Введение в XPath на примере простого парсера на C#

Вступление.

Уверен, что не раз и не два приходилось сливать какой-нибудь контент со страниц сайта. Естественно, первое, что приходит на ум - это впихнуть регулярные выражения где только можно и побольше. Люди не просто так придумали афоризм:

Некоторые люди, когда сталкиваются с проблемой, думают «Я знаю, я решу её с помощью регулярных выражений.» Теперь у них две проблемы.

Вот и мы не будет останавливаться на них, а пойдем дальше. У нас есть DOM. Почему бы не обрабатывать документ как подобает, а не просто как строку html-кода? Будем использовать библиотеку HtmlAgilityPack (http://htmlagilitypack.codeplex.com/). Официальная страничка HAP говорит нам следующее:

" This is an agile HTML parser that builds a read/write DOM and supports plain XPATH or XSLT (you actually don't HAVE to understand XPATH nor XSLT to use it, don't worry...). It is a .NET code library that allows you to parse "out of the web" HTML files. The parser is very tolerant with "real world" malformed HTML. The object model is very similar to what proposes System.Xml, but for HTML documents (or streams)."

В меру вольном переводе это звучит так:

Это парсер HTML, который строит доступный для чтения / записи DOM и поддерживает простой XPATH или XSLT (Вам не нужно понимать XPATH ни XSLT, чтобы использовать его, не волнуйтесь ...). Это. NET библиотека, которая позволяет работать с HTML файлами "вне сети". Анализатор терпим к "реальному" неправильному HTML. Объектная модель очень похожа на ту, что предлагает System.Xml, но для HTML документов (или потоков).

Хоть автор и говорит, что понимать XPath не обязательно, необходимо хотя бы знать, что это. Всезнающая Википедия дает такое определение (http://ru.wikipedia.org/wiki/XPath):

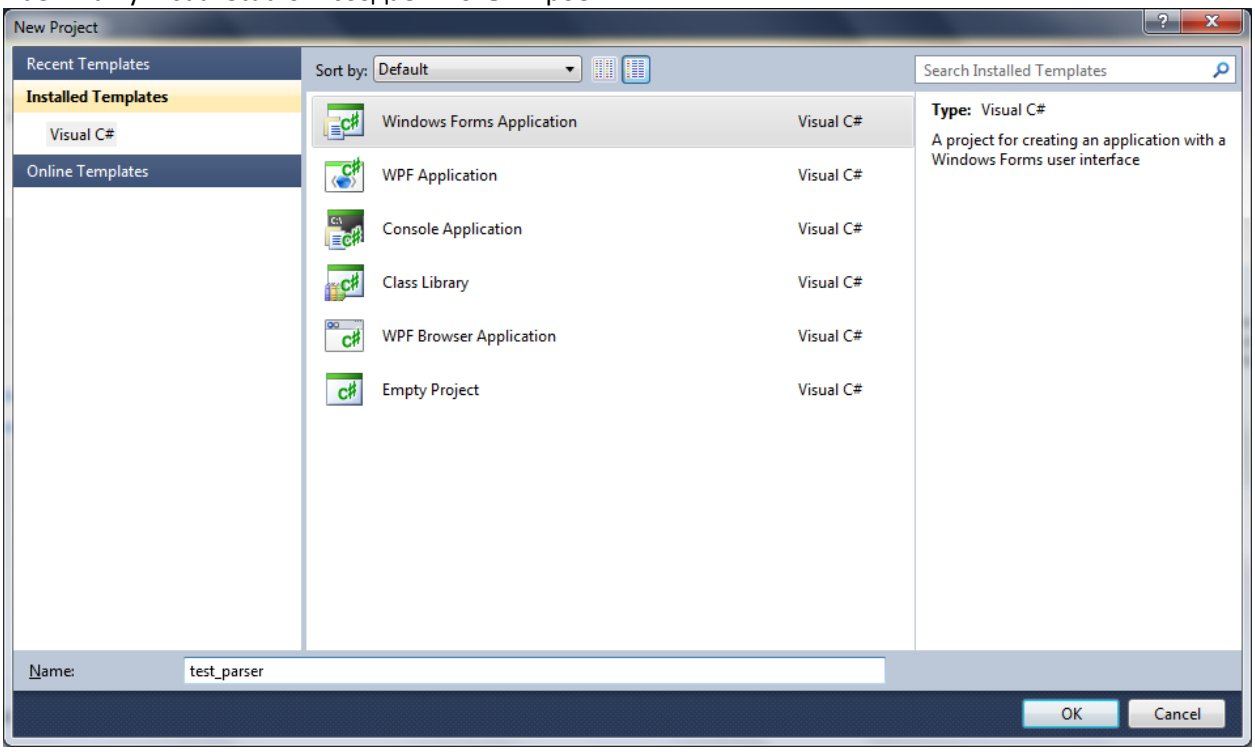
XPath (XML Path Language) — язык запросов к элементам XML-документа. Разработан для организации доступа к частям документа XML в файлах трансформации XSLT и является стандартом консорциума W3C. XPath призван реализовать навигацию по DOM в XML. В XPath используется компактный синтаксис, отличный от принятого в XML.

Начало работы

Выбираем сайт для экспериментов. Я взял сайт http://rabota-i-trud.com.ua/.

