Finding and replacing characters using wildcards

https://wordmvp.com/FAQs/General/UsingWildcards.htm

Article contributed by Graham Mayor, with thanks also to Klaus Linke

Contents

(to return to top, press Ctrl+Home)

Overview

Wildcards are like the blank pieces in Scrabble, or like the Jokers you can use in some card games to stand in for any card. You are perhaps already familiar with the "*****" and "**?**" wildcards from file matching: In the **File + Open** dialog, you can display all files with the extension ".doc" by typing "*.doc", or all files "01062001.doc", "01072001.doc", "01122001.doc"... by typing "01??2001.doc".

But the wildcard feature in Word goes way beyond that, and can be very powerful.

To begin, you must first turn Wildcards **on** in the Find/Replace dialog. To do so, bring up the **Find** dialog, click **More** and check **Use wildcards**. In a macro, set .**Find.MatchWildcards = True**. If you do not do this, Word treats the wildcard characters as if they were ordinary text.

As we'll see later, you can define ranges **[]**, groups **()**, repeats **@**, **{}**, anchors **< >** and exceptions **!**. With these regular expressions you can search for patterns in your text that have certain things in common (some pattern: for example, that they only contain certain characters, or a certain number of characters).

Note: Word uses "lazy" pattern matching: this means it will quit matching as soon as possible. Most Unix tools use "greedy" pattern matching (the algorithm tries to match as much text as possible), so if you have used such tools, beware!

The secret of using wildcard searches is to use a "pattern" that identifies the string of text that you wish to find, and ignores everything else. Wildcards are used to represent the characters or sequences of characters in that string.

Because different combinations of characters can be represented by a variety of wildcard combinations, there is often more than one way of identifying a particular string of text within a document. How you choose to represent that group of characters is therefore often a matter of individual preference; and the context of the text within the document will to a great extent dictate the most suitable combination to use on a particular occasion.

The following is a list of the characters that have a special meaning in wildcard searches **([ ] { } < > ( ) - @ ? ! * \ )**.

Note: wildcard searches are case sensitive.

It doesn't help that the list of wildcard characters in Word's Help files is almost impossible to find! The wildcard characters are all listed and described in this article, but if you need to find them in Help, the topic is called: "Type wildcards for items you want to find". But you can't get to that article directly; you must first find the topic: "Fine-tune a search by using wildcard characters", which contains a link to it!

Zen tip: when using wildcard searches: don't wrinkle your brow or bite on your tongue while thinking it through – you have to keep up a regular expression. :-|

The theory

1. **? and ***

   The two most basic wildcard characters are **?** and *****. These are essentially similar in use.

   ? is used to represent a single character and ***** represents any number of characters. On their own, these have limited use.

   s?t will find sat, set, sit, sat and any other combination of three characters beginning with "s" and ending with "t". It will also find that combination of letters *within* a word, thus it would locate the relevant (highlighted) part of inset etc.

   s*t will find all the above, but will also find "secret", "serpent", "sailing boat" and "sign over document", etc.

   Note: Unlike **?**, which always looks for the missing character, ***** will also find occurrences of the two adjacent characters (here "s" & "t") with *no* text between them e.g. "streaker". It is a blunt instrument and must be used with care, or with other characters to tie it down to only the text you require. There is no limit to the amount of text that the ***** character might match. It can end up matching all of your text in a multi-megabyte document!
2. **@**

   @ is used to find one or more occurrences of the previous character. For example, lo**@**t will find lot or loot, ful**@** will find ful or full etc.
3. **< >**

   Used with any of the above (or any other combination of wildcards and characters), you can use the tags **<** and **>** to mark the start and end of a word, respectively. Thus, building on the example we used for the "*****" character: **<s*t>** would find "secret" and "serpent" and "sailing boat", but **not** "sailing boats" or "sign over documents". It will also find "'set" in "tea-set" , but **not** "set" in "toolset".

   Again, beware of using "*", as **<s*t>** will find any block of text from a word starting with "s" to the end of the next word in the document that ends with "t", which may not always be what you had in

mind.

The **<>** tags can be used in pairs, as above; or individually, as appropriate. For instance, **ful@>** will find “full” and the appropriate part of “wilful”, but will not find “wilfully”.

