LaTeX Plugin for Sublime Text 2 and 3

by Ian Bacher and Marciano Siniscalchi

Marciano's blog:
http://tekonomist.wordpress.com

Additional contributors (*thank you thank you thank you*): first of all, Wallace Wu and Juerg Rast, who contributed code for multifile support in ref and cite completions, "new-style" ref/cite completion, and project file support. Also, skuroda (Preferences menu), Sam Finn (initial multifile support for the build command); Daniel Fleischhacker (Linux build fixes), Mads Mobaek (universal newline support), Stefan Ollinger (initial Linux support), RoyalTS (aka Tobias Schidt?) (help with bibtex regexes and citation code, various fixes), Juan Falgueras (latexmk option to handle non-ASCII paths), Jeremy Jay (basic biblatex support), Ray Fang (texttt snippet), Ulrich Gabor (tex engine selection and cleaning aux files), Wes Campaigne and 'jlegewie' (ref/cite completion 2.0!). **Huge** thanks to Daniel Shannon (aka phyllisstein) who first ported LaTeXTools to ST3. Also thanks for Charley Peng, who has been assisting users and generating great pull requests; I'll merge them as soon as possible. Also William Ledoux (various Windows fixes, env support), Sean Zhu (find Skim.app in non-standard locations), Maximilian Berger (new center/table snippet), Lucas Nanni (recursively delete temp files), Sergey Slipchenko (`$` auto-pairing with Vintage) Ian Bacher (use `kpsewhich` to find bib files in the `TEXMF` tree; reworked fill-all command; jump to tex files improvements; delete temp files improvements; builder options; improved LaTeX-cwl support), btstream (original fill-all command; LaTeX-cwl support), Richard Stein (auto-hide build panel, jump to included tex files, LaTeX-cwl support config, TEX spellcheck support, functions to analyze LaTeX documents, cache functionality), Dan Schrage (nobibliography command), PoByBolek (more biblatex command), Rafael Lerm (support for multiple lines in `\bibliography` commands).

If you have contributed and I haven't acknowledged you, email me!

Latest revision: v3.7.9 (2016-04-26).

Headline features:

- New viewers for Preview.app and Okular
- New bibliography parser available (see the settings file)
- Support for most citation commands, especially using BibLaTeX

Reminder: See the Settings section for details on the Settings system, which was updated in v3.6.1 (2016-01-01).

Note: If you've been following along with the v3.7.x release, v3.7.7 is a major change. I've restored the previous bibliography parsing from v3.6 and earlier as default. If you want to use the new parsing, you need to change the `bibliography` setting to `"new_bibliography"`. Thanks to everyone who has filed bug reports.

At present, all the reported bugs should be resolved, but I'm not certain that some don't remain (BibTeX is a wild world) and the older parsing seems to be substantially faster for those with extremely large bibliographies.

Contents

Introduction

This plugin provides several features that simplify working with LaTeX files:

- The ST build command takes care of compiling your LaTeX source to PDF using `texify` (Windows/MikTeX) or `latexmk` (OSX/MacTeX, Windows/-TeXlive, Linux/TeXlive). Then, it parses the log file and lists errors and warning. Finally, it launches (or refreshes) the PDF viewer (SumatraPDF on Windows, Skim on OSX, and Evince on Linux by default) and jumps to the current cursor position.
- Forward and inverse search with the named PDF previewers is fully supported
- Easy insertion of references and citations (from BibTeX files)
- Plugs into the "Goto anything" facility to make jumping to any section or label in your LaTeX file(s)
- Smart command completion for a variety of text and math commands is provided
- Additional snippets and commands are also provided
- The build command is fully customizable, as is the PDF previewer.

Requirements and Setup

First, you need to be running Sublime Text 2 or 3 (ST2 and ST3 henceforth, or simply ST to refer to either ST2 or ST3). For ST3, this has been tested against the latest beta builds.

Second, get the LaTeXTools plugin. These days, the easiest way to do so is via Package Control. See here for details on how to set it up (it's very easy). Once you have Package Control up and running, invoke it (via the Command Palette from the Tools menu, or from Preferences), select the Install Package command, and look for LaTeXTools.

If you prefer a more hands-on approach, you can always clone the git repository, or else just grab this plugin's .zip file from GitHub and extract it to your Packages directory (you can open it easily from ST, by clicking on **Preferences|Browse Packages**). Then, (re)launch ST. Please note that if you do a manual installation, the Package **must** be named "LaTeXTools".

I encourage you to install Package Control anyway, because it's awesome, and it makes it easy to keep your installed packages up-to-date (see the previously linked page for details).

Third, follow the OS-specific instructions below.

Finally, look at the section on Platform-Specific Settings and ensure that all your settings are correct. If you are running LaTeXTools for the first time, you may want to run the `LaTeXTools: Reset user settings to default` command from the Command Palette to get an editable copy of the settings file.

OS X

On **OSX**, you need to be running the MacTeX distribution (which is pretty much the only one available on the Mac anyway). Just download and install it in the usual way. I have tested MacTeX versions 2010–2014, both 32 and 64 bits; these work fine. MacTeX 2015 also works. On the other hand, MacTeX 2008 does *not* seem to work out of the box (compilation fails), so please upgrade.

We recommend that you also install the Skim PDF viewer, as this provides forward and inverse search and is the default viewer that LaTeXTools uses on OS X. If you don't install Skim, please see the section on Viewers below for details on how to setup a viewer.

Setup Skim.app

To configure inverse search, open the Preferences dialog of the Skim.app, select the Sync tab, then:

- uncheck the "Check for file changes" option
- choose the Sublime Text 2 or Sublime Text 3 preset (yes, Skim now supports both ST2 and ST3 by default!)

In case you are using an old version of Skim, you can always choose the Custom preset and enter (for ST3):

```
/Applications/Sublime Text.app/Contents/SharedSupport/bin/subl
```

in the Command field, and `"%file":%line` in the Arguments field. (This is correct as of 7/18/2013; you may want to double-check that ST3 is indeed in `/Applications /Sublime Text.app`; just go to the Applications folder in the Finder. Adapt as needed for ST2).

Setup LaTeXTools

Finally, edit the file `LaTeXTools.sublime-settings` in the `User` directory to make sure that the configuration reflects your preferred TeX distribution. Open the file and scroll down to the section titled "Platform settings." Look at the block for your OS, namely `"osx"`. Within that block, verify that the `"texpath"` setting is correct. Note that `"texpath"` **must** include `$PATH` somewhere.

Support for BasicTeX

If you don't want to install the entire MacTeX distro, which is pretty big, BasicTeX will also work (of course, as long as the LaTeX packages you need are included). **However**, you need to explicitly add the `latexmk` utility, which is not included by default: from the Terminal, type `sudo tlmgr install latexmk` (you will need to provide your password, assuming you are Administrator on your machine).

El Capitan

Sadly, with each OS X release, Apple deviates more and more from established Unix conventions. The latest "innovation" is that, beginning with El Capitan, applications can no longer write to `/usr`. MacTeX 2015 remedies this by creating a link to TeX binaries in `/Library/TeX`. The default LaTeXTools settings file now adds `/Library/TeX/texbin` to the `texpath`. In practice, this means the following.

Support for BasicTeX

If you don't want to install the entire MacTeX distro, which is pretty big, BasicTeX will also work (of course, as long as the LaTeX packages you need are included). **However**, you need to explicitly add the `latexmk` utility, which is not included by default: from the Terminal, type `sudo tlmgr install latexmk` (you will need to provide your password, assuming you are Administrator on your machine).

El Capitan

Sadly, with each OS X release, Apple deviates more and more from established Unix conventions. The latest "innovation" is that, beginning with El Capitan, applications can no longer write to `/usr`. MacTeX 2015 remedies this by creating

a link to TeX binaries in `/Library/TeX`. The default LaTeXTools settings file now adds `/Library/TeX/texbin` to the `texpath`. In practice, this means the following.

Windows

On **Windows**, both MikTeX and TeXLive are supported. Install either of these as usual.

We recommend that you install a version of Sumatra PDF viewer, as this is the only viewer currently supported on Windows. Its very light-weight and supports both forward and inverse search. Just download and install it in the normal way. You may have to add the SumatraPDF directory to your `PATH` environment variable or else set the `sumatra` command in the `windows` platform settings (see the section on platform settings below). If you choose not to install SumatraPDF, you might be able to use the `command` viewer to support another PDF viewer. See the Viewers section below for details.

Setup Sumatra

You now need to set up inverse search in Sumatra PDF. However, the GUI for doing this is hidden in Sumatra until you open a PDF file that has actual synchronization information (that is, an associated `.synctex.gz` file): see here. If you have one such file, then open it, go to **Settings|Options**, and enter `"C:\Program Files\Sublime Text 2\sublime_text.exe" "%f:%l"` for ST2, and `"C:\Program Files\Sublime Text 3\sublime_text.exe" "%f:%l"` for ST3, as the inverse-search command line (in the text-entry field at the bottom of the options dialog). If you don't already have a file with sync information, you can easily create one: compile any LaTex file you already have (or create a new one) with `pdflatex -synctex=1 <file.tex>`, and then open the resulting PDF file in SumatraPDF.

As an alternative, you can open a command-line console (run `cmd.exe`), and issue the following command (this assumes that SumatraPDF.exe is in your path; replace 3 with 2 for ST2 of course):

```
sumatrapdf.exe -inverse-search "\"C:\Program Files\Sublime Text 3\
    sublime_text.exe\" \"%f:%l\""
```

I'm sorry this is not straightforward—it's not my fault :-)

Setup LaTeXTools

Finally, edit the file `LaTeXTools.sublime-settings` in the `User` directory to make sure that the configuration reflects your preferred TeX distribution. Open the file and scroll down to the section titled "Platform settings." Look at the block for your OS, namely `windows`. Within that block, verify that the `texpath` setting is correct; for MiKTeX, you can leave this empty, i.e., `""`. If you do specify a path, note that it **must** include the system path variable, i.e., `$PATH` (this syntax seems

to be OK). Also verify that the `distro` setting is correct: the possible values are `"miktex"` and `"texlive"`.

