dtSearch Regular Expressions

In the AccessData Forensic Toolkit, regular expression searching capabilities has been incorporated in the dtSearch index search tab. This functionality does not use RegEx++ that we are accustomed to in the Live Search tab. dtSearch utilizes the TR1 (Technical Report 1) regular expressions.

Regular expressions in dtSearch provide a powerful syntax for searching for complicated patterns in text, such as one of several possible sequences of letters followed by a sequence of numbers. Regular expressions can also be used to express spelling variations of individual words. Regular expression patterns are arbitrary (i.e., supplied by the user dynamically) and cannot be pre-indexed.

Regular expression searching in dtSearch is limited to a single whole word. A regular expression included in the dtSearch box must be quoted and must begin with ##. An example of this is:

Apple and "##199[0-9]" – will find Apple and 1990 through 1999
Apple and "##19[0-9]+" – will find Apple and 190 through 199

However, if you wanted to look for Apple Pie, you could not use "##app.*ie" since this is two words. Only letters and numbers are searchable. You cannot search for any of the non-indexed characters as defined in the Index Search Settings in the Detailed Options section of a case creation. Also, dtSearch does not store information about line breaks so any searches that are made that include the beginning of a line or the end of a line will not work.

Search considerations using the wildcard character "*" in a regular expression does have an effect on search speed: the closer to the front of a word the expression is, the more it will slow searching. "Appl.*" will be nearly as fast as "Apple", while ".*pple" will be much slower.

NOTE: Advanced searching for Social Security Numbers and Credit Card Numbers and other number patterns can be achieved; however modifications to the dtSearch engine must be made before processing the case. For more details, see Advanced Searching on page 7 of this paper.

Element Terms

Characters and target sequences are referred to as an *Element* and can be one of the following:

- A literal character typed as the actual letter or number (a or 1).
- A '.' (period) is any single character.
- An '*' (asterisk) is a wildcard character.
- (a) is a capture group.
- \d is a decimal character.
- For hex searches, \xhh matches a hex entry (ie – \x0f).
- {2} is a repetition character.
- A ',' (comma) is a minimum character.
- (aa?) is a target sequence.
- An alternation character search is 'this|that'.
- A concatenation sequence is '(a){2,3}(b){2,3}(c)'.
- A back reference is '((a+)(b+))(c+)\3'.
- (?:subexpression) matches the sequence of characters in the target sequence that is matched by the patter between the delimiters.
- (?!:sub-expression) matches any sequence of characters in the target sequence that does not match the pattern listed in the sub-expression)
- A bracket or range expression of the form "[expr]", which matches a value or a range, similar to a "set" in the Live Pattern Search.

Examples:

- "##a" matches the target sequence "a" but does not match the target sequences "b", or "c"…etc.
- "##." matches a single character such as "a", "b", and "c"…etc.
- "##sal*" matches the target 'sale' and the target "salt' and so on.
- "##(a)" capture group, matches the target sequence "a" but does not match the target sequences "b", or "c"…etc.
- "##\d\d\d\d" matches the target sequence of four digits "1234".
- "##aa?" or {0,1} matches the target sequence of "aa" and the target sequence of "aaa".
- "##ab" matches the target sequence "ab."
- "##[b-z]" or range, matches the target sequences "b" and "c" but does not match the target sequences "a".
- "##tom|jerry" matches the target sequence of 'tom' or 'jerry'.
- "##\d{4}" or repetition, matches the target sequence of four digits "1234".
- "##(?:aa)" or target sequence, matches the target sequence of "aa" and the target sequence of "aaa", and so on.

Ordinary Character

By entering actual ASCII characters, the search will return that set of characters after the element(s) are entered. By entering ordinary characters, “##nick”, you would find said characters. However, if you wanted to look for Nick Drehel, you could not use “##nick drehel” since this is two words.

Single “Any” Character and Wildcard

The use of the any character element can be used if a letter or letters may be different, such as difference in spelling (example ‘marijuana’ and ‘marihuana’). The wildcard is used to find any combination of characters after an element is entered.

