

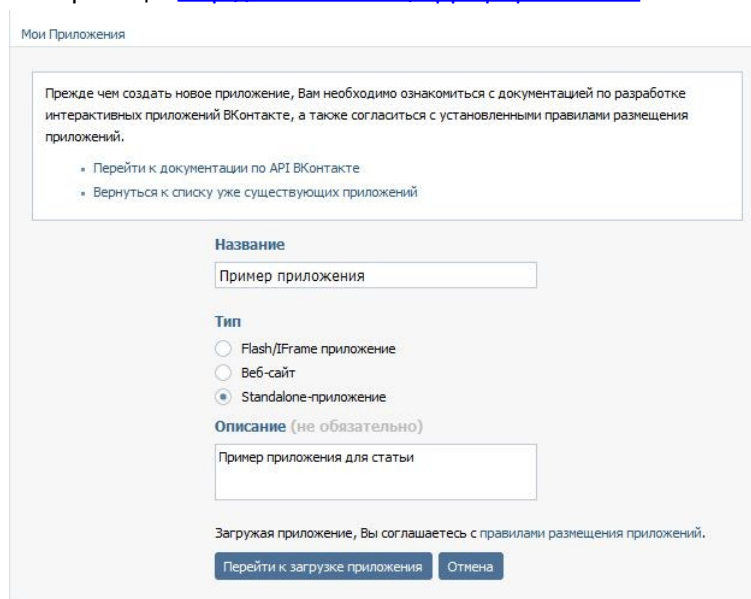
Разработка desktop-приложения для «ВКонтакте» на C#

Социальная сеть «ВКонтакте» является одним из популярных проектов, который ежедневно посещают миллионы пользователей. Помимо работы с сайтом непосредственно через интернет-браузер, Вконтакте позволяет создавать и подключать к себе desktop-приложения. В этой статье будет рассмотрена реализация простого desktop-приложения на C#.

Введение

Прежде чем начинать разработку приложения, его необходимо добавить на сайт «ВКонтакте».

Сделать это можно на странице: <http://vkontakte.ru/apps.php?act=add>



Мои Приложения

Прежде чем создать новое приложение, Вам необходимо ознакомиться с документацией по разработке интерактивных приложений ВКонтакте, а также согласиться с установленными правилами размещения приложений.

- Перейти к документации по API ВКонтакте
- Вернуться к списку уже существующих приложений

Название

Пример приложения

Тип

Flash/IFrame приложение

Веб-сайт

Standalone-приложение

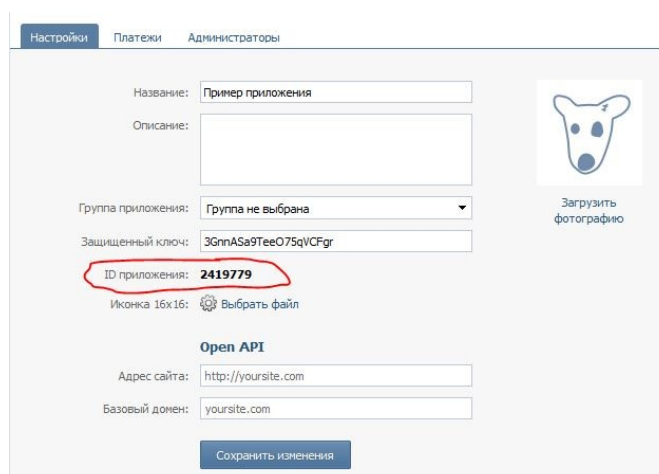
Описание (не обязательно)

Пример приложения для статьи

Загружая приложение, Вы соглашаетесь с правилами размещения приложений.

[Перейти к загрузке приложения](#) [Отмена](#)

После добавления приложения, вам будет выдан его числовой идентификатор, при помощи которого и будет осуществляться вся дальнейшая работа с приложением.



Настройки | Платежи | Администраторы


Название: Пример приложения

Описание:

Группа приложения: Группа не выбрана

Защищенный ключ: 3GnnASa9TeeO75qVCFgr

ID приложения: **2419779**


Иконка 16x16:  Выбрать файл

Open API

Адрес сайта: http://yoursite.com

Базовый домен: yoursite.com

[Сохранить изменения](#)

 Загрузить фотографию

Далее нужно ознакомиться с возможностями API и главным образом – с вариантами авторизации пользователей. Сделать это можно на странице: http://vkontakte.ru/developers.php?id=1_37230422&s=1

ВКонтакте предлагает использовать браузер для осуществления авторизации пользователя, в нашем случае речь идет о компоненте WebBrowser. Это сделано для того, чтобы приложение не

знало, какой логин и пароль вводит пользователь. Собственно можно обойтись и без WebBrowser, самостоятельно запрашивая логин и пароль у пользователя, если конечно пренебречь седьмым пунктом правил для приложений. Как именно это сделать также будет показано в этой статье, однако вероятней всего, такое приложение не пройдет проверку администрацией ВКонтакте.

Внимание. Правила размещения приложений были изменены. Прежде чем получить доступ к настройкам Ваших приложений, Вы должны ознакомиться и согласиться с новыми правилами.

Правила размещения приложений

При размещении приложений на сайте **ВКонтакте** запрещается:

1. Использовать название **ВКонтакте** в названии или описании приложения и логотип **ВКонтакте** в оформлении приложения.
2. Ограничивать частичную или полную функциональность приложения требованием от пользователя совершения факультативных действий: рассылки приглашений другим пользователям ВКонтакте, размещения записей на стенах, добавления приложения в меню быстрого доступа или согласия на получение от приложения оповещений.
3. Рассылать пользователям уведомления, которые не касаются данного приложения или содержат спам.
4. Принимать оплату услуг в приложении отличными от внутренней валюты **ВКонтакте** способами, в том числе на сторонних сайтах.
5. Загружать приложения и разработки третьих лиц, нарушая их авторские права.
6. Размещать в приложении ссылки, которые могут ввести пользователя в заблуждение относительно результатов перехода по ним.
7. Запрашивать у пользователей персональные данные: email, пароли, номера телефонов, паспортные данные и другую личную информацию.
8. Использовать или рекомендовать использовать автоматизированные скрипты для приглашения друзей пользователя в приложение.
9. Коренным образом менять суть и название приложения, загружая новое приложение на место старого.
10. Вводить пользователей в заблуждение относительно функциональности приложения.
11. Размещать приложения-заглушки, не несущие функциональной нагрузки.
12. Предоставлять пользователям возможность прямого скачивания музыкальных или видео файлов с серверов **ВКонтакте**.
13. Напрямую обращаться к скриптам **ВКонтакте**, за исключением обращений к API.
14. Размещать приложения, полностью копирующие функциональность других приложений **ВКонтакте**.
15. Открывать всплывающие окна без активных действий со стороны пользователя.
16. Поощрять пользователей за приглашения в приложение, включая публикацию записей на стенах других пользователей.
17. Создавать виджеты для официальных страниц, нарушающие Правила оформления виджетов (последняя редакция от 19.05.2011).

