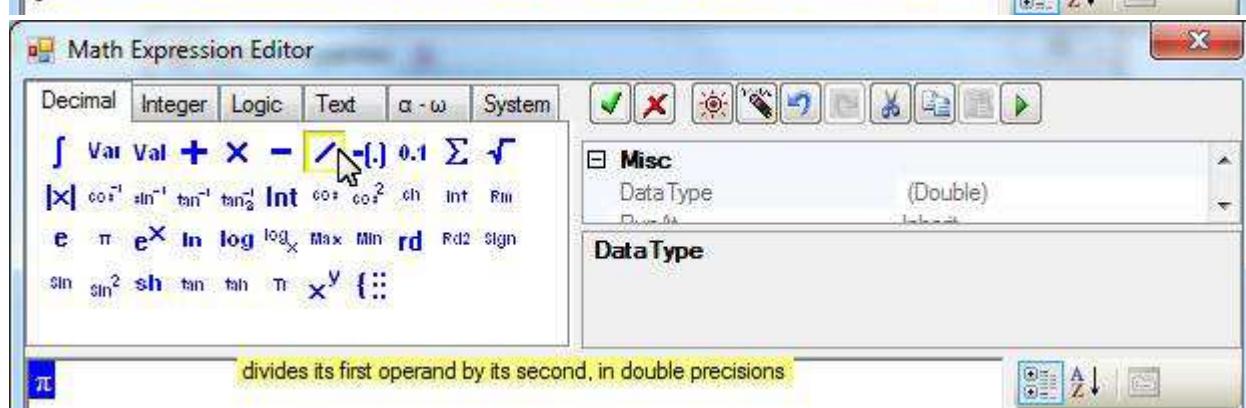
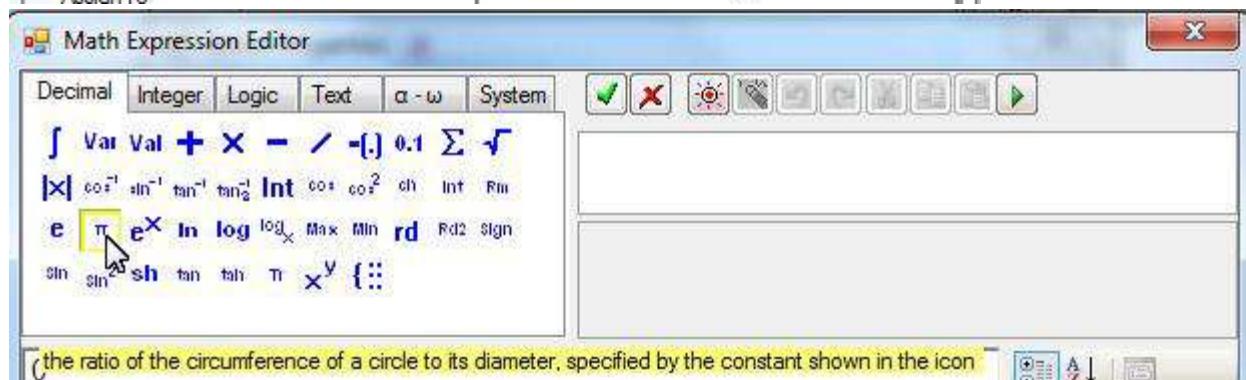
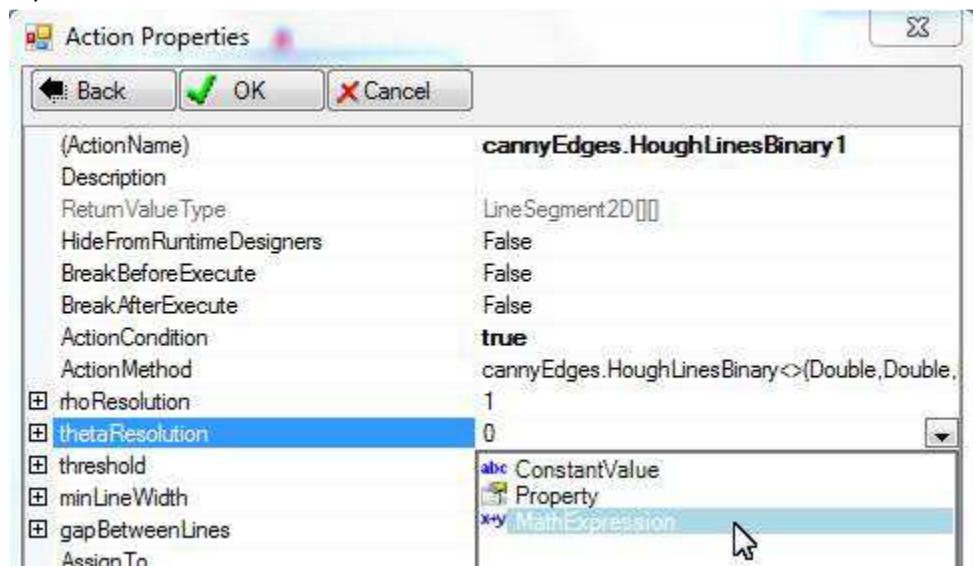
Detect lines



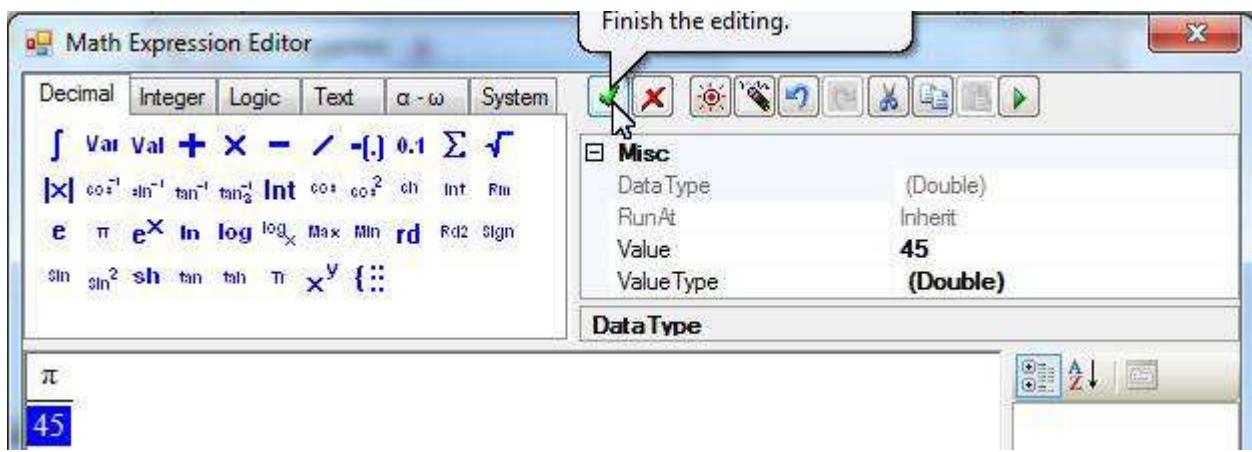
Use Generic Types and Native Libraries



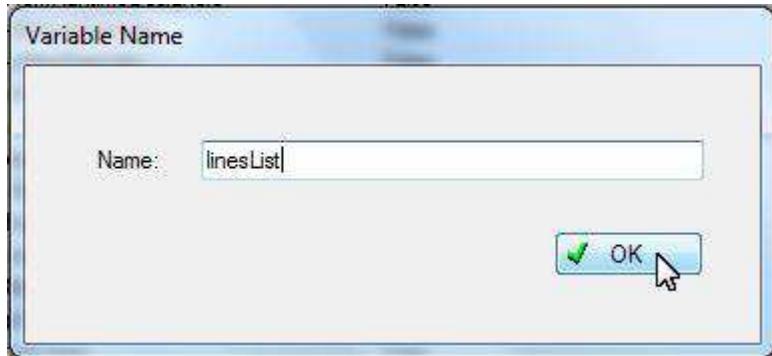
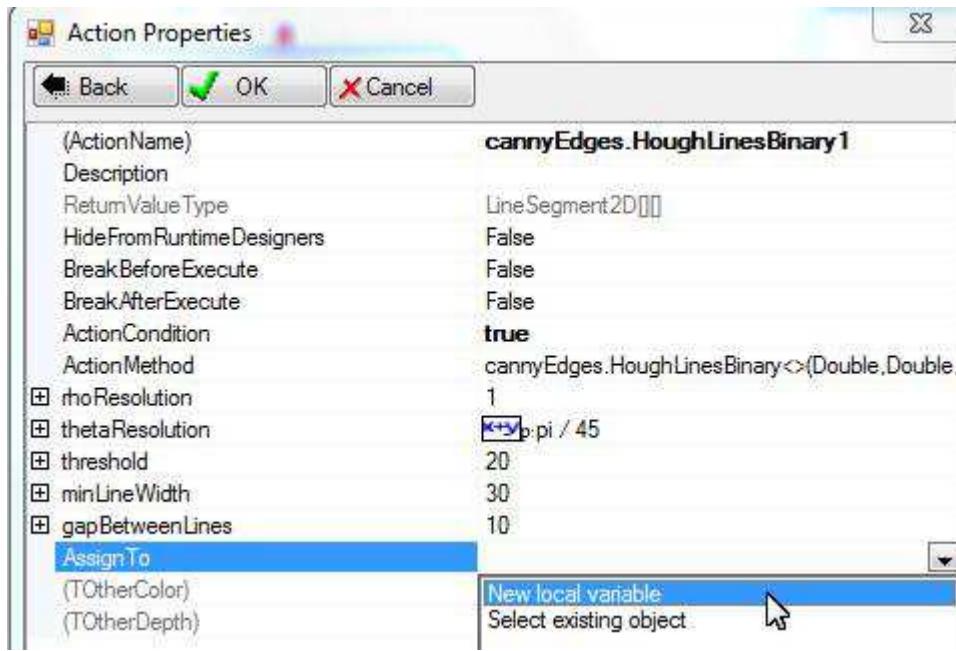
Provide action parameter values according to the Emgu tutorial. For thetaResolution, use an expression:



Use Generic Types and Native Libraries

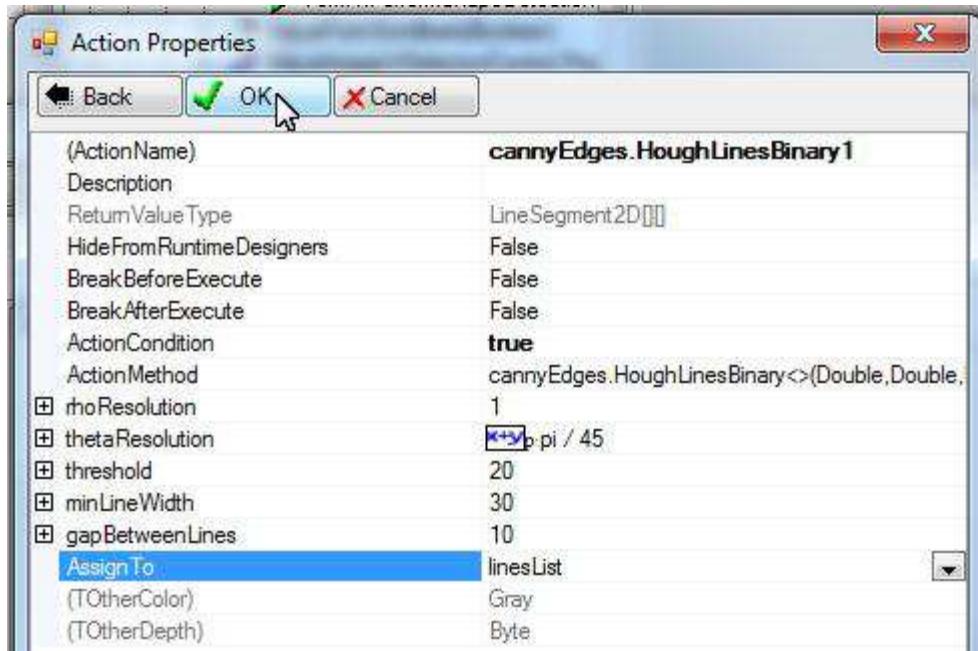


Use a new variable to get the results:

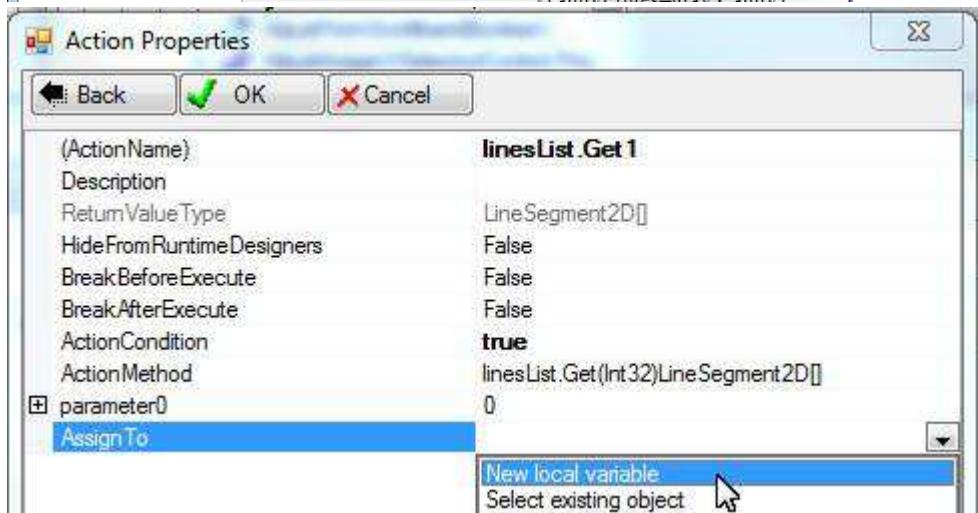
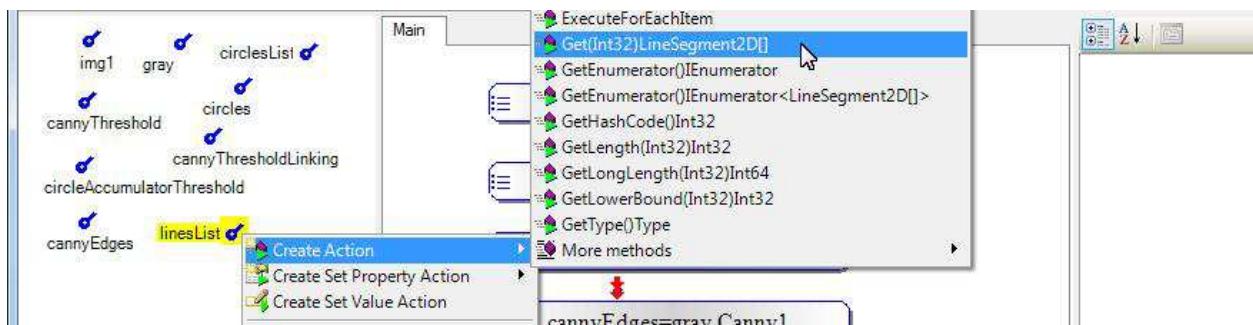


Click OK to finish creating the action:

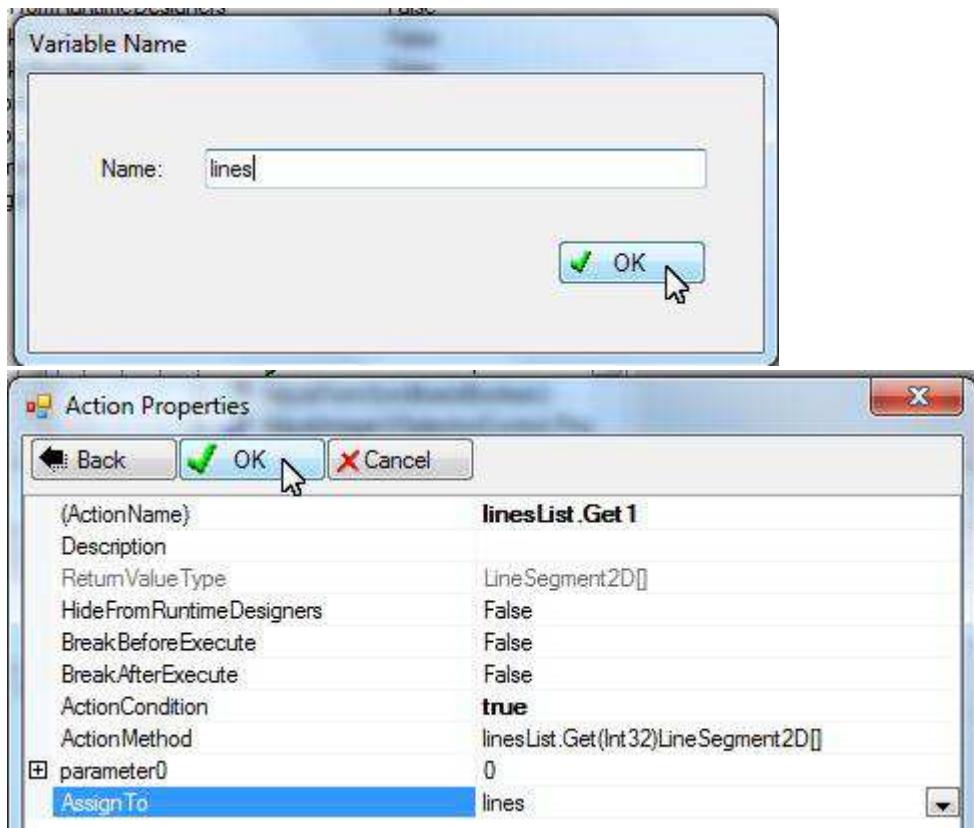
Use Generic Types and Native Libraries



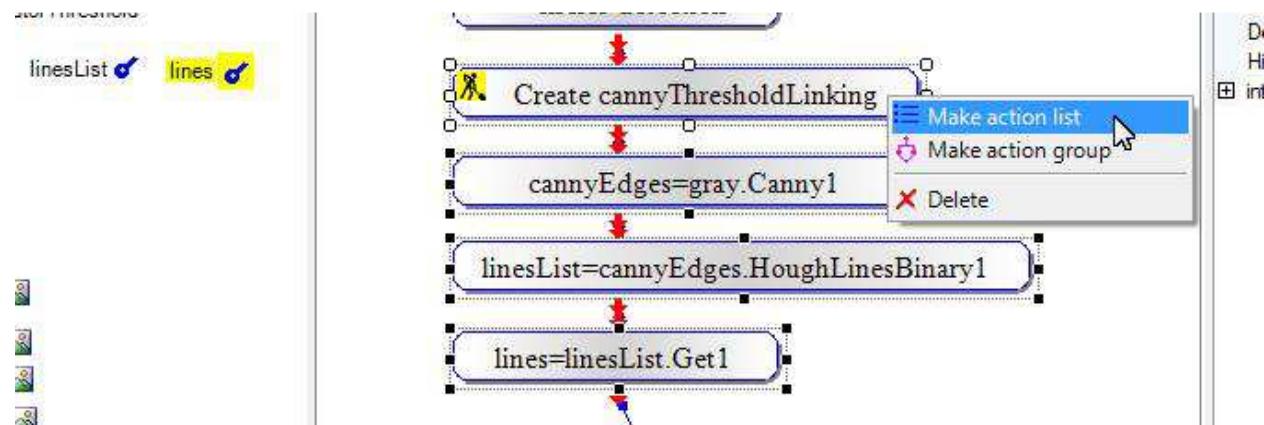
Get lines from the first item of "linesList":



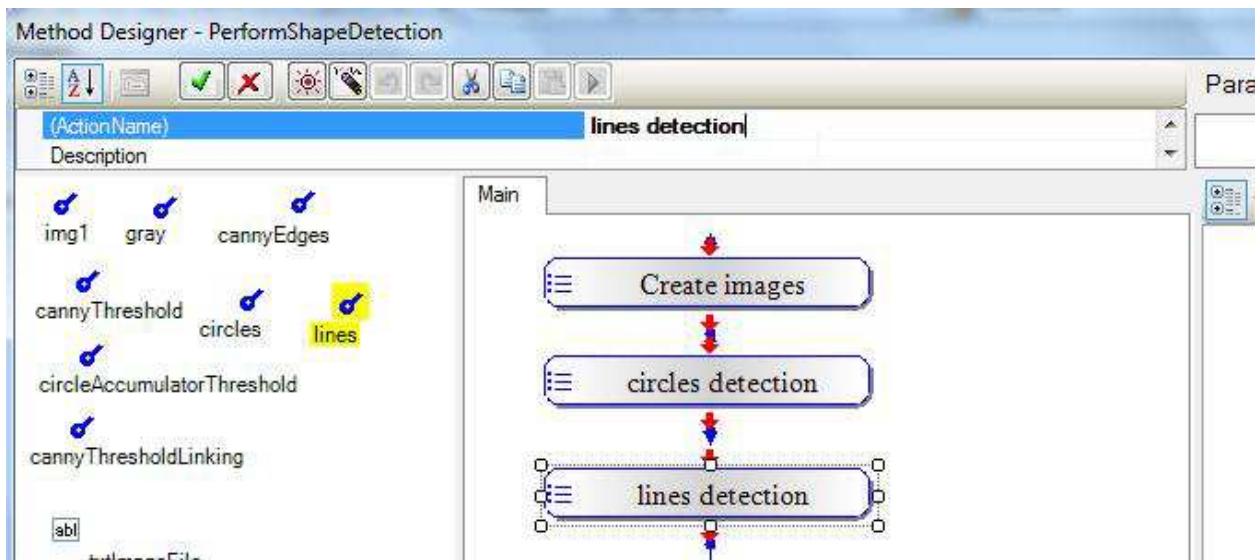
Use Generic Types and Native Libraries



We may group the above 4 action into a “lines detection” action list to make room for the editor:



Use Generic Types and Native Libraries

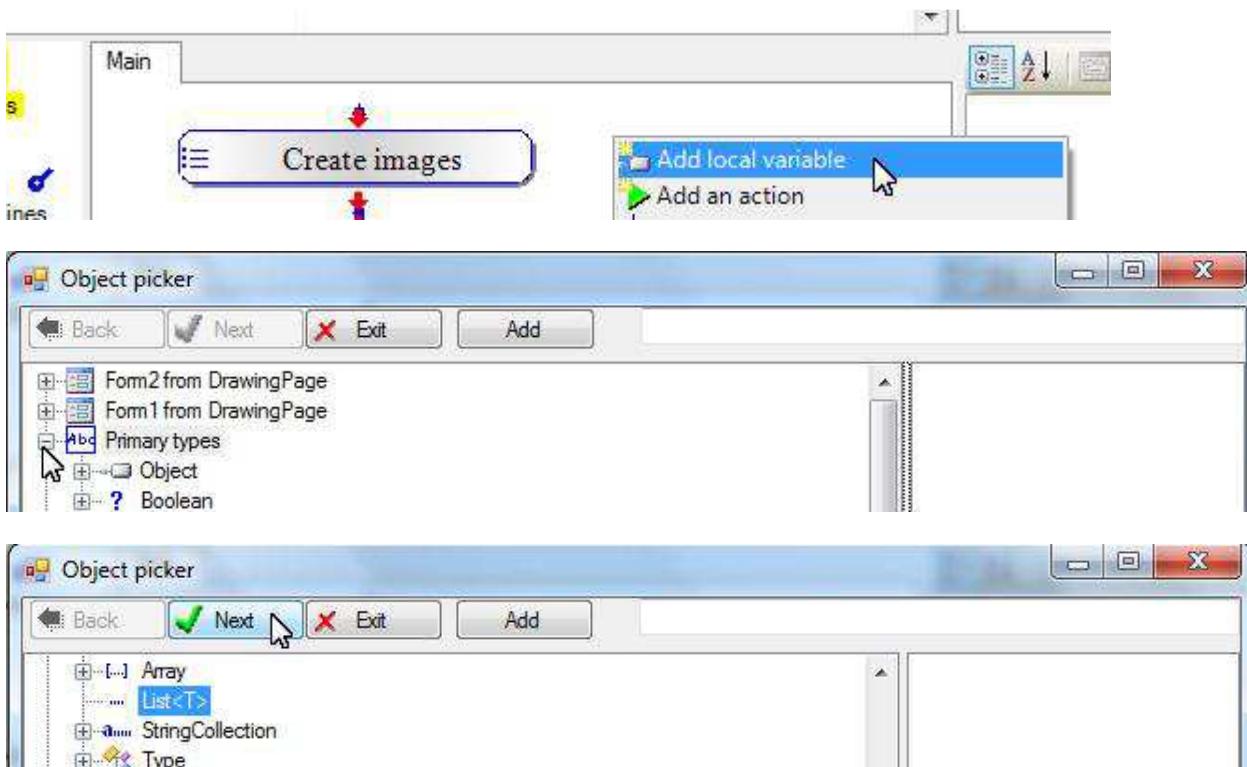


Triangles and Rectangles Detection

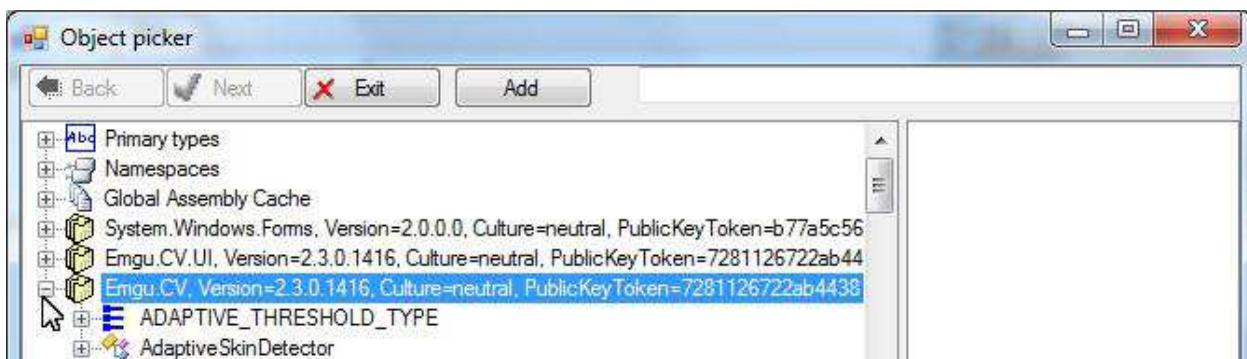
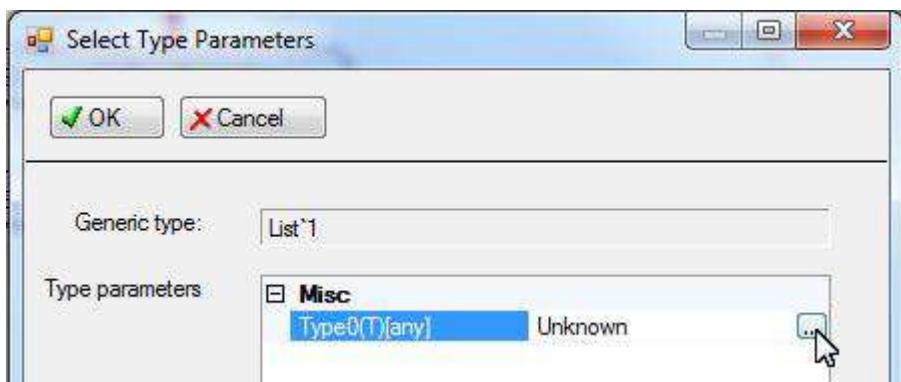
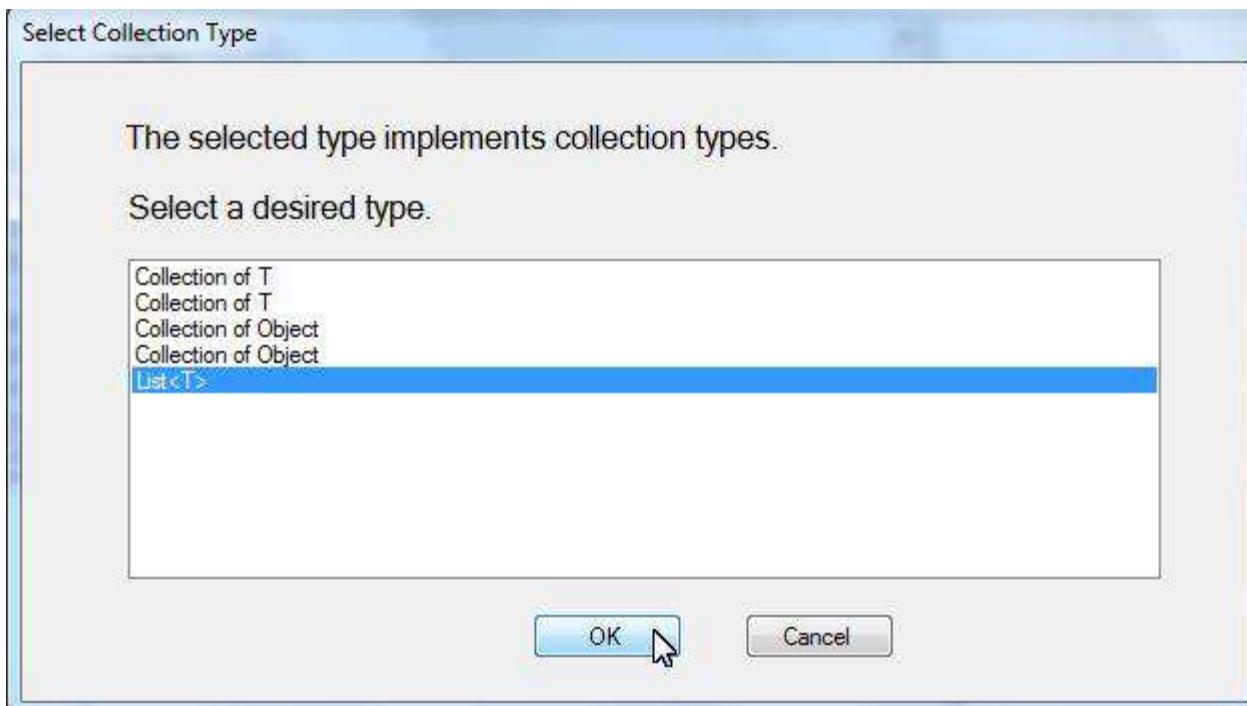
The "cannyEdges" variable has a `FindContours` method for finding contours. A triangle is a contour with 3 vertices and a rectangle is a contour with 4 vertices.

Create variable triangle list

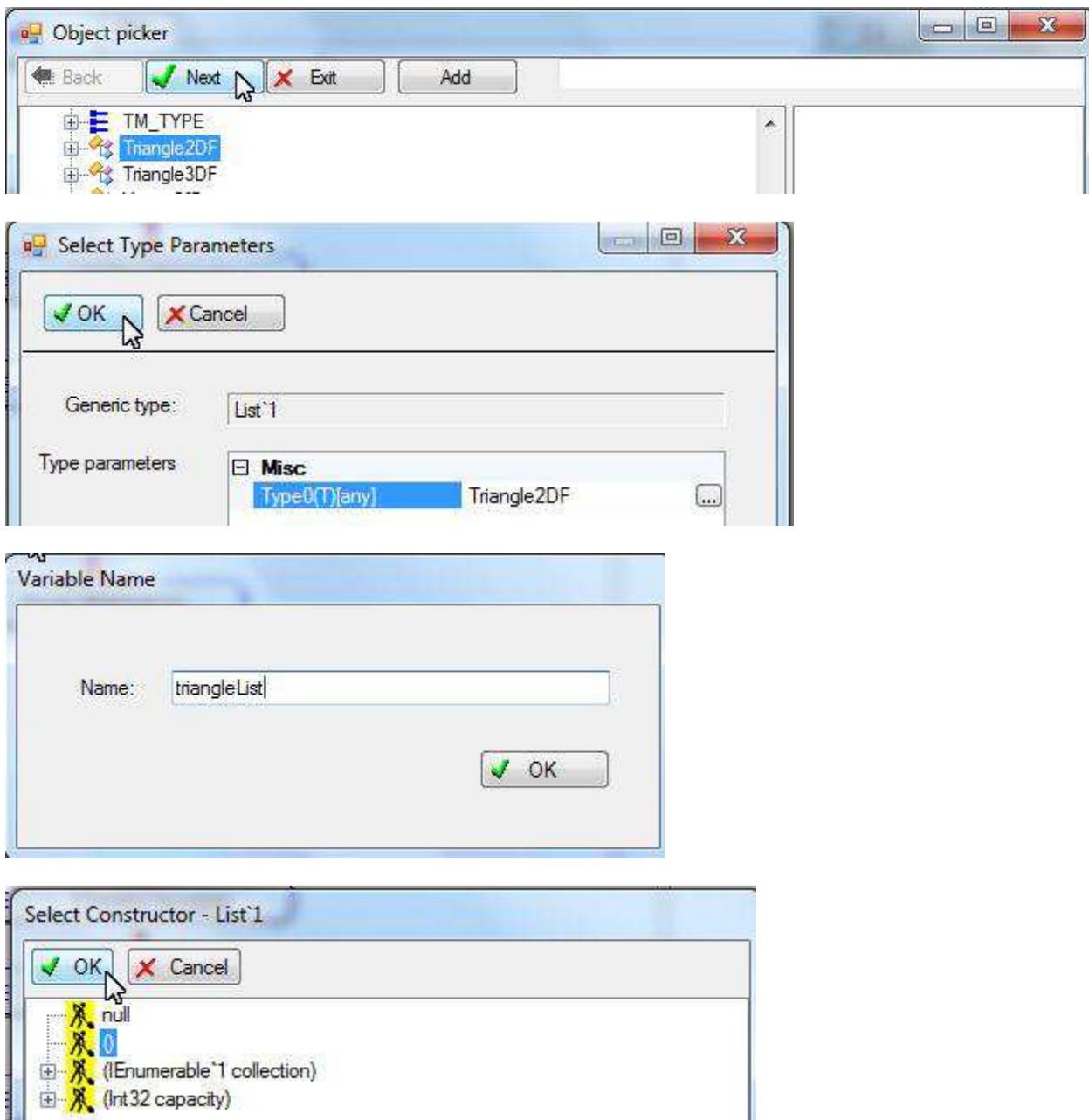
We use a variable for triangles.



Use Generic Types and Native Libraries



Use Generic Types and Native Libraries



Create variable for rectangle list

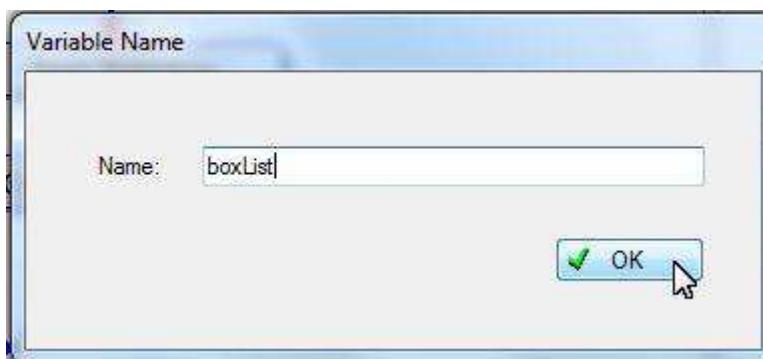
The steps are mostly the same as above. The different steps are listed below.

Select McvBox2D instead of Triangle2DF:



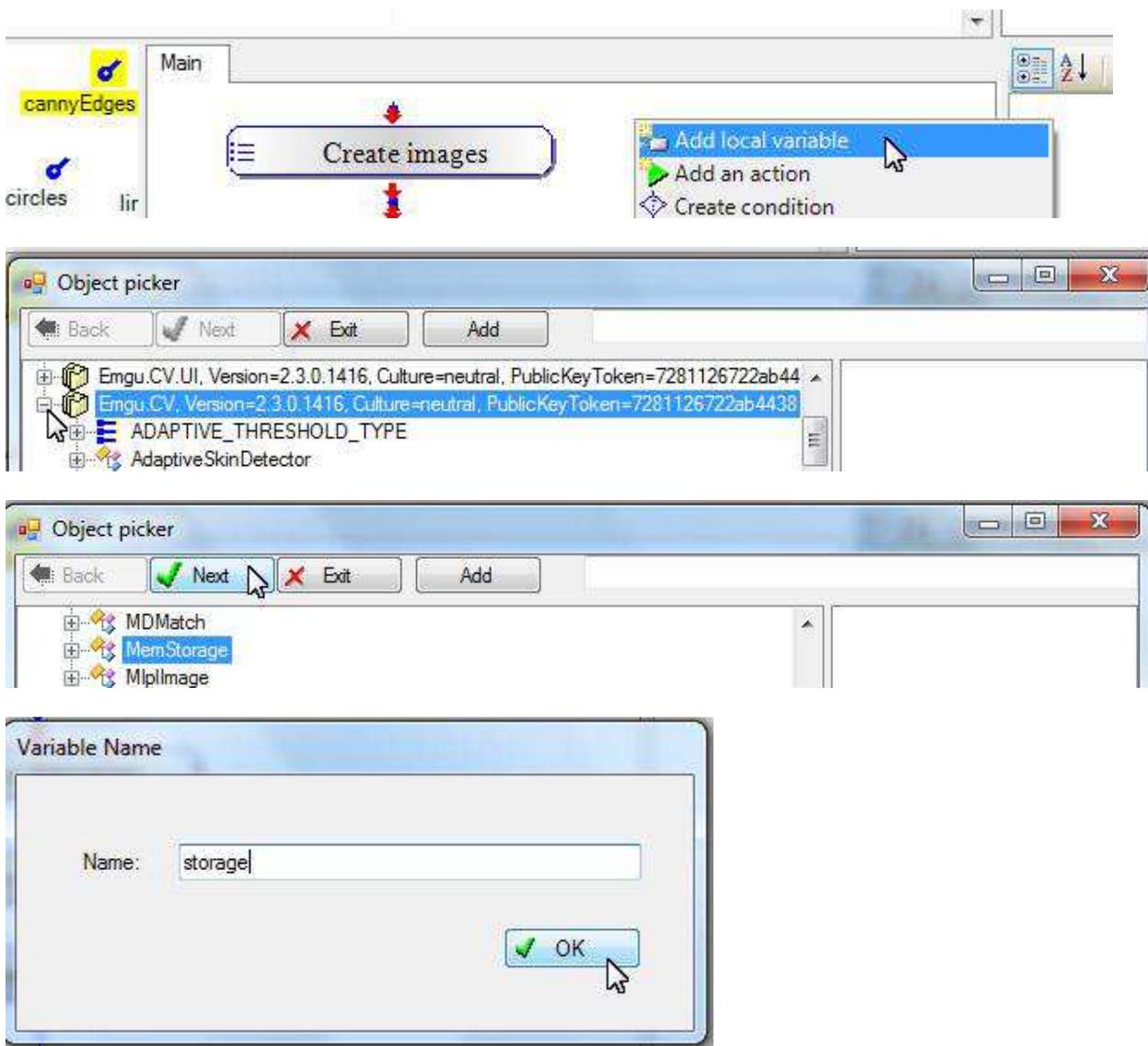
Use Generic Types and Native Libraries

Name it “boxList”:

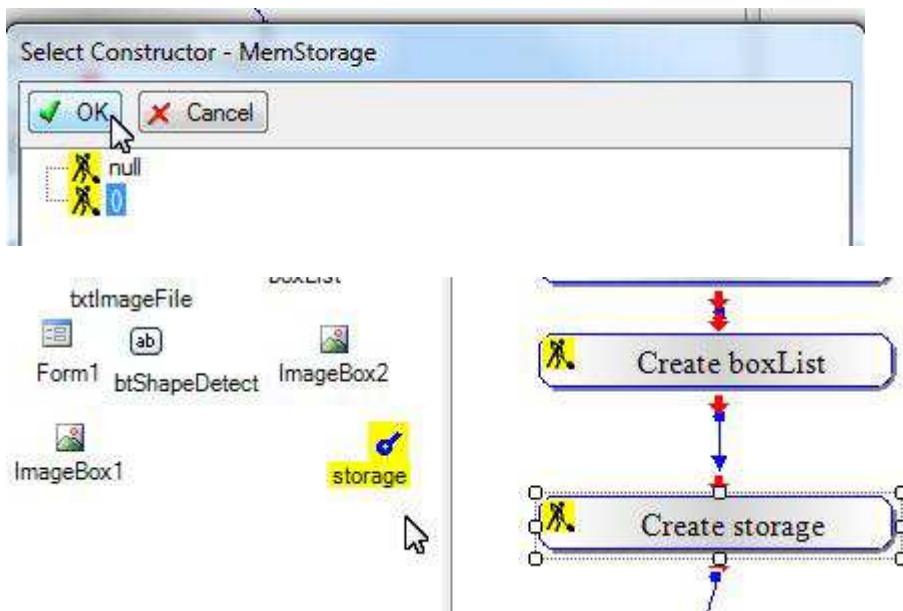


Create storage variable

A storage variable is needed for various actions.

