LaTeX document interface to the l3sys-query script: System queries for LaTeX using Lua*

LaTeX project

2024-03-28

Contents

1 Introduction **1**

2 The document level key-value interface **2**
2.1 Examples . . . 3

3 The L3 programming layer interface **3**

4 The `l3sys-query` Lua script **3**
4.1 The command line interface . . . 3
4.1.1 `ls [⟨args⟩]` . . . 4
4.1.2 `pwd` . . . 5
4.2 Spaces in arguments . . . 5
4.3 Wildcard expansion handling . . . 6

5 The LaTeX 2_ε package implementation **6**
5.1 `\QueryWorkingDirectory` . . . 6
5.2 `\QueryFiles` and `\QueryFilesTF` . . . 7

1 Introduction

TeX engines provide only very limited access to information about the system they are used on: using primitives, one can for example get the size of a single file, but not a list of files in a given location. For most documents, this is not an issue as they are self-contained. However, for cases where "dynamic" construction of parts of a document is needed based on file lists or other system-dependent data, methods to obtain this from (restricted) shell escape are desirable.

Security considerations mean that directly querying the system shell is problematic for general use. Instead, *restricted* shell escape may be used to get many details, provided a suitable tool is available to provide the information in a platform-neutral and security-conscious way. The Java program `texosquery`, written by Nicola Talbot, has been available for a number of years to provide this facility. As well as file system insight,

*This file has version number v1.0a, last revised 2024-03-28.

texosquery also provides for example locale data and other system information. However, the requirement for Java means that the script is not automatically usable when a TEX system is installed.

The LATEX team have therefore provided a Lua-based script, l3sys-query, which conforms to the security requirements of TEX Live using Lua to obtain the system information. This means that it can be used "out of the box" across platforms. The facilities provided by l3sys-query are more limited than texosquery, partly as some information is available in modern TEX systems using primitives, and partly as the aim of l3sys-query is to provide information where there are defined use cases. Requests for additional data interfaces are welcome.

This package provides a document level *Key-Value* interface to The l3sys-query functionality, although as a convenience it also summarizes the documentation for the L3 programming interface layer (prvided by the l3sys module of l3kernel, and of the command line Lua script l3sys-query which is distributed separately and provides the underlying functionality.

2 The document level key-value interface

This provides an interface to l3sys-query based on l3keys that allows the options to be specified and checked individually. Internally the supplied key values are used to build up the arguments to the commands described above in the L3 programming layer commands described in section 3

Results containing path separators *always* use /, irrespective of the platform in use.

\QueryWorkingDirectory `\QueryWorkingDirectory {⟨result cmd⟩}`

Defines supplied command ⟨*result cmd*⟩ to hold the absolute path to the current working directory of the TEX system. This is the directory (folder) from which TEX was started, so usually the location of the main document file.

\QueryFiles `\QueryFiles [⟨options⟩] {⟨spec⟩} {⟨function⟩}`
\QueryFilesTF `\QueryFilesTF [⟨options⟩] {⟨spec⟩} {⟨function⟩} {⟨pre code⟩} {⟨empty list code⟩}`

This generates a file list based on the ⟨*spec*⟩ and ⟨*options*⟩. The command then applies ⟨*function*⟩ to each item in the sequence of filenames. The ⟨*function*⟩ should be a macro body which will be passed the file path as #1.

The TF version executes the T (⟨*pre code*⟩) argument before iterating over the list and the F (⟨*empty list code*⟩) argument if the list is empty.

Note that this interface in mapping directly over the sequence of filenames does not allow some uses which are provided by the programming interface described in the following section, which allows the sequence to be manipulated before being used.

The defined keys map very closely to the options of the l3sys-query command which is described in section 4.

- The keys recursive, ignore-case, reverse, pattern take no values and map directly to the command line options of the same name.
- The key sort accepts the values date and name.
- The key type accepts d or f.

- The key `exclude` accepts a glob (or Lua pattern) matching files to be excluded. The package arranges that the quoting of the argument is automatically added if unrestricted shell escape is enabled.

Within the main ⟨*spec*⟩ argument, and the value of the `exclude` key, the following characters may need special handling. `~` may be used (which `kpathsea` uses to denote the users home directory). `\%`, or at the top level `%`, may be used to produce a literal `%` which may be especially useful if the `pattern` key is used as % is the escape character in Lua patterns.