Размещая приложение ВКонтакте, разработчик принимает [Правила размещения рекламы в приложениях](#).

Настоящие Правила размещения приложений являются неотъемлемой частью [Условий размещения приложений на сайте ВКонтакте.ру](#).

Я согласен с новыми правилами

После прохождения пользователем процедуры авторизации, дальнейшую работу с API можно осуществлять посредством специального ключа (access_token) с ограниченным сроком жизни. Запросы нужно будет отправлять на адрес типа:

https://api.vkontakte.ru/method/METHOD_NAME?PARAMETERS&access_token=ACCESS_TOKEN

ВКонтакте умеет возвращать результаты выполнения запросов как в JSON-формате, так и в XML. В нашем случае удобней будет запрашивать XML-данные. Для этого нужно будет к имени метода (METHOD_NAME) добавить расширение xml (METHOD_NAME.xml).

В теории все выглядит достаточно просто, перейдем к практической реализации.

Авторизация

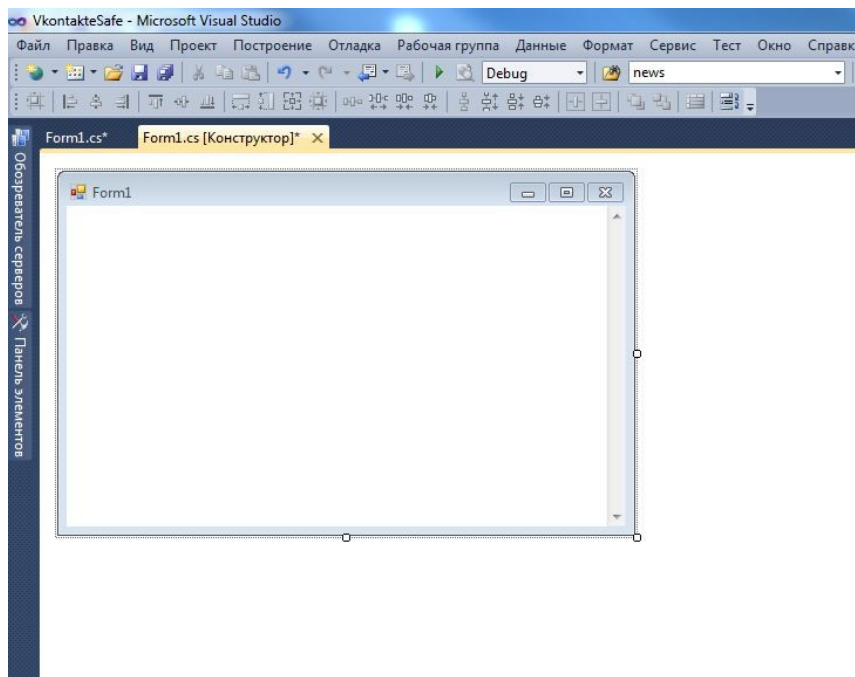
Безопасная авторизация

Как уже было сказано ранее, для авторизации пользователей в desktop-приложениях, ВКонтакте предлагает использовать компонент типа WebBrowser. И не просто предлагает, а категорически

запрещает использовать иные методы авторизации. Как говорится, хозяин – барин, и следуя правилам сделаем авторизацию через WebBrowser.

Для начала создадим новое Windows Forms приложение. В этой статье я буду использовать .NET Framework 4.0 и язык программирования C#, однако вы смело можете использовать и ранние версии .NET Framework, код должен получиться совместимым.

Разместим на форме компонент WebBrowser.



Затем, в событии загрузки формы (Form_Load) откроем в WebBrowser страницу авторизации пользователя и запросим необходимые права для нашего приложения. В адресе страницы необходимо передать числовой идентификатор нашего приложения (**client_id**), требуемые разрешения (**scope**) и указать тип запроса (**display**).

Идентификатор приложения можно узнать в настройках приложения на сайте ВКонтакте, в моем случае - **2419779**. Идентификатор приложения лучше всего хранить в отдельной переменной, чтобы можно было легко его поменять. Для этого можно сделать, например, переменную **appId**.

```
private int appId = 2419779;
```

Список возможных прав можно найти на этой странице:

<http://vkontakte.ru/developers.php?o=-1&p=%CF%F0%E0%E2%E0%20%EF%F0%E8%EB%EE%E6%E5%ED%E8%E9>

Для удобства мы будем использовать числовые значения (ВКонтакте также позволяет использовать слова) и сделаем перечисление списка прав.

```
private enum VkontakteScopeList
{
    /// <summary>
    /// Пользователь разрешил отправлять ему уведомления.
    /// </summary>
    notify = 1,
    /// <summary>
    /// Доступ к друзьям.
    /// </summary>
    friends = 2,
    /// <summary>
```

```

    /// Доступ к фотографиям.
    /// </summary>
    photos = 4,
    /// <summary>
    /// Доступ к аудиозаписям.
    /// </summary>
    audio = 8,
    /// <summary>
    /// Доступ к видеозаписям.
    /// </summary>
    video = 16,
    /// <summary>
    /// Доступ к предложениям (устаревшие методы).
    /// </summary>
    offers = 32,
    /// <summary>
    /// Доступ к вопросам (устаревшие методы).
    /// </summary>
    questions = 64,
    /// <summary>
    /// Доступ к wiki-страницам.
    /// </summary>
    pages = 128,
    /// <summary>
    /// Добавление ссылки на приложение в меню слева.
    /// </summary>
    link = 256,
    /// <summary>
    /// Доступ заметкам пользователя.
    /// </summary>
    notes = 2048,
    /// <summary>
    /// (для Standalone-приложений) Доступ к расширенным методам работы с сообщениями.
    /// </summary>
    messages = 4096,
    /// <summary>
    /// Доступ к обычным и расширенным методам работы со стеной.
    /// </summary>
    wall = 8192,
    /// <summary>
    /// Доступ к документам пользователя.
    /// </summary>
    docs = 131072
}