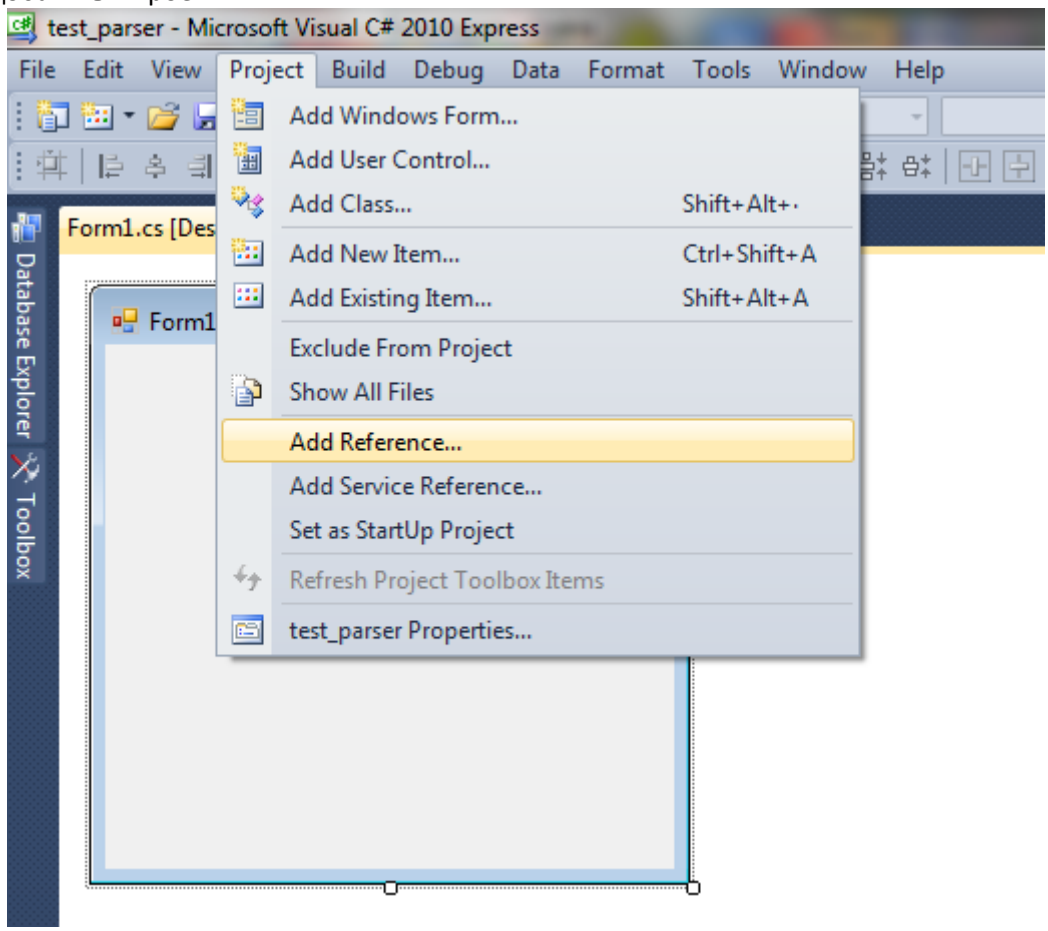
Качаем HAP с официального сайта.

Открываем нашу Visual Studio и создаем новый проект.

