TeXstudio：ユーザーマニュアル

2015 年 10 月 27 日

前書き

このマニュアルは、TeXstudio[*1]付属の HTML マニュアルを日本語に翻訳したもの[*2] に基づいて、PDF にしたものである。

訳者注：このマニュアルは **TeXstudio** 付属のものを日本語に翻訳したものであり、内容が古いままになっている可能性がある。

[*1] http://texstudio.sourceforge.net/

[*2] http://anastashia.github.io/texstudio_jp_manual/index.html

目次

表目次

図目次

第 1 章

TeXstudio の設定

TeXstudio を使用する前に、「オプション」メニュー（Mac OS X では「好みの設定」）の「TeXstudio の設定」を通じてエディターと LaTeX 関連のコマンドの設定をすべきである。注意すべきは詳細には 2 段階あることだ。より高度なオプションやあまり使用しないオプションは、左下隅の「高度なオプションの表示」にチェックをした場合にのみ表示される。

1.1 エディターの設定

エンコーディングとして UTF8 を使用したくない場合には、新規ファイルに対する既定のエンコーディングを変えることができる（「TeXstudio の設定」-> 「エディタ」 -> 「既定のフォントエンコーディング」）。忘れずに文書中のプリアンブルで同じエンコーディングをセットするように（例：UTF8 を使用している場合、`\usepackage[utf8]{inputenc}`）。

TeXstudio は UTF8 と latin1 エンコーディングファイルを自動検出できるが、既存の文書で異なるエンコーディングを使用する場合は開く前に設定ダイアログでエンコーディングを指定する必要がある（そして自動検出も無効にする必要がある）。

- 「折りたたむ」はエディターのコード折りたたみ機能を有効／無効化する（テキストのセクションを隠す）。
- 選択ボックス「字下げモード」では、Enter キーを入力後同じインデントの行にインデントされる行を揃えるか、TeXstudio の自動インデントに任せるかを選択できる。

図 1.1　エディターの設定

1.2　LaTeX 関連コマンドの設定

LaTeX は、LaTeX 文書をコンパイルしたり操作したりするための多数のコマンドラインツールを備えている。コマンドの段落ではその場所と引数が定義されている。

既定値は最近の標準的な LaTeX ディストリビューションで機能するが、それらを変更する必要があるかもしれない（「TeXstudio の設定」 -> 「コマンド」）。

コマンドを変更するには、対応する行の末尾のボタンをクリックして、ファイルブラウザでコマンドを選択するだけでよい。TeXstudio は自動的にコマンドの構文を適合する。

現在の文書の状況を扱うために多数の特殊文字／文字列を使用できる。それらは実行時に展開される：

表 1.1　コマンド中の特殊文字

特殊文字	展開後
%	現在の文書に対するルート文書の拡張子なしファイル名
@	現在の行番号
?（後ろに文字が続く）	設定ダイアログの下部の説明を参照
[txs-app-dir]	TeXstudio 実行ファイルの場所（ポータブル版の設定に有用）
[txs-settings-dir]	設定ファイル (texstudio.ini) の場所

TeXstudio での「順方向／逆方向検索」の段落で一般的なビューワーに対するいくつかのコマンド例が示されている。

また、右の「既定値に戻す」ボタンを使って既定の設定にいつでも戻すことができる。

図 1.2　コマンドの設定

1.3　ビルドシステムの設定

TeXstudio では LaTeX を変換する一般的なコマンドが提供されている。

既定の設定では “pdflatex” と組み込みの PDF ビューワーを使用する。また、別の参考文献変換器と同様に他のコマンドやビューワーを選択することができる。

「組み込み PDF ビューワー」では PDF 文書を閲覧する際に新しくウィンドウは開かれず、エディターのテキストの隣に直接表示される。

有益な別の方法として “latexmk” をコンパイルコマンドとして使用する方法がある（システムにコマンドがインストールされている場合）。“latexmk” では biblatex と索引で依存関係を非常にうまく扱える。