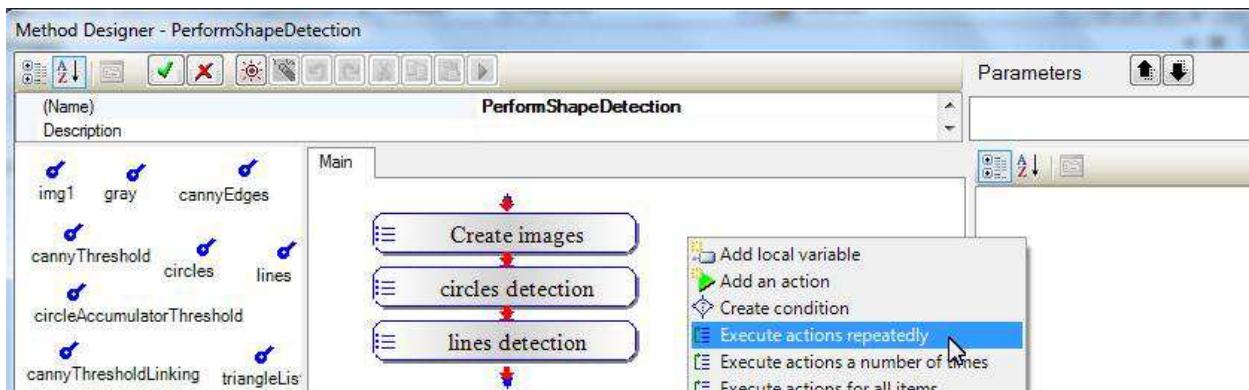


Use Generic Types and Native Libraries

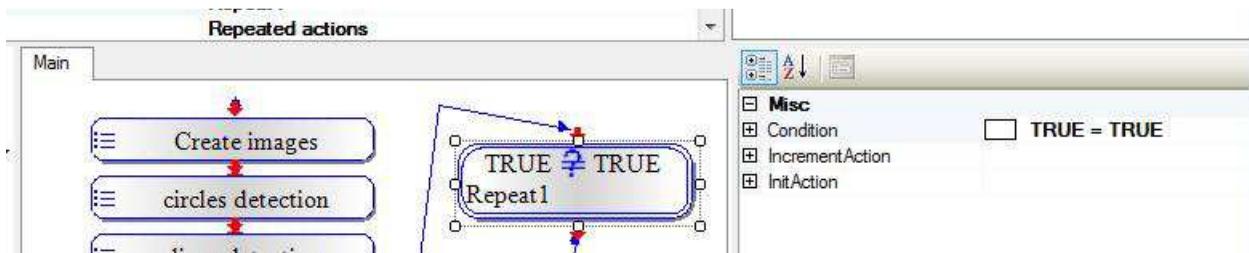


Go through shapes

We use a loop action to go through contours.

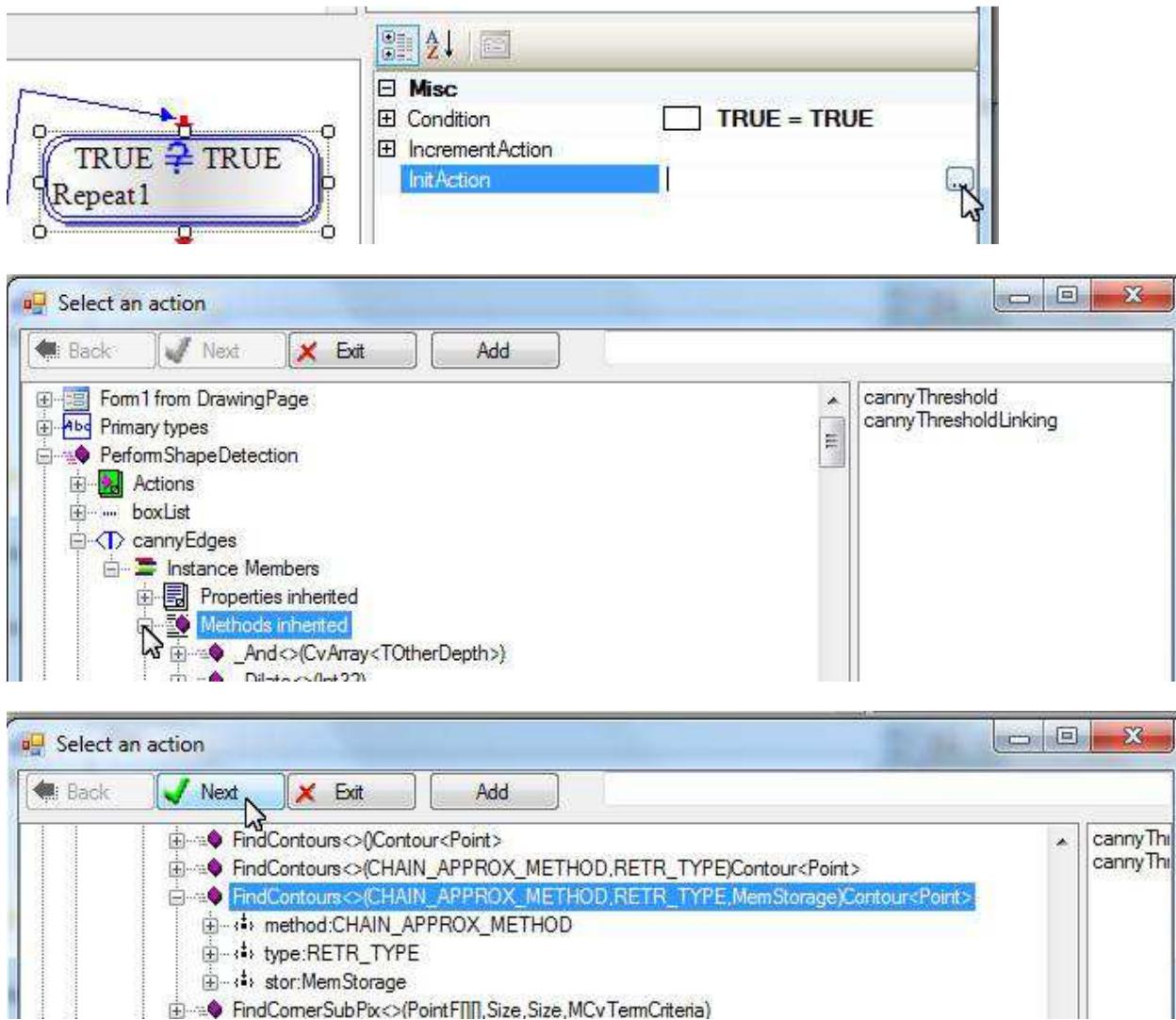


A loop action has an `InitAction` property indicating the initialization action executed before the loop starts. It has an `IncrementAction` property indicating the action to be executed after each loop. It has a `Condition` property indicating the condition to break the loop. If the `Condition` evaluates to `False` then the loop breaks.

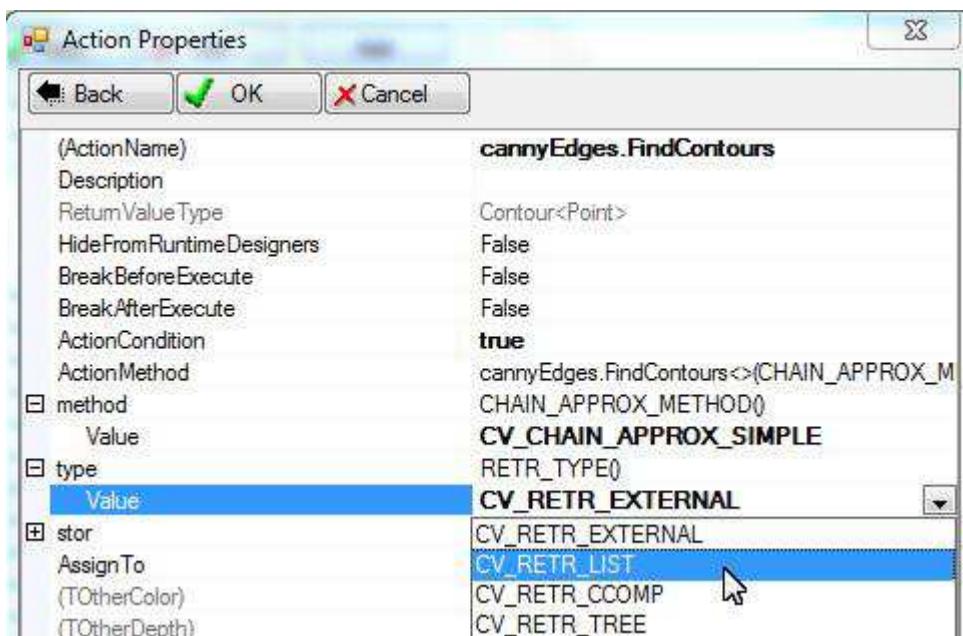
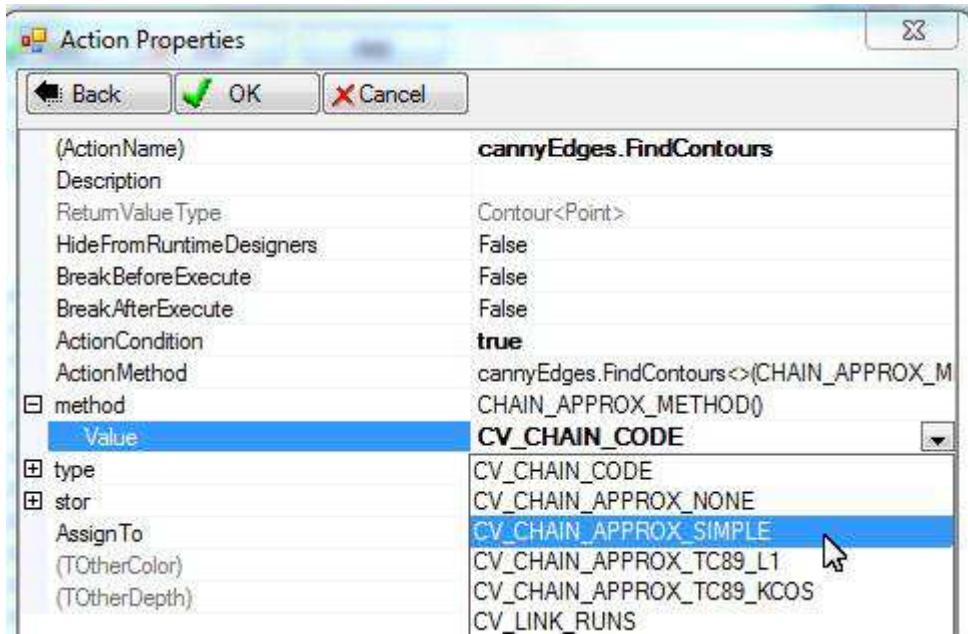


Setup initialization action

For this sample, the initialization action is to use the FindContours method of “cannyEdges” to get all shapes so that the loop action may go through each shape. This method returns the first shape. From the first shape we can get the next shape.



Use Generic Types and Native Libraries



Use Generic Types and Native Libraries

The top screenshot shows the "Action Properties" dialog for the action "cannyEdges.FindContours". The properties listed are:

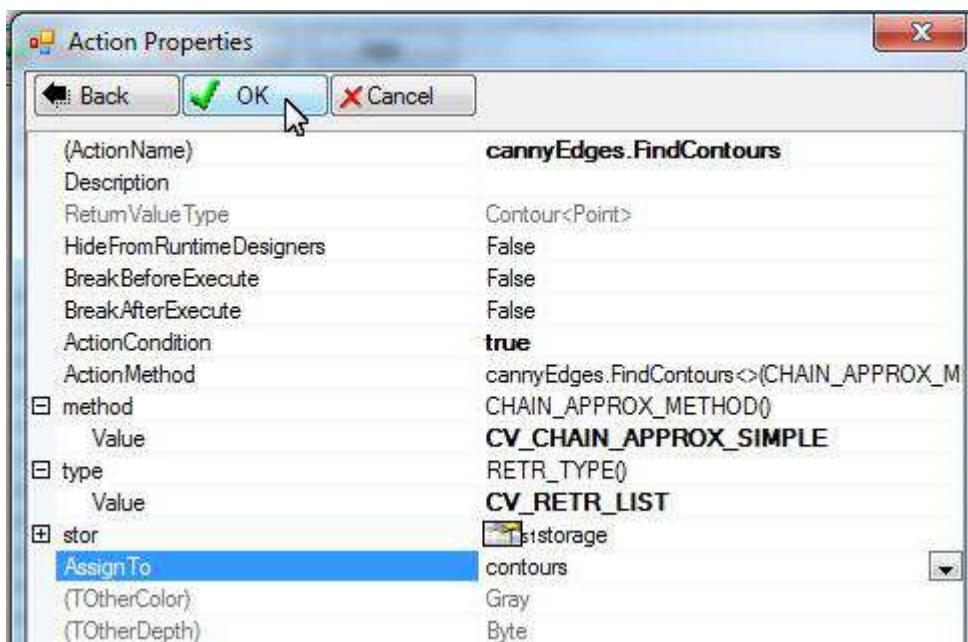
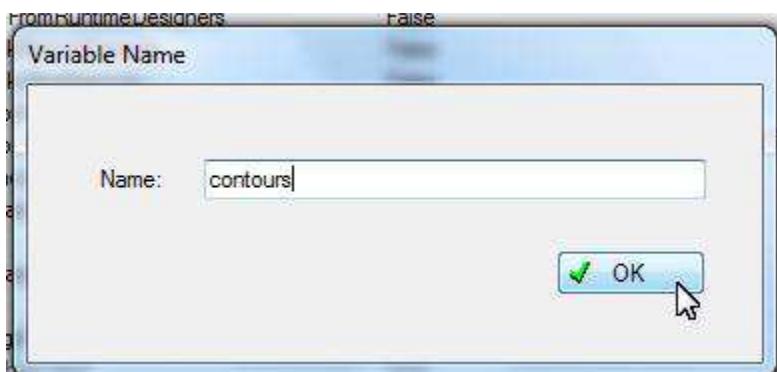
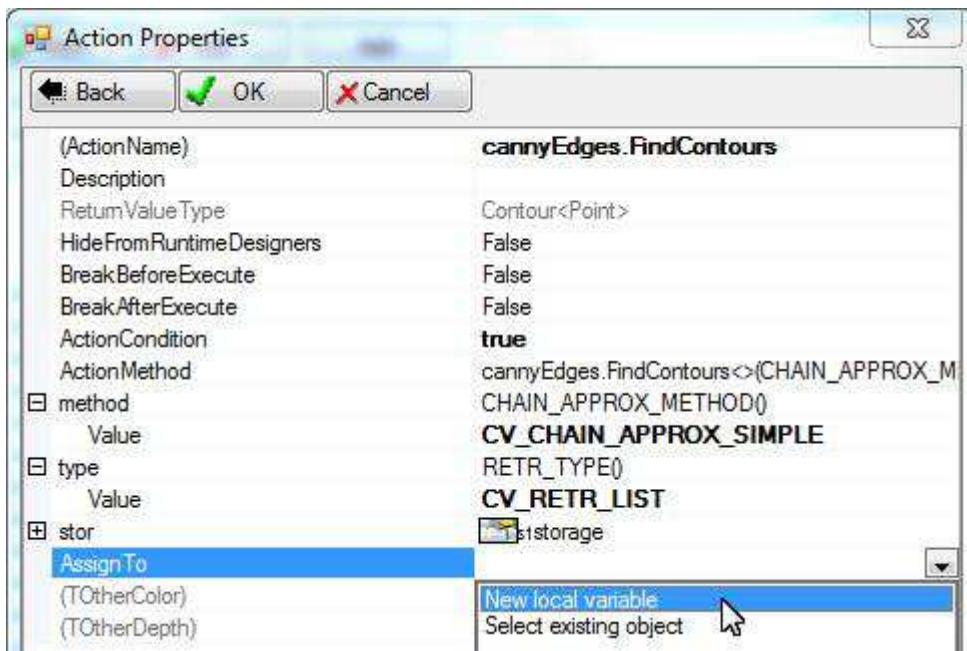
- (ActionName) cannyEdges.FindContours
- Description
- ReturnValueType Contour<Point>
- HideFromRuntimeDesigners False
- BreakBeforeExecute False
- BreakAfterExecute False
- ActionCondition true
- ActionMethod cannyEdges.FindContours<>(CHAIN_APPROX_METHOD)
- method Value CV_CHAIN_APPROX_SIMPLE
- type Value RETR_TYPE
- stor Value MemStorage()

The bottom screenshot shows the "Object picker" dialog. The left pane lists objects under "PerformShapeDetection":

- Actions
- boxList
- cannyEdges
- cannyThreshold
- cannyThresholdLinking
- circleAccumulatorThreshold
- circles
- gray
- img1
- lines
- storage
- triangleList

The right pane shows two selected items: "cannyThreshold" and "cannyThresholdLinking".

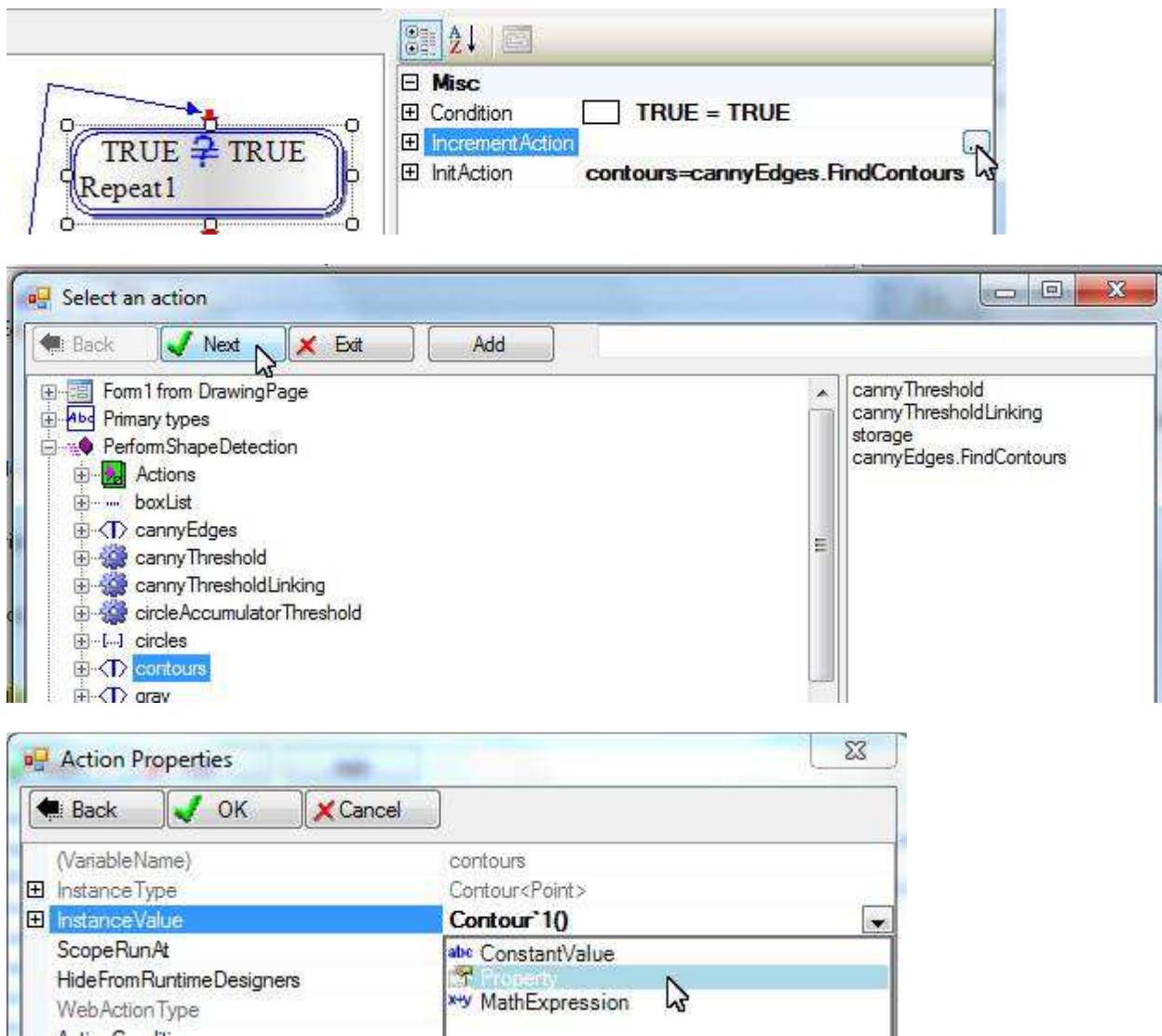
Use Generic Types and Native Libraries



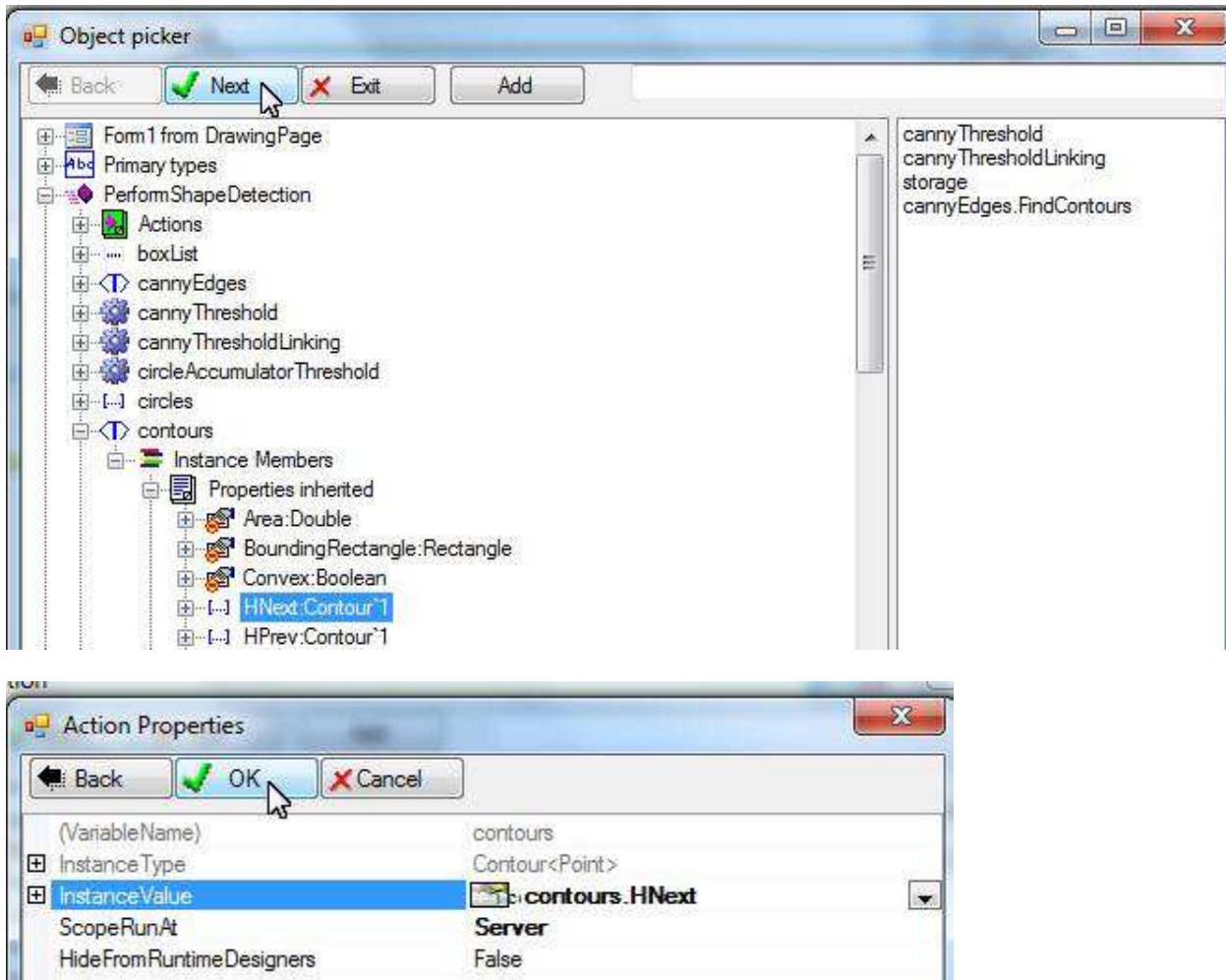
Use Generic Types and Native Libraries

Setup increment action

The “contours” has a HNext property for getting the next contours. We may use it to create an action to be executed after each loop.



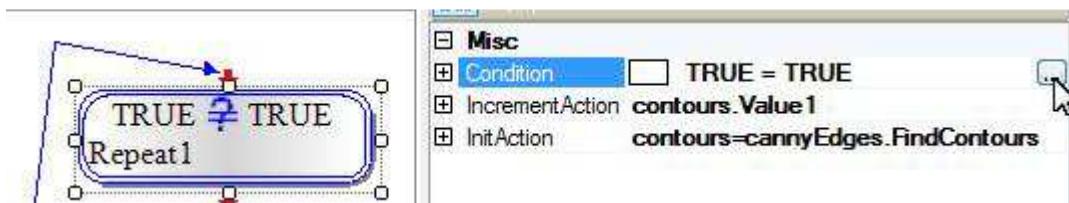
Use Generic Types and Native Libraries



After executing this action, the “contours” variable will become the next shape.

Setup loop condition

The “contours” variable represents the current shape. If there are no more shapes then “contours” variable will be null. So, the loop condition should be that the “contours” variable is not null.

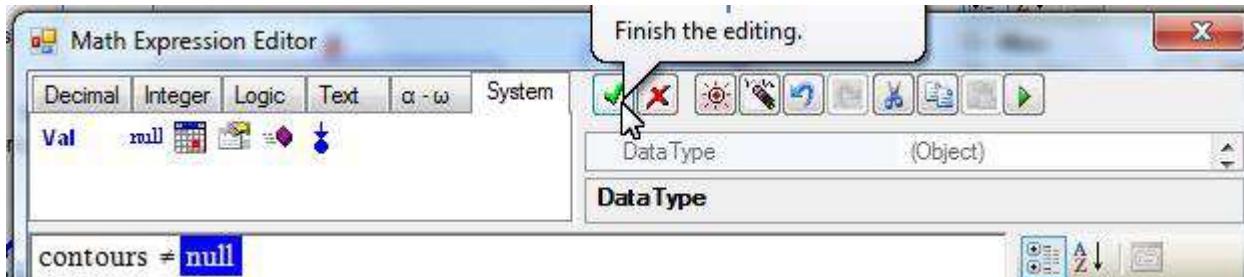


Use Generic Types and Native Libraries

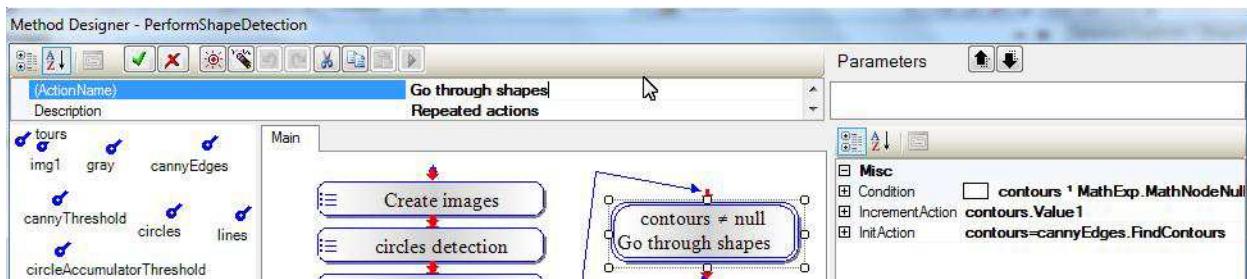
The figure consists of four screenshots of a software interface, likely a graphical programming language or a visual editor for robotics.

- Screenshot 1:** Shows the "Math Expression Editor" window. The toolbar includes tabs for Decimal, Integer, Logic, Text, α - ω, and System. The Logic tab is selected. A status bar at the bottom shows "Logic 'not equal' operator".
- Screenshot 2:** Shows the "Math Expression Editor" window. The Logic tab is selected. The status bar at the bottom shows "A property".
- Screenshot 3:** Shows the "Object picker" window. It displays a tree view of objects under "PerformShapeDetection" and a list of available actions on the right, including "cannyThreshold", "cannyThresholdLinking", "storage", "cannyEdges.FindContours", "contours.HNext", and "LimnorDesigner.Action.ActionAssign".
- Screenshot 4:** Shows the "Math Expression Editor" window. The Logic tab is selected. The status bar at the bottom shows "represent a null value."

Use Generic Types and Native Libraries

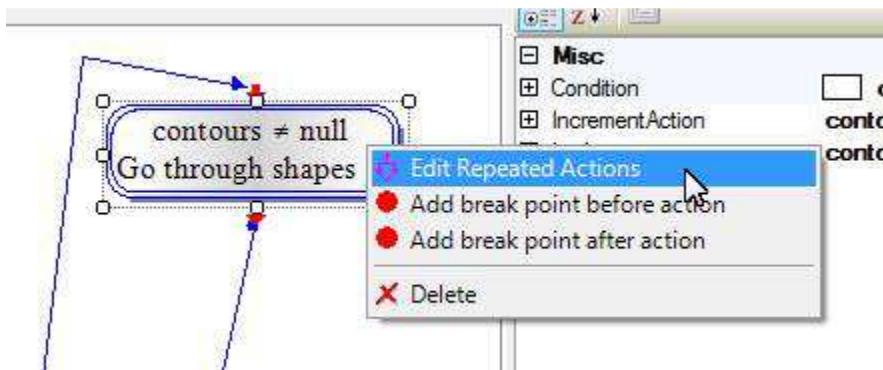


We may give a description to the loop action:

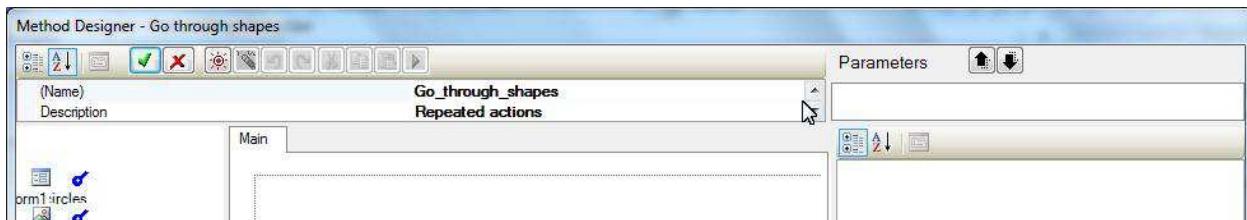


Process Each Shape

To process each shape, right-click the loop action, choose "Edit Repeated Actions":



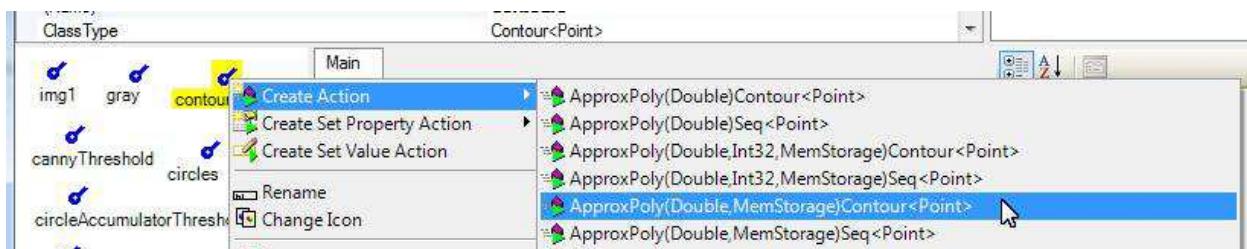
A new Method Editor Dialogue box appears for us to add actions to it:



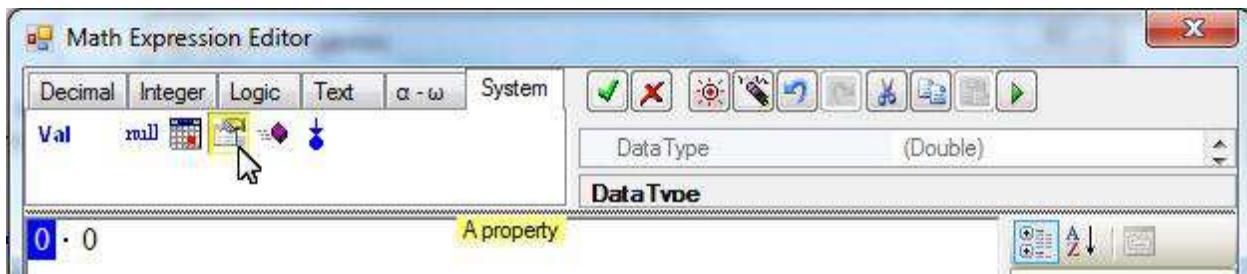
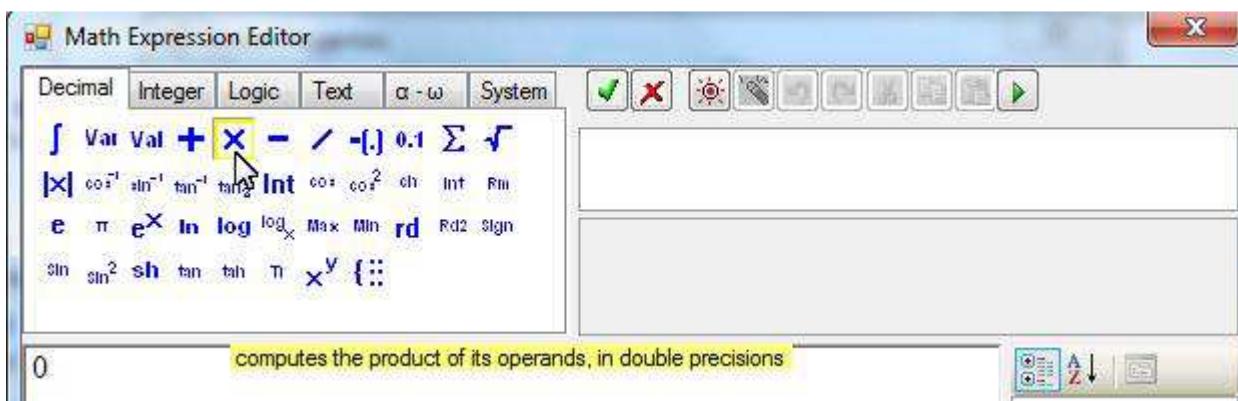
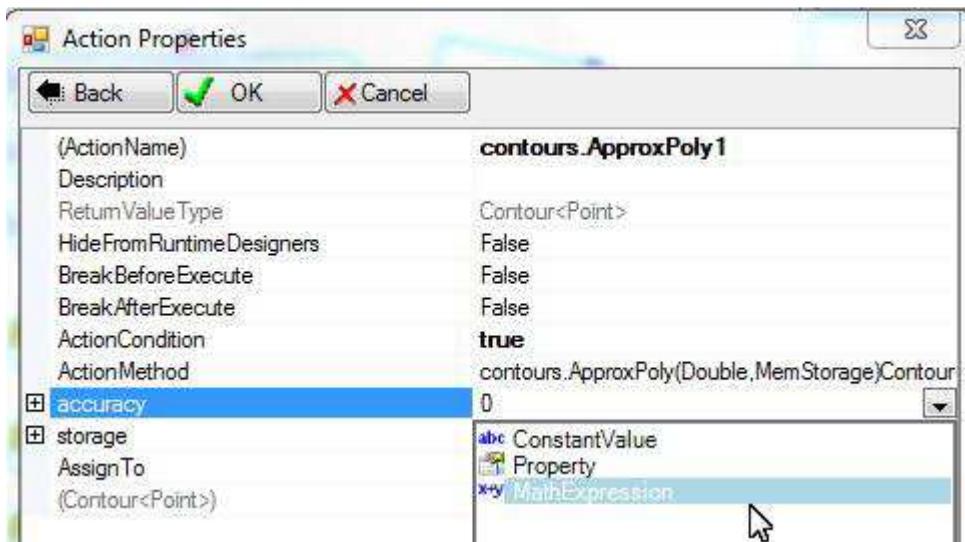
Get current contour

"contours" has a `ApproxPoly` method for getting the current contour.

Use Generic Types and Native Libraries



Set the “accuracy” to 5% of the contour perimeter:



Use Generic Types and Native Libraries

The screenshot shows two windows from the Limnor Designer software:

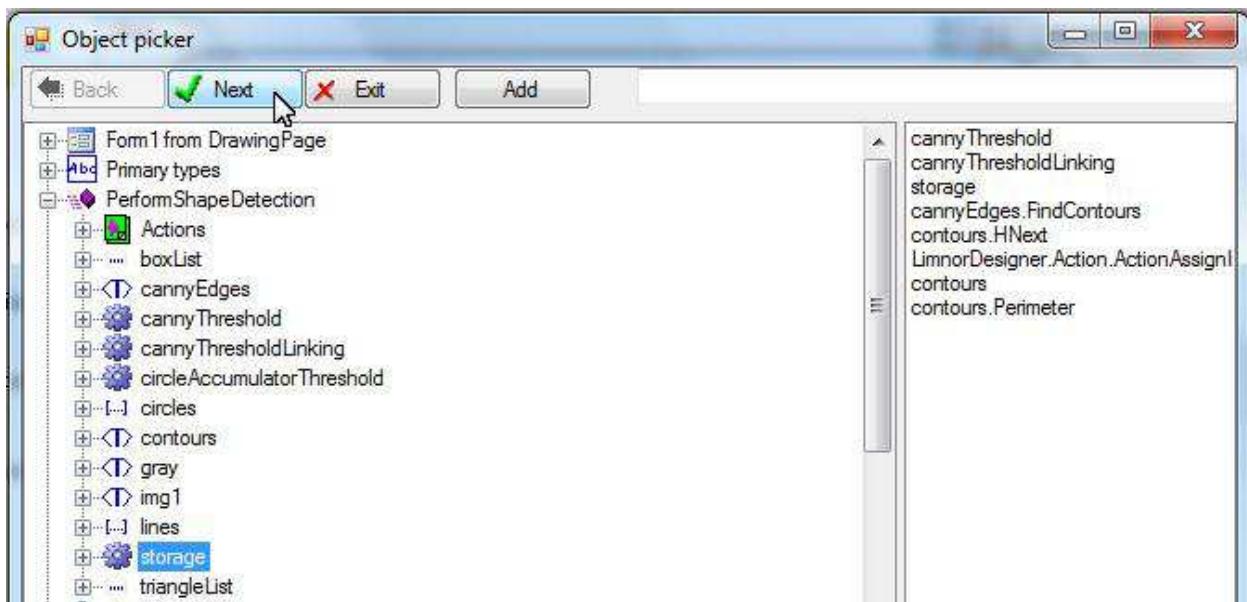
- Object picker window:** Shows a tree view of actions and their members. Under "PerformShapeDetection", the "contours" action is expanded, showing its instance members: Area:Double, BoundingRectangle:Rectangle, Convex:Boolean, HNext:Contour'1, HPrev:Contour'1, MCvContour:MCvContour, MCvSeq:MCvSeq, and Perimeter Double. The "Perimeter Double" member is highlighted with a blue selection bar.
- Math Expression Editor window:** Shows the expression "contours.Perimeter * 0.05". The "Finish the editing." button is visible at the top right. Below it, the "DataType" dropdown is set to "(Double)".

