

Министерство образования и науки РФ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Московский технологический университет»

МИРЭА

ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

КАФЕДРА ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

**Курсовая работа**

по дисциплине

«Программирование»

на тему: **«Клиент к хранилищу фотографий»**

Студент группы ИКБО-04-16 Сидоренко М.П.

Научные руководители:

к.ф.-м.н., доцент Горшенин А.К.

руководитель Мирабо Е.И.

Работа принята и проверена «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_201\_г.

Москва

2017

**СОДЕРЖАНИЕ**

Введение3

1. Постановка задачи4
2. Теоретический раздел5

Клиент-сервер5

1. Раздел проектирования6
   1. Слабые ссылки6
   2. Асинхронное выполнение7
   3. Ввод/вывод клавиатуры, видеокарты, буфера обмена9
      1. Перехват клавиши Print Screen9
      2. Захват изображения монитора9
      3. Вставка ссылки в буфер обмена10
   4. Используемые ссылки в программе11

Заключение12

Список используемых источников13

Приложения14

**ВВЕДЕНИЕ**

Программирование – это одна из интенсивно развивающихся областей в сфере информационных технологий и охватывает как теоретические вопросы, так и вопросы, непосредственно связанные с практикой. Изучаемая дисциплина основывается на использовании языка программирования C (*Си*).

Для расширения, закрепления и систематизации теоретических знаний, полученных по дисциплине «Программирование» было сформировано задание для курсовой работы.

Объектом исследования в данной работе является клиент сервера с базой данных изображений. Клиент состоит из перехвата средств ввода клавиатуры и вывода изображения, а также из отправителя, который отправляет изображение на сервер.

Целью курсовой работы является разработка клиента-отправителя к серверу изображений на языке программирования C# (*Си Шарп*). Необходимо, чтобы программа умела перехватывать нажатие клавиши захвата *изображения*; считывать, кодировать и отправлять *его*; получать ссылку общего доступа к загруженному изображению через сервер. У программы должен быть простой, но удобный интерфейс взаимодействия с пользователем.

1. **ПОСТАНОВКА ЗАДАЧИ**

Разработать отправку изображения вывода компьютера и получения ссылки общего доступа к изображению от сервера.

Реализовать возможность менять сервер, уведомлять по желанию о успешном создании изображении, опцию резервного копирования изображения в кэш.

Отправка изображения должна осуществляться после нажатия клавиши Print Screen.

Спроектировать программу таким образом, чтобы она была фоновой, с доступом к графическому интерфейсу через панель задач, и не занимала много ресурсов по сравнению с системными фоновыми процессами.

1. **ТЕОРЕТИЧЕСКИЙ РАЗДЕЛ**

**Клиент — сервер**

Описание: Функциональность системы обеспечивается набором услуг (сервисов). Каждый сервис располагается на своём сервере. Клиенты являются пользователями этих сервисов. Для получения услуги клиент обращается к серверу. Используется, когда:

1. услуги должны быть доступны из разных мест
2. требуется гибкий механизм перестройки системы по загружаемым начальным данным

Преимущества такой системы является:

1. предоставление клиентам различных услуг через сеть
2. устраняется необходимость тиражирования реализации услуг среди серверов

Недостатком является вероятным понижением скорости доступа к данным из-за проблем в сети. Поломка сервера лишает клиента услуги.[[1]](#footnote-1)

1. **РАЗДЕЛ ПРОЕКТИРОВАНИЯ**
   1. **Слабые ссылки**

Для эффективного очищения памяти используется слабые ссылки, чтобы встроенный в библиотеку .NET Framework сборщик мусора эффективно очищал временные экземпляры компонентов.

Слабые ссылки (*week reference*) используются при необходимости зранения в памяти необязательных объектов большого размера. Например, получив с SQL-сервера данные, программа может хранить их в памяти до тех пор, пока эта память не потребуется системе для своих нужд. Сохранение данных через обычную (сильную) ссылку не даёт возможности освобождения памяти. Именно в этом случае пригодятся слабые ссылки.[[2]](#footnote-2)

В сравнении с сильной ссылкой:

MyClass A = new MyClass();

Слабая ссылка имеет вид только вызова создания объекта/экземпляра и вызова конструктора:

new MyClass();

В программе используется комбинированный способ, когда в некоторый момент времени существует переменная, ссылающая на экземпляр, и после его обработки – явно уничтожить все ссылки на этот экземпляр. Отрывок из файла программы Progarm.cs:

byte[] Scr = Combine.GetScreen(); // Получение изображения экрана

if (Settings.Save) SaveScr.Save(Scr); // Сохранение изображения на диск

Resp = await Combine.SendScreen(Scr); // Отправка на сервер фото и получение ответа

Scr = null; // Явное удаление ссылки на экземпляр изображения

После каждого отработанного перехвата клавиши Print Screen дважды вызывается сборщик мусора.

При таком аккуратном подходе программа не занимает более 15 мегабайт в оперативной памяти компьютера.

* 1. **Асинхронное выполнение**

Обычно к механизму async прибегают, когда имеется длительная операция, которую можно выполнить асинхронно, освободив тем самым ресурсы. В программах с пользовательским интерфейсом асинхронность позволяет обеспечить отзывчивость интерфейса (если, конечно, операция не выполняется мгновенно). В серверном коде компромисс не столь очевиден, так как мы должны выбирать между памятью, занятой заблокированными потоками, и дополнительным процессорным временем, затрачиваемым на выполнение асинхронных методов.[[3]](#footnote-3)

В программе ScrTilla методы, которые используют жёсткий диск и сетевую карту, выполняются асинхронно для уменьшения задержек программы. Пока программа сохраняет изображение на диск, одновременно с этим изображение отправляется на сервер через сетевую карту.

Асинхронное сохранение файла на жёсткий диск (SaveScr.cs):

/// <summary>

/// Сохранение изображения на диск.

/// </summary>

/// <param name="input">Данные, которые следует сохранить.</param>

public static async void Save(byte[] input)

{

// Если папка, в которой должен храниться файл не существует,

if (!Directory.Exists(DIRECT\_NAME))

// То создать папку

Directory.CreateDirectory(DIRECT\_NAME);

FileStream fp = null;

try

{

// Открытие потока записи файла

using (

fp = new FileStream

(DIRECT\_NAME + DateTime.Now.ToString("\\\\yyyy-MM-dd hh-mm-ss.pn\\g"), FileMode.CreateNew)

)

{

// Асихронная запись инофрмации в файл.

await fp.WriteAsync(input, 0, input.Length);

}

}

catch (IOException)

{

// Если с файлом что-то случилось, и ссылка на поток существует, то поток следует закрыть

fp?.Close();

}

catch { }

}

Асинхронная отправка файла на сервер (Combine.cs):

/// <summary>

/// Отправка изображения на сервер.

