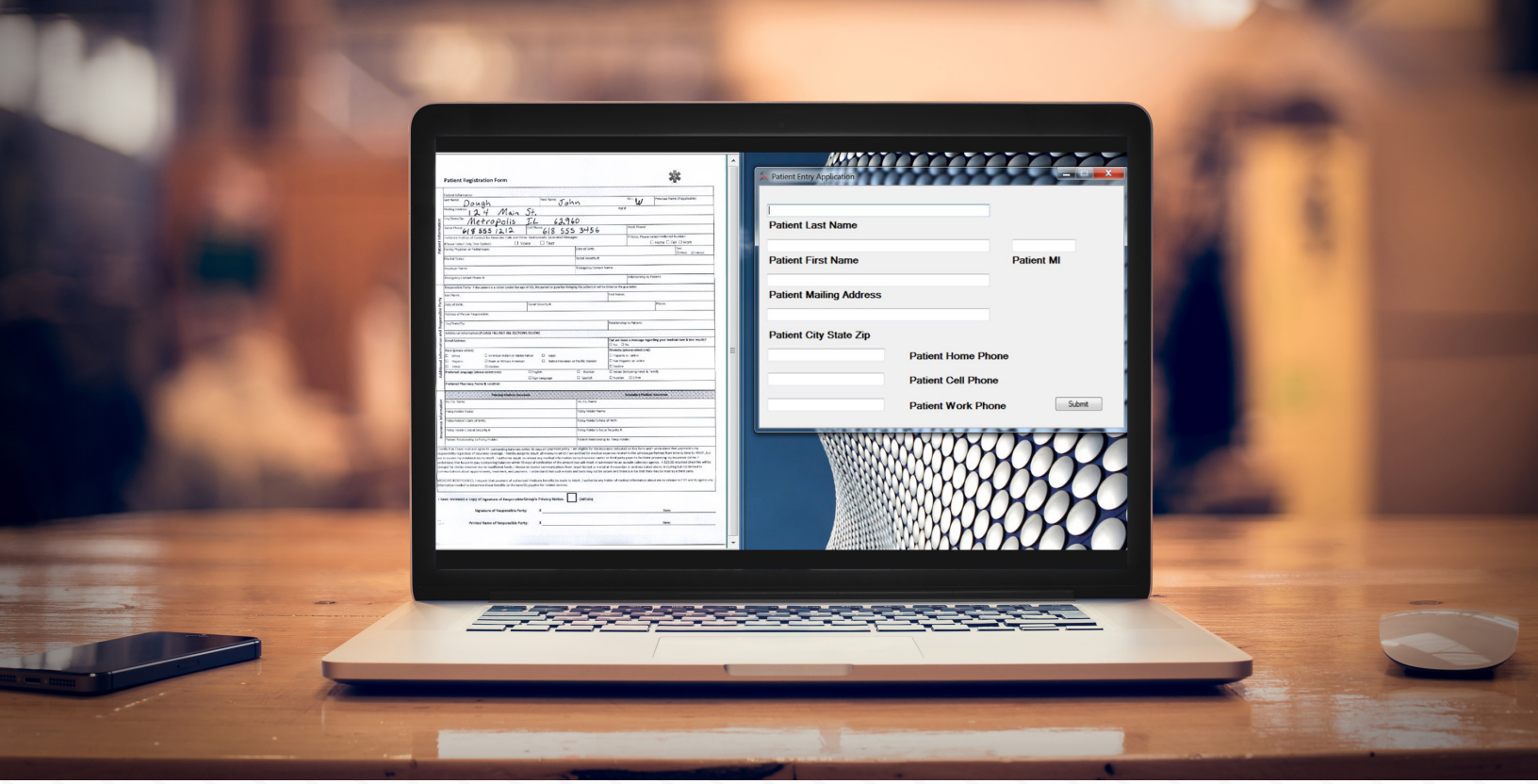
accusoft®

Improve Form Processing Text Recognition Results with Regular Expressions



Successful forms processing requires high accuracy for recognition rates. Developers usually have information about their applications that can be used to improve recognition performance when applied to the recognition process.