更に、高度なオプションでは一般的には必要ないより詳細なカスタマイズを行える。

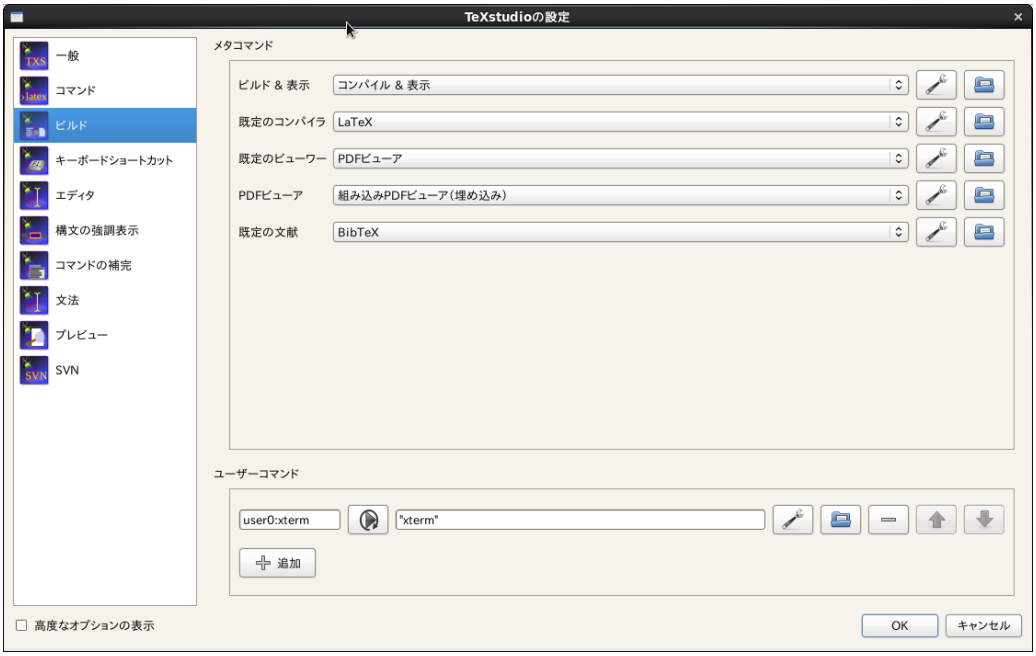

図 1.3　ビルドシステムの設定

ここでユーザーコマンドを「追加」することで定義できる。各々のユーザーコマンドには<command id>:<display name>のようなパターンを持つ名前がつく（例：user0:User Command 0）。command id は一意的でなければならず、スペースを含んではならない。ビルドシステムの高度な設定では、txs:///"<command id>を用いてそれを参照することができる。display name はツールメニューに表示されることになる。ユーザーコマンドはショートカット (alt+shift+F%n) またはツールメニュー（ツール／ユーザー）のいずれかで実行できる。

ユーザーコマンドは既知のコマンドを利用可能なコマンドのリストから選択して組み合わせてもよい。この場合スパナマークをクリックすると選択可能になる。

別な方法としては、ファイルシステムを通じて直接コマンドを選択してもよい。

図 1.4　既知のコマンドからのユーザーコマンドの設定

1.3.1　ビルドシステムの高度な設定

高度なオプションを表示するようにしている場合、ビルドシステムをより詳細に設定することができる。

すべての txs(TeXstudio) コマンドは呼び出すべき外部プログラム／ LaTeX コマンドと他の txs コマンドのリストとなっている。外部プログラムは通常のコマンドラインで呼び出すことができるが、ID“foobar” を持つ txs コマンドは txs:///foobar で呼び出す。

リストのコマンドは単なる区切りである “|” で区切られる（“|” はあるプログラムからの標準出力を次のプログラムの標準入力へ渡す「パイプ」ではない）。

これら txs コマンドにはそれぞれ一意的な ID が付けられ、「通常の」コマンドの、あるいは編集ボックスではユーザーコマンドの名前のツールチップとして表示される。いくつかの重要なコマンドはよく使用される：txs:///quick（ビルド＆表示、以前のクイックビルド）、txs:///compile（既定のコンパイラ）、txs:///view （既定のビューワー）、 txs:///latex(LaTeX)、txs:///pdflatex (pdfLaTeX)、 txs:///view-pdf（既定の PDF ビューワー）、txs:///view-pdf-external （外部 PDF ビューワー）。

例えば、典型的なビルド設定では F1 を押して txs:///quick を呼び出すと、txs:///compile が呼ばれ（まず実際に pdfLaTeX を実行する txs:///pdflatex が呼ばれる）、そして txs:///view が呼ばれて（txs:///view-pdf が呼ばれ、それによって txs:///view-pdf-internal が呼ばれる）、PDF が表示される。

