Статья Делаем «Android» который пишет сообщения в «Viber», регистрирует аккаунты в «ВКонтакте» и смотрит «YouTube» без вашего участия

Здравствуйте дорогие форумчане, в своей статье я расскажу про автоматизацию действий в приложениях на устройстве под управлением операционной системы «Android».

Введение

Сфера применения автоматизации в мобильных приложениях довольно широка, начиная от реализации автоматических регистраций заканчивая всеми возможными спамерами, кликерами, парсерами, брутерами, чекерами и т.д. В большинстве случаев запросы к серверам исходящих от официальных клиентских приложений имеют намного меньше ограничений или не имеют их вообще, в отличии от запросов из браузера или с использованием сторонних API функций, что открываем невероятное поле для творчества.

В статье будет использоваться язык программирования «C#» и среда разработки «Visual Studio 2019», все описанное без проблем можно будет реализовать и на других языках, главное, что бы под них были необходимые библиотеки. Возможно в процессе просмотра приведенного в статье кода возникнет вопрос, почему он не написан более грамотно? По моему мнению, простая «топорная» реализация нагляднее и требует минимальные знания программирования, благодаря чему порог входа в тему автоматизации приложений на «Android» станет меньше, и большее количество людей смогут реализовать свои идеи на практике, чего собственно я и стараюсь добиться. Главная цель статьи, показать инструменты для автоматизации и ответить на вопрос «Как».

Теоретические основы

В качестве устройства под управлением операционной системы «Android» будет использован эмулятор «Nox», вместо него может быть эмулятор «Genymotion» или физическое устройство подключенное через «USB».

В «Android SDK» есть командная строка «ADB» (Android Debug Bridge - отладочный мост «Android»), с помощью нее возможна отправка команд на ваш смартфон подключенный в режиме отладки через «USB». Одна из команд «ADB» позволяет передавать нажатия по экрану в определенных координатах, другая передавать нажатия системных клавиш (в конце будет список этих команд и коды основных

клавиш). Этого достаточно для реализации простейших алгоритмов, но если необходимо получать и передавать текстовые данные с экрана или элементы с которыми необходимо взаимодействовать не зафиксированы на одном месте, то нужен более комплексный подход.

Первую задачу, которую необходимо решить в данном случае, это поиск элемента. Компоновка элементов приложения «Android» представляет собой «XML» структуру, для просмотра которой мы будем использовать «UIAutomatorviewer» (графический инструмент для распознавания компонентов пользовательского интерфейса в «Android» приложении), с помощью него мы получим информацию о необходимом нам элементе, а именно все его атрибуты. Дальше нам необходимо реализовать взаимодействие с этим элементом из нашего кода на C#, для этого будем использовать фреймворк «Appium», по факту он предназначен для тестирования мобильных приложений, но подойдет и для наших нужд. С помощью «Appium» мы сможем находить элементы интерфейса на экране и взаимодействовать с ними через их атрибуты а также получать и передавать текст (который так же является атрибутом). Таким образом переходя от элемента к элементу, взаимодействуя с ними, передавая или получая необходимые значения атрибутов мы можем реализовывать различные алгоритмы, автоматизируя работу в приложениях на «Android» вне зависимости от расположения элементов на экране.

Подготовка

Для реализации всего вышесказанного нам понадобиться:

- «Nox» (https://ru.bignox.com)
- «Appium» (https://appium.io)
- «Java Development Kit» (https://www.oracle.com/ru/java/technologies/javase/javase-jdk8-downloads.html)
- «Android SDK Build-Tools» (https://androidsdkmanager.azurewebsites.net/Buildtools)
- «UIAutomatorviewer» (входит в состав «Android SDK Tools» https://dl.google.com/android/repository/tools_r25.2.3-windows.zip)

После скачивания устанавливаем «Nox», «Appium» и «Java Development Kit», думаю трудностей возникнуть не должно.