Set the storage to the variable "storage":

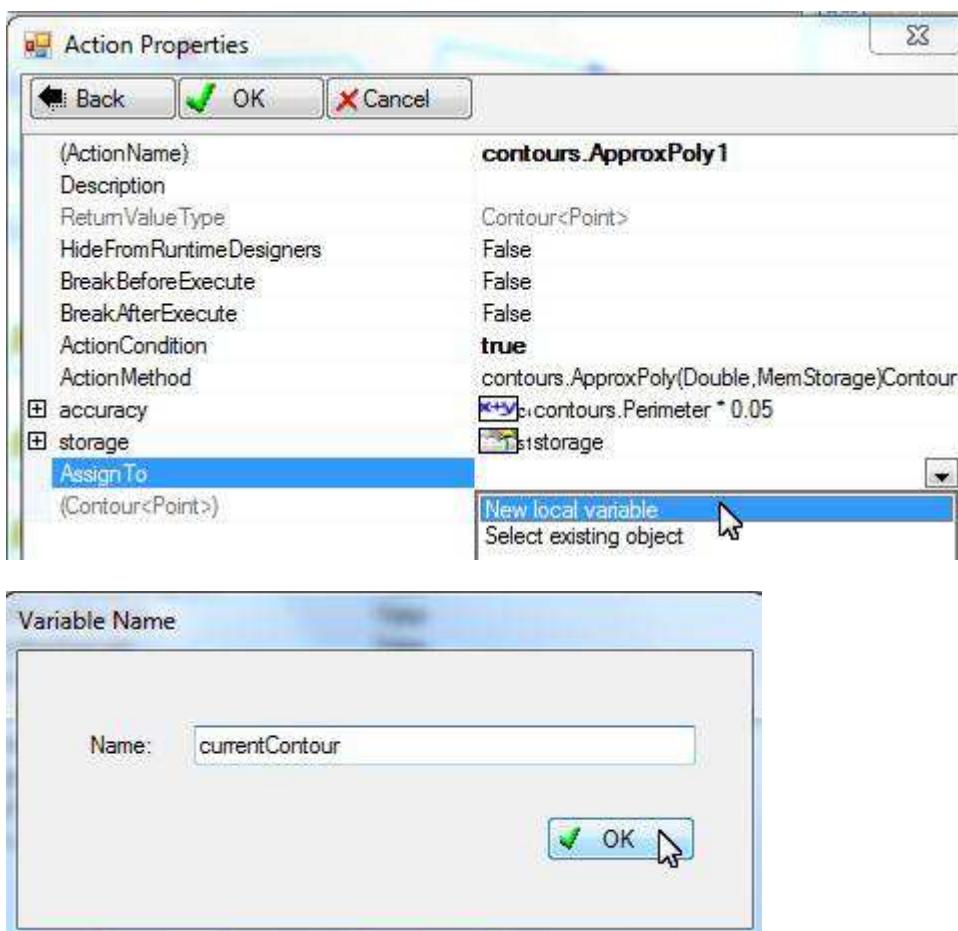
The screenshot shows the "Action Properties" dialog for the "contours.ApproxPoly1" action:

(ActionName)	contours.ApproxPoly1
Description	
ReturnValueType	Contour<Point>
HideFromRuntimeDesigners	False
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	true
ActionMethod	contours.ApproxPoly(Double, MemStorage)Contour
accuracy	$\text{contours.Perimeter} * 0.05$
storage	MemStorage()
AssignTo	abc ConstantValue Property MathExpression
(Contour<Point>)	

Use Generic Types and Native Libraries

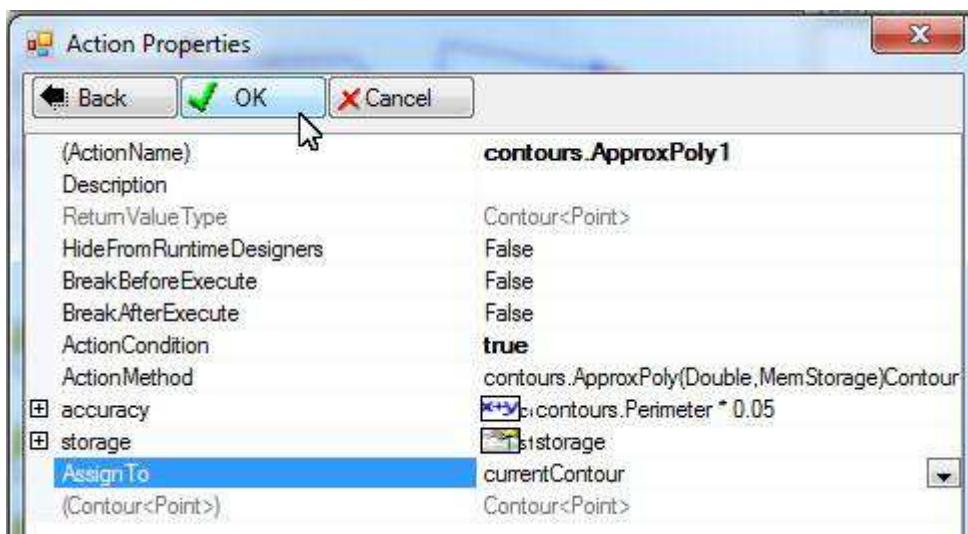


Assign the result to a new variable:

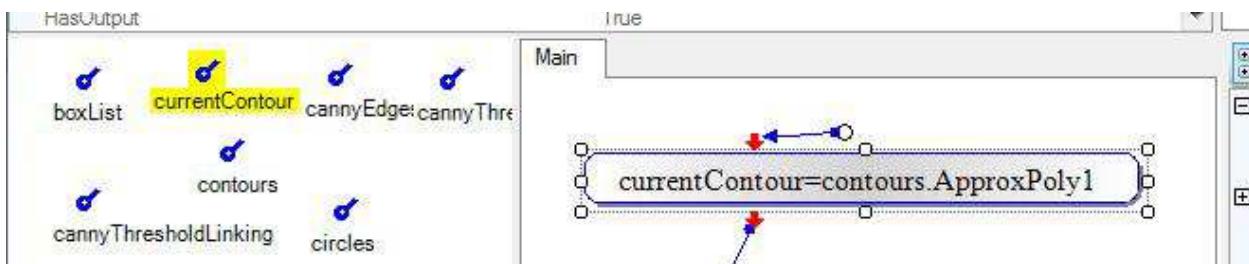


Click OK to finish creating the action.

Use Generic Types and Native Libraries

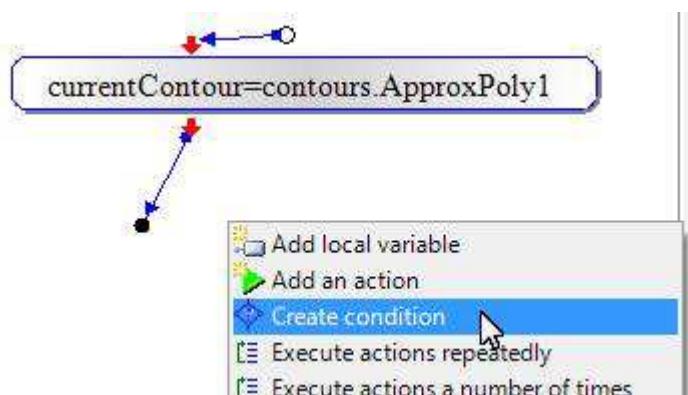


The new variable and action appear:

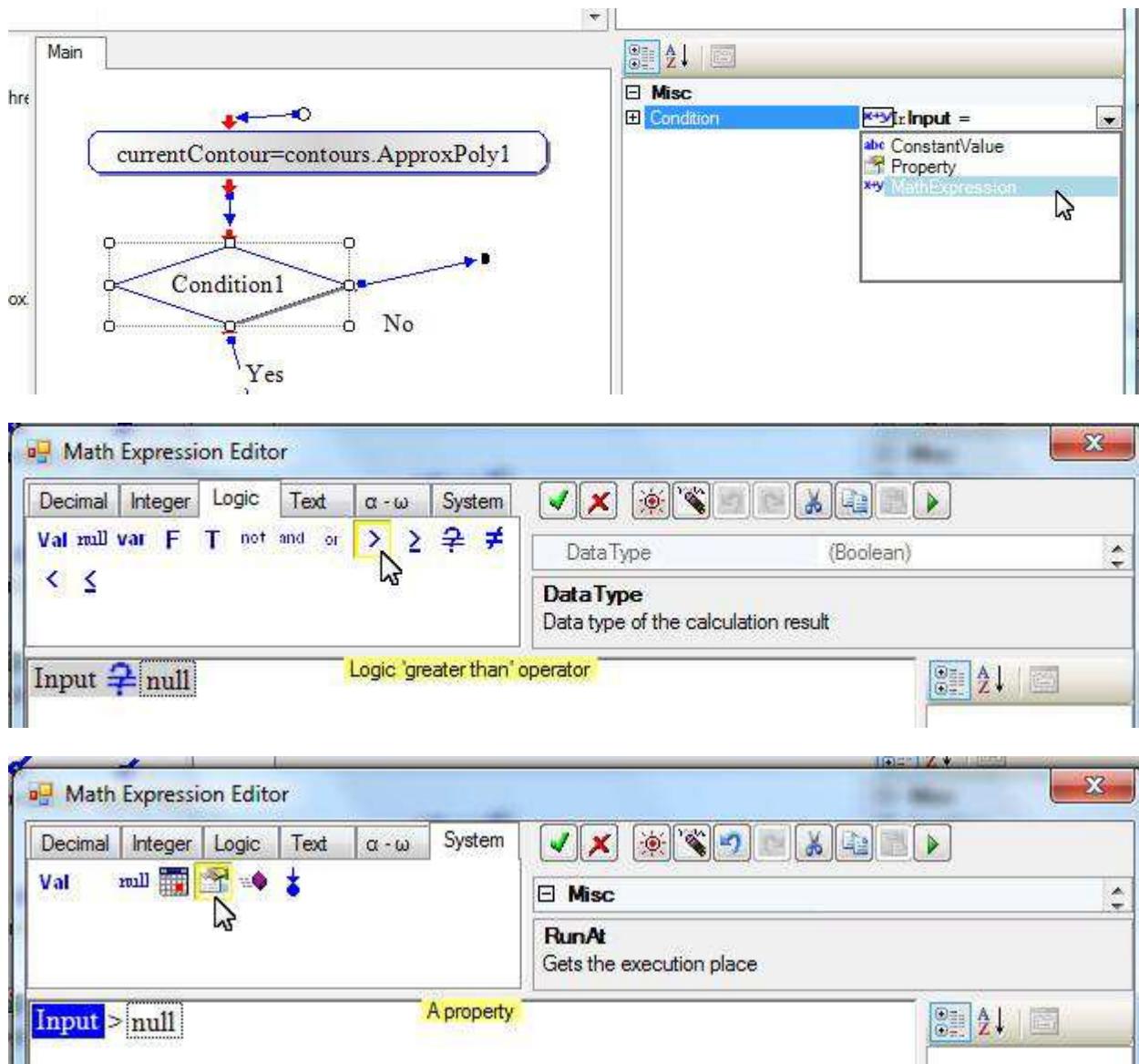


Check contour size

Only a contour's Area property is larger than 250 will be processed.



Use Generic Types and Native Libraries



Use Generic Types and Native Libraries

The image consists of three vertically stacked screenshots from a software application, likely a graphical programming environment.

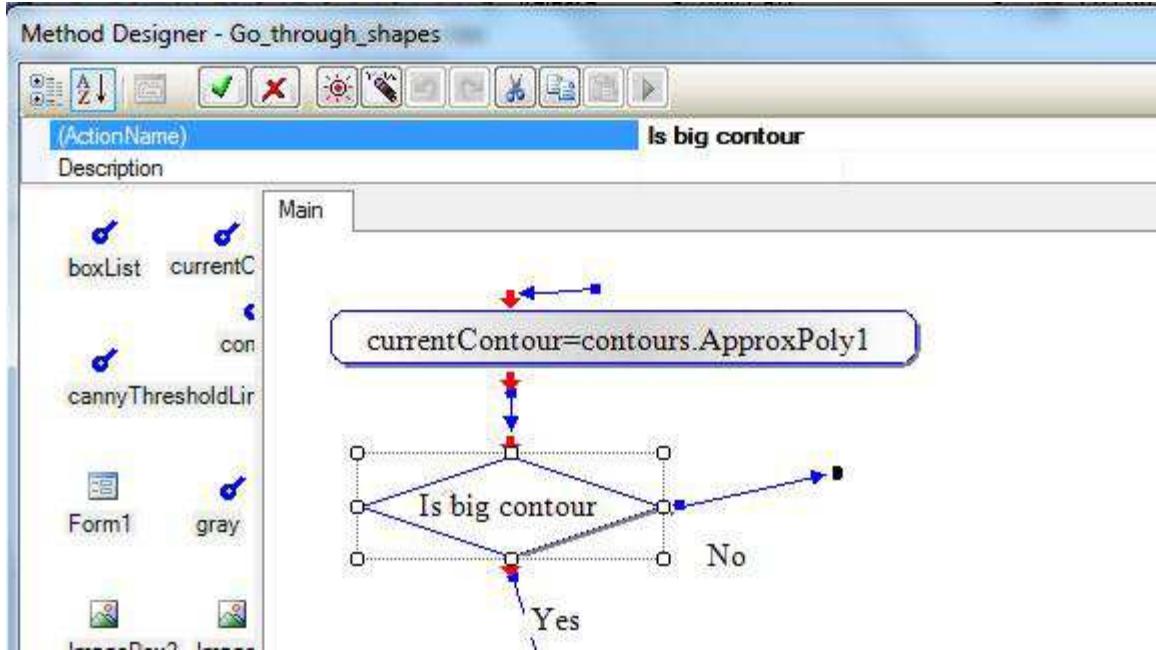
Top Screenshot: A window titled "Object picker". It shows a tree view of objects under "PerformShapeDetection". The "Area.Double" node is selected. On the right, there is a list of actions: "cannyEdges.FindContours", "LimnorDesigner.Action.ActionAssign1", and "LimnorDesigner.Action.ActionAssign2".

Middle Screenshot: A window titled "Math Expression Editor". The expression being edited is "currentContour.Area > 250". The "RunAt" dropdown is set to "Inherit". The status bar at the bottom says "currentContour.Area > 250 a constant value, the value type may be changed."

Bottom Screenshot: A window titled "Math Expression Editor" showing the same expression "currentContour.Area > 250". The "RunAt" dropdown is set to "Inherit". The status bar at the bottom says "Finish the editing." A tooltip "Finish the editing." is visible near the top right of the editor window.

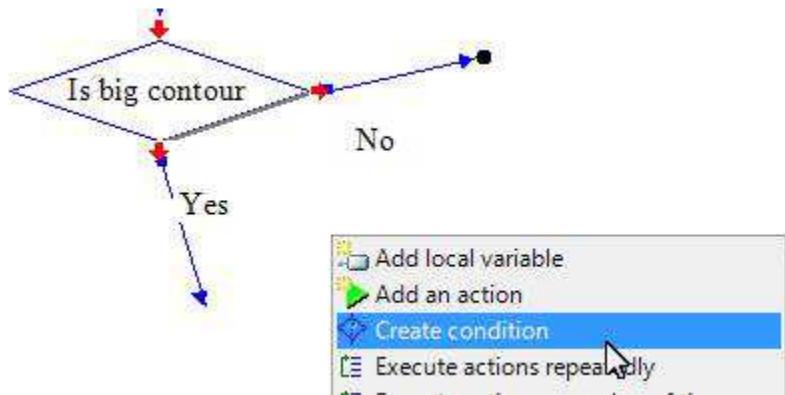
Rename the condition:

Use Generic Types and Native Libraries



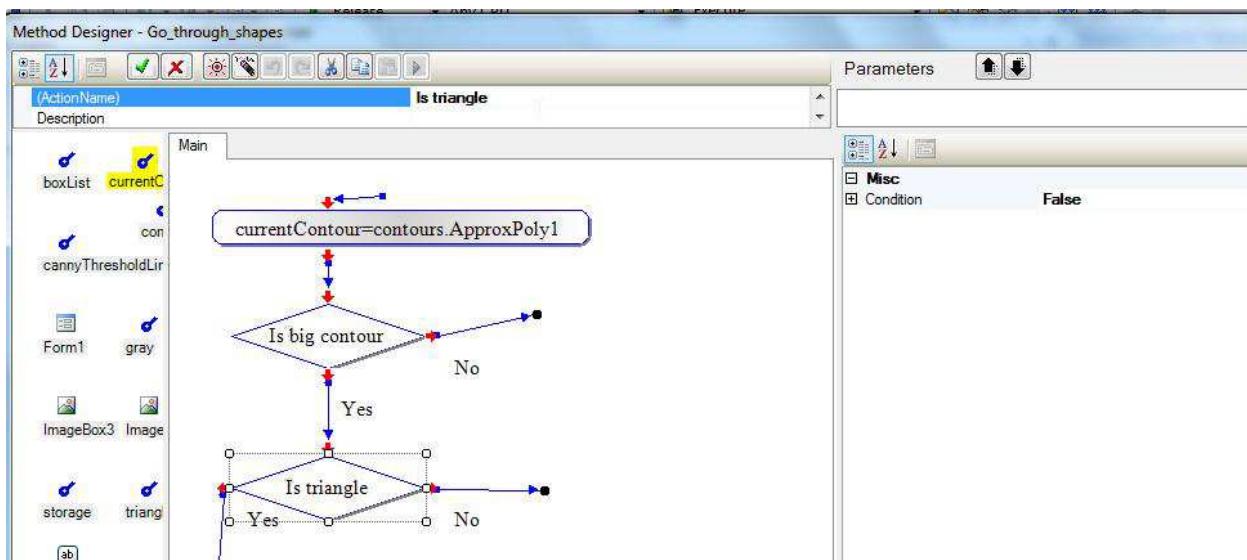
Get triangle

If a contour has 3 vertices then it is a triangle.



Rename the condition to "Is triangle":

Use Generic Types and Native Libraries



Set its condition to check if the contour has 3 points:

The first screenshot shows the 'Misc' panel with the 'Condition' field set to 'False'. A dropdown menu is open, showing options like 'ConstantValue', 'Property', and 'MathExpression'. The cursor is hovering over 'MathExpression'.

The second screenshot shows the 'Math Expression Editor' window. The toolbar includes buttons for Decimal, Integer, Logic, Text, α-ω, System, and various operators. The expression bar shows '0' and 'Logic 'equal'' operator. The status bar indicates 'A property'.

The third screenshot shows the 'Math Expression Editor' window again. The toolbar includes buttons for Decimal, Integer, Logic, Text, α-ω, System, and various operators. The expression bar shows 'TRUE' and 'TRUE'. The status bar indicates 'A property'.

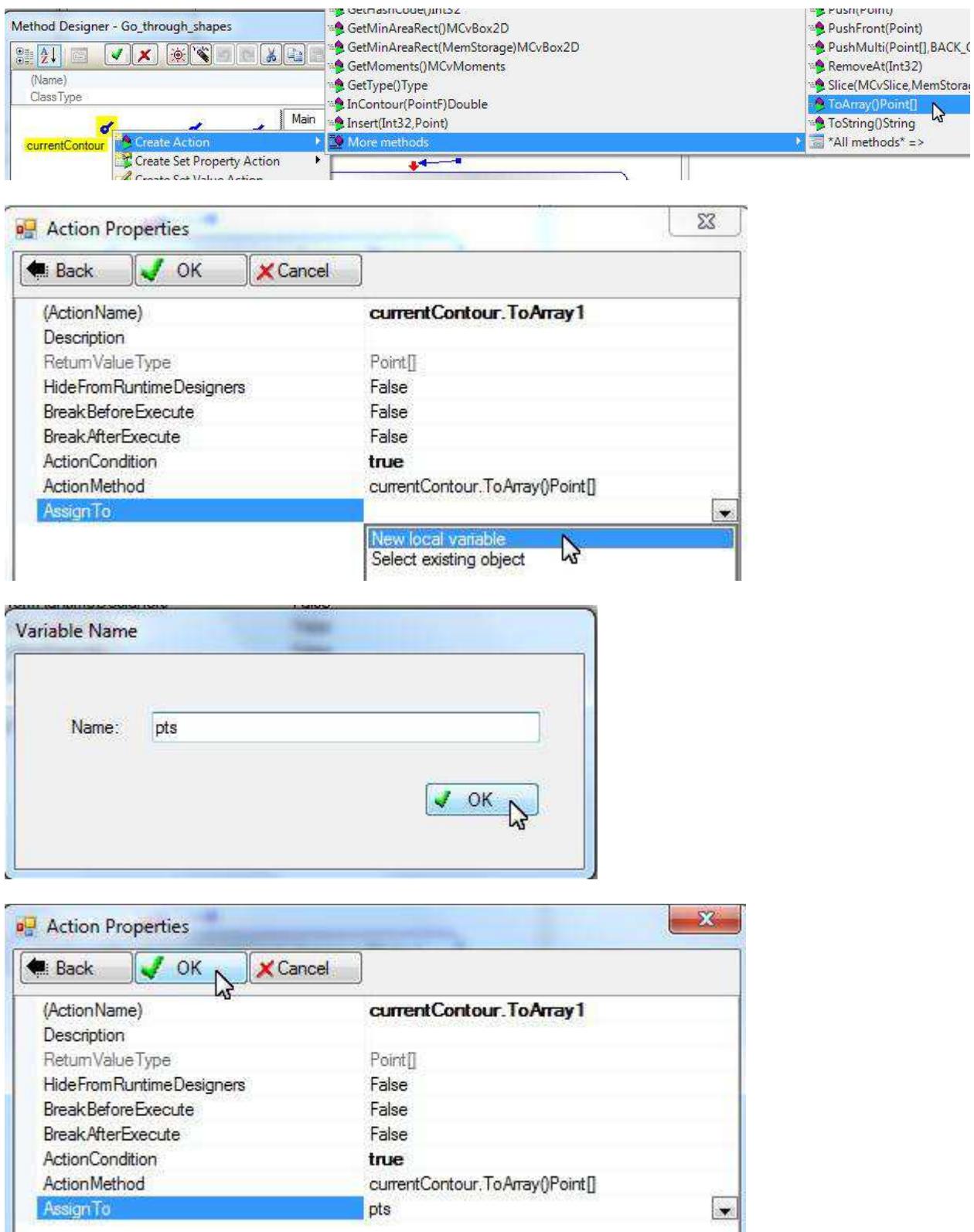
Use Generic Types and Native Libraries

The figure consists of three screenshots of a software interface:

- Object picker window:** Shows a tree view of objects and their properties. The current node is `currentContour`, which has an `Instance Members` section expanded. Under `Properties inherited`, the `Total:Int32` property is selected.
- Math Expression Editor window (Step 1):** Shows the expression `currentContour.Total`. The `Type` dropdown is set to `(Boolean)`. A tooltip says: "currentContour.Total is a constant value, the value type may be changed."
- Math Expression Editor window (Step 2):** Shows the expression `currentContour.Total` with a value of `3` assigned. The `Type` dropdown is now set to `(Int32)`. A tooltip says: "Finish the editing."

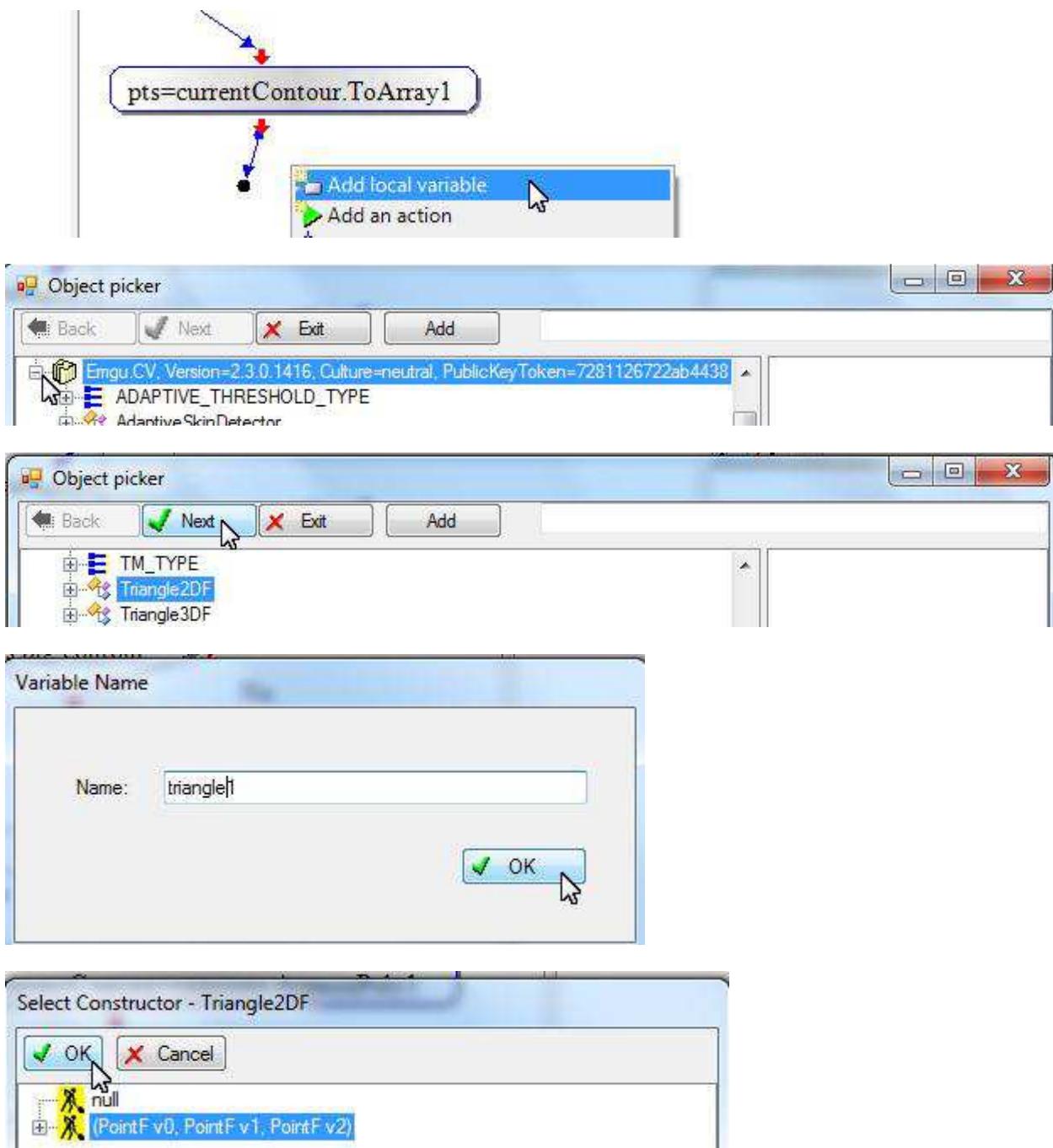
If it is a triangle then get the 3 points from the contour and use the points to create a triangle.

Use Generic Types and Native Libraries



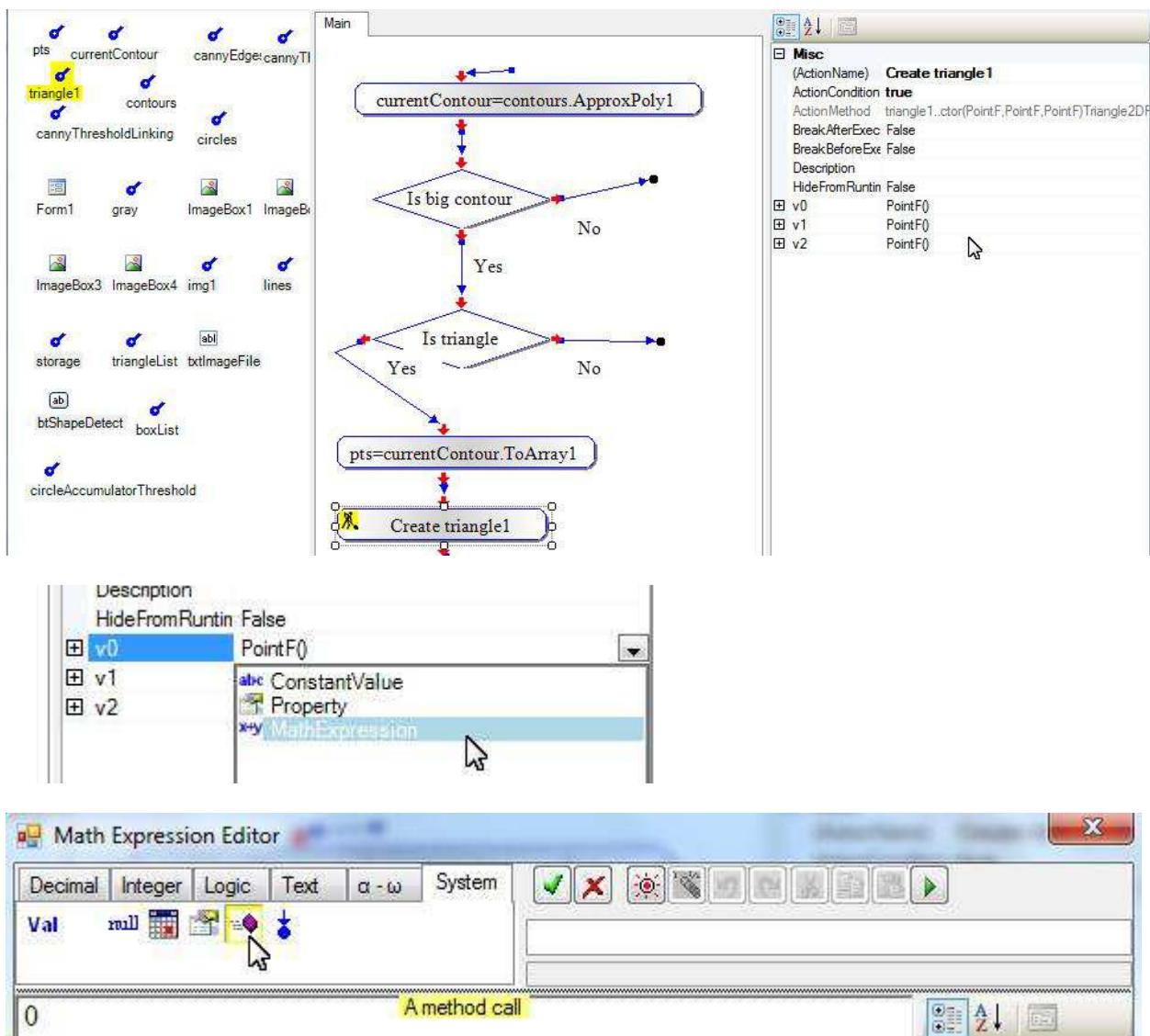
Create triangle:

Use Generic Types and Native Libraries

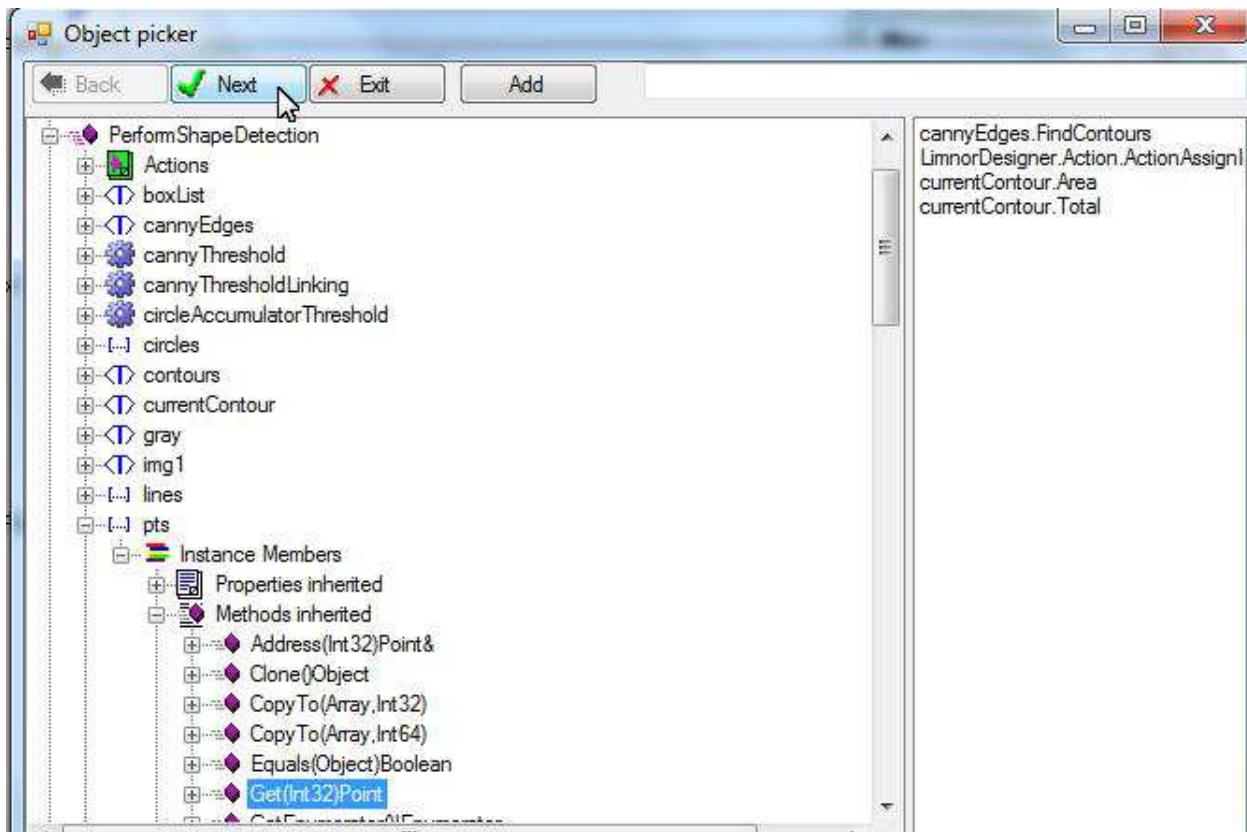


We need to specify the 3 points for the triangle:

Use Generic Types and Native Libraries



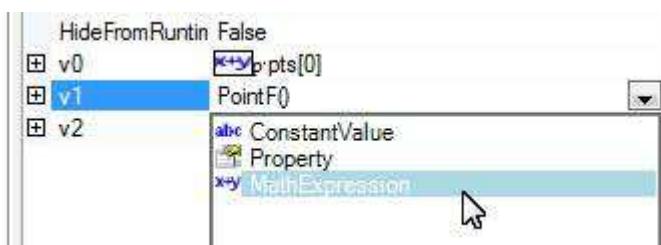
Use Generic Types and Native Libraries



0 indicates the first point in the point array:

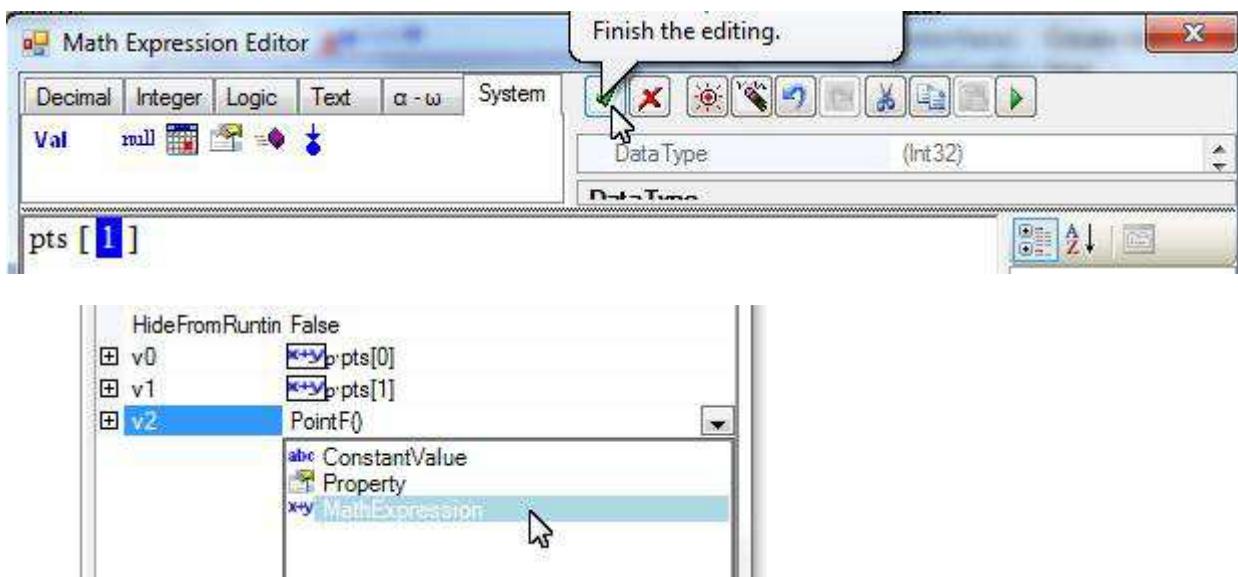


Do the same for the other 2 points of the triangle:



1 indicates the second point in the array:

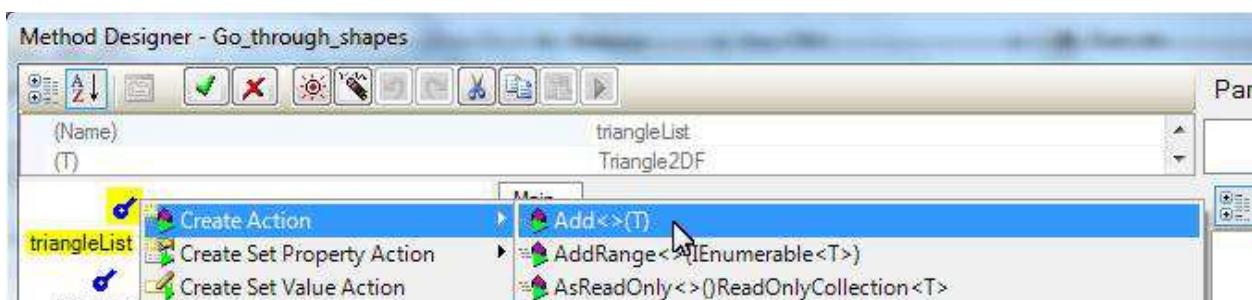
Use Generic Types and Native Libraries



2 indicates the 3rd point in the array:



Add the triangle to the list:



Use Generic Types and Native Libraries

The figure consists of three screenshots from a software interface, likely a graphical programming environment.

Screenshot 1: Action Properties

This screenshot shows the "Action Properties" dialog box. The action name is set to "triangleList.Add1". The "ActionMethod" field contains the code "triangleList.Add<(>(T) Triangle2DF0)". The "item" field is expanded, showing options like "ConstantValue", "PROPERTY", and "MathExpression".

(ActionName)	triangleList.Add1
Description	
HideFromRuntimeDesigners	False
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	true
ActionMethod	triangleList.Add<(>(T) Triangle2DF0)
item	(T)

Screenshot 2: Object picker

This screenshot shows the "Object picker" dialog box. It displays a tree view of objects and their properties. The "triangle1" object under "triangleList" is selected. A list of properties is shown on the right side.

- Form1 from DrawingPage
- Primary types
- PerformShapeDetection
 - Actions
 - boxList
 - cannyEdges
 - cannyThreshold
 - cannyThresholdLinking
 - circleAccumulatorThreshold
 - circles
 - contours
 - currentContour
 - gray
 - img1
 - lines
 - pts
 - storage
 - triangle1
- triangleList

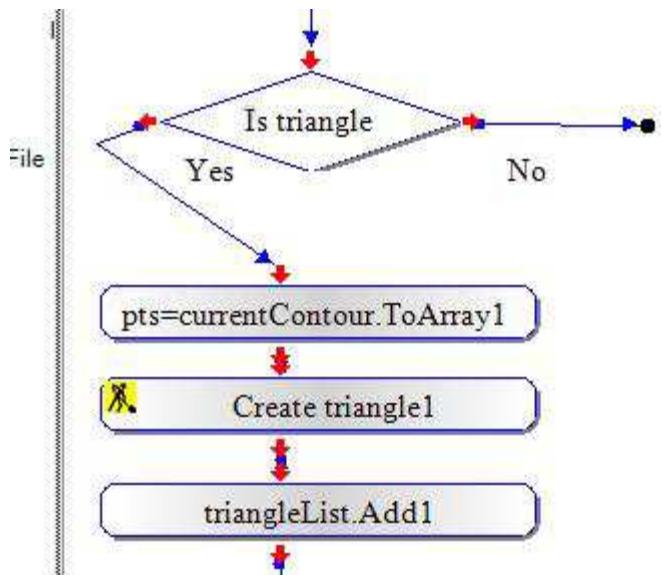
cannyEdges.FindContours
LinnorDesigner.Action.ActionAssign1
currentContour.Area
currentContour.Total
pts.Get(Int32)Point

Screenshot 3: Action Properties

This screenshot shows the "Action Properties" dialog box again, with the "item" field now containing "Triangle2DF".

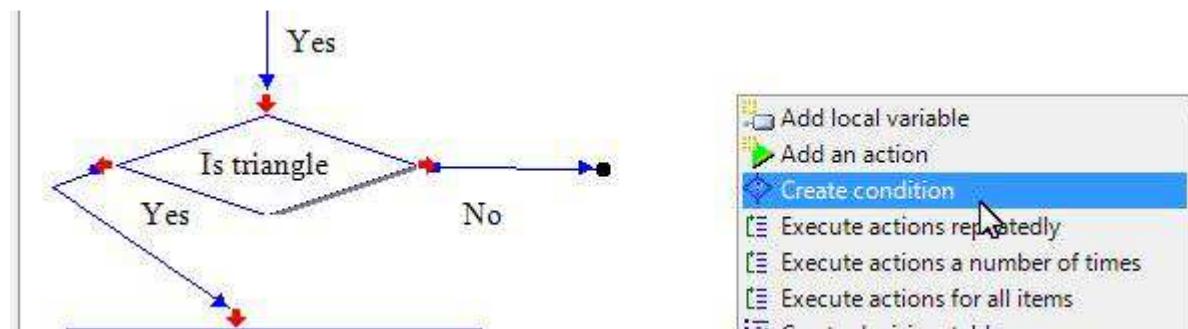
(ActionName)	triangleList.Add1
Description	
HideFromRuntimeDesigners	False
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	true
ActionMethod	triangleList.Add<(>(T) Triangle2DF)
item	(Triangle2DF)

Use Generic Types and Native Libraries

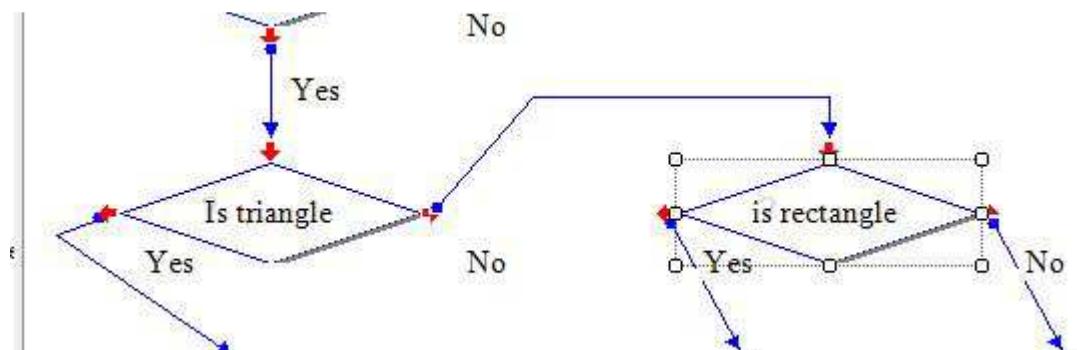


Get rectangle

If a contour has 4 vertices then it is a rectangle.