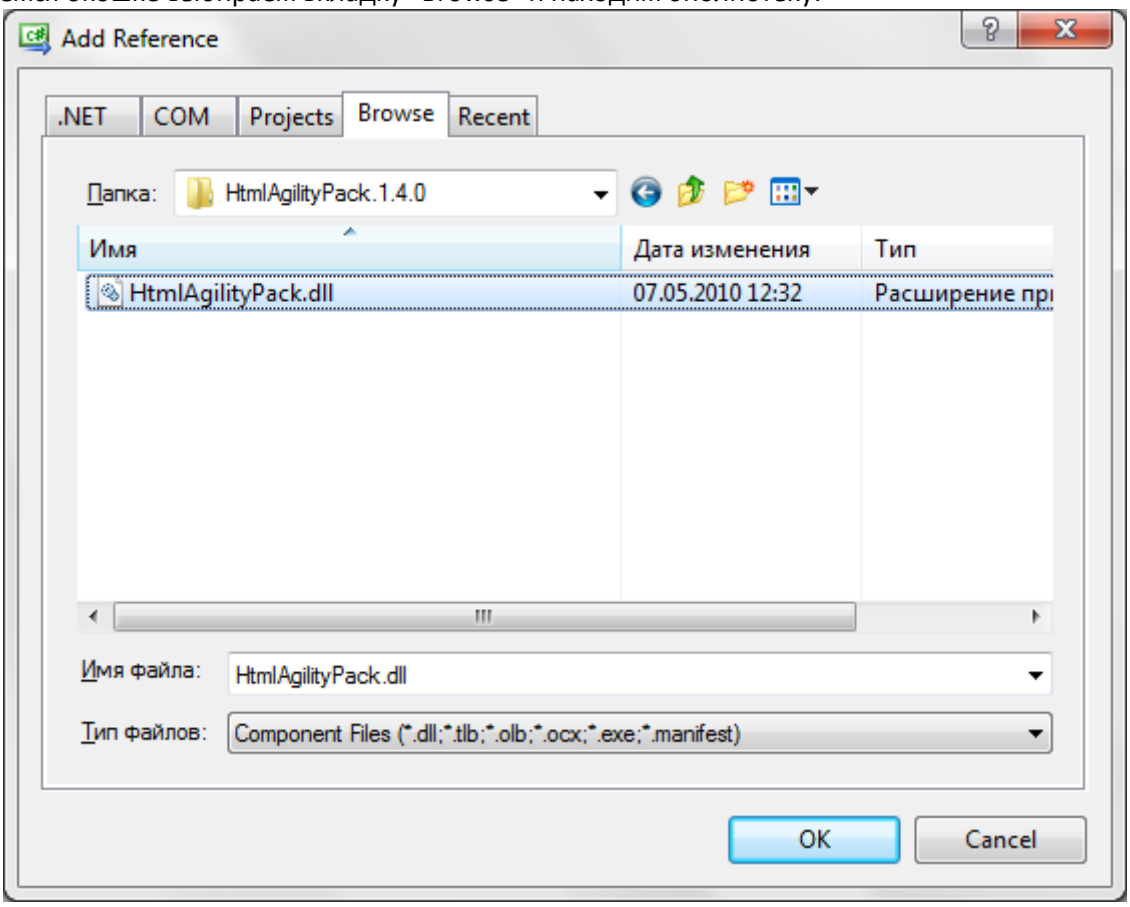


Здесь и далее разработка ведется на Visual Studio 2010 Express Edition под .NET 3.5

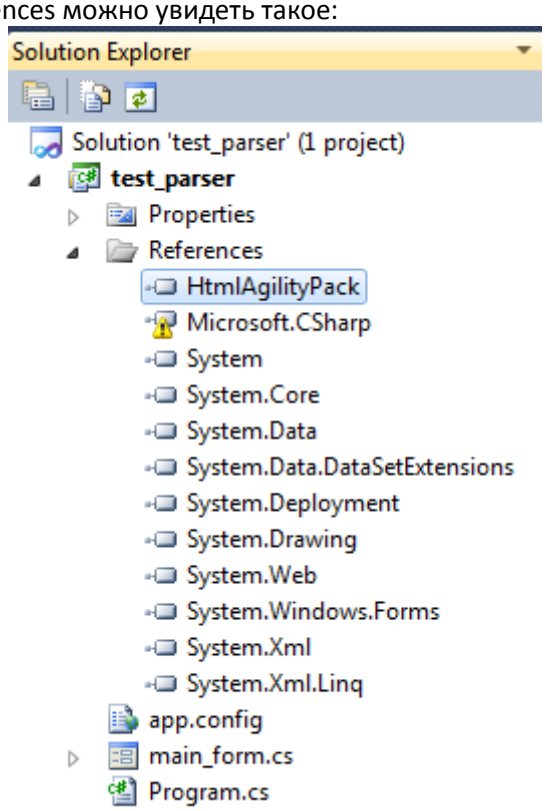
Теперь надо добавить в проект HAP.



В появившемся окошке выбираем вкладку "Browse" и находим библиотеку.



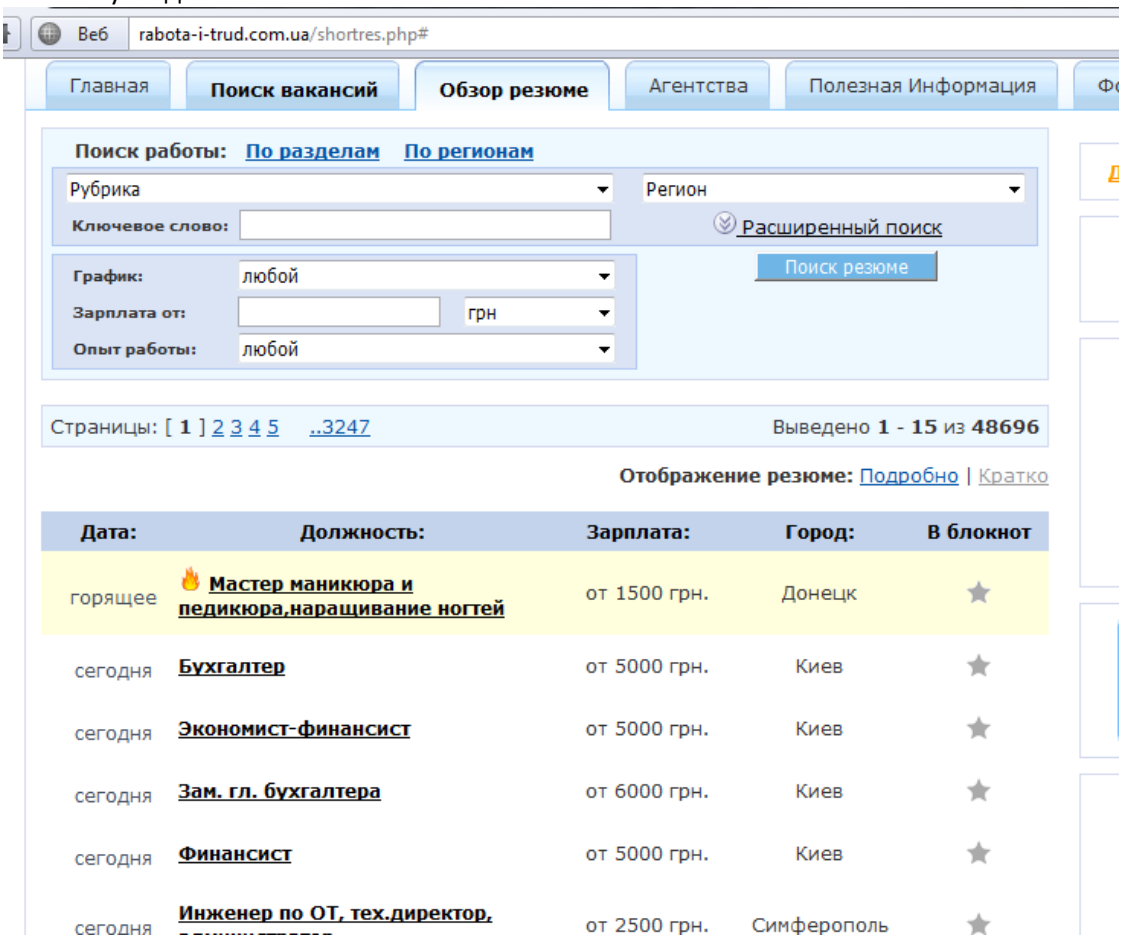
Теперь в Solution Explorer в References можно увидеть такое:



Осталось только прописать `using HtmlAgilityPack;` вначале и готово.

