`makematch.sty`
Pattern matching with wildcards
version 1.7925

David Kastrup[*]

2007/02/23

1 Using `makematch`

The basic idea of `makematch` is to compile patterns and targets (and/or lists of them) and match the former to the latter. This functionality is used extensively in QuinScape's `qstest` package for unit testing. We'll use that package for documenting usage of `makematch`; the following construct is skips the tests when `makematch.dtx` is used as a standalone file.

```
1 ⟨*dtx⟩
2 \iffalse
3 ⟨/dtx⟩
4 ⟨*test⟩
5 \RequirePackage{makematch,qstest}
6 \IncludeTests{*}
```

    `makematch` requires an eTeX-based LaTeX which should be standard for current TeX distributions.

1.1 Match patterns and targets

This package has the notion of match patterns and targets. Patterns and targets get sanitized at the time they are specified; this means that nothing gets expanded, but replaced by a textual representation consisting of spaces (with catcode 10) and other characters (catcode 12). Control words are usually followed by a single space when getting sanitized.

    Patterns and targets are actually generalized to pattern and target lists by this package: you can, when specifying either, instead give a list by using an optional argument for specifying a list separator (the lists used in `qstest` are comma-separated).

---

[*]David.Kastrup@QuinScape.de, QuinScape GmbH

Target lists are unordered: the order of targets in them is irrelevant. Leading spaces in front of each target get stripped, all others are retained.

Pattern lists similarly consist of a list of patterns, with leading spaces stripped from each pattern. In contrast to target lists, the order of pattern lists is significant, with later patterns overriding earlier ones. Also in contrast to target lists, empty patterns are removed.

There are two special characters inside of a pattern: the first is the wildcard * which matches any number of consecutive characters (including the empty string) in a target. Wildcards can occur anywhere and more than once in a pattern.

The second special character in a pattern is only recognized at the beginning of a pattern, and only if that pattern is part of a pattern list (namely, when a list separator is specified).[1] If a pattern is preceded by ! then the following pattern, if it matches, causes any previous match from the pattern list to be disregarded.

So for example, the pattern list {*,!foo} matches any target list that does not contain the match target foo.

An empty target list {} is considered to contain the empty string. Thus the pattern * matches every target list, including empty ones, while the pattern list {} does not match any keyword list, including empty ones.

1.2   The Interface

\MakeMatcher takes two mandatory arguments. The first is a macro name. This macro will become the new matcher. The second argument of \MakeMatcher is the pattern to match. An optional argument before the mandatory ones can be used for specifying a list separator in which case the first mandatory argument becomes a pattern list (only in this case are leading ! characters before list elements interpreted specially).

```
7 \begin{qstest}{\MakeMatcher}{\MakeMatcher}
8    \MakeMatcher\stylefiles{*.sty}
9    \MakeMatcher\headbang{!*}
10   \MakeMatcher[,]\truestylefiles{*.sty,!.thumbnails/*,!*/.thumbnails/*}
```

The matcher constructed in that manner is called with three arguments. The first argument is a control sequence name containing a match target (or target list) prepared using \MakeMatchTarget (see below).

Alternatively, the first argument can be a brace-enclosed list (note that you'll need *two* nested levels of braces, one for enclosing the argument, one for specifying that this is a list) which will then get passed to \MakeMatchTarget (see below) for processing before use. The inner level of braces inside of the first argument may be preceded by a bracketed optional argument specifying the list separator for this list.

The second argument of the matcher is executed if the pattern list for which the matcher has been built matches the keyword list. The third is executed if it doesn't. List separators of pattern and keyword list are completely independent

---

[1] And if ! is not the list separator of the list.

from each other. Ok, we expect the following to result just in calls of `\true` (a call of `\false` is turned into a failed expectation):

```
11   \begin{qstest}{\Makematcher literal}{\MakeMatcher}
12     \begin{ExpectCallSequence}{\true{}\false{}'%
13         .#1{\Expect*{\CalledName#1}{true}}+'}
14     \stylefiles{{xxx/.thumbnails/blubb.sty}}{\true}{\false}
15     \truestylefiles{{xxx/.thumbnails/blubb.sty}}{\false}{\true}
16     \headbang{{xxx/.thumbnails/blubb.sty}}{\false}{\true}
17     \stylefiles{[ ]{x.sty.gz .thumbnails/x.sty !x}}{\true}{\false}
18     \truestylefiles{[ ]{x.sty.gz .thumbnails/x.sty !x}}{\false}{\true}
19     \headbang{[ ]{x.sty.gz .thumbnails/x.sty !x}}{\true}{\false}
20     \end{ExpectCallSequence}
21   \end{qstest}
```

So how do we create a sanitized keyword list in a control sequence?