One specific approach to improving recognition is to provide specific data formats of the expected results. Knowing expected formats enables the recognition engine to select the best match from among the set of results. This positions the recognition engine to attain higher accuracy and higher confidence in reporting.

Some typical field formats useful in forms processing applications include currency amounts, dates, times, social security numbers, and/or taxpayer ID numbers, phone numbers, email addresses, and URLs.

One example of how this would be used is to apply an expected character pattern format to a date, which immediately eliminates any potential confusion between an “S” and a “5”, or between a “1” and a lower case “L”. It is easy to see the benefits of such an approach. For many of these formats, using regular expressions permits the recognition engine to make assumptions on the number of expected characters to return, which again improves recognition results. You may want to review this information to learn more about regular expressions, including syntax as well as logical applications.

You can learn how to use regular expressions in 30 minutes with Expresso in their tutorial here.

You can learn internals of regular expression engines here.

Regular expressions can become very complex. A simpler regular expression may not catch all valid values or may allow for some invalid values. A complex regular expression could take more time to process, is more prone to errors, and could present maintenance issues.

Without extensive source code comments, a complex regular expression can be difficult for another developer to interpret, especially years after it was written. You will want to balance the complexity and robustness of the expression with the time it will require to process, and the long-term maintenance impact of supporting very complicated regular expressions.

Examples of Regular Expressions

You can use the following to recognize dates in the MM/DD/YYYY format where month and day could be a single digit:

```
“[01]?d/[0123]?d/d{4}”
```

This would recognize all possible valid dates, but would also recognize invalid dates:

15/15/1995 0/34/2010 4/00/2000

Examples of Regular Expressions

To achieve a valid month result, the regular expression would be:

```
“((0?[1-9])|(1[012]))/[0123]?d/d{4}”
```

To achieve a valid day result, between 1 and 31, the regular expression would be:

```
“((0?[1-9])|(1[012]))/((0?[1-9])|([12]d)|(3[01]))/d{4}”
```

This would still recognize some invalid dates, such as:
4/31/2010

To ensure months and days are properly correlated, the regular expression would be:

```
“((((0?[13578])|(1[02]))/((0?[1-9])|([12]d)|(3[01])))|(((0?[469])|(10))/
((0?[1-9])|([12]d)|30))|((0?2)/((0?[1-9])|([12]d))))/d{4}”
```

This still does not limit February to 28 days in non leap years, so the following should be used:

```
“(((0?[13578])|(1[02]))/((0?[1-9])|([12]d)|(3[01])))|(((0?[469])|(10))/
((0?[1-9])|([12]d)|30))|((0?2)/((0?[1-9])|(1d)|(2[0-8]))/d{4})|(2/29/d{2}
(([02468][048])|([13579][26])))”
```

This regular expression has gotten progressively much more complex. This doesn't account for the case when the year is divisible by 100 but not 400, which shouldn't be a leap year (e.g.1900 or 2100).

Further complexity would be required to support date separators other than “/”.

For example, if the date separator characters allow for a dash or a period, you would see the following regular expression:

```
“(((0?[13578])|(1[02]))(/|-|.)((0?[1-9])|([12]
d)|(3[01])))|(((0?[469])|(10)) (/|-|.) ((0?[1-9])|([12]d)|30))|((0?2)
(/|-|.) ((0?[1-9])|(1d)|(2[0-8])) (/|-|.)d{4})|(2(/|-|.)29(/|-|.)d{2}
(([02468][048])|([13579][26])))”
```

Examples of Regular Expressions

If the date separator characters must be the same so that 3.12.2003 is supported but 3.12/2003 is not supported then the regular expression could be:

```
“((((0?[13578])|(1[02]))/((0?[1-9])|([12]
d)|(3[01])))|(((0?[469])|(10))/((0?[1-9])|([12]d)|30))|((0?2)/
((0?[1-9])|(1d)|(2[0-8]))/d{4})|(2/29/d{2}(([02468]
[048])|([13579][26])))|(((0?[13578])|(1[02]))-((0?[1-9])|([12]
d)|(3[01])))|(((0?[469])|(10))-((0?[1-9])|([12]d)|30))|((0?2)-
((0?[1-9])|(1d)|(2[0-8]))-d{4})|(2-29-d{2}(([02468]
[048])|([13579][26])))|(((0?[13578])|(1[02])).((0?[1-9])|([12]
d)|(3[01])))|(((0?[469])|(10)).((0?[1-9])|([12]d)|30))|((0?2).
((0?[1-9])|(1d)|(2[0-8])).d{4})|(2.29.d{2}(([02468][048])|([13579]
[26])))))”
```

