

Mobile Apps

Regular Web: Using Regular Expressions to build powerful-yet-light mobile applications

This article discusses:

- ActiveX Control Hosting in .NET Compact Framework
 - .NET Networking
 - .NET Regular Expressions
-

Introduction

*I*f we had to make an overview of the available commercial software that runs on Windows Mobile devices these days, especially Pocket PC's, we would have to include a really rich variety of applications:

- Utilities, like clocks, calculators, battery and memory management
- Entertainment, like TV, radio or ringtones
- Communication, like IM's or mail clients
- All kinds of mini-applications about astrology, diet or games

The list is extensive, however there is one important remark that the watchful reader might make. While an abundance of applications is available, there is still great lack in **Web integration**. Let's explain.

Current desktop software trends are in vast-majority web or at least network-based. Every popular application has at least a basic set of networking capabilities. The most popular among them are fully web-based. The huge popularity of these applications, including social networking applications like **facebook.com** or **myspace.com**, relies on the Internet's endless possibilities for obtaining both a resource and application field. Every such application uses the Internet in a different way, for example games for multiplayer mode, development platforms for online help and so on.

The advent of Web 2.0 has brought a whole new generation of applications which achieved a clean sweep over traditional stand-alone applications. Due to technology limitations but also to physical constraints mobile device applications have fallen behind on this race. First, lack of Web 2.0 enabling technologies, like AJAX, were not supported in mobile browsers until 2006. To make matters worse, even if the technologies were there, physical limitations, like small screen size or little available resources will always be a brake for web applications to have complete or even decent mobile counter-parts.

While mobile AJAX will be definitively a major field of development in the years to come, since .NET Compact Framework 1.0 and mostly after version 2.0, developers already has the tools to grasp the power of any kind of web applications. In this article I am discussing this alternative development practice that makes use of networking capabilities of the .NET CF platform in conjunction with regular expressions that assist the described applications to grab the information they actually need. The final step in our development paradigm is to use the information to deliver specific content. ActiveX controls can be used to do this. Since .NET CF does not support ActiveX control hosting we will have to refer to already described programming techniques and tools that can help in our goal to present with concrete application examples. These examples include 4 different applications:

- An application that returns relative web results in response to a user query and can display them in a browser on user demand
- An application that returns flash games found freely on the web that match his query. The user can also launch and play the game he selects
- An application that translates a sentence in English to Spanish and
- An application that can ‘think’ over a sequence of numbers and display the next number in the sequence, much like in popular IQ tests. For example, faced to an input *1,2,3,4,5* the application can predict that the next number should be *6* but also describe the relation among the numbers-in this case to be able to output “*positive integers*”

While heterogeneous at first glance, all these mini-applications are built on top of a unified framework. This framework can consume the vast information present in the Web to provide with the desired functionalities and will be described in what follows. Last but not least, this architecture leads to a minimal source and binary code size. In our example, 12 KB of binary core are enough to provide will the applications presented above.

Regular Web Framework - Overview

Before delving into more technical details, I would like to explain the general idea of how the **Regular Web** framework is built. The basic requirement is that we have mobile device, with networking capabilities that has full access to a web of applications, much like in the form they exist today. The term ‘web applications’ refers no only to enterprise scale or any industry standard-compliant services or applications, commonly referred to as ‘Web Services’, but to the whole set of dynamic World Wide Web. Each page, primarily when it is dynamic, can be thought of as being a service or a function that the framework can invoke. Every such dynamic web site is hosted on an entity that is reasonable to call ‘service

provider' and when invoked can provide with the desired information we seek to add as a functionality in our application.

The most common example of the above scenario is the use of a search engine, which is essentially an HTTP GET request. The search engine's server application will respond to our query with an HTML file that will be on-the-fly created according to our search. In this way, for any kind of information we want, we might choose different service providers. For example, if we want to make a simple web search we might turn to **google.com** or **msn.com**. For weather information we might turn to **weather.com** and for currency information to **xe.com**. However, even if we are able to receive a response, every provider always follows different formats when presenting the results. What we need for our weather application is that it will be rainy tomorrow in our town and of course not all the html code that comes along. So, we have some information over the specific format that the service provider uses. We would then be able to craft our own processing tools to extract the information from the server's response. However, there is a better alternative to this, **regular expressions**.