```

Необходимые разрешения можно сразу вынести в отдельную переменную. Пускай это будет переменная **scope**.

```

private int scope = (int)(VkontakteScopeList.audio | VkontakteScopeList.docs |
VkontakteScopeList.friends | VkontakteScopeList.link | VkontakteScopeList.messages |
VkontakteScopeList.notes | VkontakteScopeList.notify | VkontakteScopeList.offers |
VkontakteScopeList.pages | VkontakteScopeList.photos | VkontakteScopeList.questions |
VkontakteScopeList.video | VkontakteScopeList.wall);

```

Конечный код перехода на страницу авторизации ВКонтакте будет выглядеть следующим образом:

```

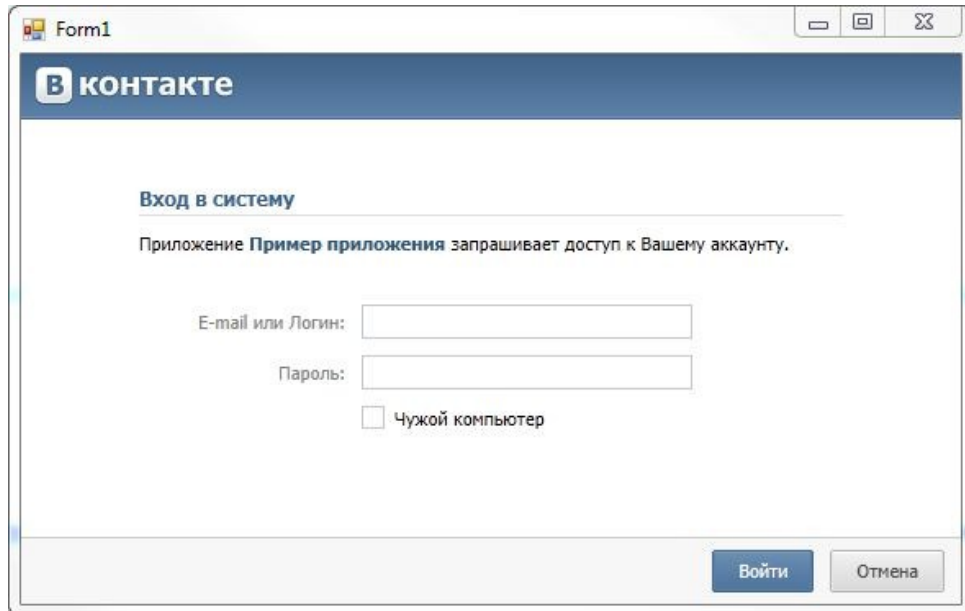
private void frmSignin_Load(object sender, EventArgs e)
{
    webBrowser1.Navigate(String.Format("http://api.vkontakte.ru/oauth/authorize?
client_id={0}&scope={1}&display=popup&response_type=token", appId, scope));
}

```

Чтобы посмотреть, что получилось, запустите приложение в режиме отладки (F5).

Важно. Проверьте, чтобы файрвол и/или антивирус не блокировали приложению доступ в интернет.

Если вы не авторизованы на сайте ВКонтакте, то в WebBrowser загрузится форма авторизации. В противном случае, в WebBrowser загрузится страничка с запросом прав для приложения.

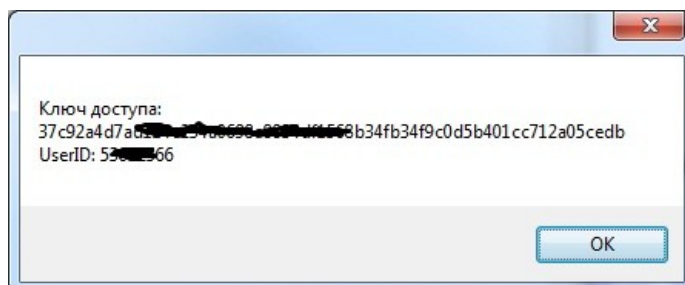


Процедура авторизации и запрос прав нам необходима для получения ключа доступа (`access_token`), при помощи которого будет осуществляться дальнейшая работа с API. ВКонтакте выдает ключ на последнем шаге, когда пользователь пройдет все необходимые проверки в WebBrowser. Отследить, что происходит в WebBrowser, можно путем обработки события **DocumentCompleted**. Проще всего отслеживать появление слова `access_token` в загруженном в WebBrowser адресе, а затем распарсить url и выдернуть из него идентификатор пользователя и ключ доступа.

```
private void webBrowser1_DocumentCompleted(object sender,
WebBrowserDocumentCompletedEventArgs e)
{
    if (e.Url.ToString().IndexOf("access_token") != -1)
    {
        string accessToken = "";
        int userId = 0;
        Regex myReg = new Regex(@"(?<name>[\w\d\x5f]+)=(?<value>[^\x26\s]+)",
RegexOptions.IgnoreCase | RegexOptions.Singleline);
        foreach (Match m in myReg.Matches(e.Url.ToString()))
        {
            if (m.Groups["name"].Value == "access_token")
            {
                accessToken = m.Groups["value"].Value;
            }
            else if (m.Groups["name"].Value == "user_id")
            {
                userId = Convert.ToInt32(m.Groups["value"].Value);
            }
        }
        // еще можно запомнить срок жизни access_token - expires_in,
        // если нужно
    }
}
```

```
    }  
    MessageBox.Show(String.Format("Ключ доступа: {0}\nUserID: {1}", accessToken,  
userId));  
    }  
}
```

Если теперь запустить программу, то после прохождения процедуры авторизации и запроса прав, появится окошко с ключом доступа и идентификатором пользователя.