コマンド設定ページでコマンドとして定義されたコマンドと、ビルド設定ページでビルドとして定義されたコマンド、ユーザーコマンドとして定義されたコマンドには違いはない。インターフェースの単純化のために GUI 側で区別しているだけである。

これは、前の定義を無視してあらゆるコマンドを望むように変更できる（ini ファイルを編集して ID を変えることさえできる）という事でもある。

しかし、常に定義されている三つの内部コマンドがあり、それらは呼び出すことのみ可能で変更はできない：

表 1.2　TeXstudio で定義済みの内部コマンド

`txs:///internal-pdf-viewer`	現在の文書に対して内部ビューワーを開く
`txs:///view-log`	現在の文書に対してログファイルを閲覧する
`txs:///conditionally-recompile-bibliography`	bib ファイルが変更されているか確認して、変更されている場合 `txs:///recompile-bibliography` を呼ぶ

内部 PDF ビューワー (`txs:///internal-pdf-viewer`) には動作を変更するために次のオプションを使用することもできる：

表 1.3　内部 PDF ビューワーのオプション

`-embedded`	ビューワーを埋め込みで開く
`-windowed`	ビューワーを別枠で開く（オプションが与えられてない場合の規定値）
`-close-(all|windowed|embedded)`	すべての開いているビューワーを閉じるか、あるいは特定のビューワーだけを閉じる
`-preserve-existing`	既存のビューワーを変更しない（つまり、常に新しいビューワーを開く）
`-preserve-(embedded|windowed)`	既存の埋め込み／別枠ビューワーを変更しない
`-preserve-duplicates`	最初に開いたビューワーでのみ PDF を開く
`-(no-)auto-close`	対応する tex ファイルを閉じた時にビューワーも閉じるかどうかを決める（既定：埋め込みの場合自動的に閉じる (auto-close)）
`-(no-)focus`	ビューワーを開いた時にビューワーに焦点を移すかどうかを決める（既定：別枠の場合焦点を移す (focus)）
`-(no-)foreground`	ビューワーを前面に持ってくるかどうかを決める（既定：前面に持って来る (foreground)）
`filename`	開くべきファイルを決める。他のコマンドと同様に、ファイルパターンをサポートしている。このパラメータが与えられていない場合、TXS は “?am.pdf”（つまりメインファイルの絶対パス）を使用する。このパラメータが絶対パスでのファイル名でない場合、メインファイルのあるディレクトリと「オプション -> TeXstudio の設定 -> ビルド -> ビルドオプション -> 追加の検索パス -> PDF ファイル」のディレクトリの中を探す。

また、呼び出されるサブコマンドの引数を引数変更子または新しい引数の追加で変更することもできる。これらの変更子は呼び出しリストを通じて渡されるので、直接呼び出されるサブコマンドが別のコマンドの単なるラッパーでも、最終的に呼び出されるプログラムの引数は常に変更される：

表 1.4　コマンドのオプションの例

`txs:///foobar -xyz`	`xyz` オプションを追加
`txs:///foobar[-xyz=123]`	`xyz` オプションの値を 123 に変更（つまり、`foobar` で定義されたあらゆる `xyz` オプションを削除して変更）
`txs:///foobar{-xyz=123}`	他の値を持つ `xyz` オプションを無視して、`foobar` コマンドラインから`-xyz=123` を削除
`txs:///foobar{-xyz}`	値にかかわらず、あらゆる`-xyz` オプションを `foobar` コマンドラインから削除
`txs:///foobar{}`	実行可能ファイル名のみ残して、すべてのオプションを `foobar` コマンドラインから削除

最後に、ini ファイルを変更することでしか変えられない隠しオプションもある：Tools/Kind/LaTeX、 Tool-

s/Kind/Rerunnable、 Tools/Kind/Pdf、Tools/Kind/Stdout、 Tools/Kind/Viewer。これらはそれぞれ LaTeX コンパイラ（例えば、後にログを表示する）、再実行可能（警告がある場合にコマンドを繰り返し呼び出す）、PDF 生成器（例えば、pdflatex）、標準出力へ出力するコマンド（例えば、bibtex）、そしてビューワー（例えば一度だけ開く）として扱われるコマンドのリストである。