Исследуем целевой сайт

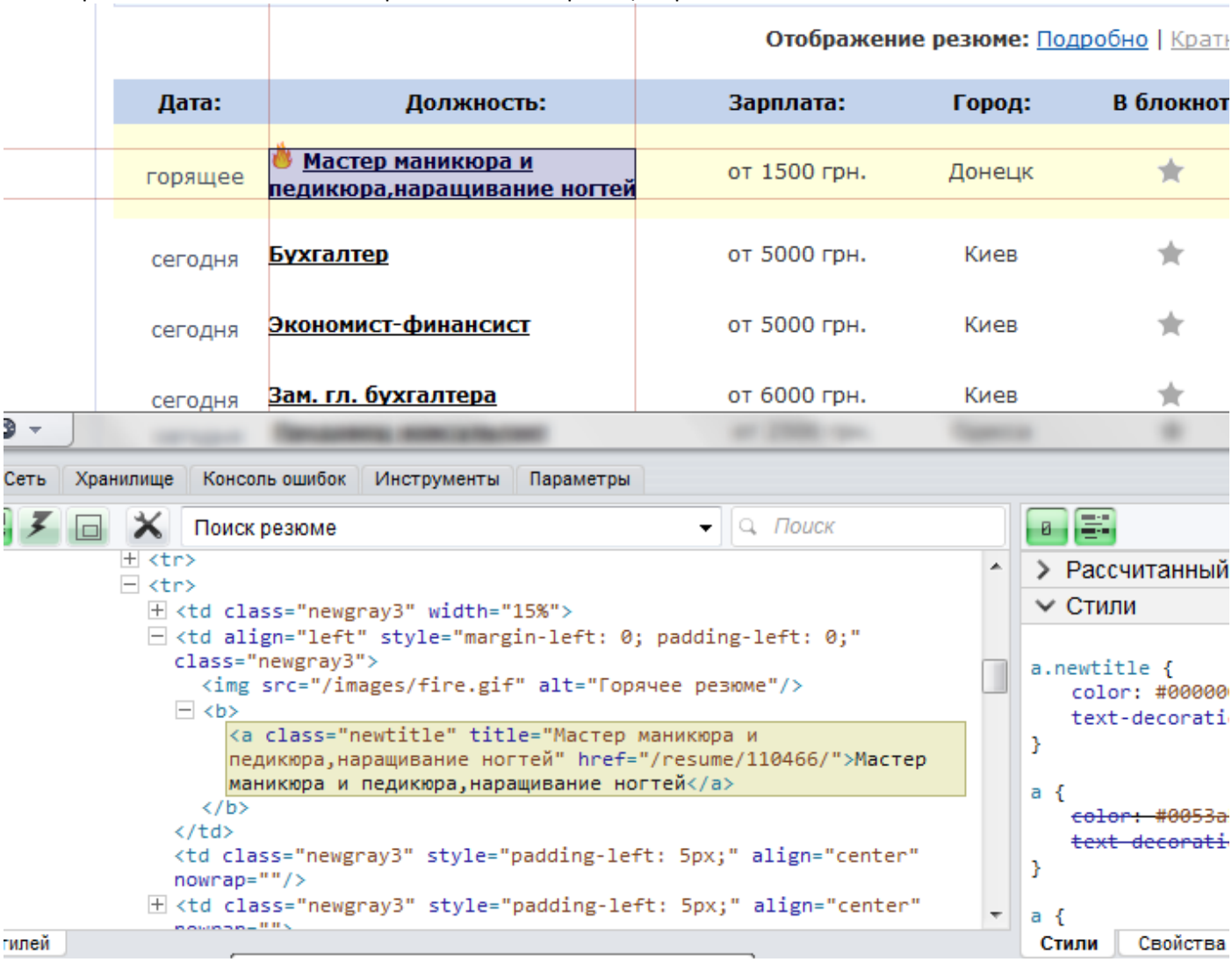
Всякие резюме находятся по ссылке http://rabota-i-trud.com.ua/shortres.php. Тут же можно провести поиск по критериям. Все как у людей ☺.



