TECHNOLOGY CORNER

A Regular Expression Training App

Nick V. Flor
Marketing, Information & Decision Sciences
Anderson School of Management
University of New Mexico
nickflor@unm.edu

ABSTRACT

Regular expressions enable digital forensic analysts to find information in files. The best way for an analyst to become proficient in writing regular expressions is to practice. This paper presents the code for an app that allows an analyst to practice writing regular expressions.

INTRODUCTION

For many of us old-time Unix programmers, our first experience writing regular expressions was within the Vi editor – where we used them to search and replace text in our C-code. As a fascinating historical note, Ken Thompson, the architect of regular expressions in Unix (Thompson, 1968), credits Kleene (1956) with the theory of regular expressions. In terms of Kleene's theory, a regular expression was a neural network representing a regular event, where a neuron fired in response to some stimulus (Kleene, 1956; p. 62). So, although we tend to think of regular expressions as originally a programming tool, its first historical application was as a neuroscience modeling tool.

Over time, regular expressions have found important applications outside of programming. In particular, today's digital forensic analysts use regular expressions to find information evidence in computer logs and in user files (see Stewart & Uckelman, 2011 and Vasiliadis, Polychronakis, Antonatos, Markatos, & Ioannidis, 2009, for recent examples).

However, due to the history of regular expressions as a tool for programmers, non-programmers often find the syntax cryptic and difficult to learn. For example, the following is a regular expression for recognizing valid e-mails typed into a textbox (Microsoft, n.d.): `` ^(?("")(""[^""]+?""@)|(([0-9a-z]((\.(?!\.))|[-!#\$%&'\*\+/=\?\^`\{\}\|~\w])*)(?<=[0-9a-z])@))(?(\[)(\[(\d{1,3}\.){3}\d{1,3}\])|(([0-9a-z][-\w]*[0-9a-z]*\.)+[a-z0-9]{2,17}))$ ``

How does one become proficient in writing regular expression? This paper presents an app that I use to train students in writing regular expressions. The app is written in C# for Windows 8 devices, but the code is brief and the objects used are found in many other languages and platforms. Thus, it can be easily ported and extended to non-Windows devices (see Appendix A for app's source code). The intent of this article is not to teach users how to write regular expressions as there are many good online resources including the "Regular Expression Language – Quick Reference" (http://msdn.microsoft.com/en-us/library/az24scfc.aspx).

USER INTERFACE

The user interface consists of two primary input boxes (refer to Figure 1): (1) a one-line textbox where a user can enter a regular expression; and (2) a multi-line textbox where a user can either type or paste the data to be searched. To recreate this interface so that it works with the code in Appendix A, you must name the textboxes **txRegEx** and **txSource**, as well as naming the "Go" button **btnGo**.

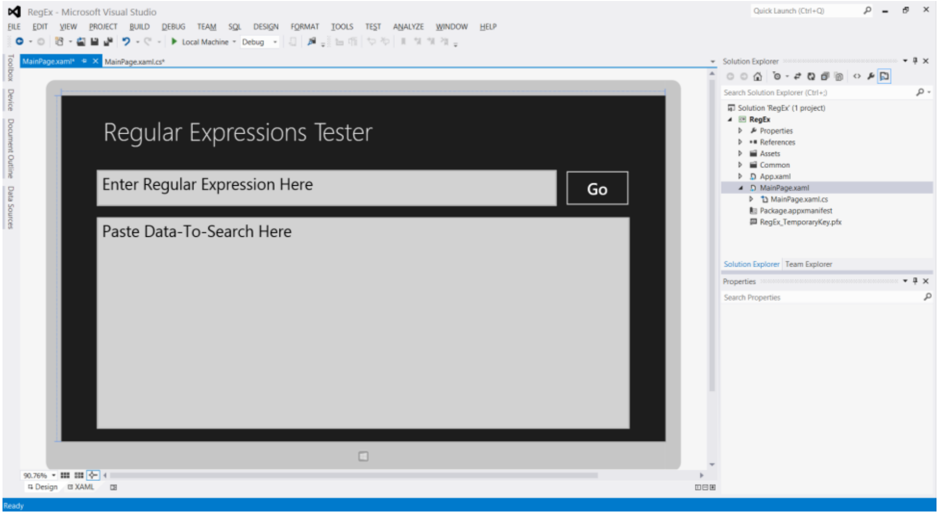
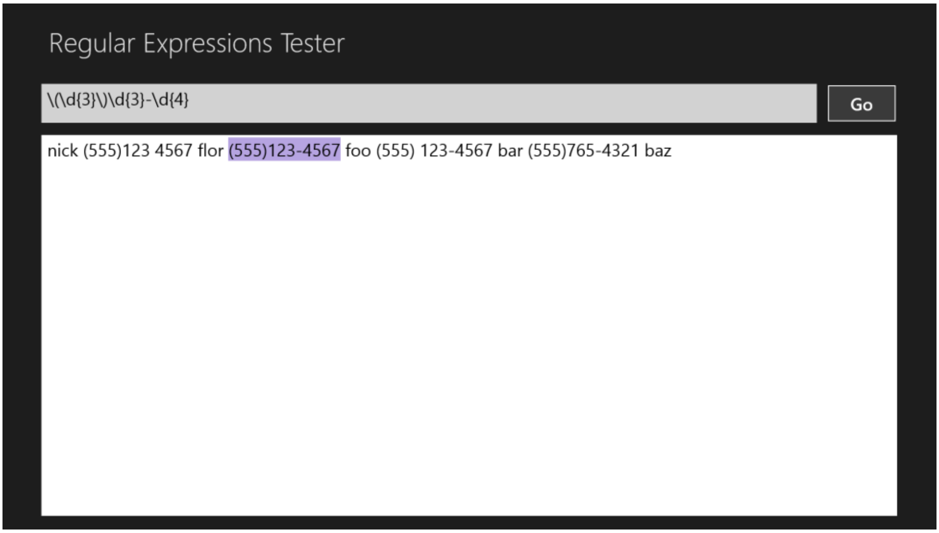