1.4　実行環境の詳細

1.4.1　環境変数

実行時に利用できる環境変数は、TeXstudio を起動した状況で利用できる環境変数と同じである。特にこれは PATH に当てはまる。Linux/OS X では TeXstudio を起動する方法に依存するかもしれない。GUI で起動した場合とシェルから起動した場合で PATH の設定が異なるかもしれない（何故ならいくつかの変数はシェルの状況でのみ定義されるからだ（例：~/.bashrc を通じて））。

既定では、TeXstudio ではコマンド内の環境変数は構文解析される。構文は利用しているオペレーションシステムによって異なる。変数 MYVAR は、Windows では `%MYVAR` と書かれ、Linux と OS X では`$MYVAR` と書かれる。Windows では環境変数は大文字子文字は区別されないが、Linux と OS X では区別される。環境変数の構文解析はオプションの「ビルド」部分で無効化できる。

1.4.2　作業ディレクトリ

作業ディレクトリは、ルートドキュメントのパスに設定される。

1.4.3　シェルの機能

設定で指定された全てのコマンド（つまり「コマンド」と「ユーザーコマンド」）は直接実行される。シェルが関わることはない。従ってほとんどのシェル機能は機能しない。

出力リダイレクト

TeXstudio で提供される出力リダイレクト機能は限られている。メッセージパネル (`> txs:///messages`) へ出力するか、出力を抑制する (`> /dev/null`) ことだけ行うことができる。既定の設定はコマンドによって異なる。stderr を同じターゲットにすることは可能である：`2> txs:///messages`, `2> /dev/null`。さらに、`2>&1` を用いて stdout と同じターゲットへリダイレクトすることもできる。

典型的な活用事例としては、コマンドの出力すべてを抑制する事が挙げられる：`>/dev/null 2>&1`

注：Linux/Unix 表記 (`> /dev/null`) の代わりに、Windows 表記 (`> nul`) を使用しても良い。なぜならこれらのコマンドは TXS が直接解釈するので、両方の表記は全てのオペレーションシステムで機能する。

他のシェル機能を使用する

シェル機能が必要な場合、明示的にシェルを起動しなければならない。ユーザーコマンドでこれを直接行うことも可能である：

```
1 sh -c "/path/to/testscript foo > bar"
```

Windows の場合:

```
1 cmd /C "/path/to/testscript.bat foo > bar"
```

もしくはユーザーコマンドでラッパースクリプトを呼び出し、

```
1 /path/to/wrapperscript foo bar
```

ラッパースクリプト内で実際の作業を行う方法もある：

```
1 #!/bin/sh
2 # I am wrapperscript
3 /path/to/testscript $1 > $2
```

1.5　一般的な事項の設定

このパネルではいくつかの一般的な外見の設定を行うことができる。

- TeXstudio では「スタイル」と「配色」を選択できる。「配色」のモダンは texmaker 1.9 に似ている。
- 記号リストは「タブ形式」で表示されるか（以前の形式、タブ形式が有効な場合）、あるいは記号の空きを多くするため記号リストのそばの小さな記号タブとして表示される。
- ログビューワーもエラー一覧表やログビュー、プレビューワーなどの間をより速く移動できるようにタブ形式で表示される。
- メニューの言語をシステムの設定を無視して直接変更することができる。