Авторизация для реальных пацанов

Всё, что делается пользователем через браузер - можно сделать программно. В этой части статьи мы пройдем процедуру авторизации и получение ключа доступа (access_token) без использования WebBrowser. Но я еще раз хочу обратить ваше внимание на то, что это запрещено правилами ВКонтакте и приложение, использующее подобный метод авторизации, скорей всего не пройдет проверку и будет запрещено администрацией ВКонтакте.

Прежде чем что-то начинать делать, нужно разобраться, как это вообще работает. Для этого придется анализировать трафик, который проходит при авторизации и добавлении разрешений пользователем в браузере. Посмотреть трафик можно при помощи локального прокси-сервера - **Fiddler2**. <http://kbyte.ru/ru/Programming/Tools.aspx?id=15&mode=show>

Но опять же, нестись сломя голову изучать трафик не стоит, ибо API ВКонтакте позволяет работать с различными типами устройств и наверняка можно получить максимально простой и понятный трафик от него. Страница авторизации принимает параметр **display**, который позволяет указать тип окна авторизации в зависимости от устройства, на котором работает приложение. **Display** может иметь следующие значения: page, popup, touch и wap. В случае с безопасной авторизацией мы использовали **popup**, что вполне оптимально. При программной имитации действия пользователя, разумней будет использовать, например **wap**, т.к. очевидно, что выдаваемый html-код будет оптимизирован под мобильные телефоны, т.е. довольно простым.

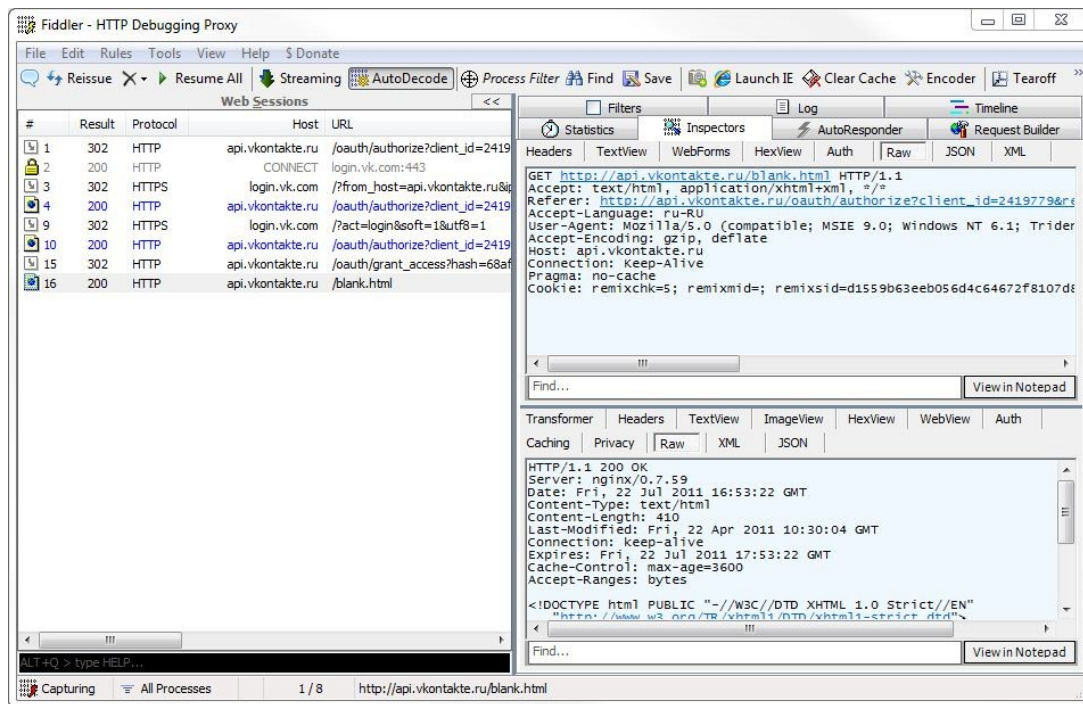
Итак, запустим **Fiddler2** и браузер – **Internet Explorer** (Fiddler к нему цепляется автоматически, без каких-либо дополнительных настроек).

Примечание. Если вы авторизованны на сайте ВКонтакте, то выйдите, чтобы иметь возможность отследить весь цикл взаимодействия браузера с сайтом.

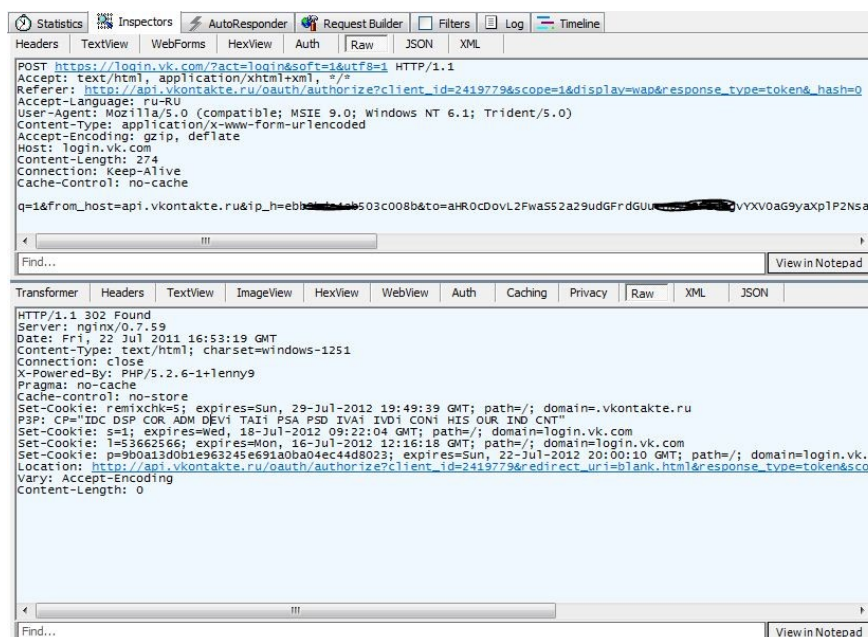
В адресную строку Internet Explorer вставим ссылку запроса прав для приложения:
[http://api.vkontakte.ru/oauth/authorize?
client_id=2419779&scope=1&display=wap&response_type=token](http://api.vkontakte.ru/oauth/authorize?client_id=2419779&scope=1&display=wap&response_type=token)

Примечание. В параметр `client_id` укажите идентификатор вашего приложения. Остальные параметры оставьте без изменений.