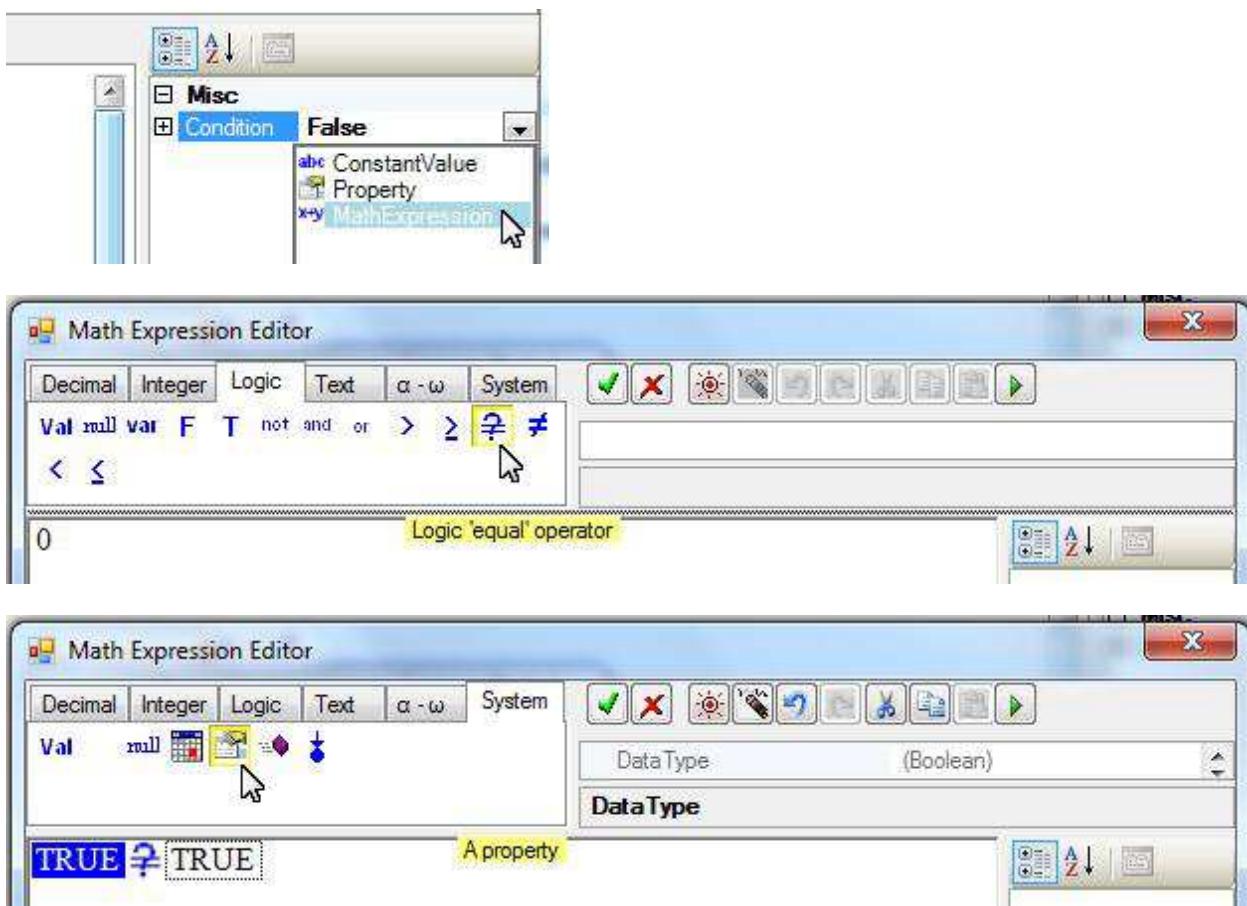


Change its name to "Is rectangle":



Set its condition to check if the contour has 4 vertices:

Use Generic Types and Native Libraries



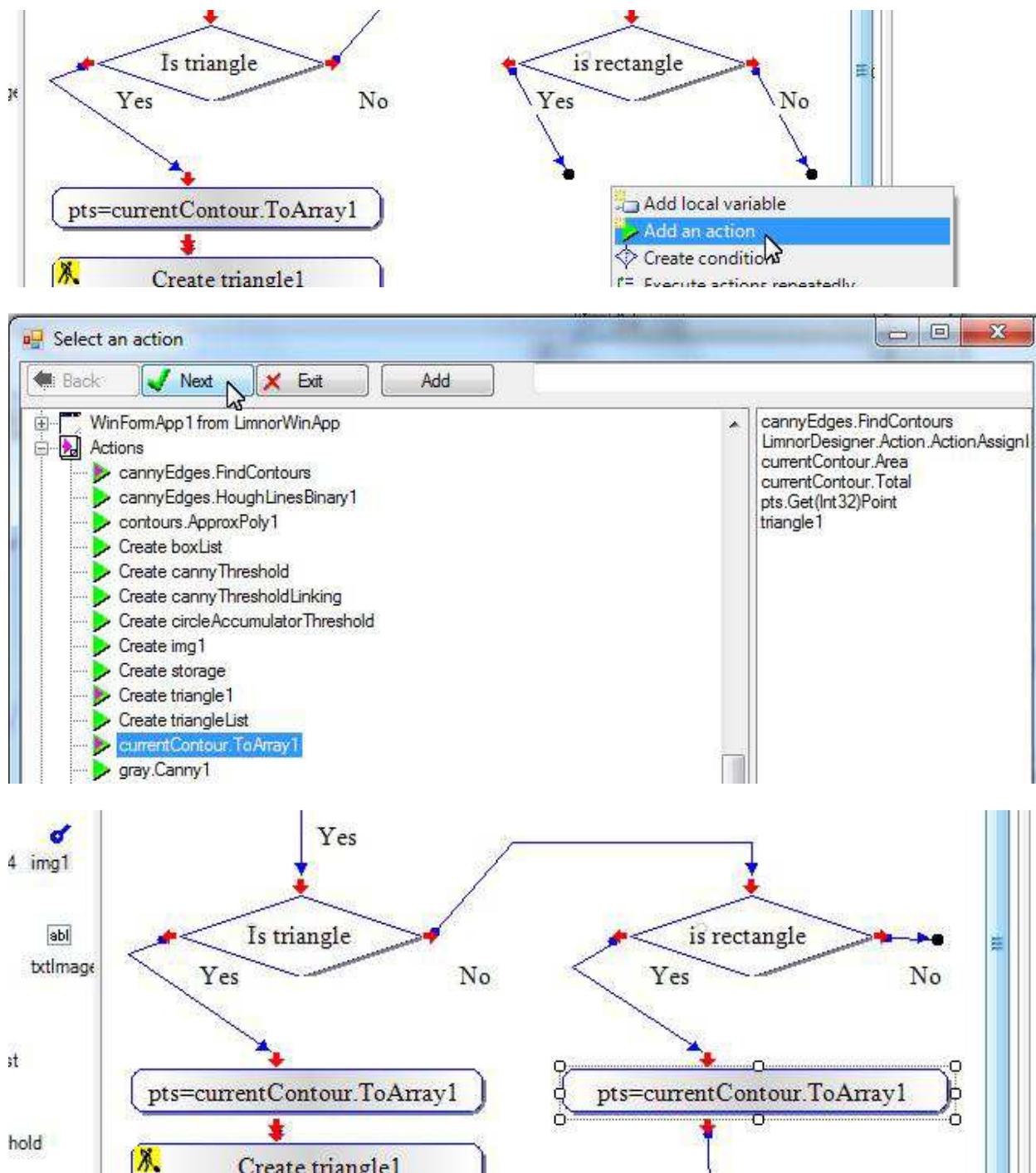
Use Generic Types and Native Libraries

The figure consists of three screenshots of the Limnor Designer software interface:

- Object picker window:** Shows the "PerformShapeDetection" action node expanded. Under "currentContour", the "Instance Members" section is selected, showing various properties like Area:Double, BoundingRectangle:Rectangle, Convex:Boolean, etc. The "Total:Int32" property is highlighted with a blue selection bar.
- Math Expression Editor (initial state):** The expression "currentContour.Total" is entered. The "Val" button is highlighted with a yellow selection bar. The "DataType" dropdown shows "(Boolean)". A tooltip says: "currentContour.Total a constant value, the value type may be changed."
- Math Expression Editor (final state):** The expression "currentContour.Total" is now evaluated as the value "4". The "Val" button is no longer highlighted. The "DataType" dropdown shows "(Int32)". A tooltip says: "Finish the editing."

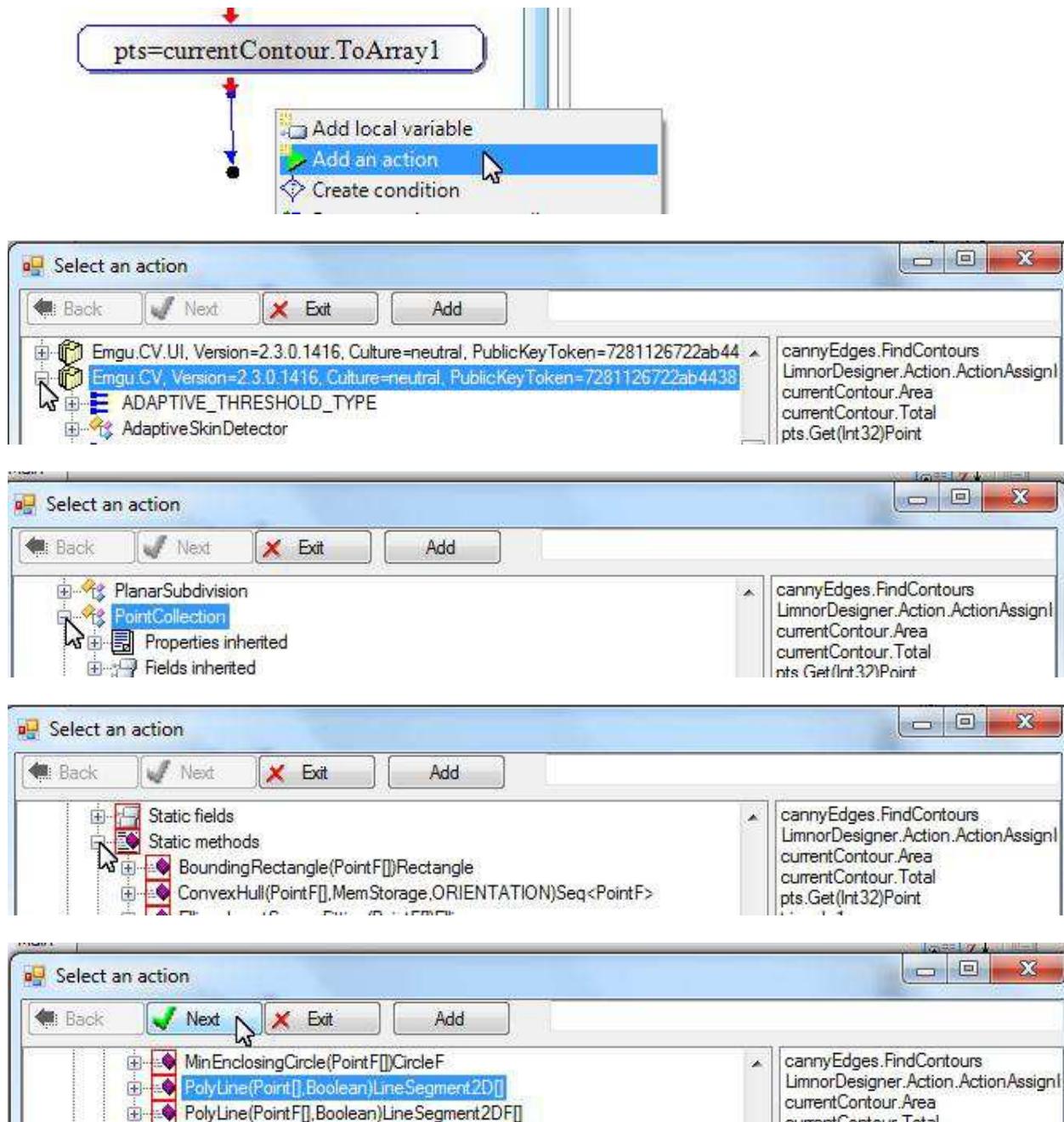
Determine if all the angles in the contour are within [80, 100] degree by forming the line segments from the contour vertices.

Use Generic Types and Native Libraries



Create line segments from the point array:

Use Generic Types and Native Libraries



Use Generic Types and Native Libraries

The image consists of two screenshots from a software application, likely Limnor Designer.

Screenshot 1: Action Properties

This screenshot shows the "Action Properties" dialog box. The "ActionName" is set to "PointCollection.PolyLine". The "ReturnValueType" is "LineSegment2D[]". The "points" property is currently set to "Point[]()", which is highlighted in blue. A dropdown menu is open next to it, showing three options: "ConstantValue", "Property", and "MathExpression".

(ActionName)	PointCollection.PolyLine
Description	
ReturnValueType:	LineSegment2D[]
HideFromRuntimeDesigners	False
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	true
ActionMethod	PointCollection.PolyLine(Point[], Boolean) LineSegm
points	Point[]()
closed	
closed	
AssignTo	

Screenshot 2: Object picker

This screenshot shows the "Object picker" dialog box. It lists various objects and variables under the "PerformShapeDetection" action. The "pts" variable is selected and highlighted in blue. The right pane displays the properties of the selected object, including "cannyEdges.FindContours", "LimnorDesigner.Action.ActionAssign1", "currentContour.Area", "currentContour.Total", "pts.Get(Int32)Point", "triangle1", and "currentContour.ToArray1".

- + Form1 from DrawingPage
- + Abc Primary types
- PerformShapeDetection
 - + Actions
 - + <T> boxList
 - + <T> cannyEdges
 - + <T> cannyThreshold
 - + <T> cannyThresholdLinking
 - + <T> circleAccumulatorThreshold
 - + ... circles
 - + <T> contours
 - + <T> currentContour
 - + <T> gray
 - + <T> img1
 - + ... lines
 - + ... pts
 - + <T> storage

Use Generic Types and Native Libraries

The figure consists of three screenshots illustrating the configuration of a PolyLine action and its assignment.

Screenshot 1: Action Properties Dialog

This dialog shows the configuration of a PolyLine action:

(ActionName)	PointCollection.PolyLine
Description	
ReturnValueType	LineSegment2D[]
HideFromRuntimeDesigners	False
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	true
ActionMethod	PointCollection.PolyLine(Point[], Boolean) LineSegm
points	pts
closed	True
closed	True
AssignTo	edges

A dropdown menu is open under "AssignTo" with the option "New local variable" selected.

Screenshot 2: Variable Name Dialog

This dialog shows the creation of a new local variable:

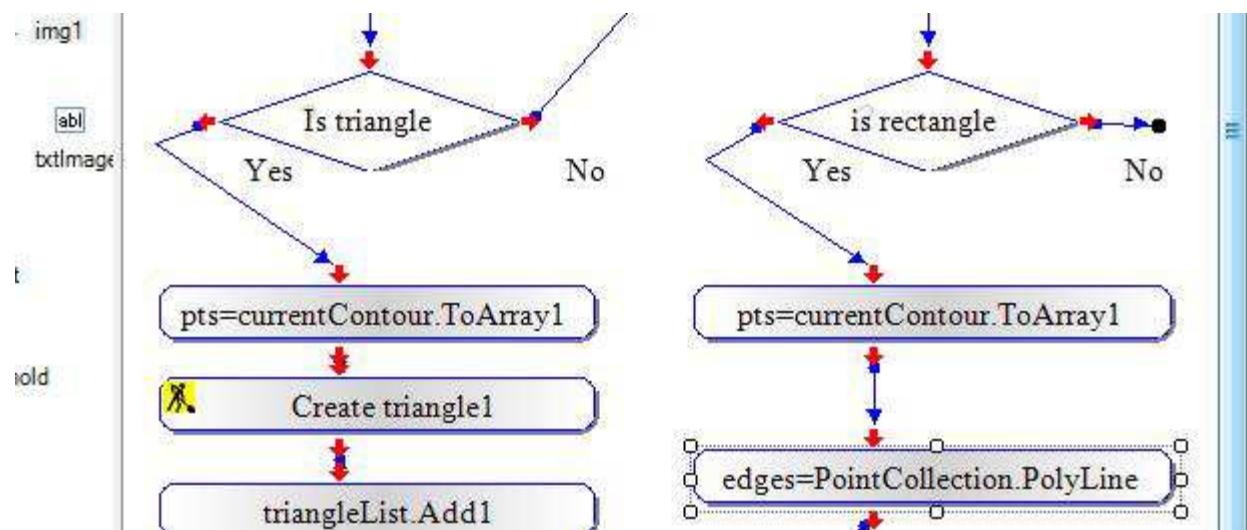
Name: edges

Screenshot 3: Action Properties Dialog

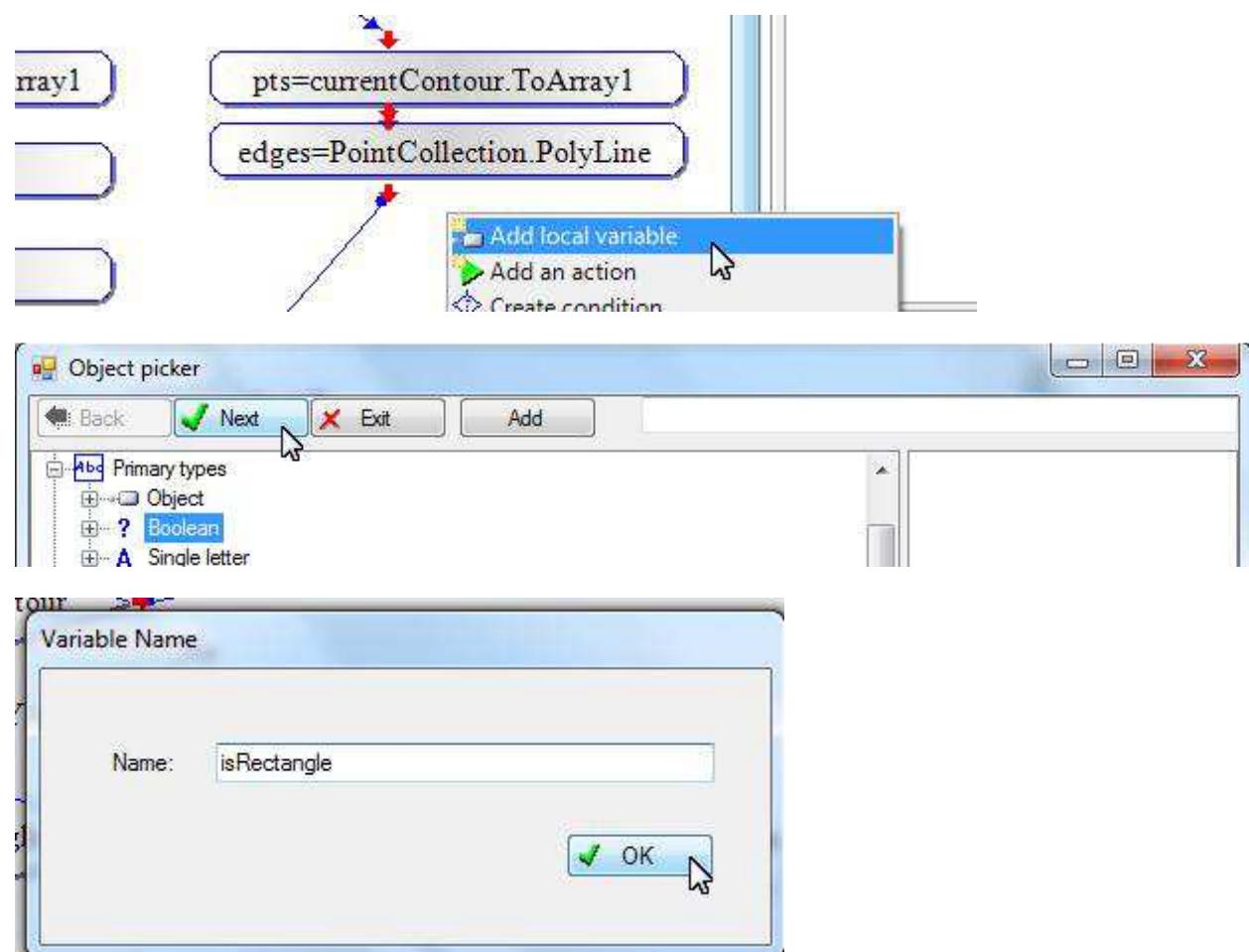
This dialog shows the final configuration of the PolyLine action with the "AssignTo" field set to "edges".

(ActionName)	PointCollection.PolyLine
Description	
ReturnValueType	LineSegment2D[]
HideFromRuntimeDesigners	False
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	true
ActionMethod	PointCollection.PolyLine(Point[], Boolean) LineSegm
points	pts
closed	True
closed	True
AssignTo	edges

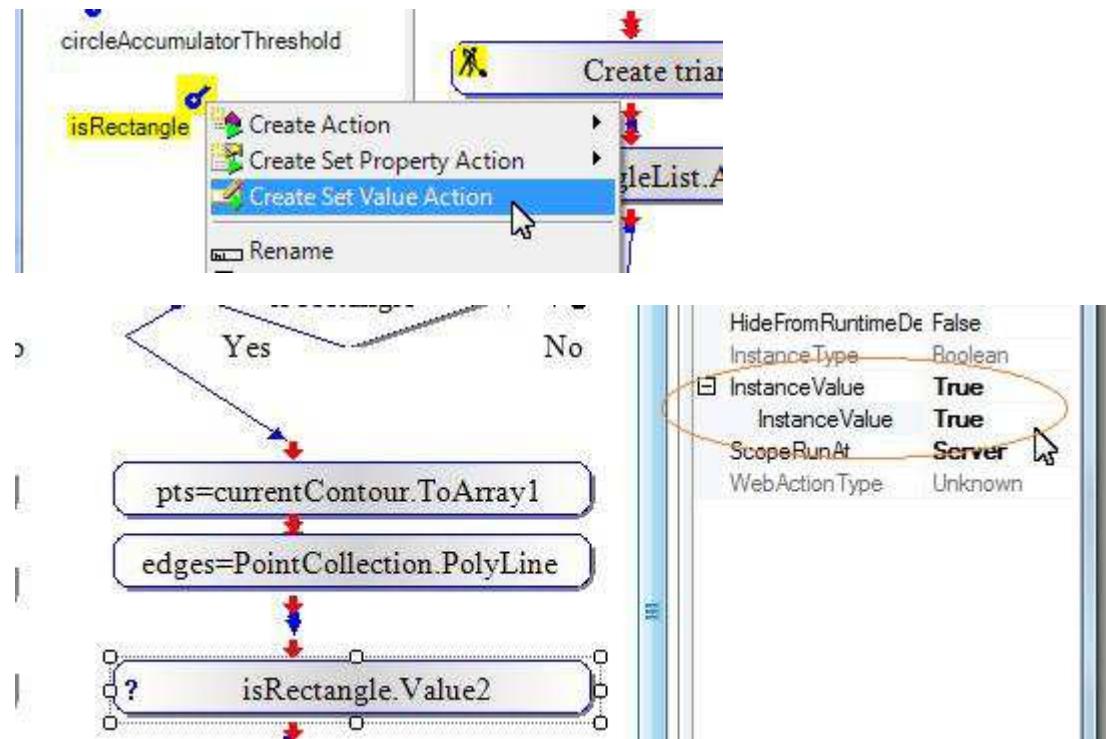
Use Generic Types and Native Libraries



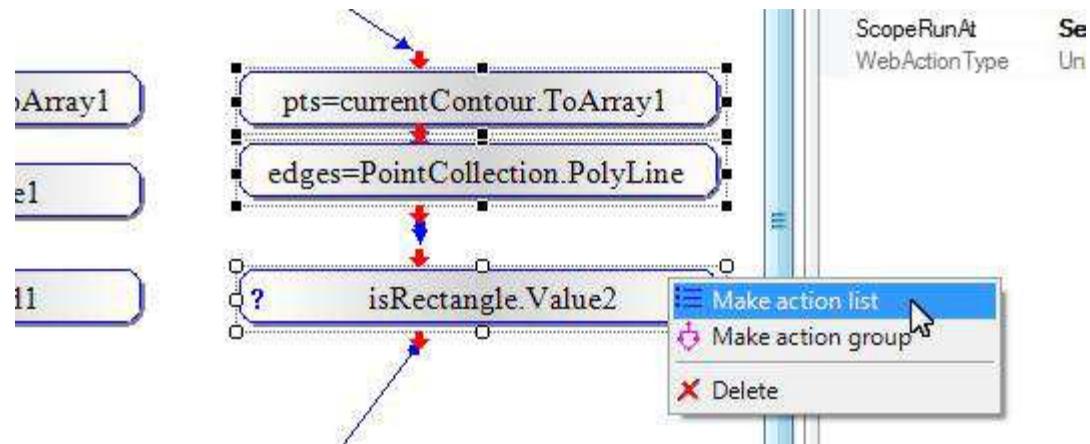
Each angle in a rectangle is about 90 degree. We'll go through the edges to see if there is an angle outside of [80, 100] degree. Before doing that we create a Boolean variable and set it True, indicating that we assume the contour is a rectangle. If we found an angle outside of [80, 100] degree then we set the variable to False, indicating that actually the contour is not a rectangle.



Use Generic Types and Native Libraries

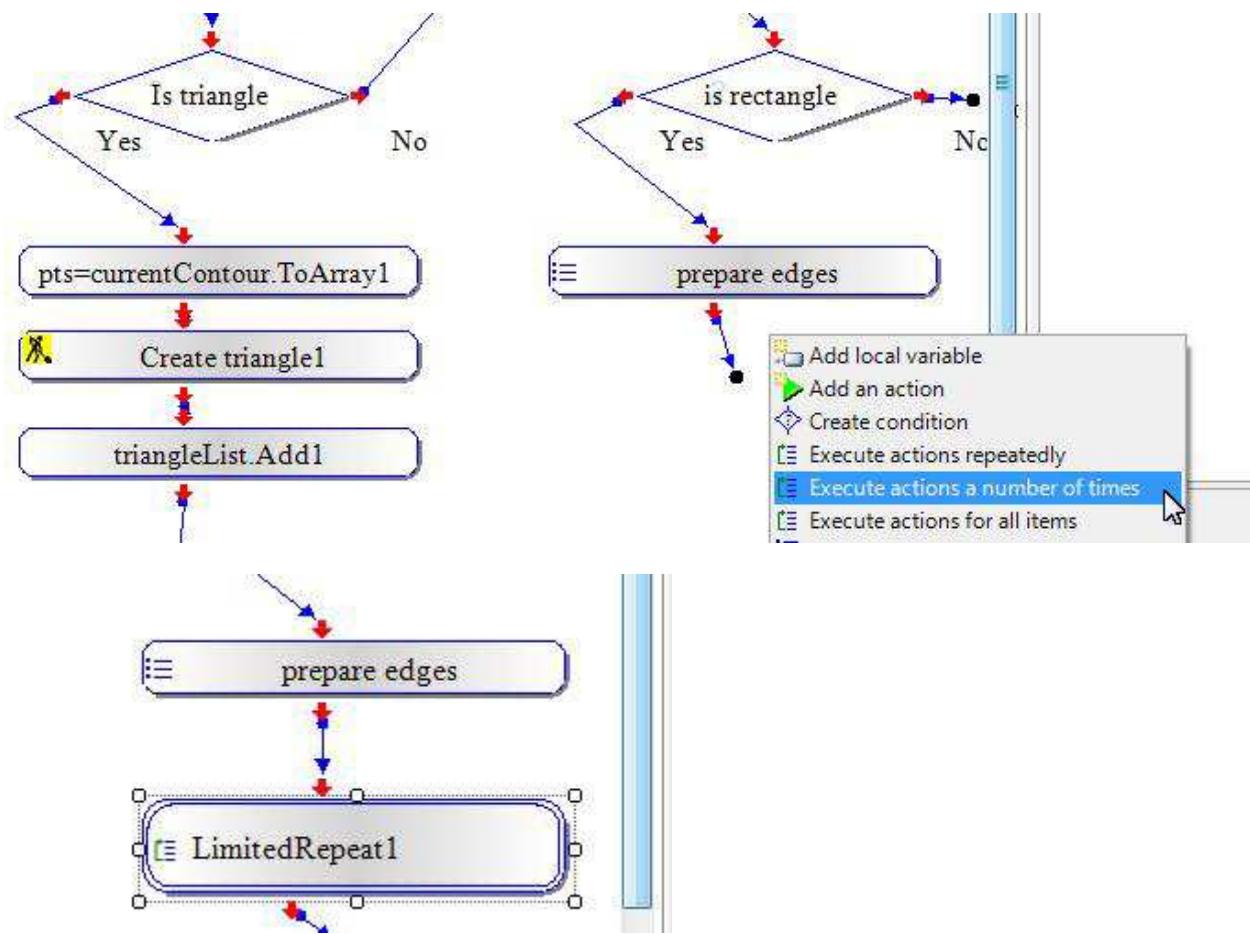


Make the 3 new actions into an action list to make room in the editor:

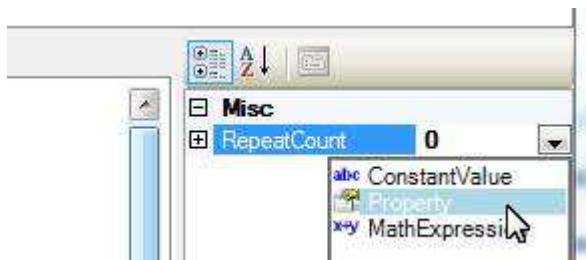


Create an action to go through edges:

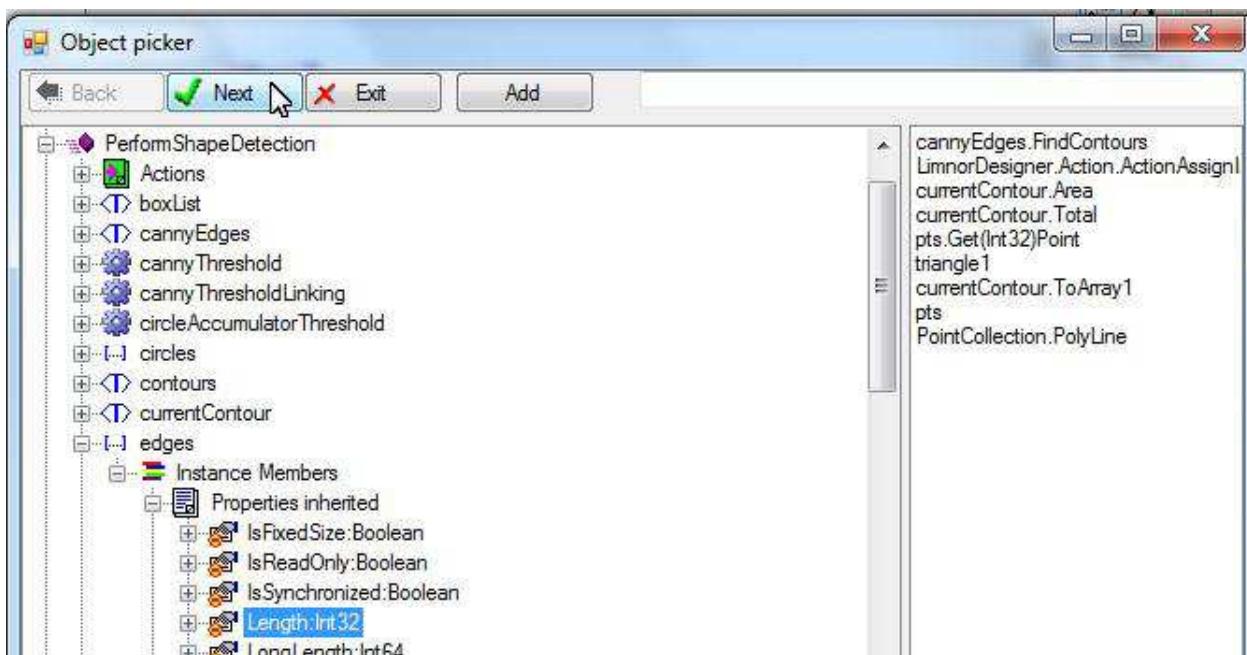
Use Generic Types and Native Libraries



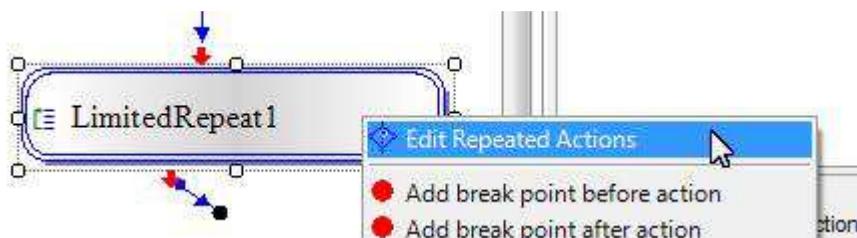
Set its repeat count to the number of edges:



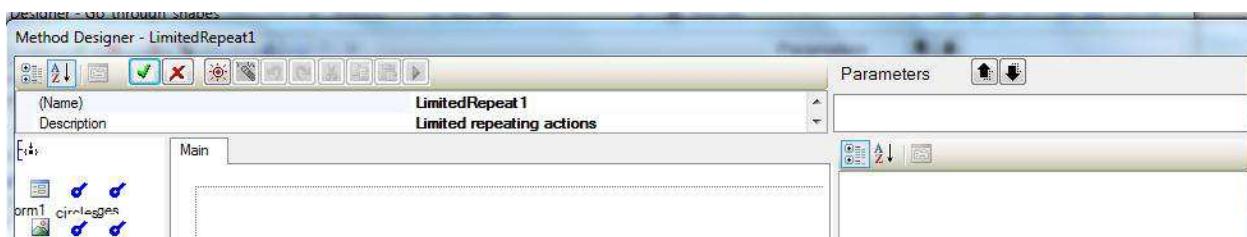
Use Generic Types and Native Libraries



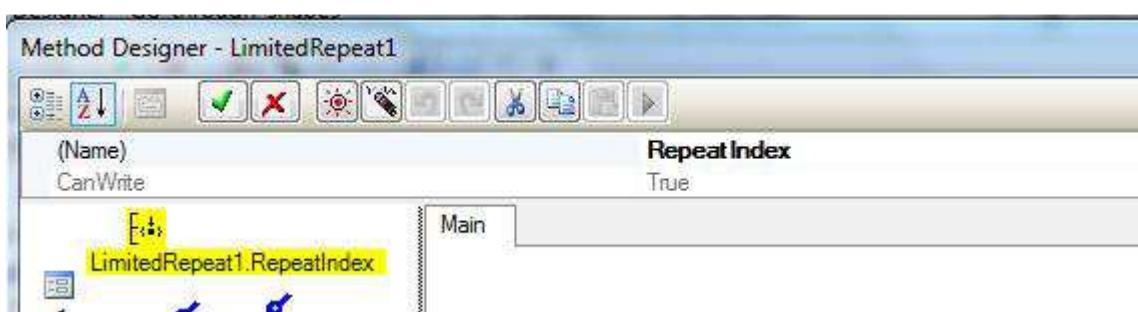
To create actions to process each edge, right-click the loop action, choose “Edit Repeated Actions”:



A new Method Editor appears:



Note the “RepeatedIndex” icon. It represents the edge index, from 0 to number of edges minus 1.



Use Generic Types and Native Libraries

We'll use the "RepeatedIndex" to get the two edges for checking the angle.

The screenshot shows three windows from the Limnor Designer interface:

- Method Designer - LimitedRepeat1**: Shows a code editor with the following snippet:

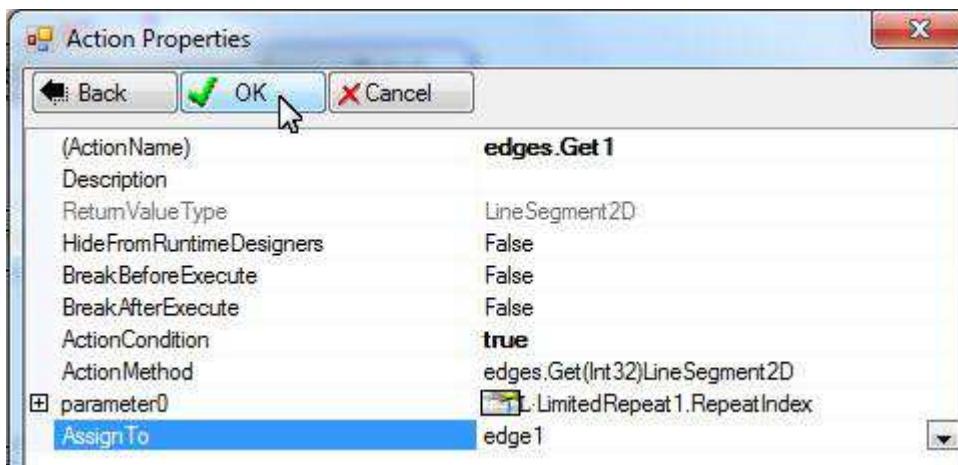
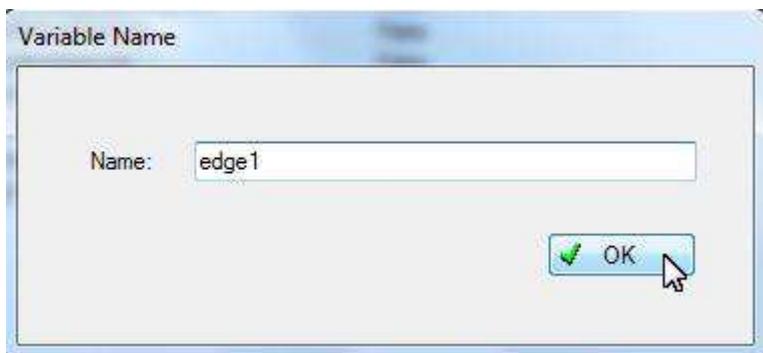
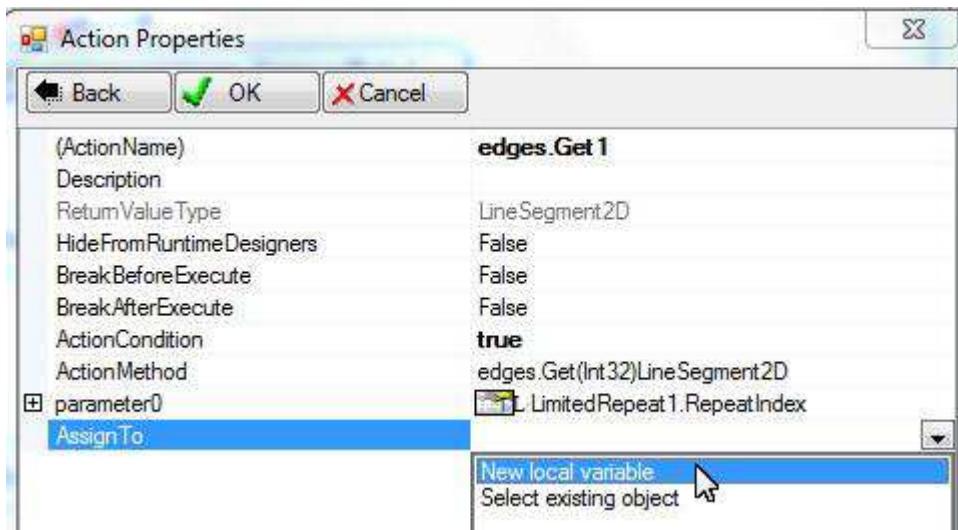
```
edges RepeatIndex
```

A context menu is open over the word "edges", with the "Create Action" option selected. A tooltip "CALCULATED PROPERTY" is visible above the menu.
- Action Properties**: A dialog for the action "edges.Get1".