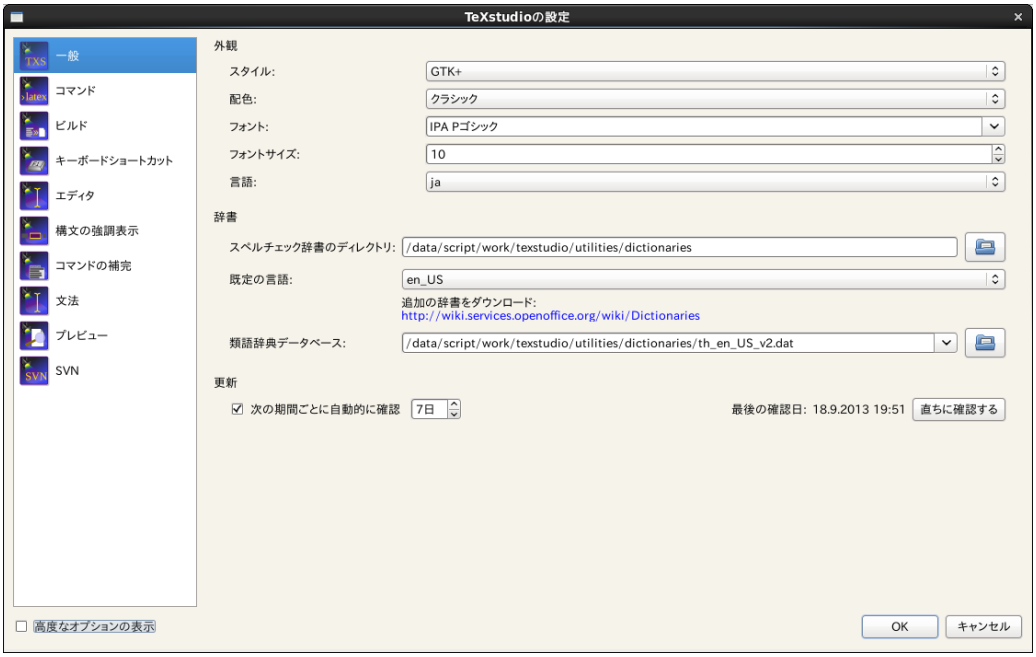

図 1.5　一般的な設定

1.5.1　スペルチェッカーの設定

TeXstudio では統合されたスペルチェック機能を提供している。それはダイアログを通してもしくは入力中に直接利用できる。LaTeX コマンドの外側のテキスト全てがチェックされる。さらに、LaTeX コマンドのオプション中のテキストもチェックされる。TeXstudio はオプションが自然なテキストを含んでいるかどうか判断する。そして補完単語リスト中の定義を調べることでスペルチェックが行われる。補完単語リストに関するさらなる情報は、自動補完の設定と cwl 形式の解説の段落を参照のこと。

スペルチェックは OpenOffice や LibreOffice、Firefox など幅広く使われている Hunspell の辞書形式を利用している。それぞれの辞書は 2 つのファイル (`.dic` と `.aff`) からなる。TeXstudio にはフランス語、英語、ドイツ語の辞書が同梱されている。辞書ディレクトリパスに追加の辞書を配置することで、それを使用できる。追加の辞書を入手する特に便利な方法は、http://wiki.services.openoffice.org/wiki/Dictionaries または LibreOffice の辞書拡張をダウンロードして、TeXstudio のオプションの**辞書のインポート**ボタンを使ってインポートすることだ。

ユーザーはオプションで辞書の探索パスを一つ以上指定することができる。複数のパスを指定する場合、セミコロンで区切る必要がある。パスには特殊文字 `[txs-app-dir]` と `[txs-settings-dir]` を使用できる。これらはそれぞれ TeXstudio の実行ファイルと設定ファイル (`texstudio.ini`) のパスへと展開される。これは USB スティックでポータブル版を使用する際に特に役立つ。というのも、その場合使用しているコンピュータによって実際のプログラムの配置が異なるからだ。

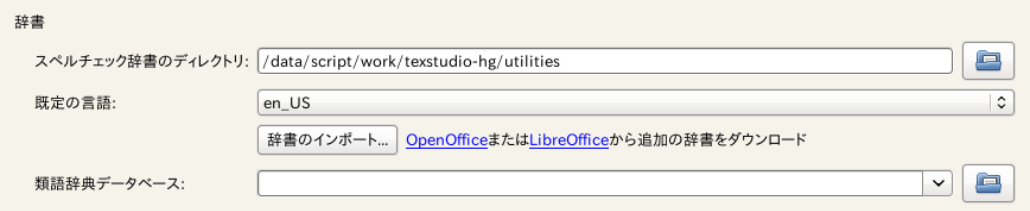

図 1.6　スペルチェックオプション

簡便のために、TeXstudio ではスペルチェッカーの言語を好きに選択できる。しかし、別の言語で書かれたファイルに対して頻繁に作業する場合規定の振る舞いを上書きしたいと思うかもしれない。これを行うには二つの方法がある。一つ目はステータス行の言語メニューを通じてファイルの言語を明示する方法だ。この設定はファイルを閉じると即座に失われる。ファイルの言語を永続的に保存するために、TeXstudio は特別な「マジックコメント」`% !TeX spellcheck = de_DE` をサポートしている。このコメントがファイルにある場合、ファイルを開くとその言語に自動的に設定される。