Пройдем весь цикл, от авторизации до установки разрешений, и посмотрим, что у нас будет в отчёте **Fiddler2**.



Первый запрос идет к странице, адрес которой мы вставили в адресную строку браузера. На ней мы указали наш логин и пароль. Далее, на скрине выше, под номерами: 2, 3 и 4 идет мусор, на который можно не обращать внимание. После нажатия на кнопку «**Войти**» данные с формы авторизации отправляются на страницу: <https://login.vk.com/?act=login&soft=1&utf8=1>



После чего сервер выдает 302-ой HTTP-код – перенаправление на страницу запроса разрешений у пользователя для нашего приложения, а также куки авторизации.

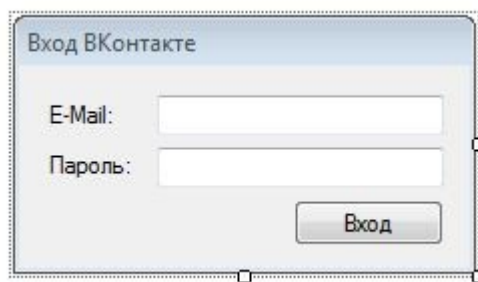
После нажатия пользователем на кнопку «**Разрешить**», форма отправляется на страницу типа: http://api.vkontakte.ru/oauth/grant_access?hash=68af114762009eaf68&client_id=2419779&settings=1&redirect_uri=blank.html&response_type=token&state=

Где опять происходит перенаправление с 302-ым HTTP-кодом на страницу содержащую ключ доступа, идентификатор пользователя и срок годности ключа.

Ну что ж, дело за малым - повторить все эти действия программно. Вместо `WebBrowser` мы будем использовать классы `HttpRequest` и `HttpResponse`. По аналогии с безопасной авторизацией, создадим две локальные переменные – `appId` и `scope`, а также перечисление `VkontakteScopeList` (см. предыдущую часть статьи).

Примечание. Переменные и перечисление лучше всего сделать один раз в `static` классе, чтобы они были доступны в рамках всего приложения.

Также создадим форму с двумя текстовыми полями и одной кнопкой. Имя первого текстового поля будет `tbLogin`, а второго – `tbPassword`. Кнопку назовем `btnSignin`, весь последующий код будет писаться в обработчике нажатия кнопки (`btnSignin_Click`).



Как и в случае с браузером, первым шагом будет обращение к странице авторизации и запроса прав для приложения.

```
HttpRequest myReq =  
(HttpRequest)HttpRequest.Create(String.Format("http://api.vkontakte.ru/oauth/  
authorize?client_id={0}&scope={1}&display=wap&response_type=token", appId, scope));  
HttpResponse myResp = (HttpResponse)myReq.GetResponse();  
StreamReader myStream = new StreamReader(myResp.GetResponseStream(), Encoding.UTF8);  
string html = myStream.ReadToEnd();
```

Примечание. Для использования классов `HttpRequest` и `HttpResponse` необходимо импортировать (при помощи директивы `using`) пространство имен `System.Net`. Для использования класса `StreamReader`, требуется пространство имен `System.IO`. Класс `Encoding` находится в пространстве имен `System.Text`, которое также необходимо импортировать.

Содержимое страницы будет помещено в переменную `html`. Нам нужно найти на этой странице форму авторизации и передать в следующем запросе все её элементы, указав логин и пароль пользователя. Проще всего сначала найти форму.

```
Regex myReg = new Regex("<form(.*)>(?(form_body).*)</form>", RegexOptions.IgnoreCase |  
RegexOptions.Multiline | RegexOptions.Singleline);
```



```

if (!myReg.IsMatch(html) || (html = myReg.Match(html).Groups["form_body"].Value) == "")
{
    MessageBox.Show("Не удалось получить форму авторизации. Проверьте шаблон регулярного
выражения.");
    return;
}

```

Примечание. Для использования класса **Regex** и перечисления **RegexOptions** необходимо импортировать пространство имен **System.Text.RegularExpressions**.

Поиск формы происходит с использованием регулярных выражений. В случае успеха, содержимое формы будет помещено в переменную `html`. Затем нам нужно выдернуть из формы все элементы и сформировать из них параметры для следующего запроса, подставив логин и пароль пользователя.

```

myReg = new Regex("<input(.*)name=\"(?<name>[^\x22]+)\"(.*)((value=\"(?<value>[^\x22]*)\"(.*))|(.?))>", RegexOptions.IgnoreCase | RegexOptions.Multiline |
RegexOptions.Singleline);
NameValuePairCollection qs = new NameValueCollection();
foreach (Match m in myReg.Matches(html))
{
    string val = m.Groups["value"].Value;
    if (m.Groups["name"].Value == "email")
    {
        val = tbLogin.Text;
    }
    else if (m.Groups["name"].Value == "pass")
    {
        val = tbPassword.Text;
    }
    qs.Add(m.Groups["name"].Value, HttpUtility.UrlEncode(val));
}

```

Примечание. Здесь для кодирования параметров url используется класс `HttpUtility`, который находится в сборке **System.Web.dll**. Вам необходимо подключить эту сборку к проекту <http://yolper.ru/49>. Если вы используете Visual Studio 2010 и клиентский профиль .NET Framework 4, вам может потребоваться переключить в обычный профиль .NET Framework <http://yolper.ru/50>, иначе сборки `System.Web.dll` может не быть в списке доступных сборок. После подключения сборки, необходимо импортировать пространство имен **System.Web**. Класс **NameValueCollection** принадлежит пространству имен **System.Collections.Specialized**, которое также необходимо импортировать.

Все найденные элементы формы перебираются циклом и помещаются в коллекцию `qs` без изменений. Вместо логина (**email**) и пароля (**pass**) подставляются данные текущего пользователя из текстовых полей – **tbLogin** и **tbPassword**.