The regular expressions (or **RegExp**) libraries in .NET are used to process the providers' HTTP responses. Processing is therefore fast and has a unified programming interface inside the framework. To be more specific, each service provider is modeled as having a list of services/operations it can offer (e.g. web search, weather, news etc) but also as having a specific **pattern** for each service that it offers. This imposes a different programming model that has different challenges than the traditional one. While the latter, links different modules and services both on compile and run time, using a qualified name and input arguments, e.g. `System.Net.Dns.GetHostEntry(string host)`, the former links to services only on runtime and on a network connection, using an interface consisting of the services' URL and the pattern to match, e.g. `Google.webSearch(string url, string pattern, string query)`. Using the .NET regular expressions engine, we have in hands a very fast and efficient method of obtaining any kind of information that is hosted in the web. This information, in the most interesting cases, can be manipulated only by special software components, like media or game players. The next challenge we face is manipulating the various formats of the different content we have obtained.

To achieve this to the greatest possible extent it would be convenient to use existing software modules capable of doing the job. For example, if we had a WAV file to 'use' it would be better to re-use existing media players, like the Windows Media Player. Windows technologies have defined interfaces like the COM interface that enable component-based application development. A special category of COM interfaces, namely the ActiveX components can be used for our purposes. However, there are obstacles in this step. Although the .NET Compact Framework 2.0 has support for COM Interoperability it lacks control hosting capabilities. This basically means that if we wished to use Windows Media Player to play our audio file, we would have to launch it as an external application. This [Hosting ActiveX Controls in the .NET Compact Framework 2.0](#) MSDN article provides with all the necessary information and techniques on how to bypass this limitation and extend the .NET CF's capabilities, on which this article and the accompanying applications are heavily based.

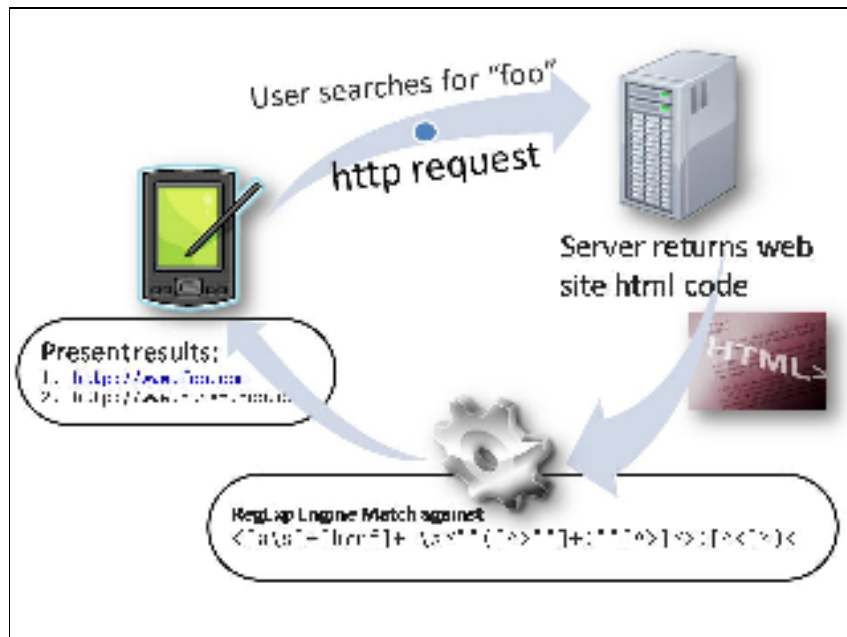


Figure 1. *Integrating web search functionality*

As a conclusion to the Regular Web Framework overview, Figure 1 demonstrates a scenario in which an application wishes to perform web search over user defined search terms. While the server responds with a typical HTML-encoded stream, the RegExp engine is used in order to parse the results. The list of links we acquire can be further used for other purposes rather than simple demonstration. A WebBrowser control could be used for example to manipulate the results. While the pattern that is showed in the figure can be used to retrieve hyperlinks in an HTML page independently of the site we use for our query, inside the framework each site and service is associated with a unique pattern. In this way, a library of software modules can be functionally replaced with a library of regular expressions patterns.

