Joshua Stimpert

Stark State College

December 10, 2017

Screenshots and simple image manipulation with C#

C# Fall 2017

# Screenshots and simple image manipulation with C#

## IntroDuction

For the Fall 2017 C# Final at Stark State, I was requested to explore an area of Visual Basic we had not covered in class, create a program that gave examples of the topic, and prepare a follow along lab like the ones we have had for assignments in class.

The topic I have decided to research is how to take screenshots with code, use GDI to manipulate the data, and save the manipulated image with a filestream. In order to accomplish this, I had to learn many different concepts I had personally not used yet in this class and integrate them together in a stable and effective manner.

Following is my walkthrough on how to create a screenshot capturing and basic image editing program that will crop the screenshots received by the capture.

## Walkthrough

### 1. Concepts Covered

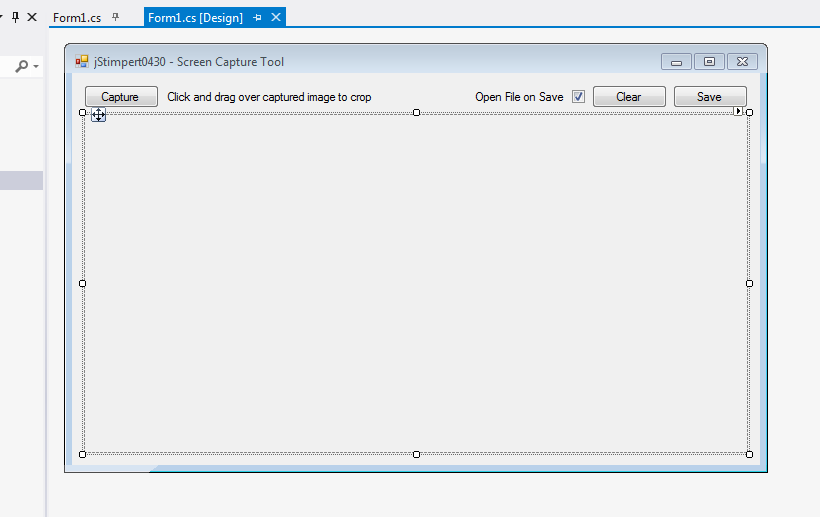
* The use of graphics objects and Picture Boxes to create and display the captured image.
* Use of the CopyFromScreen function to capture a screenshot.
* Use of GDI methods to add a rectangle around what is being cropped in real time.
* Use of filestream to push data into a file once finished.

### 2.1 Initial form setup

First, create a new project and give it a name of *username-Screen Capture Tool.* Then add the following objects to the default form—

* 3x Button
* 1x Panel
* 1x Picture Box
* 1x Check Box
* 1x Label

Once all of the objects have been added, rearrange and rename everything to look like what is shown and place the picture box inside the panel.



*2.1 Layout of initial form*

Once the layout has been completed, click on the picture box object and set size to auto in the properties, anchor the capture button and the label to the top left, then anchor the save, clear, and open file checkbox to the top right. Finally, anchor the panel to the right, bottom, and left.

### 2.2 Adding required namespaces

1. Once everything has been added and arranged, double click on some blank space on the form to open the .cs file for the form.

2. Add the following namespaces—

using System.Drawing.Imaging;

using System.IO;

This will allow us to use system IO functions and functions to draw graphics onto our form.

3. Declare the following variables at the highest scope for the form—

Rectangle selectionRect = new Rectangle();

Color penColor = Color.Red;

bool openFileOnSave = true;

### 2.3 Capture Screen Method

This will be our main method for the program, it will capture an image of the current screen by declaring a bitmap, graphics object, and sending that data to the picture box in the form where it can be later manipulated.

1. Start by declaring a method named “CaptureScreen” with 4 integer arguments that default to 0 and returns nothing.

public void CaptureScreen(int xValue = 0, int yValue = 0, int height = 0, int width = 0) {}

2. The first thing we’ll want to do is force garbage collection to get rid of any old screenshot that might be hanging around from a previous button press. This collection happens automatically, but not frequent enough to avoid the possibility of large amounts of garbage data flooding the ram if the capture button is pressed repeatedly. Since we need the data in the screenshot after the scope needed to dispose the image has expired, we’ll need to count on the garbage collector. Add the follow two lines of code at the top of your new function—

System.GC.Collect();

System.GC.WaitForPendingFinalizers();

3. After these lines, declare a new rectangle to be used as the area of the screen that will be captured. Call it “captureArea” and declare it blank for now, we’ll give it data before it’s used.

