InDesign MAGAZINE 59
March 2014

SPECIAL REPRINT!

GET A GRIP ON

PLUS:

PRESENTING WITH INDESIGN

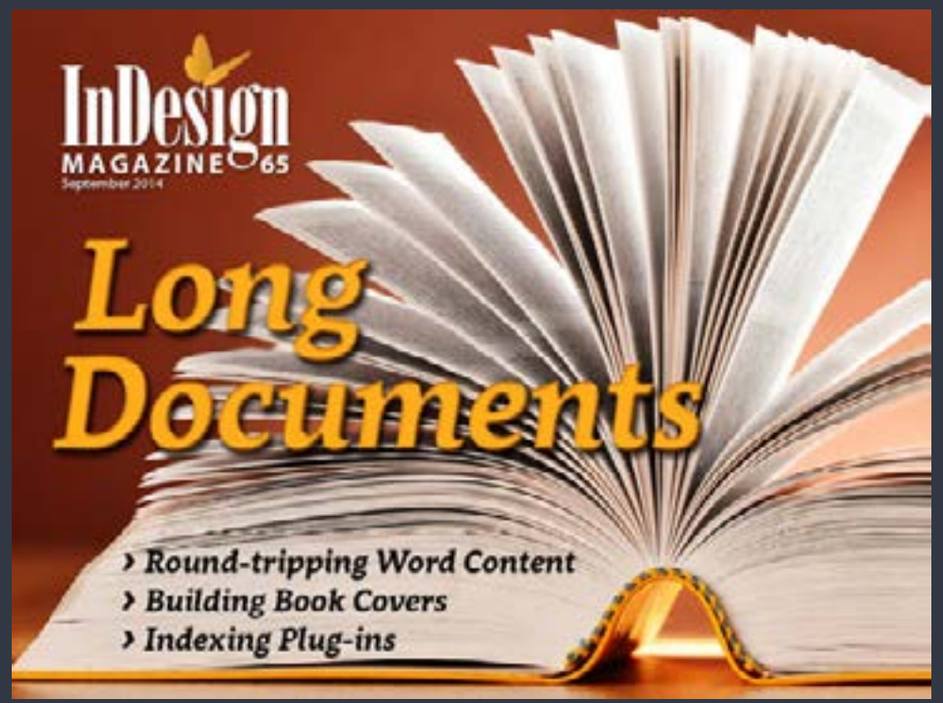
INREVIEW—TIMETRACKER

BEST OF THE BLOG

Become a Member of InDesignSecrets!

Premium members receive *InDesign Magazine*, plus many other great benefits!

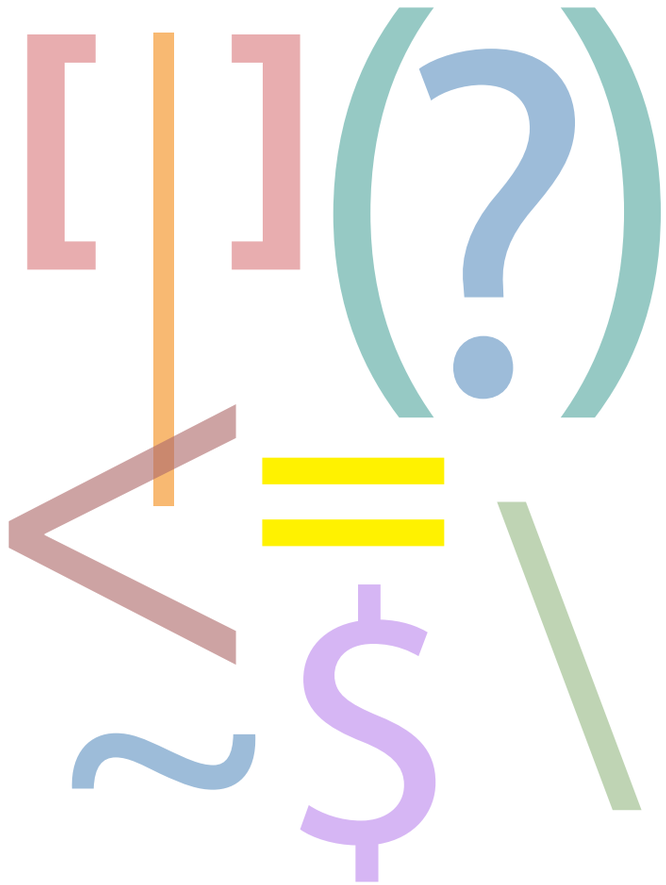
Visit indesignsecrets.com/membership for all the great reasons to join. Use the discount code WORKFLOW to get $10 off!



By Peter Kahrel

Getting a Grip on GREP

You've heard about it. Now here's your chance to learn it! (Don't worry, it sounds harder than it is.)



Most people use InDesign the way they cook dinner: they know the basics, they can intuitively throw together some ingredients, or they can follow a basic recipe. But it turns out that if you know a little bit about food science, the chemistry behind the process, you can achieve wonders and impress both yourself and everyone around you. Similarly, by learning a little bit about some geeky stuff, you can do wonders that others can only dream of. If you're looking for a way to supercharge your InDesign skills, there are few things as good as learning a bit of GREP.