図 1.7　スペルチェックメニュー

注記：Ctrl+Shift+F7 でのスペルチェックはカーソル位置から開始されるのであって、文書の最初からではない。

もし対話式のスペルチェッカーが有効な場合（既定）、間違って綴られた単語すべてに赤い波下線が引かれる。その単語で右クリックすると、考えられる訂正語のリストのメニューが表示される。このコンテキストメニューでは無視するリストにその単語を追加することもできる。使用している辞書がとても大きい ($>$ 5MB) 場合、コンテキストメニューを開いて考えられる候補を表示するのに数秒かかるかもしれない。もし候補が不必要なら、右クリック中に Shift を押すと待つ必要がなくなる。

辞書の内部構造が複雑である（例えば、様々な変化形を持つ単語を生成する規則を含む）ので、単純に単語を辞書に追加することはできない。そのかわり辞書に単語がない場合、その単語を無視するリストに追加してスペルチェッカーが何も言わないようにすることができる。無視するリストは通常辞書と同じディレクトリに保存される。それは拡張子が ".ign" のプレーンテキストファイルである。もし辞書と同じディレクトリに保存できない（例えばアクセス権限がない）場合、そのリストはユーザーの設定ディレクトリに保存される。

1.5.2　類語辞典の設定

類語辞典では OpenOffice.org 2.x のデータベースを使用している。GPL のフランス語、アメリカ英語、ドイツ語のデータベースのみが TeXstudio に同梱されている。

ユーザーは次の場所から他のデータベースをダウンロードできる:http://wiki.services.openoffice.org/wiki/Dictionaries

1.5.3　LaTeX 構文チェッカーの設定

LaTeX 構文チェッカーでは、コマンドが正しいかどうか判断するために考えられる完全なコマンドのリストを利用している。更に、そのコマンドリストには、コマンドがその文脈で有効かどうか、数式モードでのみ有効かあるいは表モードでのみ有効かを決めるための追加情報が部分的に含まれている。

1.5.4　文法チェッカーの設定

文法チェッカーは LanguageTool の標準的な http API に基づいていて、LanguageTool(LT) と Java を別個にインストールする必要がある。

一度 LanguageTool をインストールすると、LanguageTool のスタンドアローンアプリケーションを起動し、その後 TeXstudio を起動することで文法チェッカーを利用できる。LanguageTool はアドレスが http://localhost:8081/ であるローカルで起動するサーバーを作成し、TeXstudio は起動時にそこに自動的に接続される。接続が確立されたら、すべての入力された段落が LT に送られ、ちょっとした後に考えられる文法上の間違いが強調表示される。

TeXstudio で LanguageTool を自動的に起動させるには、設定ダイアログの文法ページで LT jar のパスを入力する必要がある。Java の実行ファイルが既定の PATH にない場合、そのパスもそこで設定する必要がある。

高度な設定モードでは、ある LT の規則を「特別」として特徴づけすることもできる。その「特別」な規則に一致したものは異なる／カスタマイズ可能な方法で強調表示される。これは、例えば LT ですべての動詞またはすべての副詞を強調表示する独自の規則を作成することで、文体の解析を行うのに有益である。

LanguageTool とは独立して、TeXstudio では繰り返しの悪い（不正確な／俗語的な）単語もチェックされる。繰り返しの確認では、後ろのいくつかの単語を見て、身近なところの短い単語の繰り返しや前方 10 単語までの長い単語の繰り返しがしるし付けされる。これらの距離や長さは高度な文法設定のページで変更できる。

1.6　自動補完の設定

TeXstudio では、補完用の既知のコマンドの数をかなり増やした、kile の補完単語のリストを採用している。構文チェックだけでなく補完の有効なコマンドリストを選択するため、TeXstudio では`\documentclass` と`\usepackage` が使われていることが認識される。しかし、TeXstudio では「TeXstudio の設定」 -> 「コマンドの補完」で追加の単語リストを選択することができる。単語リストの名前とパッケージ名は対応関係がある。例えばリスト latex.cwl は標準的な LaTeX コマンドを含んでいる。

自動補完に関連して、TeXstudio では挙動を好みに合わせることができる。オプションは次のものが利用できる：

- 補完の有効化：これは自明である
- 大文字と小文字の区別：例えば`\la` から`\Large` の補完……
- 最初の文字で大文字と小文字の区別を行うかどうか
- 共通の接頭語の自動補完：リストに一つしか項目がない場合や補完リストのすべての項目の開始文字が共通である場合、Tab キーを押した場合と同様に、共通の文字が直接挿入される。
- 非文字キャラクタが押された時に選択したテキストを補完：補完モードではスペースのような非文字キャラクタを押すと、選択した単語を受け入れることになる。これによって入力が加速されうる。
- ツールチップヘルプの有効化：補完リストで選択した LaTeX コマンドのヘルプをツールチップとして表示する。