Rectangle captureArea = new Rectangle();

4. Add an if statement to check to see if the function has been provided a rectangle. Since it can be sent a valid rectangle at 0,0 we can just check it against the size to see if it actually exists, if not make a rectangle the size of the current main screen at 0,0; otherwise, copy rectangle sent.

if (height != 0 && width != 0)

captureArea = new Rectangle(xValue, yValue, height, width);

else

captureArea = Screen.AllScreens[0].Bounds;

5. Next, we’ll create a new bitmap object the same size as the rectangle set previously.

Bitmap screenshot = new Bitmap(captureArea.Width, captureArea.Height, PixelFormat.Format32bppArgb);

6a. In order to send information to the bitmap we have just made, we need to create and use a graphics object to copy image information from the screen. Create a new graphic object named screenshotGrabber and point it the screenshot bitmap created previously.

Graphics screenshotGrabber = Graphics.FromImage(screenshot);

6b. Next we’ll use the CopyFromScreen function of the graphics class to grab the image data contained at the coordinates and size specified by the provided rectangle. We can do this by accessing the specific top left X and Y coordinates and the size contained within the rectangle object.

screenshotGrabber.CopyFromScreen(captureArea.Location.X, captureArea.Location.Y, 0, 0, captureArea.Size);

7. Now that we finally have image data in a usable format, we can set the image data in the picture box to that held in the current screenshot bitmap.

pictureBox1.Image = screenshot;

Close the method and add a call with no arguments to the Capture button’s press event on your form.

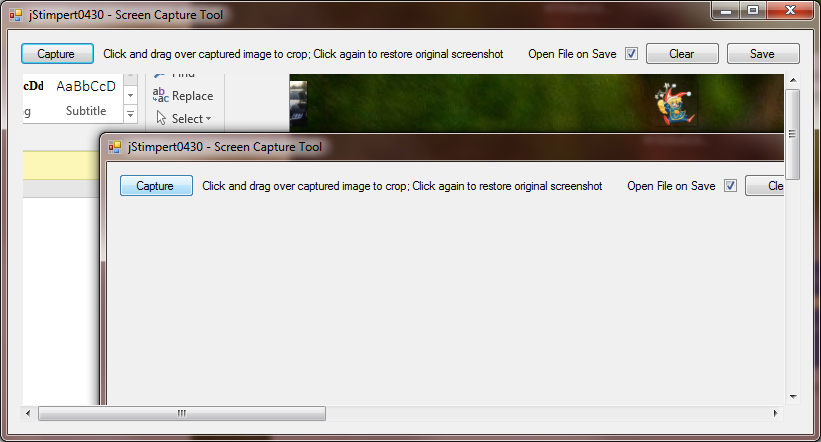
private void button2\_Click(object sender, EventArgs e)

{

CaptureScreen();

}

Give it a try to see if everything works correctly now. When the capture button is pressed, a fullscreen screenshot of your main monitor will display in the picturebox at full size with scrollbars to pan left/right or up/down.



*2.3 Functioning full screen capture and display*

public void CaptureScreen(int xValue = 0, int yValue = 0, int height = 0, int-- --width = 0)

{

Rectangle captureArea = new Rectangle();

System.GC.Collect();

System.GC.WaitForPendingFinalizers();

if (height != 0 && width != 0)

{

captureArea = new Rectangle(xValue, yValue, height, width);

}

else

{

captureArea = Screen.AllScreens[0].Bounds;

}

Bitmap screenshot = new Bitmap(captureArea.Width, captureArea.Height,-- --PixelFormat.Format32bppArgb);

Graphics screenshotGrabber = Graphics.FromImage(screenshot);

screenshotGrabber.CopyFromScreen(captureArea.Location.X,-- --captureArea.Location.Y, 0, 0, captureArea.Size);

pictureBox1.Image = screenshot;

}

*2.3 Completed CaptureScreen Method*

### 2.4 Draw Rectangle With Click and Drag

Now that we have an image, we’ll need to provide the user a way to crop a particular area of it. We’ll do this by using the Mouse\_Down, Mouse\_Up, and Mouse\_Move events of the pictureBox. When the user has their cursor over the pictureBox then clicks and drags, the application will draw a red rectangle showing which area will be included in the crop and then send that rectangle information to the CaptureScreen function.