Далее настраиваем эмулятор «Nox». Создаем папку «tools» в месте установки эмулятора «C:\Program Files\Nox\bin» и разархивируем туда «Android SDK Build-Tools». После чего запускаем мультиплеер «Nox», в нем можем использовать эмулятор по умолчанию или создать новый, запускаем, заходим в настройки системы «Android»,

включаем режим разработчика («Настройки» - «О планшете» - «Номер сборки», щелкаем по полю несколько раз пока не появиться уведомление), затем в появившейся новой вкладке для разработчиков активируем пункт «Отладка по USB».

После заходим в настройки эмулятора (значок в правом верхнем углу окна) и во вкладке «Общие настройки» активируем «Рут».

Также во вкладке «Настройка производительности» можете изменить настройки разрешения на более компактные. После этого выключаем эмулятор.

Далее настроим фреймворк «Appium», запускаем его, в главном окне нажимаем «Edit Configuration», в поле «ANDROID_HOME» указываем путь к папке «bin» эмулятора «Nox», в поле «JAVA_HOME» указываем путь к «JDK», должно получится как на скриншоте ниже:

Теперь нам требуется выйти из приложение и перезагрузить компьютер. После перезагрузки снова запустим «Appium» и перейдем во вкладку «Advanced». Необходимо в поле «Server Address» указать «127.0.0.1» и ниже отметить пункт «Allow Session Override», в итоге должно получиться как на скриншоте ниже:

Поле «Server Port» оставляем как есть, по умолчанию «4723», именно через него мы будем взаимодействовать с сервером «Appium». На данном этапе сервер пока не запускаем.

Далее запустим «Visual Studio», создадим новое консольное приложение и установим требуемые пакеты через диспетчер пакетов «NuGet» в «Visual Studio». Находим и устанавливаем «Appium.WebDriver» и «DotNetSeleniumExtras.WaitHelpers» со всеми требуемыми зависимостями.

После чего запускаем ранее созданный нами эмулятор «Nox» и переходим к настройке «UIAutomatorviewer», в принципе настраивать там нечего, главное запустить. Архив с «Android SDK Tools» разархивируйте по любому пути, главное, чтобы он состоял только из латинских букв. Проблема при запуске «UIAutomatorviewer» может возникнуть из-за несовпадении версий «ADB» используемой эмулятором «Nox» и «Android SDK Tools», для того что бы этого не произошло необходимо скопировать файлы «adb.exe», «AdbWinApi.dll» и «AdbWinUsbApi.dll» расположенные в месте установки эмулятора «Nox» в папку «tools» расположенную в месте распаковки «Android SDK Tools». После чего из этой папки запускаем «uiautomatorviewer.bat» и нажимаем на иконку сверху, если в окне появился скриншот экрана эмулятора то все работает, наведя указателем мыши на компонент он подсветиться и слева отобразится его место в иерархии структуры экрана а также его атрибуты, щелкнув по элементы мы зафиксируем его параметры.

Как было написано выше, компоновка элементов не экране представляет собой «XML» структуру, для поиска в ней необходимых компонентов воспользуемся языком запросов «XPath», в рамках данной статьи мною будут использованы атрибуты «text», «resource-id», «content-desc» и «class», т.к. совокупность этих значений чаще всего уникальны для каждого компонента экрана, что позволит не делать сложные запросы. Если компонент не имеет уникальных идентификаторов то такой способ не подойдет, но есть еще два, первый это составление сложного запроса «XPath» который ищет опираясь на взаимное расположение компонентов, второй это передача нажатия по экрану в требуемых координатах с помощью передачи команды через «ADB».

Есть важная особенность, **«UIAutomatorviewer» и «Appium» одновременно работать не будут**, по этой причине сервер «Appium» мы будем запустим непосредственно перед запуском нашего консольного приложения.