TeXlive has one main advantage over MikTeX: it supports file names and paths with spaces.

`PATH` Issues

Recent versions of MikTeX add themselves to your path automatically, but in case the build system does not work, that's the first thing to check. TeXLive can also add itself to your path.

Linux

Linux support is coming along nicely. However, as a general rule, you will need to do some customization before things work. This is due to differences across distributions (a.k.a. "fragmentation"). Do not expect things to work out of the box.

Install TeXLive

You need to install TeXlive; if you are on Ubuntu, note that `apt-get install texlive` will get you a working but incomplete setup. In particular, it will *not* bring in `latexmk`, which is essential to LaTeXTools. You need to install it via `apt-get install latexmk`. If on the other hand you choose to install the TeXlive distro from TUG, `latexmk` comes with it, so you don't need to do anything else.

Setup LaTeXTools

You also need to edit the file `LaTeXTools.sublime-settings` in the `User` directory to make sure that the configuration reflects your preferred TeX distribution. Open that file and scroll down to the section titled "Platform settings." Look at the block for your OS, namely `"linux"`. Within that block, verify that the `"texpath"` setting is correct. Notice that this **must** include `$PATH` somewhere, or things will not work.

You may also have to set the `command` option in `"builder_settings"`, which tells the builder how to invoke `latexmk`. By default (i.e., if `command` is empty or not given) it is `["latexmk", "-cd", "-e", "-f", "-pdf", "-interaction=nonstopmode", "-synctex=1"]`.

If you customize the command to include a custom PDF command, users have reported the following possible issues and fixes (thanks!), so if you get a "Cannot compile!" error, try the following:

- some distros do *not* want a space before and after the `=` in `$pdflatex = %E`. But some *do* want the space there (sigh!)
- sometimes `latexmk` is not on the `PATH`, or the path is not correctly picked up by ST. In this case, instead of `"latexmk"`, use `"/usr/bin/latexmk"` or wherever `latexmk` is in your system.
- some distros require quoting the `$pdflatex` assignment, as in `"$pdflatex = \"'%E -interaction=nonstopmode -synctex=1 %S %O'\""`

There are several variants to deal with; each distro is a little bit different, so there are basically no universal defaults. There's not much I can do about it. Good luck!

Also, to get inverse search working on ST3, make sure you set the `sublime` option in `LaTeXTools.sublime-settings` correctly; the Ubuntu package from the ST web page uses `subl`, but check from the command line first.

Setup Evince

By default LaTeXTools assumes you are using Evince (Document Viewer) as your PDF viewer. Support is also available for Okular and other viewers that can be run via the command line. See the section on Viewers below for details on how to setup other viewers.

If you opt to use Evince, which is installed by default on Ubuntu and any distro that provides the Gnome desktop, you don't need to configure anything. Backward and forward search Work For Me (TM). Hopefully they will work for you, too, but let me know if this is not the case.

General Features

Project Files

Project files are fully supported! You should consult the subsection on project-specific settings for further details.

Multi-file documents

Multi-file documents are supported as follows. If the first line in the current file consists of the text `%!TEX root = <master file name>`, then tex & friends are invoked on the specified master file, instead of the current one. Note: the only file that gets saved automatically is the current one. Also, the master file name **must** have a valid tex extension (i.e., one configured in the `tex_file_exts` settings), or it won't be recognized.

As an alternative, to using the `%!TEX root = <master file name>` syntax, if you use a Sublime project, you can set the `TEXroot` option (under `settings`):

```
{
    ... <folder-related settings> ...

    "settings": {
        "TEXroot": "yourfilename.tex"
    }
}
```

Note that if you specify a relative path as the `TEXroot` in the project file, the path is determined *relative to the location of the project file itself*. It may be less ambiguous to specify an absolute path to the `TEXroot` if possible.

Spell-checking

LaTeXTools parses the `%!TEX spellcheck` directive to set the language for the spell-checker integrated in Sublime Text. The Dictionaries package is recommended and supported. If you have additional dictionaries, you can add them using the `tex_spellcheck_paths` setting, which is a mapping from the locales to the dictionary paths. Each locale must be lowercase and use only a hyphen as a separator. The dictionary paths must be compatible with those used by Sublime Text's spell-checker. For example `{"en-us": "Packages/Language - English/en_US.dic"}` would be a valid value. For more on Sublime Text support for spell checking, see the relevant online documentation.

Support for non-`.tex` files

LaTeXTools has some degree of support for LaTeX documents that are in files with an extension other than `.tex`. In particular, this feature is designed to work well with alternative extensions, such as `.ltx`. Other extensions such as `.Rnw` and `.tikz` are supported, but, for now they will be treated as standard LaTeX documents (patches are always welcome!).

This behaviour is controlled through two settings, `tex_file_exts` and `latextools_set_syntax`. For more on these settings, see the documentation on General Settings.

Note that while the extension detection is used by features of LaTeXTools, including, other features—especially the completions—depend on the syntax of the file being set to `LaTeX` as well.

Package Documentation

You can consult the documentation for any LaTeX package by invoking the `View Package Documentation` command via the Command Palette (for now). This relies on your system's `texdoc` command.

Caching

LaTeXTools uses a cache to store relevant information about your document and improve the performance of commands. The contents of this cache may become outdated. If a command seems to be returning old data, simply clear the cache using either the command delete temporary files or the dedicated command to clear the cache. For more details on the cache, see the Cache settings section below and the section on the Cache.

Builder Features

Most of the builder features are controlled through the `LaTeXTools.sublime-settings` file. See, in particular, the section on builder settings.

Default Builder

The default builder (called the `traditional` builder) supports several additional features.

TeX Engine Selection

If the first line of the current file consists of the text `%!TEX program = <program>`, where `program` is `pdflatex`, `lualatex` or `xelatex`, the corresponding engine is selected. If no such directive is specified, `pdflatex` is the default. Multi-file documents are supported: the directive must be in the *root* (i.e. master) file. Also, for compatibility with TeXshop, you can use `TS-program` instead of `program`. **Note**: for this to work, you must **not** customize the `command` option in `LaTeXTools.sublime-settings`. If you do, you will not get this functionality. Finally, if you use project files, the `program` builder setting can also be customized there, under `settings`.

TeX Options

You can pass command-line options to your engine in two ways (thanks Ian Bacher!). One is to use a `%!TEX options = ...` line at the top of your file. The other is to use the `options` builder setting in your settings file. This can be useful, for instance, if you need to allow shell escape. Finally, if you use project files, the `options` builder setting can also be customized there (again, under `settings`).

Customizing the compilation command

It is possible to customize the command run by setting the `command` option under Builder Settings. See the section on Builder Settings for details.

Note: If you customize the command, the TeX engine selection facility may no longer work because it relies on a specific compilation command. However, if you want to customize or replace `latexmk`/`texify`, you probably know how to select the right TeX engine, so this shouldn't be a concern. Also note that if you are using `latexmk` and you set the `$pdflatex` variable, the TeX options facility will not function, as `latexmk` does not support this.

If you change the compilation command, you are responsible for making it work on your setup. Only customize the compilation command if you know what you're doing.

Other Builders

If the default builder doesn't meet your needs for any reason, please see the section on Alternative Builders below.

Viewers

By default, LaTeXTools supports the following viewers, depending on platform:

- On OS X, Skim
- On Windows, Sumatra
- On Linux, Evince

However, it is possible to use other programs to view PDF files. Currently the only non-default viewers supported are Preview.appp on OS X and Okular on Linux (patches welcome!). Preview can be selected by changing the `viewer` setting in your LaTeXTools preferences to `"preview"`. Okular can be used by changing the `viewer` setting in your LaTeXTools preferences to `"okular"`. See the Viewer Settings section for more details. For a manner of supporting other viewers, please see the section on Alternate Viewers below.

Keybindings

Keybindings have been chosen to make them easier to remember, and also to minimize clashes with existing (and standard) ST bindings. I am taking advantage of the fact that ST supports key combinations, i.e. sequences of two (or more) keys. The basic principle is simple:

- **Most LaTeXTools facilities are triggered using `Ctrl+l` (Windows, Linux) or `Cmd+l` (OS X), followed by some other key or key combination**
- Compilation uses the standard ST "build" keybinding, i.e. `Ctrl-b` on Windows and Linux and `Cmd-b` on OS X. So does the "goto anything" facility (though this may change).

For example: to jump to the point in the PDF file corresponding to the current cursor position, use `Ctrl-l, j`: that is, hit `Ctrl-l`, then release both the `Ctrl` and the `l` keys, and quickly type the `j` key (OS X users: replace `Ctrl` with `Cmd`). To wrap the selected text in an `\emph{}` command, use `Ctrl-l, Ctrl-e`: that is, hit `Ctrl-l`, release both keys, then hit `Ctrl-e` (again, OS X users hit `Cmd-l` and then `Cmd-e`).

`Ctrl-l` (`Cmd-l` on OS X) is the standard ST keybinding for "expand selection to line"; this is **remapped** to `Ctrl-l,Ctrl-l` (`Cmd-l,Cmd-l` on OS X). This is the *only* standard ST keybinding that is affected by the plugin—an advantage of new-style keybindings.

Most plugin facilities are invoked using sequences of 2 keys or key combinations, as in the examples just given. A few use sequences of 3 keys or key combinations.

Henceforth, I will write `C-` to mean `Ctrl-` for Linux or Windows, and `Cmd-` for OS X. You know your platform, so you know what you should use. In a few places, to avoid ambiguities, I will spell out which key I mean.