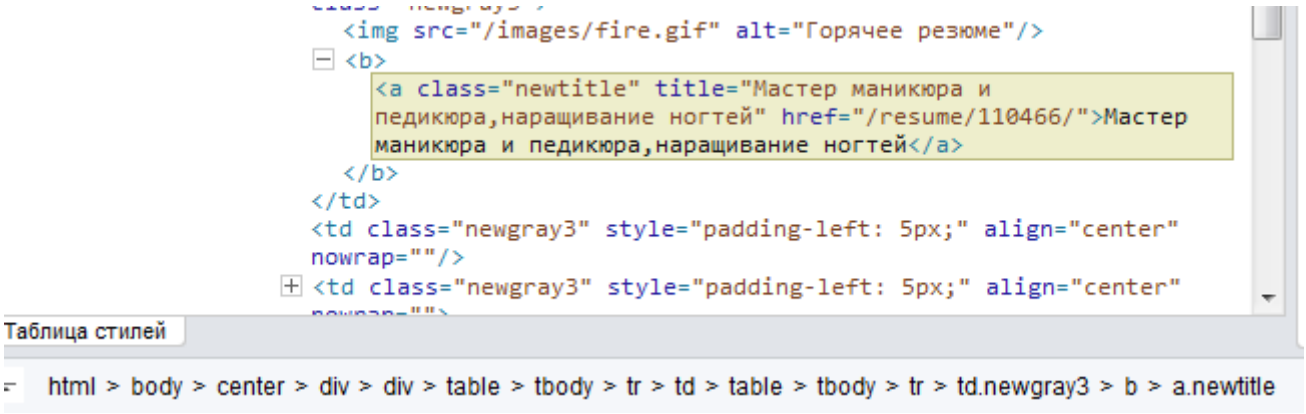
Парсинг этого сайта состоит из двух частей:

- Найти ссылки на интересующие нас страницы.
- Выбрать из этих страниц необходимую информацию.

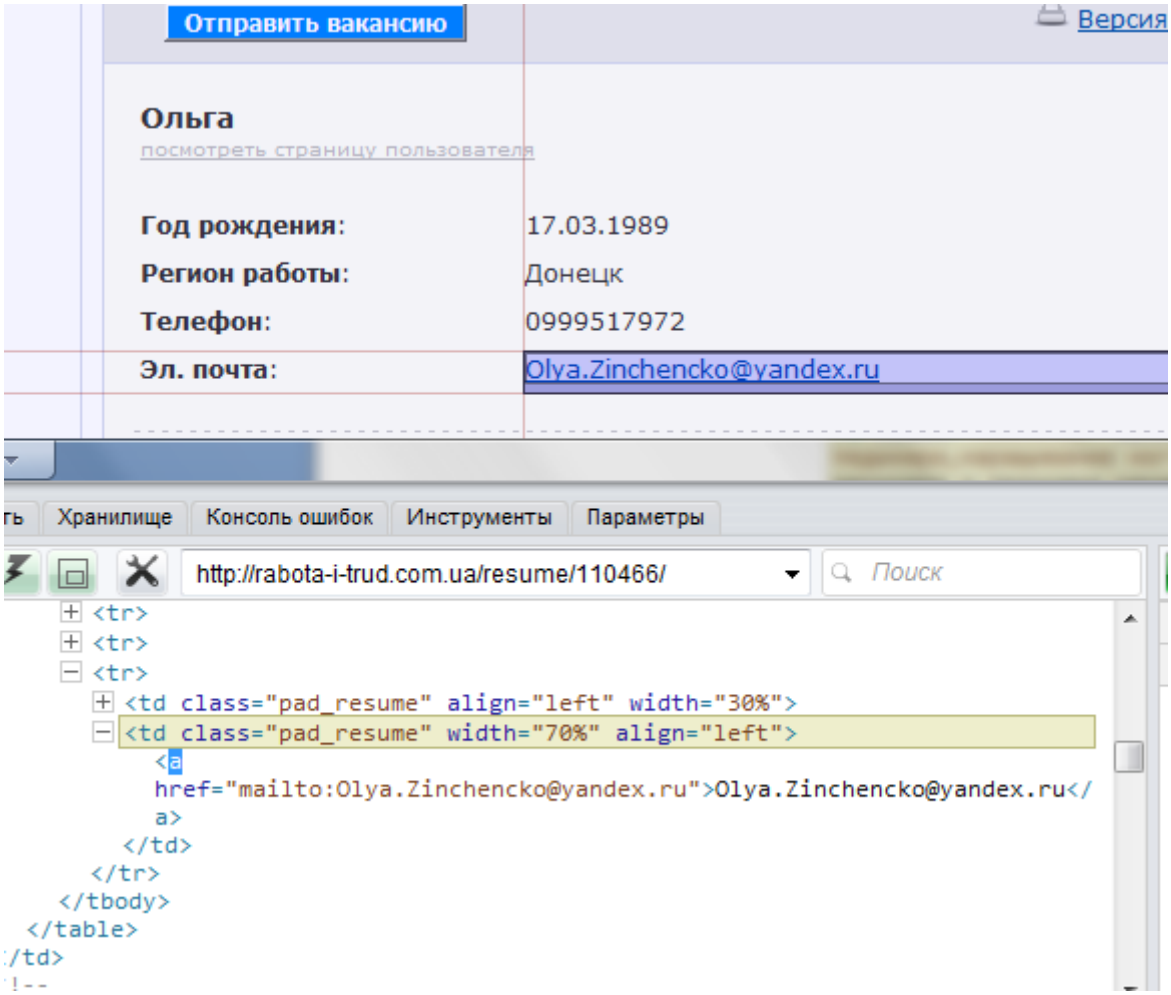
Посмотрим на "отличительные" черты ссылок на страницы с резюме.