2.1 Examples

- Include every `png` file in the current directory.

```
\QueryFiles{*.png}{\includegraphics{#1}\par}
```

- Input every TEX file with a filename matching `chapter[0-9].tex`.

```
\QueryFiles[pattern]{chapter%d.*%.tex}{\input{#1}}
```

3 The L3 programming layer interface

`\sys_get_query:nN`	`\sys_get_query:nN {⟨cmd⟩} {⟨tl var⟩}`
`\sys_get_query:nnN`	`\sys_get_query:nnN {⟨cmd⟩} {⟨spec⟩} {⟨tl var⟩}`
`\sys_get_query:nnnN`	`\sys_get_query:nnnN {⟨cmd⟩} {⟨options⟩} {⟨spec⟩} {⟨tl var⟩}`

Sets the ⟨*tl var*⟩ to the information returned by the `l3sys-query` ⟨*cmd*⟩, potentially supplying the ⟨*options*⟩ and ⟨*spec*⟩ to the query call. The valid ⟨*cmd*⟩ names are at present

- `pwd` Returns the absolute path to the current working directory
- `ls` Returns a directory listing, using the ⟨*spec*⟩ to select files and applying the ⟨*options*⟩ if given

The ⟨*spec*⟩ should be a file glob and will automatically be passed to the script without shell expansion.

`\sys_split_query:nN`	`\sys_split_query:nN {⟨cmd⟩} {⟨seq⟩}`
`\sys_split_query:nnN`	`\sys_split_query:nnN {⟨cmd⟩} {⟨spec⟩} {⟨seq⟩}`
`\sys_split_query:nnnN`	`\sys_split_query:nnnN {⟨cmd⟩} {⟨options⟩} {⟨spec⟩} {⟨seq⟩}`

Works as described for `\sys_get_query:nnnN`, but sets the ⟨*seq*⟩ to contain one entry for each line returned by `l3sys-query`.

4 The `l3sys-query` Lua script

4.1 The command line interface

The command line interface to

`l3sys-query` ⟨*cmd*⟩ [⟨*option(s)*⟩] [⟨*args*⟩]

where ⟨*cmd*⟩ can be one of the following:

- `ls`
- `ls` ⟨*args*⟩
- `pwd`

The ⟨*cmd*⟩ are described below. The result of the ⟨*cmd*⟩ will be printed to the terminal in an interactive run; in normal usage, this will be piped to the calling TEX process. Results containing path separators *always* use `/`, irrespective of the platform in use.

As well as these targets, the script recognizes the options

- `--exclude` Specification for directory entries to exclude
- `--ignore-case` Ignores case when sorting directory listings
- `--pattern` (`-p`) Treat the ⟨*args*⟩ as Lua patterns rather than converting from wildcards
- `--recursive` (`-r`) Enables recursive searching during directory listings
- `--reverse` Causes sorting to go from highest to lowest rather than lowest to highest
- `--sort` Sets the method used to sort entries returned by `ls`
- `--type` Selects the type of entry returned by `ls`

The action of these options on the appropriate ⟨*cmd(s)*⟩ is detailed below.

4.1.1 `ls` [⟨*args*⟩]

Lists the contents of one or more directories, in a manner somewhat reminiscent of the Unix command `ls` or the Windows command `dir`. The exact nature of the output will depend on the ⟨*args*⟩, if given, along with the prevailing options. Note that the options names are inspired by ideas from the Unix commands `ls` and `find` as well as the Windows command `dir`: they therefore do not map directly to those of any one of the command line tools that they somewhat mirror.

When no ⟨*args*⟩ are given, all entries in the current directory will be listed, one per line in the output. This will include both files and subdirectories. Each entry will include a path relative to the current directory: for files *in* the current directory, this will be `./`. The order of results will be determined by the underlying operating system process: unless requested *via* an option, no sorting takes place.

As standard, the ⟨*args*⟩ are treated as a file/path name potentially including `?` and `*` as wildcards, for example `*.png` or `file?.txt`.

```
l3sys-query ls '*.png'
```

Some care is needed in preventing expansion of such wildcards by the shell or `texlua` process: these are detailed in Section 4.3. In this section, `'` is used to indicate a character being used to suppress expansion: this is for example normal on macOS and Linux.

Removal of entries from the listing can be achieved using the `--exclude` option, which should be given with a ⟨*xarg*⟩, for example

```
l3sys-query ls --exclude '*.bak' 'graphics/*'
```

Directory entries starting `.` are traditionally hidden on Linux and macOS systems: these "dot" entries are excluded from the output of `l3sys-query`. The entries `.` and `..` for the current and parent directory are also excluded from the results returned by `l3sys-query` as they can always be assumed.