Compiling LaTeX files

Keybinding: `C-b` (standard ST keybinding)

LaTeXTools offers a fully customizable build process. This section describes the default process, also called "traditional" because it is the same (with minor tweaks) as the one used in previous releases. However, see below for how to customize the build process.

The default ST Build command takes care of the following:

- It saves the current file
- It invokes the tex build command (`texify` for MikTeX; `latexmk` for TeXlive and MacTeX).
- It parses the tex log file and lists all errors, warnings and, if enabled, bad boxes in an output panel at the bottom of the ST window: click on any error/warning/bad boxes to jump to the corresponding line in the text, or use the ST-standard Next Error/Previous Error commands.
- It invokes the PDF viewer for your platform and performs a forward search: that is, it displays the PDF page where the text corresponding to the current cursor position is located.

Toggling window focus following a build

Keybinding: `C-l,t,f` (yes, this means `C-l`, then `t`, then `f`)

By default, after compilation, the focus stays on the ST window. This is convenient if you like to work with the editor and PDF viewer window open side by side, and just glance at the PDF output to make sure that all is OK. If

however the editor and viewer windows overlap (e.g. if you have a small screen), you may prefer the viewer window to get the focus (i.e. become the foremost window) after compiling. To this end, you can use the `toggle_focus` command to change this behavior. The first time you invoke this command, the focus will shift to the viewer window after compiling the current file; if you invoke the command again, the post-compilation focus reverts to the editor window. Every time you invoke `toggle_focus`, a message will appear in the status bar.

You can change the default focus behavior via the `keep_focus` option: see the "Settings" section below.

Toggling PDF syncing (forward search) following a build

Keybinding: `C-l,t,s`

By default, after compilation, LaTeXTools performs a 'forward search' so that the PDF viewer shows the point in the PDF file corresponding to the current cursor position in ST (by the way, you can trigger a forward search at any other time, not just when you are compiling: see below). If for whatever reason you don't like this behavior, you can turn it off using the `toggle_fwdsync` command. As for `toggle_focus`, a message will appear in the status bar to reflect this.

You can also change the default sync behavior via the `forward_sync` option: see the "Settings" section below.

Checking the status of toggles and defaults

Keybinding: `C-l,t,?`

This causes the status message to list the default settings of the focus and sync options, and their current toggle values. It also display the status of the ref/cite auto trigger toggles (see below).

Removing temporary files from build

Keybinding: `C-l,backspace`

This deletes all temporary files from a previous build (the PDF file is kept). Subfolders are traversed recursively. This command also clears the LaTeXTools cache.

Two settings allow you to fine-tune the behavior of this command. `temp_files_exts` allows you to specify which file extensions should be considered temporary, and hence deleted. `temp_files_ignored_folders` allows you to specify folders that should not be traversed. A good example are `.git` folders, for people who use git for version control.

NOTE: If you use the output directory feature, the above is ignored, and the entire output_directory is simply deleted. If you are using the auxiliary directory feature *without* using an output directory, the auxiliary directory will be cleared and the normal process will be run.

Clearing the cache

Keybinding: `C-l,C-d,C-c`

This clears the LaTeXTools cache. It is useful if the LaTeXTools cache information gets too out of date, but you want to maintain the LaTeX build files, such as `.aux`.

Forward and Inverse Search

Keybinding: `C-l,j` (for forward search; inverse search depends on the previewer)

When working in an ST view on a TeX document, `C-l,j` will display the PDF page where the text corresponding to the current cursor position is located; this is called a "forward search". The focus is controlled by the `C-l,t,f` toggle setting and the `keep_focus` option.

If you are viewing a PDF file, then hitting `CMD+Shift+Click` in Skim (OSX), double-clicking in Sumatra (Windows), or hitting `Ctrl+click` in Evince (Linux) will bring you to the location in the source tex file corresponding to the PDF text you clicked on. This is called "inverse search".

To open a PDF file without performing a forward search, use `C-l,v`.

For support of forward and inverse search in other viewers, see the viewer section below.

References and Citations

Keybinding: *autotriggered* by default (see below). Otherwise, `C-l,x` for 'cross-reference,' or `C-l,C-f` (via the Fill Helper facility: see below). These are fully equivalent ways of invoking ref/cite completions.

The basic idea is to help you insert labels in `\ref{}` commands and bibtex keys in `\cite{}` commands. The appropriate key combination shows a list of available labels or keys, and you can easily select the appropriate one. Full filtering facilities are provided.

Notes:

1. In order to find all applicable labels and bibtex keys, the plugin looks at the **saved** file. So, if you invoke this command and do not see the label or key you just entered, perhaps you haven't saved the file.

2. Only bibliographies in external `.bib` files are supported: no `\bibitem...`. Sorry. It's hard as it is.

3. Multi-file documents are fully supported.

Now for the details. (Many of these features were contributed by Wes Campaigne and jlewegie, whom I thank profusely.)

By default, as soon as you type, for example, `\ref{` or `\cite{`, a quick panel is shown (this is the fancy drop-down list ST displays at the top of the screen), listing, respectively, all the labels in your files, or all the entries in the bibliographies you reference your file(s) using the `\bibliography{}` command. This is the default *auto-trigger* behavior, and it can be a big time saver. You can, however, turn it off, either temporarily using a toggle, or permanently by way of preference settings: see below. Once the quick panel is shown, you can narrow down the entries shown by typing a few characters. As with any ST quick panel, what you type will be fuzzy-matched against the label names or, for citations, the content of the first displayed line in each entry (by default, the author names, year of publication, short title and citation key: see below). This is *wildly* convenient, and one of the best ST features: try it!

If auto-triggering is off, when you type e.g. `\ref{`, ST helpfully provides the closing brace, leaving your cursor between the two braces. Now, you need to type `C-l,x` to get the quick panel showing all labels in the current file. You can also type e.g. `\ref{aa` [again, the closing brace is provided by ST], then `C-l, x`, and LaTeXTools will show a list of labels that fuzzy-match the string `aa`.

In either case, you then select the label you want, hit Return, and LaTeXTools inserts the **full ref command**, as in `\ref{my-label}`. The LaTeX command `\eqref` works the same way. Citations from bibtex files are also supported in a similar way. Use `\cite{}`, `\citet{}`, `\citeyear{}` etc.

One often needs to enter **multiple citations**, as e.g. in `\cite{paper1,paper2}`. This is easy to do: either cite the first paper, e.g. `\cite{paper1}` and then, *with your cursor immediately before the right brace*, type a comma (`,`). Again, the default auto-trigger behavior is that the quick panel with appear, and you can select the second paper. If auto-trigger is off, then you enter the comma, then use the shortcut `C-l,x` to bring up the quick panel (note: you *must* add the comma before invoking the shortcut, or you won't get the intended result). Of course, you can enter as many citations as you want.

The display of bibliographic entries is *customizable*. There is a setting, `cite-panel-format`, that controls exactly what to display in each of the two lines each entry gets in the citation quick panel. Options include author, title, short title, year, bibtex key, and journal. This is useful because people may prefer to use different strategies to refer to papers—author-year, short title-year, bibtex key (!), etc. Since only the first line in each quick panel entry is searchable, how you present the information matters. The default should be useful for most people; if you wish to change the format, check the `LaTeXTools.sublime-settings` file for

detailed information. (As usual, copy that file to the `User` directory and edit your copy, not the original).

Thanks to recent contributed code, **multi-file documents** are *fully supported*. LaTeXTools looks for references, as well as `\bibliography{}` commands, in the root file and in all recursively included files. Please see the information on **Multi-file documents** in the section on **General Features** for details on how to setup multi-file documents.

LaTeXTools now also looks `\addbibresource{}` commands, which provides basic compatibility with biblatex.

Old-style, deprecated functionality

For now, completions are also injected into the standard ST autocompletion system. Thus, if you hit `Ctrl-space` immediately after typing, e.g., `\ref{}`, you get a drop-down menu at the current cursor position (not a quick-panel) showing all labels in your document. However, the width of this menu is OK for (most) labels, but not really for paper titles. In other words, it is workable for references, but not really for citations. Furthermore, there are other limitations dictated by the ST autocompletion system. So, this is **deprecated**, and I encourage you to use auto-trigger mode or the `C-l,x` or `C-l,C-f` keybindings instead.

Toggle auto trigger mode on/off

Keybinding: `C-l,t,a,r` for references; `C-l,t,a,c` for citations

These toggles work just like the sync and focus toggles above. Indeed, `C-l,t,?` will now also display the status of the auto trigger toggles. Check the status bar for feedback (i.e. to see what the current state of the toggle is), but remember the message stays on for only a few seconds. `C-l,t,?` is your friend.

Fill Helper: filling in package and file names automatically

Keybinding: *autotriggered* by default (see below). Otherwise, `C-l,C-f`.

Thanks to the amazing work by users btstream and Ian Bacher, LaTeXTools now offers a list of available files and packages when using commands such as `\usepackage`, `\include`, `\includegraphics`, `\includesvg` and `\input`. Assuming auto-completion is toggled on (the default):

- when you type `\usepackage{`, a list of available package is displayed in the ST drop-down menu. Pick the one you need, and it will be inserted in your file, with a closing brace.

- when you type any of the file-related input commands, a list of files in the current directory is displayed (suitably filtered, so graphics files are displayed for `\includegraphics`).

To toggle autocompletion on or off, use the `fill_auto_trigger` setting, or the `c-l,t,a,f` toggle.

In order for package autocomplete to work, you need to create a cache first. You can do it using the Command Palette: select `LaTeXtools: Build cache for LaTeX packages`.

The `C-l,C-f` keyboard shortcut also works for `\ref` and `\cite` completion. Basically, wherever you can use `C-l,x`, you can also use `C-l,C-f`.