/// </summary>

/// <param name="image">Изображение, которое следует отправить.</param>

/// <returns>Ответ от сервера</returns>

public static async Task<json\_st.ResponsePost> SendScreen(byte[] image)

{

// Создание контейнера для отправки

var requ = new MultipartFormDataContent();

// Конвектируем массив байтов в контент HTTP протокола

var ImContent = new ByteArrayContent(image);

// Указываем тип конента

ImContent.Headers.ContentType = MediaTypeHeaderValue.Parse("image/png");

// Добавление сведений о контенте в контейнере

requ.Add(ImContent, "up\_image", "image.png");

// Отправка изображения

HttpResponseMessage response = await cl.PostAsync(Settings.UPLOAD, requ);

// Уничтожение контейнера

requ.Dispose();

// Уничтожение контента HTTP

ImContent.Dispose();

// Конвектирование ответа JSON в структуру C#

return await Newtonsoft.Json.JsonConvert.DeserializeObjectAsync<json\_st.ResponsePost>

(await response.Content.ReadAsStringAsync());

}

* 1. **Ввод/вывод клавиатуры, видеокарты, буфера обмена**
     1. **Перехват клавиши Print Screen**

Для перехвата нажатия на клавиатуре клавиши Print Screen был написан класс PrtScr\_Hook, который реализует старт и паузу перехвата, хранения списка методов, которые требуется вызвать при нажатии клавиши. Он использует класс globalKeyboardHook, который был взят с открытого форума компании Microsoft. Данный класс реализует загрузку и использование библиотек user32.dll и kernel32.dll, использование следующих методов из них:

Начать перехват клавиатуры:

static extern IntPtr SetWindowsHookEx(int idHook, keyboardHookProc callback, IntPtr hInstance, uint threadId);

Закончить перехват клавиатуры:

static extern bool UnhookWindowsHookEx(IntPtr hInstance);

Вызов следующего перехвата:

static extern int CallNextHookEx(IntPtr idHook, int nCode, int wParam, ref keyboardHookStruct lParam);

Загрузка библиотеки:

static extern IntPtr LoadLibrary(string lpFileName);

* + 1. **Захват изображения монитора**

В данной программе идёт перехват изображения главного монитора компьютера. Для этого используется встроенные возможности .Net пространство имён System.Drawing. Данный метод находится в статическом классе Combine

/// <summary>

/// Получает изображение экрана.

/// </summary>

/// <returns>Фото экрана.</returns>

public static byte[] GetScreen()

{

// Получение размеров главного экрана и создание пустого изображения

Bitmap bitmap = new Bitmap(Screen.PrimaryScreen.Bounds.Width, Screen.PrimaryScreen.Bounds.Height);

// Создаём поверхность рисования

Graphics graphics = Graphics.FromImage(bitmap as Image);

// Срисовываем то, что на экране

graphics.CopyFromScreen(0, 0, 0, 0, bitmap.Size);

// Закрываем рисование

graphics.Dispose(); graphics = null;

// Отправляем результат

return ImageToByte(bitmap);

}

* + 1. **Вставка ссылки в буфер обмена**

Для безопасной вставки в буфер обмена в многопоточной программе был написан специальный класс Clipboard\_s, который в отдельном потоке опрашивает буфер обмена, свободен ли он для записи:

[STAThread]

public static void ToClipboard(string Text)

{

// Выполнить операцию требуется в отдельном потоке

new Thread((delegate ()

{

// Эту операцию требуется делать, пока не будет результат

while (true)

{

lock (ProtectStaticThread) // Защитный замок

if (GetOpenClipboardWindow() == IntPtr.Zero) // Если свободен буфер обмена

{

try

{

// То отправляем текст в буфер обмена

Clipboard.SetText(Text);

}

catch

{

// В случае ошибки, пробуем закрыть буфер обмена с нашей стороны

CloseClipboard();

Thread.Sleep(1000);

continue;

}

// Очищаем память после работы с буфером обмена

GC.Collect();

GC.WaitForPendingFinalizers();

GC.Collect();

return;

}

else

{

// Если буфер обмена занят, ожидаем секундочку

Thread.Sleep(1000);

}

}

}))

// Запускать поток требуется в STA режиме

{ ApartmentState = ApartmentState.STA }.Start();

}

* 1. **Используемые ссылки в программе**
     + Newtonsoft.Json – Пространство имён, содержащее инструменты для перевода данных в JSON запись и обратно.
     + PresentationCore – Пространство имён, который открывает доступ к использованию класса Clipboard.
     + System – Пространство имён содержит фундаментальные и базовые классы, определяющие часто используемые типы значений и ссылочных данных, события и обработчики событий, интерфейсы, атрибуты и исключения обработки.
     + System.Drawing – Пространство имён обеспечивает доступ к базовым функциональным возможностям графического интерфейса GDI+.
     + System.Net.Http – Пространство имён предоставляет интерфейс программирования для современных HTTP-приложений.
     + System.Windows.Forms – Пространство имён System.Windows.Forms содержит классы для создания приложений на базе Windows, которые в полной мере используют возможности богатого пользовательского интерфейса, доступные в операционной системе Microsoft Windows.

**ЗАКЛЮЧЕНИЕ**

В результате изучения стандартов сервера была написана соответствующая программа-клиент.

Программа реализована на использовании асинхронных операций, http клиента, перехвате ввода/вывода клавиатуры, изображения экрана, буфера обмена.

Был реализован следующий функционал:

* + - * перехват клавиши Print Screen;
      * получение сведений о сервере;
      * отправка изображения экрана на сервер;
      * отправка ссылки общего доступа в буфер обмена;
      * резервное копирование отправляемых файлов;
      * настройка адреса сервера, резервного копирования и уведомлений;
      * оперативная работа с памятью: быстрое очищение ресурсов;
      * обработка большого количества исключений;
      * графический интерфейс.

**СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ**

1. Орлов С.А., Программная инженерия. Технологии разработки программного обеспечения – 5-у изд., обновлённое и доп. – СПб.: Питер, 2016. – 640 с.
2. Агуров П.В., C#. Сборник рецептов. – СПб.: БХВ-Петербург, 2007. – 432 с.: ил. ISBN 5-94157-969-1
3. Алекс Дэвис, Асинхронное программирование в C# 5.0. / Пер. с англ. Слинкин А. А. – М.: ДМК Пресс, 2013. – 120 с.: ил. ISBN 978-5-94074-886-1
4. <http://microsoft.com/>

**ПРИЛОЖЕНИЯ**

1. Листинг файла AssemblyInfo.cs

using System.Reflection;

using System.Runtime.CompilerServices;

using System.Runtime.InteropServices;

// Управление общими сведениями о сборке осуществляется с помощью

// набора атрибутов. Измените значения этих атрибутов, чтобы изменить сведения,

// связанные со сборкой.