`\MakeMatchTarget`   is called with two mandatory arguments, the first being a control sequence name where the keyword list in the second argument will get stored in a sanitized form: it is converted without expansion to characters of either "special" or "space" category (catcodes 12 and 10, respectively), and any leading spaces at the beginning of an element is removed. Without an optional bracketed argument, not more than sanitization and leading space stripping is done. If an optional bracketed argument before the mandatory arguments is specified, it defines the list separator: this has to be a single sanitized character token (either a space or a character of category "other") that is used as the list separator for the input (the finished list will actually always use the macro `\,` as a list separator).

```
22   \begin{qstest}{\Makematcher with \MakeMatchTarget}%
23                 {\MakeMatcher,\MakeMatchTarget}
24     \MakeMatchTarget\single{xxx/.thumbnails/blubb.sty}
25     \MakeMatchTarget[ ]\multiple{x.sty.gz .thumbnails/x.sty !x}
26     \begin{ExpectCallSequence}{\true{}\false{}'%
27         .#1{\Expect*{\CalledName#1}{true}}+'}
28     \stylefiles{\single}{\true}{\false}
29     \truestylefiles\single{\false}{\true}
30     \headbang\single{\false}{\true}
31     \stylefiles{\multiple}{\true}{\false}
32     \truestylefiles\multiple{\false}{\true}
33     \headbang\multiple{\true}{\false}
34     \end{ExpectCallSequence}
35   \end{qstest}
```

`\MatchedTarget`   This will after a match process contain the target matched by the last matching pattern (if several targets in a match target list match, only the first of those is considered and recorded), regardless of whether the corresponding pattern was negated with `!`.

`\RemoveMatched`   After a successful match, you can call `\RemoveMatched` with one argument: the control sequence name where the list was kept, and the match will get removed from the list. If every list element is removed, the list will be identical to `\@empty`.

```
36   \begin{qstest}{\MatchedTarget}
37                 {\MakeMatcher,\MakeMatchTarget,\MatchedTarget}
```

```
38     \MakeMatchTarget\single{xxx/.thumbnails/blubb.sty}
39     \MakeMatchTarget[ ]\multiple{x.sty.gz .thumbnails/x.sty !x}
40     \begin{ExpectCallSequence}{\true{}\false{}'%
41        .#1{\Expect*{\CalledName#1}{true}}+'}
42     \stylefiles{\single}{\true}{\false}
43     \Expect*{\single}{xxx/.thumbnails/blubb.sty}
44     \Expect*{\meaning\MatchedTarget}*{\meaning\single}
45     \RemoveMatched\single
46     \Expect*{\meaning\single}{macro:->}
47     \truestylefiles\single{\false}{\true}
48     \headbang\single{\false}{\true}
49     \stylefiles{\multiple}{\true}{\false}
50     \Expect*{\MatchedTarget}{.thumbnails/x.sty}
51     \RemoveMatched\multiple
52     \Expect\expandafter{\multiple}{x.sty.gz\,!x}
53     \truestylefiles\multiple{\false}{\true}
54     \Expect*{\meaning\MatchedTarget}{undefined}
55     \headbang\multiple{\true}{\false}
56     \Expect*{\MatchedTarget}{!x}
57     \RemoveMatched\multiple
58     \Expect*{\multiple}{x.sty.gz}
59   \end{ExpectCallSequence}
60  \end{qstest}
61 \end{qstest}
```

1.3   Notes on sanitization

Note that sanitization to printable characters has several consequences: it means that the control sequence `\,` will turn into the string `\` followed by the end of the keyword. Note also that single-character control sequences with a nonletter name are not followed by a space in sanitization. This means that sanitization depends on the current catcodes. Most particularly, sanitizing the input `\@abc12` will turn into `\@abc 12` when @ is of catcode letter, but to `\@abc12` when @ is a nonletter.

So sanitization cannot hide all effects of catcode differences. It is still essential since otherwise braces would cause rather severe complications during matching.

Another curiosity of sanitization is that explicit macro parameter characters (usually `#`) get duplicated while being sanitized.

So this is the end of the documentation section, so we end our test file setup by complementing the beginning:

```
62 ⟨/test⟩
63 ⟨*dtx⟩
64 \fi
65 ⟨/dtx⟩
```