(ActionName)	edges.Get1
Description	
ReturnValueType	LineSegment2D
HideFromRuntimeDesigners	False
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	true
ActionMethod	edges.Get(Int32)LineSegment2D
parameter0	0

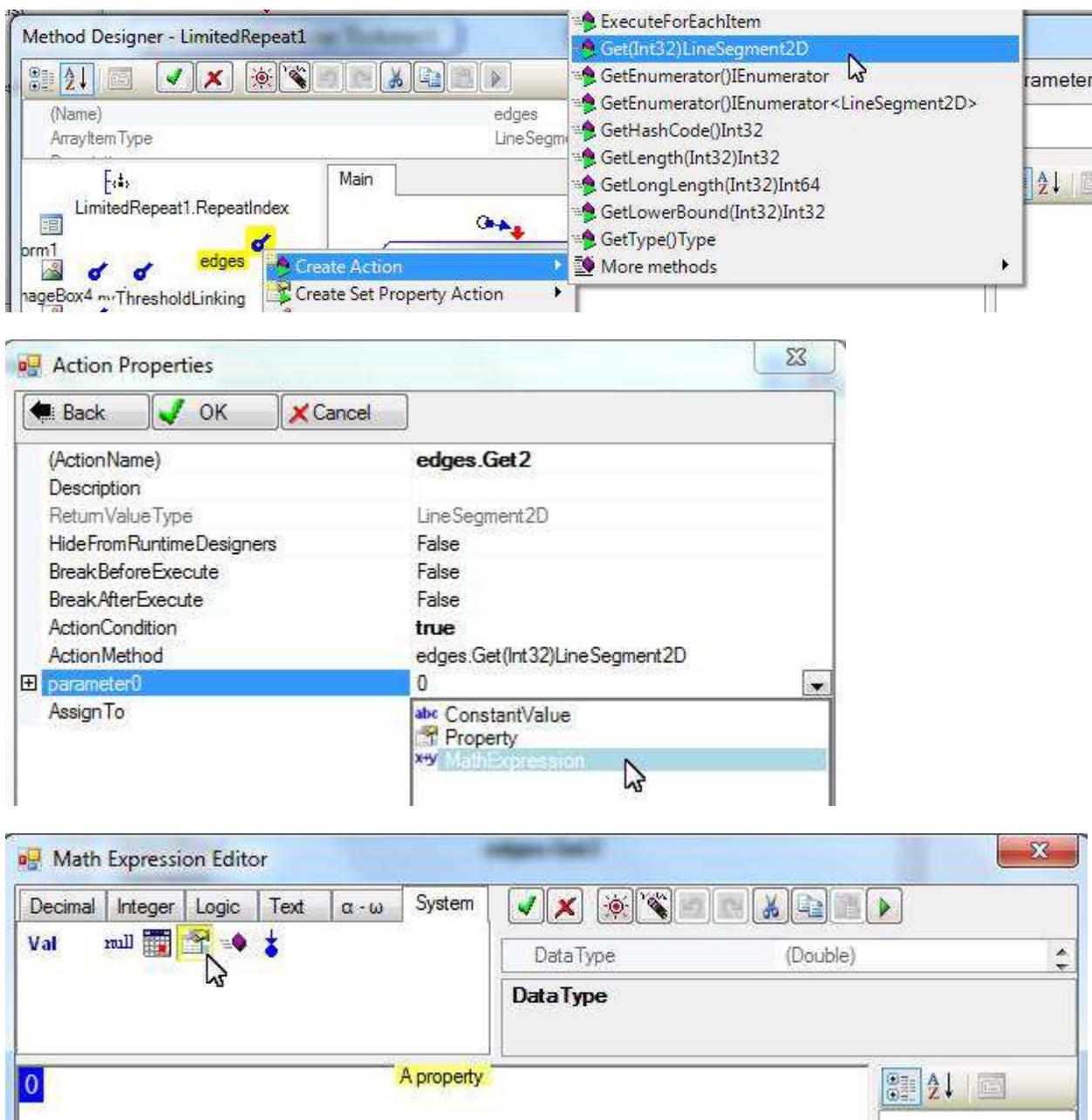
The "parameter0" row has a dropdown menu open, showing options: ConstantValue, Property, and MathExpression. The "Property" option is currently selected.
- Object picker**: A search window showing a tree view of objects and a list of properties.
 - Tree view:
 - Form1 from DrawingPage
 - Primary types
 - PerformShapeDetection
 - Actions
 - boxList
 - cannyEdges
 - cannyThreshold
 - cannyThresholdLinking
 - circleAccumulatorThreshold
 - circles
 - contours
 - currentContour
 - edges
 - gray
 - img1
 - isRectangle
 - LimitedRepeat1 RepeatIndex
 - lines
 - List view (right side):
 - cannyEdges.FindContours
 - LimnorDesigner.Action.ActionAssign1
 - currentContour.Area
 - currentContour.Total
 - pts.Get(Int32)Point
 - triangle1
 - currentContour.ToArray1
 - pts
 - PointCollection.PolyLine
 - edges.Length

Use Generic Types and Native Libraries

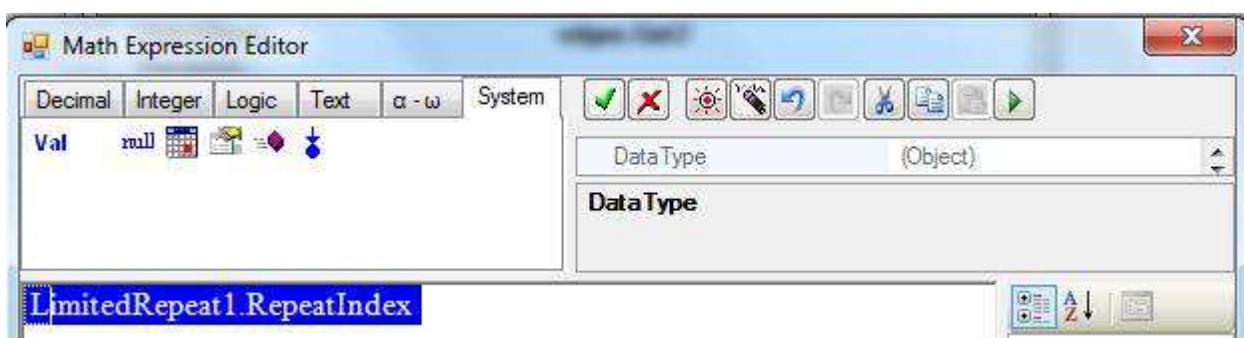
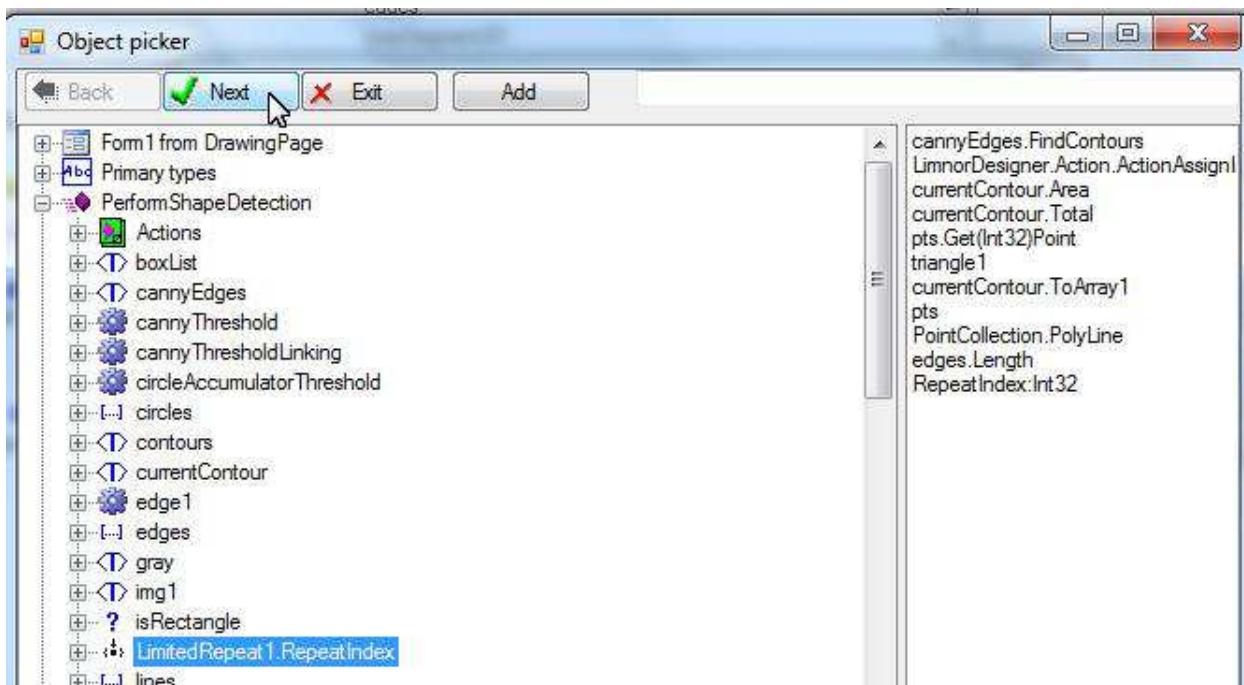


Get the second edge:

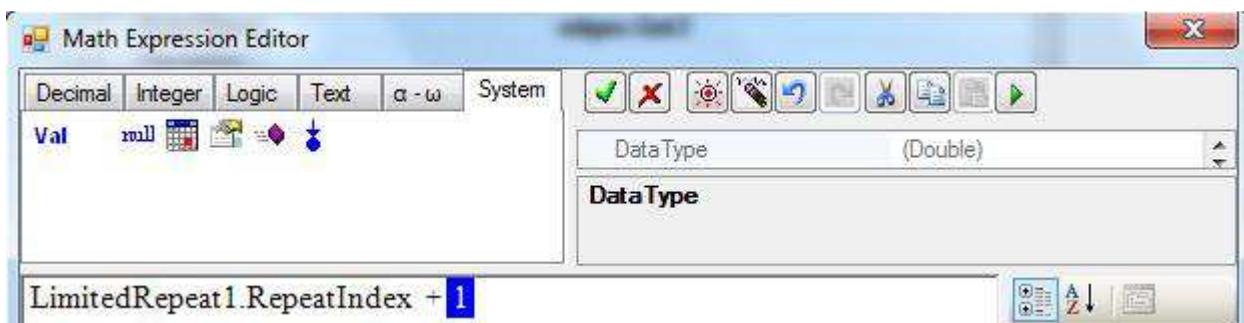
Use Generic Types and Native Libraries



Use Generic Types and Native Libraries

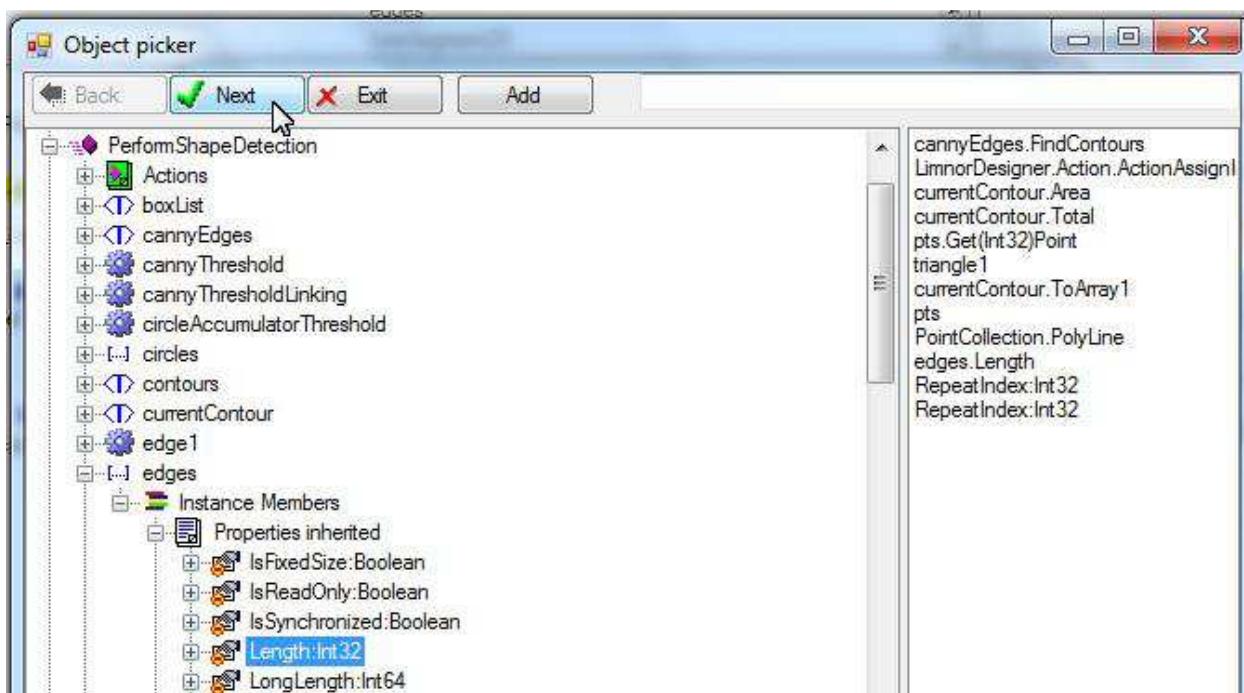
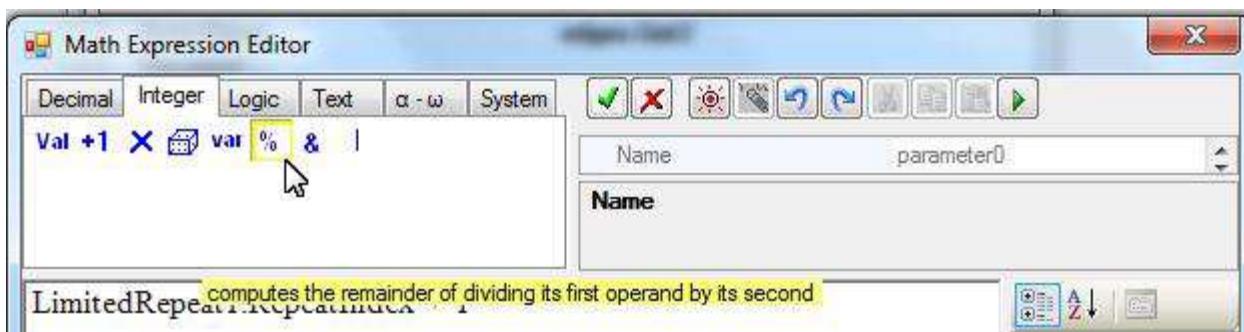


Press “+1”:



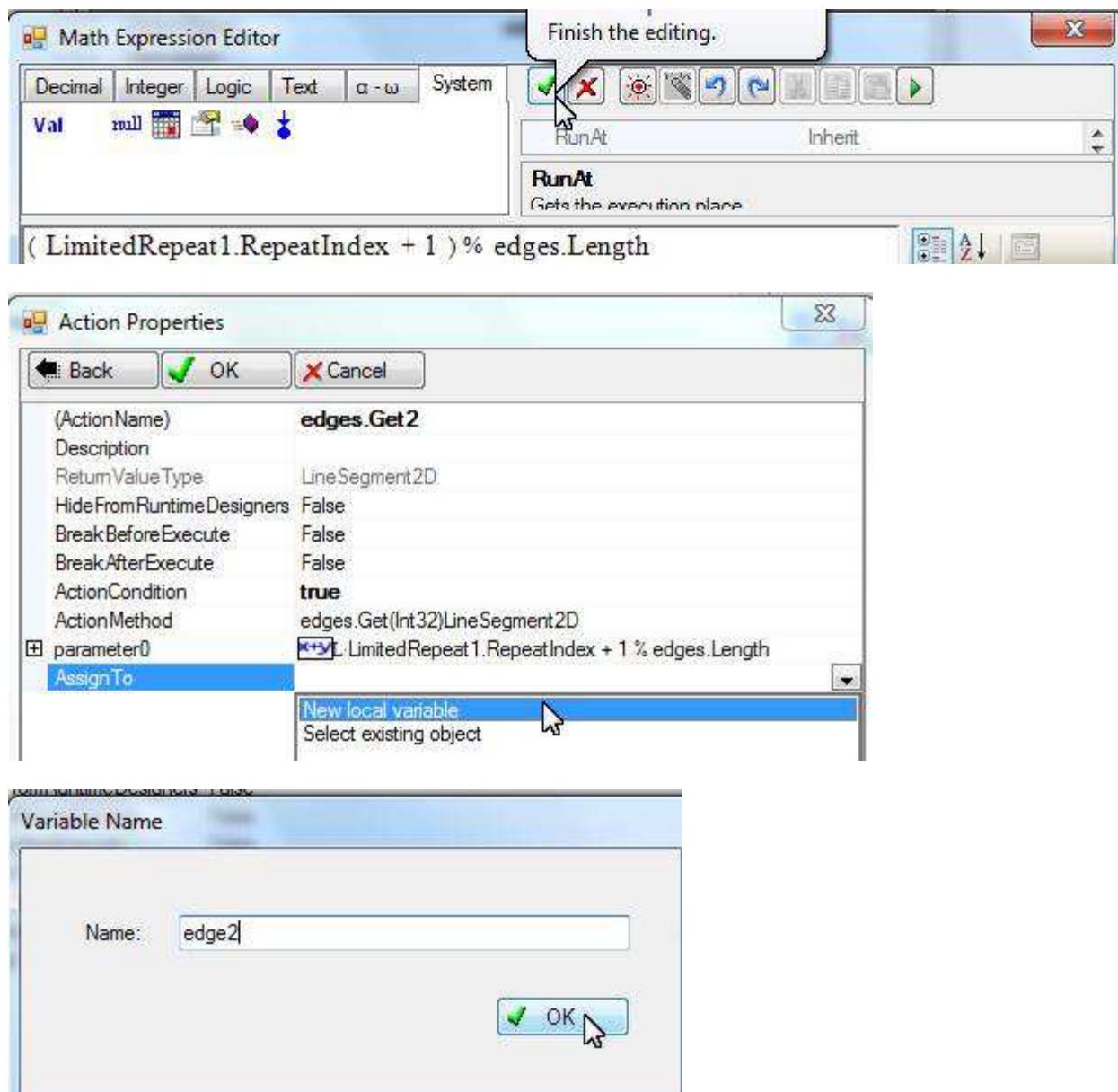
Make sure nothing is highlighted. Click “%”:

Use Generic Types and Native Libraries

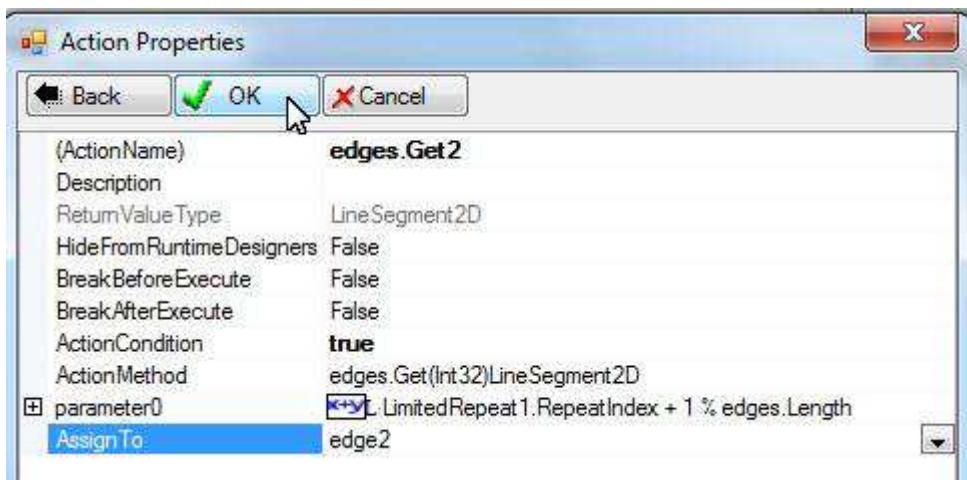


Use Generic Types and Native Libraries

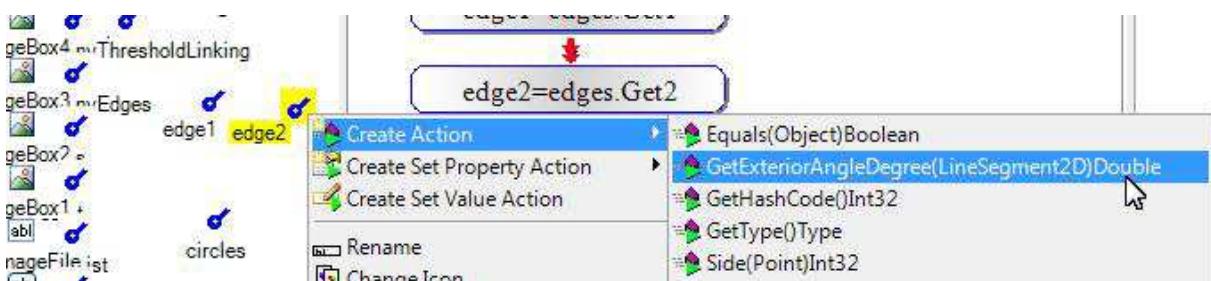
This expression is the array index for the second edge:



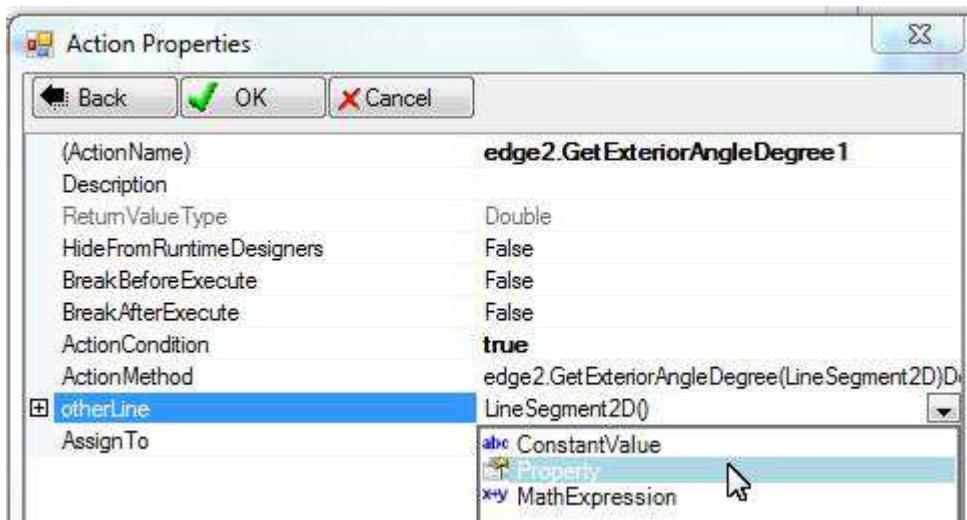
Use Generic Types and Native Libraries



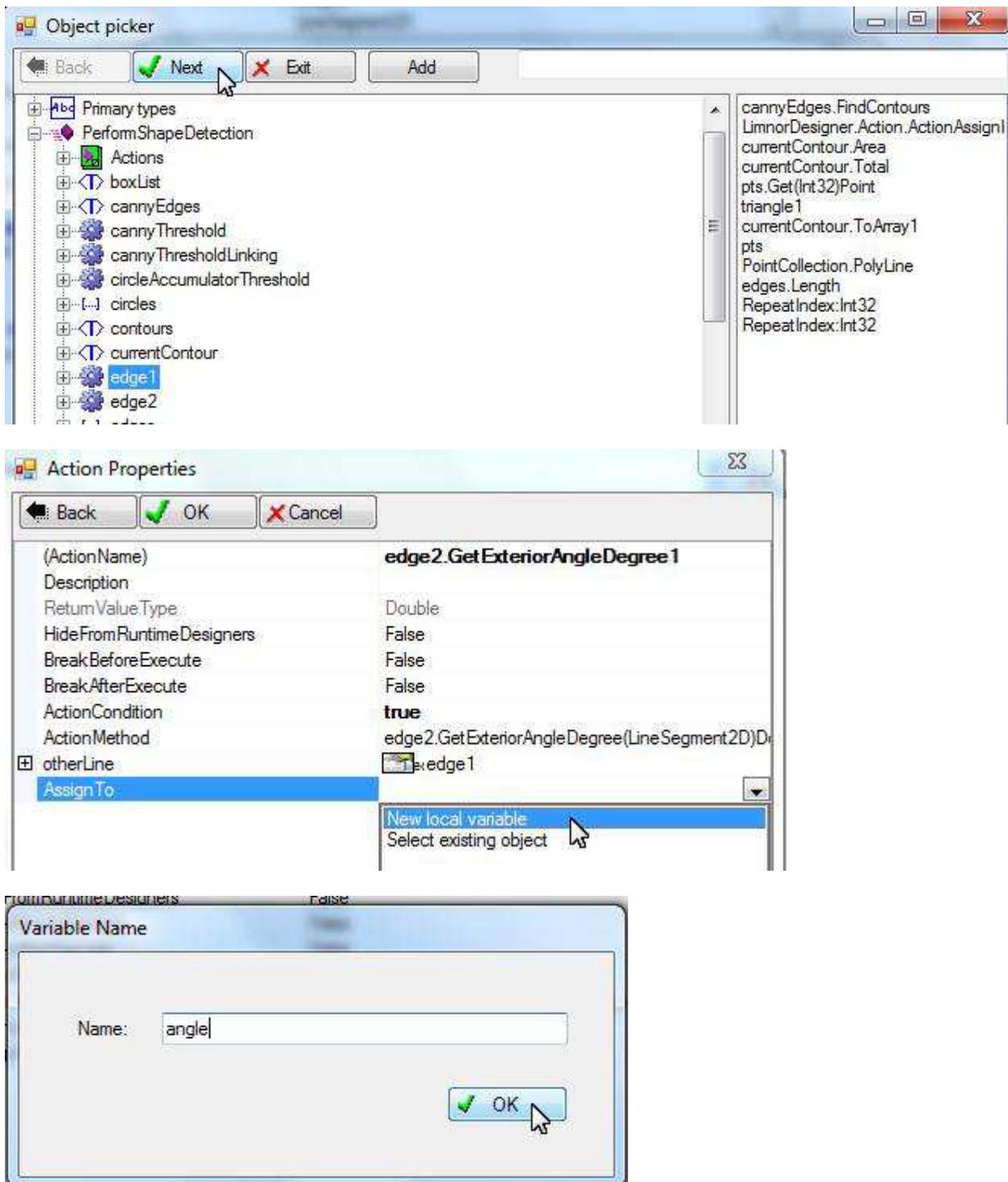
Use the second edge's GetExteriorAngleDegree method to get angle:



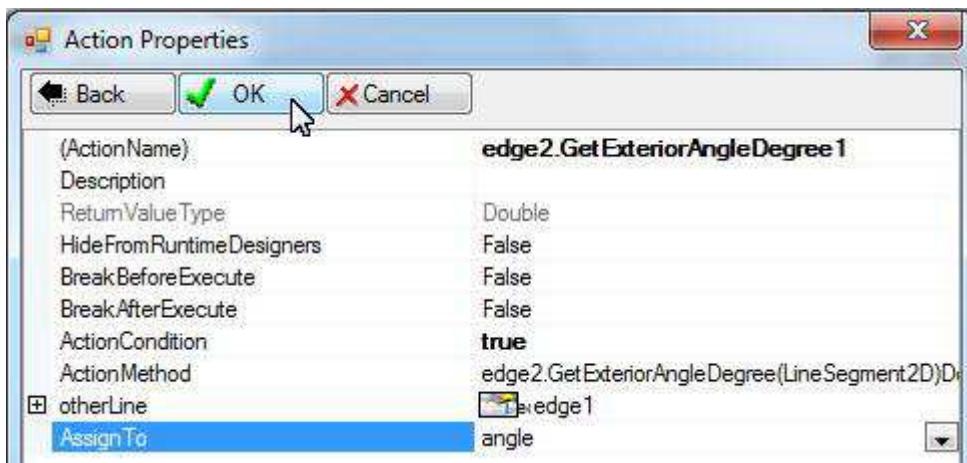
Set "otherLine" to edge1:



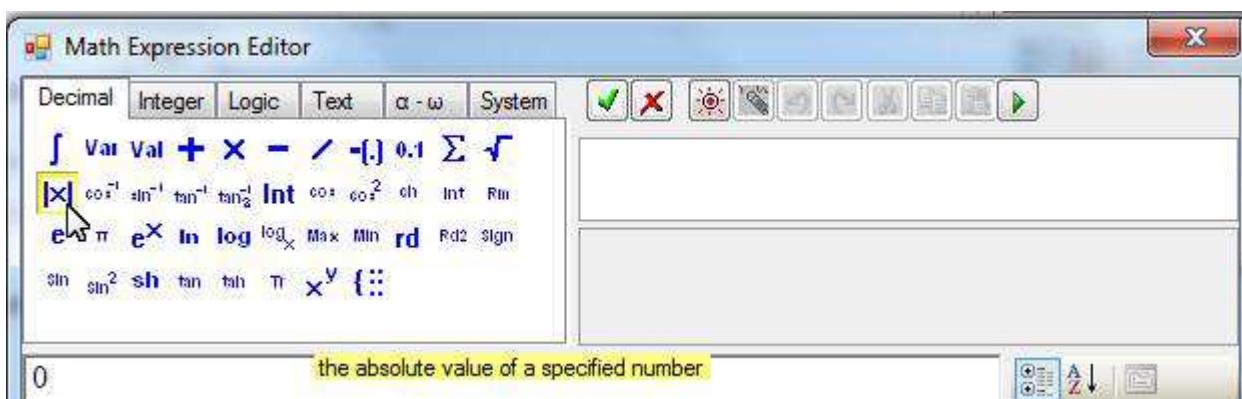
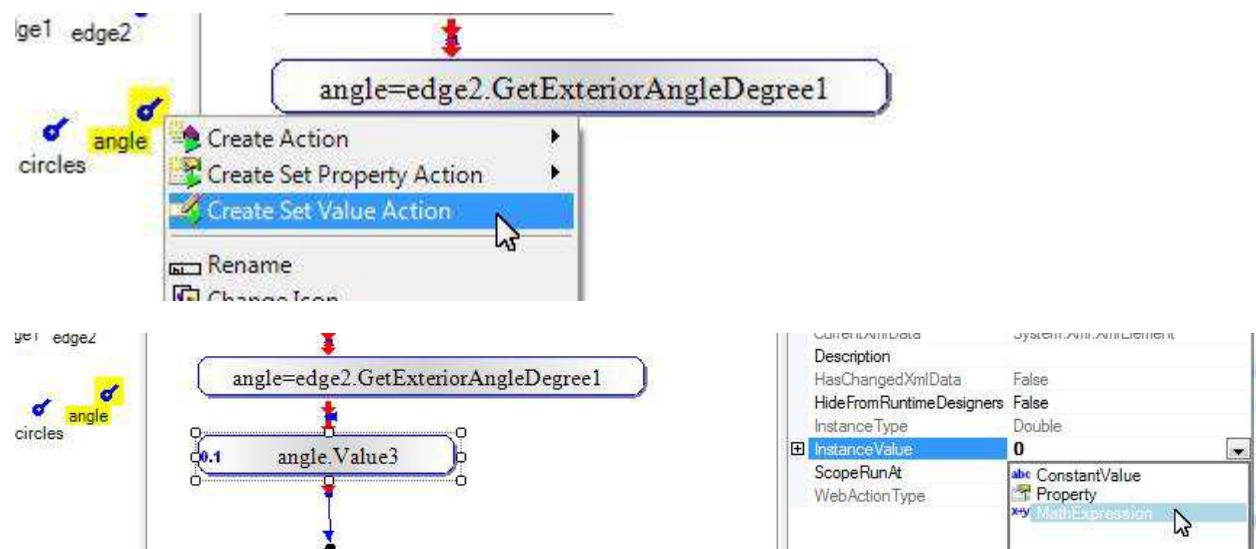
Use Generic Types and Native Libraries



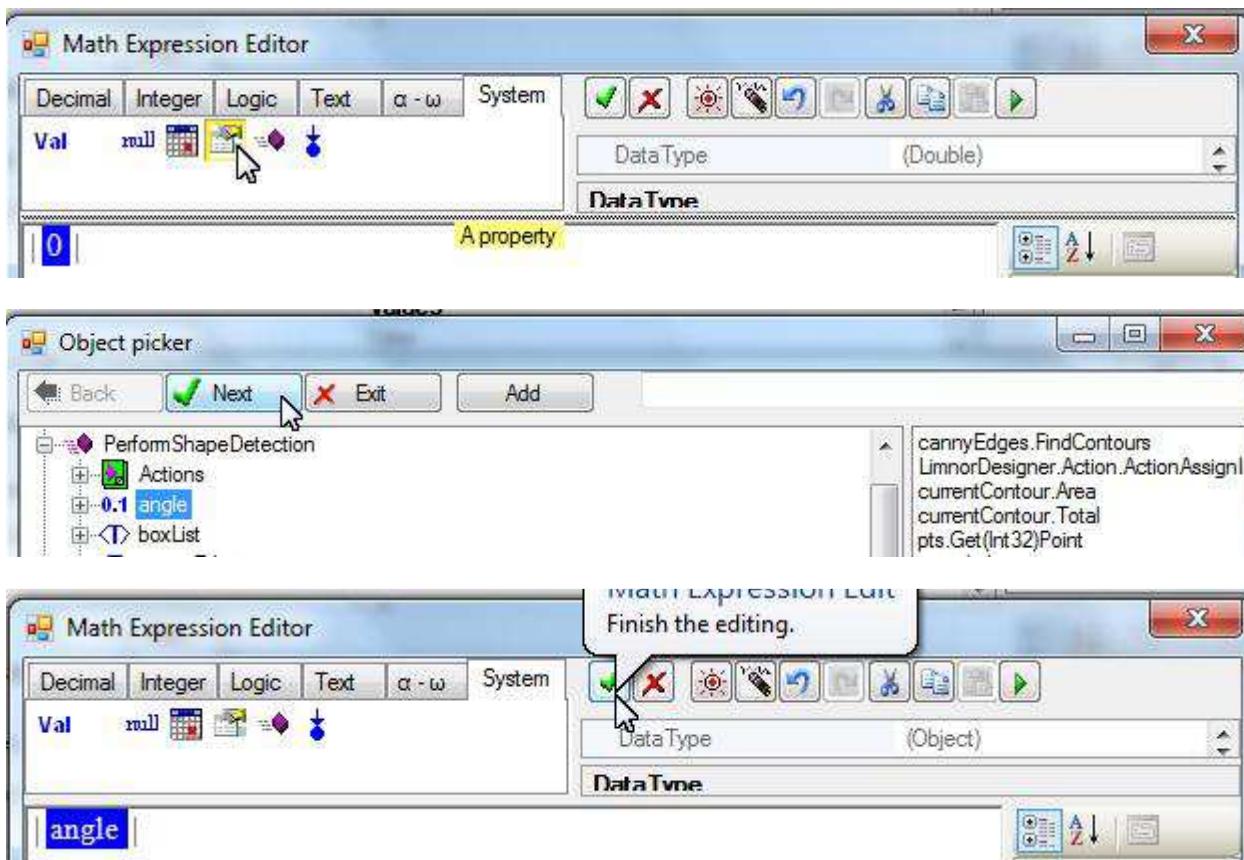
Use Generic Types and Native Libraries



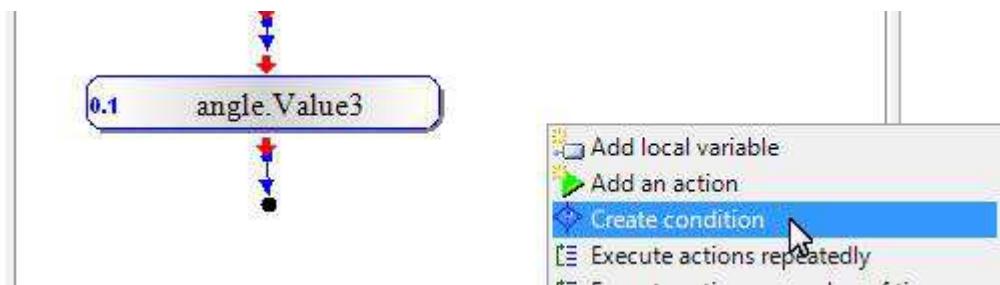
Make the angle into an absolute value:



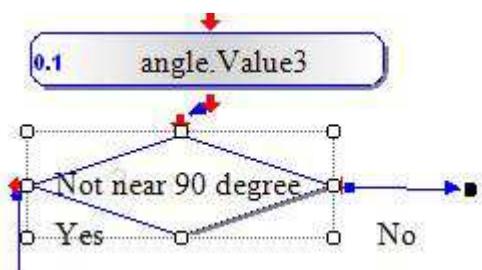
Use Generic Types and Native Libraries



Add a condition to check whether the angle is away from 90 degree too much, that is, the angle is smaller than 80 or larger 100:



Rename the condition "Not near 90 degree":



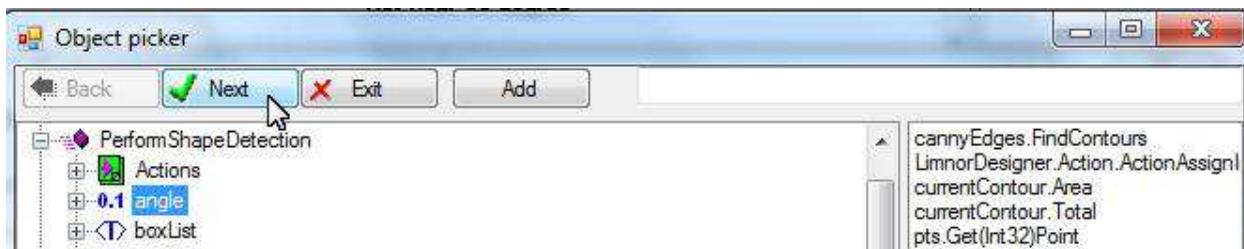
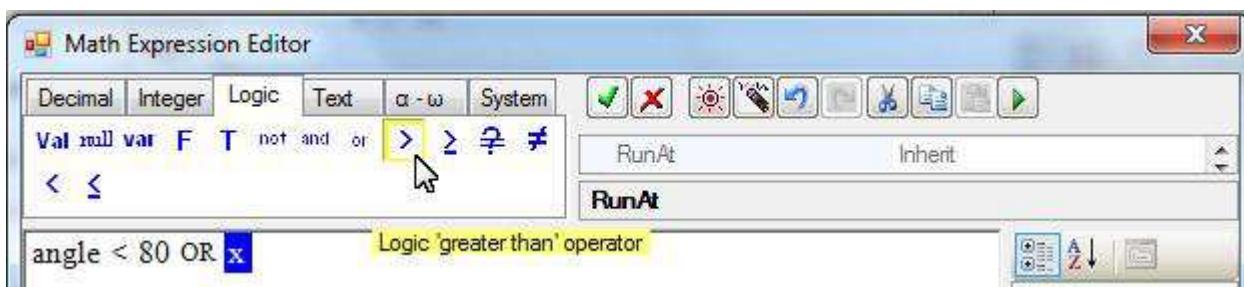
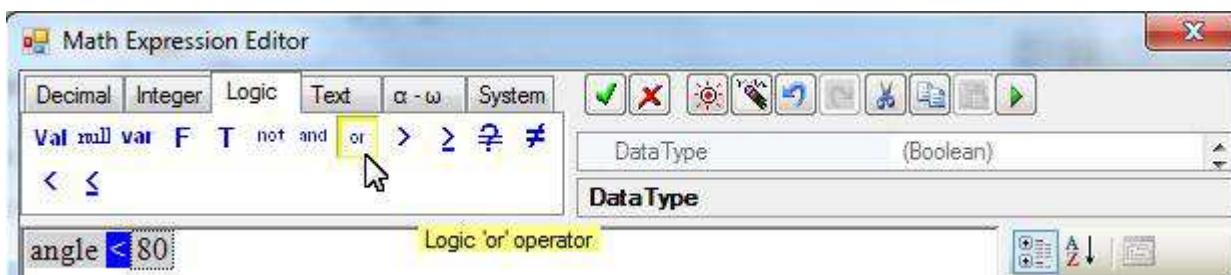
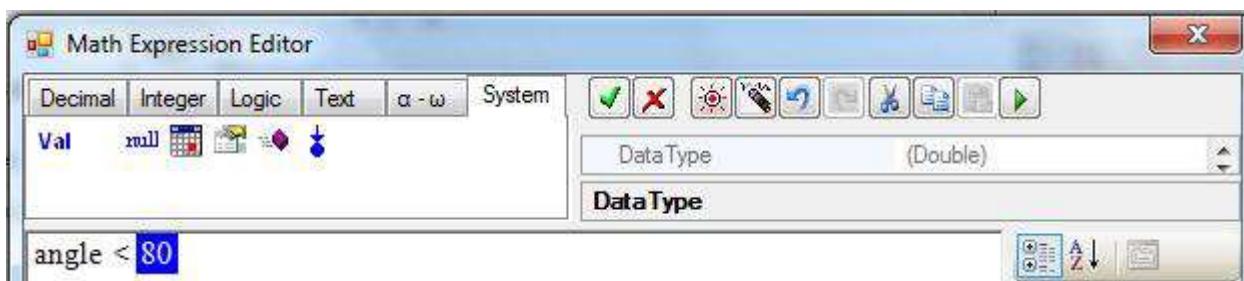
Set its condition:

Use Generic Types and Native Libraries

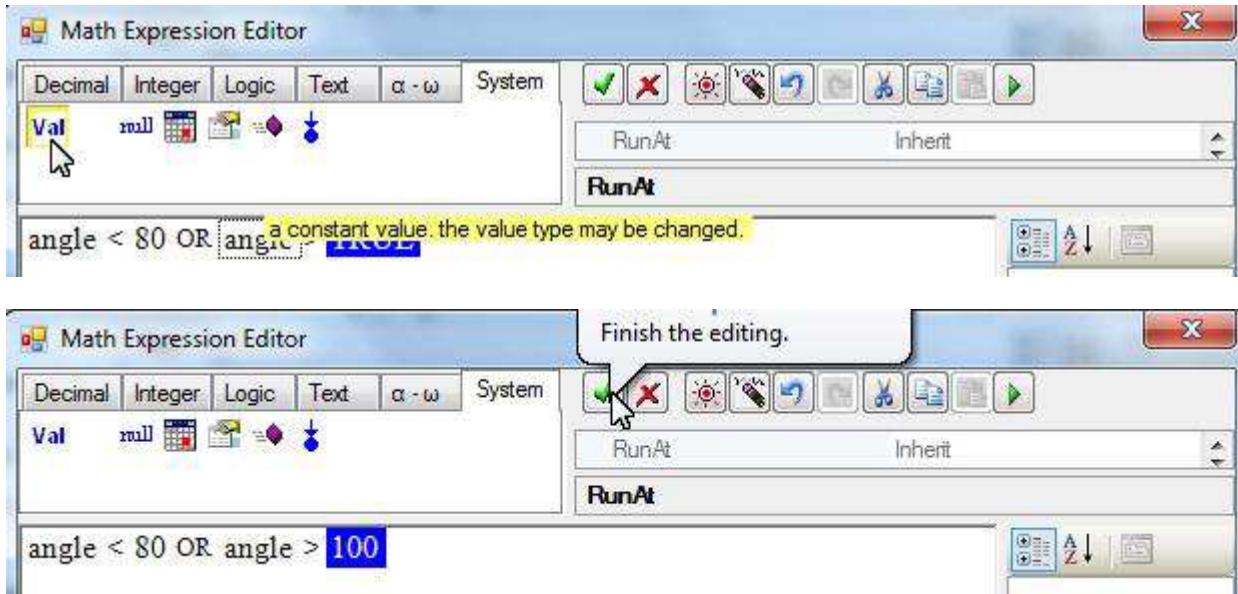
The screenshots illustrate the use of generic types and native libraries in a software environment, likely a programming or configuration tool.