[assembly: AssemblyTitle("ScrTilla")]

[assembly: AssemblyDescription("")]

[assembly: AssemblyConfiguration("")]

[assembly: AssemblyCompany("Microsoft")]

[assembly: AssemblyProduct("ScrTilla")]

[assembly: AssemblyCopyright("Copyright © Microsoft 2017")]

[assembly: AssemblyTrademark("")]

[assembly: AssemblyCulture("")]

// Параметр ComVisible со значением FALSE делает типы в сборке невидимыми

// для COM-компонентов. Если требуется обратиться к типу в этой сборке через

// COM, задайте атрибуту ComVisible значение TRUE для этого типа.

[assembly: ComVisible(false)]

// Следующий GUID служит для идентификации библиотеки типов, если этот проект будет видимым для COM

[assembly: Guid("03e5439d-fb45-43b8-b8c8-6efb40328bcb")]

// Сведения о версии сборки состоят из следующих четырех значений:

//

// Основной номер версии

// Дополнительный номер версии

// Номер сборки

// Редакция

//

// Можно задать все значения или принять номера сборки и редакции по умолчанию

// используя "\*", как показано ниже:

// [assembly: AssemblyVersion("1.0.\*")]

[assembly: AssemblyVersion("1.0.0.0")]

[assembly: AssemblyFileVersion("1.0.0.0")]

1. Листинг файла FormPreferences.cs

using System;

using System.Windows.Forms;

namespace ScrTilla

{

public partial class FormPreferences : Form

{

public FormPreferences()

{

InitializeComponent();

bufferServer = Settings.ADDRESS;

textBox1.Text = Settings.ADDRESS;

checkBox1.Checked = Settings.Save;

checkBox2.Checked = Settings.Notifications;

timer1.Interval = 100;

timer1\_Tick(this, null);

}

/// <summary>

/// Создаёт экземпляр формы настроек

/// </summary>

/// <param name="Closed">Вызывается после закрытия формы.</param>

/// <param name="e">Сюда передастся метод, который следует вызывать в том случае, если нужно вывести форму на экран.</param>

public FormPreferences(FormClosedEventHandler Closed, ref Action e) : this()

{

base.FormClosed += Closed;

e = ToMaximizame;

}

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

{

}

private void FormPreferences\_FormClosing(object sender, FormClosingEventArgs e)

{

// save

}

private void textBox1\_TextChanged(object sender, EventArgs e)

{

if(!timer1.Enabled && bufferServer != textBox1.Text)

{

timer1.Start();

}

}

public void ToMaximizame()

{

this.Show();

this.WindowState = FormWindowState.Normal;

}

/// <summary>

/// true, если в данный момент сервер думает

/// </summary>

private bool IsServerLoad = false;

/// <summary>

/// То, что клиент сейчас проверяет

/// </summary>

private string bufferServer = "";

private async void timer1\_Tick(object sender, EventArgs e)

{

timer1.Stop();

if (!IsServerLoad)

{

IsServerLoad = true;

bufferServer = textBox1.Text;

pictureBox1.Image = Properties.Resources.Address\_test\_load;

if (await Settings.SetSettingsByAddress(bufferServer))

{

pictureBox1.Image = Properties.Resources.Address\_test\_ok;

}

else pictureBox1.Image = Properties.Resources.Address\_test\_error;

IsServerLoad = false;

}

if (bufferServer != textBox1.Text) timer1.Start();

}

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

{

Settings.ToDefault();

}

/// <summary>

/// Пользователь отметил галочку "Уведомлять"

/// </summary>

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

/// <param name="e"></param>

private void checkBox2\_CheckedChanged(object sender, EventArgs e)

{

Settings.Notifications = checkBox2.Checked;

}

/// <summary>

/// Пользователь отметил галочку "Сохранить"

/// </summary>

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

/// <param name="e"></param>

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

{

Settings.Save = checkBox1.Checked;

}

private void button1\_Click(object sender, EventArgs e)

{

SaveScr.Clear();

}

}

}

1. Листинг файла FormPreferences.Designer.cs

namespace ScrTilla

{

partial class FormPreferences

{

/// <summary>

/// Required designer variable.

/// </summary>

private System.ComponentModel.IContainer components = null;

/// <summary>

/// Clean up any resources being used.

/// </summary>

/// <param name="disposing">true if managed resources should be disposed; otherwise, false.</param>

protected override void Dispose(bool disposing)

{

if (disposing && (components != null))

{

components.Dispose();

}

base.Dispose(disposing);

}

#region Windows Form Designer generated code

/// <summary>

/// Required method for Designer support - do not modify

/// the contents of this method with the code editor.

/// </summary>

private void InitializeComponent()

{

this.components = new System.ComponentModel.Container();

this.labelAddress = new System.Windows.Forms.Label();

this.textBox1 = new System.Windows.Forms.TextBox();

this.checkBox1 = new System.Windows.Forms.CheckBox();

this.checkBox2 = new System.Windows.Forms.CheckBox();

this.pictureBox1 = new System.Windows.Forms.PictureBox();

this.button1 = new System.Windows.Forms.Button();

this.button2 = new System.Windows.Forms.Button();

this.timer1 = new System.Windows.Forms.Timer(this.components);

((System.ComponentModel.ISupportInitialize)(this.pictureBox1)).BeginInit();

this.SuspendLayout();

//

// labelAddress

//

this.labelAddress.AutoSize = true;

this.labelAddress.Location = new System.Drawing.Point(12, 9);

this.labelAddress.Name = "labelAddress";

this.labelAddress.Size = new System.Drawing.Size(45, 13);

this.labelAddress.TabIndex = 0;

this.labelAddress.Text = "Address";

//

// textBox1

//

this.textBox1.Location = new System.Drawing.Point(12, 25);

this.textBox1.Name = "textBox1";

this.textBox1.Size = new System.Drawing.Size(132, 20);

this.textBox1.TabIndex = 1;

this.textBox1.TextChanged += new System.EventHandler(this.textBox1\_TextChanged);

//

// checkBox1

//

this.checkBox1.AutoSize = true;

this.checkBox1.Location = new System.Drawing.Point(12, 51);

this.checkBox1.Name = "checkBox1";

this.checkBox1.Size = new System.Drawing.Size(51, 17);

this.checkBox1.TabIndex = 2;

this.checkBox1.Text = "Save";

this.checkBox1.UseVisualStyleBackColor = true;

this.checkBox1.CheckedChanged += new System.EventHandler(this.checkBox1\_CheckedChanged);

