Modern Regex

Max Kleiner

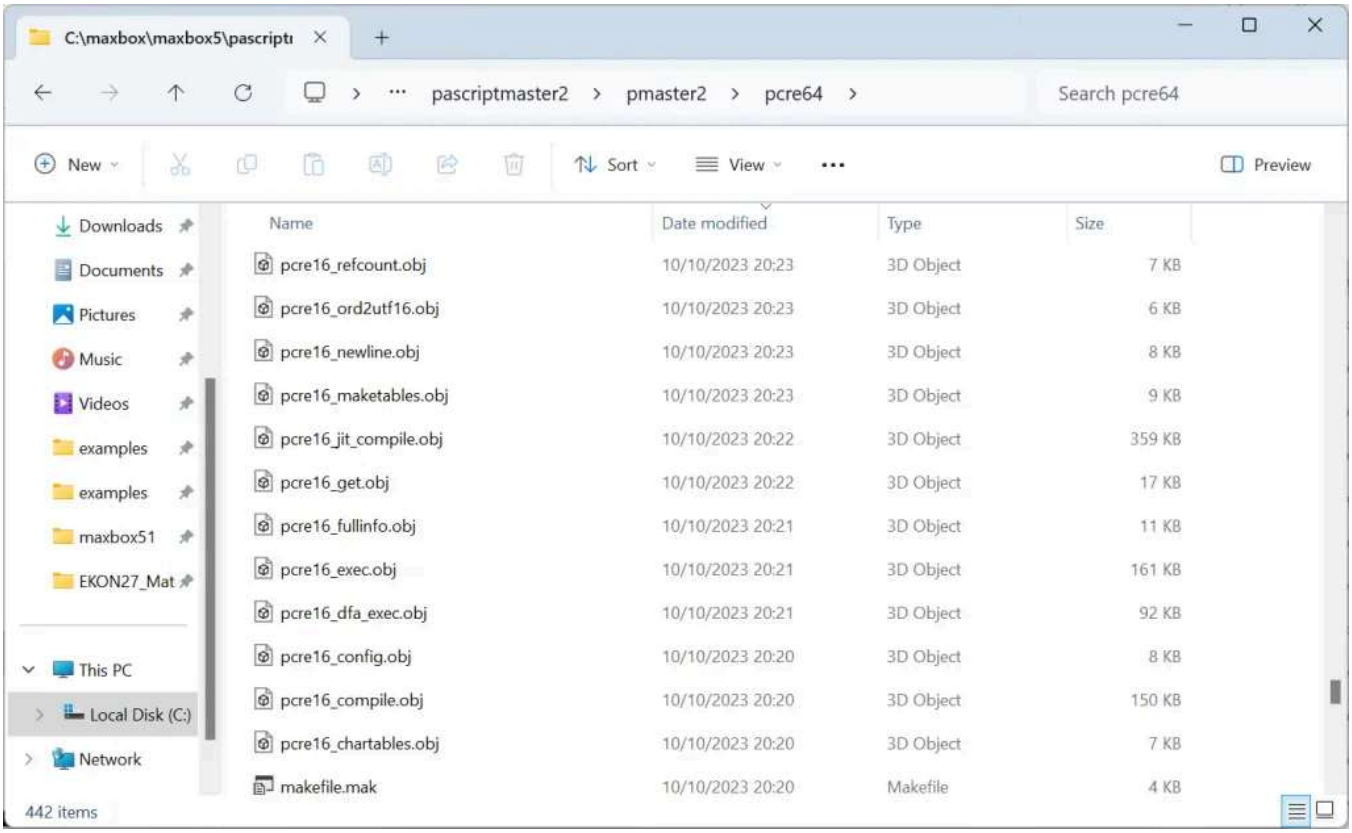
5 min read · 22 hours ago

EKON 25

The king of code is back, namely regular expressions as the Rex of Code in scripts.

TPerlRegEx is a Delphi VCL wrapper around the open-source PCRE (Perl-Compatible Regular Expressions) library. It provides powerful regular expression capabilities similar to those found in the Perl programming language. This version of TPerlRegEx is compatible with the TPerlRegEx class in the RegularExpressionsCore unit in Delphi XE. You can use TPerlRegEx to perform pattern matching, search, and replace operations using regular expressions.

The supplied pcrelib.dll contains PCRE 7.9, compiled with Unicode support.