Теперь остался последний подготовительный шаг, это узнать порт через который произошло подключение к «ADB», системное имя необходимого приложения и его текущего окна (так называемого активити «Activity») для отправки команд приложению, запущенному в эмуляторе «Nox». Для этого запустим из командной строки «Windows» приложение «adb.exe» расположенное в месте установки эмулятора «Nox» с параметром «device». В моем случае:
Code:

```
"C:\Program Files\Nox\bin\adb.exe" devices
```

В результате получим:
Code:

```
List of devices attached
127.0.0.1:62035 device
```

Как видим, наш эмулятор подключен к «127.0.0.1:62035», теперь необходимо получить информации о приложении, для этого необходимо перейти в режим консоли отправив команду «shell». Если подключено несколько эмуляторов то перед командой нужно добавить «-s 127.0.0.1:62035», что обозначит на какой именно эмулятор будет отправлена команда. В моем случае команда следующая:
Code:

```
"C:\Program Files\Nox\bin\adb.exe" shell
```

После этого мы зайдем в консоль «Android» (по факту это Linux-консоль). Для получения идентификаторов приложения нам необходимо отправить команду:
Code:

```
dumpsys window windows | grep -E 'mCurrentFocus|mFocusedApp'
```

В результате получим:
Code:

```
mCurrentFocus=Window{1ca0e2ba u0 com.vphone.launcher/com.vphone.launcher.Launcher}
mFocusedApp=null
```

Системное имя необходимого приложения и название его активити расположены в первой строчке. Так как мы предварительно не запустили нужное нам приложение, то в ответ получили информацию о запущенном лаунчере, системное имя которого «com.vphone.launcher» а название его текущего активити «.Launcher». Для получения информации о другом приложении нам необходимо будет его установить, запустить и отправить соответствующую команду снова. Установку всех приложений рекомендую осуществлять через скаченные пакеты «APK», а не через «Play Market». **Иногда полученный таким образом активити не совпадает со стартовым**, в таком случае узнать стартовый можно через приложение «Activity Launcher», которое необходимо установить на эмулятор.

Теперь перейдем в «Visual Studio» и пропишем список используемых ресурсов:
C#:

```
using System;
using System.Text;
using System.Diagnostics;
using System.IO;
using System.Net;
using System.Threading;
using OpenQA.Selenium;
using OpenQA.Selenium.Appium;
using OpenQA.Selenium.Appium.Android;
using OpenQA.Selenium.Appium.Enums;
using OpenQA.Selenium.Support.UI;
```

А также напишем методы, которые будут использоваться для всех описываемых ниже примеров.
Передача команд эмулятору через консоль «ADB»:
C#:

```
//параметры:
//string pathAdb - расположение файла «adb.exe»,
//string device – IP адрес и порт через который подключен эмулятор
//string command - сама команда передаваемая эмулятору
//возвращает результат отправленной команды
public static string AdbCommand(string pathAdb, string device, string command)
{
    command = String.Format("-s {0} {1}", device, command);
    ProcessStartInfo psiOpt = new ProcessStartInfo(pathAdb, command);
    psiOpt.WindowStyle = ProcessWindowStyle.Hidden;
    psiOpt.RedirectStandardOutput = true;
    psiOpt.UseShellExecute = false;
    psiOpt.CreateNoWindow = true;
    psiOpt.StandardOutputEncoding = Encoding.UTF8;
    Process procCommand = Process.Start(psiOpt);
    StreamReader srIncoming = procCommand.StandardOutput;
    Encoding E = srIncoming.CurrentEncoding;
    procCommand.WaitForExit();
    string rez = srIncoming.ReadToEnd();
    return rez;
}
```