Regular Web Framework – Technical details

.NET Networking basics

There is no need for sophisticated networking techniques in order for the framework and the application built upon it, to run. The System.Net and System.Net.Sockets provide with complete support for all basic but also complex networking tasks. In addition to supporting HTTP-based protocols it also supports SOAP-based communication (Web Services) and secure connections (SSL)

The Crawler class inside the applications is able of fetching a page (make HTTP GET request and receive response) and also making HTTP POST requests. The System.Net provides with two very convenient classes: `HttpRequest` and `HttpWebResponse`. Creating a GET request is as easy as the following snippet:

```
HttpRequest request = (HttpRequest)WebRequest.Create("http://foo.com");  
  
while issuing the request and reading the response can be easily done by  
  
HttpWebResponse response = (HttpWebResponse) request.GetResponse();  
  
System.IO.StreamReader reader = new System.IO.StreamReader(response.GetResponseStream());  
  
string responseAsString = (new StringBuilder(reader.ReadToEnd()).ToString());
```

To issue a POST request the following snippet can be used:

```
HttpRequest request = (HttpRequest) WebRequest.Create("http://foo.com");  
  
ASCIIEncoding encoding = new ASCIIEncoding();  
  
byte[] postData = encoding.GetBytes(postDataAsString);  
  
req.Method = "POST";  
  
req.ContentType = "application/x-www-form-urlencoded";  
  
req.ContentLength = data.Length;  
  
System.IO.Stream reqStream = req.GetRequestStream();  
  
reqStream.Write(postData,0,postData.Length);  
  
reqStream.Close();
```

and then create and read the response object as in the previous example. When not working on .NET Compact Framework it might be very convenient to use the `System.Net.WebClient` class. An example usage might be:

```

WebClient client = new WebClient();

System.Collections.Specialized.NameValueCollection postValuesCollection = new
System.Collections.Specialize.NameValueCollection();

postValuesCollection.add("language", "en");

//..add post parameters here....

byte[] response = client.UploadValues("http://foo.com", "POST", postValuesCollection);

string responseAsString = Encoding.ASCII.GetString(response);

```

The applications presented in this article do not require more than that. The Crawler that is provided inside the framework is able to basic HTTP communication that suffices to build all desired functionalities. In addition, since there are not parallel tasks involved we have avoided mentioning the Asynchronous model of .NET network programming, which basically consists of the Begin/End versions of the relevant functions. Having built the tools to communicate with the web environment it is time to process server responses. This work is carried by the .NET RegExp engine.

.NET Regular Expressions

*R*egular expressions are a very powerful text-processing tool and form the enabling technology of pattern matching. For computer scientists, regular expressions exist from the moment Ken Thompson incorporated them into the QED editor. Now, Perl is the language of reference when it comes to text processing and all other implementations of regular expressions in other platforms (like .NET or Java) follow at least Perl's syntax and notation.

.NET functions and methods for regular expressions reside in the assembly System.Text.RegularExpressions, which additionally presents with some interesting capabilities, like right-to-left matching and the more 'artistic' on-the-fly expression compilation. The basic classes are the Regex, Match, Group, Capture and their collections for the last three. An interesting class is also provided, namely the MatchEvaluator that enables expression-based patterns that can be used for on-the-fly processing operations.

However, rich functionality can be achieved simply by using procedural-based patterns, which technically interprets to using the Regex, Match and Group classes which are also the most heavily used classes in our applications. As an example of use consider a sentence like: "Please send me an email to fooman@foo.om with subject Online Application." and that we had to retrieve the e-mail address parts, we might apply "(([\w\d]+)@([\w\d]+\.[\w\d]+))" as a quick and dirty pattern to match which matches the general template X@X.X where X is any text with alphanumeric characters(including underscore). The code might look like the following:

```

string pattern = @"([\w\d]+)@([\w\d]+\.[\w\d]+)";
string input = "Please send an e-mail to fooman@fo.com with subject Online Application.";
Regex reg = new Regex(pattern, RegexOptions.IgnoreCase);

```

```
string username;  
string domain;  
MatchCollection mc = reg.Matches(input);  
username = mc[0].Groups[2].Value;  
domain = mc[0].Groups[3].Value;
```

The above code outputs the username-fooman- and the domain-foo.com- to the local variables. The linear time in which the .NET regexp engine operates guarantees speed even when processing large amounts of data, for example when processing web page HTML code which we use in our applications. In addition, no expert knowledge is needed to start creating useful libraries for the mobile applications. For example, in the translation application the pattern is very simple since it declares only of one group.