RegEx - Microsoft Visual Studio
Quick Launch (Ctrl+Q)
FILE EDIT VIEW PROJECT BUILD DEBUG TEAM SQL DESIGN FORMAT TOOLS TEST ANALYZE WINDOW HELP
Local Machine
Debug
MainPage.xaml*
MainPage.xaml.cs*
Toolbox Device Document Outline Data Sources
Regular Expressions Tester
Enter Regular Expression Here
Go
Paste Data-To-Search Here
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'RegEx' (1 project)
RegEx
Properties
References
Assets
Common
App.xaml
MainPage.xaml
MainPage.xaml.cs
Package.appxmanifest
RegEx_TemporaryKey.pfx
Solution Explorer Team Explorer
Properties
Search Properties
90.76%
Design
XAML
Ready

Figure 1 The User Interface for the Regular Expression Self-Learning App

SAMPLE OPERATION

Suppose a user wants to write a regular expression to find all phone numbers in a file. The user first guesses that the correct regular expression is *\(\d{3}\)\d{3}-\d{4}* and enters it into the first textbox. That regular expression looks for 3 digits inside matching parentheses, followed by 3 digits, a dash, and 4 digits. The user pastes test data that contains phone numbers into the second textbox, then presses the “Go” button. Upon doing so, the program highlights the information that matches the regular expression (see Figure 2).

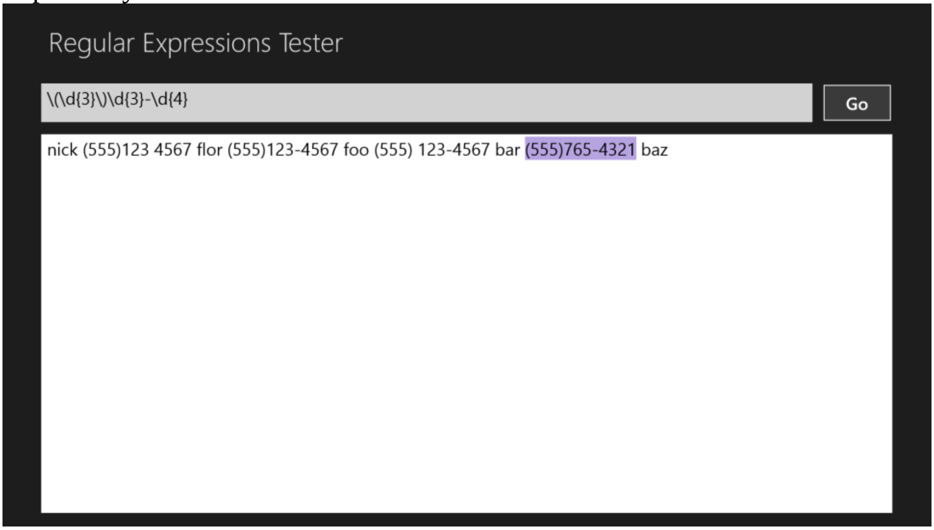


```
\(\d{3}\)\d{3}-\d{4}
```

```
nick (555)123 4567 flor (555)123-4567 foo (555) 123-4567 bar (555)765-4321 baz
```

Figure 2 After tapping the “Go” button, the program highlights the first match

Continuing to press “Go” will cycle between all strings that match the pattern (see Figure 3). Note that the first and third phone numbers were not highlighted because of the lack of a dash and the inclusion of a space after the area code, respectively.



```
\(\d{3}\)\d{3}-\d{4}
```

```
nick (555)123 4567 flor (555)123-4567 foo (555) 123-4567 bar (555)765-4321 baz
```

Figure 3 Pressing “Go” again will highlight other matches.