Инициализация сессии «Appium»:
C#:

```
//параметры:
//string device – IP адрес и порт через который подключен эмулятор
//string appName - системное имя приложения
//string appActivity – имя текущего окна приложения
//string appiumPort – порт сервера «Appium»
//bool reset – требуется ли сбрасывать состояние приложения
//возвращает класс «AndroidDriver» через который мы будем взаимодействовать с компонентами на
экране
public static AndroidDriver<IWebElement> Initialization(string device, string appName, string
appActivity, string appiumPort, bool reset)
{
    AndroidDriver<IWebElement> driver;
    DriverOptions capabilities = new AppiumOptions();
    capabilities.AddAdditionalCapability(MobileCapabilityType.AutomationName,
AutomationName.Appium);
    capabilities.AddAdditionalCapability(MobileCapabilityType.DeviceName, "Android Emulator "
+ device);
    capabilities.AddAdditionalCapability(MobileCapabilityType.Udid, device);
    capabilities.AddAdditionalCapability(MobileCapabilityType.NewCommandTimeout, 180);
    capabilities.AddAdditionalCapability(MobileCapabilityType.NoReset, reset);
    capabilities.AddAdditionalCapability(AndroidMobileCapabilityType.AppActivity,
appActivity);
    capabilities.AddAdditionalCapability(AndroidMobileCapabilityType.AppPackage, appName);
    driver = new AndroidDriver<IWebElement>(new Uri(String.Format("http://127.0.0.1:
{0}/wd/hub", appiumPort)), capabilities);
    return driver;
}
```

Взаимодействие с компонентами на экране:
C#:

```
//параметры:
//string driver – результат инициализации сессии «Appium»
//string class- значение атрибута «Class» необходимого компонента
//string attributeName– название атрибута по которому ищем компонент
//string attributeValue– значение атрибута по которому ищем компонент
//string timeWait– максимальное время ожидания появления элемента на экране
//string text– передаваемый элементу текст

//проверка наличия компонента на экране
public static bool ElementChek(AndroidDriver<IWebElement> driver, string attributeClass,
string attributeName, string attributeValue, double timeWait)
{
    try
    {
        WebDriverWait wait = new WebDriverWait(driver, TimeSpan.FromSeconds(timeWait));

wait.Until(SeleniumExtras.WaitHelpers.ExpectedConditions.PresenceOfAllElementsLocatedBy(By.XPa
[contains(@{1}, '{2}')]", attributeClass, attributeName, attributeValue))));
        return true;
    }
    catch (WebDriverTimeoutException)
    {
        return false;
    }
}
//нажатие на компонент
public static void ElementTap(AndroidDriver<IWebElement> driver, string attributeClass,
string attributeName, string attributeValue)
{
    if (ElementChek(driver, attributeClass, attributeName, attributeValue, 5))
        driver.FindElement(By.XPath(String.Format("//{0}[contains(@{1}, '{2}')]",
attributeClass, attributeName, attributeValue))).Click();
    else
        Console.WriteLine(String.Format("Компонент attributeClass:{0}, attributeName:{1},
attributeValue:{2} не найден", attributeClass, attributeName, attributeValue));

}
//получение текстового значения компонента
public static string ElementTextGet(AndroidDriver<IWebElement> driver, string attributeClass,
string attributeName, string attributeValue)
{
    if (ElementChek(driver, attributeClass, attributeName, attributeValue, 5))
        return driver.FindElement(By.XPath(String.Format("//{0}[contains(@{1}, '{2}')]",
attributeClass, attributeName, attributeValue))).Text;
    else
        Console.WriteLine(String.Format("Компонент attributeClass:{0}, attributeName:{1},
attributeValue:{2} не найден", attributeClass, attributeName, attributeValue));
    return null;
}
//отправка текстового значения компонента
public static void ElementTextSend(AndroidDriver<IWebElement> driver, string attributeClass,
```

```
string attributeName, string attributeValue, string text)
{
    if (ElementChek(driver, attributeClass, attributeName, attributeValue, 5))
        driver.FindElement(By.XPath(String.Format("//{0}[contains(@{1}, '{2}')]",
attributeClass, attributeName, attributeValue))).SendKeys(text);
    else
        Console.WriteLine(String.Format("Компонент attributeClass:{0}, attributeName:{1},
attributeValue:{2} не найден", attributeClass, attributeName, attributeValue));
}
```

Как вы уже наверное заметили, перед каждым взаимодействием с компонентом происходит проверка его наличия на экране, это необходимо для исключения ошибок по причине задержки отклика интерфейса приложения.