Examples:

- "##(a*)" matches the target sequence "a", the target sequence "aa", and so on.
- "##a*" matches the target sequence "a", the target sequence "aa", and so on.
- "##(a.)" matches the target sequence "aa", the target sequence "ab", but will not find the target sequence the target sequence “aaa”.
- "##a." matches the target sequence "aa", the target sequence "ab", but will not find the target sequence the target sequence “aaa”.
- “##.*ick” matches the target sequence “nick”, the target sequence “click”, and so on.
- “##mari.uana” matches the target sequence “marijuana” and the target sequence “marihuana”.

Capture Group

A capture group marks its contents as a single unit in the regular expression and labels the target text that matches its contents. The label that is associated with each capture group is a number, which is determined by counting the opening parentheses that mark capture groups up to and including the opening parenthesis.

Example:

- “##(ab)*” matches the target sequence “ab”, the target sequence ‘abab”, and so on.
- “##(a+)(b+)” matches the target sequence “ab, the target sequence “aab”, the target sequence “abb”, and so on.
- “##ab+” matches the target sequence “abb” but does not match the target sequence “abab.”
- “##(ab)+” matches the target sequence “abab” but does not match the target sequence “abb.”
- “##((a+)(b+))(c+)" matches the target sequence "aabbbc" and associates capture group 1 with the subsequence "aabbb", capture group 2 with the subsequence "aa", capture group 3 with "bbb", and capture group 4 with the subsequence "c".

Repetition

Any element can be followed by a repetition count.

Examples:

- "##(a{2})" matches the target sequence "aa" but not the target sequence "a" or the target sequence "aaa".
- "##(a{2,})" matches the target sequence "aa", the target sequence "aaa", and so on, but does not match the target sequence "a".

A repetition count can also take the following form:

- "?" - Equivalent to "{0,1}".

Examples:

- "a?" matches the target sequence "" and the target sequence "a", but not the target sequence "aa".
- “##(aa?)(bbbb?)(c)” matches the target sequence “aabbbbc” and the target sequence “abbbc”.

Decimal Character

You can locate any set of decimals by using the ‘\d” character element in the expression.

Examples:

- "##\d\d\d\d" matches the target sequence "1234".
- “##\d[3} matches the target sequence “123”.
- “##\d{3}\d\d\d\d” matches the target sequence ‘1234567”.
- Visa and “##\d{4}” will match any files that contain the word ‘visa’ and any four digits.

Alternation

A concatenated regular expression can be followed by the character '|' and another concatenated regular expression. Any number of concatenated regular expressions can be combined in this manner. The resulting expression matches any target sequence that matches one or more of the concatenated regular expressions.

Example:

- "##(nick|houston)" matches the target sequence "nick", or the target sequence “houston”.

Concatenation

Regular expression elements, with or without *repetition counts*, can be concatenated to form longer regular expressions. The resulting expression matches a target sequence that is a concatenation of the sequences that are matched by the individual elements.

Examples:

- "##(a){2,3}(b){2,3}(c)" matches the target sequence "aabbc", the target sequence “aaabbbc”.
- “##(\d{4}){4}” matches the target sequence of “1234123412341234” (16 digits – no spaces).

Back Reference

A back reference marks its contents as a single unit in the regular expression grammar and labels the target text that matches its contents. The label that is associated with each capture group is a number, which is determined by counting the opening parentheses that mark capture groups up to and including the opening parenthesis that marks the current capture group. A back reference is a backslash that is followed by a decimal value N. It matches the contents of the Nth *capture group*. The value of N must not be more than the number of capture groups that precede the back reference.

Example:

- "((a+)(b+))(c+)\3" matches the target sequence "aabbbcbbb". The back reference "\3" matches the text in the third capture group, that is, the "(b+)". It does not match the target sequence "aabbbcbb".
    - The first capture group is ((a+)(b+))
    - The second capture group is (a+)
    - The third capture group is (b+)
    - The fourth capture group is (c+)

Bracket or Character Range

A character range in a bracket expression adds all the characters in the range to the character set that is defined by the bracket expression. To create a character range, put the character '-' between the first and last characters in the range. Doing this puts into the set all characters that have a numeric value that is more than or equal to the numeric value of the first character, and less than or equal to the numeric value of the last character.

