How did wildcards work in MS-DOS?

devblogs.microsoft.com/oldnewthing/20071217-00

December 17, 2007

Raymond Chen

The rules were simple but led to complicated results.

MS-DOS files were eleven characters long with an implicit dot between characters eight and nine. Theoretically, spaces were permitted anywhere, but in practice they could appear only at the end of the file name or immediately before the implicit dot.

Wildcard matching was actually very simple. The program passed an eleven-character pattern; each position in the pattern consisted either of a file name character (which had to match exactly) or consisted of a question mark (which matched anything). Consider the file "`ABCD····TXT`", where I've used `·` to represent a space. This file name would more traditionally be written as `ABCD.TXT`, but I've written it out in its raw 11-character format to make the matching more obvious. Let's look at some patterns and whether they would match.

Pattern	Result	Explanation
`ABCD····TXT`	Match	exact
`???????????`	Match	all positions are a wildcard so of course they match
`ABCD????···`	No match	space (position 9) does not match `T`
`A?CD····???`	match	perfect match at `A`, `C`, `D`, and the spaces; wildcard match at the question marks

The tricky part is converting the traditional notation with dots and asterisks into the eleven-character pattern. The algorithm used by MS-DOS was the same one used by CP/M, since MS-DOS worked hard at being backwards compatible with CP/M. (You may find some people who call this the FCB matching algorithm, because file names were passed to and from the operating system in a structure called a File Control Block.)

1. Start with eleven spaces and the cursor at position 1.
2. Read a character from the input. If the end of the input is reached, then stop.

3. If the next character in the input is a dot, then set positions 9, 10, and 11 to spaces, move the cursor to position 9, and go back to step 2.
4. If the next character in the input is an asterisk, then fill the rest of the pattern with question marks, move the cursor to position 12, and go back to step 2. (Yes, this is past the end of the pattern.)
5. If the cursor is not at position 12, copy the input character to the cursor position and advance the cursor.
6. Go to step 2.

Let's parse a few patterns using this algorithm, since the results can be surprising. In the diagrams, I'll underline the cursor position.

First, let's look at the traditional " `ABCD.TXT` ".

Input	Pattern	Description
	`···········`	Initial conditions
A	`A··········`	Copy to cursor and advance the cursor
B	`AB·········`	Copy to cursor and advance the cursor
C	`ABC········`	Copy to cursor and advance the cursor
D	`ABCD·······`	Copy to cursor and advance the cursor
.	`ABCD·······`	Blank out positions 9, 10, and 11 and move cursor to position 9
T	`ABCD····T··`	Copy to cursor and advance the cursor
X	`ABCD····TX·`	Copy to cursor and advance the cursor
T	`ABCD····TXT`	Copy to cursor and advance the cursor

The final result is what we expected: `ABCD····TXT`.

Let's look at a weird case: the pattern is `ABCDEFGHIJKL`.

Input	Pattern	Description
	`···········`	Initial conditions
A	`A··········`	Copy to cursor and advance the cursor
B	`AB·········`	Copy to cursor and advance the cursor

C	`ABC·̲·······`	Copy to cursor and advance the cursor
D	`ABCD·̲······`	Copy to cursor and advance the cursor
E	`ABCDE·̲·····`	Copy to cursor and advance the cursor
F	`ABCDEF·̲····`	Copy to cursor and advance the cursor
G	`ABCDEFG·̲···`	Copy to cursor and advance the cursor
H	`ABCDEFGH·̲··`	Copy to cursor and advance the cursor
I	`ABCDEFGHI·̲·`	Copy to cursor and advance the cursor
J	`ABCDEFGHIJ·̲`	Copy to cursor and advance the cursor
K	`ABCDEFGHIJK`	Copy to cursor and advance the cursor

Sure, this was extremely boring to watch, but look at the result: What you got was equivalent to `ABCDEFGH.IJK`. The dot is optional if it comes after exactly eight characters!

Next, let's look at the troublesome `A*B.TXT`.

Input	Pattern	Description
	`·̲··········`	Initial conditions
A	`A·̲·········`	Copy to cursor and advance the cursor
*	`A??????????`	Fill rest of pattern with question marks and move to position 12
B	`A??????????`	Do nothing since cursor is at position 12
.	`A???????·̲··`	Blank out positions 9, 10, and 11 and move cursor to position 9
T	`A???????T·̲·`	Copy to cursor and advance the cursor
X	`A???????TX·̲`	Copy to cursor and advance the cursor
T	`A???????TXT`	Copy to cursor and advance the cursor

Notice that the result is the same as you would have gotten from the pattern `A*.TXT`. Any characters other than a dot that come after an asterisk have no effect, since the asterisk moves the cursor to position 12, at which point nothing changes the parse state except for a dot, which clears the last three positions and moves the cursor.

I won't work it out here, but if you stare at it for a while, you'll also discover that `*.*` is the same as `*` by itself.

In addition to the rules above, the MS-DOS command prompt had some quirks in its parsing. If you typed `DIR .TXT`, the command prompt acted as if you had typed `DIR *.TXT`; it silently inserted an asterisk if the first character of the pattern was a dot. This behavior was probably by accident, not intentional, but it was an accident that some people came to rely upon. When we fixed the bug in Windows 95, more than one person complained that their `DIR .TXT` command wasn't working.

The FCB matching algorithm was abandoned during the transition to Win32 since it didn't work with long file names. Long file names can contain multiple dots, and of course files can be longer than eleven characters, and there can be more than eight characters before the dot. But some quirks of the FCB matching algorithm persist into Win32 because they have become idiom.

For example, if your pattern ends in `.*`, the `.*` is ignored. Without this rule, the pattern `*.*` would match only files that contained a dot, which would break probably 90% of all the batch files on the planet, as well as everybody's muscle memory, since everybody running Windows NT 3.1 grew up in a world where `*.*` meant *all files*.

As another example, a pattern that ends in a dot doesn't actually match files which end in a dot; it matches files with no extension. And a question mark can match zero characters if it comes immediately before a dot.

There may be other weird Win32 pattern matching quirks, but those are the two that come to mind right away, and they both exist to maintain batch file compatibility with the old 8.3 file pattern matching algorithm.

Raymond Chen

Follow