Compiled with a makefile.mak for maXbox5 for example

By default, OBJ files are used (like above), but you can use the DLL if you have multiple apps using TPerlRegEx and want to save space by linking the OBJ files only once. There's no need to add the pcre unit to your uses clause; it's used internally by TPerlRegEx. TPerlRegEx is licensed under the Mozilla Public License, version 1.1. To download the latest version of TPerlRegEx, visit the official page.

https://www.pcre.org/original/doc/html/pcre16.html

Starting with release 8.30, it is possible to compile a PCRE library that supports 16-bit character strings, including UTF-16 strings, as well as or instead of the original 8-bit library. The majority of the work to make this possible was done by Zoltan Herczeg. The two libraries contain identical sets of functions, used in exactly the

same way. Only the names of the functions and the data types of their arguments and results are different.

To avoid over-complication and reduce the documentation maintenance load, most of the PCRE documentation describes the 8-bit library, with only occasional references to the 16-bit library.

Usage Example:

```
var reg: TPerlRegEx; begin reg := TPerlRegEx.Create(nil); try reg.RegEx := 'ab'
```

```
var reg: TPerlRegEx;
begin
  reg := TPerlRegEx.Create(nil);
  try
    reg.RegEx := 'ab';
    reg.Replacement := '◆';
    reg.Subject := 'ababab';
    reg.ReplaceAll;
    ShowMessage(reg.Subject); // Returns: ◆◆◆
  finally
    reg.Free;
  end;
end;
```

Or you want to replace a pattern-string at the fly or at runtime:

```
with TPerlRegEx.Create do try Subject :='<p>This is text.<br/> This is line 2</
```

```
with TPerlRegEx.Create do
   try
     Subject :='<p>This is text.<br/> This is line 2</p>';
     RegEx := '<[^>]*>';
     //RegEx:= '([</pbr>])*'
     replacement:= ' ';
     replaceall();
```

```
      writeln('res: '+subject);
    finally
      Free;
    end;
```

The output is: res: This is text. This is line 2

A more modern implementation is to code with a **TMatch** and **TMatchCollection** class. This example demonstrates the use of TMatchCollection and TGroupCollection. This example assumes that you have placed a TButton, a TEdit and a TMemo on a form.

```
var Form1: TForm1; mycoll: TMatchCollection; myenum: TMatchCollectionEnumerator
```