На этом подготовительную часть можно считать завершенной, мною специально не были упомянуты конкретные приложения так как она универсальна а примеры будут продемонстрированы дальше.

Отправка сообщений в «Viber»

Устанавливаем, открываем и авторизуемся в «Viber» на эмуляторе, получаем всю необходимую информацию о запущенном приложении через «ADB», способом указанные выше, используя полученные значения инициализируем сессию «Appium» и указываем контакт которому будем отправлять сообщение и его текст:
C#:

```
string device = "127.0.0.1:62035";
string appName = "com.viber.voip";
string appActivity = ".HomeActivity";
string appiumPort = "4723";
string contact = ""; //контакт которому отправляем сообщение
string message = ""; //текст сообщения
AndroidDriver<IWebElement> driver = Initialization(device, appName, appActivity, appiumPort,
false);
```

После чего открываем «UIAutomatorviewer» и просматриваем параметры необходимых компонентов.

Переходим к реализации алгоритма:
C#:

```
//шаг 1
ElementTap(driver, "android.widget.ImageButton", "resource-id", "fab_compose");
//шаг 2
ElementTextSend(driver, "android.widget.EditText", "resource-id", "search_src_text",
contact);
//шаг 3
ElementTap(driver, "android.widget.LinearLayout", "resource-id", "new_num_layout");
//шаг 4
if (ElementChek(driver, "android.widget.TextView", "resource-id", "alertTitle", 5))
{
    Console.WriteLine(String.Format("Контакт {0} не найден", contact));
}
else
{
    //шаг 5
    ElementTextSend(driver, "android.widget.EditText", "resource-id", "send_text", message);
    //шаг 6
    ElementTap(driver, "android.widget.FrameLyout", "resource-id", "btn_send");
    Console.WriteLine("Сообщение отправлено");
}
driver.CloseApp();
Console.ReadKey();
```

Нечто подобное можно реализовать почти в любом мессенджере, для реализации массовых рассылок необходимо добавление регистрации нового аккаунта при блокировке старого и использование прокси серверов.

Регистрация в «ВКонтакте»

Для регистрации аккаунта мы будем использовать виртуальные номера, которые можно арендовать на одном из множества сайтов. Мною в статье будет использован сайт «sms-activate.ru», можете использовать любой другой где есть возможность работы через API. Чтобы заказать номер для регистрации в «ВКонтакте» воспользуемся запросом:

`http://sms-activate.ru/stubs/handler_api.php?api_key=$api_key&action=getNumber&service=vk&country=0`

Вместо «$api_key» указываем свой ключ API. В качестве ответа получим строку вида:

`ACCESS_NUMBER:$id:$number`

Где «$id» это код запроса, «$number» номер для активации. После чего производим регистрацию с присланным номером, необходимо немного подождать пока придет код и получить его используя запрос:

`http://sms-activate.ru/stubs/handler_api.php?api_key=$api_key&action=getStatus&id=$id`

Если код еще не пришел то в ответ получим:

`STATUS_WAIT_CODE`

Если код пришел то:

```
STATUS_OK:$code
```

Где «$code» код активации.