Examples:

- "[0-7]" represents the set of characters { '0', '1', '2', '3', '4', '5', '6', '7' }. It matches the target sequences "0", "1", and so on, but not "a".
- "[h-k]" represents the set of characters { 'h', 'i', 'j', 'k' }.
- "[0-24]" represents the set of characters {'0', '1', '2', '4' }.
- "[0-2]" represents the set of characters { '0', '1', '2' }.

An individual character in a bracket expression adds that character to the character set that is defined by the expression. If the bracket expression begins with a “^” then this defines that the expression will consider all characters except for those listed.

Examples:

- "[abc]" matches the target sequences "a", "b", or "c", but not the sequence "d".
- "[^abc]" matches the target sequence "d", but not the target sequences "a", "b", or "c".
- "[a^bc]" matches the target sequences "a", "b", "c", or "^", but not the target sequence "d".

Advanced Searching - TR1 Regular Expressions for Number Patterns

If order to achieve dtSearch capability in FTK for search strings such as Social Security Numbers, Credit Card Numbers, Employee Identification Numbers, Telephone Numbers, and etc. where a period or hyphen is present. Certain steps must be done during the pre-processing phase of the case.

NOTE: *Currently, FTK cannot include search patterns with spaces.*

Normal dtSearch strings for credit card numbers or social security numbers

The normal dtSearch wildcard string can be utilized as long as the hyphen is set to be indexed as a space:

Social Security Numbers - === == ====
Returns "123-45-6789"
Will not return "123 45 6789"

Credit Card Numbers (16 digits) - ==== ==== ==== ====
Returns "1234-1234-1234-1234"
Will not return "1234 1234 1234 1234"

Number Patterns

dtSearch TR1 Regular Expression for find number patterns as we do in Live Searches for such things as Credit Card Numbers, Social Security Numbers, xxxxxxxx. Certain pre-processing options **MUST** be completed by the examiner before this function will work.

Pre-Processing Options

If the examiner wants to utilize the dtSearch TR1 Regular Expression functions for looking for number patterns, they must complete the following pre-processing options:

1. Start a new case
2. Select **Detailed Options**
3. Click **Indexing Options** next to dtSearch Text Index
4. On the Indexing Options dialog window
   - For Hyphen Treatments – set to **Hyphen**
   - In the Spaces section – **remove the period**
   - In the Spaces section **– remove the left and right parenthesis**
   - In the Letters section – click **Add** and in all 4 spaces, type a "**.**" period and the left and right parenthesis, then click **OK**
5. Process the case

Examples of TR1 Regular Expressions for Number Patterns

For Credit Card Numbers -

"##(\d{4}[\.\-])(\d{4}[\.\-])(\d{4}[\.\-])(\d{4})"

The first three groups are composed of – (\d{4}[\s\.\-]). The expression is looking for four digits followed by a space, period, or hyphen. This group is repeated three times and followed by the group looking for the ending 4 digits.

We can shorten that expression by writing it "##(\d{4})[\s\.\-]){3}(\d{4})".

This will find 1234-5678-1234-5678 or 1234.5678.1234.5678.

For Social Security Numbers –

"##(\d{3}[\.\-])(\d{2}[\.\-])(\d{4})".

This will find 123-45-6789 or 123.45.6789.

For U.S. Telephone Numbers –

"##(\d[\-\.])?(\(?\d{3}[\-\.\)])?([\-\.]?\d{3}[\-\.])(\d{4})"

This will find:

- 567-8901
- 234-567-8901
- 1-234-567-8901
- (234)567-8901
- (234)-567-8901
- 567.8901
- 234.567.8901
- 1.234.567.8901
- (234)567.8901
- (234).567.8901

References

Regular Expressions – dtSearch Support.
http://support.dtsearch.com/webhelp/dtsearch/regular_.htm

MSDN: TR1 Regular Expressions. http://msdn.microsoft.com/en-us/library/bb982727.aspx