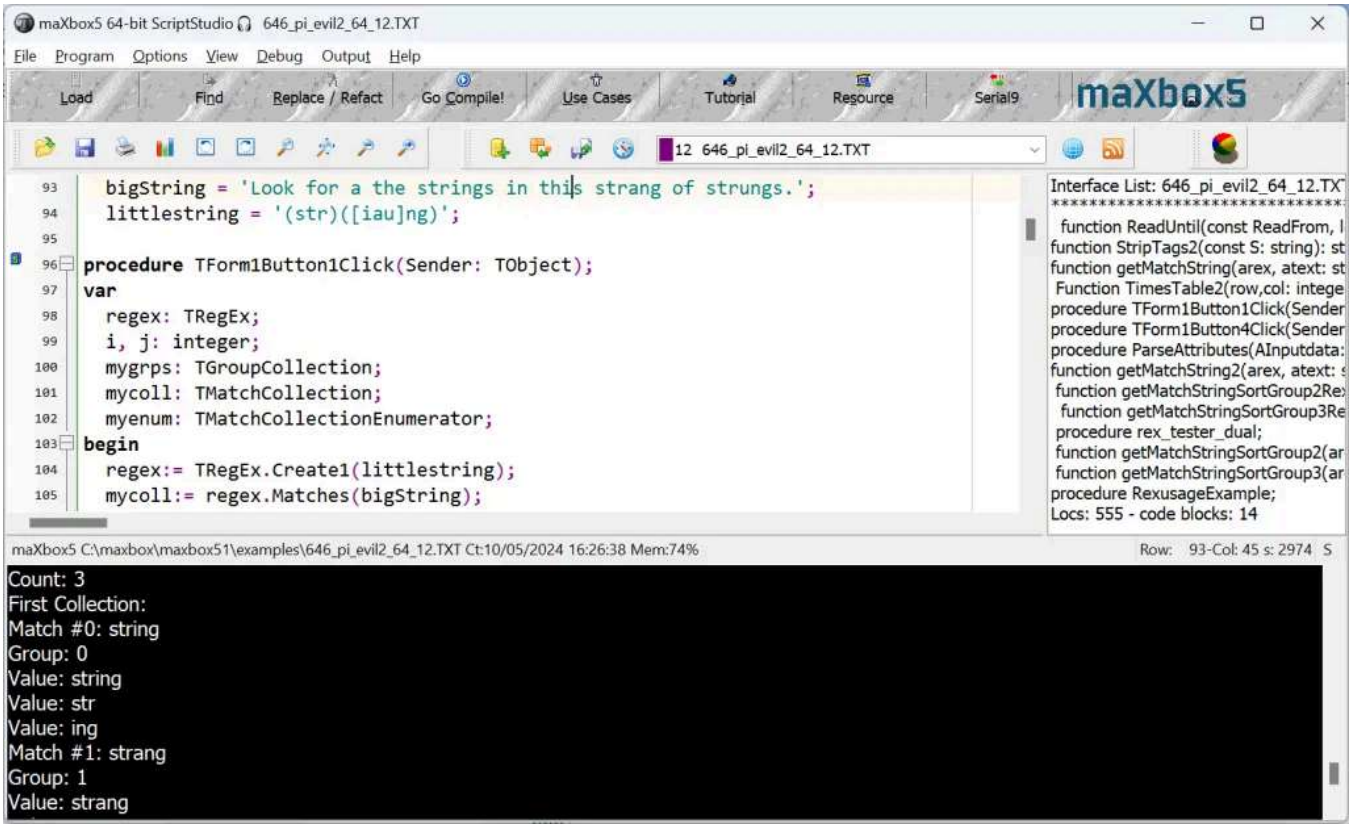
```
var
  Form1: TForm1;
  mycoll: TMatchCollection;
  myenum: TMatchCollectionEnumerator;

implementation

{$R *.dfm}

// Creates and lists the match collection, the matches in that
// collection and the groups in those matches.
procedure TForm1.Button1Click(Sender: TObject);
const
  bigString = 'Look for a the strings in this strang of strungs.';
  littlestring = '(str)([iau]ng)';
var
  regex: TRegEx;
  i, j: integer;
  mygrps: TGroupCollection;
begin
  regex:= TRegEx.Create(littlestring);
  mycoll:= regex.Matches(bigString);
  Edit1.Text:= 'Count: ' + IntToStr(mycoll.Count);
  memo1.Lines.Add('First Collection: ');
  for i:= 0 to mycoll.Count-1 do begin
    memo1.Lines.Add('Match #' + IntToStr(i) + ': ' + mycoll.Item[i].Value);
    memo1.Lines.Add('Group: ' + IntToStr(i));
    mygrps:= mycoll.Item[i].Groups;
    for j:= 0 to mygrps.Count-1 do
```

```
        memo1.Lines.Add('Value: ' + mygrps.Item[j].Value);
    end;
  end;
```



Code as script:
https://sourceforge.net/projects/maxbox/files/Examples/13_General/646_pi_evil2_64_12.TXT/download

The item of a TMatchCollection returns the Match identified by index from the collection (ex. **tmatches[it-1].value**] below).

https://docwiki.embarcadero.com/CodeExamples/Alexandria/en/TMatchCollectionCount_(Delphi)

In general matches from a TRegEx returns all the matches present in the input string an is useful to iterate through a group or captured group:

```
function getMatchString2(arex, atext: string): string; var Match: TMatch; tMatc
```

```
function getMatchString2(arex, atext: string): string;
var  Match: TMatch;  tMatches: TMatchCollection;
       myenum: TMatchCollectionEnumerator;
 begin
   with TRegEx.Create1(arex)  do
   try
     it:= 0;
     { Match format search...}
     result:= result+CRLF;
     if ismatch(atext) then
         tMatches:=Matches(aText);
     writeln('captured groups: '+itoa(tmatches.count ));
       repeat
         Inc(it);
         result:= result+Format(#09'%d: %-12s',[it, tmatches[it-1].value])
         if it mod 5=0 then
           result:= result+#13#10;
       //until match(atext).success;  //MatchNext < 0;
       until it = tmatches.count;
   finally
     Free;
   end;
   WriteLn('Done REX2 - Hit NOthing to exit');
 end;
```

```
PI EXplore5:
1: 33 2: 88 3: 99 4: 44 5: 99
6: 11 7: 66 8: 44 9: 55 10: 22
11: 111 12: 11 13: 555 14: 44 15: 22
16: 44 17: 88 18: 66 19: 33 20: 44
21: 33 22: 66 23: 66 24: 33 25: 00
26: 66 27: 55 28: 88 29: 88 30: 00
31: 11 32: 33 33: 88 34: 66 35: 11
36: 33 37: 11 38: 11 39: 11 40: 44
41: 99 42: 88 43: 22 44: 11 45: 33
46: 33 47: 44 48: 66 49: 22 50: 77
51: 66 52: 000 53: 77 54: 77 55: 77
56: 44 57: 22 58: 22 59: 99 60: 11
61: 44 62: 77 63: 77 64: 99 65: 11
66: 999999 67: 99 68: 44 69: 55 70: 22
71: 33 72: 44 73: 11 74: 88 75: 000
76: 88 77: 33 78: 77 79: 66 80: 55
81: 11 82: 88 83: 77 84: 77 85: 22
86: 66 87: 00 88: 66 89: 111
```