As mentioned earlier, the concept of 'linking' to web sites as if they were on-device library modules imposes a new programming model. This model requires for the method invocation parameters to be different from usual case: the URL of the site and the specific operation required. The framework keeps a dictionary of associated to such function arguments and uses the Regexp engine in order to construct the output. The concept of invoking methods that reside on web servers is not new and it is actually the basic concept of Web Services. However there are several important differences between these two models. On the one hand Web Services are built based on industry standards and XML-based protocols to which both the consumer and the provider must comply. On the other hand, in our model, there is no need for a priori conventions, since the client is in total control of what he wants to 'hear' from the server. In addition, ignoring standardization issues and slow industry adaptation to WS protocols, it is also important that providers expose as Web Services only the services that they are *willing to* expose. Using pattern matching, the consumer can receive *all* the services he is *able to* handle. Resource constraints and low quality of service (bandwidth problems, slow user interface) might be obstacles for the client reaching out for all web applications that are available.

.NET CF Control Hosting

*A*s mentioned earlier, the .NET Compact Framework lacks support for control hosting. However everything that is needed to accomplish this task is already in place along with a rich set of tools that come along with Microsoft Visual Studio and the .NET CF SDK. There is no better reference to the methodology than [Hosting ActiveX Controls in the .NET Compact Framework 2.0](#) MSDN article. In what follows we will briefly refer to some of its key points.

Essentially to host a control in the .NET CF it is necessary to have the COM interface definitions and create the associated control class that inherits from the AxHost class. Manually this can be a tedious task. Fortunately, the .NET SDK provides with the tool, the **AxImp.exe**, that can be used to produce the necessary Interop code but also provide with the control source code. It is usual for COM components to share a common type library

both on desktop and mobile device versions. This is true for example for the Windows Media Player. In this case it is possible to use the **wmp.dll** located in the system32 Windows folder to create the interfaces for the mobile counter-part. From the command prompt, the command `AxImp.exe C:\WINDOWS\system32\wmp.dll /source` will generate the **AxWindowsMediaPlayer** control in the Interop library but also the source code for this. Taking care out of some details, it is possible to compile the source file against the .NET CF and be able to use the control on the device.

The applications

*T*his article comes along with some example applications that are built upon the Regular Web framework and it is intended to depict the extent of functionality that can be achieved by implementing and using the concepts described in previous sections.

The demo application is comprised of 4 'mini-apps' which I will describe in what follows. In overall, the application is based on a class (RegularWeb.Core) which has the role of the module manager. The services are grouped based on the domain that provides them. Each of this domain is a class in the RegularWeb.Components namespace and inherits the RegularWeb.Components.RegularWebComponent abstract class which exposes the appropriate interfaces for service invocation.

Web Search Mini-app

The application uses google.com for web searching. This case is straightforward. The user enters a query and the application invokes a specially crafted URL to invoke an HTTP GET request over the site. The result is parsed by the RegExp engine using the pattern for hyperlink retrieval. The matching yields three groups, one for the URL, the anchor text and the total HREF tag. The results are presented to the user, any of which he can select and browse through the integrated WebBrowser control.

Game Search Mini-app

The game search is more interesting. The user enters a game he wishes to play (e.g. tic tac toe) then the application returns flash files that match his search. Upon selection the Flash control is activated and plays the game on-the-fly. This creates the feeling that the user has an unlimited number of games installed in his device. It should be noted that the Flash Lite from Adobe should be installed in the device. The player is free and is available from the Adobe site [under this location](#).

Translation Mini-app

The framework is used also to consume translation services. In the demo application, the user can enter a text phrase which will be translated in Spanish.

'I-am-smart' Mini-app

In this scenario the user enters a sequence of integers delimited by commas and asks the computer what is the next number. A special component in the framework, has integrated a number of AT&T research projects. In this case special patterns that parse the Online Encyclopedia of Integer Sequences is used in order to predict the next number in the sequence.