Unfortunately, GREP looks scary, and so people think it *is* scary. Not so! Like so many things that are reputedly difficult, GREP can be readily understood and used successfully at basic levels. First, let's define what it is and what it's good for. GREP is a search tool that you can employ just like the search tools in applications such as MS Word and Notepad to find literal text like *nonplussed* or *understand*. But you can make GREP searches more interesting and more useful by adding certain codes to search for text *patterns* instead of literal text. With GREP's codes, you can do things like "find all words that consist of capital letters," "find all words that end in *ful*," "apply No Break to the last word of all paragraphs," and countless others.

In this article I'll show you how to formulate GREP search patterns that are much more powerful and useful than InDesign's standard search-and-replace

tools. I won't go into every aspect of GREP, but you can find some fantastic resources at InDesignSecrets.com/grep to fill in the gaps in your understanding of the basics of GREP. And at the end of the article I'll point you to other resources with more advanced techniques.

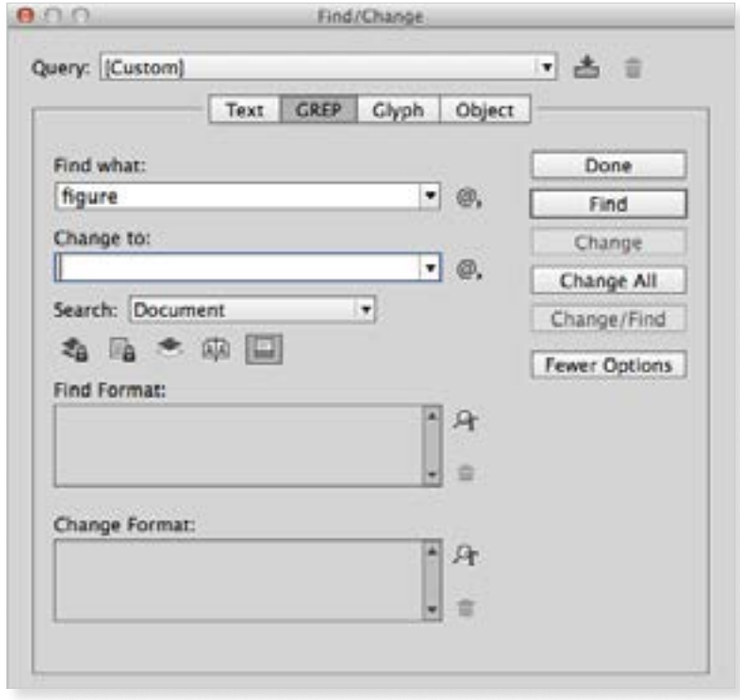
Finding Alternate Spellings

As I mentioned earlier, GREP is good at finding patterns, so it's perfect for finding different spellings of the same word. We find spelling differences in variants of the same language, such as British English and American English, but consistent spelling errors, too, are in fact cases of spelling "variation."

Examples of spelling differences in British and American English are *grey–gray*, *centre–center*, and *colour–color*. Notice that these three examples show three different types of variation, each of which can be found with a different GREP search. We'll deal with them one by one.

The Find/Change Window

GREP shows up in two places in InDesign: the Find/Change dialog box and the GREP Styles feature (typically used inside a paragraph style definition). The former lets you manually find text patterns and either apply formatting to them, or change the text to something else. The latter lets you tell InDesign to automatically search for text patterns and apply formatting to them. GREP Styles cannot change text; they can only apply character formatting (in the form of a character style) to the text. For the sake of this article, I will discuss only GREP inside the Find/Change dialog box. (For more on GREP and GREP Styles, see 5 Cool Things You Can Do with GREP Styles later in this issue.)



To open the Find/Change dialog box, press Ctrl/Cmd+F (or choose **Edit > Find/Change**), then select the GREP tab (or press Ctrl/Cmd+2). The Find What field is where you enter what you want to search for. For example, to look for the word *figure*, enter `figure` in the Find What field, and click Find to start the search. If the word occurs in the document, InDesign highlights it. There, that wasn't so hard, was it?

The bracket: "match any ONE of these"

The first one, *grey–gray*, we can find typing the following into the GREP Find What field:

```
gr[ea]y
```

You can read `[ea]` as "e *or* a." A bracketed string matches just *one* character in the text, and you can place as many characters in a bracketed string as you like. For example, `b[aeiou]t` can be read as "b, followed by a vowel, followed by t," and will find *bat*, *bet*, *bit*, *bot*, and *but*. It will NOT find *bait*.

The pipe: "this or that"

Because the second type of variation, *centre–center*, involves two characters, it's not convenient to use the bracketed notation. Instead, we list alternatives like this:

```
cent(re|er)
```

Here, the alternatives are grouped in parentheses and separated by a pipe `(|)`. Alternatives don't have to be the same length. For example, `thr(u|ough)` matches *thru* and *through*, and `(X|Christ)mas` matches *Xmas* and *Christmas*. And the alternatives can consist of any character; they don't have to be letters. Thus, `who(se|'s)` matches *whose* and *who's*. And finally, you can list any number of alternatives, as in the following search term:

```
(pro|de|pre)scribe
```

which matches *proscribe*, *describe*, and *prescribe*.

Apart from searching alternate spellings of the same word, you can use the pipe notation to find different words altogether. For example,

```
perhaps|maybe
```

finds both *perhaps* and *maybe*. Note that in this case it's not necessary to add parentheses. In the earlier example of `cent(re|er)`, the parentheses were needed to isolate the alternatives from the main part of the word, *cent*.