//

// checkBox2

//

this.checkBox2.AutoSize = true;

this.checkBox2.Location = new System.Drawing.Point(12, 74);

this.checkBox2.Name = "checkBox2";

this.checkBox2.Size = new System.Drawing.Size(84, 17);

this.checkBox2.TabIndex = 2;

this.checkBox2.Text = "Notifications";

this.checkBox2.UseVisualStyleBackColor = true;

this.checkBox2.CheckedChanged += new System.EventHandler(this.checkBox2\_CheckedChanged);

//

// pictureBox1

//

this.pictureBox1.Image = global::ScrTilla.Properties.Resources.Address\_test\_load;

this.pictureBox1.Location = new System.Drawing.Point(150, 25);

this.pictureBox1.Name = "pictureBox1";

this.pictureBox1.Size = new System.Drawing.Size(21, 20);

this.pictureBox1.SizeMode = System.Windows.Forms.PictureBoxSizeMode.Zoom;

this.pictureBox1.TabIndex = 3;

this.pictureBox1.TabStop = false;

//

// button1

//

this.button1.Location = new System.Drawing.Point(12, 97);

this.button1.Name = "button1";

this.button1.Size = new System.Drawing.Size(75, 23);

this.button1.TabIndex = 4;

this.button1.Text = "Clear cash";

this.button1.UseVisualStyleBackColor = true;

this.button1.Click += new System.EventHandler(this.button1\_Click);

//

// button2

//

this.button2.Location = new System.Drawing.Point(96, 97);

this.button2.Name = "button2";

this.button2.Size = new System.Drawing.Size(75, 23);

this.button2.TabIndex = 5;

this.button2.Text = "Reset";

this.button2.UseVisualStyleBackColor = true;

this.button2.Click += new System.EventHandler(this.button2\_Click);

//

// timer1

//

this.timer1.Tick += new System.EventHandler(this.timer1\_Tick);

//

// FormPreferences

//

this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);

this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;

this.AutoValidate = System.Windows.Forms.AutoValidate.EnablePreventFocusChange;

this.ClientSize = new System.Drawing.Size(182, 131);

this.Controls.Add(this.button2);

this.Controls.Add(this.button1);

this.Controls.Add(this.pictureBox1);

this.Controls.Add(this.checkBox2);

this.Controls.Add(this.checkBox1);

this.Controls.Add(this.textBox1);

this.Controls.Add(this.labelAddress);

this.FormBorderStyle = System.Windows.Forms.FormBorderStyle.FixedSingle;

this.Icon = global::ScrTilla.Properties.Resources.paomedia\_small\_n\_flat\_lightning;

this.MaximizeBox = false;

this.MinimizeBox = false;

this.Name = "FormPreferences";

this.ShowIcon = false;

this.ShowInTaskbar = false;

this.Text = "ScrTilla: FormPreferences";

this.FormClosing += new System.Windows.Forms.FormClosingEventHandler(this.FormPreferences\_FormClosing);

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

((System.ComponentModel.ISupportInitialize)(this.pictureBox1)).EndInit();

this.ResumeLayout(false);

this.PerformLayout();

}

#endregion

private System.Windows.Forms.Label labelAddress;

private System.Windows.Forms.TextBox textBox1;

private System.Windows.Forms.CheckBox checkBox1;

private System.Windows.Forms.CheckBox checkBox2;

private System.Windows.Forms.PictureBox pictureBox1;

private System.Windows.Forms.Button button1;

private System.Windows.Forms.Button button2;

private System.Windows.Forms.Timer timer1;

}

}

1. Листинг файла FormTaskbar.cs

using System;

using System.Windows.Forms;

namespace ScrTilla

{

public partial class FormTaskbar : Form

{

public FormTaskbar()

{

InitializeComponent();

notifyIcon1.ContextMenu = new ContextMenu(new MenuItem[] {

new MenuItem("Setting", n\_OpenForm),

new MenuItem("Exit", n\_Close)

});

//notifyIcon1.ShowBalloonTip(0, "ScrTilla", "Запущен", ToolTipIcon.Info);

}

private void n\_Close(object sender, EventArgs e)

{

Application.Exit();

}

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

{

this.Hide();

}

private bool PerefencesOpen = false;

private Action ToShowPreferences;

private void n\_OpenForm(object sender, EventArgs e)

{

if (!PerefencesOpen)

{

PerefencesOpen = true;

new ScrTilla.FormPreferences(PrefencesClosed, ref ToShowPreferences).ShowDialog();

}

else

{

ToShowPreferences.Invoke();

}

}

private void PrefencesClosed(object sender, FormClosedEventArgs e)

{

PerefencesOpen = false;

}

public void ClipboardSet(object sender, KeyEventArgs e)

{

if(Settings.Notifications) notifyIcon1.ShowBalloonTip(0, "ScrTilla", "Link copied", ToolTipIcon.Info);

}

}

}

}

1. Листинг файла FormTaskbar.Designer.cs

namespace ScrTilla

{

partial class FormTaskbar

{

/// <summary>

/// Required designer variable.

/// </summary>

private System.ComponentModel.IContainer components = null;

/// <summary>

/// Clean up any resources being used.

/// </summary>

/// <param name="disposing">true if managed resources should be disposed; otherwise, false.</param>

protected override void Dispose(bool disposing)

{

if (disposing && (components != null))

{

components.Dispose();

}

base.Dispose(disposing);

}

#region Windows Form Designer generated code

/// <summary>

/// Required method for Designer support - do not modify

/// the contents of this method with the code editor.