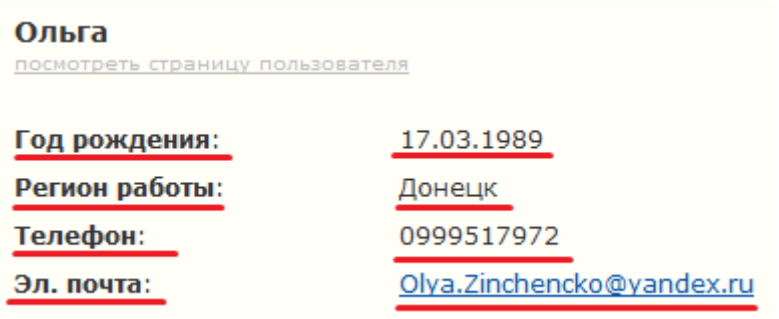
Сразу бросается в глаза, что ссылки на резюме имеют атрибут class со значением newtitle. Просмотрев код страницы, приходим к выводу, что ни у каких других элементов на страницы атрибута class с таким значением нет. Так что **можем использовать его как критерий поиска**. А что если посмотреть с другой стороны? Что если взять полный путь от корня документа?



На рисунке он показан в самом низу. При программной реализации к этому еще вернемся.
Теперь надо посмотреть на страницу с резюме и выбрать критерии поиска DOM-узлов.



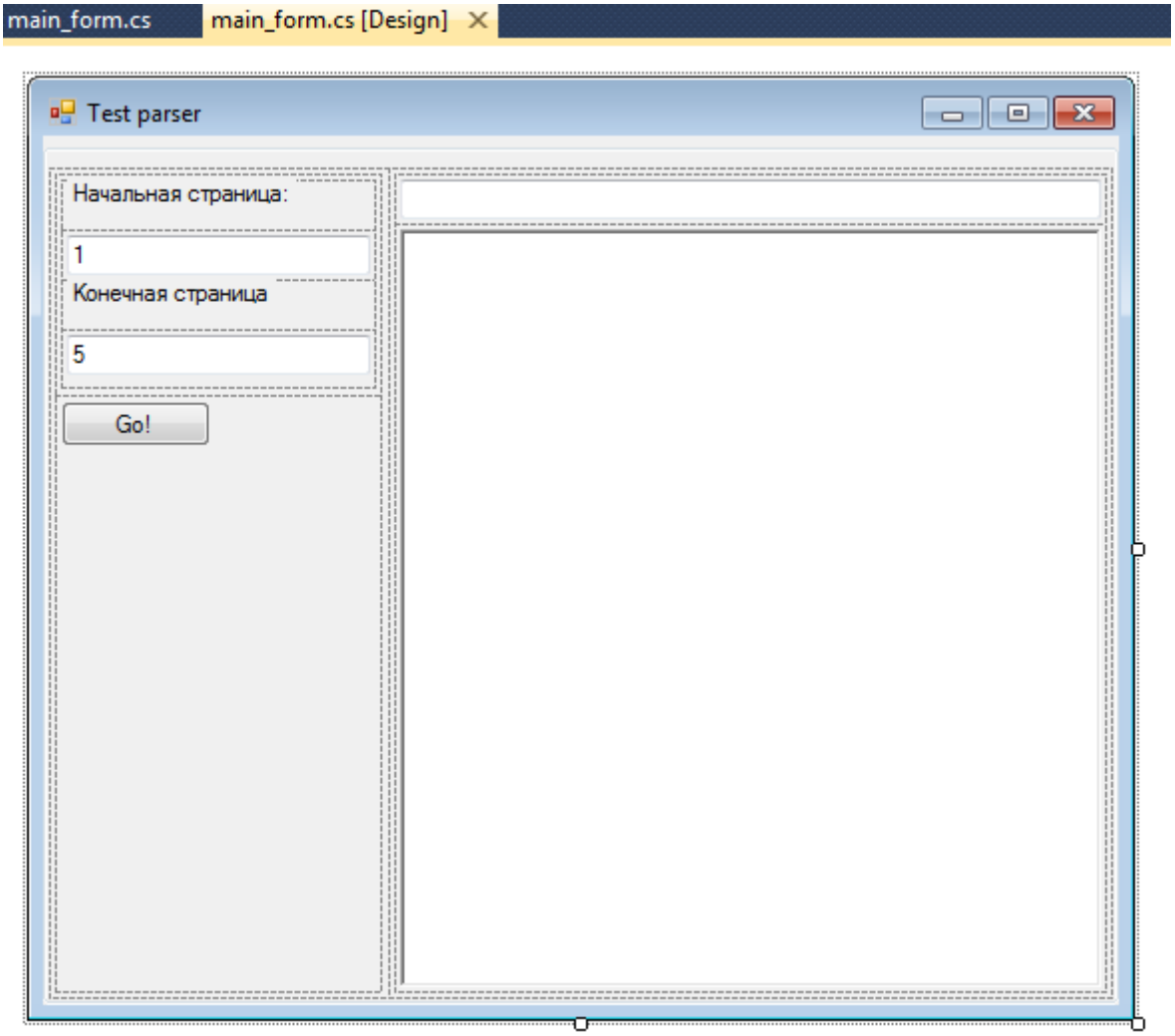
Тут тоже все оказалось не сложно. Нам подходят элементы td (ячейки таблицы), у который class = pad_resume. В такой выборке мы получим



Отбросить левый столбец можно просто обходя список DOM-узлов с шагом +2 (начиная со второго элемента).

Кодим. Кодим. Кодим!

Для начала набросаем на форму элементы ввода/вывода. Пользователь должен вводить url с критериями выборки, первую страницу поиска и последнюю (это 3 TextBox'a). На выходе пользователь получаем много-много напарсенной информации (для простоты поставим RichTextBox). Еще парочка строк с описанием полей ввода и кнопка запуска парсинга. Получается что-то такое:



Пунктирные линии - это отголосок моей "любви" к "резиновым" дизайнам.