The question mark: "there or not"

For the third type of variation, *colour–color*, you use yet a different method. This variation is determined by the presence or absence of a letter, here *u*. In GREP-speak we indicate the possible presence of a character by adding a question mark after it:

```
colou?r
```

The question mark can apply to any character; it doesn't have to be a letter. So `it'?s` matches *its* and *it's*.

The scope of `?` is just one preceding character—in other words, only the character immediately to the left of the question mark is optional—so that the search `colou?r` matches both *color* and *colour*. To make more characters optional—say, a prefix or a suffix—you place them in parentheses. Thus, to find the words *cop* and *copper*, you use this search term:

```
cop(per)?
```

Combinations

The three different methods to find alternate spelling can be combined. For example, to find *setup*, *set-up*, and *set up*, you could use the following search term:

```
set[- ]?up
```

which you can read as "*set*, possibly followed by a hyphen or a space, followed by *up*."

Alternatives can be made optional, too. For example, to find the word *claim* and its inflections, use this search pattern:

```
claim(s|ed|ing)?
```

which matches *claim*, *claims*, *claimed*, and *claiming*. By placing `?` after `(s|ed|ing)`, the whole list of alternatives is made optional. If you leave out `?`, *claim* is not found—only the three inflected forms are.

GREP is case-sensitive

You may have noticed that GREP searches are case-sensitive. For example, the search term *color* doesn't match *Color*. GREP searches, then, are case-sensitive by default. This is not a limitation of InDesign's version of GREP, by the way; it's a standard feature of GREP. In fact, it's not a limitation at all: it actually makes using case-sensitivity much more flexible, something which we'll return to later.

There are several ways to make GREP case-*in*sensitive, but for the moment let's look at just one:

```
[Cc]olor
```

As you see, this comes down to the "alternative spelling" approach we outlined earlier: in essence, we're saying that *Color* and *color* are alternative spellings of the same word, which is equivalent to setting a case-insensitive option.

Finding series of characters

Earlier, we mentioned this search term:

```
b[aeiou]t
```

and that it can be read as "b, followed by a vowel, followed by t," matching *bat*, *bet*, *bit*, *bot*, and *but*. The search term finds just these five words because `[aeiou]` matches just one character. But we can change that slightly by adding a plus symbol, which in GREP means "one or more." All of a sudden the expression becomes much more interesting:

```
b[aeiou]+t
```

Tip: Write Out Space Characters

Space characters are often not so easy to spot in GREP expressions. To make them visible in your code, use `\x{0020}`, which is the Unicode notation for the space character. Or use the code `\s`, which stands for any white space (including paragraph breaks and tabs). For example, `set[-\s]?up` has the same meaning as `set[- ]?up` but is much clearer since someone looking at the code doesn't have to guess if there's a space there, or what is meant by the invisible space after the dash.

The simple addition of `+` makes the search term match series of vowels, so that in addition to the five three-letter words, it will now find *bait*, *boat*, *beat*, and *beaut* as well.

Tip: Use Backslashes to Find Literal Characters

The examples show that [,], and + have a special meaning in GREP. If you want to search these characters in a text, you need to tell InDesign that you want to use that character literally, not as a special character. You do that by writing a backslash before the character. Thus, A\+ matches the text A+, whereas A+ would find A, AA, AAA, etc. Writing a backslash before a character is often called "escaping."

The + operator is used frequently because it's so useful in defining patterns. For example, consider finding words of two or more syllables. In English, two-syllable words are characterized by sequences of at least vowels–consonants–vowels (*oboe*, *eerie*, etc.). The following search term will find that type of word:

```
[aeiouy]+[bcdfghjklmnpqrstvwxz]+[aeiouy]+
```

The pattern reads "one or more vowels, followed by one or more consonants, followed by one or more vowels."

To include two-syllable words that start with a consonant, add the consonants as an optional class using the ? operator:

```
([bcdfghjklmnpqrstvwxz]+)?[aeiouy]+[bcdfg jklmnpqrstvwxz]+[aeiouy]+
```

Note that by placing the first [b...z]+ in parentheses, the ? operator applies to that whole string, not just to +.

Finally, note that the + operator can be used on single character too: the search term e+ matches any sequence of one or more es, as in *beer* and *wheeeee!*.

Characters and Classes of Characters

In the examples we saw earlier, we were looking for literal text such as *center* and *copper*. Even when we used some GREP codes to find word patterns—for example, searching for gr[ae]y to find both *gray* and *grey*—we were still looking for literal text; in other words, we were looking for characters.

One step in the direction of a more general class of characters was an example we used above: [aeiouy], which you can read as "any vowel." It's easy to come up with other classes of characters: [bdfhklt] finds all ascender letters and [gjpqy] finds all descender letters. And you could come up with still more character classes, such as [ij], the class of letters that in English have a dot. All these are custom classes in the sense that you define them yourself for a particular purpose.

However, apart from these custom classes, there are a number of standard GREP character classes. They are not defined in terms of "real" characters as we have been doing until now; instead, they use so-called wildcards (or you could call them "meta-characters"). The most popular GREP wildcards can be found in the Find/Change dialog box under the @ > **Wildcards** menu (see Figure 1, next page).

