

Comparison of Regular Expression Standard Syntax & Microsoft's CAtlRegEx class supported Syntax.

There are many implementations of Regular Expression available to developers and users alike. Technology Pathways is implementing Microsoft's CAtlRegEx class when adding regular expression capabilities to ProDiscover version 6.0. The following chart provides a comparison of many common regular expression syntax formats to the support found in Microsoft's CAtlRegEx class and ProDiscover's implementation.

Character	Description	Example	Supported By CAtlRegEx Class
Any character except <code>[^\\$. ?*\+()</code>	All characters except the listed special characters match a single instance of themselves. { and } are literal characters, unless they're part of a valid regular expression token (e.g. the {n} quantifier).	a matches a	Yes
<code>\</code> (backslash) followed by any of <code>[^\\$. ?*\+()</code> {	A backslash escapes special characters to suppress their special meaning.	<code>\+</code> matches +	Yes
<code>\Q...\E</code>	Matches the characters between <code>\Q</code> and <code>\E</code> literally, suppressing the meaning of special characters.	<code>\Q+-*\E</code> matches <code>+-*</code>	No
<code>\xFF</code> where FF are 2 hexadecimal digits	Matches the character with the specified ASCII/ANSI value, which depends on the code page used. Can be used in character classes.	<code>\xA9</code> matches © when using the Latin-1 code page.	No
<code>\n</code> , <code>\r</code> and <code>\t</code>	Match an LF character, CR character and a tab character respectively. Can be used in character classes.	<code>\r\n</code> matches a DOS/Windows CRLF line break.	Only <code>\n</code>
<code>\a</code> , <code>\e</code> , <code>\f</code> and <code>\v</code>	Match a bell character (<code>\x07</code>), escape character (<code>\x1B</code>), form feed (<code>\x0C</code>) and vertical tab (<code>\x0B</code>)		Only <code>\a</code>

	respectively. Can be used in character classes.		
\cA through \cZ	Match an ASCII character Control+A through Control+Z, equivalent to \x01 through \x1A. Can be used in character classes.	\cM\cJ matches a DOS/Windows CRLF line break.	No
[(opening square bracket)	Starts a character class. A character class matches a single character out of all the possibilities offered by the character class. Inside a character class, different rules apply. The rules in this section are only valid inside character classes. The rules outside this section are not valid in character classes, except \n, \r, \t and \xFF		Yes
Any character except ^-]\ add that character to the possible matches for the character class.	All characters except the listed special characters.	[abc] matches a, b or c	Yes
\ (backslash) followed by any of ^-]\	A backslash escapes special characters to suppress their special meaning.	[\^] matches ^ or]	Yes
- (hyphen) except immediately after the opening [Specifies a range of characters. (Specifies a hyphen if placed immediately after the opening [)	[a-zA-Z0-9] matches any letter or digit	Yes
^ (caret) immediately after the opening [Negates the character class, causing it to match a single character <i>not</i> listed in the character class. (Specifies a caret if placed anywhere except after the opening [)	[^a-d] matches x (any character except a, b, c or d)	Yes
\d, \w and \s	Shorthand character classes matching digits, word characters, and whitespace. Can be used inside and outside character classes.	[\d\s] matches a character that is a digit or whitespace	Only \d, \w
\D, \W and \S	Negated versions of the above.	\D matches a	No

	Should be used only outside character classes. (Can be used inside, but that is confusing.)	character that is not a digit	
[b]	Inside a character class, \b is a backspace character.	[\b\t] matches a backspace or tab character	Yes
.	Matches any single character except line break characters \r and \n. Most regex flavors have an option to make the dot match line break characters too.	. matches x or (almost) any other character	Yes
^ (caret)	Matches at the start of the string the regex pattern is applied to. Matches a position rather than a character. Most regex flavors have an option to make the caret match after line breaks (i.e. at the start of a line in a file) as well.	^. matches a in abc\ndef. Also matches d in "multi-line" mode.	No
\$ (dollar)	Matches at the end of the string the regex pattern is applied to. Matches a position rather than a character. Most regex flavors have an option to make the dollar match before line breaks (i.e. at the end of a line in a file) as well. Also matches before the very last line break if the string ends with a line break.	.\$ matches f in abc\ndef. Also matches c in "multi-line" mode.	No
\A	Matches at the start of the string the regex pattern is applied to. Matches a position rather than a character. Never matches after line breaks.	\A. matches a in abc	No
\Z	Matches at the end of the string the regex pattern is applied to. Matches a position rather than a character. Never matches before line breaks, except for the very last line break if the string ends with a line break.	.\Z matches f in abc\ndef	No
\z	Matches at the end of the string	.\z matches f in	No