Matches returns all the matches present in the **Input** string in the form of a TMatchCollection instance. If the **Pattern** parameter is not present the regular expression used is specified in the TRegEx constructor.

StartPos specifies the starting position to start the search. TMatchCollection has no public constructor. It is created as the return value of the Matches method. The collection is populated with one TMatch instance for each match found in the input string. The Count property is the length of the TMatchCollection set. **Length** specifies the substring, starting at **StartPos** to match with the regular expressions.

TEE Trio as SNCF CC 6500 Brabant-Mistral-Etendard

Originally published at http://maxbox4.wordpress.com on May 10, 2024.

Regular Expressions | Data Science | Data Analysis

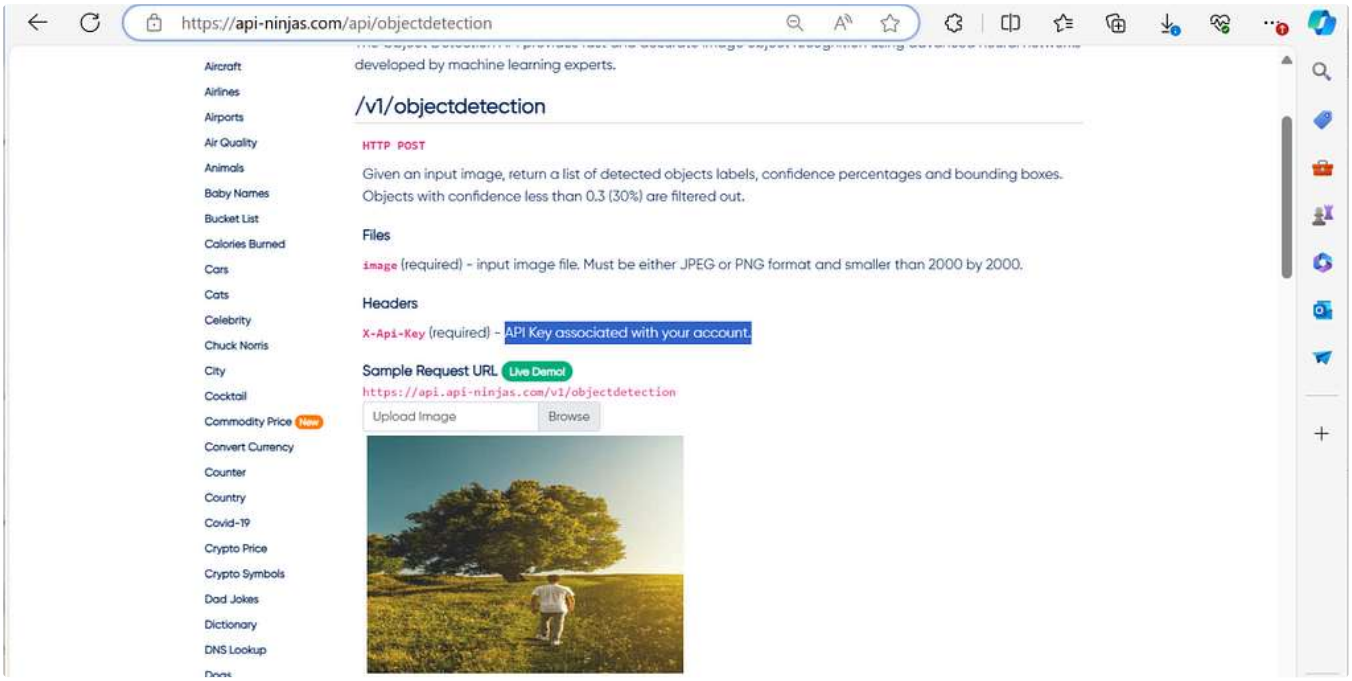
Edit profile

Written by Max Kleiner

27 Followers

Max Kleiner's professional environment is in the areas of OOP, UML and coding - among other things as a trainer, developer and consultant.