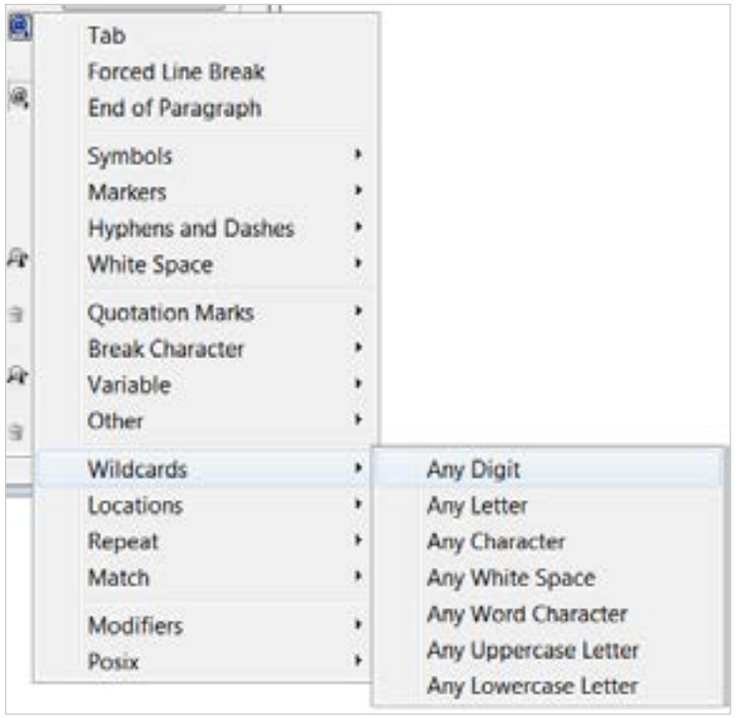


Figure 1: The wildcards menu

The wildcard \d matches the digits 0 through 9. Using the + operator, \d+ finds one or more digits, in other words, numbers, though only whole numbers. To find decimals as well, we have to create a class: [\d.]+. This search term finds (English) decimals such as *2.3* and *67.22*. To find numbers with thousands separators too, simply add the comma: [\d.,]+. Now we can locate *1,234.56*, as well as *3.456,12*.

See how flexible GREP classes are? We can combine literal characters and meta-characters in one class. And we can go even further if we want to find money amounts. All we need to do is add the currency symbols we're interested in:

```
[$£€¥\d.,]+
```

Letters

InDesign gives us three classes for letters: *Any letter*, *Any Uppercase letter*, and *Any Lowercase Letter*. They are straightforward: \l (that's a lowercase L) matches lowercase letters, \u finds uppercase letters, and with *Any letter*, well, Adobe gave us a class that we could have done ourselves: [\u\l], which of course is the class of upper- and lowercase letters.

With the two wildcards \u and \l we can do some sophisticated things, such as finding names. Let's define a name for the moment as two consecutive words that start with an uppercase letter, such as *Jane Hudson* and *James Morecambe*. To find

The Double Dollar Sign Bug

The strangest of InDesign's GREP bugs is an inability to find more than one instance of the dollar sign $ in a single paragraph. The $ has a special meaning in GREP (it locates the end of a paragraph; see further down) and should be found by using \$, but InDesign finds just the first instance in each paragraph. Try it: in a text with lots of dollar amounts, enter \$ in the Find What field, and press Find to highlight the first dollar symbol. Then press Find Next to find the next dollar sign. Though you can see several, InDesign says "Search is completed," meaning, "No more dollars found." The only way to find all dollar signs is to use the symbol's Unicode value, \x{0024}, or to use the dollar sign in the form of a character class, [$].

one word starting with a capital letter, we need the search term `\u\l+`: an uppercase letter followed by one or more lowercase letters. To find two such words, we simply use `\u\l+\x{0020}\u\l+` (remember that we use `\x{0020}` for the space character because the space character is not always easy to spot in a search pattern).

Search patterns like the one above can become a bit difficult to read, so we'll use a different format whenever it seems appropriate, as follows (but when you write such search patterns in the Find What field of the Find/Change dialog box, you must leave out those comments and write the pattern as one line):

GREP	What it finds
`\u\l+`	uppercase letter followed by one or more lowercase letters
`\x{0020}`	space character
`\u\l+`	same as the first line

But of course names aren't always as simple as this. For example, some people have double-barreled surnames such as Trevor-Roper. Such a name can be captured by saying that we're after an uppercase letter followed by a class consisting of lowercase letters, uppercase letters, and a hyphen: `\u[-\u\l]+`. This pattern, by the way, also captures names with irregular capitalization such as *LaGuardia*.

To locate prefixed names, like John von Neumann or Rip van Winkle, we have the following search patterns:

GREP	What it finds
`u\l+`	uc letter followed by lc letters
`\x{0020}`	space character
`(v[ao]n\x{0020})?`	maybe van or von, followed by a space character
`\u[-\u\l]+`	uc letter followed by lc, uc, hyphen

And because GREP is case-sensitive, the name prefix should in fact be written as `([Vv][oa]n\x{0020})?`, so that we capture *Van*, *van*, *Von*, and *von*. As you see, matching names can be tricky, and the search pattern in its current state fails to capture various other possibilities, such as various prefixes *de*, *du*, *le*, *van de*, and *von der*, to mention just a few. With what you've learned so far, I'll leave it up to you to formulate a pattern that matches all names.

Tip: Hyphens Go First

We used `\u[-\u\l]+` to match hyphenated names like Trevor-Roper. Though earlier we said that the order in a bracketed class is not important, for reasons that go beyond the scope of this article, the place of the hyphen is important. Whenever you want to include a hyphen in a bracketed class, always place it in first position, as we have done here.