As you can see, regular expressions can get very long and complex. There must be a tradeoff between a simpler, easy to understand regular expression, and a complex regular expression that more specifically describes the expected data contents.

The Use of Iterative Regular Expressions

One approach to reducing complexity in each regular expression is to in effect "stack" the application of regular expressions to meet desired results. One could parse the string to gain enough information to route the logic through its appropriate next validation layer (regular expression). In the example above, we could parse the separator characters, and then craft the regular expression appropriately for each expected separator value.

Here is a simple date regular expression with multiple separator characters:

Step one: Simple date expression

```
"((0?[1-9])|(1[012]))(/|-|.)((0?[1-9])|([12]d)|(3[01]))(/|-|.)d{4}"
```

We now know that we meet the general format of a date, and the separator characters provided are acceptable separator characters. To test that all separator characters are the same value, i.e. all slash or all hyphen, we could use this regular expression:

Step two: Date separator consistency expression

```
"(d*/d*/d*)|(d*-d*-d*)|(d*.d*.d*)"
```

We now know that the separator characters are the same value throughout the date.
If a string successfully passes these the simple date expression and the date separator consistency expression, then we know it is in the general format of a date, it has acceptable separator characters, and the separator characters are the same values. Note: We have not yet fully validated the dates.

To validate the month, day, and year we could use:

Step three: Date validity expression

```
"(((0?[13578])|(1[02]))(/|-|.)((0?[1-9])|([12]
d)|(3[01])))|(((0?[469])|(10)) (/|-|.) ((0?[1-9])|([12]d)|30))|((0?2)
(/|-|.) ((0?[1-9])|(1d)|(2[0-8])) (/|-|.)d{4})|(2(/|-|.)29(/|-|.)d{2}
(([02468][048])|([13579][26])))"
```

We now know that the digits provided produce a valid date.

By using a combination of simpler regular expressions, we can achieve the goal of precisely validating, or as close as possible, our data while still having regular expressions that can be maintained.

Using Regular Expressions During Recognition

In forms processing, where the recognition accuracy rate can be critical for success, it is beneficial if the evaluation of the regular expression is integrated into the recognition process. Providing the expected data format to the recognition engine permits the recognition system to use that information to better select from multiple possible results.

You can use an OCR or ICR SDK when attempting to generate the most accurate result. In any OCR or ICR processing there can always be alternate values for any single character. Many times these alternative values are the correct result. The difficulty is in knowing which alternate value is best.

Many OCR recognition engines use language dictionaries to assist in selecting the results that make the most "sense". When processing forms, the data contents are generally in a specialized format and dictionary results may not apply. Additionally, some specialized formats such as currency amounts, dates, times, social security numbers, phone numbers, email addresses, and URLs appear on many forms and can be defined by regular expressions. Forms frequently use fields such as part numbers or serial numbers that require a specific format, and can be defined by a regular expression. Integrating the regular expression into the recognition system logic provides the best possible accuracy result.

Accusoft offers SmartZone that supports the integration of regular expressions directly into the recognition engine, improving forms processing recognition results. Additionally, these SDKs offer predefined, expected formats for currency amounts, dates, times, social security numbers and/or taxpayer ID numbers, phone numbers, email addresses, and URLs.

Using Regular Expressions During Recognition

Developers can also augment the pre-defined formats by using their own custom regular expressions, designed to support their specific formatting requirements. This creates a powerful combination of general format requirements enhanced by formatting information and available only with an intimate knowledge of expected data values. SmartZone uses a regular expression engine integrated into the recognition engine to achieve the best possible accuracy on data that can be defined by a regular expression.

Download the SmartZone ICR/OCR SDK and experiment with regular expressions using your own forms.