More from Max Kleiner



Max Kleiner in Nerd For Tech

Object Detection API

We call it AIM and this stands for Artificial Intelligence Machine.

3 min read · Apr 27, 2024

4

Max Kleiner in Nerd For Tech

Post API Image Pipeline

The Object Detection API (API) provides fast and accurate image object recognition using advanced neural networks developed by machine...

4 min read · Apr 17, 2024

3

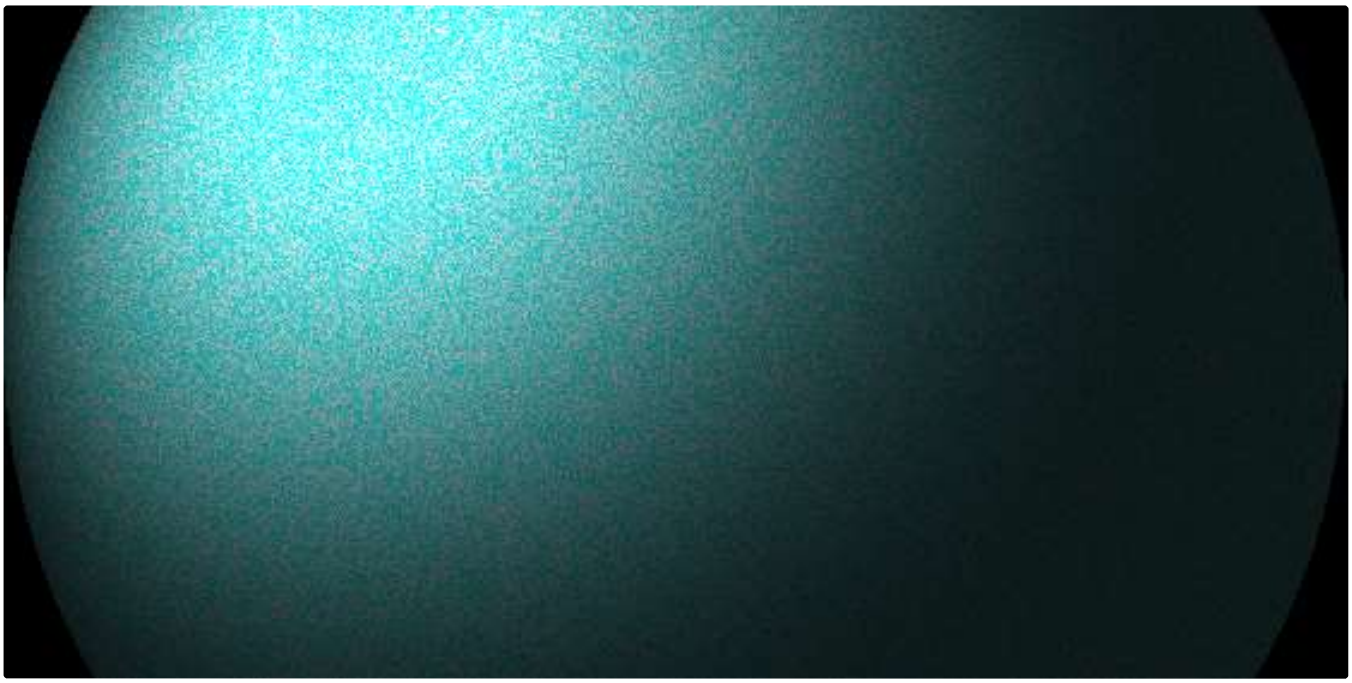
Max Kleiner in Nerd For Tech

OCR with a Neural Net

This API recognizes and reads a text embedded in pictures or photos. Image to Text API uses a neural net (LSTM) based OCR engine which is...

4 min read · Apr 8, 2024

1



Max Kleiner

The Geometer

There's a course about computational geometry which I scripted from the source in maXbox. The origin you can find at:

5 min read · Dec 15, 2020

46 1

See all from Max Kleiner

Recommended from Medium