4. **[ ]**

Square brackets are always used in pairs and are used to identify specific characters or ranges of characters. For example:

[abc] will find *any* of the letters a, b, or c. **[F]** will find upper case “F”.

[A-Z] will find any upper case letter; **[0-9]** will find any single number; **[13579]** will find any odd numbers; **[0-9A-z]** will find any numbers *or* letters.

You can use any character or series of characters in a range **[ ]**, including the space character. Characters are processed in alphanumeric order – lowest first. If you are uncertain which character is lower than another, look in the Insert + Symbol dialog.

5. ****

If you wish to search for a character that has a special meaning in wildcard searches – the obvious example being “?” – then you can do so by putting a backslash in front of it:

[\?] will **not** find “****” followed by any character; but will find “?” instead.

If you wish to find the backslash itself then you need to precede that with a backslash **[\\]**. It's best if you always put the double-backslash into its own range, as in **[\\]** – sometimes it doesn't work, otherwise.

As previously mentioned, the following is a list of the characters that have a special meaning in wildcard searches **([ ] { } < > ( ) - @ ? ! * \ )**.

6. **[!]**

[!] is very similar to **[ ]** except in this case it finds any character **not** listed in the box so **[!o]** would find every character except “o”. You can use ranges of characters in exactly the same was as with [ ], thus [!A-Z] will find everything except upper case letters.

7. **{ }**

Curly brackets are used for counting occurrences of the previous character or expression.

{n} This finds exactly the number “n” of occurrences of the previous character (so for example, **a{2}** will find “aa”).

{n,} finds at least the number “n” occurrences; so **a{2,}** will find “aa” and “aaaa”).

{n,m} finds text containing between “n” and “m” occurrences of the previous character or expression; so **a{2,3}** will find “aa” and “aaa”, but only the first 3 characters in “aaaa”).

Note: Counting can be used with individual characters or more usefully with sets of characters e.g. [deno]{4} will match done, node, eden); or with bracketed groups: (ts,){3} will match ts, ts, ts, .

(Unfortunately, there is no wildcard to search for “zero or more occurrences” in Word wildcard searches; [!^13]{0,} does not work).

8. **( )**

Round brackets have no effect on the search pattern, but are used to divide the pattern into logical sequences where you wish to re-assemble those sequences in a different order during the replace – or to replace only part of that sequence. They must be used in pairs and are addressed by number in the replacement e.g.

(John) (Smith) replaced by \2 \1 (note the spaces in the search and replace strings) – will produce Smith John

or replaced by \2 alone will give Smith.

Note: The placeholders \1, \2 etc., can also be used in the search string to identify recurring text. e.g.

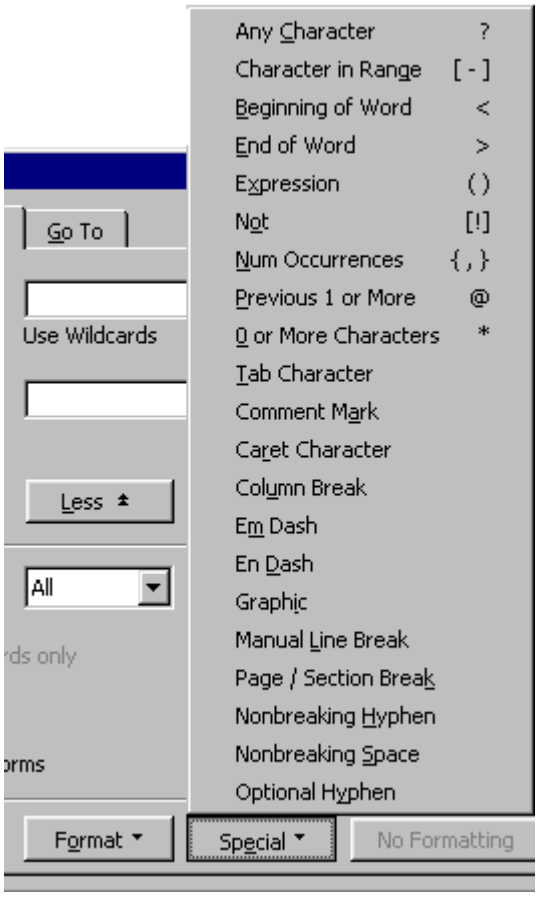
Fred Fred could be written (Fred) \1.