Теперь все-таки кодим ☺.
Нам понадобится подключить несколько namespace:

```
using System.Net;
using System.Web;
using System.IO;
using System.Threading;
```

Парсинг будет идти в отдельном потоке. Подробнее о том, как грамотно писать многопоточное приложение, можно прочитать в статье deface (https://forum.xaknet.ru/showthread.php?t=11675). Посему, не буду останавливаться на этом.

Нам нужен метод для получения HTML-кода страницы. В Интернете существуют сотни примеров как это можно сделать. Возьмем один из них и немного допилим:

```
public string getRequest(string url)
        {
            try
            {
                var httpWebRequest = (HttpWebRequest) WebRequest.Create(url);
                httpWebRequest.AllowAutoRedirect = false;//Запрещаем автоматический редирект
                httpWebRequest.Method = "GET"; //Можно не указывать, по умолчанию используется GET.
                httpWebRequest.Referer = "http://google.com"; // Реферер. Тут можно указать любой
URL
                using (var httpWebResponse = (HttpWebResponse) httpWebRequest.GetResponse())
                {
                    using (var stream = httpWebResponse.GetResponseStream())
                    {
                        using (var reader = new StreamReader(stream,
Encoding.GetEncoding(httpWebResponse.CharacterSet)))
```

```
                    {
                        return reader.ReadToEnd();
                    }
                }
            }
        }
        catch
        {
            return String.Empty;
        }
    }
```

Готово.

Теперь надо полученный HTML-код обработать. Для начала создаем объект класса HtmlDocument:

```
HtmlAgilityPack.HtmlDocument doc = new HtmlAgilityPack.HtmlDocument();
```

Загружаем в doc полученный HTML:

```
doc.LoadHtml(getRequest(url));
```

Надо сказать несколько слов о синтаксисе XPath, который мы будем использовать. Базой языка XPath являются *оси*(ниже приведены только те, которые были использованы мной):

- attribute:: — Возвращает множество атрибутов текущего элемента (сокращенно - @).
- descendant-or-self:: — Возвращает полное множество потомков и текущий элемент (сокращенно - //).

Полный список можно найти по ссылке - http://ru.wikipedia.org/wiki/XPath#.D0.9E.D1.81.D0.B8

Так же есть ряд функций для работы с множествами:

- [] — дополнительные условия выборки
- / — определяет уровень дерева

Полный список можно найти по ссылке - http://ru.wikipedia.org/wiki/XPath#.D0.A4.D1.83.D0.BD.D0.BA.D1.86.D0.B8.D0.B8_.D1.81_.D0.BC.D0.BD.D0.BE.D0.B6.D0.B5.D1.81.D1.82.D0.B2.D0.B0.D0.BC.D0.B8

Итак, необходимо составить правило для выборки ссылок на страницы с резюме. Мы говорили, что это элементы a с атрибутом class равным newtitle.

1. Для начала: //
2. Теперь надо указать, что выбираются ссылки: //a
3. Но надо конкретизировать, какие ссылки мы выбираем: //a[]
4. Надо ссылки с атрибутом class равным newtitle: **//a[@class='newtitle']**

Есть.
При просмотре страниц мы говорили, что можно взять полный путь от корня документа:
/html/body/center/div/div[2]/table[2]/tr/td/table[3]/tr/table/tr/td[2]/b/a
Оба правила отвечают одинаковым наборам элементов.
У созданного объекта doc есть свойство `DocumentNode` (указывает на верхний узел документа). У него же в свою очередь есть методы `SelectNodes` и `SelectSingleNode`. Первый выбирает коллекцию элементов, а второй - только один. Нам нужен первый метод.

```
HtmlNodeCollection c = doc.DocumentNode.SelectNodes("//a[@class='newtitle']");
```

Важно! Этот метод может вернуть null, если не будет найдено элементов.
Значит надо сделать проверку:

```
if (c != null)
```