/// </summary>

private void InitializeComponent()

{

this.components = new System.ComponentModel.Container();

System.ComponentModel.ComponentResourceManager resources = new System.ComponentModel.ComponentResourceManager(typeof(FormTaskbar));

this.notifyIcon1 = new System.Windows.Forms.NotifyIcon(this.components);

this.SuspendLayout();

//

// notifyIcon1

//

this.notifyIcon1.Visible = true;

this.notifyIcon1.Icon = Properties.Resources.paomedia\_small\_n\_flat\_lightning;

//

// FormTaskbar

//

resources.ApplyResources(this, "$this");

this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;

this.ControlBox = false;

this.MaximizeBox = false;

this.MinimizeBox = false;

this.Name = "FormTaskbar";

this.Opacity = 0D;

this.ShowIcon = false;

this.ShowInTaskbar = false;

this.WindowState = System.Windows.Forms.FormWindowState.Minimized;

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

this.ResumeLayout(false);

}

#endregion

private System.Windows.Forms.NotifyIcon notifyIcon1;

}

}

1. Листинг файла Clipboard\_s.cs

using System;

using System.Runtime.InteropServices;

using System.Threading;

using System.Windows;

namespace ScrTilla

{

static class Clipboard\_s

{

[STAThread]

public static void ToClipboard(string Text)

{

// Выполнить операцию требуется в отдельном потоке

new Thread((delegate ()

{

// Эту операцию требуется делать, пока не будет результат

while (true)

{

lock (ProtectStaticThread) // Защитный замок

if (GetOpenClipboardWindow() == IntPtr.Zero) // Если свободен буфер обмена

{

try

{

// То отправляем текст в буфер обмена

Clipboard.SetText(Text);

}

catch

{

// В случае ошибки, пробуем закрыть буфер обмена с нашей стороны

CloseClipboard();

Thread.Sleep(1000);

continue;

}

// Очищаем память после работы с буфером обмена

GC.Collect();

GC.WaitForPendingFinalizers();

GC.Collect();

return;

}

else

{

// Если буфер обмена занят, ожидаем секундочку

Thread.Sleep(1000);

}

}

}))

// Запускать поток требуется в STA режиме

{ ApartmentState = ApartmentState.STA }.Start();

}

/// <summary>

/// Организует защитный ключ для однопоточного обращения к статическому методу

/// </summary>

private static object ProtectStaticThread = new object();

[DllImport("user32.dll")]

private static extern IntPtr GetOpenClipboardWindow();

[DllImport("user32.dll")]

private static extern bool CloseClipboard();

[DllImport("user32.dll")]

private static extern bool OpenClipboard(IntPtr hWndNewOwner);

}

}

1. Листинг файла Combine.cs

using System.Drawing;

using System.Net.Http;

using System.Net.Http.Headers;

using System.Windows.Forms;

using System.IO;

using System.Threading.Tasks;

namespace ScrTilla

{

static class Combine

{

/// <summary>

/// Получает изображение экрана.

/// Недостатки: не поддерживает масштабирование Windows,

/// не тестировался при нескольких мониторах,

/// не запрашивает область экрана, как и ...

/// ... не реагирует на зажатую клавишу alt

/// </summary>

/// <returns>Фото экрана</returns>

public static byte[] GetScreen()

{

// Получение размеров главного экрана и создание пустого изображения

Bitmap bitmap = new Bitmap(Screen.PrimaryScreen.Bounds.Width, Screen.PrimaryScreen.Bounds.Height);

// Создаём поверхность рисования

Graphics graphics = Graphics.FromImage(bitmap as Image);

// Срисовываем то, что на экране

graphics.CopyFromScreen(0, 0, 0, 0, bitmap.Size);

// Закрываем рисование

graphics.Dispose(); graphics = null;

// Отправляем результат

return ImageToByte(bitmap);

}

private static HttpClient cl = new HttpClient();

/// <summary>

/// Отправка изображения на сервер

/// </summary>

/// <param name="image">Изображение, которое следует отправить.</param>

/// <returns>Ответ от сервера</returns>

public static async Task<json\_st.ResponsePost> SendScreen(byte[] image)

{

// Создание контейнера для отправки

var requ = new MultipartFormDataContent();

// Конвектируем массив байтов в контент HTTP протокола

var ImContent = new ByteArrayContent(image);

// Указываем тип конента

ImContent.Headers.ContentType = MediaTypeHeaderValue.Parse("image/png");

// Добавление сведений о контенте в контейнере

requ.Add(ImContent, "up\_image", "image.png");

// Отправка изображения

HttpResponseMessage response = await cl.PostAsync(Settings.UPLOAD, requ);

// Уничтожение контейнера

requ.Dispose();

// Уничтожение контента HTTP

ImContent.Dispose();

// Конвектирование ответа JSON в структуру C#

return await Newtonsoft.Json.JsonConvert.DeserializeObjectAsync<json\_st.ResponsePost>

(await response.Content.ReadAsStringAsync());

}

/// <summary>

/// Получение сведений от сервера.

/// </summary>

/// <param name="HTTPaddress">Адрес севера в формате http://address. Если не задать, то значение будет взято Settings.HTTP\_ADDRESS</param>

/// <returns>Ответ от сервера</returns>

public static async Task<json\_st.InfoGet> GetInfo(string HTTPaddress = null)

{

try

{

return await Newtonsoft.Json.JsonConvert.DeserializeObjectAsync<json\_st.InfoGet>(

await (await cl.GetAsync((HTTPaddress == null || HTTPaddress.Length == 0 ? Settings.HTTP\_ADDRESS : HTTPaddress) + "/info", HttpCompletionOption.ResponseContentRead)).Content.ReadAsStringAsync());

}

catch

{

return new json\_st.InfoGet();//await new Task<json\_st.InfoGet>(delegate() { return new json\_st.InfoGet(); });

}

}

// http://stackoverflow.com/questions/7350679/convert-a-bitmap-into-a-byte-array

private static byte[] ImageToByte(Image img)

{

using (var stream = new MemoryStream())

{

img.Save(stream, System.Drawing.Imaging.ImageFormat.Png);

return stream.ToArray();

}

}

}

}

1. Листинг файла json\_strucs.cs

namespace ScrTilla

{

class json\_st

{

public class ResponsePost

{

public ResponsePost()

{

filename = message = "";

}

public string filename { get; set; }

public int code { get; set; }

public string message { get; set; }

}

public class InfoGet

{

public InfoGet()

{

version = url = upload\_dir = server\_name = "";

}

/// <summary>

/// Версия сервера.

/// </summary>

public string version;

/// <summary>

/// url сервера для сборки изображения.

/// </summary>

public string url;

/// <summary>

/// Адрес, куда нужно делать POST для загрузки изображения.

/// </summary>

public string upload\_dir;

/// <summary>

/// Человеческое имя сервера для отображения.

/// </summary>

public string server\_name;

public override string ToString()

{

return "version: " + version + "\r\nurl: " + url + "\r\nupload\_dir: " + upload\_dir + "\r\nserver\_name: " + server\_name;

}

}

}

}

1. Листинг файла Program.cs

using System;

using System.Diagnostics;

using System.Linq;

using System.Windows.Forms;

namespace ScrTilla

{

static class Program

{

/// <summary>

/// Запускает диагностику по работе с другими процессами.