Round brackets are perhaps the most useful aspect of complex wildcard search and replace operations.

9. **^**

The **^** ("caret") character is not specific to wildcard searches but it sometimes has to be used slightly differently from normal, when searching for wildcards.

In the Find and Replace dialog, if you click in the "Find what" and "Replace with" boxes, and click the "Special" button, you will see a list of supported special characters that you can use; select one and a code will be inserted in the box for you. The ones near the bottom of the list insert special **^** codes, such as **^t** for a tab. Once you know what the codes are, you can type them straight in without using the "Special" button.



☹Unfortunately, the range of options available from the "Special" button when you do a wildcard search is more limited than in an ordinary search. Some notable examples, and their workarounds:

- You may wish to identify a character string by means of a paragraph mark ¶. The normal search string for this would be **^p**. ^p does **not** work in wildcard search strings! It **must**, however, be used in replace strings, but when searching, you must use the substitute code **^13**.
  If you use ^13 in a replace string, invalid characters, that look like paragraph marks but aren't, will be inserted – so beware!
- Wildcard searches will also not find footnote/endnote marks – substitute **^2**.
- In normal searches you can use **^u** followed by the character number to find a character by code (which is useful for finding non-keyboard characters). That doesn't work in wildcard searches (and even in the replacement text of normal searches). So you have to either paste the character itself into the search text, or use the Alt-XXXX-trick to type it (the latter doesn't work in Word 97).
- There doesn't seem to be any way to find fields in a wildcard search. Even **^19** (the field code

character code) doesn't work in wildcard searches – this seems to be a bug.

For most other special characters and objects, you can use the same codes as in simple searches (^l = manual line break, ^g = graphic, ^^ = caret ...).

In many cases, characters can be addressed using their character numbers. The obvious example is ^13 for "paragraph", as mentioned above. Unfortunately, this has not been implemented in a completely predictable or reliable manner, and if you have a need to search using character numbers, ensure that the search works with a test sample before committing to it.

So much for the theory. How are the wildcards used in practice?

The practice

Example 1: Transpose first name and surname

There are many occasions when you are presented with blocks of text or numbers etc., where the order of the text is not what you might require in the final document. Swapping the placement of forename and surname as above is one such example – and don't forget you can add to the replacement, even when using bracketed replacements

For instance, you may wish **John Smith** to appear as **Smith, John**.

Or, more likely, you may have a column of names in a table, where you wish to exchange all the surnames with all the forenames.

John Smith
Brian Jones
Alan Williams

You could do them one at a time, but by replacing the names with wildcards, you can do the lot in one pass.

Let's then break up the names into logical sequences that can only represent the names:

- At its simplest, we have here two words – John and Smith. They can be represented by**<*>[space]<*>** – where **[space]** represents a single press of the spacebar.
- Add the round brackets **(<*>)[space](<*>)** and replace with **\2[space]\1**
- Run the search on the column of names and all are swapped. Run it again and they are swapped back.

Note: If you get it wrong, remember that Word's "undo" function (**Ctrl+Z**) is very powerful and has a long memory! 🙂

If some of the names contained middle names and/or initials

- If some of the names contained middle names and/or initials, you would first have to convert your table to text (separated by paragraph marks). Select **Table + Convert Table to Text**.

  Or if there is more than one column in your table, paste your "Name" column into a new document, and *then* convert that column to text.
- You could then replace:
  (<*) ([!]@)^13

with:
\2, \1^p

This would convert:

John F. Kennedy	▶	Kennedy, John F.
J. Smith		Smith, J.
John Smith		Smith, John

- Finally, convert the text back to a table. (Select **Table + Convert Text to Table)**.

  If there was more than one column in your original table, then when converting the text back to a table, be sure to select "Paragraphs", where it says "Separate text at". Then, in your original table, delete the old column and paste in the new one.

Example 2: Transposing dates

Another useful example might be the changing of UK format dates to US format dates – or vice versa.

7th August 2001 to August 7, 2001

(For a similar example, see also Transpose dates from mm/dd/yy to yy/mm/dd.)