Jumping to sections and labels

Keybinding: `C-r` (standard ST keybinding)

The LaTeXtools plugin integrates with the awesome ST "Goto Anything" facility. Hit `C-r`to get a list of all section headings, and all labels. You can filter by typing a few initial letters. Note that section headings are preceded by the letter "S", and labels by "L"; so, if you only want section headings, type "S" when the drop-down list appears.

Selecting any entry in the list will take you to the corresponding place in the text.

Jumping to included files

Keybinding: `C-l, C-o` (only works if the cursor is in the same line as the include command)

LaTeX files

To open a LaTeX file, which is included with `\input`, `\include` or `\subfile` just position the cursor inside the include command and press `C-l, C-o`. This will open the included file in Sublime Text.

If necessary missing folders and the file will be created. In this case the magic root entry will be written into the file. Hence this command can be used to comfortably create files and open files.

Image files

To open an image, which is included with `\includegraphics` just position the cursor inside the command and press `C-l, C-o`. This will open the image.

The program to open the image can be configured in the LaTeXTools settings in the `open_image_command` attribute.

The following settings are provided:

- `image_types`: a list of the image file types used in the `\includegraphics` command. This list is also used in the Fill Helper and to determine missing extensions to open images. When opening an image the `image_types`-list will be matched from left to right.
- `open_image_command`: the command/program to open an image used in the `\includegraphics` command. This commands can be configured OS-specific. For each OS you can create a list, which will be searched top-down for the matching extension. Each entry in the list has a `command` and `extension` field. The command is a string and will be executed with the file path appended, if the extension matches the extension of the file. You can optionally use `$file` inside the string to insert the file path at an arbitrary position. The `extension` can either be a string or a list of string. If it is missing, the command will be executed for every file type.

LaTeX commands and environments

Keybindings: `C-l,c` for commands and `C-l,e` for environments

To insert a LaTeX command such as `\color{}` or similar, type the command without backslash (i.e. `color`), then hit `C-l,c`. This will replace e.g. `color` with `\color{}` and place the cursor between the braces. Type the argument of the command, then hit Tab to exit the braces.

Similarly, typing `C-l,e` gives you an environment: e.g. `test` becomes

```
\begin{test}

\end{test}
```

and the cursor is placed inside the environment thus created. Again, Tab exits the environment.

Note that all these commands are undoable: thus, if e.g. you accidentally hit `C-l,c` but you really meant `C-l,e`, a quick `C-z`, followed by `C-l,e`, will fix things.

Wrapping existing text in commands and environments

Keybindings: `C-l,C-c`, `C-l, C-n`, etc.

The tab-triggered functionality just described is mostly useful if you are creating a command or environment from scratch. However, you sometimes have existing text, and just want to apply some formatting to it via a LaTeX command or environment, such as `\emph` or `\begin{theorem}...\end{theorem}`.

LaTeXTools' wrapping facility helps you in just these circumstances. All commands below are activated via a key binding, and *require some text to be selected first*. Also, as a mnemonic aid, *all wrapping commands involve typing `C-l,C-something` (which you can achieve by just holding the `C-` key down after typing `l`).

- `C-l,C-c` wraps the selected text in a LaTeX command structure. If the currently selected text is `blah`, you get `\cmd{blah}`, and the letters `cmd` are highlighted. Replace them with whatever you want, then hit Tab: the cursor will move to the end of the command.
- `C-l,C-e` gives you `\emph{blah}`, and the cursor moves to the end of the command.
- `C-l,C-b` gives you `\textbf{blah}`
- `C-l,C-u` gives you `\underline{blah}`
- `C-l,C-t` gives you `\texttt{blah}`
- `C-l,C-n` wraps the selected text in a LaTeX environment structure. You get `\begin{env}`,`blah`, `\end{env}` on three separate lines, with `env` selected. Change `env` to whatever environment you want, then hit Tab to move to the end of the environment.

These commands also work if there is no selection. In this case, they try to do the right thing; for example, `C-l,C-e` gives `\emph{}` with the cursor between the curly braces.

You can also *change the current environment* using the `C-l,C-Shift-n` shortcut. Note well how this works. First, the cursor must be inside the environment you are interested in. Second, the command selects the environment name in the `\begin{env}` command and also in the `\end{env}` command (using ST's multiple-selection support). This way you can rename the environment as needed. *Remember to exit multiple-selection mode* when you are done by pressing the `ESC` key.

Completions

Completions

Command completion, snippets, etc.

By default, ST provides a number of snippets for LaTeX editing; the LaTeXTools plugin adds a few more. You can see what they are, and experiment, by selecting **Tools|Snippets|LaTeX** and **Tools|Snippets|LaTeXTools** from the menu.

In addition, the LaTeXTools plugin provides useful completions for both regular and math text; check out files `LaTeX.sublime-completions` and `LaTeX math.sublime-completions` in the LaTeXTools directory for details. Some of these are semi-intelligent: i.e. `bf` expands to `\textbf{}` if you are typing text, and to `\mathbf{}` if you are in math mode. Others allow you to cycle among different completions:

e.g. `f` in math mode expands to `\phi` first, but if you hit Tab again you get `\varphi`; if you hit Tab a third time, you get back `\phi`.

LaTeX-cwl support

LaTeXTools supports the `LaTeX-cwl` autocompletion package. If the package is installed, support is automatically enabled. By default, as soon as one types, e.g., `\te`, a popup is shown displaying possible completions, including e.g. `\textit` and the like.

The following settings are provided:

- `cwl_list`: a list of paths to the `cwl` files
- `command_completion`: when to show that cwl completion popup. The possible values are:
  - `prefixed` (default): show completions only if the current word is prefixed with a `\`
  - `always`: always show cwl completions
  - `never`: never display the popup
- `env_auto_trigger`: if `true`, autocomplete environment names upon typing `\begin{` or `\end{` (default: `false`)

Settings

LaTeXTools supports user-defined settings. The settings file is called `LaTeXTools .sublime-settings`. A default version resides in the LaTeXTools plugin directory and **must not be edited**. This contains default settings that will work in many cases, for standard configurations of TeX distros and PDF viewers. You can however create another settings file in your `User` directory; again, the file must be named `LaTeXTools.sublime-settings`.

You can create and edit such a file manually. It is a standard Sublime Text JSON file; the settings currently honored by LaTeXTools are listed below. However, the *simplest way to create a settings file* is to open the **Preferences | Package Settings | LaTeXTools** submenu and select the **Settings - User** option. If you do not currently have a `LaTeXTools.sublime-settings` settings file in your `User` directory (e.g., if you are installing LaTeXTools for the first time), you will be given the option to create one. The newly created settings file will be an exact copy of the default one, and will open in a tab for you to customize.

If you *do* already have an existing `LaTeXTools.sublime-settings` file in your `User` directory, the `Settings - User` option will open that file in a tab for you to further customize. Similarly, the `Settings - Default` option will open the default settings file in a tab, in *read-only mode*. This may be useful for you to copy from, or if you want to see what other options may be available without consulting this README file.

If at any time you wish to erase your customizations and start afresh, you can simply delete the `LaTeXTools.sublime-settings` file in your `User` directory. (Again, *warning*: do *not* touch the settings file in the LaTeXTools plugin directory!) Alternatively, from the **Preferences | Package Settings | LaTeXTools** submenu, or from the Command Palette, you can choose **Reset user settings to default**. This will delete any existing settings file in `User` and create a copy of the default one. This will *remove all your customizations.*

(Historical note: This is no longer relevant in 2016, but just for the record, if you have a pre-2014, old-style settings file, this option will import it).

Warning: in general, tweaking options can cause breakage. For instance, if on Linux you change the default `python2` setting (empty by default) to a non-existent binary, forward and inverse search will stop working. With great power comes great responsibility! If you think you have found a bug, *delete your settings file in the `User` directory, or use the **Reset user settings to default** command before reporting it!* Thanks :-)

The following options are currently available (defaults in parentheses):

General Settings

- `cite_auto_trigger` (`true`): if `true`, typing e.g. `\cite{` brings up the citation completion quick panel, without the need to type `C-l,x`. If `false`, you must explicitly type `C-l,x`.
- `ref_auto_trigger` (`true`): ditto, but for `\ref{` and similar reference commands
- `fill_auto_trigger` (`true`): ditto, but for package and file inclusion commands (see Fill Helper feature above)
- `cwl_autoload` (`true`): whether to load cwl completions based on packages (see the LaTeX-cwl feature)
- `cwl_completion` (`prefixed`): when to activate the cwl completion poput (see LaTeX-cwl feature above)
- `cwl_list` (`[ "tex.cwl", "latex-209.cwl", "latex-document.cwl", "latex-l2tabu .cwl", "latex-mathsymbols.cwl"]`): list of cwl files to load
- `keep_focus` (`true`): if `true`, after compiling a tex file, ST retains the focus; if `false`, the PDF viewer gets the focus. Also note that you can *temporarily* toggle this behavior with `C-l,t,f`. **Note**: If you are on either Windows or Linux you may need to adjust the `sublime_executable` setting for this to work properly. See the **Platform settings** below.
- `forward_sync` (`true`): if `true`, after compiling a tex file, the PDF viewer is asked to sync to the position corresponding to the current cursor location in ST. You can also *temporarily* toggle this behavior with `C-l,t,s`.
- `aux_directory` (`""`): specifies the auxiliary directory to store any auxiliary files generated during a LaTeX build. Note that the auxiliary directory option is only useful if you are using MiKTeX. Path can be specified using either an absolute path or a relative path. If `aux_directory` is set from the

project file, a relative path will be interpreted as relative to the project file. If it is set in the settings file, it will be interpreted relative to the main tex file. In addition, the following special values are honored:

- `<<temp>>`: uses a temporary directory in the system temp directory instead of a specified path; this directory will be unique to each main file, but does not persist across restarts.
- `<<cache>>`: uses the ST cache directory (or a suitable directory on ST2) to store the output files; unlike the `<<temp>>` option, this directory can persist across restarts.
- `<<project>>`: uses a sub-directory in the same folder as the main tex file with what should be a unique name; note, this is probably not all that useful and you're better off using one of the other two options or a named relative path
- `output_directory` (`""`): specifies the output directory to store any file generated during a LaTeX build. Path can be specified using either an absolute path or a relative path. If `output_directory` is set from the project file, a relative path will be interpreted as relative to the project file. If it is set in the settings file, it will be interpreted relative to the main tex file. In addition, output_directory honors the same special values as `auxiliary_directory`.
- `copy_output_on_build` (`true`): if `true` and you are using an `output_directory`, either set via the setting or the `%!TEX` directive, this instructs LaTeXTools to copy to resulting pdf to the same folder as the main tex file. If you are not using `output_directory` or it is set to `false`, it does nothing. If it is a list of extensions, it will copy each file with the same name as your main tex file and the given extension to the same folder as your main tex file. This is useful for copying, e.g., .synctex.gz or .log files.
- `temp_files_exts`: list of file extensions to be considered temporary, and hence deleted using the `C-l, backspace` command.
- `temp_files_ignored_folders`: subdirectories to skip when deleting temp files.
- `tex_file_exts` (`['.tex']`): a list of extensions that should be considered TeX documents. Any extensions in this list will be treated exactly the same as `.tex` files. See the section on Support for non-`.tex` files.
- `latextools_set_syntax` (`true`): if `true` LaTeXTools will automatically set the syntax to `LaTeX` when opening or saving any file with an extension in the `tex_file_exts` list.
- `tex_spellcheck_paths` (`{}`): A mapping from the locales to the paths of the dictionaries. See the section Spell-checking.

Platform-Specific Settings

This section refers to setting that can be found in a platform-specific block for each platform, i.e., `"osx"`, `"windows"`, or `"linux"`.

All Platforms

- `texpath` (varies): the path to TeX & friends

Windows

- `distro` (`miktex`): either `miktex` or `texlive`, depending on your TeX distribution
- `sumatra` (`""`): leave blank or omit if the SumatraPDF executable is in your `PATH` and is called `SumatraPDF.exe`, as in a default installation; otherwise, specify the *full path and file name* of the SumatraPDF executable.
- `sublime_executable` (`""`): this is used if `keep_focus` is set to true and the path to your sublime_text executable cannot be discovered automatically. It should point to the full path to your executable `sublime_text.exe`.
- `keep_focus_delay` (`0.5`): this is used if `keep_focus` is set to true. It controls how long (in seconds) the delay is between the completion of the `jump_to_pdf` command and the attempt to refocus on Sublime Text. This may need to be adjusted depending on your machine or configuration.

Linux

- `python2` (`""`): name of the Python 2 executable. This is useful for systems that ship with both Python 2 and Python 3. The forward/backward search used with Evince require Python 2.
- `sublime` (`sublime-text`): name of the ST executable. Ubuntu supports both `sublime-text` and `subl`; other distros may vary.
- `sync_wait` (1.5): when you ask LaTeXTools to do a forward search, and the PDF file is not yet open (for example, right after compiling a tex file for the first time), LaTeXTools first launches evince, then waits a bit for it to come up, and then it performs the forward search. This parameter controls how long LaTeXTools should wait. If you notice that your machine opens the PDF, then sits there doing nothing, and finally performs the search, you can decrease this value to 1.0 or 0.5; if instead the PDF file comes up but the forward search does not seem to happen, increase it to 2.0.
- `sublime_executable` (`""`): this is used if `keep_focus` is set to true and the path to your sublime_text executable cannot be discovered automatically. It should point to the full path to your executable `sublime_text`.
- `keep_focus_delay` (`0.5`): this is used if `keep_focus` is set to true. It controls how long (in seconds) the delay is between the completion of the `jump_to_pdf` command and the attempt to refocus on Sublime Text. This may need to be adjusted depending on your machine or configuration.

Builder Settings

Note: for the time being, you will need to refer to the `LaTeXTools.sublime-settings` file for detailed explanations. Also, since the new build system is meant to be fully customizable, if you use a third-party builder (which hopefully will become available!), you need to refer to its documentation.

- `builder` (`"traditional"`): the builder you want to use. Leave blank (`""`) or set to `"default"` or `"traditional"` for the traditional (`latexmk`/`texify`) behavior.
- `builder_path` (`""`): builders can reside anywhere Sublime Text can access. Specify a path *relative to the Sublime text Packages directory*. In particular, `User` is a good choice. If you use a third-party builder, specify the builder-provided directory.
- `display_bad_boxes` (`false`): if `true` LaTeXTools will display any bad boxes encountered after a build. Note that this is disabled by default.
- `builder-settings`: these are builder-specific settings. For the `default` /`traditional` builder, the following settings are useful:
  - `program` (unset): one of `pdflatex` (the default), `xelatex` or `lualatex`. This selects the TeX engine.
  - `command` (unset): the precise `latexmk` or `texify` command to be invoked. This must be a list of strings. The defaults (hardcoded, not shown in the settings file) are:
    * (TeXLive): `["latexmk", "-cd", "-e", "-f", "-%E", "-interaction=nonstopmode", "-synctex=1"]`
    * (MiKTeX): `["texify", "-b", "-p", "--engine=%E", "--tex-option=\"--synctex=1\""]`
  - `options` (unset): allows you to specify a TeX option, such as `--shell-escape`. This must be a tuple: that is, use `options: ["--shell-escape"]`
  - `env` (unset): a dictionary of key-values corresponding to environment variables that should be set for the environment the build is run in. Note that `env`, if it is set, must be set at the platform-specific level, e.g., under the `osx`, `windows`, or `linux` keys. This is useful for setting, e.g., `TEXINPUTS`.
  - In addition, there can be platform-specific settings. An important one for Windows is `distro`, which must be set to either `miktex` or `texlive`.

Viewer settings

- `viewer` (`""`): the viewer you want to use. Leave blank (`""`) or set to `"default` "for the platform-specific viewer. Can also be set to `"preview"` if you want to use Preview on OS X, `"okular"` if you want to use Okular on Linux or `"command"` to run arbitrary commands. For details on the `"command"` option, see the section of the viewer documentation above.
- `viewer_settings`: these are viewer-specific settings. Please see the section on Viewers or the documentation on Alternate Viewers for details of what should be set here.

Bibliographic references settings

- `bibliography` (`"traditional_bibliography"`): specifies the bibliography plugin to use to handle extracting entries from a bibliography.

- `cite-panel-format` and `cite_autocomplete_format`: see the section on ref/cite completion, and the comments in `LaTeXTools.sublime-settings`

Cache settings

- `hide_local_cache` (`true`): Whether the local cache should be hidden in the sublime cache path (`true`) or in the same directory as the root file (`false`). See the section LaTeXTools Cache.
- `local_cache_life_span` (`30 m`): The lifespan of the local cache, specified in the format `" d x h X m X s"` where `X` is a natural number `s` stands for seconds, `m` for minutes, `h` for hours, and `d` for days. Missing fields will be treated as 0 and white-spaces are optional. Hence you can write `"1 h 30 m"` to refresh the cached data every one and a half hours. If you use `"infinite"` the cache will not be invalidated automatically. A lower lifespan will produce results, which are more up to date. However it requires more recalculations and might decrease the performance. See the section LaTeXTools Cache.

Project-Specific Settings

The above settings can be overridden on a project-specific basis if you are using Sublime Text's project system. To override these settings, simply create a `"settings"` section in your project file. The structure and format is the same as for the `LaTeXTools.sublime-settings` file. Here is an example:

```
{
    ... <folder-related options here> ...

    "settings" : {
        "TEXroot": "main.tex",
        "tex_file_exts": [".tex", ".tikz"],
        "builder_settings": {
            "program": "xelatex",
            "options": "--shell-escape"
        }
    }
}
```

This sets `main.tex` as the master tex file (assuming a multi-file project), and allows `.tikz` files to be recognized as tex files, but only for the current project. Furthermore (using the default, i.e., traditional builder), it forces the use of `xelatex` instead of the default `pdflatex`, and also adds the `--shell-escape` option—again, for the current project only.

Note: tweaking settings on a project-specific level can lead to even more subtle issues. If you notice a bug, in addition to resetting your `LaTeXTools.sublime-settings` file, you should remove at LaTeXTools settings from your project file.

Alternative Builders

Basic Builder

The basic builder is a simple, straight-forward build system. It differs from the `simple` builder in that: 1) whereas the simple builder is intended as an example of how to create a builder, the basic builder is intended to be an operational build system, 2) it supports all of the various builder settings that the `traditional` builder does, with the exception of the `command` setting, 3) it supports biber and biblatex more generally, and 4) it supports the output and auxiliary directory behavior on MiKTeX without installing any additional components, as recent versions of `texify` cannot be coerced into passing the necessary options to pdflatex and friends.

Script Builder

LaTeXTools now supports the long-awaited script builder. It has two primary goals: first, to support customization of simple build workflows and second, to enable LaTeXTools to integrate with external build systems in some fashion.

Note that the script builder should be considered an advanced feature. Unlike the "traditional" builder it is not designed to "just work," and is not recommend for those new to using TeX and friends. You are responsible for making sure your setup works. Please read this section carefully before using the script builder.

For the most part, the script builder works as described in the Compiling LaTeX files section *except that* instead of invoking either `texify` or `latexmk`, it invokes a user-defined series of commands. Note that although the Script Builder supports **Multi-file documents**, it does not support either the engine selection or passing other options via the `%!TEX` macros.