/// </summary>

/// <returns>True - требуется закрыть приложение. False - программа может продолжать работу.</returns>

private static bool ProcessRepair()

{

try

{

// Приложение "ScrTilla.exe" было найдено в списке выполняемых процессов. Новый экземпляр приложения запускать опасно из-за повторного перехвата "Print Screen"; запускать опасно из-за файла настроек Settings.json, который используется всеми копиями процессов "ScrTilla.exe". Вы можете продолжить запуск нового экземпляра для подключения к иному серверу, но данная функция не предусмотрена данным приложением из-за статического экземпляра настроек.

switch (MessageBox.Show("The application is already running.\n\nThe application \"" + Process.GetCurrentProcess().ProcessName + "\" was found in the list of running processes. A new instance of the application is dangerous to run because of the repeated interception of the \"Print Screen\"; It's dangerous to run because of Settings.json settings file, which is used by all copies of the processes \"" + Process.GetCurrentProcess().ProcessName + "\". You can continue to start a new instance to connect to a different server, but this function is not provided by this application because of a static instance of the settings.\nContinue?",

Process.GetCurrentProcess().ProcessName, MessageBoxButtons.YesNo, MessageBoxIcon.Warning))

{

case DialogResult.Yes:

{

switch (MessageBox.Show("Close other processes with the name \"" + Process.GetCurrentProcess().ProcessName + "\"?\nYes - Close other.\nNo - Don't close processes.",

"ScrTilla.exe", MessageBoxButtons.YesNo, MessageBoxIcon.Information))

{

case DialogResult.Yes:

{

foreach (var p in Process.GetProcessesByName("ScrTilla.exe"))

{

if (p.Id != Process.GetCurrentProcess().Id) p.Close();

}

return false;

}

default: return false;

}

}

default: return true;

}

}

catch (Exception e)

{

MessageBox.Show(e.ToString(), Process.GetCurrentProcess().ProcessName, MessageBoxButtons.OK, MessageBoxIcon.Error);

return true;

}

}

static void Main(string[] args)

{

if (Process.GetProcesses().Count(p => p.ProcessName == Process.GetCurrentProcess().ProcessName) > 1)

{ // Найден иной процесс с данным названием

if(ProcessRepair()) return;

}

if (!System.IO.File.Exists("Newtonsoft.Json.dll"))

{ // Не найдена важнейшая библиотека для чтения и записи в стандарте Json

switch(MessageBox.Show("Newtonsoft.Json.dll not found!\nContinue?", "ScrTilla", MessageBoxButtons.YesNo, MessageBoxIcon.Error))

{

case DialogResult.No: return;

case DialogResult.None: return;

case DialogResult.Cancel: return;

}

}

PrtScr\_Hook.StartHook(PrtHooked);

using (ScrTilla.FormTaskbar f = new ScrTilla.FormTaskbar())

{

PrtScr\_Hook.PrintScreen += f.ClipboardSet;

Application.Run(f);

PrtScr\_Hook.PrintScreen -= f.ClipboardSet;

}

PrtScr\_Hook.StopHook(PrtHooked);

}

private static async void PrtHooked(object sender, KeyEventArgs e)

{

PrtScr\_Hook.StopHook(PrtHooked); // Во время обработки приостанавливаем перехват

json\_st.ResponsePost Resp = new json\_st.ResponsePost(); // Хранилище ответа

try

{

byte[] Scr = Combine.GetScreen(); // Шаг 2: получаем изображение

if (Settings.Save) SaveScr.Save(Scr); // Сохраняем при надобности на диск

Resp = await Combine.SendScreen(Scr); // Шаг 3: отправка изображения

Scr = null;

}

catch

{

Resp.message = "Error"; // Если не удалось отправить изображение

}

if (Resp.filename.Length > 4 && Resp.code != 415 && Resp.code != 0)

{ // Шаг 4: Если результат положительный, то отправить его в буфер обмена

Clipboard\_s.ToClipboard(Settings.HTTP\_ADDRESS + "/" + Resp.filename);

}

GC.Collect(); // Вызываем сборщик мусора дважды

PrtScr\_Hook.StartHook(PrtHooked); // Возобновление шага 1

GC.Collect();

}

}

}

1. Листинг файла PrtScr\_Hook.cs

using System;

using System.Collections.Generic;

using System.Runtime.InteropServices;

using System.Windows.Forms;

namespace ScrTilla

{

/// <summary>

/// Класс реализует слежку над клавишей Print Screen

/// </summary>

static class PrtScr\_Hook

{

/// <summary>

/// Вызывается при нажатии клавиши Print Screen

/// </summary>

public static event KeyEventHandler PrintScreen

{

add

{

gkH.KeyDown += value;

}

remove

{

gkH.KeyDown -= value;

}

}

/// <summary>

/// Статус, включено ли отслеживание клавиши PrintScreen

/// </summary>

public static bool Hook { get; private set; }

/// <summary>

/// Начинает слежку над клавишей Print Screen

/// </summary>

/// <param name="act">Метод, который должен вызываться при нажатии клавиши Print Screen</param>

public static void StartHook(KeyEventHandler act = null)

{

if (Hook == false)

{

if (act != null) PrintScreen += act;

gkH.hook();

Hook = true;

Application.ApplicationExit += AppExit;

}

}

/// <summary>

/// Завершение слежение над клавишей Print Screen

/// </summary>

/// <param name="act">Метод, который вызывался при нажатии клавиши Print Screen</param>

public static void StopHook(KeyEventHandler act = null)

{

if (Hook == true)

{

if(act != null) PrintScreen -= act;

gkH.unhook();

Hook = false;

}

Application.ApplicationExit -= AppExit;

}

private static globalKeyboardHook gkH = new globalKeyboardHook();

/// <summary>

/// Событие при заврешнеие приложения. Безопасное отключение.

/// </summary>

private static void AppExit(object sender, EventArgs e)

{

Application.ApplicationExit -= AppExit;

StopHook();

}

#region Чужой код[[4]](#footnote-4)

/// <summary>

/// A class that manages a global low level keyboard hook

/// Не смотря на то, что данный класс можно модернизировать под мои задачи, я этого не сделал из-за сохранения инакпсюляции и принципа чёрного ящика.

/// </summary>

protected class globalKeyboardHook

{

#region Constant, Structure and Delegate Definitions

/// <summary>

/// defines the callback type for the hook

/// </summary>

public delegate int keyboardHookProc(int code, int wParam, ref keyboardHookStruct lParam);

public struct keyboardHookStruct

{

public int vkCode;

public int scanCode;

public int flags;

public int time;

public int dwExtraInfo;

}

const int WH\_KEYBOARD\_LL = 13;

const int WM\_KEYDOWN = 0x100;

const int WM\_KEYUP = 0x101;

const int WM\_SYSKEYDOWN = 0x104;

const int WM\_SYSKEYUP = 0x105;

#endregion

#region Instance Variables

/// <summary>

/// Handle to the hook, need this to unhook and call the next hook

/// </summary>

IntPtr hhook = IntPtr.Zero;

public keyboardHookProc hookProcDelegate;

#endregion

#region Events

/// <summary>

/// Occurs when one of the hooked keys is pressed

/// </summary>

public event KeyEventHandler KeyDown;

/// <summary>

/// Occurs when one of the hooked keys is released

/// </summary>

public event KeyEventHandler KeyUp;

#endregion

#region Constructors and Destructors

/// <summary>

/// Initializes a new instance of the <see cref="globalKeyboardHook"/> class and installs the keyboard hook.