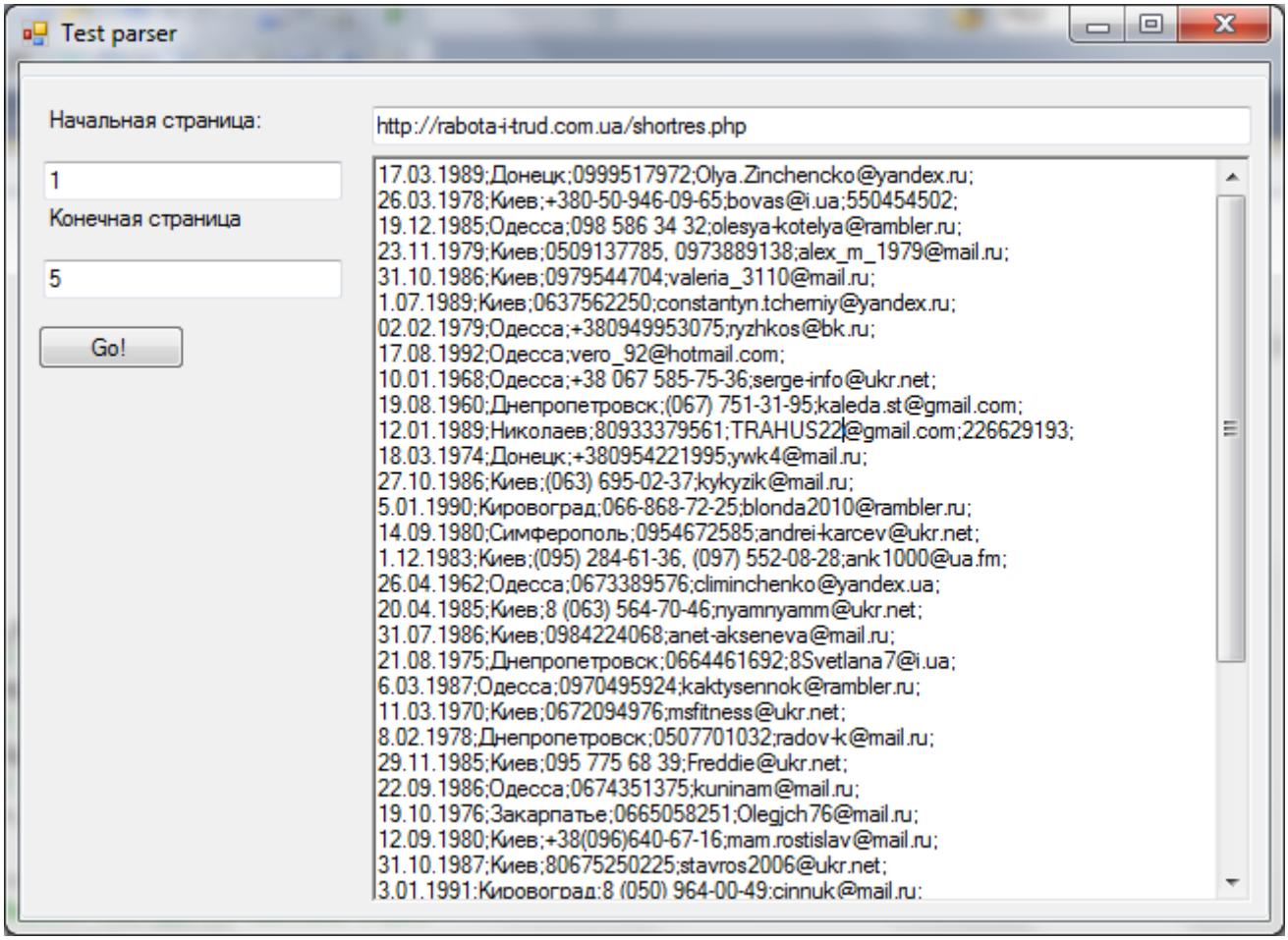
Далее в цикле обрабатываем каждый элемент коллекции. У этих элементов нас интересует атрибут href (доступ к нему можно получить через массив атрибутов Attributes). И, если он не null, то загружаем страничку по этой ссылке. На загруженной страничке с резюме нас интересуют ячейки таблицы с атрибутом class равным pad_resume.
Правило для выборки будет таким:
//td[@class='pad_resume']

Как говорилось раньше, нам нужны только четные ячейки.
Да, если взять правило **//td[@class='list_info_resume']**, то можно получить другие данные на странице с резюме (Образование, Опыт работы, Дополнительная информация, Пожелания к будущей работе).

Полный код метода приведен ниже:

```
private void start()
        {
            my_delegate = new add_text(add_text_method);
            // В цикле обрабатываем странички с first_page по last_page (те, что указали в полях
ввода)
            for (int i = this.first_page - 1; i < this.last_page; i++)
            {
                string content = getRequest(tb_url.Text + this.param_separator + "p=" + i);
                HtmlAgilityPack.HtmlDocument doc = new HtmlAgilityPack.HtmlDocument();
                doc.LoadHtml(content);
                // Получаем список ссылок на страницы с резюме
                HtmlNodeCollection c = doc.DocumentNode.SelectNodes("//a[@class='newtitle']");
                if (c != null)
                {
                    // Обрабатываем каждую страницу (парсим из нее выбранные данные)
                    foreach (HtmlNode n in c)
                    {
                        if (n.Attributes["href"] != null)
                        {
                            // Загружаем страничку с резюме
                            string u = main_url + n.Attributes["href"].Value;
                            string res_cn = getRequest(u);
                            // Парсим страничку с резюме
                            HtmlAgilityPack.HtmlDocument d = new HtmlAgilityPack.HtmlDocument();
                            d.LoadHtml(res_cn);
                            // Выбираем ячейки с нужными данными
                            HtmlNodeCollection pads =
d.DocumentNode.SelectNodes("//td[@class='pad_resume']");
                            if (pads != null)
                            {
                                // Нам нужны только четные ячейки
                                int j = 1;
                                while (j < pads.Count) {
                                    // Записываем текст из ячейки в RTB (убираем все лишнее через
trim)
                                    rtb_output.Invoke(my_delegate, new object[] {
pads[j].InnerText.Trim() + ";" });
                                    j = j + 2;
                                }
                                rtb_output.Invoke(my_delegate, new object[] { "\n" });
                            }
                        }
                    }
                }
            }
            a_delegate = new set_text(set_text_method);
            // Убираем дубли
            rtb_output.Invoke(a_delegate, new object[] { array_unique(rtb_output.Lines) });\
            // Делаем элементы формы активными
            tlp_main.Enabled = true;
            // Завершаем поток
            tr.Abort();
        }
```

Результат работы программы видно на рисунке ниже:



Выводы

Как видим, работа с Xpath очень проста и позволяет получить требуемый результат без особых усилий. В этом небольшом мануале показана малая часть возможностей XPath и HAP. Сокрытое остается на рассмотрение читателей.