To give an example of how most of the wildcard characters could be used in one search sequence to find any UK date formatted as above, the following search pattern will do the trick:

[0-9]{1,2}[dhnrst]{2} <[AFJMNSOD]*> [0-9]{4}

Breaking it down:

- **[0-9]** looks for any single digit number, but dates can have two numbers; so to restrict that to two, we use the "count" function {}. We want to find dates with 1 or 2 numbers so:
  [0-9]{1,2}
- Next bit is the ordinal "th" – Ordinals will be "nd", "st", "rd", or "th" so identify those letters specifically:
  [dhnrst]

  There will always be two letters, so restrict the count to 2:
  [dhnrst]{2}
- Next comes the space. You can insert a space with the spacebar **[space]**.
- The month always begins with one of the following capital letters – AFJMNSOD. We don't know how many letters this month has so we can use the blanket "*" to represent the rest. And we are only interested in that word so we will tie it down with <> brackets:
  <[AFJMNSOD]*>
- There's another space **[space]** followed by the year. Years will always have four numbers so:
  [0-9]{4}
- Finally, add the round brackets to provide a logical break-up of the sequence:
  ([0-9]{1,2}**)(**[dhnrst]{2}**)**[space]**(**<[AFJMNSOD]*>**)**[space]**(**[0-9]{4}**)**

  and replace with:
  \3[space]\1,[space]\4

  (where **[space]** represents pressing the spacebar once) to re-order the sequence.
- US style manuals specify that ordinals should *not* be used in dates, but if you did want to keep the ordinals (so converting 7th August 2001 to August 7th, 2001), you could replace

([0-9]{1,2}[dhnrst]{2}**)**[space]**(**<[AFJMNSOD]*>**)**[space]**(**[0-9]{4}**)**

with:
\2[space]\1,[space]\3

- You can use the same logic in reverse to convert August 7th, 2001 to 7th August 2001; or to convert August 7, 2001 to 7 August 2001. Unfortunately you can't **add** the ordinals using a wildcard Find & Replace, if they're not there to begin with – you would need to use a macro if you wanted to do that.

Example 3: Adding or removing the period in salutations

Assume you are parsing addresses and wish to separate the honorific from the name. American usage puts a period at the end (Mr., Mrs., Dr.) while British usage omits the period.

([DM][rs]{1,2})( )

will find Mr Ms Mrs Dr without the period and

\1.\2

will put one in.

or vice versa:

([DM][rs]{1,2}).

will find Mr. Ms. Mrs. Dr. with the period and

\1

will take it out.

Example 4: Duplicate paragraphs (and rows)

- **(*^13)\1\1** will match any sequence of three identical paragraphs.
- If you replace:

  (*^13)\1

  with

  \1

  it will delete all consecutive duplicate paragraphs in the document. Repeat until nothing is found, to delete all duplicate paragraphs in the document (as long as you have sorted the text first).
- To delete duplicate rows in a table (provided you have no merged cells), you can convert the table to text (Table + Convert to Text, using a tab delimiter); delete the duplicate paragraphs using the above method, then convert the text back to a table.

Example 5: Tags

\<([!\<\>]@\>)*\</\1

will match any well-formed XML element including start-tag and end-tag such as:

some text

or

<customer-name>John Smith</customer-name>

See also:
Delete html tags or sgml tags or other bracketed tags (<example>) from a document without affecting other text.

Example 6: Formatting

By building up appropriate patterns, you can search for almost any combination of characters.

Of course you can also restrict your searches by specifying some style or formatting, or add formatting for the replacement text. See Finding and replacing non-printing characters (such as paragraph marks), other special characters, and text formatting for more on this.

A nice trick if you want to apply formatting to **a part** (but not all) of the search text is to put in "tags" in a first replacement.

In a find/replace, you can only change the formatting of the whole find-text; so you would need to do two find-replaces to get the job done. In the first, you would "tag" the text that has to be formatted; in the second, you format them (and remove the tags).

Find what:
(something)(something else)(another string)

Replace with:
\1$$\2##\3

and then remove the tags and apply the formatting in a second replace:

Find what:
$$(*)##

Replace with:
\1 ((bold))

Imagine, for instance, that you've got a text file, and headings are marked up by having 3 empty paragraphs before them and an empty paragraph after.