- Screenshot 1:** A context menu is open over a "Condition" node in a tree view. The menu items include "False", "abc ConstantValue", "Property", and "x+y MathExpression".
- Screenshot 2:** The "Math Expression Editor" window is shown. The toolbar includes buttons for Decimal, Integer, Logic, Text, α - ω, and System. The logic section shows operators like Val, null, F, T, not, and, or, >, ≥, ≠, <, and ≤. The expression bar shows "0" and the tooltip "Logic less than operator".
- Screenshot 3:** The "Math Expression Editor" window again, showing the expression "TRUE < TRUE" and the tooltip "A property". The "DataType" dropdown is set to (Boolean).
- Screenshot 4:** The "Object picker" window is open, showing a tree structure with nodes like "PerformShapeDetection", "Actions", "0.1 angle", and "boxList". To the right, a list of properties is shown: cannyEdges.FindContours, LimnorDesigner.Action.ActionAssign, currentContour.Area, currentContour.Total, pts.Get(Int32)Point.
- Screenshot 5:** The "Math Expression Editor" window once more, showing the expression "angle < TRUE" and the tooltip "a constant value, the value type may be changed". The "DataType" dropdown is set to (Boolean).

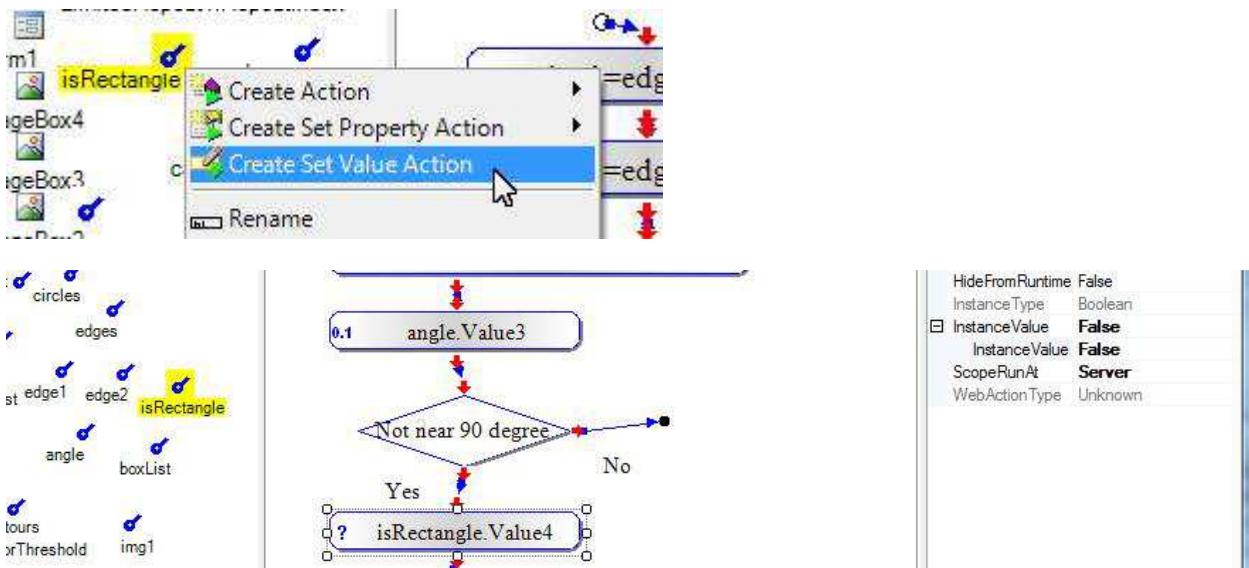
Use Generic Types and Native Libraries



Use Generic Types and Native Libraries

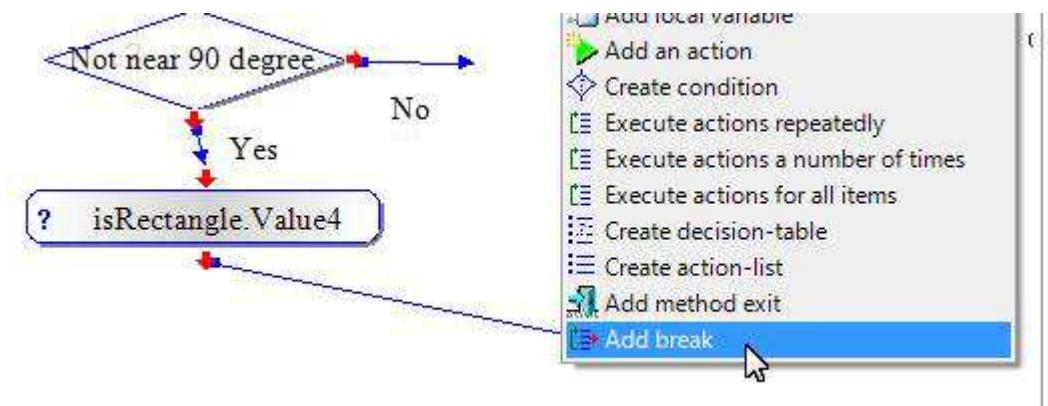


If the angle is away from 90 degree too much then set isRectangle to false and break the loop action.

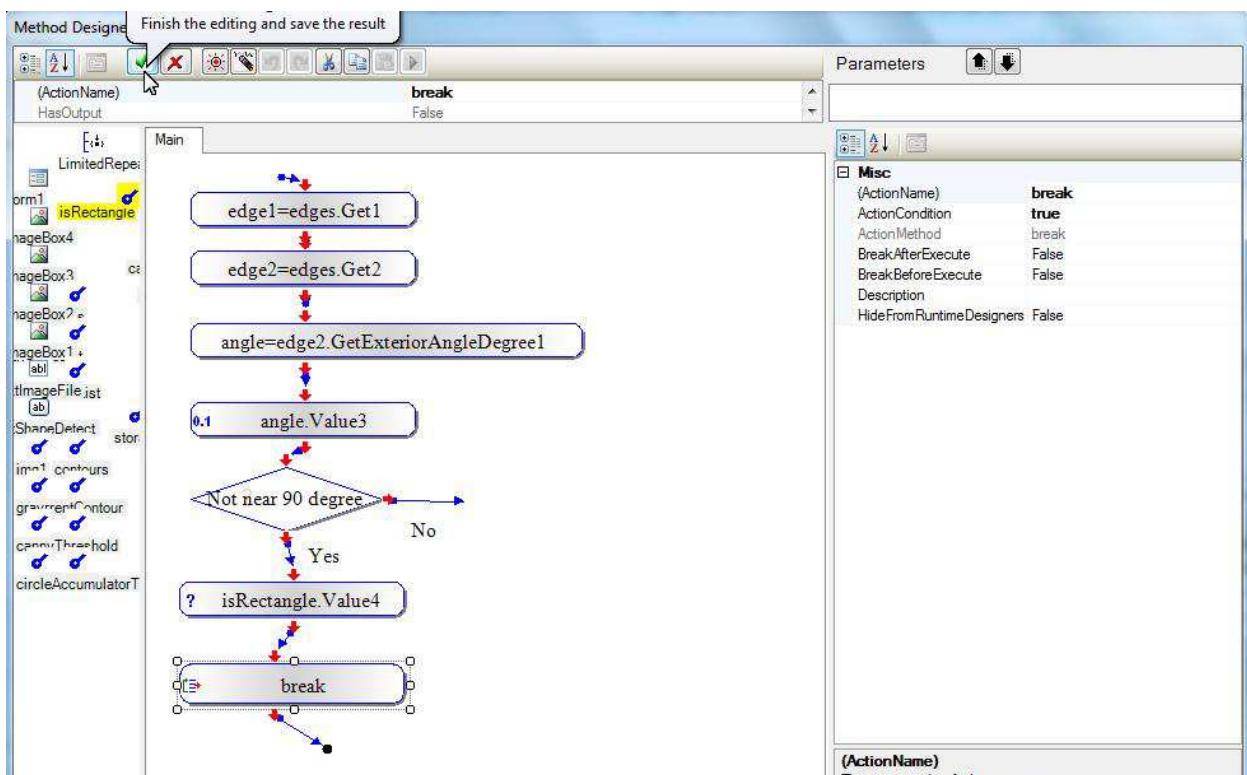


Break the loop action:

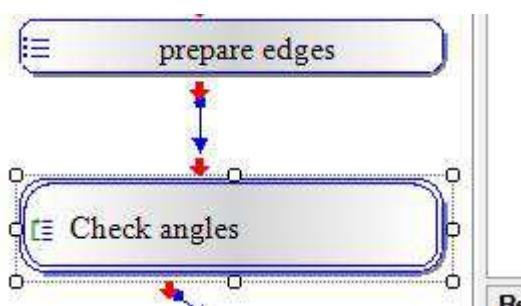
Use Generic Types and Native Libraries



These are the angle checking actions:

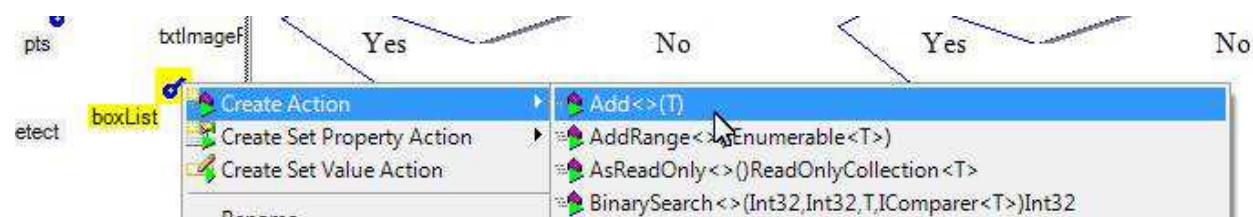


Change the loop action name to “check angles”:



If “Check angles” does not set isRectangle to false then add the contour to the rectangle list:

Use Generic Types and Native Libraries

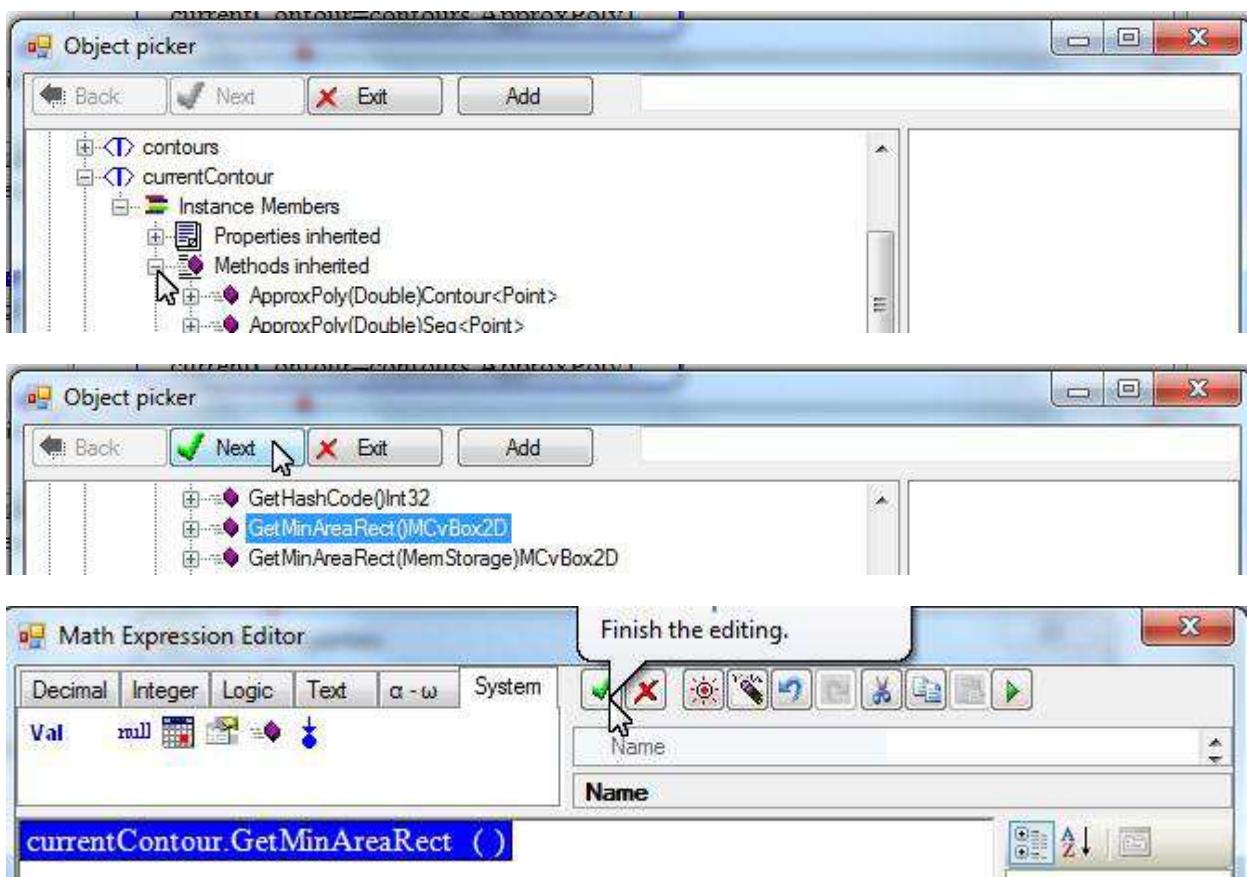


Use an expression to create a rectangle from the contour:

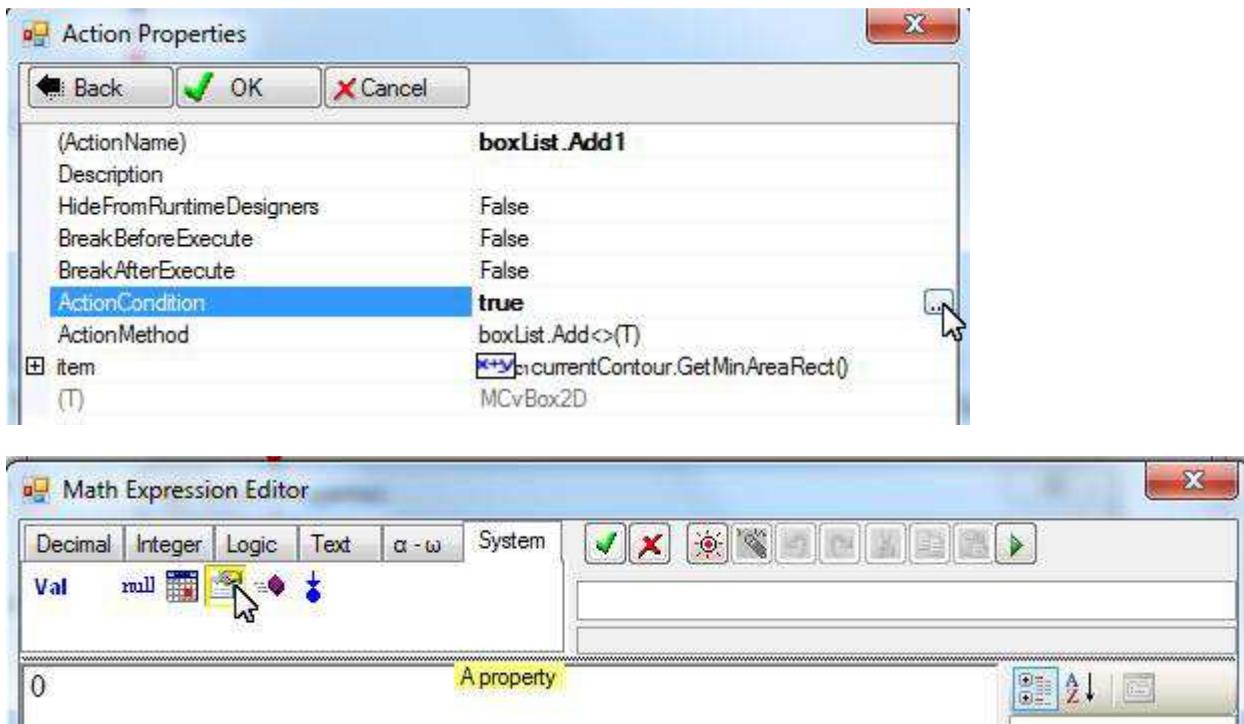
The image contains three screenshots illustrating the use of expressions:

- Action Properties Dialog:** Shows the 'boxList.Add1' action selected. The 'item' field is expanded, showing a dropdown menu with options like 'ConstantValue', 'Property', and 'MathExpression'. The 'MathExpression' option is highlighted.
- Math Expression Editor:** A separate window showing a toolbar with buttons for Decimal, Integer, Logic, Text, and System. Below the toolbar, there's a text input field with '0' and a status bar that says 'A method call'.
- Object picker Dialog:** Shows a list of objects: 'Form1 from DrawingPage', 'Primary types', 'PerformShapeDetection' (which is selected), and 'Actions'.

Use Generic Types and Native Libraries



Set the ActionCondition to isRectangle:



Use Generic Types and Native Libraries

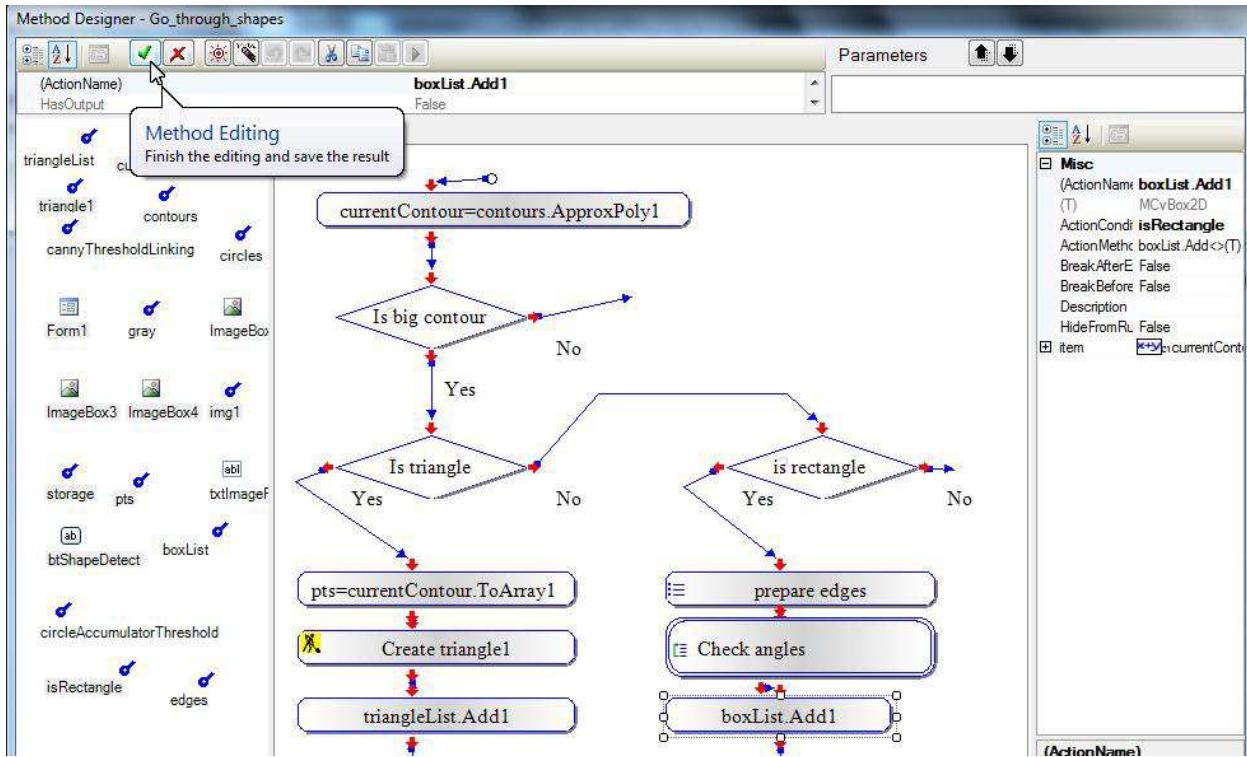
The image consists of three vertically stacked windows from a software application:

- Object picker**: A tree view of variables and actions. The node `isRectangle` is selected. The status bar shows the path: `currentContour.GetMinAreaRect()MC`.
- Math Expression Editor**: Shows the expression `isRectangle` being edited. A tooltip says "Finish the editing." The status bar shows `CanWrite False`. The toolbar includes icons for Decimal, Integer, Logic, Text, System, and various operators.
- Action Properties**: An action configuration window for `boxList.Add1`. The `ActionCondition` field is set to `isRectangle`. The `ActionMethod` field contains the code:

```
boxList.Add<>(T)
    <--> currentContour.GetMinAreaRect()
    MCvBox2D
```

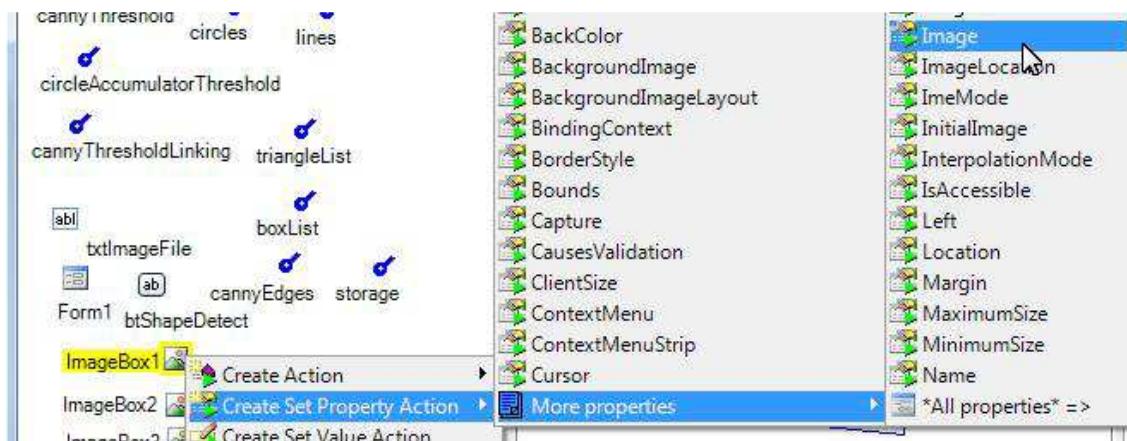
This completes the triangle and rectangle detection:

Use Generic Types and Native Libraries

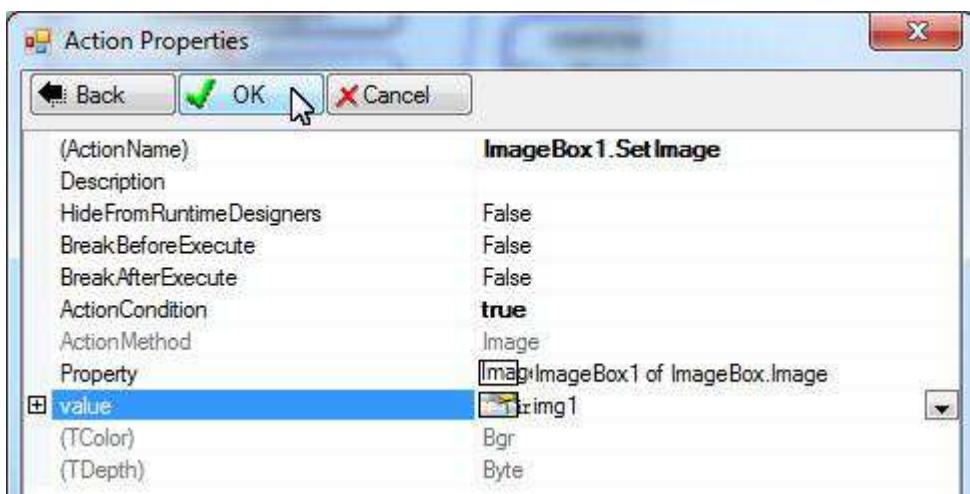
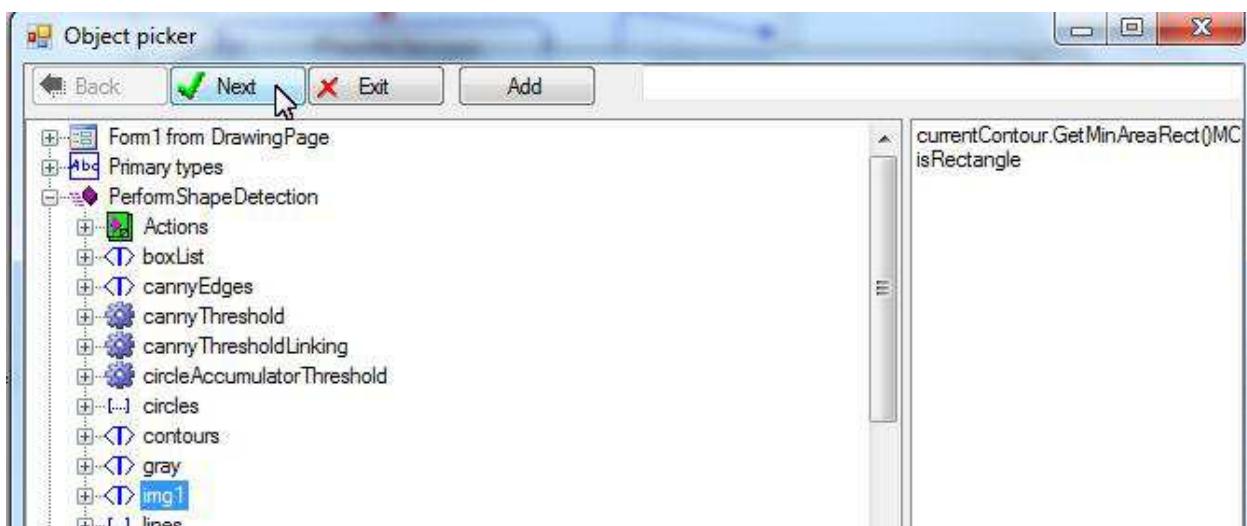
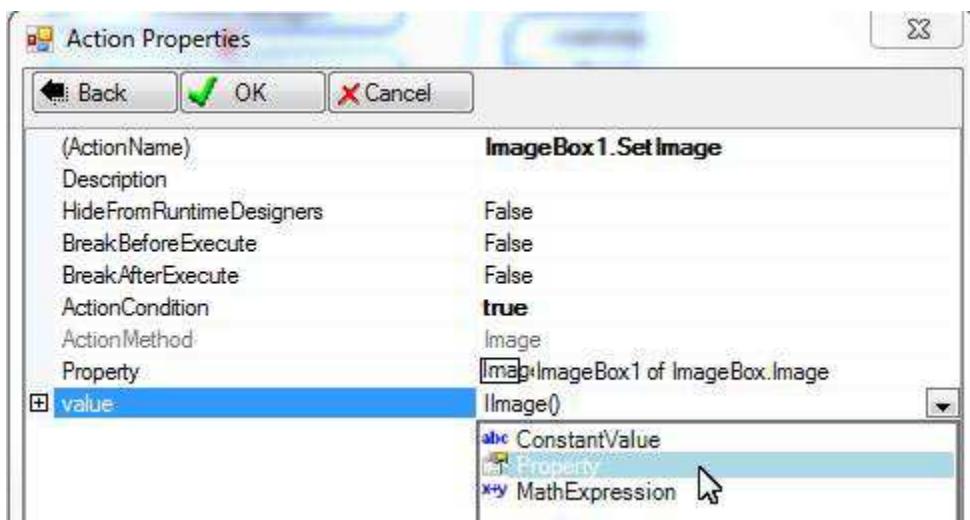


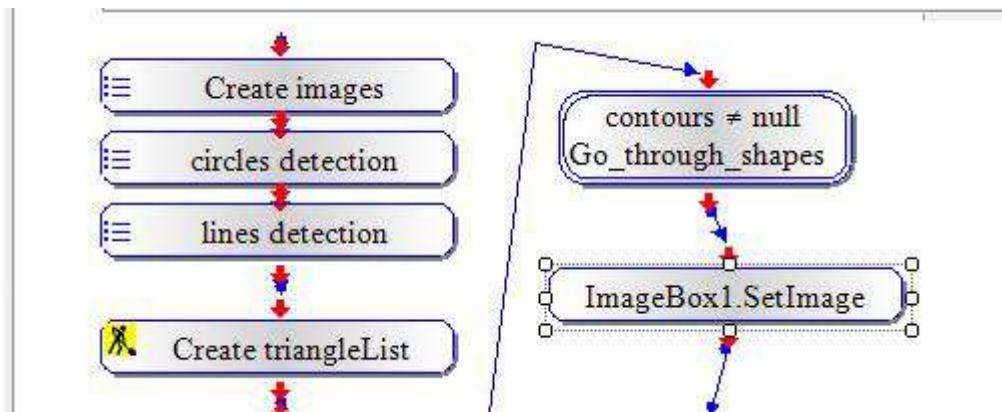
Show original image

Display the original image on the first image box.



Use Generic Types and Native Libraries



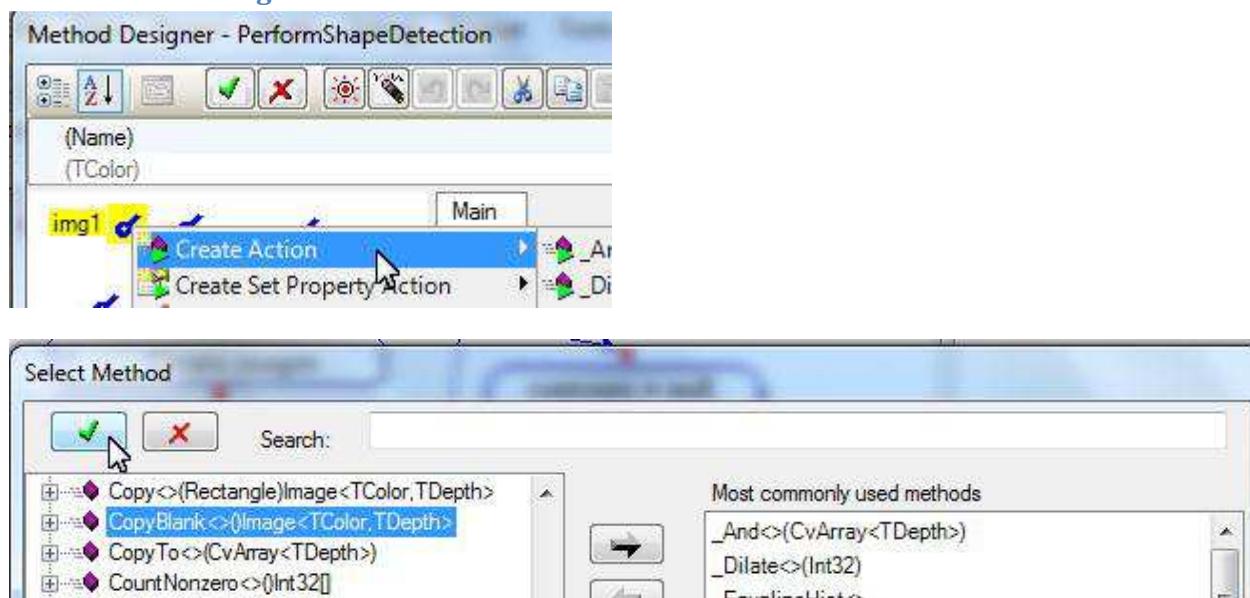


Display triangles and rectangles

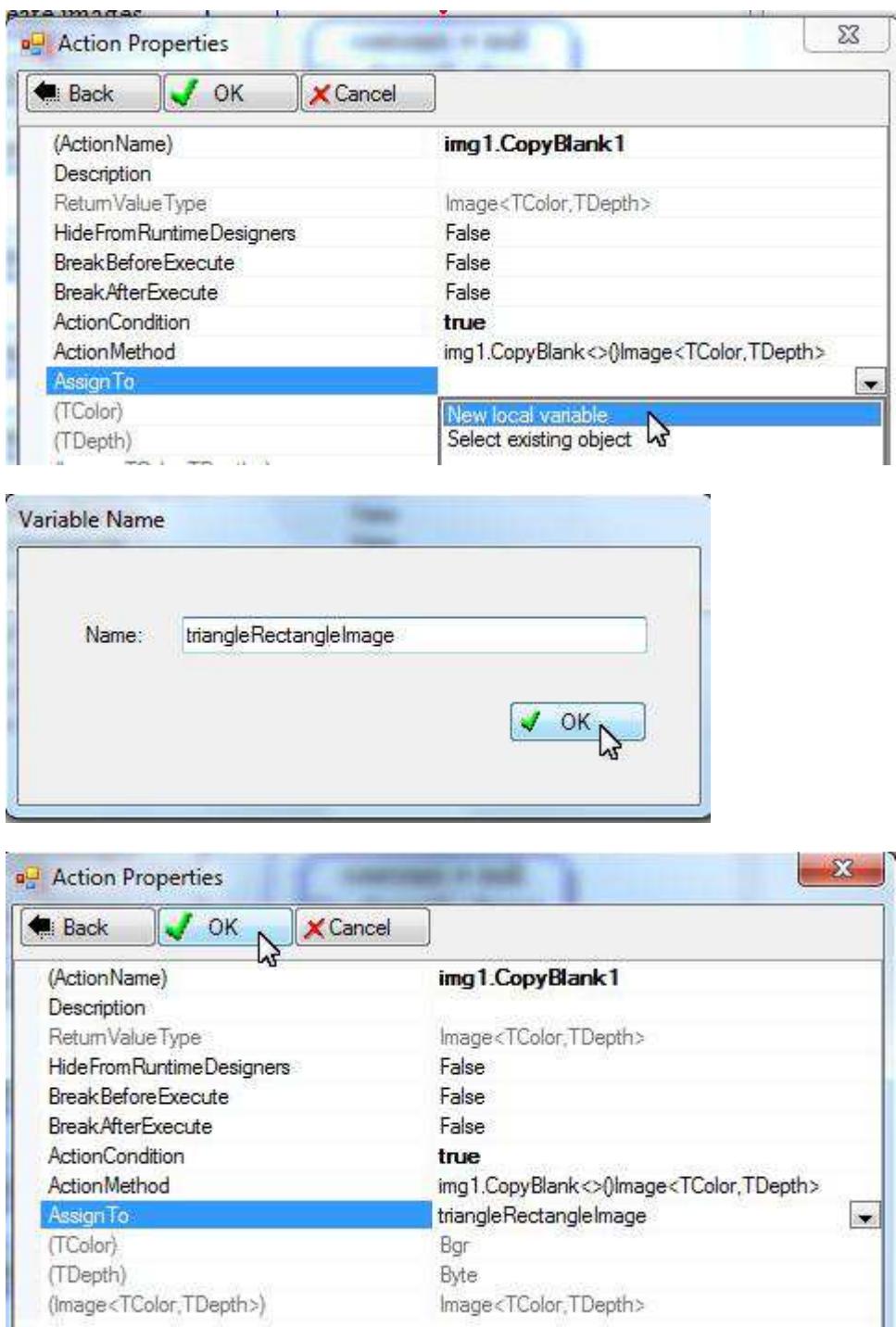
Triangles and rectangles are displayed on the second image box.

Create a blank image. Draw all the triangles and rectangles to the blank image. Assign the image to the second image box.

Create a blank image



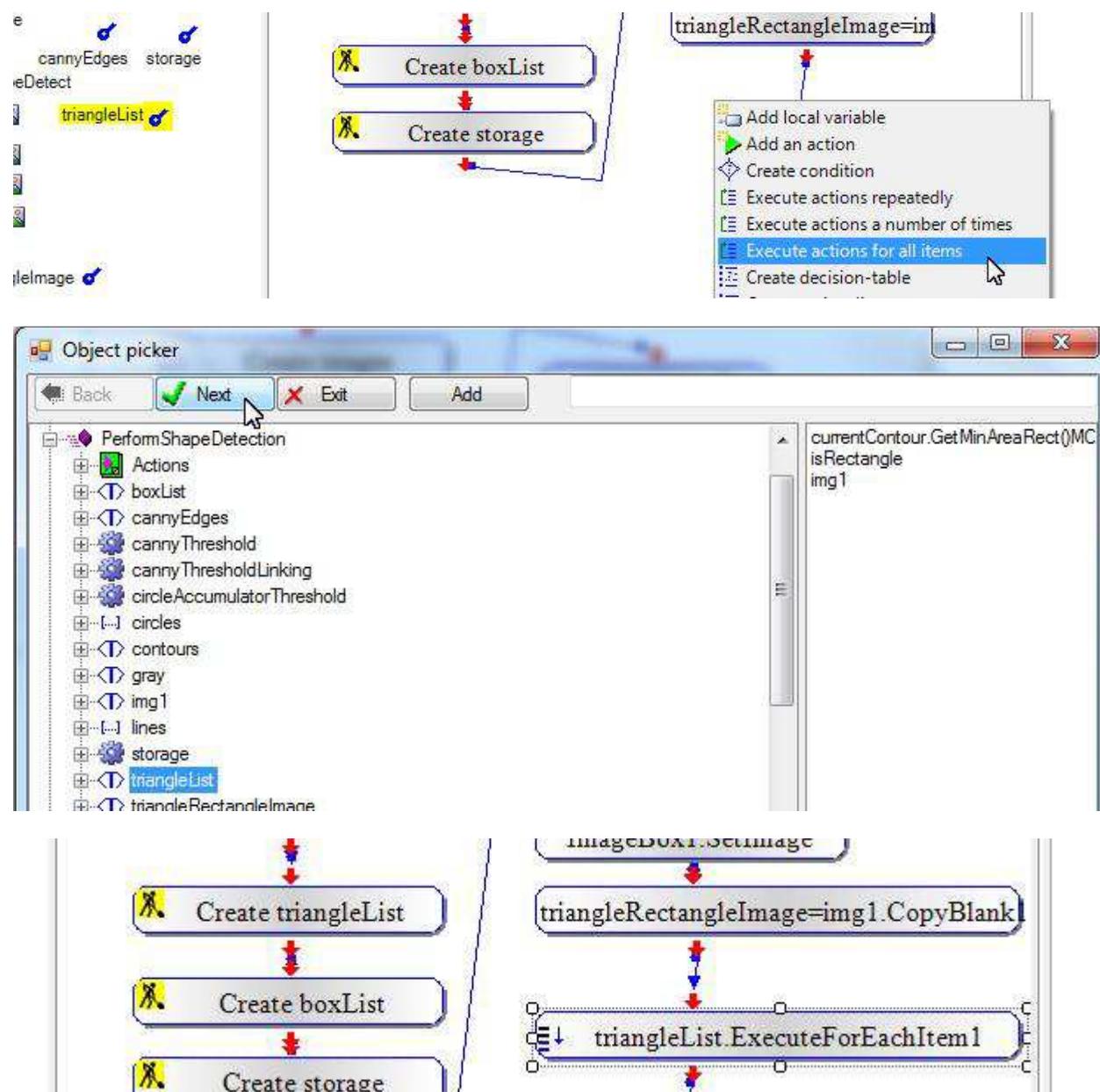
Use Generic Types and Native Libraries



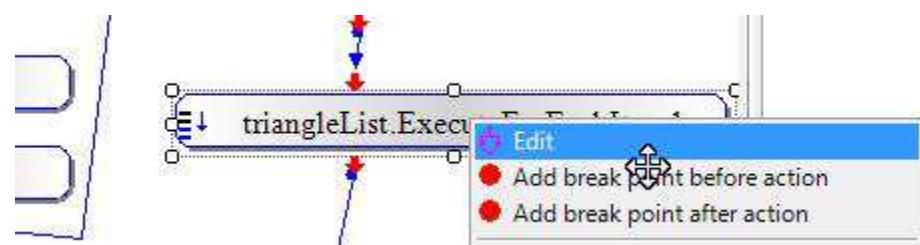
Go through all triangles

Create a loop action to go through all triangles.

Use Generic Types and Native Libraries

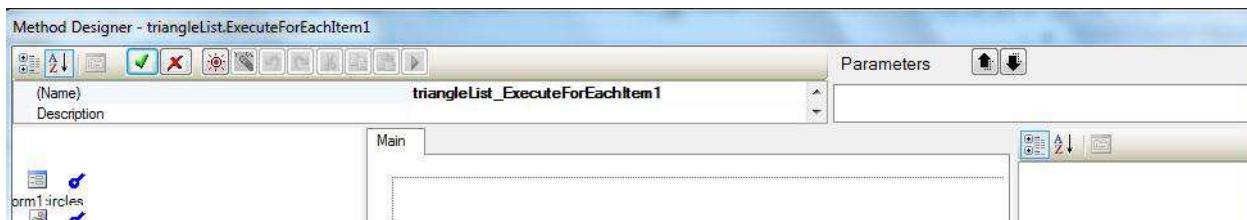


To show each triangle, right-click the loop action, select “Edit”:

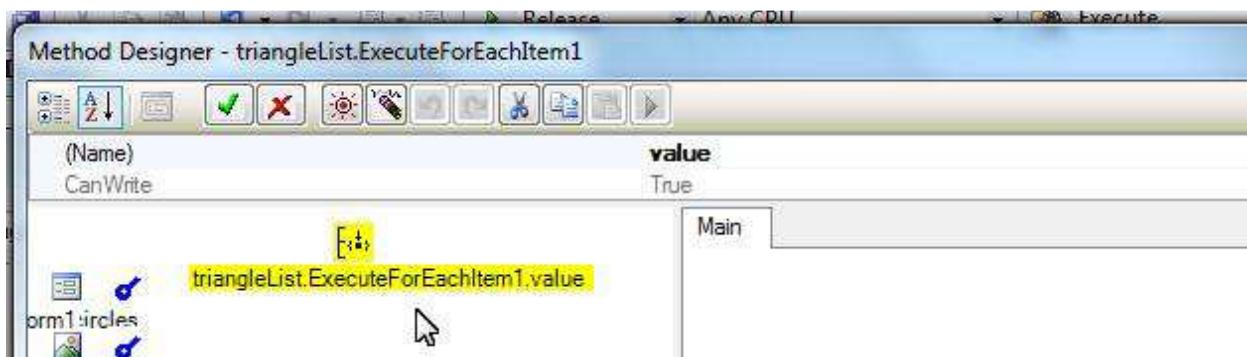


A new Method Editor appears.