- プレースホルダーの使用：補完されたコマンドに記入すべきオプションがある場合、「プレースホルダー」がその場所に配置される。Ctrl+Right/Ctrl+Left でそこに移動できる。

好みのパッケージが補完（と構文チェック）にない場合、「packagename.cwl」というファイルを設定ディレクトリに配置することで独自のリストを提供できる。このディレクトリは Linux では“`~/.config/texstudio`”、Windows では通常“`c:\Documents and Settings/User/AppData/Roaming/texstudio`”である。基本的にファイルは有効なコマンドのリストを含んでいる。正確な書式と例の解説は cwl 形式の解説にある。

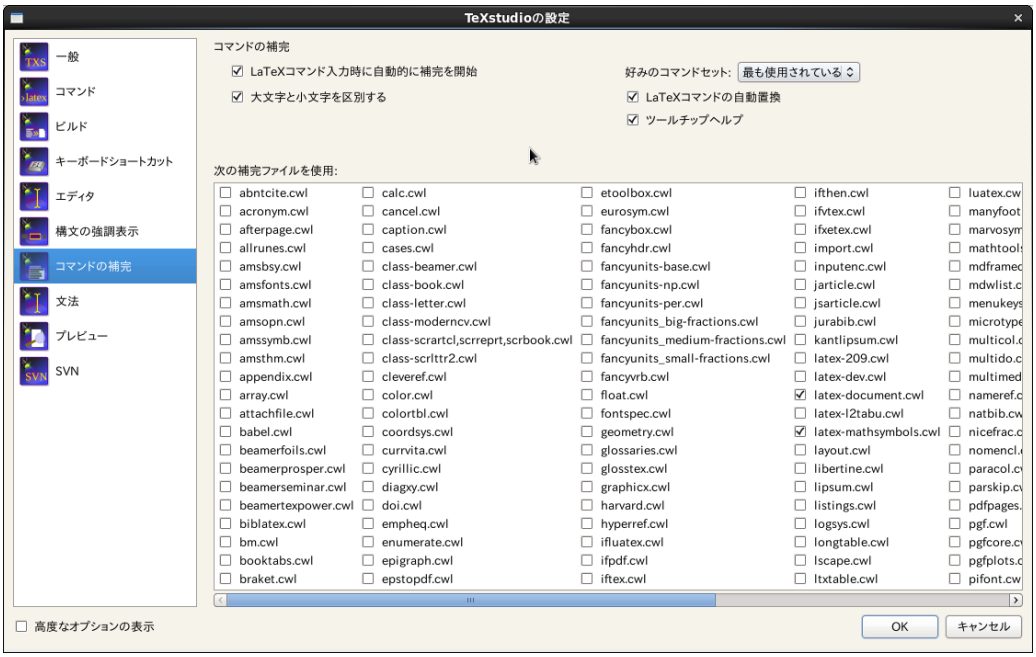

図 1.8　補完の設定

1.7　ショートカットの設定

ショートカットは、「現在のショートカット」または「追加のショートカット」の場所でダブルクリックを行うことにより変更できる。ショートカットは、ドロップダウンリストから選択するか、直接テキストとして入力することができる。ショートカットを規定値に設定もしくは完全に除去するのであれば、リストの最上部にある「<default>」または「<none>」を選択すればよい。