/// </summary>

public globalKeyboardHook()

{

hookProcDelegate = new keyboardHookProc(hookProc);

//hook();

}

/// <summary>

/// Releases unmanaged resources and performs other cleanup operations before the

/// <see cref="globalKeyboardHook"/> is reclaimed by garbage collection and uninstalls the keyboard hook.

/// </summary>

~globalKeyboardHook()

{

unhook();

}

#endregion

#region Public Methods

/// <summary>

/// Installs the global hook

/// </summary>

public void hook()

{

IntPtr hInstance = LoadLibrary("User32");

hhook = SetWindowsHookEx(WH\_KEYBOARD\_LL, hookProcDelegate, hInstance, 0);

}

/// <summary>

/// Uninstalls the global hook

/// </summary>

public void unhook()

{

if (hhook != IntPtr.Zero)

{

UnhookWindowsHookEx(hhook);

hhook = IntPtr.Zero;

}

}

/// <summary>

/// The callback for the keyboard hook

/// </summary>

/// <param name="code">The hook code, if it isn't >= 0, the function shouldn't do anyting</param>

/// <param name="wParam">The event type</param>

/// <param name="lParam">The keyhook event information</param>

/// <returns></returns>'

public int hookProc(int code, int wParam, ref keyboardHookStruct lParam)

{

if (code >= 0)

{

Keys key = (Keys)lParam.vkCode;

if (key == Keys.PrintScreen)

{

KeyEventArgs kea = new KeyEventArgs(key);

if ((wParam == WM\_KEYDOWN || wParam == WM\_SYSKEYDOWN) && (KeyDown != null))

{

KeyDown(this, kea);

}

else if ((wParam == WM\_KEYUP || wParam == WM\_SYSKEYUP) && (KeyUp != null))

{

KeyUp(this, kea);

}

if (kea.Handled)

return 1;

}

}

return CallNextHookEx(hhook, code, wParam, ref lParam);

}

#endregion

#region DLL imports

/// <summary>

/// Sets the windows hook, do the desired event, one of hInstance or threadId must be non-null

/// </summary>

/// <param name="idHook">The id of the event you want to hook</param>

/// <param name="callback">The callback.</param>

/// <param name="hInstance">The handle you want to attach the event to, can be null</param>

/// <param name="threadId">The thread you want to attach the event to, can be null</param>

/// <returns>a handle to the desired hook</returns>

[DllImport("user32.dll")]

static extern IntPtr SetWindowsHookEx(int idHook, keyboardHookProc callback, IntPtr hInstance, uint threadId);

/// <summary>

/// Unhooks the windows hook.

/// </summary>

/// <param name="hInstance">The hook handle that was returned from SetWindowsHookEx</param>

/// <returns>True if successful, false otherwise</returns>

[DllImport("user32.dll")]

static extern bool UnhookWindowsHookEx(IntPtr hInstance);

/// <summary>

/// Calls the next hook.

/// </summary>

/// <param name="idHook">The hook id</param>

/// <param name="nCode">The hook code</param>

/// <param name="wParam">The wparam.</param>

/// <param name="lParam">The lparam.</param>

/// <returns></returns>

[DllImport("user32.dll")]

static extern int CallNextHookEx(IntPtr idHook, int nCode, int wParam, ref keyboardHookStruct lParam);

/// <summary>

/// Loads the library.

/// </summary>

/// <param name="lpFileName">Name of the library</param>

/// <returns>A handle to the library</returns>

[DllImport("kernel32.dll")]

static extern IntPtr LoadLibrary(string lpFileName);

#endregion

}

#endregion Чужой код

}

}

1. Листинг файла SaveScr.cs

using System;

using System.IO;

namespace ScrTilla

{

static class SaveScr

{

/// <summary>

/// Имя директории, где лежит файл

/// </summary>

private static readonly string DIRECT\_NAME = Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData) + "\\ScrTilla\\Cashe";

static SaveScr()

{

if (!Directory.Exists(DIRECT\_NAME)) Directory.CreateDirectory(DIRECT\_NAME);

}

/// <summary>

/// Сохранение изображения на диск.

/// </summary>

/// <param name="input">Данные, которые следует сохранить.</param>

public static async void Save(byte[] input)

{

// Если папка, в которой должен храниться файл не существует,

if (!Directory.Exists(DIRECT\_NAME))

// То создать папку

Directory.CreateDirectory(DIRECT\_NAME);

FileStream fp = null;

try

{

// Открытие потока записи файла

using (

fp = new FileStream

(DIRECT\_NAME + DateTime.Now.ToString("\\\\yyyy-MM-dd hh-mm-ss.pn\\g"), FileMode.CreateNew)

)

{

// Асихронная запись инофрмации в файл.

await fp.WriteAsync(input, 0, input.Length);

}

}

catch (IOException)

{

// Если с файлом что-то случилось, и ссылка на поток существует, то поток следует закрыть

fp?.Close();

}

catch { }

}

internal static void Clear()

{

if(Directory.Exists(DIRECT\_NAME)) Directory.Delete(DIRECT\_NAME, true);

Directory.CreateDirectory(DIRECT\_NAME);

}

}

}

1. Листинг файла Settings.cs

using System;

using System.IO;

using Newtonsoft.Json;

using System.Threading.Tasks;

namespace ScrTilla

{

static class Settings // Статический, так как одно приложение имеет один файл настроек

{

private struct Conteiner

{

/// <summary>

/// Место загрузки изображений

/// </summary>

public string UPLOAD;

/// <summary>

/// Адрес сервера с конкетенацией: http://ADDRESS

/// </summary>

public string HTTP\_ADDRESS;

/// <summary>

/// Место хранений изображений (не используется?)

/// </summary>

public string PNGs;

/// <summary>

/// Адрес сервера DNS или IP или полное местоположение обработчика на сервере

/// Пример: s.mtudev.ru

/// </summary>

public string ADDRESS;

/// <summary>

/// Пользователь хочет сохранять изображения на компьютере

/// </summary>

public bool Save;

/// <summary>

/// Пользователь хочет получать уведомления

/// </summary>

public bool Notifications;

/// <summary>

/// Создаёт экземпляр с настройками по-умолчанию.