Thus, the user gets an interactive and a visual indication that his or her regular expression was not quite correct. Thus, the user can try entering another regular expression such as: _\(\d{3}\)\s*\d{3}(-|\s)\d{4}_, which would capture all the phone numbers in the test data. And by entering new test data, the user can further refine the regular expression—ideally until a regular expression is developed that handles all possible phone numbers.

HOW THE CODE WORKS

The first step in writing an app that uses regular expressions is to include the namespace **System.Text.RegularExpressions** (refer to Appendix A). This namespace defines the main objects used to process regular expressions including: **Regex**, **MatchCollection**, and **Match**.

The **Regex** is the primary object. The **Regex.Matches** method takes two strings as input: (1) the string to be searched and (2) a string denoting the regular expression. The method then returns a **MatchCollection** consisting of zero or more **Match**es. The line of code that applies the regular expression to the test data is:

```
mcAllMatches = Regex.Matches(txSource.Text, txRegEx.Text);
```

A **Match** instance contains several properties including **.Index**, which contains the position in the searched string where the pattern was found, and **.Length**, the number of characters in the pattern. Highlighting a match in the test data is accomplished by setting the test data’s **.SelectionStart** property to the **Match**’s **.Index**, and setting the **.SelectionLength** property to **.Length**. The lines of code that highlight the match are:

```
    txSource.SelectionStart = mCurrMatch.Index;
    txSource.SelectionLength = mCurrMatch.Length;
    txSource.Focus(Windows.UI.Xaml.FocusState.Programmatic);
```

DISCUSSION AND CONCLUSION

As one can see, writing a regular expression trainer is straightforward in Windows. Possible extensions to the code include reading the data from a file, and saving the results found to a file. Another good extension would be to specify a URL containing the data, or to read the data from a database. I should mention that in order to make the code easy to understand and to extend, I removed the input and error handling from the app. I hope you enjoy this little app and that you will use it to train either yourself or others in the writing of regular expressions.

REFERENCES

Kleene, S. C. (1956). Representation of Events in Nerve Nets and Finite Automata. In *Automata Studies* (pp. 3-41), C. Shannon, J. McCarthy, and W. Ashby (Eds.). Princeton, NJ: Princeton University Press.

Microsoft (n.d.). How to: Verify that Strings Are in Valid Email Format.

Microsoft.com. Retrieved 1/25/2013 from http://msdn.microsoft.com/en-us/library/01escwtf.aspx

Stewart, J., & Uckelman, J. (2011). Searching Massive Data Streams Using Multipattern Regular Expressions. In G. Peterson & S. Shenoi (Eds), *Advances in Digital Forensics VII* (pp. 49-63).

Vasiliadis, G., Polychronakis, M., Antonatos, S., Markatos, E., & Ioannidis, S. (2009). Regular expression matching on graphics hardware for intrusion detection. In E. Kirda, S. Jha, D. Balzarotti (Eds.) *Recent Advances in Intrusion Detection* (pp. 265-283). Springer Berlin/Heidelberg.

Thompson, K. (1968). Regular Expression Search Algorithm. *Communications of the ACM*, 419-422.

APPENDIX A CODE FOR REGULAR EXPRESSIONS TRAINER

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using Windows.Foundation;
using Windows.Foundation.Collections;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Controls.Primitives;
using Windows.UI.Xaml.Data;
using Windows.UI.Xaml.Input;
using Windows.UI.Xaml.Media;
using Windows.UI.Xaml.Navigation;
using System.Text.RegularExpressions;
namespace RegEx
{
    public sealed partial class MainPage : Page
    {
        public MainPage()
        {
            this.InitializeComponent();
        }
        protected override void OnNavigatedTo(NavigationEventArgs e)
        {
        }
        string sLastSearch="";
        MatchCollection mcAllMatches;
        Match mCurrMatch;
        private void btnGo_Tapped(object sender, TappedRoutedEventArgs e)
        {
            if (txRegEx.Text != sLastSearch)
            {
                sLastSearch = txRegEx.Text;
                mcAllMatches = Regex.Matches(txSource.Text, txRegEx.Text);
                if (mcAllMatches.Count!=0)
                {
                    mCurrMatch = mcAllMatches[0];
                    txSource.SelectionStart = mCurrMatch.Index;
                    txSource.SelectionLength = mCurrMatch.Length;
                    txSource.Focus(Windows.UI.Xaml.FocusState.Programmatic);
                }
            }
            else
            {
                if (mcAllMatches.Count!=0)
                {
                    mCurrMatch = mCurrMatch.NextMatch();
                    if (mCurrMatch.Success == false) mCurrMatch = mcAllMatches[0];
                    txSource.SelectionStart = mCurrMatch.Index;
                    txSource.SelectionLength = mCurrMatch.Length;
                    txSource.Focus(Windows.UI.Xaml.FocusState.Programmatic);
                }
            }
        }
        private void btnGo_DoubleTapped(object sender, DoubleTappedRoutedEventArgs e)
        {
            sLastSearch = "";
        }
    }
}
```