The script builder is controlled through two settings in the *platform-specific* part of the `builder_settings` section of `LaTeXTools.sublime-settings`, or of the current project file (if any):

- `script_commands` — the command or list of commands to run. This setting **must** have a value or you will get an error message.
- `env` — a dictionary defining any environment variables to be set for the environment the command is run in.

The `script_commands` setting should be either a string or a list. If it is a string, it represents a single command to be executed. If it is a list, it should be either a list of strings representing single commands or a list of lists, though the two may be mixed. For example:

```
{
    "builder_settings": {
        "osx": {
            "script_commands":
```

```
                "pdflatex -synctex=1 -interaction=nonstopmode"
        }
    }
}
```

Will simply run `pdflatex` against the master document, as will:

```
{
    "builder_settings": {
        "osx": {
            "script_commands":
                ["pdflatex -synctex=1 -interaction=nonstopmode"]
        }
    }
}
```

Or:

```
{
    "builder_settings": {
        "osx": {
            "script_commands":
                [["pdflatex", "-synctex=1 -interaction=nonstopmode"]]
        }
    }
}
```

More interestingly, the main list can be used to supply a series of commands. For example, to use the simple pdflatex -> bibtex -> pdflatex -> pdflatex series, you can use the following settings:

```
{
    "builder_settings": {
        "osx": {
            "script_commands": [
                "pdflatex -synctex=1 -interaction=nonstopmode",
                "bibtex",
                "pdflatex -synctex=1 -interaction=nonstopmode",
                "pdflatex -synctex=1 -interaction=nonstopmode"
            ]
        }
    }
}
```

Note, however, that the script builder is quite unintelligent in handling such cases. It will not note any failures nor only execute the rest of the sequence if required. It will simply continue to execute commands until it hits the end of the chain of commands. This means, in the above example, it will run `bibtex` regardless of whether there are any citations.

It is especially important to ensure that, in case of errors, TeX and friends do not stop for user input. For example, if you use `pdflatex` on either TeXLive or MikTeX, pass the `-interaction=nonstopmode` option.

Each command can use the following variables which will be expanded before it is executed:

Variable	Description
`$file`	The full path to the main file, e.g., *C:\Files\document.tex*
`$file_name`	The name of the main file, e.g., *document.tex*
`$file_ext`	The extension portion of the main file, e.g., *tex*
`$file_base_name`	The name portion of the main file without the, e.g., *document*
`$file_path`	The directory of the main file, e.g., *C:\Files*
`$aux_directory`	The auxiliary directory set via a `%!TEX` directive or the settings
`$output_directory`	The output directory set via a `%!TEX` directive or the settings

For example:

```
{
    "builder_settings": {
        "osx": {
            "script_commands": [[
                "pdflatex",
                "-synctex=1"
                "-interaction=nonstopmode",
                "$file_base_name"
            ]]
        }
    }
}
```

Note that if none of these variables occur in the command string, the `$file_base_name` will be appended to the end of the command. This may mean that a wrapper script is needed if, for example, using `make`.

Commands are executed in the same path as `$file_path`, i.e. the folder containing the main document.

Supporting output and auxiliary directories

If you are using LaTeXTools output and auxiliary directory behavior there are some caveats to be aware of. First, it is, of course, your responsibility to ensure that the approrpiate variables are passed to the appropriate commands in your script. Second, `pdflatex` and friends do not create output directories as needed. Therefore, at the very least, your script must start with either `"mkdir $output_directory"` (Windows) or `"mkdir -p $output_directory"` and a corresponding command if using a separate `$aux_directory`. Note that if you `\include` (or otherwise attempt anything that will `\@openout` a file in a subfolder), you will need to ensure the subfolder exists. Otherwise, your run of `pdflatex` will fail.

Finally, unlike Biber, bibtex (and bibtex8) does not support an output directory parameter, which can make it difficult to use if you are using the LaTeXTools

output directory behavior. The following work-arounds can be used to get BibTeX to do the right thing.

On Windows, run BibTeX like so:

```
cd $aux_directory & set BIBINPUTS=\"$file_path:%BIBINPUTS%\" &
    bibtex $file_base_name
```

And on OS X or Linux, use this:

```
"cd $output_directory; BIBINPUTS=\"$file_path;$BIBINPUTS\" bibtex
    $file_base_name"
```

In either case, these run bibtex *inside* the output / auxiliary directory while making the directory containing your main file available to the `BIBINPUTS` environment variable. Note if you use a custom style file in the same directory, you will need to apply a similar work-around for the `BSTINPUTS` environment variable.

Caveats

LaTeXTools makes some assumptions that should be adhered to or else things won't work as expected:

- the final product is a PDF which will be written to the output directory or the same directory as the main file and named `$file_base_name.pdf`
- the LaTeX log will be written to the output directory or the same directory as the main file and named `$file_base_name.log`
- if you change the `PATH` in the environment (by using the `env` setting), you need to ensure that the `PATH` is still sane, e.g., that it contains the path for the TeX executables and other command line resources that may be necessary.

In addition, to ensure that forward and backward sync work, you need to ensure that the `-synctex=1` flag is set for your latex command. Again, don't forget the `-interaction=nonstopmode` flag (or whatever is needed for your tex programs not to expect user input in case of error).

Customizing the Build System

Since the release on March 13, 2014 (v3.1.0), LaTeXTools has had support for custom build systems, in addition to the default build system, called the "traditional" builder. Details on how to customize the traditional builder are documented above. If neither the traditional builder nor the script builder meet your needs you can also create a completely custom builder which should be able to support just about anything you can imagine. Let me know if you are interested in writing a custom builder!

Custom builders are small Python scripts that interact with the LaTeXTools build system. In order to write a basic builder it is a good idea to have some basic

familiarity with the Python language. Python aims to be easy to understand, but to get started, you could refer either to the Python tutorial or any of the resources Python suggests for non-programmers or those familiar with other programming languages.

LaTeXTools comes packaged with a small sample builder to demonstrate the basics of the builder system, called `SimpleBuilder` which can be used as a reference for what builders can do.

Note that if you are interested in creating your own builder, please pay attention to the Caveats section above.

The Basics

A really simple Builder

Every builder consists of a single Python `class` called "WhateverBuilder" (so, for example, `"TraditionalBuilder"`, `"SimpleBuilder"`, etc.) which is a sub-class of a class called `"PdfBuilder"`. Note that the class name should be unique (i.e., it can't share a name with any of the built-in builders) and it must end with `"Builder"`. Each builder class implements a single generator function called `commands()` which is responsible for generating a list of commands to be run.

Below is a really simple builder that does nothing to demonstrate the basic structure of a builder:

```
# the pdfBuilder module will always be available on the PYTHONPATH
from pdfBuilder import PdfBuilder

class ReallySimpleBuilder(PdfBuilder):
    # for now, we are ignoring all of the arguments passed to the
    # builder
    def __init__(self, *args):
        # call the __init__ method of PdfBuilder
        # this does some basic initialization, which we'll discuss
        # in more detail later
        super(ReallySimpleBuilder, self).__init__(*args)

        # now we do the initialization for this builder
        # the only thing that must be set here is the builder name
        self.name = "Really Simple Builder"

    # commands is a generator function that yields the commands to
    # be run
    def commands(self):
        # display a message in the build output console
        self.display("\n\nReallySimpleBuilder")

        # yield is how we pass the command to be run back to
        # LaTeXTools
        #
        # each yield should yield a tuple consisting of the command
```

```
        # to be run and a message to be displayed, if any
        #
        # for the ReallySimpleBuilder, we yield ("", None) which
        # tells LaTeXTools there is no command to be run and no
        # message to be displayed
        yield ("", None)
```

To use this, save it to a file called `"reallySimpleBuilder.py"` in your Sublime Text `User` package (you can find this folder by selecting **Preferences|Browse Packages...** or running the **Preferences: Browse Packages** command). Then, in your LaTeXTools preferences, change the `"builder"` setting to `"reallySimple"` and change the `"builder_path"` setting to `"User"`. Try compiling a document. You should see the following:

```
[Compiling ...]

ReallySimpleBuilder
```

Notice how the message we set using `self.display()` gets displayed.

Also notice how the *name* of the Python file matches the name of the builder (except with the first letter lower-cased) and the name given to the setting. These **must** match in order for LaTeXTools to be able to find and execute your builder.

Generating Basic Commands

The `PdfBuilder` base class provides access to some very basic information about the tex document being compiled, which can be gathered from the following variables:

Variable	Description
`self.tex_root`	the full path of the main tex document, e.g. *C:\path\to\tex_root.tex*
`self.tex_dir`	the full path to the directory containing the main tex document, e.g. *C:\path\to*
`self.tex_name`	the name of the main tex document, e.g. *tex_root.tex*
`self.base_name`	the name of the main tex document without the extension, e.g. *tex_root*
`self.tex_ext`	the extension of the main tex document, e.g. *tex*

(Note that all of these refer to the main tex document, specified using the `%!TEX root` directive or the `"TEXroot"` setting)

With this is mind, we can now write a builder that actually does something useful. Below is a sample builder that simply runs the standard `pdflatex`, `bibtex`, `pdflatex`, `pdflatex` pattern.

```
from pdfBuilder import PdfBuilder
```

```
# here we define the commands to be used
# commands are passed to subprocess.Popen which prefers a list of
# arguments to a string
PDFLATEX = ["pdflatex", "-interaction=nonstopmode", "-synctex=1"]
BIBTEX = ["bibtex"]

class BasicBuilder(PdfBuilder):
    def __init__(self, *args):
        super(BasicBuilder, self).__init__(*args)

        # now we do the initialization for this builder
        self.name = "Basic Builder"

    def commands(self):
        self.display("\n\nBasicBuilder: ")

        # first run of pdflatex
        # this tells LaTeXTools to run:
        #  pdflatex -interaction=nonstopmode -synctex=1 tex_root
        # note that we append the base_name of the file to the
        # command here
        yield(PDFLATEX + [self.base_name], "Running pdflatex...")

        # LaTeXTools has run pdflatex and returned control to the
        # builder
        # here we just add text saying the step is done, to give
        # some feedback
        self.display("done.\n")

        # now run bibtex
        yield(BIBTEX + [self.base_name], "Running bibtex...")

        self.display("done.\n")

        # second run of pdflatex
        yield(
            PDFLATEX + [self.base_name],
            "Running pdflatex again..."
        )

        self.display("done.\n")

        # third run of pdflatex
        yield(
            PDFLATEX + [self.base_name],
            "Running pdflatex for the last time..."
        )

        self.display("done.\n")
```

To use this, save it to a file called `"basicBuilder.py"` then change your `"builder"` setting to `"basic"`. When you compile a document, you should see the following output:

```
[Compiling ...]

BasicBuilder: Running pdflatex...done.
```

```
Running bibtex...done.
Running pdflatex again...done.
Running pdflatex for the last time...done.

...
```

Since this builder actually does a build, there may be additional messages displayed after the last message from our builder. These usually come from LaTeXTools log-parsing code which is run after the build completes.