For more complex matching, the ⟨*args*⟩ can be treated as a Lua pattern using the `--pattern` (`-p`) option; this also applies to the ⟨*xarg*⟩ argument to the `--exclude` option. For example, the equivalent to wildcard `*.png` could be obtained using

```
l3sys-query ls --pattern '^.*%.png$'
```

The results returned by `ls` can be sorted using the `--sort` option. This can be set to `none` (use the order from the file system: the default), `name` (sort by file name) or `date` (sort by date last modified). The sorting order can be reversed using `--reverse`. Sorting normally takes account of case: this can be suppressed with the `--ignore-case` option.

The listing can be filtered based on the type of entry using the `--type` option. This takes a string argument, one of `d` (directory) or `f` (file).

As standard, only the path specified as part of the ⟨*args*⟩ is queried. However, if the `--recursive` (`-r`) option is set, the query is applied within all subdirectories. Subdirectories starting with `.` (macOS and Linux hidden) are excluded from recursion.

For security reasons, only paths within the current working directory can be queried, thus for example `graphics/*.png` will list all `png` files in the `graphics` subdirectory, but `../graphics/*.png` will yield no output.

4.1.2 `pwd`

Returns the current working directory from which `l3sys-query` is run. From within a TEX run, this will (usually) be the directory containing the main file, assuming a command such as

```
pdflatex main.tex
```

The `pwd` command is unaffected by any options.

4.2 Spaces in arguments

Since `l3sys-query` is intended primarily for use with restricted shell escape calls from TEX processes, handling of spaces is unusual. It is not possible to quote spaces in such a call, so for example whilst

```
l3sys-query ls "foo *"
```

does work from the command prompt to find all files with names starting `foo␣`, it would not work *via* restricted shell escape. To circumvent this, `l3sys-query` will collect all command line arguments after any ⟨*options*⟩, and combine these as a space-separated ⟨*args*⟩, for example allowing

```
l3sys-query ls foo '*'
```

to achieve the same result as the first example. The result is that the ⟨*args*⟩ will only every be interpreted by `l3sys-query` as a single argument. It also means that spaces cannot be used at the start or end of the argument, nor can multiple spaces appear between non-space arguments.

4.3 Wildcard expansion handling

The handling of wildcards needs some further comment for those using `l3sys-query` from the command line: the expl3 interface described in Section 3 handles this aspect automatically for the user.

On macOS and Linux, the shell normally expands globs, which include the wildcards `*` and `?`, before passing arguments to the appropriate command. This can be suppressed by surrounding the argument with `'` characters, hence the formulation

```
l3sys-query ls '*.png'
```

earlier.

On Windows, the shell does no expansion, and thus arguments are passed as-is to the relevant command. As such, `'` has no special meaning here. However, to allow quoting of wildcards from the shell in a platform-neutral manner, `l3sys-query` will strip exactly one set of `'` characters around each argument before further processing.

It is not possible to use `"` quotes at all in the argument passed to `l3sys-query` from TEX, as the TEX system removes all `"` in `\input` while handling space quoting.

Restricted shell escape prevents shell expansion of wildcards entirely. On non-Windows systems, it does this by ensuring that each argument is `'` quoted to ensure further expansion. Thus a TEX call such as

```
\input|"l3sys-query ls '*.png'"
```

will work if `--shell-escape` is used as the argument is passed directly to the shell, but in restricted shell escape will give an error such as:

```
! I can't find file `"|l3sys-query ls '*.png'"'.
```

The LATEX interfaces described above adjust the quoting used depending on the `shell-escape` status.

5 The LATEX 2_ε package implementation

```
⟨*package⟩
\ExplSyntaxOn
⟨@@=queryfiles⟩
```

The package should eventually work in restricted shell escape but will do nothing useful if the process was started with `--no-shell-escape`.

```
\sys_if_shell:F{
  \PackageWarningNoLine{l3sys-query}
  {Shell ~Escape ~is ~disabled.\MessageBreak All ~queries ~will ~return ~empty ~results}
  }
```

5.1 \QueryWorkingDirectory

`\QueryWorkingDirectory` is a direct call to the `pwd` command provided by `l3sys-query`.

```
\NewDocumentCommand\QueryWorkingDirectory {m} {
  \sys_get_query:nN {pwd} #1
}
```

5.2 \QueryFiles and \QueryFilesTF

The declarations of these commands are done in two steps to allow catcode chanes before the arguments are read. This allows the use of % and ^^ in patterns at least if the command is not nested in another command argument. (\% may be used to generate % in all cases).