Any white space

It's convenient that we can use a single wildcard to find any kind of white space. `\s` matches all spaces (the normal space

character, en- and em-dashes, thin spaces, half spaces, etc.), but also tabs and paragraph returns.

Any word character

This wildcard, \w, captures letters, digits, and the underscore character. I mention it here for completeness' sake.

Any character

This wildcard, the dot ., punches well above its weight: this little fellow matches everything in a paragraph! Well, almost. It doesn't match the paragraph mark (it could be made to, but we'll not go into that here). If we add the + operator, we say in effect "one or more of everything (except the paragraph mark)," and that must be whole paragraphs. It's easy to try out: type .+ in the Find What field, and click **Find/Find Next** a few times. You'll see that each time you press **Find Next**, the next paragraph is highlighted.

The Footnote Bug

The dot wildcard doesn't match footnote markers. This bug has been with us since InDesign CS3 despite multiple reports and requests.

Replacing Text

Replacing text using the GREP panel can be very straightforward, and just as simple as in any other application that offers a search-and-replace feature (Notepad, MS Word, etc.). Simply type a search term in the Find What field, a replacement text in the Change To field, and do the replacement. For example, to replace multiple paragraph marks with a single one, click in the Find What field, and then click the @ icon and choose **End Of Paragraph** from the menu. This inserts \r into the Find What field. Now type \r+, so that the Find what field reads \r\r+ (i.e., find two or more paragraph breaks). In the Change To field, type \r, and then press **Change all**.

As in the Find What field, you can see a list of special characters by clicking the @ icon. Apart from the last item, all items are the same as in the special character list for the Find What field. However, it's that last item, Found, that makes GREP replacements exciting (Figure 2).

Click on Found to see what's in that list. All you see is *Found Text*, *Found 1*, *Found 2*, . . . *Found 9*. Of course right now you're asking, what do Found 1, Found 2, etc. refer to?

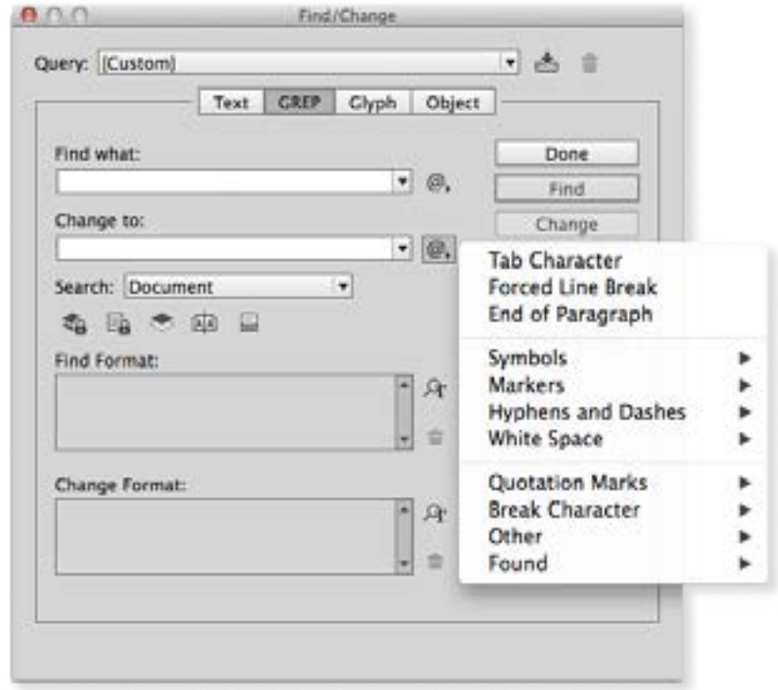


Figure 2: Change to special characters

They refer to what is matched by items in parentheses in the Find What field! We can list these references in any order we want, so we can in fact change the order of the found items. An example will make this clear.

Let's go back to the simplest form of the search pattern for names that we used earlier. Recall that we used this expression:

```
\u\l+
\x{0020}
\u\l+
```

to find names like *Jim Donegal*. What we want to do is reverse the order of the first name and the surname and add a comma after the surname, so that we get *Donegal, Jim*. To achieve this, first we'll add parentheses to the parts that we want to refer to later—the first name and the surname:

```
(\u\l+)\x{0020}(\u\l+)
```

That's all. What is matched by the first line will be Found 1, and what's matched by the third line will be Found 2. Now use the special characters list to insert Found 2, the surname, in the Change to field, which you'll see appear as $2. Type a comma and a space, and then enter the reference to Found 1 by typing $1. The Change To field should contain the following line:

```
$2, $1
```

Now, with GREP replacements you should always be very careful. Don't rush into **Change All** straight away; first click **Find,** then **Change**. If the result looks good, click **Find Next** and **Change**. If it still looks good, use **Change/Find** or **Change All**.

Like many specialized skills, using GREP can seem mysterious and daunting at first. But as the examples in this article show, almost anyone can understand and use GREP. All it takes is a little patience and practice, and you too can wield the amazing power of GREP in InDesign. Give it a try! You may soon wonder how you ever got along without it.

Peter Kahrel is a Scripting Engineer at Typefi Systems and the author of *GREP in InDesign*, as well as books on scripting and automating InDesign published by O'Reilly Media.