Затем данные нужно отправить на страницу <https://login.vk.com/?act=login&soft=1&utf8=1> методом **POST**. Здесь важно правильно указать **ContentType**, и отключить автоматическое перенаправление (**AllowAutoRedirect**). Также в запросе нужно создать **CookieContainer**, чтобы иметь возможность работать с куками (Cookie).

```

byte[] b = System.Text.Encoding.UTF8.GetBytes(String.Join("&", from item in qs.AllKeys
select item + "=" + qs[item]));

```

```

myReq = (HttpWebRequest)HttpWebRequest.Create("https://login.vk.com/?
act=login&soft=1&utf8=1");
myReq.CookieContainer = new CookieContainer();
myReq.Method = "POST";

```

```

myReq.ContentType = "application/x-www-form-urlencoded";
myReq.ContentLength = b.Length;
myReq.GetRequestStream().Write(b, 0, b.Length);
myReq.AllowAutoRedirect = false;

myResp = (HttpWebResponse)myReq.GetResponse();

```

Примечание. Для корректной работы конструкции **from ... in ... select** необходимо импортировать пространство имен **System.Linq**.

Никакого html-содержимого в ответе сервера не будет. При успешном прохождении авторизации, сервер вернет куки (Cookie). Куки мы запишем в отдельную переменную, для последующего использования.

```

CookieContainer cc = new CookieContainer();
foreach (Cookie c in myResp.Cookies)
{
    cc.Add(c);
}

```

Также сервер вернет информацию о перенаправлении на страницу запроса у пользователя разрешений для нашего приложения. Адрес страницы находится в HTTP-заголовке **Location**. Мы сделаем перенаправление вручную методом **GET**, при этом важно не забыть передать полученные куки (Cookie), иначе сервер опять возвратит страницу авторизации.

```

if (!String.IsNullOrEmpty(myResp.Headers["Location"]))
{
    // делаем редирект
    myReq = (HttpRequest)HttpRequest.Create(myResp.Headers["Location"]);
    myReq.CookieContainer = cc; // передаем куки
    myReq.Method = "GET";
    myReq.ContentType = "text/html";

    myResp = (HttpWebResponse)myReq.GetResponse();
    myStream = new StreamReader(myResp.GetResponseStream(), Encoding.UTF8);
    html = myStream.ReadToEnd();
}
else
{
    // что-то пошло не так
    MessageBox.Show("Ошибка. Ожидался редирект.");
    return;
}

```

В переменной html должно быть содержимое страницы запроса разрешений для приложения. Чтобы проверить так это или нет, мы попытаемся найти в теле страницы имя и ссылку пользователя. Делать мы это будем, как и положено реальным пацанам, при помощи регулярных выражений. Если имя пользователя не будет найдено, значит процедура авторизации прошла неудачно.

```

myReg = new Regex("Вы авторизованы как <b><a href=\"(?<ur1>[^\x22]+)\">{<user>[^\x3c]+}</a></b>", RegexOptions.IgnoreCase | RegexOptions.Multiline | RegexOptions.Singleline);
if (!myReg.IsMatch(html))
{
    MessageBox.Show("Ошибка: Авторизация не прошла.");
    return;
}

```

```
MessageBox.Show(String.Format("Авторизация успешно прошла.\nПользователь {0}",  
myReg.Match(html).Groups["user"].Value));
```

Затем нужно получить форму, чтобы в последующем её отправить. Но в отличие от первой формы (формы авторизации), в форме запроса разрешений для приложения нас будет интересовать в первую очередь адрес, на который она будет отправлена. Собственно больше ничего в этой форме нам и не нужно, однако если в ней в будущем появятся дополнительные важные элементы, их придется парсить и передавать в запросе.

```
myReg = new Regex("<form(?:.*)action=\"(?:<post_url>[^\x22]+)\"(?:.*)>(?:<form_body>.*?)</form>", RegexOptions.IgnoreCase | RegexOptions.Multiline | RegexOptions.Singleline);  
if (!myReg.IsMatch(html))  
{  
    MessageBox.Show("Не удалось получить форму. Проверьте шаблон регулярного выражения.");  
    return;  
}
```

Полученный адрес может быть относительным, поэтому его нужно немного дописать.

```
string url = myReg.Match(html).Groups["post_url"].Value;  
if (!url.ToLower().StartsWith("http://")) { url =  
String.Format("http://api.vkontakte.ru{0}", url); }
```

Вот и все, делаем последний запрос.

```
myReq = (HttpRequest)HttpRequest.Create(url);  
myReq.CookieContainer = cc; // не забываем передавать куки  
myReq.Method = "POST";  
myReq.ContentType = "application/x-www-form-urlencoded";  
myReq.AllowAutoRedirect = false;
```

```
myResp = (HttpResponse)myReq.GetResponse();
```

И выдергиваем ключ доступа из адреса, на который перенаправляет нас сервер.

```
if (!String.IsNullOrEmpty(myResp.Headers["Location"]))  
{  
    myReg = new Regex(@"(?:<name>[\w\d\x5f]+)=(?:<value>[^\x26\s]+)", RegexOptions.IgnoreCase  
| RegexOptions.Singleline);  
    foreach (Match m in myReg.Matches(myResp.Headers["Location"]))  
    {  
        if (m.Groups["name"].Value == "access_token")  
        {  
            accessToken = m.Groups["value"].Value;  
        }  
        else if (m.Groups["name"].Value == "user_id")  
        {  
            userId = m.Groups["value"].Value;  
        }  
        // еще можно запомнить срок жизни access_token - expires_in,  
        // если нужно  
    }  
    MessageBox.Show(String.Format("Ключ доступа: {0}\nUserID: {1}", accessToken, userId));  
}  
else  
{  
    MessageBox.Show("Ошибка. Ожидался редирект.");  
}
```

В итоге получилось то же самое, что и при «безопасной авторизации», только без участия пользователя, а может быть даже и без его ведома.