Use Generic Types and Native Libraries



Note the “value” icon. It represents the triangle to be used. We’ll draw it to the blank image.



Right-click the blank image named “triangleRectangleImage”; select its Draw method.



Use Generic Types and Native Libraries

The screenshot illustrates the configuration of an action named "triangleRectangleImage.Draw1" using the Action Properties dialog and the Object picker.

Action Properties Dialog:

- (ActionName) triangleRectangleImage.Draw1
- Description
- HideFromRuntimeDesigners False
- BreakBeforeExecute False
- BreakAfterExecute False
- ActionCondition true
- ActionMethod triangleRectangleImage.Draw<>(IConvexPolygonF)
- polygon**: A dropdown menu showing options like ConstantValue, Property, and MathExpression. The "Property" option is selected.
- color**: A dropdown menu showing options like ConstantValue, Property, and MathExpression. The "Property" option is selected.
- thickness**: A dropdown menu showing options like ConstantValue, Property, and MathExpression. The "Property" option is selected.
- (TColor): A dropdown menu showing options like ConstantValue, Property, and MathExpression. The "Property" option is selected.

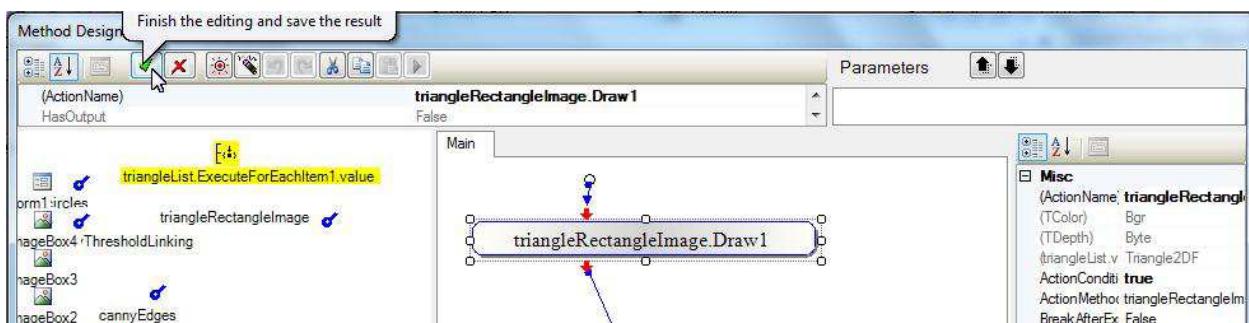
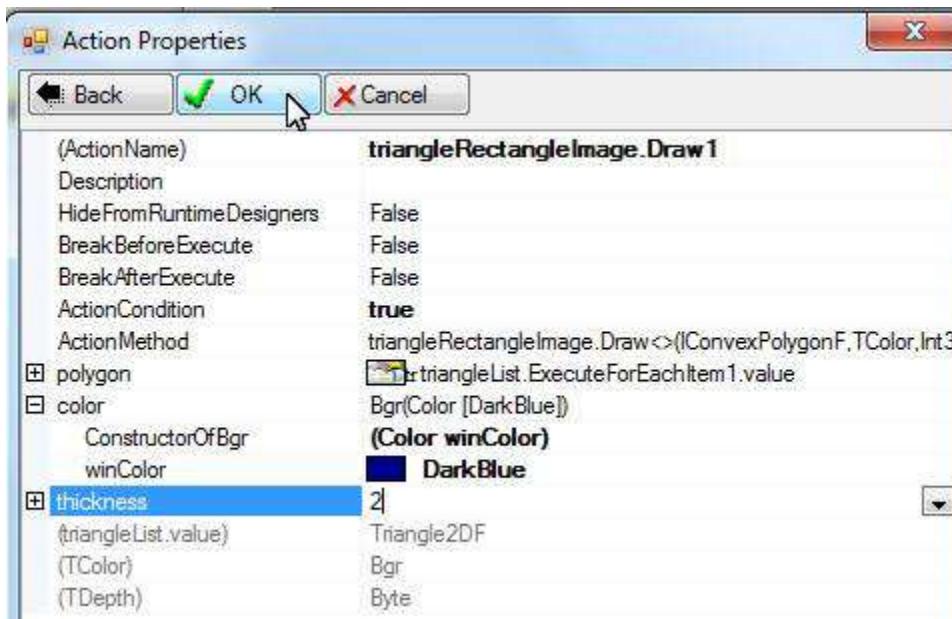
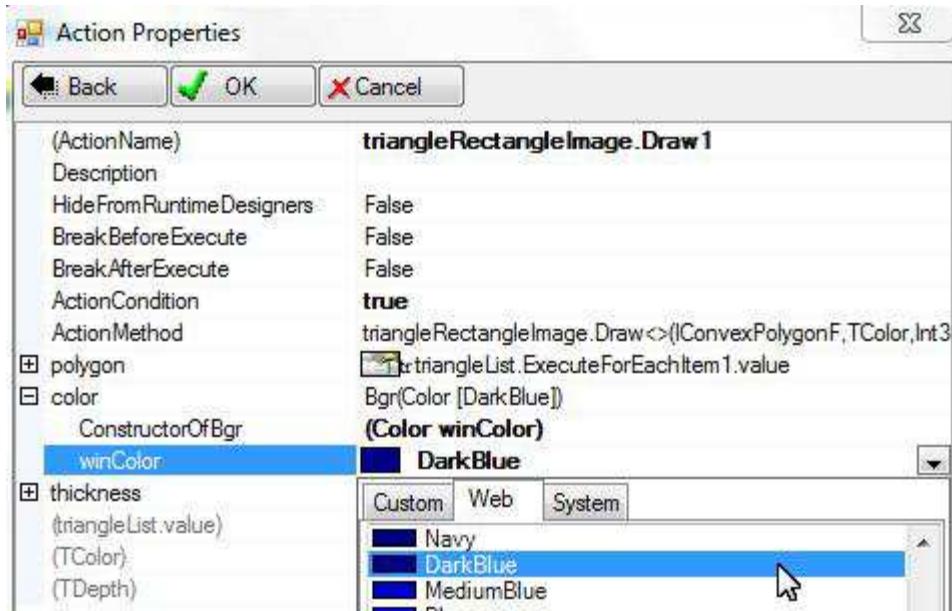
Object picker Dialog:

- Shows a tree view of objects under "PerformShapeDetection".
 - Actions
 - boxList
 - cannyEdges
 - cannyThreshold
 - cannyThresholdLinking
 - circleAccumulatorThreshold
 - circles
 - contours
 - gray
 - img1
 - lines
 - storage
 - triangleList
- Shows a list of variables:
 - currentContour.GetMinAreaRect()MC
 - isRectangle
 - img1
 - triangleList
- triangleList.ExecuteForEachItem1.value**: This item is highlighted in the list.

Action Properties Dialog (Continued):

- (ActionName) triangleRectangleImage.Draw1
- Description
- HideFromRuntimeDesigners False
- BreakBeforeExecute False
- BreakAfterExecute False
- ActionCondition true
- ActionMethod triangleRectangleImage.Draw<>(IConvexPolygonF, TColor, Int3)
- polygon**: A dropdown menu showing options like ConstantValue, Property, and MathExpression. The "Property" option is selected.
- color**: A dropdown menu showing options like ConstantValue, Property, and MathExpression. The "Property" option is selected. The "ConstructorOfBgr" option is also visible.
- thickness**: A dropdown menu showing options like ConstantValue, Property, and MathExpression. The "Property" option is selected. The value "(Double blue, Double green, Double red)" and "(Color winColor)" are listed.
- (TColor): A dropdown menu showing options like ConstantValue, Property, and MathExpression. The "Property" option is selected.

Use Generic Types and Native Libraries



Go through all rectangles

Create a loop action to go through all rectangles.

Use Generic Types and Native Libraries

The screenshot displays a software interface for programmatic image processing, likely using OpenCV or a similar library.

Code Snippet: triangleRectangleImage=img1.CopyBlank

Object Picker: Shows a tree structure under "PerformShapeDetection" with nodes: Actions, boxList, cannyEdges, and cannyThreshold. To the right is a list of variables: currentContour.GetMinAreaRect()MC, isRectangle, img1, triangleList, and triangleList.value.

Method Designer - boxList.ExecuteForEachItem1: A screenshot of the "Method Designer" window for the action "boxList.ExecuteForEachItem1". The action parameters are set to "triangleRectangleImage" and "Bgr". The "Create Action" button is highlighted.

Select Method: A dialog box showing methods for "boxList.ExecuteForEachItem1": Draw<>(Ellipse, TColor, Int32), Draw<>(ConvexPolygonF, TColor, Int32), and Draw<>(LineSegment2D, TColor, Int32). The "Draw<>(ConvexPolygonF, TColor, Int32)" method is selected.

Use Generic Types and Native Libraries

The image consists of three vertically stacked windows from a software interface.

Top Window: Action Properties

(ActionName) triangleRectangleImage.Draw2
Description
HideFromRuntimeDesigners False
BreakBeforeExecute False
BreakAfterExecute False
ActionCondition true
ActionMethod triangleRectangleImage.Draw<>(IConvexPolygonF, TColor, Int32)
polygon IConvexPolygonF()
+ color abc ConstantValue
+ thickness Property
(TColor) MathExpression
(TDepth)

Middle Window: Object picker

Back Next Exit Add

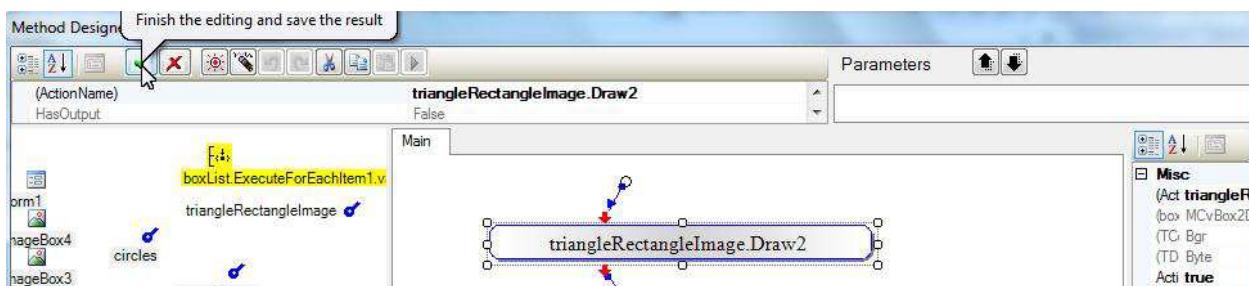
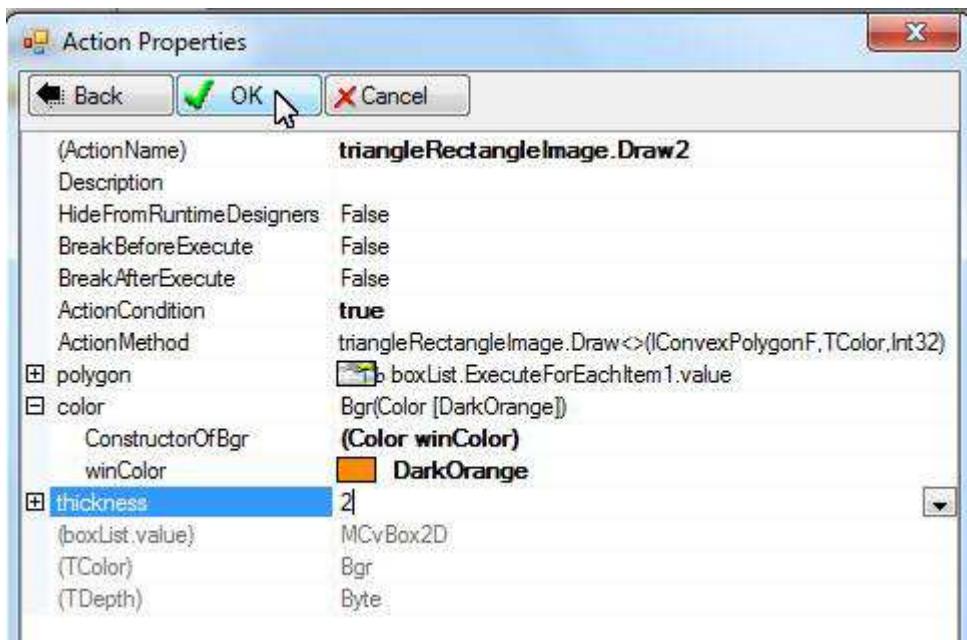
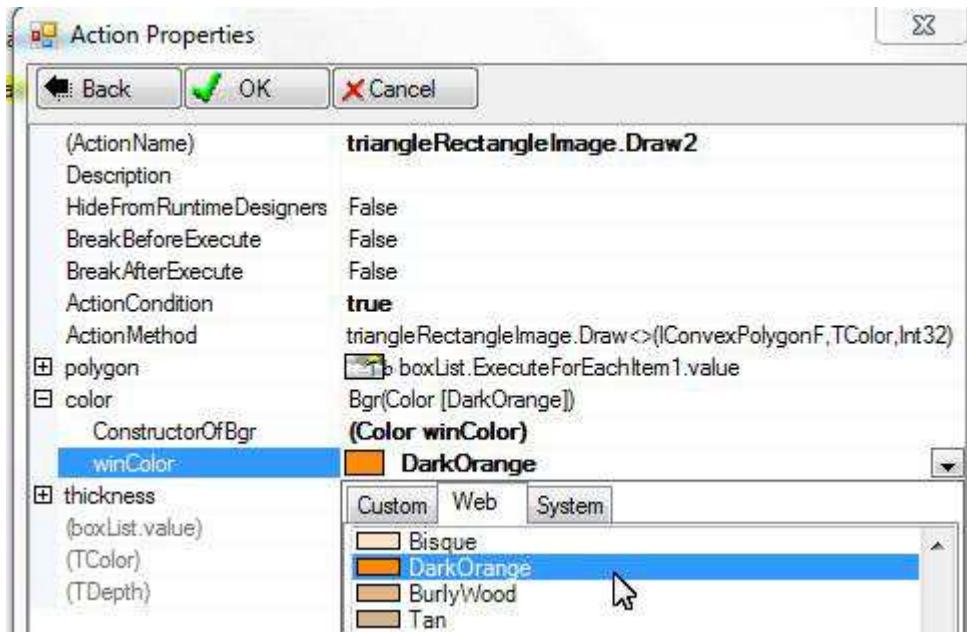
Form1 from DrawingPage
Primary types
PerformShapeDetection
Actions
boxList
boxList.ExecuteForEachItem1.value
cannyEdges

currentContour.GetMinAreaRect(MC)
isRectangle
img1
triangleList
triangleList.value
boxList

Bottom Window: Action Properties

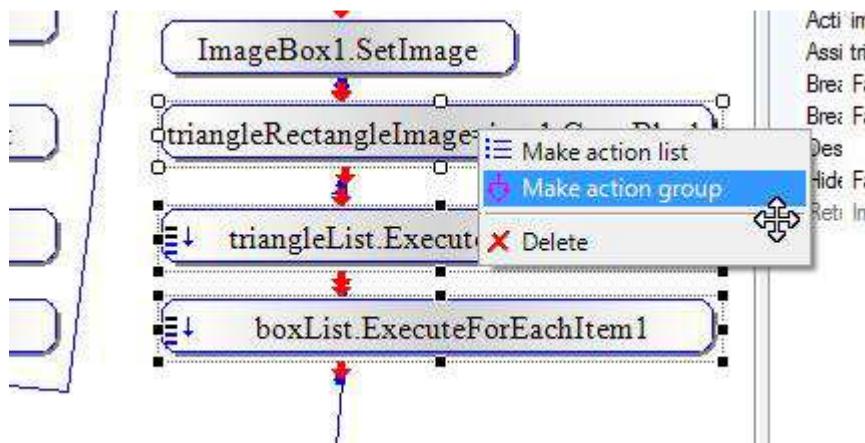
(ActionName) triangleRectangleImage.Draw2
Description
HideFromRuntimeDesigners False
BreakBeforeExecute False
BreakAfterExecute False
ActionCondition true
ActionMethod triangleRectangleImage.Draw<>(IConvexPolygonF, TColor, Int32)
polygon boxList.ExecuteForEachItem1.value
color Bgr()
ConstructorOfBgr
(Double blue, Double green, Double red)
(Color winColor)

Use Generic Types and Native Libraries

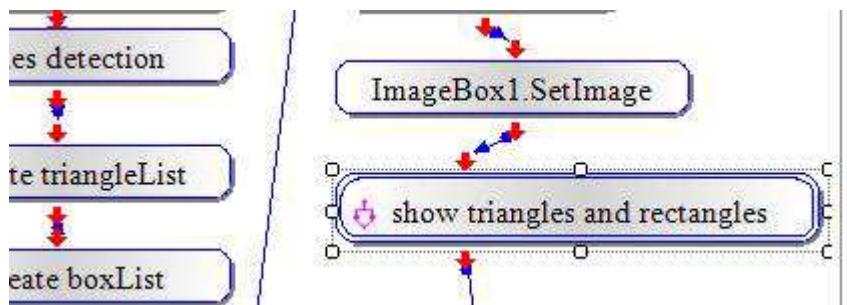


Make action group

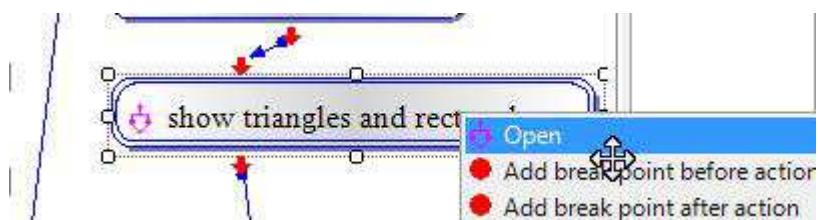
We may group the actions to make room in the editor. This step is optional.



Name the action group “show triangles and rectangles”:

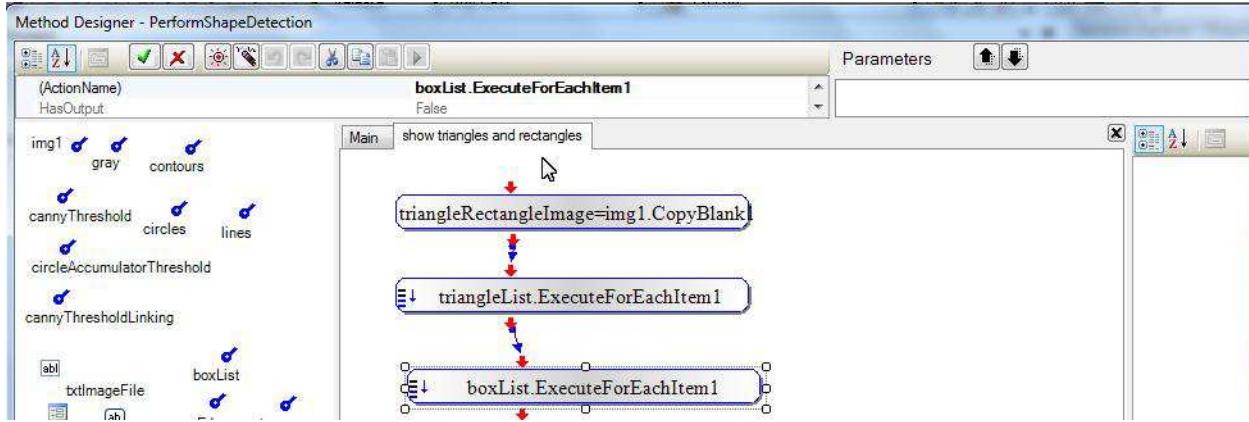


To modify the action group, right-click it and choose “Open”:



A new tab page is created to show the action group:

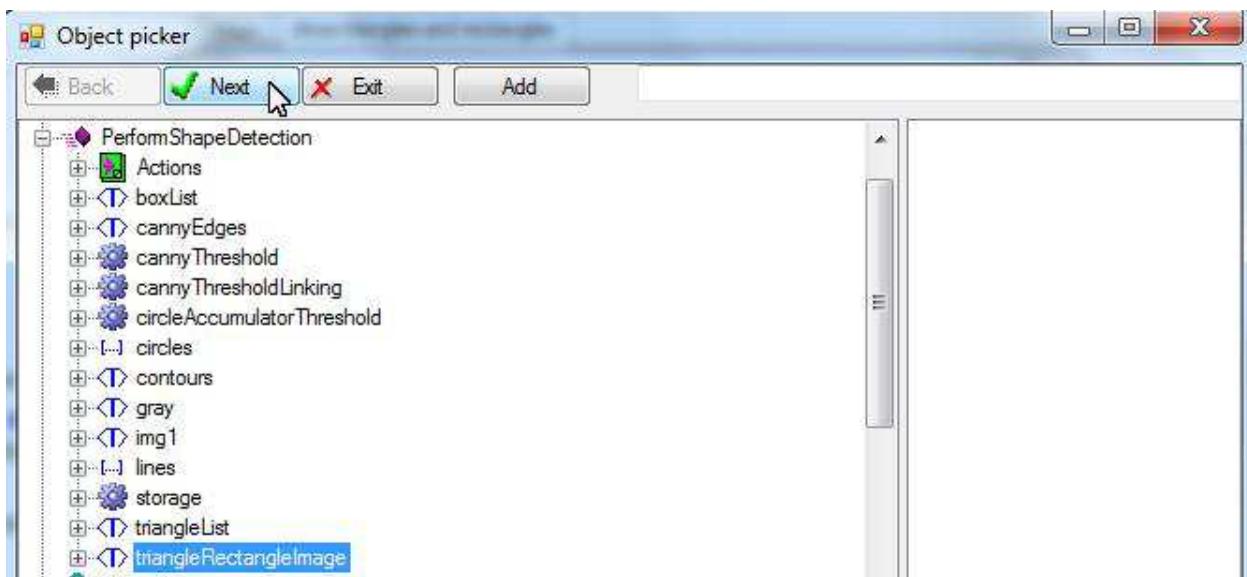
Use Generic Types and Native Libraries



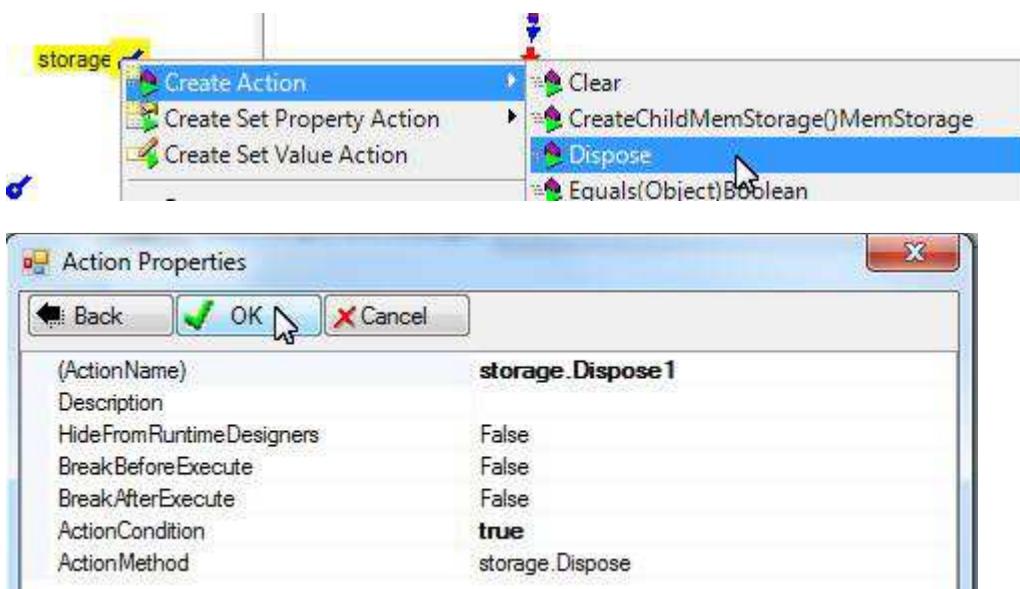
Add an action to display the image on the second image box:

This screenshot shows the 'Action Properties' dialog for an action named 'ImageBox2.SetImage'. The 'Property' section is expanded, showing the 'value' field. The value is currently set to 'Image' and is highlighted. A context menu is open over the 'value' field, with the 'ImageBox2' item selected. A list of properties for 'ImageBox2' is shown, including 'AutoSize', 'BackColor', 'BackgroundImage', etc. The 'Image' option is highlighted in blue. The 'OK' button is visible at the bottom of the dialog.

Use Generic Types and Native Libraries

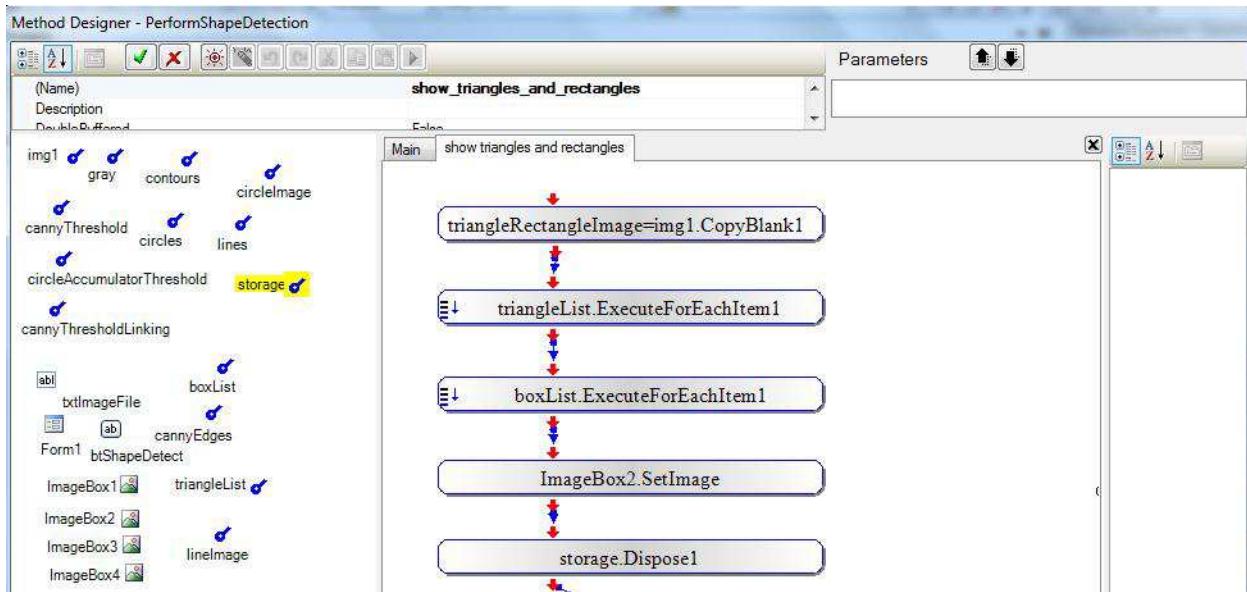


By now the storage object is no longer needed. We may call its Dispose method to free the memory it takes.

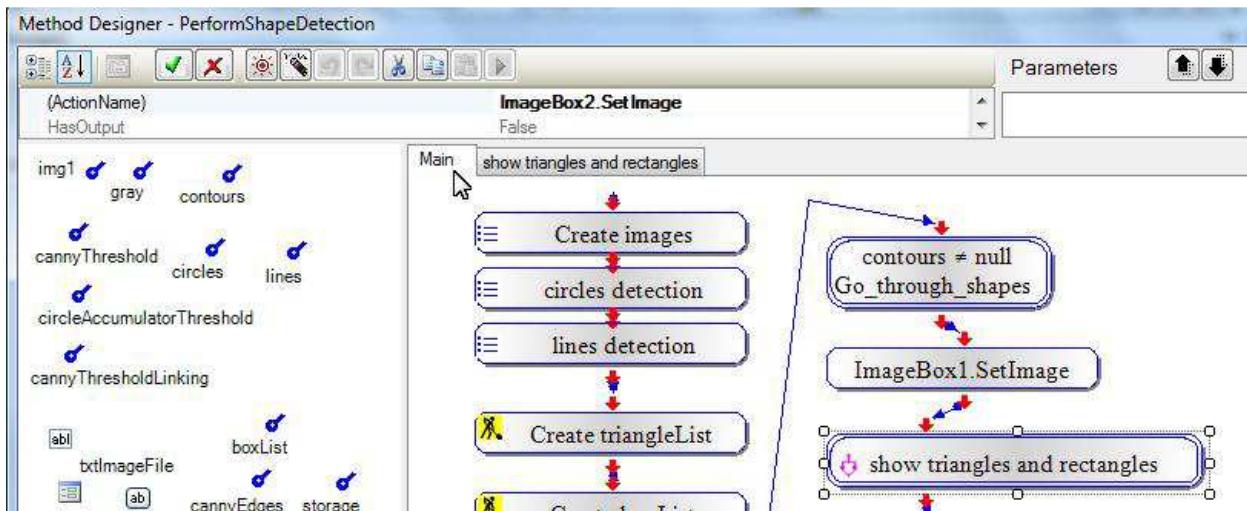


These are the actions for showing triangles and rectangles on the second image box:

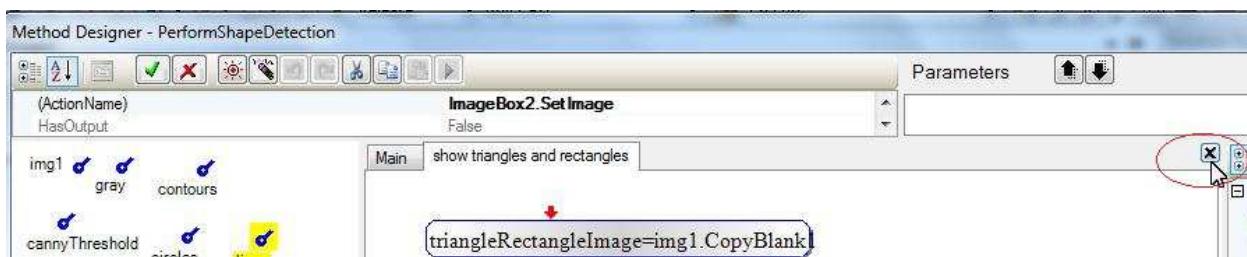
Use Generic Types and Native Libraries



We main switch back to the main routine:



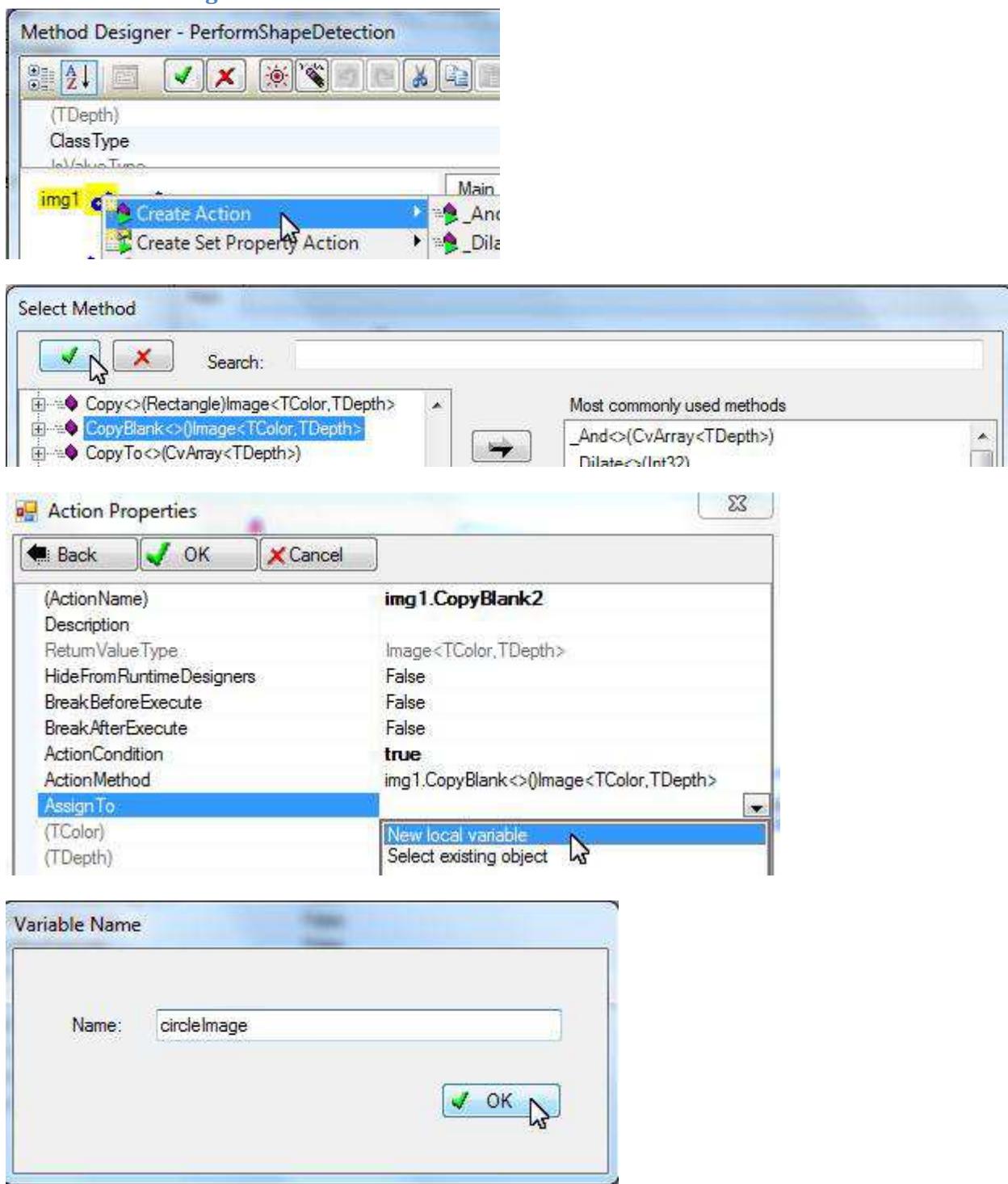
Or close the action group:



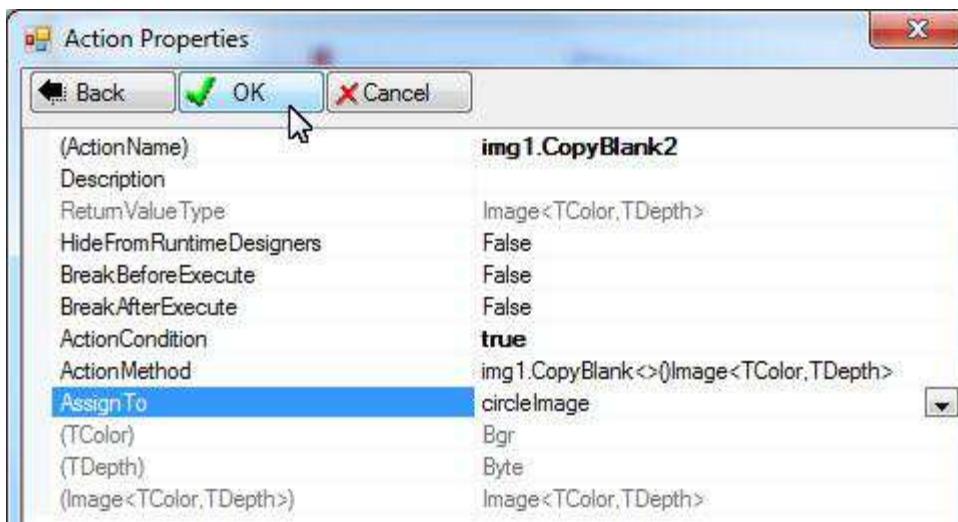
Display Circles

We'll follow the same action pattern: create a blank image; drawing all circles on the image; set the image to an image box.

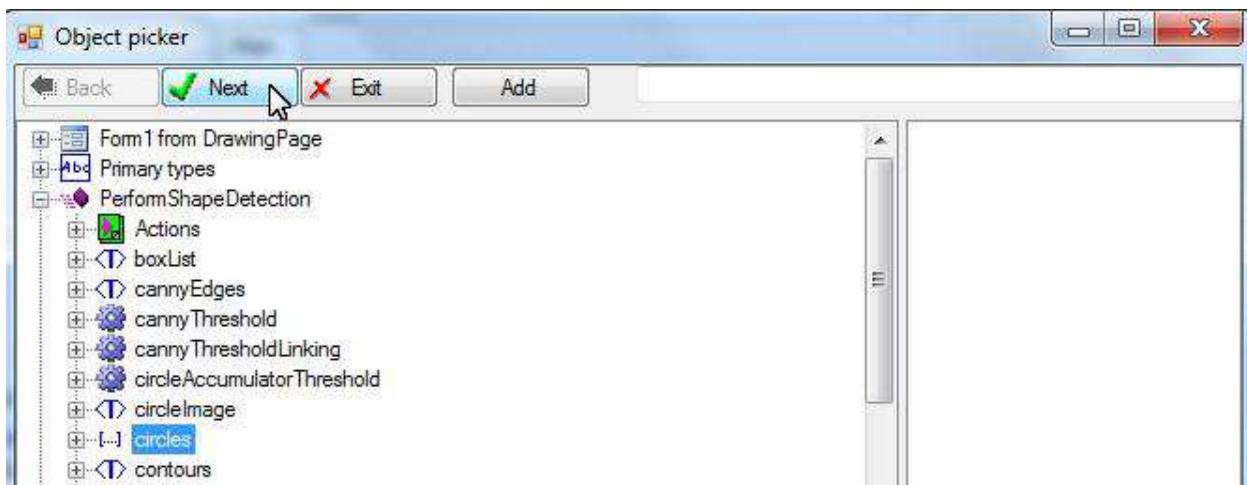
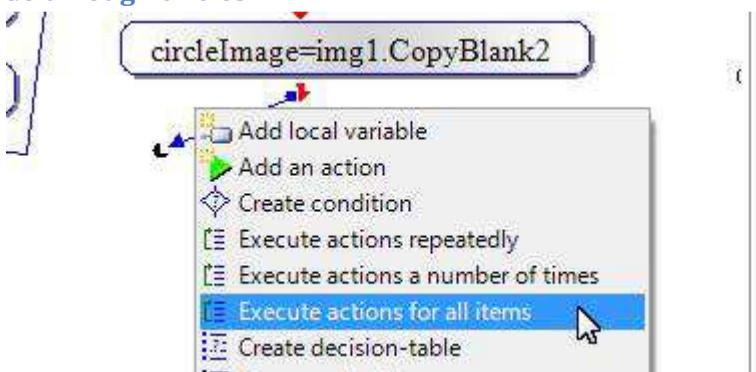
Create a blank image



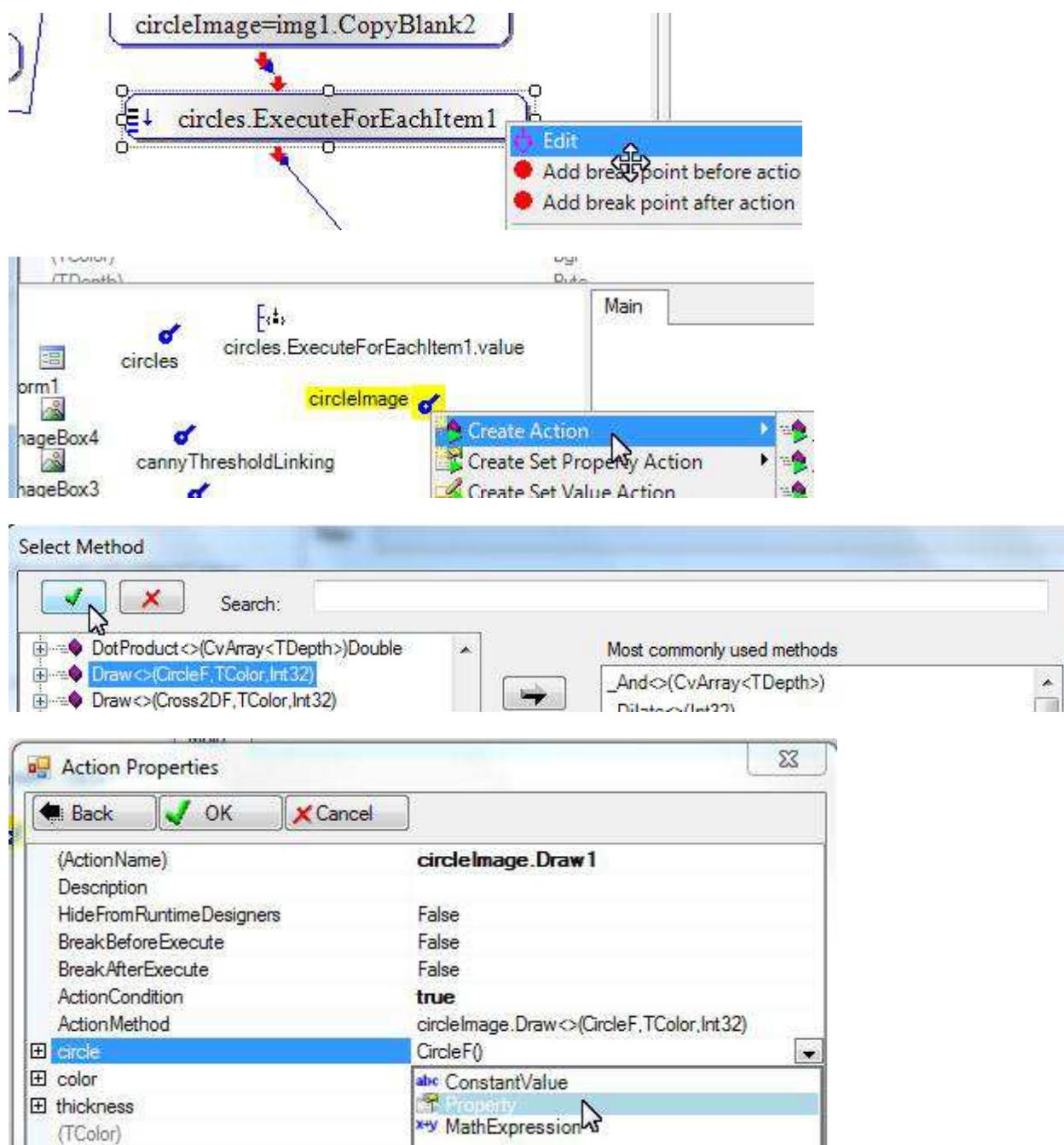
Use Generic Types and Native Libraries



Go through circles



Use Generic Types and Native Libraries



Use Generic Types and Native Libraries

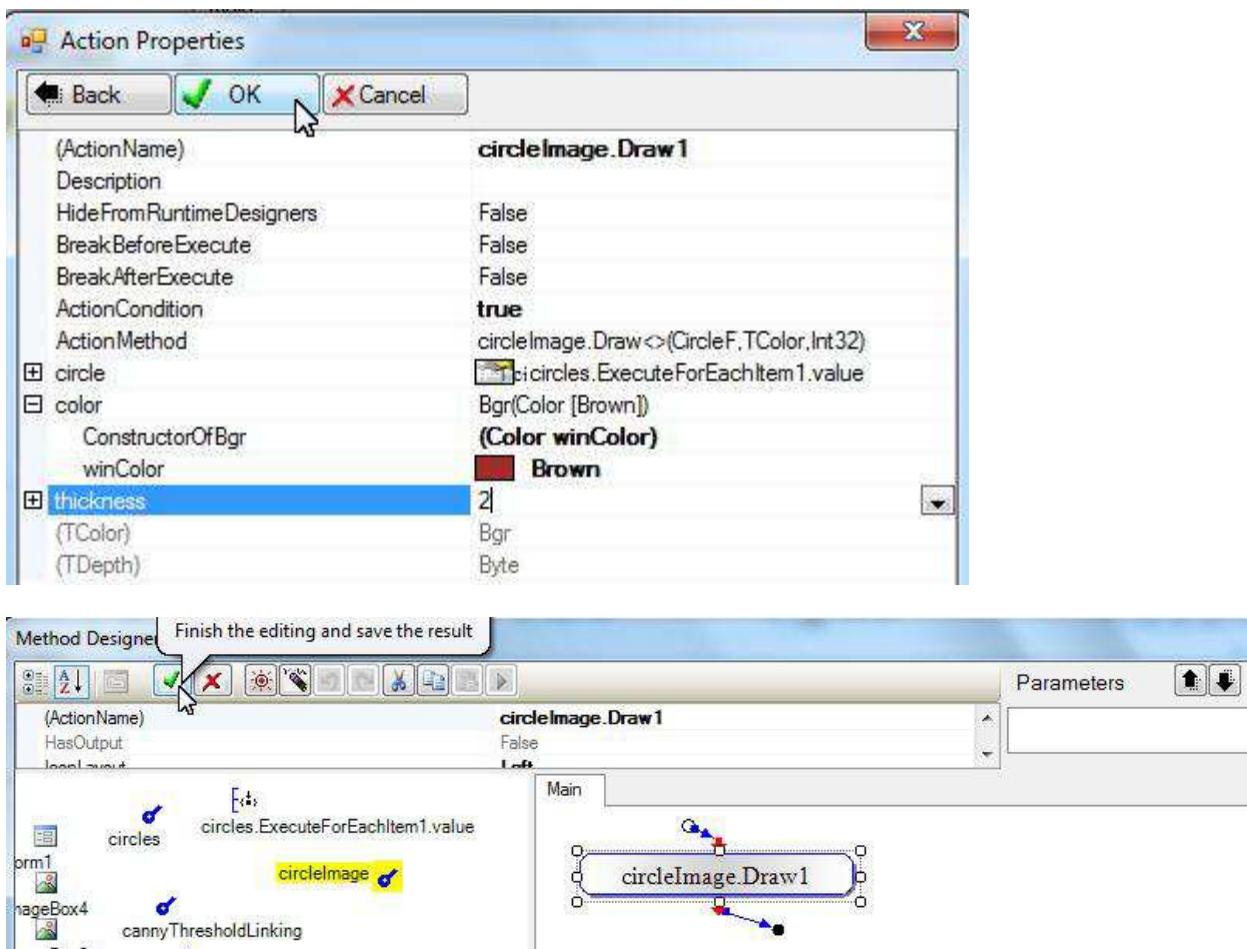
The image consists of three vertically stacked windows from a software interface, likely a visual programming environment.