G

\K

A smoother, speedier way to look back

When a traditional lookbehind expression won't do the trick, try this obscure but powerful alternative.

In Issue 63 of *InDesign Magazine*, David Blatner discussed InDesign's positive lookbehind feature. He ended his piece with the words "the code after the = symbol must specify an exact number of characters. That is, you might expect the expression `(?<=\s+)\u` would mean 'an uppercase letter after a string of one or more spaces.' Unfortunately, the `+` ('one or more') part makes it fail because it's too open-ended."

The only way in which you could do variable-length expressions in a lookbehind was to list each variation in a separate lookbehind and group them as alternatives. Thus, to find numbers preceded by the word *Map*, *Figure*, or *Table*, you had to resort to an unwieldy expression such as `((?<=Map )|(?<=Figure )|(?<=Table ))\d+`. But this worked only if you knew the range of variation, so, yes, David was right, you can't use the `.` (dot) and `+` operators in lookbehinds.

That, until very recently, was what we all thought. But in CS6, Adobe introduced an operator that does allow lookbehinds that match variable-length text, namely `\K` (a classic Adobe Special: introduce something quite useful but don't tell anybody about it). Using this class, David's expression `(?<=\s+)\u` can be rendered as `\s+\K\u`. And the example I gave can be recast as `(Map|Figure|Table)\s\K\d+`.

What `\K` actually means is "Keep the text matched so far out of the overall match." This sounds strange, but to understand it, we can use a simple example, `a\Kb`. This matches the *b* in *ab*. To do so, first it searches for *a*. When *a* is matched, `\K` says "remove *a* from the overall match." The search resumes, looking for (and matching) *b*.

You can use `\K` just about anywhere. The main limitation is that it doesn't work with negatives, so you can't use it to find something that is *not* preceded by something else. But on the other hand, `\K` GREPs execute (marginally) quicker than classic lookbehinds. Try it! —*Peter Kahrel*

G

Limiting Matches

Among a plethoric passel of parentheses, how do you find the one you want?

GREP Level: Medium

Copyeditors dread bibliographies, especially if a publisher insists on their own format. That insistence can easily lead to many repetitive corrections that numb both the mind and the fingers. But we can use that repetitiveness to our advantage, because where there's a consistent pattern of errors, there's an opportunity to fix them with GREP.

One type of correction that regularly occurs is the use and placement of parentheticals, such as publication years and names of publishers. These are often wrong, and in order to fix them, we typically need to find just the first parenthetical in a paragraph or only the last one. But GREP expressions are gluttons—they want to consume as much content as possible—so if you look for something like `\(.+?\)`, you end up with *all* parentheticals. If you want to match just the first, or just the last, parenthetical, you need to take some special measures.

To find only the first parenthetical in a paragraph, the idea is to look for an opening parenthesis which has no other opening parenthesis between it and the beginning of the paragraph. This GREP expression does that: `^.*?\K\(.+?\)`. The expression works as follows: from the start of the paragraph (`^`), we match any sequence of characters (`.*`) until we hit the first opening parenthesis. Then we discard whatever we matched so far: that's what `\K`, the magnificent and only recently discovered modifier, does (see *InDesign Magazine* 73). From there, we match up to the next closing parenthesis. Because the part between the beginning of the paragraph and the first opening parenthesis is discarded, we in effect match just the first parenthetical.

Matching only the last parenthetical in a paragraph is in a way the mirror image of matching only the first one: find a parenthetical such that there is no opening parenthesis between the matched closing parenthesis and the end of the paragraph: `\([^)]+\)(?=[^\(]+$)`. The first part of the expression, `\([^)]+\)`, matches a parenthetical. The part of the expression that ensures that we match just the last parenthetical is `(?=[^\(]+$)`, which reads "from here to the end of the paragraph, and no character is `(`."

—*Peter Kahrel*

G

Reveal Codes

Normally, you can't target text with mixed formatting in a GREP search. But with this workaround, it's a cinch.

GREP Level: Medium

A limitation of GREP searches (and of normal text searches too) is that everything you're looking for must be in the same style. Thus, it's not possible to formulate a GREP expression along the lines of "find certain punctuation in italic that is followed by a non-italic space." This can be a bad limitation, but fortunately there's a way around this problem: you can reveal all or some formatting codes.

Using GREP expressions, you can temporarily add HTML-like tags that spell out formatting in a way that you can use and manipulate. For example, to show all italic formatting, enter `.+` in the Find What field, `<i>$0</i>` in the Change To field, and then set Italic in the Find Format panel and Regular (or Roman, or whatever the non-italic style is called) in the Change Format panel (Figure 1). The find expression matches everything in italic. The replace expression uses `$0`, which stands for "whatever was matched by the find expression." Instead of the HTML-style tags `<i>` and `</i>` that we used here, you can of course use any form: `%i%` and `%/i%` would do fine too, as would `@i@` and `#i#`—it doesn't matter much.

After running this query, your text could contain things like `;</i>` (where there's a space after the closing angle bracket). To find italic punctuation followed by a non-italic space, we can now search for `[:;,.]</i>\x20` (you'll recall that `\x20` stands for the space character). To get that punctuation out of italics, search using the GREP expression `([:;,.])</i>\x20`, and use `</i>$1\x20` as the replacement string.