Find what:
^13{4}([!^13]@^13)^13

Replace with:
^p<H1>\1

Then Find what:
\<H1\>(*)

Replace with:
\1 ((style "Heading 1"))

This will remove the empty paragraphs and format the headings.

See also: Apply the built-in Heading 1 paragraph style to all paragraphs containing text in ALL CAPS

More examples

See also:
Replace any instance of the left square bracket character "[" that happens to be the fifth character in a paragraph

How to make urls (and delimiters such as \, /, : and @) wordwrap in Word

How to replace text in quotation marks with italic or highlighted text minus the quotes

Tips for advanced users

- You can paste any (Unicode) character (unfortunately **not** characters from decorative (Symbol) fonts) into your search expressions. So copying the first and last characters from the Greek or Cyrillic subsets into a search:

  [;-ώ]
  would match any Greek character:
  α β γ δ ε ...

  <[Ё-ґ]@>
  matches any Cyrillic word:

  Вы можете помочь мне?
  ("Can you help me please?")

  Note: in Word 97, the characters sometimes display in the dialog box as squares, but they do work.
- In Word 2000+, you can type in Unicode characters with the Alt-key (make sure NumLock is on, then hold down the Alt-key and type the numbers on the numeric keypad). Since all characters from decorative fonts (Symbol-, Wingdings-fonts ...) are kept in a special code page from &HF000 to &HF0FF, you can search for them with [Alt61472-Alt61695]. (See also Finding and replacing symbols).
- [a-c] will not **only** match "a", "b", "c", but **also**:
  àáâãäåæąāă

  [a-à] will **not** match all characters from U+0061 (a) to U+00E0 (à).

  For some discussion on Unicode sorting and wildcards see the Unicode Regular Expression Guidelines. In general, even if the sorting used in Word is not very transparent, you usually get the results you would expect.
- If you are using VBA and doing string comparisons, the "Like" operator (covered in VBA Help) gives you much of the functionality of a wildcard search.
- If you are using VBA, you might want to look at the RegExp object (from VBScript, which you can include in your macro projects, and which offers some features not included in Word wildcards). Choose Tools + References in the VB editor, and check "Microsoft VBScript Regular Expressions". You can get help if you call the VBScript editor and search its online help for RegExp. A list of supported wildcards is given in the help on the Pattern-property.

Gremlins to be aware of (for advanced users)

- Sometimes Word will get confused if it encounters "escaped" brackets **\(** or **\)**. For example:

(\\)
will match any character, not only a backslash.

Workaround: use
([\\])
instead.

- ([a-z]\() throws an error – it should find an “a(”.

  Workaround: Use ([a-z][\(]) instead.
- Word starts matching again **after** the previous match/replacement; so for example **^13*^13** will match only every second paragraph in a Replace operation.

  If that isn't what you want, you could (in this example) use *^13 instead.

  Other times, if this isn't what you want, you have to do multiple Find & Replaces. For example, supposing you wanted to subscript all the numbers in any chemical formula such as C2H4. You could tag the numbers that have to be subscripted; you want to subscript any group of numbers ([0-9]@) that follow a letter > ([A-Za-z]), and that are not followed by another number ([!0-9]):

  Find what: **([A-Za-z])([0-9]@)([!0-9])**

  Replace with: **\1_{\2}\3**

  That search will for example match “C2H” in “C2H4” (letter + number + non-number), and then continue searching **after** the “H”. So you need to run this replacement twice, before doing the Find and& Replace that applies the formatting.

  But sometimes, Word finding every second instance of your search string may **be** what you want, and you can make use of the feature. (For instance, if you want to format the text between two identical tags).
- Not a bug but still annoying: You have to escape any special character even if you type its code; so ^92 will have the same problems as typing the backslash.
- The construction **{0,}** (find zero or more of the preceding item) is refused as incorrect syntax. This concept is available in Unix regular expression matching, so it's a curious omission.
- You don’t always have to “escape” the special characters, if the context makes it clear that the special meaning isn’t wanted. [abc-] matches “-”, and [)(] matches “)” or “(”. This may sometimes make your searches behave differently from what you expected.