Object picker window: Shows a tree view of actions under 'PerformShapeDetection'. The 'circleImage' node is expanded, and its child 'circleImage.Draw1' is selected. A list box on the right shows the variable 'circles'.

Action Properties window (top): Shows the properties for 'circleImage.Draw1'. The 'color' property is set to 'Bgr'. The 'ConstructorOfBgr' dropdown is open, showing '(Double blue,Double green,Double red)' and '(Color winColor)'. The '(Color winColor)' option is selected.

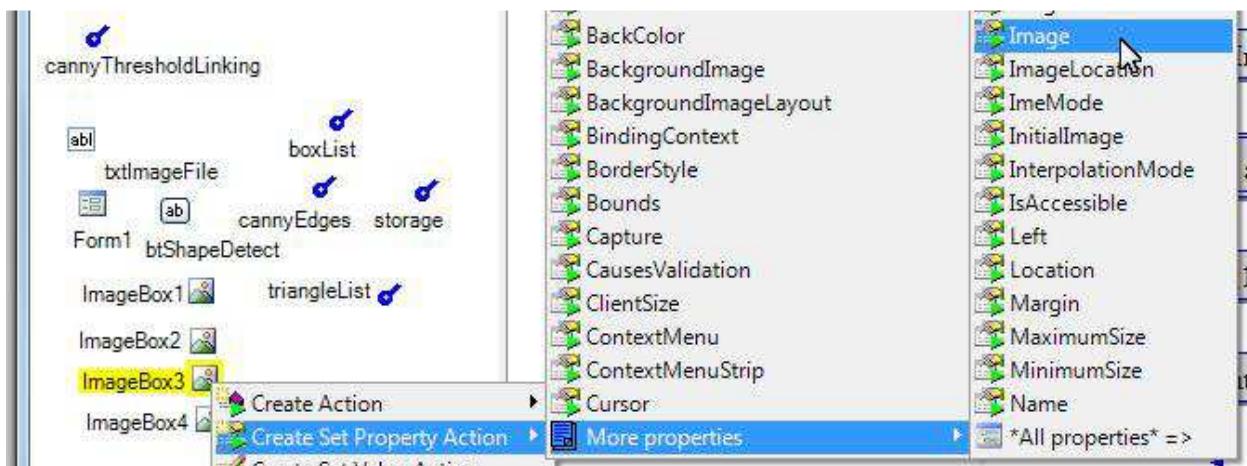
Action Properties window (bottom): Shows the properties for 'circleImage.Draw1'. The 'color' property is set to 'Bgr(Color [Brown])'. The 'winColor' dropdown is open, showing color swatches for 'Red', 'Brown', 'Firebrick', and 'IndianRed'. The 'Brown' swatch is selected.

Use Generic Types and Native Libraries



Display image

Display the image on the third image box:



Use Generic Types and Native Libraries

The figure consists of three vertically stacked screenshots from a software application's configuration interface.

Screenshot 1: Action Properties Dialog

This dialog shows the properties for an action named "ImageBox3.SetImage". The "value" property is currently set to "ImageBox3 of ImageBox.Image". A dropdown menu is open, showing options like "ConstantValue" and "MathExpression".

(ActionName)	ImageBox3.SetImage
Description	
HideFromRuntimeDesigners	False
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	true
ActionMethod	Image
Property	ImageBox3 of ImageBox.Image
value	ImageBox3 of ImageBox.Image

Screenshot 2: Object picker Dialog

This dialog shows a tree view of objects. Under "PerformShapeDetection", the "circleImage" object is selected. To the right, the variable "circles" is shown with its value "circles.value".

Screenshot 3: Action Properties Dialog

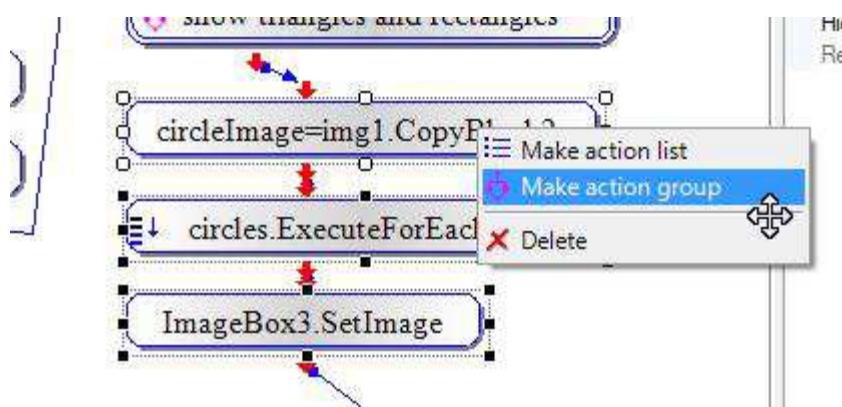
This dialog shows the same "ImageBox3.SetImage" action properties. The "value" property has been changed to "circleImage".

(ActionName)	ImageBox3.SetImage
Description	
HideFromRuntimeDesigners	False
BreakBeforeExecute	False
BreakAfterExecute	False
ActionCondition	true
ActionMethod	Image
Property	ImageBox3 of ImageBox.Image
value	circleImage
(TColor)	Bgr
(TDepth)	Byte

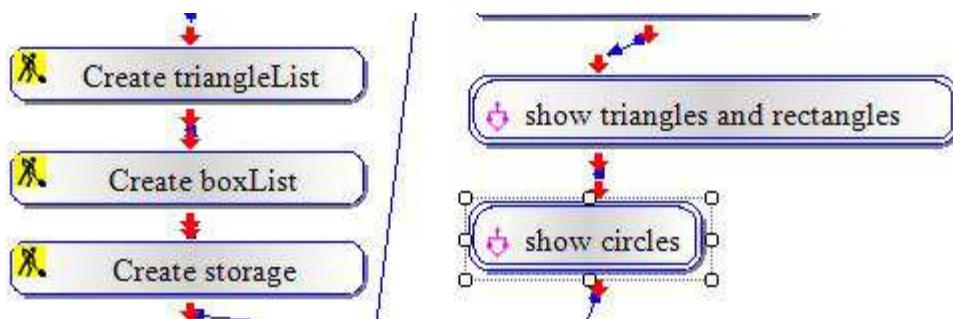
Make action group

This step is not necessary. But it helps make the Editor clean.

Use Generic Types and Native Libraries



Name the action group "show circles":



Display Lines

Create blank image

Method Designer - PerformShapeDetection

(Name) img1
(TColor) Bgr
(TDepth) Auto

Main

img1 Create Action

Select Method

Search:

Copy<>(Rectangle)Image<TColor,TDepth>
CopyBlank<>()Image<TColor,TDepth>
CopyTo<>(CvArr<TDepth>)

Most commonly used methods
_And<>(CvArray<TDepth>)

This screenshot shows the Method Designer and Select Method dialog boxes. In the Method Designer, 'img1' is selected, and a context menu is open with 'Create Action' highlighted. In the Select Method dialog, 'CopyBlank<>()Image<TColor,TDepth>' is selected.

Use Generic Types and Native Libraries

The first screenshot shows the 'Action Properties' dialog box for an action named 'img1.CopyBlank3'. The 'AssignTo' field is selected, showing options: 'New local variable' (highlighted) and 'Select existing object'. The second screenshot shows a 'Variable Name' dialog box with 'Name:' set to 'linelImage' and 'OK' button highlighted. The third screenshot shows the 'Action Properties' dialog box again, with 'linelImage' selected in the 'AssignTo' dropdown under the 'Image<TColor, TDepth>' row.

Action Properties

(ActionName) img1.CopyBlank3

Description

ReturnValueType Image<TColor, TDepth>

HideFromRuntimeDesigners False

BreakBeforeExecute False

BreakAfterExecute False

ActionCondition true

ActionMethod img1.CopyBlank<>()

AssignTo

(TColor)

(TDepth)

New local variable

Select existing object

Variable Name

Name: linelImage

OK

Action Properties

(ActionName) img1.CopyBlank3

Description

ReturnValueType Image<TColor, TDepth>

HideFromRuntimeDesigners False

BreakBeforeExecute False

BreakAfterExecute False

ActionCondition true

ActionMethod img1.CopyBlank<>()

AssignTo

linelImage

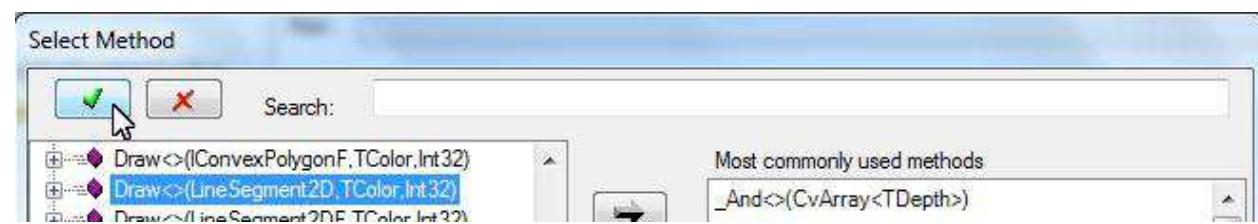
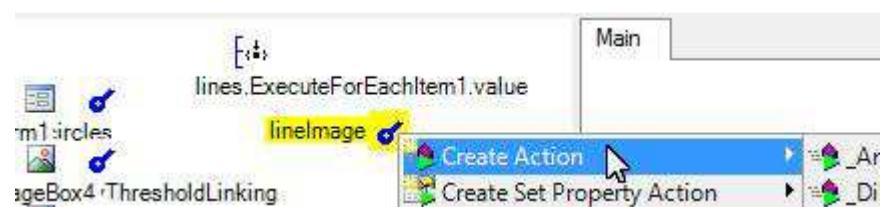
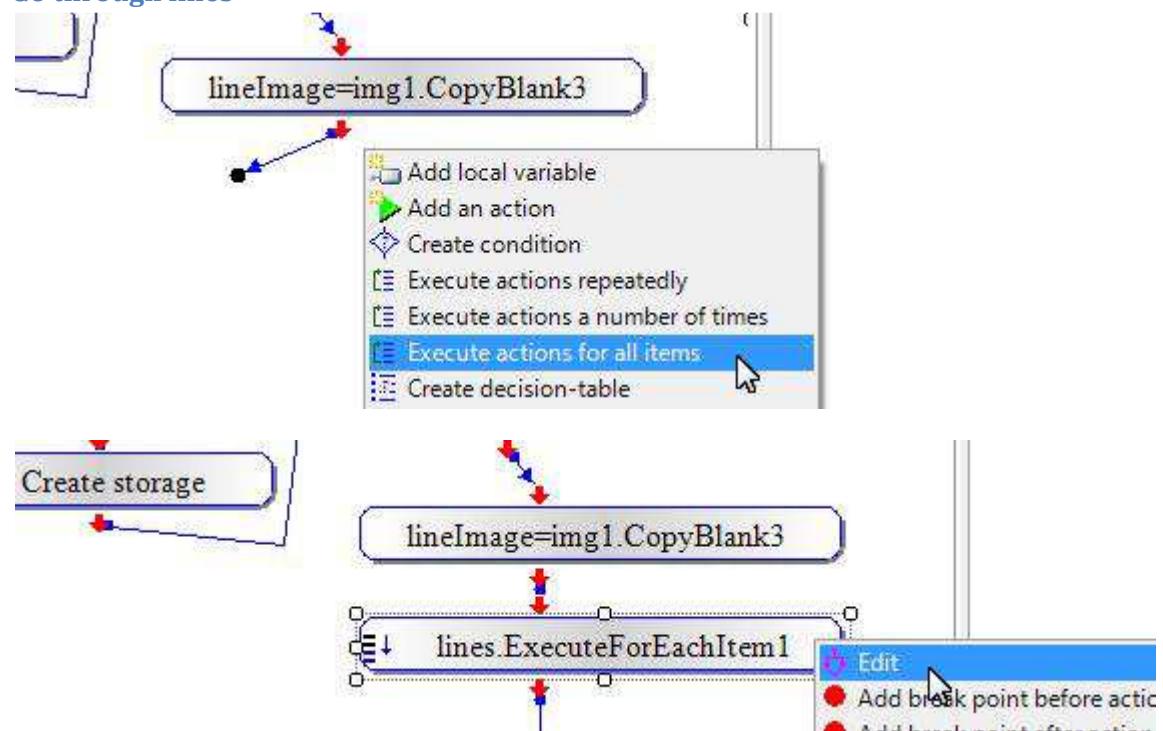
(TColor) Bgr

(TDepth) Byte

(Image<TColor, TDepth>) Image<TColor, TDepth>

Use Generic Types and Native Libraries

Go through lines

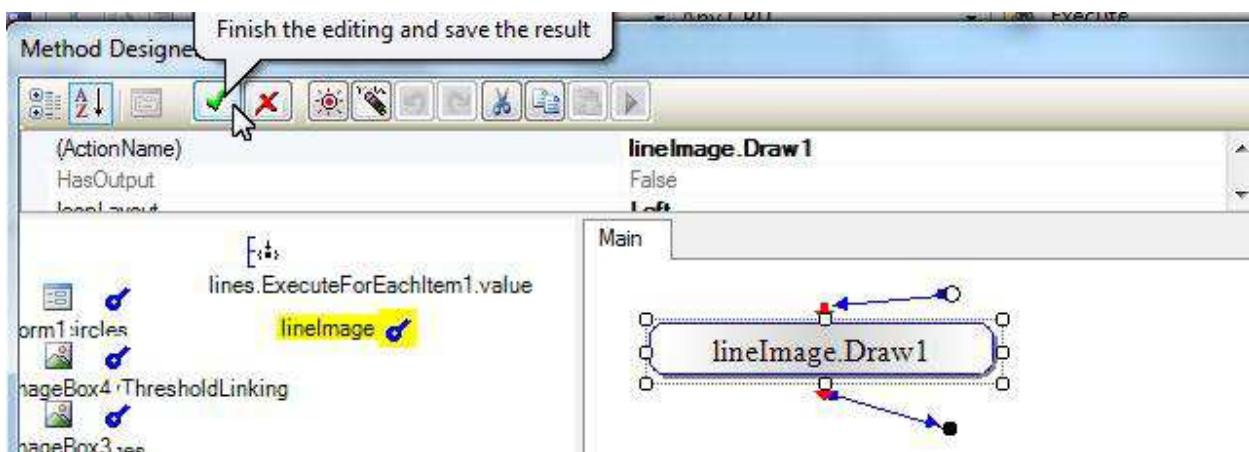
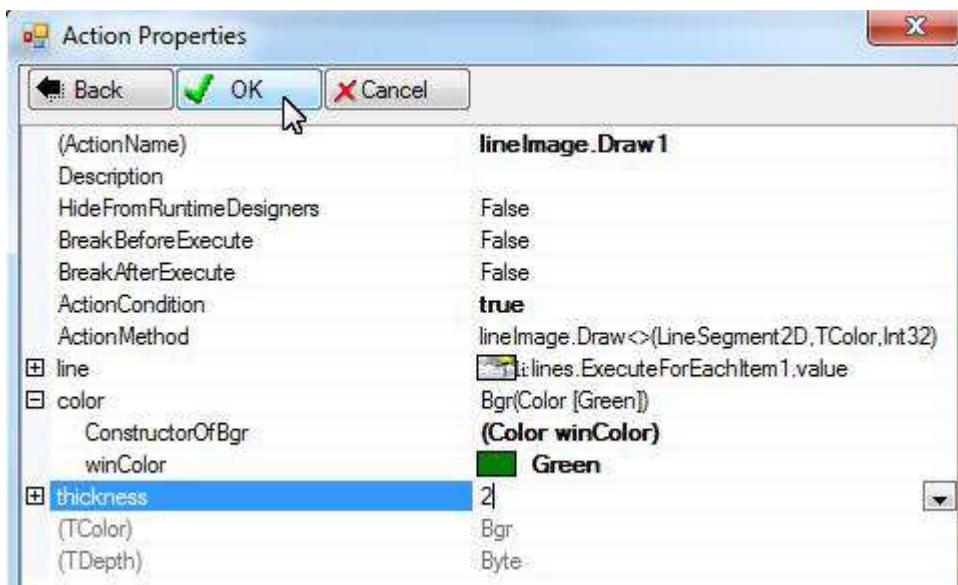
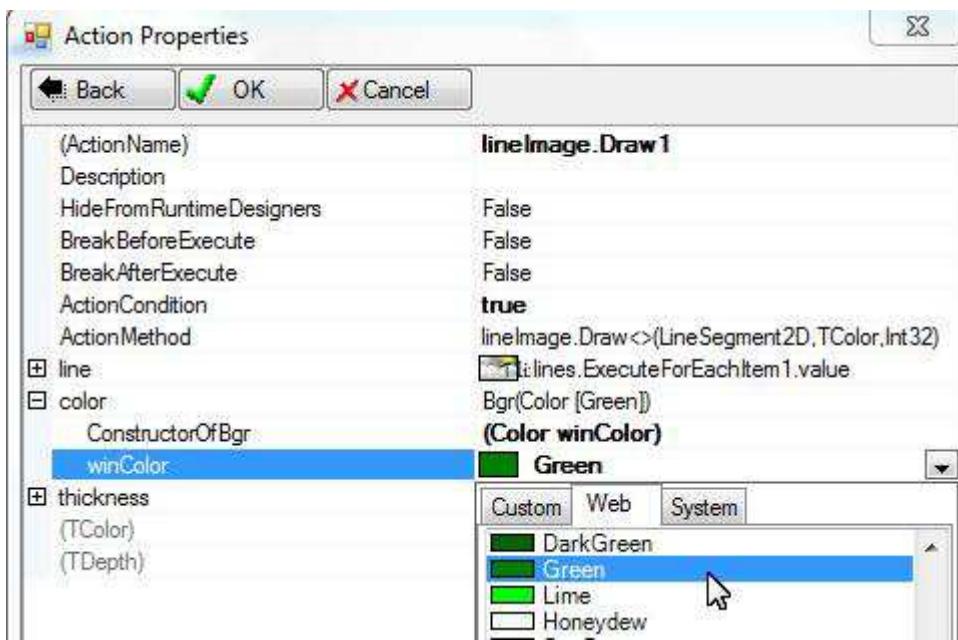


Use Generic Types and Native Libraries

The figure consists of three windows from a software interface:

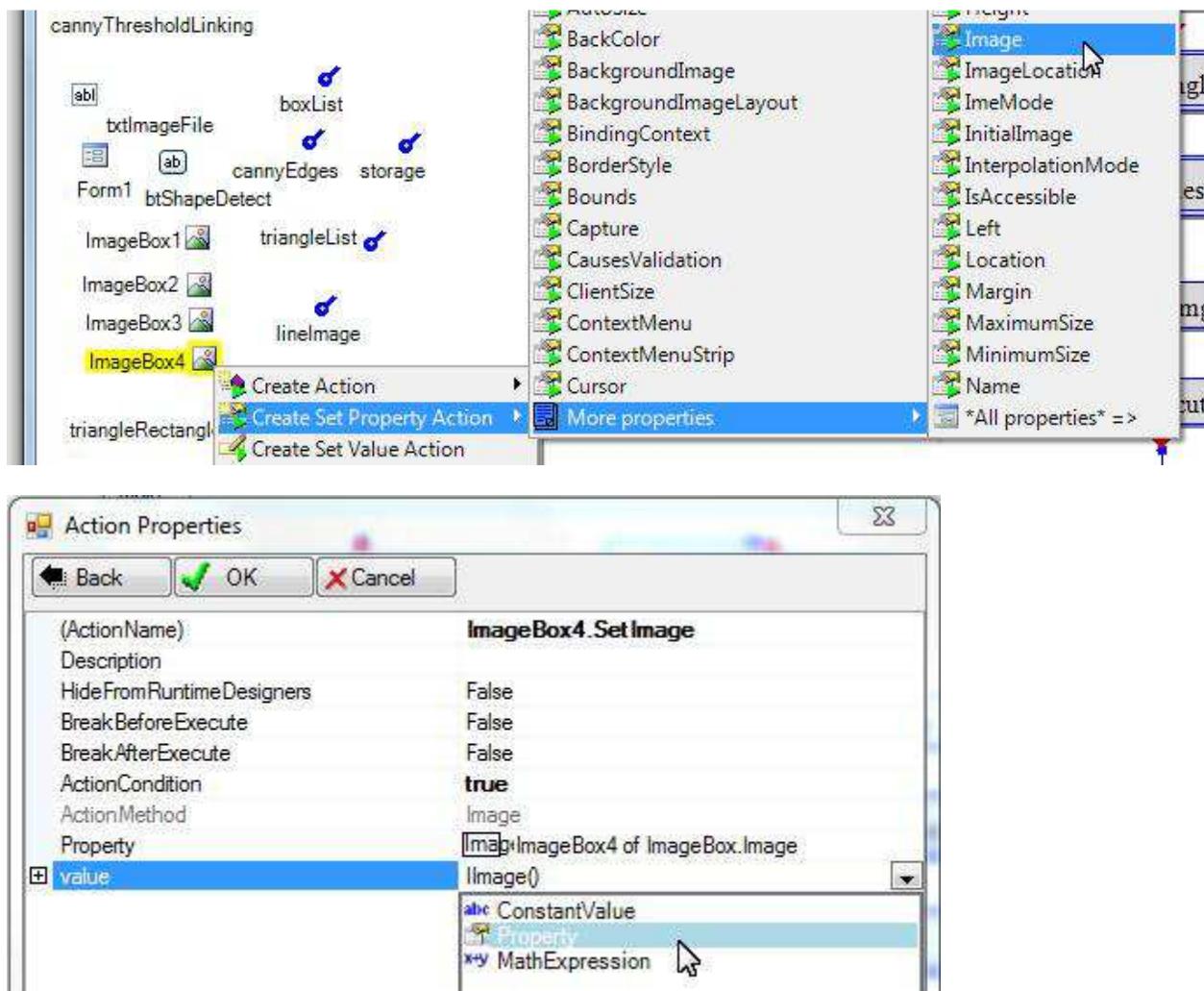
- Action Properties Dialog (Top):** Shows the configuration of an action named "lineImage.Draw1". The "ActionName" field is set to "lineImage.Draw1". The "ActionMethod" dropdown is set to "lineImage.Draw<>(LineSegment2D,TColor,Int32)". The "line" section is expanded, showing "color" and "thickness". The "color" dropdown is currently set to "ConstantValue". A context menu is open over the "color" dropdown, with options: "ConstantValue" (disabled), "Property", and "MathExpression".
- Object picker Dialog (Middle):** Shows a tree view of objects. The "PerformShapeDetection" node is expanded, showing various sub-nodes like "Actions", "boxList", "cannyEdges", etc. The "lines" node is also expanded, showing "ExecuteForEachItem1.value". To the right of the tree, there is a list of items: "circles", "circles.value", "circleImage", and "lines".
- Action Properties Dialog (Bottom):** Shows the configuration of the same action "lineImage.Draw1". The "ActionMethod" dropdown is now set to "lineImage.Draw<>(:lines.ExecuteForEachItem1.value)". The "line" section is expanded, showing "color". The "color" dropdown is set to "ConstructorOfBgr", which has a context menu open. The menu shows "(Double blue,Double green,Double red)" and "(Color winColor)".

Use Generic Types and Native Libraries



Display image

The lines are displayed on the 4th image box:



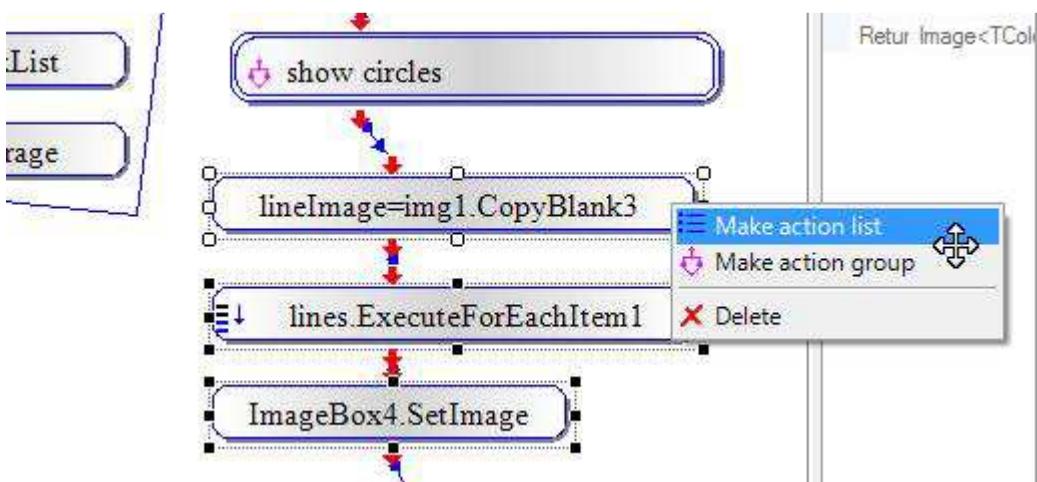
Use Generic Types and Native Libraries

The 'Object picker' window shows a tree view of objects and variables. The 'PerformShapeDetection' node has several children, including 'circleImage' which is selected. To the right, a list of variables is shown: circles, circles.value, circleImage, lines, and lines.value.

The 'Action Properties' window shows the configuration for the 'ImageBox4.Set Image' action. The 'Property' dropdown is set to 'Image' and the 'value' dropdown is set to 'lineImage'.

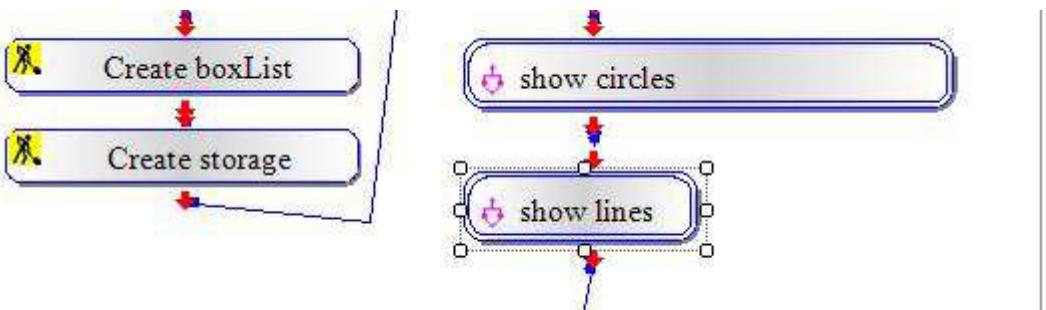
Make action group

This step is optional. It helps to make the diagram cleaner.



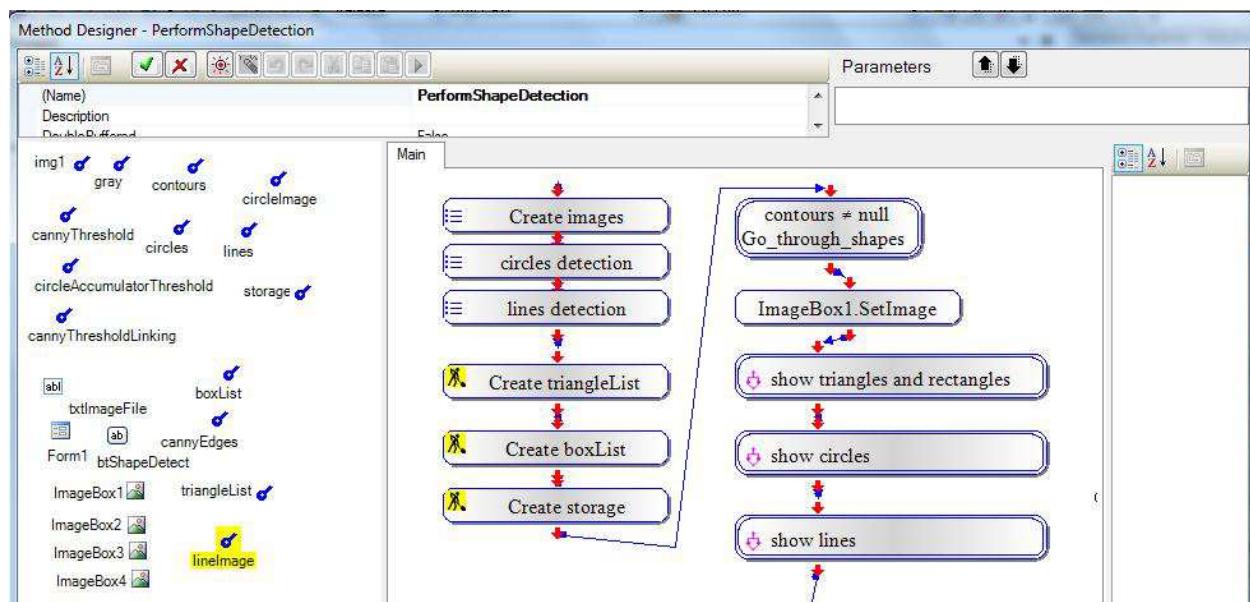
Use Generic Types and Native Libraries

Name the action group “show lines”:

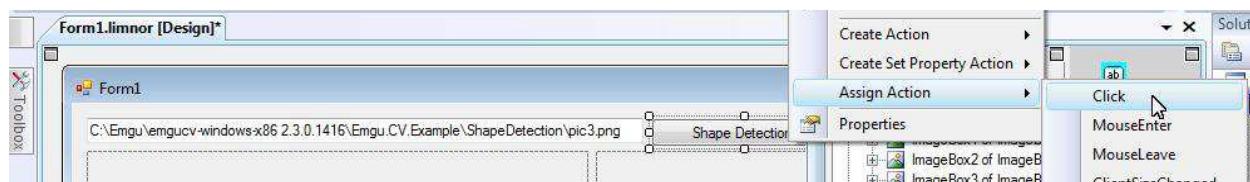


Execute the method

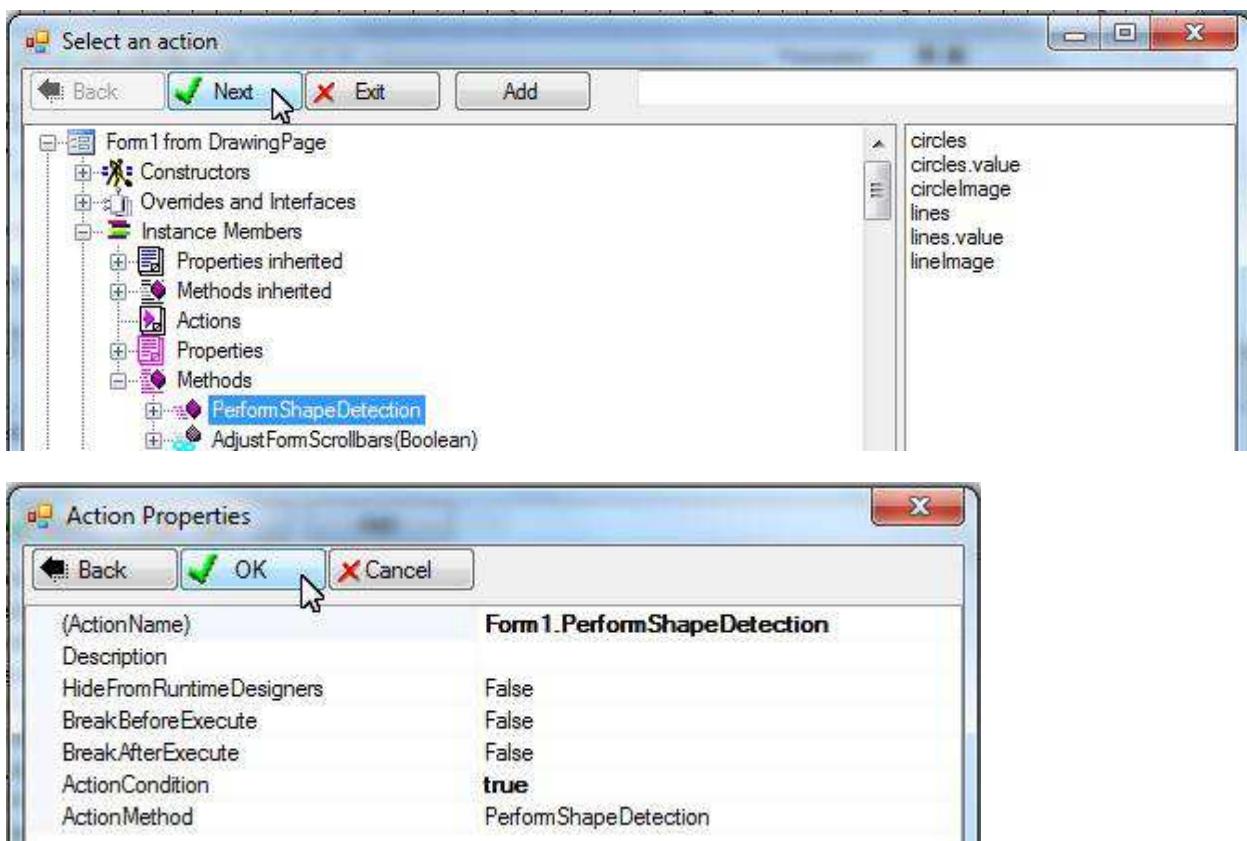
The method is done.



Assign the method to the button:



Use Generic Types and Native Libraries



We may test the project now.

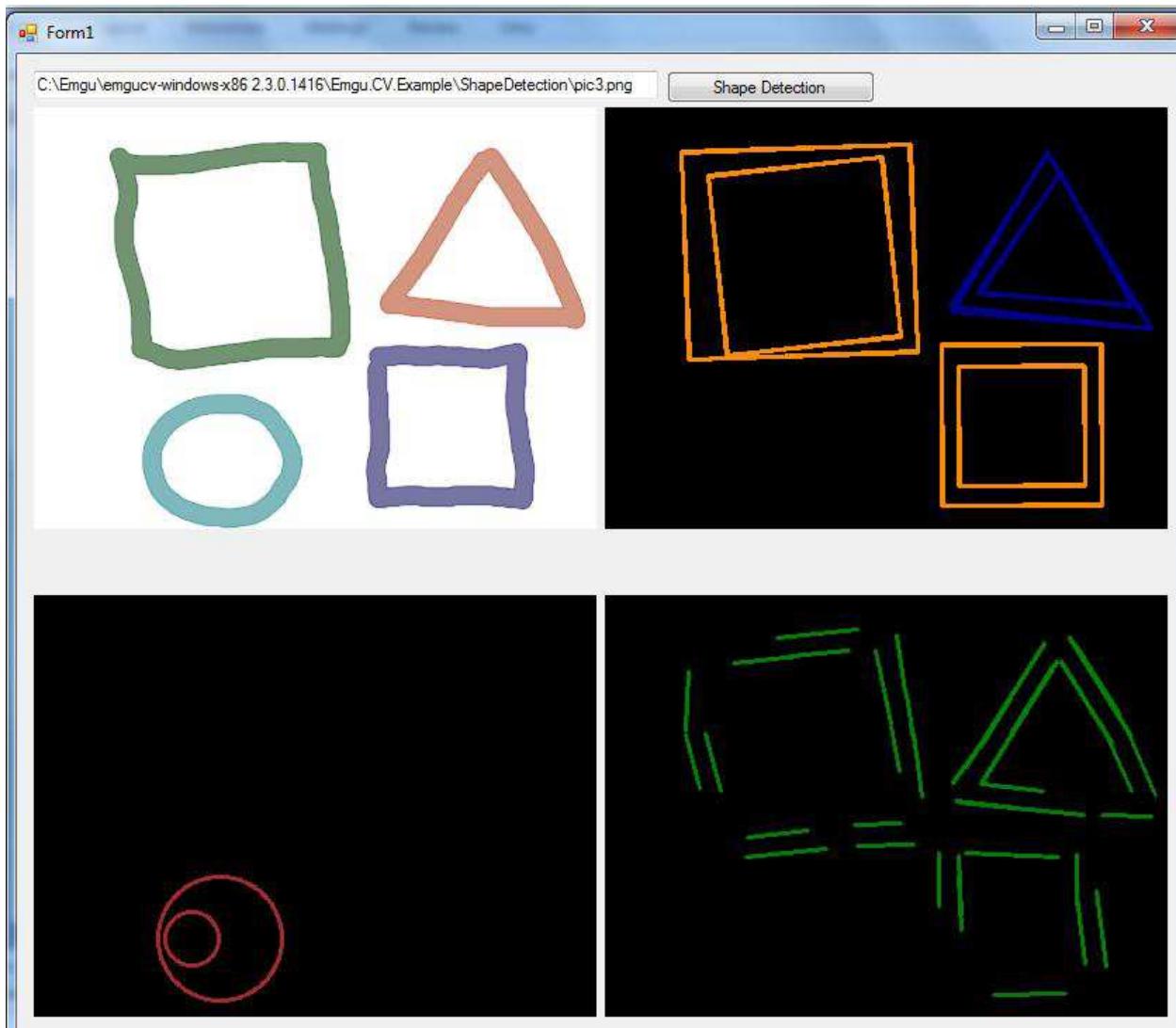


The form appears. Click the “Start Detection” button:



The original image and the detected shapes appear on the form.

Use Generic Types and Native Libraries



Feedback

Please send your feedback to support@limnor.com