1. The first thing we’ll want to do is add a paint event for the pictureBox so that we can draw on the image. We’ll need to create a pen and tell the paint event to draw a rectangle at the selection rectangle. We can use the pen color variable set up at the start so that we can hide it later when taking the screenshot and we’ll use “using” since everything can be discarded when it is not in use.

private void pictureBox1\_Paint(object sender, PaintEventArgs e)

{

using (Pen pen = new Pen(penColor, 1))

{

e.Graphics.DrawRectangle(pen, selectionRect);

}

}

2. Now that we’re ready to paint, we can start building the rectangle with the Mouse\_Down event in the pictureBox panel. When the user first clicks the mouse down while inside the paint box, we want to record the coordinates they have clicked into the rectangle created at the start of this example. This is easily accomplished by storing the mouse pointer location as the rectangles location

selectionRect.Location = e.Location;

3a. Once we have the start coordinates, we can start building and drawing the rectangle in real time. Inside the Mouse\_Move event for the paintBox, we first want to check to see if the user is holding the left mouse button down. If they are that means they have already clicked it and there is a valid rectangle start point to use.

if (e.Button == MouseButtons.Left){}

3b. Since this event is called every time the mouse is moved, we can compare the mouse’s current position against the position originally recorded when the mouse button was pressed to build a rectangle in real time with the use of Math.Min and Math.Max functions and some subtraction. Inside the if statement add the following code—

int x = Math.Min(selectionRect.Location.X, e.X);

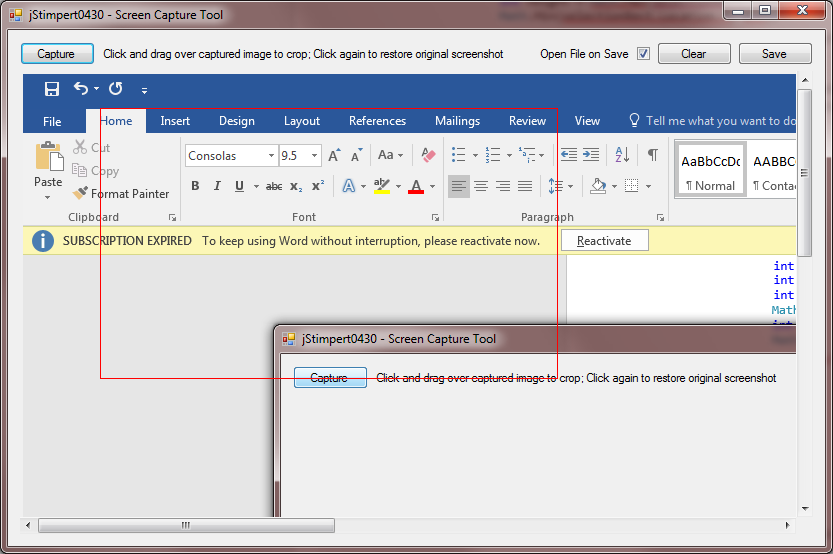
int y = Math.Min(selectionRect.Location.Y, e.Y);

int width = Math.Max(selectionRect.Location.X, e.X) - Math.Min(selectionRect.Location.X, e.X);

int height = Math.Max(selectionRect.Location.Y, e.Y) - Math.Min(selectionRect.Location.Y, e.Y);

selectionRect = new Rectangle(x, y, width, height);

3c. While the rectangle is technically being drawn, the form itself isn’t updating to show it so we manually have to refresh at the end of the move event to redraw the rectangle. Do so by adding Refresh(); inside the if statement after creating the rectangle.

**

*2.4 Working Cropping Rectangle*

We can now draw our rectangle in real time over our captured image, however it doesn’t disappear when we let go of the mouse button. This might be desirable in some applications, but for this situation we’re going to add an additional event the Mouse\_Up event in the pictureBox that will finish cropping the screen and cleaning up our box.

### 2.5 Use Rectangle to Crop

Now that we have a rectangle built, we can send that information to the CaptureScreen method set up earlier, this time with the arguments provided by the rectangle we have just made.

1a. We’ll continue using the mouse events in the pictureBox, Add a Mouse\_Up event. The first thing we’ll need to do when the user let’s go of the mouse is start some preparation to capture the screen. Since we don’t want the rectangle to show in the screenshot, we’ll first set the pen color to transparent.

penColor = Color.Transparent;

1b. Next we’ll want delete any garbage data left over in the pictureBox currently to prevent any undesirable effects. We can do this by setting the image to null;

pictureBox1.Image = null;

1c. In order to display the changes before screen capture we need to manually redraw the form with the Refresh(); function again.

2. Now that we’re finally ready to capture the area we want to crop, we can make the call to the CaptureScreen method. Pass in the X, Y, Width, and Height coordinates of the rectangle as the arguments for the function.