Interacting with Output

Of course, sometimes it is necessary not just to run a series of commands, but also to react to the output of those commands to determine the next step in the process. This is what the `SimpleBuilder` does to determine whether or not to run BibTeX, searching the output for a particular pattern that `pdflatex` generates to determine whether or not to run BibTeX.

The output of the previously run command is available after LaTeXTools returns control to the builder in the variable `self.out`. This consists of anything written to STDOUT and STDERR, i.e., all the messages you would see if running the command from the terminal / command line.

Building on our previous example, here's a builder that checks to see if BibTeX (only) needs to be run. This example makes use of Python's re library, which provides operations for dealing with regular expressions, a way of matching patterns in strings.

```
from pdfBuilder import PdfBuilder

import re

PDFLATEX = ["pdflatex", "-interaction=nonstopmode", "-synctex=1"]
BIBTEX = ["bibtex"]

# here we define a regular expression to match the output expected
# if we need to run bibtex
# this matches any lines like:
#  Warning: Citation: `aristotle:ethics' on page 2 undefined
CITATIONS_REGEX = re.compile(
    r"Warning: Citation `.+' on page \d+ undefined"
)

class BibTeXBuilder(PdfBuilder):
    def __init__(self, *args):
        super(BibTeXBuilder, self).__init__(*args)

        # now we do the initialization for this builder
        self.name = "BibTeX Builder"

    def commands(self):
        self.display("\n\nBibTeXBuilder: ")
```

```
        # we always run pdflatex
        yield(PDFLATEX + [self.base_name], "Running pdflatex...")

        # here control has returned to the builder from LaTeXTools
        # we display the same message as last time...
        self.display("done.\n")

        # and now we check the output to see if bibtex needs to be
        # run
        # search will scan the entire output for any match of the
        # pattern we defined above
        if CITATIONS_REGEX.search(self.out):
            # if a matching bit of text is found, we need to run
            # bibtex
            # now run bibtex
            yield(BIBTEX + [self.base_name], "Running bibtex...")

            self.display("done.\n")

            # we only need to run the second and third runs of
            # pdflatex if we actually ran bibtex, so these remain
            # inside the same `if` block code to run bibtex

            # second run of pdflatex
            yield(
                PDFLATEX + [self.base_name],
                "Running pdflatex again..."
            )

            self.display("done.\n")

            # third run of pdflatex
            yield(
                PDFLATEX + [self.base_name],
                "Running pdflatex for the last time..."
            )

            self.display("done.\n")
```

To use this builder, save it to a file called `"bibTeXBuilder.py"` and change the `"builder"` setting to `"bibTeX"`. When using this builder, you should see that it only runs `bibtex` and the final two `pdflatex` commands if you have any citations.

More Advanced Topics

The following sections deal with more advanced concepts in creating builders or with features that need careful handling for one reason or another.

Allowing the user to set options

Sometimes having a static series of commands to build a document is not enough and you'd want to give the user an opportunity to, for example, tell the builder what command to run to generate a bibliography. LaTeXTools provides your

builder with access to the settings in the `build_settings` block of your LaTeXTools preferences through the variable `self.builder_settings`. Note that when allowing the use of settings it is important to verify that values you get make sense and that if no value is supplied, you provide a sane default.

Our builder from the previous example could be modified to support either bibtex or biber as the bibliography program depending on a user setting like so:

```
from pdfBuilder import PdfBuilder

import re
import sublime

PDFLATEX = ["pdflatex", "-interaction=nonstopmode", "-synctex=1"]
# notice that we do not define the bibliography command here, since
# it will depend on settings that can only be known when our builder
# is initialized

CITATIONS_REGEX = re.compile(
    r"Warning: Citation `.+' on page \d+ undefined"
)
# BibLaTeX outputs a different message from BibTeX, so we must catch
# that too
BIBLATEX_REGEX = re.compile(
    r"Package biblatex Warning: Please \(re\)run \S*"
)

class BibBuilder(PdfBuilder):
    def __init__(self, *args):
        super(BibBuilder, self).__init__(*args)

        # now we do the initialization for this builder
        self.name = "Bibliography Builder"

        # here we get which bibliography command to use from the
        # builder_settings
        # notice that we draw this setting from the
        # platform-specific portion of the builder_settings block
        # this allows the setting to be changed for each platform
        self.bibtex = self.builder_settings.get(
            sublime.platform(), {}).get('bibtex') or 'bibtex'
        # notice that or clause here ensures that self.bibtex will
        # be set to 'bibtex' if the 'bibtex' setting is unset or
        # blank.

    def commands(self):
        self.display("\n\nBibBuilder: ")

        # we always run pdflatex
        yield(PDFLATEX + [self.base_name], "Running pdflatex...")

        # here control has returned to the builder from LaTeXTools
        self.display("done.\n")

        # and now we check the output to see if bibtex / biber needs
        # to be run
        if (
```

```
            CITATIONS_REGEX.search(self.out) or
            BIBLATEX_REGEX.search(self.out)
        ):
            # if a matching bit of text is found, we need to run the
            # configured bibtex command
            # note that we wrap this value in a list to ensure that
            # a list is yielded to LaTeXTools
            yield(
                [self.bibtex] + [self.base_name],
                "Running " + self.bibtex + "..."
            )

            self.display("done.\n")

            # second run of pdflatex
            yield(
                PDFLATEX + [self.base_name],
                "Running pdflatex again..."
            )

            self.display("done.\n")

            # third run of pdflatex
            yield(
                PDFLATEX + [self.base_name],
                "Running pdflatex for the last time..."
            )

            self.display("done.\n")
```

To use this builder, save it to a file called `"bibBuilder.py"` and change the `"builder"` setting to `"bib"`. You should be able to change what command gets run when a bibliography is needed by changing the `"bibtex"` setting in `"builder_settings"`.

Assuming you are on OS X, you might use something like this to run biber instead of bibtex with this builder:

```
{
    "builder_settings": {
        "osx": {
            "bibtex": "biber"
        }
    }
}
```

Important Notice that all interaction with the settings occurred in builder's `__init__()` function. This is to ensure that the builder works on ST2 as well as ST3. For more on this, see the section on Interacting with the Sublime API below.

Passing `Popen` Objects

With v3.6.2 (released January 18, 2016), the main execution loop has been modified to support accepting not only strings and lists as we have seen above, but also to allow a builder to yield `Popen` objects as well. This might be useful if

you need more fine-grained control over the exact way a command is executed, perhaps using a custom environment or using the user's shell. For the most part, it is recommended that you avoid creating custom `Popen` objects unless you cannot get the standard tools to do what you need.

In principle, using `Popen` objects is pretty much the same as using strings or lists of commands, you simply yield them, and LaTeXTools allows them to run before returning to your code. However, some caveats must be observed.

Caveats:

- It is important to ensure that your `Popen` objects are always created with `stdout` set to `subprocess.PIPE`. Otherwise, you will not get any output to respond to. It is recommended that you set `stderr` to `subprocess.STDOUT` so that you also get any error output.
- On OS X and Linux, it is important to set the `preexec_fn` to `os.setsid`. Otherwise, your builder could fail to kill any spawned or forked processes that are started as part of the build.
- If you override the `env`, please be sure that you properly copy the `PATH` variable from `os.environ` to your new environment, especially on OS X or otherwise take account of the user's configured `texpath` setting. Otherwise, your builder may be unable to find LaTeX and friends. This is especially true on more recent versions of OS X.
- In addition to the above, please make sure you have read through and understood the other caveats in the **Caveats** section of this document.