В «Visual Studio» получение номера телефона для регистрации будет выглядеть таким образом:

C#:

```
string apiKey = ""; //API-ключ
string resultGetPhone;
HttpWebRequest requestGetPhone = (HttpWebRequest)WebRequest.Create(String.Format("http://sms-
activate.ru/stubs/handler_api.php?api_key={0}&action=getNumber&service=vk&country=0",
apiKey));
HttpWebResponse responseGetPhone = (HttpWebResponse)requestGetPhone.GetResponse();
using (StreamReader sr = new StreamReader(responseGetPhone.GetResponseStream(),
Encoding.UTF8, true))
{
    resultGetPhone = sr.ReadToEnd();
    sr.Close();
}
string id = resultGetPhone.Split(':')[1];
string phone = resultGetPhone.Split(':')[2].Substring(1);
```

Получение кода активации:

C#:

```
string code = null;
while (code == null)
{
    string resultGetCode = "";
    HttpWebRequest requestGetCode =
(HttpWebRequest)WebRequest.Create(String.Format("http://sms-
activate.ru/stubs/handler_api.php?api_key={0}&action=getStatus&id={1}", apiKey, id));
    HttpWebResponse responseGetCode = (HttpWebResponse)requestGetCode.GetResponse();
    using (StreamReader sr = new StreamReader(responseGetCode.GetResponseStream(),
Encoding.UTF8, true))
    {
        resultGetCode = sr.ReadToEnd();
        sr.Close();
    }
    if (resultGetCode != "STATUS_WAIT_CODE")
    {
        code = resultGetCode.Split(':')[1];
    }
    else
    {
        Thread.Sleep(5000);
    }
}
```

Эту часть кода необходимо будет вставить в момент ожидания прихода кода в приложении.

Также добавим простенький генератор пароля, который будем использовать при регистрации:
C#:

```
string line = "qwertyuiopasdfghjklzxcvbnm0123456789";
Random rnd = new Random();
char[] pwd = new char[10];
for (int i = 0; i < pwd.Length; i++)
    pwd[i] = line[rnd.Next(line.Length)];
string password = new string(pwd);
```

Далее получим всю необходимую информацию об эмуляторе, приложении и инициализируем сессию «Appium». Обратите внимание что **в вызове метода «Initialization» передается значение «true» параметра «reset»**, это обусловлено тем что при запуске приложения «ВКонтакте» нам необходимо сбросить состояние приложения для регистрации нового аккаунта:
C#:

```
string device = "127.0.0.1:62035";
string appName = "com.vkontakte.android";
string appActivity = ".MainActivity";
string appiumPort = "4723";
AndroidDriver<IWebElement> driver = Initialization(device, appName, appActivity, appiumPort,
true);
```

Затем просматриваем параметры необходимых компонентов.

Реализуем все вышеописанные шаги:
C#:

```
//шаг 1
ElementTap(driver, "android.widget.TextView", "resource-id", "sign_up_button");
//шаг 2
ElementTextSend(driver, "android.widget.EditText", "resource-id", "phone_edit_text", phone);
//шаг 3
ElementTap(driver, "android.widget.FrameLayout", "resource-id", "continue_btn");
//шаг 4
//получение кода активации
string code = null;
while (code == null)
{
    string resultGetCode = "";
    HttpWebRequest requestGetCode =
(HttpWebRequest)WebRequest.Create(String.Format("http://sms-
activate.ru/stubs/handler_api.php?api_key={0}&action=getStatus&id={1}", apiKey, id));
    HttpWebResponse responseGetCode = (HttpWebResponse)requestGetCode.GetResponse();
    using (StreamReader sr = new StreamReader(responseGetCode.GetResponseStream(),
Encoding.UTF8, true))
    {
        resultGetCode = sr.ReadToEnd();
        sr.Close();
    }
    if (resultGetCode != "STATUS_WAIT_CODE")
    {
        code = resultGetCode.Split(':')[1];
    }
    else
    {
        Thread.Sleep(5000);
    }
}
//вставляем полученный код в соответствующее поле
ElementTextSend(driver, "android.widget.EditText", "resource-id", "code_edit_text", code);
ElementTap(driver, "android.widget.FrameLayout", "resource-id", "continue_btn");
//шаг 5
ElementTextSend(driver, "android.widget.EditText", "resource-id", "first_name", "Иван");
ElementTextSend(driver, "android.widget.EditText", "resource-id", "last_name", "Иванович");
//шаг 6
ElementTap(driver, "android.widget.FrameLayout", "resource-id", "continue_btn");
//шаг 7
ElementTap(driver, "android.widget.TextView", "resource-id", "choose_birthday");
ElementTap(driver, "android.widget.Button", "resource-id", "button1");
ElementTap(driver, "android.widget.FrameLayout", "resource-id", "continue_btn");
//шаг 8
ElementTextSend(driver, "android.widget.EditText", "resource-id", "vk_password", password);
ElementTextSend(driver, "android.widget.EditText", "resource-id", "vk_repeat_password",
password);
ElementTap(driver, "android.widget.FrameLayout", "resource-id", "continue_btn");
//шаг 9
ElementTap(driver, "android.widget.TextView", "resource-id", "skip");
//шаг 10
```

```
ElementTap(driver, "android.widget.TextView", "resource-id", "skip");
//шаг 11
ElementTap(driver, "android.widget.TextView", "resource-id", "skip");

driver.CloseApp();
Console.WriteLine(String.Format("{0}:{1}", phone, password));
Console.ReadKey();
```

