makematch, a LaTeX package for pattern matching with wildcards

David Kastrup
David dot Kastrup (at) QuinScape dot de

Abstract

The makematch package has been factored out from the qstest package and has the purpose of matching patterns with wildcards against targets. There is a generalization provided for matching (ordered) pattern lists against (unordered) target lists, in which case one can use commata or other separators (including spaces) for separating the list elements. The wildcard * matches zero or more arbitrary characters. Prepending ! to a pattern will cause a match of it to revert possible matches from earlier in the pattern list.

Matching, for example, the pattern list

```
test*, !test10*b, !fails
```

with the target list

```
fails, test20
```

will lead to a non-match: while test20 is matched by the pattern test*, the additional matching pattern !fails later in the list reverts this match.

Both pattern and target lists get 'sanitized' (converted into a unique printing form where no TeX characters are interpreted specially) and compiled into a form which makes the matching itself quite efficient.

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 skips the tests when makematch.dtx is used as a standalone file.

```
⟨*dtx⟩
\iffalse
⟨/dtx⟩
⟨*test⟩
\RequirePackage{makematch,qstest}
\IncludeTests{*}
```

makematch requires LaTeX to be based on ε-TeX, 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 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).

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.

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

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

The `\MakeMatcher` command 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).

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

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

```
\begin{qstest}{\Makematcher literal}
              {\MakeMatcher}
  \begin{ExpectCallSequence}
     {\true{}\false{}%
     `.#1{\Expect*{\CalledName#1}{true}}+'}
    \stylefiles
       {{xxx/.thumbnails/blubb.sty}}
       {\true}{\false}
    \truestylefiles
      {{xxx/.thumbnails/blubb.sty}}
      {\false}{\true}
    \headbang
      {{xxx/.thumbnails/blubb.sty}}
      {\false}{\true}
    \stylefiles
      {[ ]{x.sty.gz .thumbnails/x.sty !x}}
      {\true}{\false}
    \truestylefiles
      {[ ]{x.sty.gz .thumbnails/x.sty !x}}
      {\false}{\true}
    \headbang
      {[ ]{x.sty.gz .thumbnails/x.sty !x}}
      {\true}{\false}
  \end{ExpectCallSequence}
\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 "other" or "space" category (catcodes 12 and 10, respectively), and any leading spaces at the beginning of an element are removed. Without an optional bracketed argument, nothing 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).

```
\begin{qstest}{\Makematcher%
               &\MakeMatchTarget}%
              {\MakeMatcher,%
               \MakeMatchTarget}
  \MakeMatchTarget\single
    {xxx/.thumbnails/blubb.sty}
  \MakeMatchTarget[ ]
    \multiple{x.sty.gz
      .thumbnails/x.sty !x}
  \begin{ExpectCallSequence}
       {\true{}\false{}%
         `.#1{\Expect*{\CalledName#1}
                      {true}}+'}
    \stylefiles{\single}{\true}{\false}
    \truestylefiles\single{\false}{\true}
    \headbang\single{\false}{\true}
    \stylefiles{\multiple}{\true}{\false}
    \truestylefiles\multiple{\false}{\true}
    \headbang\multiple{\true}{\false}
  \end{ExpectCallSequence}
\end{qstest}
```

After a match process `\MatchedTarget` will 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 `!`. 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`.

```
\begin{qstest}{\MatchedTarget}
              {\MakeMatcher,%
               \MakeMatchTarget,%
               \MatchedTarget}
  \MakeMatchTarget\single
    {xxx/.thumbnails/blubb.sty}
  \MakeMatchTarget[ ]\multiple
    {x.sty.gz .thumbnails/x.sty !x}
  \begin{ExpectCallSequence}
      {\true{}\false{}%
      `.#1{\Expect*{\CalledName#1}
                   {true}}+'}
    \stylefiles{\single}
      {\true}{\false}
    \Expect*{\single}
          {xxx/.thumbnails/blubb.sty}
    \Expect*{\meaning\MatchedTarget}
          *{\meaning\single}
    \RemoveMatched\single
    \Expect*{\meaning\single}
          {macro:->}
    \truestylefiles\single
       {\false}{\true}
    \headbang\single
       {\false}{\true}
    \stylefiles{\multiple}
       {\true}{\false}
    \Expect*{\MatchedTarget}
          {.thumbnails/x.sty}
    \RemoveMatched\multiple
    \Expect\expandafter{\multiple}
          {x.sty.gz\,!x}
    \truestylefiles\multiple
       {\false}{\true}
    \Expect*{\meaning\MatchedTarget}
          {undefined}
    \headbang\multiple
       {\true}{\false}
    \Expect*{\MatchedTarget}{!x}
    \RemoveMatched\multiple
    \Expect*{\multiple}{x.sty.gz}
  \end{ExpectCallSequence}
\end{qstest}
\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.

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

```
⟨/test⟩
⟨*dtx⟩
\fi
⟨/dtx⟩
```

2 Conclusions and outlook

`makematch` sets out to solve the task of pattern matching with wildcards in a very efficient manner. One basic restriction for some applications might be that it is restricted to comparing sanitized token lists. This has the effect that it is not possible to hide material from matching by enclosing it in braces. On the other hand, TEX will strip enclosing braces around a matched argument, making it unreliable to repeat matches or what to expect from a matched string.

In a later version, possibly starred forms of the commands will be provided that omit the sanitization. Those will not be able to match several characters with a meaning particular to TEX (such as `#`, `{` or `}`), but will probably come handy in other situations, like parsing keyword lists yielding TEX arguments. While it is possible to do this with the current code, using `\scantokens` for turning them active again, this can cause matches leading to unpaired braces, and it will not make it possible to hide commata from the matching by enclosing them in braces.