An example where this could be helpful is in selecting the bibliography engine to run using the MiKTeX-only command `texify`. As the documentation states, the bibliography processing program can be selected by setting the `%BIBTEX%` environment variable. Expanding on our previous example, here is a builder that uses `texify` to run a standard latex build with the bibliography processor selected by a user-configurable option:

```
from pdfBuilder import PdfBuilder

import copy
import os
import sublime
# we need to import subprocess as we will be creating one of our own
import subprocess


class MikbibBuilder(PdfBuilder):
    def __init__(self, *args):
        super(BibBuilder, self).__init__(*args)

        # now we do the initialization for this builder
        self.name = "MiKTeX Bibliography Builder"

        self.bibtex = self.builder_settings.get(
```

```
            sublime.platform(), {}).get('bibtex') or 'bibtex'

    def commands(self):
        self.display("\n\nMiKTeX BibliographyBuilder: ")

        # MiKTeX is Windows-specific, so it doesn't make sense to
        # try to run it on a non-Windows machine
        if sublime.platform() != 'windows':
            yield(
                "",
                "The MiKTeX Bibliography Builder can only be used"
                " on Windows"
            )
            return

        # we create this startupinfo object to ensure that the
        # Windows console does not appear
        startupinfo = subprocess.STARTUPINFO()
        startupinfo.dwFlags |= subprocess.STARTF_USESHOWWINDOW

        # now we construct the environment
        # first creating a copy of the current environment
        env = copy.copy(os.environ)
        # then setting the %BIBTEX% variable to the user-configured
        # setting
        env['BIBTEX'] = self.bibtex
        # note that LaTeXTools ensures that the %PATH% variable
        # contains the `texpath` setting by default

        # here we construct our Popen object
        p = subprocess.Popen(
            ['texify', '-b', '-p', '--texoption="--synctex=1"'],
            # here we use the startupinfo object created above
            startupinfo=startupinfo,
            # we redirect any output to stderr to stdout
            stderr=subprocess.STDOUT,
            # ensure that the output to stdout is available to
            # LaTeXTools
            stdout=subprocess.PIPE,
            # finally, we pass in the environment that should be
            # used including our modified %BIBTEX% value
            env=env
        )

        # now we yield the Popen object to LaTeXTools to be run
        yield(p, "Running texify...")

        # here control has returned to the builder from LaTeXTools
        self.display("done.\n")
        # because texify runs the whole pdflatex, bibtex, pdflatex,
        # pdflatex cycle, we don't need to do anything else.
```

To use this builder, save it to a file called `"mikbibBuilder.py"` and change the `"builder"` setting to `"mikbib"`. You should be able to change what command gets run when a bibliography is needed by changing the `"bibtex"` setting in `"builder_settings"` as with the previous builder.

Interacting with the Sublime API

Sublime Text provides a very rich API that could be of use to builders. However, it is advisable that any interactions with the Sublime Text API happen in the `__init__()` function, if possible. This is because the `__init__()` function is run on the main Sublime thread whereas the `commands()` function is called from a separate thread which cannot safely interact with the Sublime API on ST2. `commands()` is run on a separate thread.

Alternative Viewers

Command Viewer

Some support for other viewers is provided via the `command` viewer, which allows the execution of arbitrary commands to view a pdf or perform a forward search.

Using the command viewer requires that you configure the command(s) to be run in the platform-specific part of the `viewer_settings` block in your LaTeXTools preferences. There are three commands available:

- `forward_sync_command`: the command to executing a forward search (`ctrl + l, j` or `cmd + l, j`).
- `view_command`: the command to simply view the PDF document.

Of these, on `view_command` needs to be specified, though you will not have forward search capabilities unless you specify a `forward_sync_command` as well.

The following variables will be substitued with appropriate values inside your commands:

Variable	Description
`$pdf_file`	full path of PDF file, e.g. *C:\Files\document.pdf*
`$pdf_file_name`	name of the PDF file, e.g. *document.pdf*
`$pdf_file_ext`	extension of the PDF file, e.g. *pdf*
`$pdf_file_base_name`	name of the PDF file without the extension, e.g. *document*
`$pdf_file_path`	full path to directory containing PDF file, e.g. *C:\Files*
`$sublime_binary`	full path to the Sublime binary

In addition, the following variables are available for the `forward_sync_command` only:

Variable	Description
`$src_file`	full path of the tex file, e.g. *C:\Files\document.tex*
`$src_file_name`	name of the tex file, e.g., *document.tex*
`$src_file_ext`	extension of the tex file, e.g. *tex*

Variable	Description
`$src_file_base_name`	name of the tex file without the extension, e.g. *document*
`$src_file_path`	full path to directory containing tex file, e.g. *C:\Files*
`$line`	line to sync to
`$col`	column to sync to

If none of these variables occur in the command string, the `$pdf_file` will be appended to the end of the command.

Commands are executed in the `$pdf_file_path`, i.e., the folder containing the `$pdf_file`.

For example, you can use the command viewer to support Okular with the following settings:

```
{
    "viewer": "command",

    "viewer_settings": {
        "linux": {
            "forward_sync_command": "okular --unique $pdf_file#src:$line$src_file",
            "view_command": "okular --unique"
        }
    }
}
```

LaTeXTools Cache

LaTeXTools uses a cache to store relevant information about your document and improve the performance of commands. The cache is made up of a series of files. Where these files are stored depends on your settings. By default, we try to keep them invisible, so they are stored in the Sublime cache path. However, by changing the `hide_local_cache` setting, you can have them stored in a hidden folder in the same directory as your tex root.

The local cache also has a lifespan, after which it will be invalidated. The lifespan starts when the first entry is inserted in the cache and the whole cache will be deleted after the lifespan. This can be set in the `local_cache_life_span` setting. The format is `"X d X h X m X s"`, where `X` is a natural number `s` stands for seconds, `m` for minutes, `h` for hours, and `d` for days. Missing fields will be treated as 0 and white-spaces are optional. Hence you can write `"1 h 30 m"` to refresh the cached data every one and a half hours. If you use `"infinite"` the cache will not be invalidated automatically. A lower lifespan will produce results, which are more up to date. However it requires more recalculations and might decrease the performance.

```
        self.display("done.\n")
```

Troubleshooting

Path issues

Many LaTeXTools problems are path-related. The `LaTeXTools.sublime-settings` file attempts to set up default path locations for MiKTeX, TeXLive and MacTeX, but these are not guaranteed to cover all possibilities. Please let me know if you have any difficulties.

On Mac OS X, just having your `$PATH` set up correctly in a shell (i.e., in Terminal) does *not* guarantee that things will work when you invoke commands from ST. If something seems to work when you invoke `pdflatex` or `latexmk` from the Terminal, but building from within ST fails, you most likely have a path configuration issue. One way to test this is to launch ST from the Terminal, typing

```
/Applications/Sublime Text.app/Contents/SharedSupport/bin/subl
```

(and then Return; this is for ST23of course) at the prompt. If things do work when you run ST this way, but they fail if you launch ST from the Dock or the Finder, then there is a path problem. From the Terminal, type

```
echo $PATH
```

and take note of what you get. Then, run ST from the Dock or Finder, open the console (with `Ctrl+ ``) and type

```
import os; os.environ['PATH']
```

and again take note of what you see in the output panel (right above the line where you typed the above command). Finally, look at the `texpath` keyword in the `osx` section of the `LaTeXTools.sublime-settings` file For things to work, every directory that you see listed from the Terminal must be either in the list displayed when you type the `import os...` command in the ST console, or else it must be explicitly specified in the `texpath` setting found in `LaTeXTools.sublime-settings`. If this is not the case, add the relevant paths to the `texpath` setting and *please let me know*, so I can decide whether to add the path specification to the default build file. Thanks!

On Linux, do note that your login shell may be different from the shell that launched Sublime Text. This can mean that LaTeXTools does not inherit your `$PATH` correctly, particularly if you modify your `$PATH` in `.bash_profile` or `.bashrc` or other, shell-specific files (X Windows is run via `/bin/sh` rather than `/bin/bash`). If you have a similar problem, follow the same procedure as above, although you should launch the `sublime_text` executable from a shell.

Non-ASCII characters and spaces in path and file names

Another *significant* source of issues are **Unicode characters in path and file names**. On TeXLive-based platforms, LaTeXTools tries to handle these by

telling `latexmk` to `cd` to each source file's directory before running `pdflatex`. This seems to help some. However, things seem to vary by platform and locale, so I cannot make any guarantees that your Unicode path names will work. Keep in mind that TeX itself has issues with Unicode characters in file names (as a quick Google search will confirm).

Spaces in paths and file names *are* supported. As far as I know, the only limitation has to do with multifile documents: the root document's file name cannot contain spaces, or the `%!TEX = <name>` directive will fail. I may fix this at some point, but for now it is a limitation.

Compilation hangs on Windows

On Windows, sometimes a build seems to succeed, but the PDF file is not updated. This is most often the case if there is a stale pdflatex process running; a symptom is the appearence of a file with extension `.synctex.gz(busy)`. If so, launch the Task Manager and end the `pdflatex.exe` process; if you see a `perl.exe` process, end that, too. This kind of behavior is probably a bug: LaTeXTools should be able to see that something went wrong in the earlier compilation. So, *please let me know*, and provide me with as much detail as you can (ideally, with a test case). Thanks!

Log parsing issues, and good vs. bad path/file names (again!)

As noted in the Highlights, the new parser is more robust and flexible than the old one—it "understands" the log file format much, much better. This is the result of *manually* and *painstakingly* debugging a fair number of users' log files. The many possible exceptions, idiosyncrasies, warts, etc. displayed by TeX packages is mind-boggling, and the parsing code reflects this :-(

Anyway, hopefully, errors should now occur only in strange edge cases. Please *let me know on github* if you see an error message. I need a log file to diagnose the problem; please upload it to gist, dropbox, or similar, and paste a link in your message on github. Issue #104 is open for that purpose.

There are *two exceptions* to this request. First, the *xypic* package is very, very badly behaved. I have spent more time debugging log files contaminated by xypic than I have spent fixing all other issues. Seriously. Therefore, first, parsing issues are now reported as "warnings" if the xypic package is used (so compilation and previewing continues); second, I cannot promise I will fix the issue even if you report it. Thanks for your understanding.

The second exception has to do with file and path names. In order to accommodate the many possible naming conventions across platforms and packages, as well as the different ways in which file names can occur in logs, I had to make some assumptions. The key one is that *extensions cannot contain spaces*.

The reason is that the regex matching file names uses a period (".") followed by non-space characters, followed by a space as denoting the end of the file name. Trust me, it's the most robust regex I could come up with. So, you can have spaces in your base names, and you can even have multiple extensions; however, you cannot have spaces in your extensions. So, "This is a file.ver-1.tex" is OK; "file.my ext" (where "my ext" is supposed to be the extension) is *not OK*.

Finally, I have done my best to accommodate non-ASCII characters in logs. I cannot promise that everything works, but I'd like to know if you see issues with this.