Таким образом в результате имеем пару логин, пароль для входа в созданный аккаунт, пример демонстрирует получение номера, кода активации и регистрацию в приложении одного аккаунта. Для реализации автоматических регистраций большего количества аккаунтов необходимо предусмотреть использование прокси серверов.

Просмотр видео в YouTube

Для просмотра необходимого видео нам необходим будет его идентификатор, который мы возьмем из адресной строки браузера. Например из ссылки на видео «https://www.youtube.com/watch?v=5qap5aO4i9A» идентификатором на видео будет «5qap5aO4i9A».

При получении необходимой информации о приложении через консоль мы получим имя активити «.WatchWhileActivity» которое не является стартовым и при попытке запустить приложение используя его мы получим ошибку, поэтому необходимо воспользоваться «Activity Launcher» в котором найдем название необходимого нам активити «.HomeActivity».

C#:

```
string adbPath = @"C:\Program Files\Nox\bin\adb.exe";
string device = "127.0.0.1:62035";
string appName = ;
string appActivity = ".HomeActivity";
string appiumPort = "4723";
AndroidDriver<IWebElement> driver = Initialization(device, appName, appActivity, appiumPort,
true);
```

Далее просмотрим параметры необходимых компонентов.

Теперь реализуем алгоритм в коде.
C#:

```
//шаг 1
ElementTap(driver, "android.widget.ImageView", "resource-id", "menu_item_view");
//шаг 2
ElementTextSend(driver, "android.widget.EditText", "resource-id", "search_edit_text",
"5qap5aO4i9A");
//шаг 3
AdbCommand(adbPath, device, @"shell input keyevent 66");
//шаг 4
ElementTap(driver, "android.widget.ImageView", "index", "0");
Console.WriteLine("Нажмите любую клавишу для остановки просмотра видео");
Console.ReadKey();
driver.CloseApp();
```

Таким образом нами был реализован способ автоматического просмотра видео в «YouTube», если добавить авторизацию то появиться возможность писать в чат стримов или оставлять комментарии, что может быть использовано для рассылок, рекламы своего контента и т.п.

Заключение

Благодаря автоматизации приложений на «Android» вы можете эмулировать любые действия в любом клиентском приложении, легко просматриваемая структура интерфейса позволяет в кратчайшие строки реализовать необходимый алгоритм. В рамках данной статьи не были упомянуты возможности использования нескольких эмуляторов, перехват и анализ сетевого траффика, использование прокси серверов, изменения идентификаторов эмулятора и многое другое что касается темы автоматизации «Android», если материал вызовет интерес и будет положительно оценен на форуме то в дальнейшем постараюсь раскрыть вышеупомянутые темы в следующих статьях.