To reinstate the italic formatting, use the Find What string `<i>(.+?)</i>`, the Change To string `$1`, and set Italic in the Change Format panel (make sure that you leave the Find Format panel blank).

—Peter Kahrel

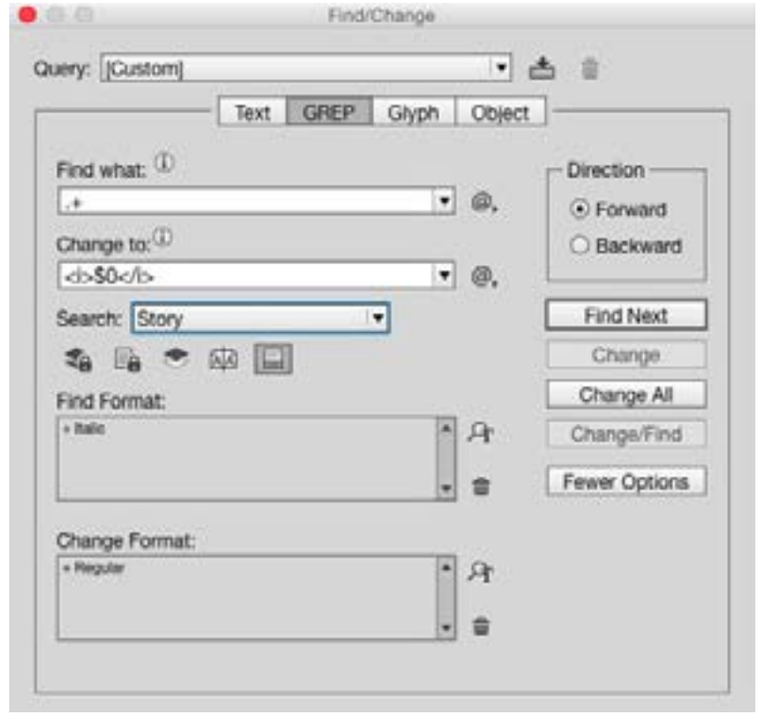


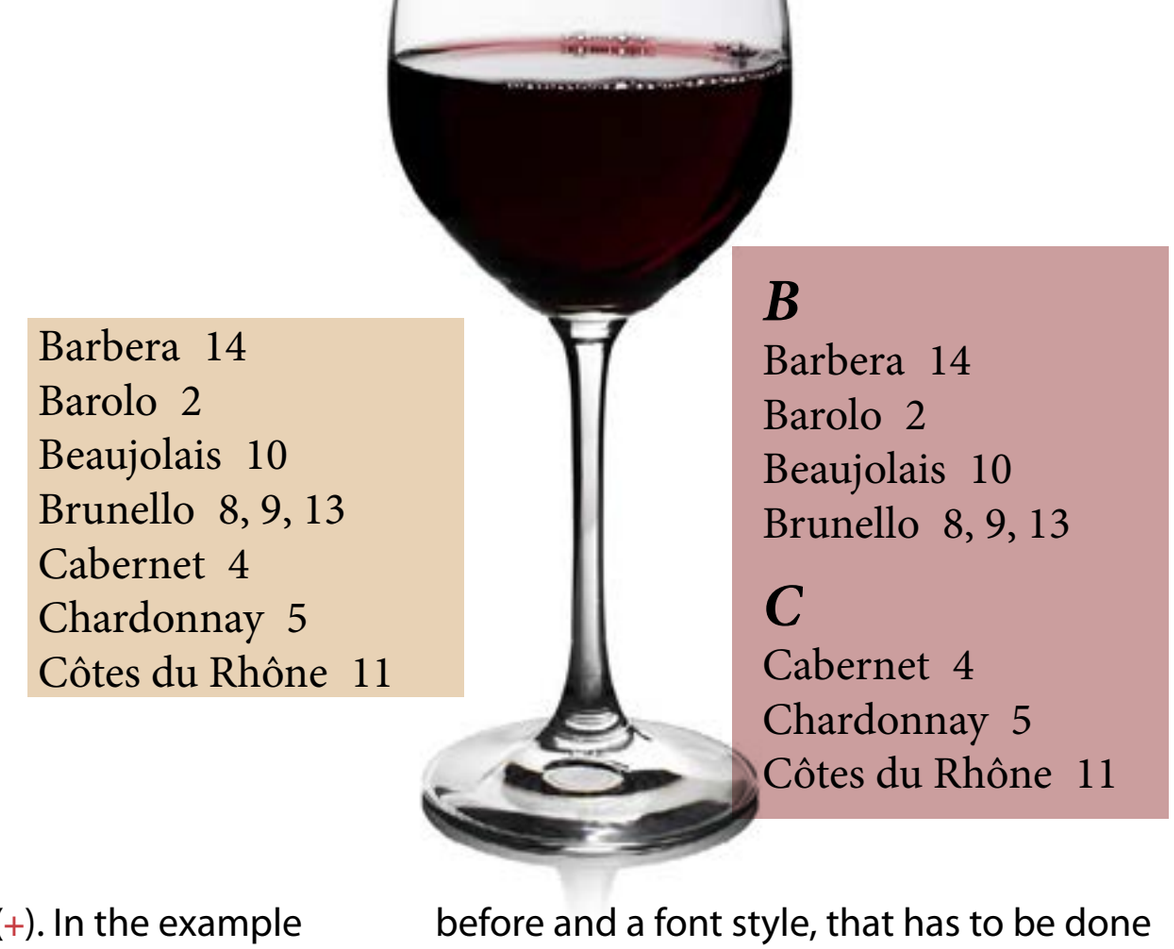
Figure 1: A GREP Find/Change to wrap all italic characters in tags that you can use in another GREP operation.