	the regex pattern is applied to. Matches a position rather than a character. Never matches before line breaks.	abc\ndef	
\b	Matches at the position between a word character (anything matched by \w) and a non-word character (anything matched by [^\w] or \W) as well as at the start and/or end of the string if the first and/or last characters in the string are word characters.	.\b matches c in abc	No
\B	Matches at the position between two word characters (i.e the position between \w\w) as well as at the position between two non-word characters (i.e. \W\W).	\B.\B matches b in abc	No
(pipe)	Causes the regex engine to match either the part on the left side, or the part on the right side. Can be strung together into a series of options.	abc def xyz matches abc, def or xyz	Yes
? (question mark)	Makes the preceding item optional. Greedy, so the optional item is included in the match if possible.	abc? matches ab or abc	Yes
??	Makes the preceding item optional. Lazy, so the optional item is excluded in the match if possible. This construct is often excluded from documentation because of its limited use.	abc?? matches ab or abc	Yes
* (star)	Repeats the previous item zero or more times. Greedy, so as many items as possible will be matched before trying permutations with less matches of the preceding item, up to the point where the preceding item is not matched at all.	".*" matches "def" "ghi" in abc "def" "ghi" jkl	Yes
? (lazy star)	Repeats the previous item zero or	".?" matches	Yes

	more times. Lazy, so the engine first attempts to skip the previous item, before trying permutations with ever increasing matches of the preceding item.	"def" in abc "def" "ghi" jkl	
+ (plus)	Repeats the previous item once or more. Greedy, so as many items as possible will be matched before trying permutations with less matches of the preceding item, up to the point where the preceding item is matched only once.	".+" matches "def" "ghi" in abc "def" "ghi" jkl	Yes
+? (lazy plus)	Repeats the previous item once or more. Lazy, so the engine first matches the previous item only once, before trying permutations with ever increasing matches of the preceding item.	".+?" matches "def" in abc "def" "ghi" jkl	Yes
{n} where n is an integer ≥ 1	Repeats the previous item exactly n times.	a{3} matches aaa	No
{n,m} where $n \geq 0$ and $m \geq n$	Repeats the previous item between n and m times. Greedy, so repeating m times is tried before reducing the repetition to n times.	a{2,4} matches aaaa, aaa or aa	No
{n,m}? where $n \geq 0$ and $m \geq n$	Repeats the previous item between n and m times. Lazy, so repeating n times is tried before increasing the repetition to m times.	a{2,4}? matches aa, aaa or aaaa	No
{n,} where $n \geq 0$	Repeats the previous item at least n times. Greedy, so as many items as possible will be matched before trying permutations with less matches of the preceding item, up to the point where the preceding item is matched only n times.	a{2,} matches aaaaa in aaaaa	No
{n,}? where $n \geq 0$	Repeats the previous item n or more times. Lazy, so the engine first matches the previous item n times, before trying permutations with ever increasing matches of the preceding item.	a{2,}? matches aa in aaaaa	No

As with any language there are many ways to express the same thought with Regular Expressions and thus several “Expressions” can be formed with the same search goal in mind. The following examples are simple approaches to finding commonly desired data and may return false positives along with the desired results. With some experience investigators may find ways to *fine-tune* the examples provided.

Syntax Examples

To search for 10-digit phone number like "(800) 325-3535" or "650 555 1212", use:

```
(?\d\d\d) ]\b?\d\d\d[-]\d\d\d\d
```

To search for a HTML Href's, use:

```
href\b*=\b*\q
```

To search for US Social Security Numbers (SSN), use:

```
\d\d\d[-]\d\d[-]\d\d\d\d
```

To search for IPv4 dotted decimal notation addresses, use:

```
{[0-9]+[.][0-9]+[.][0-9]+[.][0-9]+}
```

To search for email address, use:

```
{[0-9a-z_\.\-]+}@{[0-9a-z_\.\-]+}
```

To search for a 4 digit time value, use:

```
{\d\d:\d\d [ap]m} or {\d\d:\d\d:\d\d [ap]m} for 6 digits
```

A handy tool for creating and testing expressions conforming to the Microsoft standard is called appropriately “MfcRegex.exe” and can be found online at <http://www.codeproject.com/KB/string/mfcregex.aspx>

Sources: <http://www.regular-expressions.info/refadv.html> for standard Regular Expression Syntax. MSDN help for its supported Regular Expression Syntax.