Работа с API

Имея на руках ключ доступа, можно выполнять любые запросы к API (конечно, при условии, что от пользователя были получены необходимые права для приложения). Поскольку вся работа с API, по сути, ограничивается обращением к странице типа:

https://api.vkontakte.ru/method/METHOD_NAME.xml?PARAMETERS&access_token=ACCESS_TOKEN

то разумней всего сделать одну функцию, которая будет выполнять запросы к API и возвращать результат в XML. Эта функция должна принимать имя API-метода, а также дополнительные параметры, в зависимости от запроса. Так как в разных API-методах может быть разное количество параметров, для их передачи можно использовать коллекцию типа **NameValueCollection**.

```
private XmlDocument ExecuteCommand(string name, NameValueCollection qs)
{
    XmlDocument result = new XmlDocument();
    result.Load(String.Format("https://api.vkontakte.ru/method/{0}.xml?
access_token={1}&{2}", name, accessToken, String.Join("&", from item in qs.AllKeys select
item + "=" + qs[item])));
    return result;
}
```

Для примера, можно запросить у ВКонтакте, при помощи созданной функции, детальную информацию о пользователе.

```
NameValueCollection qs = new NameValueCollection();
qs["uid"] = userId.ToString();
MessageBox.Show(ExecuteCommand("getProfiles", qs).InnerXml);
```

Полный список поддерживаемых API-методов ВКонтакте находится здесь:

<http://vkontakte.ru/developers.php?o=-1&p=%CE%EF%E8%F1%E0%ED%E8%E5%20%EC%E5%F2%EE%E4%EE%E2%20API>

Реальный пример

Для удобства работы с API ВКонтакте можно сделать отдельный класс, например **VKAPI**. В перспективе, этот класс должен содержать все функции и методы, которые есть в API. Но мы этого делать не будем, по крайней мере, я точно не буду, а вот вы вполне можете совершить подвиг. Поскольку для выполнения запросов нам понадобится **access_token**, то его можно хранить в нашем классе и принимать в конструкторе.

```
public string AccessToken = "";

public VKAPI(string accessToken)
{
    this.AccessToken = accessToken;
}
```

Примечание. В этот класс также можно добавить информацию о времени жизни ключа доступа, запоминать время инициализации класса и проверять продолжительность его жизни с продолжительностью жизни ключа.

Вся работа с сервером ВКонтакте будет происходить в функции **ExecuteCommand**, которую мы написали в предыдущей части статьи, её в принципе можно оставить без изменений и поместить, как есть, в наш класс (**VKAPI**). Единственно, в приложениях типа **Windows Forms**, выполнение удаленных запросов будет подвешивать основной поток, в котором работает программа, поэтому было бы неплохо вынести эти запросы в отдельный поток.

Использование потоков

Если вы хотите забивать свою голову потоками, то пропустите эту часть статьи.

Чтобы иметь возможность отслеживать ход выполнения удаленного запроса, сделаем форму, назовём её **frmProgress**. Также эта форма будет выполнять функции посредника для передачи данных из одного потока в другой. На форму можно поместить какую-нибудь анимированную картинку. В код этой формы сделаем открытый метод, который будет помещать данные в свойство формы **Tag**. Это необходимо, поскольку форма будет находиться в основном потоке, а запрос выполняться – в другом, и прямой доступ к ней будет невозможен.

```
private delegate void DelegateSetTag(object t);
public void SetTag(object t)
{
    if (this.InvokeRequired)
    {
        this.Invoke(new DelegateSetTag(SetTag), t);
        return;
    }
    this.Tag = t;
    Application.DoEvents();
}
```

Примечание. Я использую свойство **Tag**, т.к. оно имеет тип **object** и в него можно поместить данные любого типа. При желании, в форме можно сделать для каждого типа данных отдельные, строго типизированные свойства.

Форму прогресса мы будем запускать, как диалог (**ShowDialog**). Это позволит приостановить выполнение программы в основном потоке, до тех пор, пока форма прогресса не будет закрыта. Чтобы закрыть форму из потока, нам понадобится добавить в неё еще один метод.

```
private delegate void DelegateCloseMe();
public void CloseMe()
{
    if (this.InvokeRequired)
    {
        this.Invoke(new DelegateCloseMe(CloseMe));
        return;
    }
    this.Close();
}
```


Теперь немного изменим код функции **ExecuteCommand**. Точнее мы сделаем из одной функции – две. Первая – **ExecuteCommand**, создаст экземпляр формы **frmProgress**, в свойство **Tag** этой формы мы передадим в виде **Hashtable** имя аргумента и дополнительные параметры.

```
frmProgress frm = new frmProgress();
frm.Owner = Program.applicationContext.MainForm;
```

```
Hashtable pars = new Hashtable();
pars.Add("name", name);
pars.Add("qs", qs);
frm.Tag = pars;
```

Затем мы сделаем поток и передадим ему созданный экземпляр формы **frmProgress**. Поток будет выполняться отдельно от основного, однако программа будет заблокирована, т.к. экземпляр формы прогресса будет запущен методом **ShowDialog**.

```
Thread t = new Thread(ExecuteCommandThread);
t.IsBackground = true;
t.Start(frm);
```

```
frm.ShowDialog();
```

В конце работы функции, когда экземпляр формы **frmProgress** будет закрыт, она (функция) вернет ответ, полученный от сервера ВКонтакте в виде **XmlDocument** из свойства **Tag** закрытой формы прогресса.

```
return (XmlDocument)frm.Tag;
```

Собственно второй функцией, а точнее методом, будет - **ExecuteCommandThread**, который и выполняться в отдельном потоке. Этот метод будет принимать созданный в функции **ExecuteCommand** экземпляр формы **frmProgress**, из свойства **Tag** получит имя аргумента и дополнительные параметры.

```
frmProgress frm = (frmProgress)args;
string name = ((Hashtable)frm.Tag)["name"].ToString();
NameValueCollection qs = (NameValueCollection)((Hashtable)frm.Tag)["qs"];
```

Затем, выполнит удаленный запрос к серверу ВКонтакте.

```
XmlDocument result = new XmlDocument();
result.Load(String.Format("https://api.vkontakte.ru/method/{0}.xml?access_token={1}&{2}",
name, this.AccessToken, String.Join("&", from item in qs.AllKeys select item + "=" +
qs[item])));
if (result.SelectSingleNode("error") != null)
{
    throw new Exception("Error. " + result.SelectSingleNode("error/error_msg"));
}
```

В случае успеха, передаст полученный ответ обратно в экземпляр формы **frmProgress** в свойство **Tag** и закроет форму.

```
frm.SetTag(result);
frm.CloseMe();
```

После закрытия формы выполнение функции **ExecuteCommand** будет продолжено и она вернет полученные XML-данные, как я уже писал выше.

Вызывать функцию **ExecuteCommand** можно также как и раньше, но если раньше приложение во время запросов подвисало, то теперь этого происходить не будет.