G

Adding Section Heads

With GREP, it's never too late to organize content with headers.

GREP Level: Medium



If you ever have to add section letters to an existing index, you can do that quickly using a GREP query.

You use a single query: Find what: `^(\u).+\r(\1.+\r)+` and Change to: `$1\r$0`. Translation: match and capture a capital (`(\u)`) at the beginning of a paragraph (`^`), and then match all following characters in the paragraph (`.+`) up to and including the return character (`\r`). Then match a letter that's the same (`\1`) as the one we captured earlier, followed by all characters up to the end of that paragraph (`.+\r`), group that, and find as many as possible of the same (`+`). In the example above, `^(\u).+\r` matches *Barbera 14* (and the return character), and `(\1.+\r)+` matches all following lines that start with *B*. And there's your section.

Now, to insert the section letter (which is the letter we matched by `^(\u)`), we replace the section with the letter we captured (`$1`) followed by a return (`\r`) and the entire section (`$0`)—remember that `$0` stands for "everything that was matched by the Find What expression."

If you want to apply a paragraph style to the section letters, e.g., to add some space before and a font style, that has to be done using a very simple, separate query. At Find What, enter `^\u$`, and in the Change format panel, enter the paragraph style—in other words, apply a paragraph style to all one-letter paragraphs. Make sure the Change To field is empty, and then click Change All.

The drawback of this method is that it makes a mess of any formatting, such as italics. You can convert formatted text to text tags, insert the section letters, and then convert the text tags back to the formatting (see *InDesign Magazine*, issue 76 for details).

—Peter Kahrel

G

\x

Unicode

Target any range of characters: Hebrew or Hirigana, Cherokee or currency, Devanagari or dingbats.

GREP Level: Medium

GREP is an excellent tool for finding characters in certain Unicode ranges and applying a character style to them. I've used this method to apply a phonetic font to phonetic characters and an Arabic font to Arabic script. To accomplish this, you need to know InDesign's notation for Unicode characters and the limits of, in this case, the phonetic and Arabic Unicode ranges.

The limits of Unicode ranges can be found on the website of the Unicode consortium. There you'll find that the basic Arabic characters range from Unicode values 0600 to 06FF. InDesign's format for Unicode characters is `\x{0000}`, so in InDesign's notation the basic Arabic characters range from `\x{0600}` to `\x{06FF}`.

Assuming that your document contains a character style (e.g., "arabic") that sets an Arabic font, you can apply that character style to all Arabic characters in the basic range like this: in the Find What field, enter `[\x{0600}-\x{06FF}]+`, and set the character style in the Change Format panel. The bracket notation is used in GREP expressions to define a range of characters (it's called a character class); and we use the plus operator to apply the character to series of Arabic characters.

Looking at the chart more closely, however, you notice that things are slightly more complicated: Arabic is contained not in one range, but in four. Apart from the basic range 0600–06FF, we have Arabic Extended-A (08A0–08FF), Arabic Presentation Forms-A (FB50–FDFF), and Arabic Presentation Forms-B (FE70–FEFF). In our document we're interested only in one additional class, Extended-A. To find all characters in two ranges—here, Unicode ranges 0600–06FF and 08A0–08FF—we simply add the latter to the class we defined earlier: `[\x{0600}-\x{06FF}\x{08A0}-\x{08FF}]+`.

—Peter Kahrel

G

(?<=)

Lookahead

Targeting a string by what follows it

GREP Level: Medium

Many find-and-replace actions involve finding a string and replacing—or applying some formatting to—only part of the found string. But if you find a string and want to apply formatting to just part of it, you have to select the part of the string you're interested in, do the replacement or formatting, and then find the next occurrence. All this can become very tedious very quickly.

Fortunately, GREP offers a way to do conditional finds, such as "find the word Figure only if it's followed by a digit." These conditionals are called "lookahead," and their general format is `(?=)`. In our example of finding instances of the word *Figure* only if they're followed by a digit, we can use the query `Figure(?=\s\d)`. Note that we include in the lookahead `\s` (the space after the word *Figure)* and the digit `\d`. If you try this, you'll see that *Figure* is highlighted, but the space and digits are not. This means that whatever we do now applies only to *Figure*. For example, if you want to italicize these instances of *Figure*, just set Italics in the Change Format panel. Now you can click Change All (or the more cautious Change and Change/Find) to process all remaining instances.

In lookaheads, you can use other GREP constructs as well. Say you want to capture instances of the word *Figure* not only when they're followed by a digit, but also by the symbol #, which you use as a placeholder, for instance. This is possible by using the character class `[\d#]`, which defines both digits and # as possible characters following *Figure*: `Figure(?=\s[\d#])`. If a character class is not suitable, for example, when you use a multi-character placeholder such as #@#, then you can use alternatives inside the lookahead: `Figure(?=\s(\d|#@#))`.

Lookahead has a negative counterpart that lets you match text when it's not followed by some other text. The format of these so-called negative lookaheads is `(?!)`. For example, to find instances of the word *Figure* when it's *not* followed by a digit, use `Figure(?!\s\d)`.

—*Peter Kahrel*