CaptureScreen(selectionRect.X, selectionRect.Y, selectionRect.Width, selectionRect.Height);

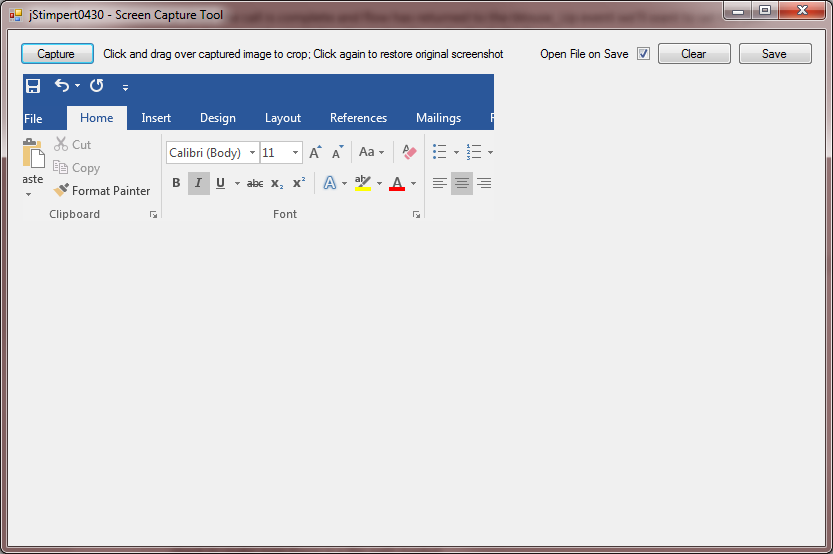
This function call will recapture a section of the image shown in the picture box and put replace the data currently held in the new resized bitmap.

3. Once the call is complete and flow has returned to the Mouse\_Up event we’ll want to set the pen color back to red, reinitialize the rectangle, and refresh the form.

penColor = Color.Red;

selectionRect = new Rectangle();

Refresh();

**

*2.5 Cropped image made from larger screenshot*

This completes the basic functionality of our screen capturing and cropping program, however, the user still doesn’t have a way to save the data captured for storage. The next section will deal with saving the image capture with a filestream and a saveFileDialog.

### 2.6 Save Image in PictureBox

The first thing we’ll want to do is add logic to the checkbox added at the beginning.

1. Change the openFileOnSave Boolean depending on the check state of the check box.

private void checkBox1\_CheckedChanged(object sender, EventArgs e)

{

if (checkBox1.Checked)

openFileOnSave = true;

else

openFileOnSave = false;

}

2. Now we’ll want to handle the actual saving of the file. Add an event to the save button’s click and insert an if statement that checks to see if there is an image in the pictureBox at all.

if (pictureBox1.Image != null){}

3. If we have an image to save we’ll want to create a new save dialog with a filter to force jpegs and a title of “Save a Screenshot”, then we’ll want to show it. Inside the if statement add the following—

SaveFileDialog saveImageDialog = new SaveFileDialog();

saveImageDialog.Filter = "Jpeg Image|\*.jpg";

saveImageDialog.Title = "Save a Screenshot";

saveImageDialog.ShowDialog();

4. Once returned from the showDialog() function, we’ll want to check to see if the user did input a file path for us to use. Add another if statement inside the previous if statement that checks to make sure the filename isn’t empty.

if (saveImageDialog.FileName != "")

5. Now that we’re certain we have a file location and image we can finally open a filestream to the file specified by the user. Do so by creating a new filestream and pointing at openFile() function of the saveFileDialog.

System.IO.FileStream fs = (System.IO.FileStream)saveImageDialog.OpenFile();

6. With the filestream open we can use the save function of the image class to send the screenshot data to the file in an image format.

pictureBox1.Image.Save(fs, System.Drawing.Imaging.ImageFormat.Jpeg);

7. Once the data has finished sending we need to close the file stream, we can do this by simply calling the close function of the fs object.

fs.Close();

8. Now that the file has saved we can add another if statement to either open the file if the user has checked the box, or show a messagebox saying the file has saved otherwise.

if (openFileOnSave == true) System.Diagnostics.Process.Start(saveImageDialog.FileName.ToString());

else

MessageBox.Show("Save Successful", "File Saved");

9. Finally, Close the final 2 if’s and add an else statement that shows a messagebox telling the user they must take a screenshot before they can save data.

MessageBox.Show("Must first capture a screenshot to save.","Error: No Screenshot");

private void button4\_Click(object sender, EventArgs e)

{

if (pictureBox1.Image != null)

{

SaveFileDialog saveImageDialog = new SaveFileDialog();

saveImageDialog.Filter = "Jpeg Image|\*.jpg";

saveImageDialog.Title = "Save a Screenshot";

saveImageDialog.ShowDialog();

if (saveImageDialog.FileName != "")

{

System.IO.FileStream fs = (System.IO.FileStream)saveImageDialog.OpenFile();

pictureBox1.Image.Save(fs, System.Drawing.Imaging.ImageFormat.Jpeg);

fs.Close();

if (openFileOnSave == true) System.Diagnostics.Process.Start(saveImageDialog.FileName.ToString() );

else

MessageBox.Show("Save Successful", "File Saved");

}

}

else

MessageBox.Show("Must first capture a screenshot to save.","Error: No Screenshot");

}

*2.6 Completed Save Function*