Описание API-функций в классе VKAPI и их использование

Как я уже говорил, в рамках этой статьи мы не будем описывать все поддерживаемые методы API ВКонтакте. Сделаем лишь некоторые из них. Например, сделаем функцию получения деталей профиля пользователя. Получение информации о пользователе делается api-методом **getProfiles**, который может принимать следующие параметры:

uids - перечисленные через запятую ID пользователей (максимум 1000 штук);

domains - перечисленные через запятую адреса пользователей (используется вместо uids);

fields - перечисленные через запятую поля анкет, необходимые для получения;

name_case - падеж для склонения имени и фамилии пользователя.

В нашем случае мы сделаем все по-простому, и функция будет принимать лишь идентификатор пользователя, данные которого нужно получить, но при этом сама функция будет запрашивать максимально возможную информацию о пользователе.

```
public XmlDocument GetProfile(int uid)
{
    NameValueCollection qs = new NameValueCollection();
    qs["uid"] = uid.ToString();
    qs["fields"] =
"uid,first_name,last_name,nickname,domain,sex,bdate,city,country,timezone,photo,has_mobil
e,rate,contacts,education,online";
    return ExecuteCommand("getProfiles", qs);
}
```

В случае успеха, функция **GetProfile** вернет Xml-данные пользователя, которые будут находиться в ветке **response/user**. Если сервер ВКонтакте вернет информацию об ошибке, то в программе произойдет исключение (см. функцию **ExecuteCommand**).

Но давайте попробуем использовать нашу функцию. Первым делом создадим экземпляр класса **VKAPI**, в которой нужно передать ключ доступа, полученный при авторизации пользователя.

```
VKAPI myVK = new VKAPI(accessToken);
```

Затем создадим переменную **profile** типа **XmlDocument**.

```
XmlDocument profile;
```

И, собственно, поместим в эту переменную профиль пользователя (в данном случае - мой), полученный при помощи функции - **GetProfile**.

```
profile = myVK.GetProfile(103639273);
```

Теперь, используя метод **SelectSingleNode** экземпляра **XmlDocument (profile)**, можно вывести информацию о пользователе в наше приложение.

```
MessageBox.Show(String.Format("Имя: {0}\r\nФамилия: {1}\r\nПол: {2}",
profile.SelectSingleNode("response/user/first_name").InnerText,
profile.SelectSingleNode("response/user/last_name").InnerText,
profile.SelectSingleNode("response/user/sex").InnerText));
```

Но нужно учитывать, что некоторых элементов профиля может не быть (пользователь мог указать неполную информацию о себе, либо эта информация защищена настройками приватности).

Поэтому необходимо проверять, есть требуемое поле в полученном xml-документе, или нет. Для этого лучше сделать отдельную хелпер-функцию.

```
public string GetDataFromXmlNode(XmlNode input)
{
    if (input == null || String.IsNullOrEmpty(input.InnerText))
    {
        return "нет данных";
    }
    else
    {
        return input.InnerText;
    }
}
```

Используя эту функцию, можно не боясь возникновения исключения, вывести информацию о пользователе.

```
MessageBox.Show(String.Format("Имя: {0}\r\nФамилия: {1}\r\nПол: {2}",
GetDataFromXmlNode(profile.SelectSingleNode("response/user/first_name")),
GetDataFromXmlNode(profile.SelectSingleNode("response/user/last_name")),
GetDataFromXmlNode(profile.SelectSingleNode("response/user/sex"))));
```

С текстовыми данными вроде бы все понятно, а вот что делать с бинарными, такими как, например, фотография пользователя? В xml-документе фотографии как таковой нет, может быть лишь ссылка на неё. Если ссылка на фотографию есть, мы можем загрузить её и поместить в **PictureBox**. Проще всего загрузить фотографию при помощи функции **DownloadData** класса **WebClient**.

```
if (profile.SelectSingleNode("response/user/photo") != null && !
String.IsNullOrEmpty(profile.SelectSingleNode("response/user/photo").InnerText))
{
    PictureBox picPhoto = new PictureBox();
    WebClient wc = new WebClient();
    byte[] b = wc.DownloadData(profile.SelectSingleNode("response/user/photo").InnerText);
    MemoryStream m = new MemoryStream(b);
    picPhoto.Image = Image.FromStream(m);

    picPhoto.Width = picPhoto.Height = 50;
    picPhoto.Visible = true;
    this.Controls.Add(picPhoto);
}
```

Далее, мы можем сделать в нашем классе **VKAPI** функции для получения названий стран и городов, т.к. в профиле возвращаются их числовые идентификаторы. Функции будут принимать код страны/города и возвращать название в виде строки. По сути, они одинаковые и отличаются лишь названием api-метода.

```
public string GetCity(int id)
```

```

{
    if (id <= 0) { return "нет данных"; }
    NameValueCollection qs = new NameValueCollection();
    qs["api_id"] = Program.appId.ToString();
    qs["cids"] = id.ToString();
    XmlDocument city = ExecuteCommand("getCities", qs);
    Return city.SelectSingleNode("response/city/name").InnerText;
}

public string GetCountry(int id)
{
    if (id <= 0) { return "нет данных"; }
    NameValueCollection qs = new NameValueCollection();
    qs["api_id"] = Program.appId.ToString();
    qs["cids"] = id.ToString();
    XmlDocument country = ExecuteCommand("getCountries", qs);
    Return country.SelectSingleNode("response/country/name").InnerText;
}

```

Примечание. Лучше всего создать свою локальную базу данных стран и городов ВКонтакте, чтобы не делать постоянно запросы к удаленному серверу.

Ну и напоследок, можно сделать функцию, которая будет помещать на стену пользователя текстовое сообщение.

```

public bool WallPost(int uid, string message)
{
    NameValueCollection qs = new NameValueCollection();
    qs["owner_id"] = uid.ToString();
    qs["message"] = message;
    ExecuteCommand("wall.post", qs);
    return true;
}

```

Послесловие

Социальная сеть «ВКонтакте» имеет довольно обширный набор api-методов, которые достаточно просто использовать. В этой статье я рассмотрел лишь малую часть из них. В примере, который вы можете скачать по ссылке ниже, реализация работы с API ВКонтакте немного отличается, от описанной в статье. Если у вас возникнут какие-либо вопросы, обращайтесь на форум.

Алексей Немиро
2011-07-24