/// </summary>

public static Conteiner Default

{

get

{

return

new Conteiner()

{

UPLOAD = "http://s.mtudev.ru/upload",

HTTP\_ADDRESS = "http://s.mtudev.ru",

PNGs = "http://s.mtudev.ru",

ADDRESS = "s.mtudev.ru",

Save = false,

Notifications = false

}

;

}

}

}

/// <summary>

/// Установить PNGs и URI в зависимости от address.

/// </summary>

/// <param name="address">DNS или IP сервера.</param>

/// <returns>true, если настройки применены успешно.</returns>

public static async Task<bool> SetSettingsByAddress(string address)

{

if (address == null) return false;//UPLOAD = HTTP\_ADDRESS = ADDRESS = PNGs = "";

json\_st.InfoGet info = await Combine.GetInfo("http://" + address);

if (info.server\_name == null || info.server\_name.Length == 0 || info.upload\_dir == null || info.upload\_dir.Length == 0) return false;

cont.ADDRESS = address;

cont.HTTP\_ADDRESS = "http://" + address;

cont.UPLOAD = HTTP\_ADDRESS + info.upload\_dir;

cont.PNGs = HTTP\_ADDRESS;

update();

return true;

}

/// <summary>

/// Имя директории, где лежит файл

/// </summary>

private static readonly string DIRECT\_NAME = Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData) + "\\ScrTilla";

/// <summary>

/// Имя файла

/// </summary>

private static readonly string FILE\_NAME = (Environment.SpecialFolder.LocalApplicationData) + "\\ScrTilla\\settings.json";

/// <summary>

/// Контейнер, которых хранит в оперативной памяти настройки

/// </summary>

private static Conteiner cont;

/// <summary>

/// Сбрасывает настройки по-умлочанию.

/// </summary>

public static void ToDefault()

{

cont = Conteiner.Default;

update();

}

/// <summary>

/// Принудительно вызвать конструктор класса

/// </summary>

public static void RefreshClass()

{

StreamReader fp = null;

try

{

if (File.Exists(FILE\_NAME))

{

using (fp = new StreamReader(FILE\_NAME, System.Text.Encoding.UTF8))

{

cont = JsonConvert.DeserializeObject<Conteiner>(fp.ReadToEnd());

}

// если что-то не нашлось в настройках, создаём это сами:

// При таком подходе вполне возможно, что будет ADDRESS указывать на одно место, а HTTP\_ADDRESS на default

// Чтобы это исправить, пользователю надо будет поприменить настройки в форме.

Conteiner Def = Conteiner.Default;

cont.ADDRESS = cont.ADDRESS == null ? Def.ADDRESS : cont.ADDRESS;

cont.HTTP\_ADDRESS = cont.HTTP\_ADDRESS == null ? Def.HTTP\_ADDRESS : cont.HTTP\_ADDRESS;

cont.PNGs = cont.PNGs == null ? Def.PNGs : cont.PNGs;

cont.UPLOAD = cont.UPLOAD == null ? Def.UPLOAD : cont.UPLOAD;

}

else

{

CreateNewSettingFile();

}

}

catch (JsonException)

{

fp?.Close();

CreateNewSettingFile();

}

catch (Exception e)

{

System.Windows.Forms.MessageBox.Show("Нет доступа к \"" + FILE\_NAME + "\"\n\n" + e.Message, "ScrTilla", System.Windows.Forms.MessageBoxButtons.OK, System.Windows.Forms.MessageBoxIcon.Warning);

}

}

/// <summary>

/// Конструктор статического класса

/// </summary>

static Settings()

{

cont = new Conteiner();

RefreshClass();

}

/// <summary>

/// Создаёт новый файл настроек

/// </summary>

private static void CreateNewSettingFile()

{

cont = Conteiner.Default;

if (!Directory.Exists(DIRECT\_NAME))

{

Directory.CreateDirectory(DIRECT\_NAME);

}

if (File.Exists(FILE\_NAME))

{

File.Delete(FILE\_NAME);

}

update();

}

#region Свойства

/// <summary>

/// Свойство представляет собой адрес до загрузки фотографий

/// </summary>

public static string UPLOAD

{

get

{

return cont.UPLOAD;

}

set

{

cont.UPLOAD = value;

update();

}

}

/// <summary>

/// Свойство представляет собой конкетенацию http:// и ADDRESS

/// </summary>

public static string HTTP\_ADDRESS

{

get

{

return cont.HTTP\_ADDRESS;

}

set

{

cont.HTTP\_ADDRESS = value;

update();

}

}

/// <summary>

/// Путь до места хранения PNGs (не используется?)

/// </summary>

public static string PNGs

{

get

{

return cont.PNGs;

}

set

{

cont.PNGs = value;

update();

}

}

/// <summary>

/// Адрес как DNS или IP сервера

/// </summary>

public static string ADDRESS

{

get

{

return cont.ADDRESS;

}

set

{

cont.ADDRESS = value;

update();

}

}

/// <summary>

/// Получает или задаёт: нужно ли сохранять изображение на компьютере?

/// </summary>

public static bool Save

{

get

{

return cont.Save;

}

set

{

cont.Save = value;

update();

}

}

/// <summary>

/// Получает или задаёт: Нужно ли присылать пользователю уведомления?

/// </summary>

public static bool Notifications

{

get

{

return cont.Notifications;

}

set

{

cont.Notifications = value;

update();

}

}

#endregion Свойства

/// <summary>

/// Записать изменения в файл настроек

/// </summary>

private static void update()

{

StreamWriter fp = null;

try

{

using (fp = new StreamWriter(FILE\_NAME, false, System.Text.Encoding.UTF8))

{ // Записать в файл настроек все настройки программы в JSON формате

fp.Write(JsonConvert.SerializeObject(cont));

}

}

catch { fp?.Close(); }

}

}

}

1. Орлов С.А., Программная инженерия. Технологии разработки программного обеспечения – 5-у изд., обновлённое и доп. – СПб.: Питер, 2016. – 640 с. [↑](#footnote-ref-1)
2. Агуров П.В., C#. Сборник рецептов. – СПб.: БХВ-Петербург, 2007. – 432 с.: ил. ISBN 5-94157-969-1 [↑](#footnote-ref-2)
3. Алекс Дэвис, Асинхронное программирование в C# 5.0. / Пер. с англ. Слинкин А. А. – М.: ДМК Пресс, 2013. – 120 с.: ил. ISBN 978-5-94074-886-1 [↑](#footnote-ref-3)
4. Код взят из официального форума сайта <http://microsoft.com>/ [↑](#footnote-ref-4)