Variables for saving the current definition of \% and ~.

```
\tl_new:N\l_query_percent_tl
\tl_new:N\l_query_tilde_tl
```

Allow % and ^^ at the top level.

```
\NewDocumentCommand\QueryFiles {} {
  \group_begin:
    \char_set_catcode_other:N \%
    \char_set_catcode_other:N \^
    \QueryFiles_inner
  }
\char_set_catcode_active:N \~
\NewDocumentCommand\QueryFiles_inner {O{}m}{
  \group_end:
  \tl_set:Nn\l_tmpa_tl{}
  \cs_set_eq:NN  \l_query_percent_tl \%
  \cs_set_eq:NN  \% \c_percent_str
  \cs_set_eq:NN  \l_query_tilde_tl ~
  \cs_set_eq:NN  ~ \c_tilde_str
  \keys_set:nn{QueryFiles}{#1}
  \exp_args:NnV\sys_split_query:nnnN {ls} \l_tmpa_tl {#2} \l_tmpa_seq
  \cs_set_eq:NN  \% \l_query_percent_tl
  \cs_set_eq:NN  ~ \l_query_tilde_tl
  \seq_map_inline:Nn\l_tmpa_seq
}
```

This duplicates rather than shares code so as to read the function and TF arguments with normal catcode regime. (This could probably be optimised.)

```
\NewDocumentCommand\QueryFilesTF {} {
  \group_begin:
    \char_set_catcode_other:N \%
    \QueryFilesTF_inner
 }
\NewDocumentCommand\QueryFilesTF_inner {O{}m}{
  \group_end:
  \tl_set:Nn\l_tmpa_tl{}
  \cs_set_eq:NN  \l_query_percent_tl \%
  \cs_set_eq:NN  \% \c_percent_str
  \cs_set_eq:NN  \l_query_tilde_tl ~
  \cs_set_eq:NN  ~ \c_tilde_str
  \keys_set:nn{QueryFiles}{#1}
  \exp_args:NnV\sys_split_query:nnnN {ls} \l_tmpa_tl {#2} \l_tmpa_seq
  \cs_set_eq:NN  \% \l_query_percent_tl
  \cs_set_eq:NN  ~ \l_query_tilde_tl
  \seq_if_empty:NTF \l_tmpa_seq \use_iii:nnn \__queryfiles_aux:nnn
}
\char_set_catcode_space:N \~
```

```
\cs_new:Npn  \__queryfiles_aux:nnn #1#2#3 {
    #2
    \seq_map_inline:Nn\l_tmpa_seq {#1}
 }
```

Defining the keys. Most take no value and simply add a `l3sys-query --` option to the command linebeing constructed.

```
\keys_define:nn {QueryFiles} {
recursive .code:n  =\tl_put_right:Nn \l_tmpa_tl {--recursive ~ } ,
recursive .value_forbidden:n = true ,
ignore-case .code:n  =\tl_put_right:Nn \l_tmpa_tl {--ignore-case ~ } ,
ignore-case .value_forbidden:n = true ,
reverse .code:n  =\tl_put_right:Nn \l_tmpa_tl {--reverse ~ } ,
reverse .value_forbidden:n = true ,
exclude .code:n  =\tl_put_right:Ne \l_tmpa_tl {
  --exclude ~
  \sys_if_shell_restricted:F'
  \exp_not:n{#1}
  \sys_if_shell_restricted:F'
  ~ } ,
exclude .value_required:n = true ,
```

The `type` key checks the supplied value is valid `d` for directories or `f` for files The default behaviour lists both.

```
type .choices:nn = {d,f}
                   {\tl_put_right:Nn \l_tmpa_tl {--type ~ #1 ~ }} ,
```

The `sort` key checks the supplied value is valid `date` or `name`.

```
sort .choices:nn = {date,name}
{\tl_put_right:Nn \l_tmpa_tl {--sort ~ #1 ~ }} ,
pattern .code:n  =\tl_put_right:Nn \l_tmpa_tl {--pattern ~ } ,
pattern .value_forbidden:n = true ,
}
\ExplSyntaxOff
```

⟨/package⟩