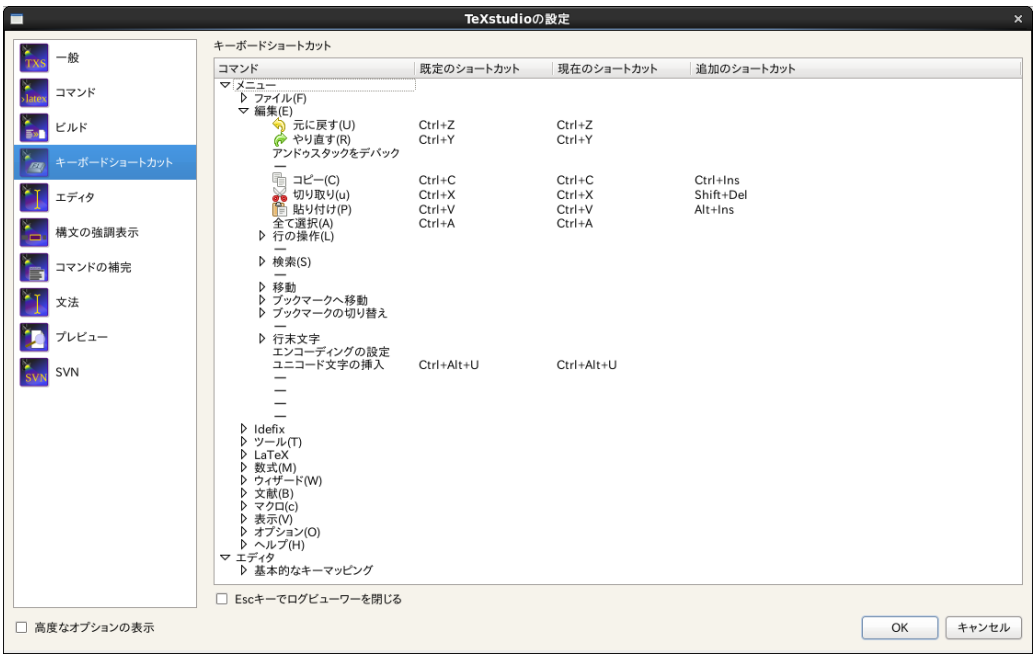

図 1.9　ショートカットの設定

1.8　Latex/Math メニューの設定（高度なオプション）

Math/Latex メニューはユーザーの好みに合わせることができる。このメニューに対して、項目の名前を変更したり、新規に LaTeX コードを追加したりすることが可能である。各項目は、ダブルクリックすることで直接編集することができる。

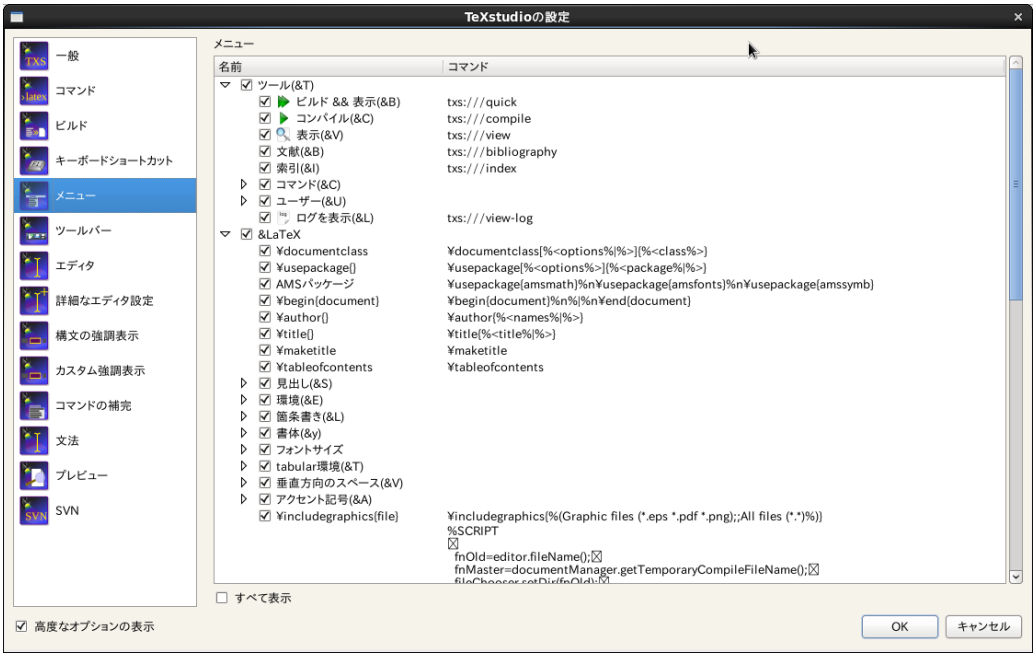

図 1.10　メニューのカスタマイズ

1.9　カスタムツールバーの設定（高度なオプション）

TXS にはカスタムツールバーがひとつ存在する。このツールバーは LaTeX、Math、ユーザーメニューの要素から成り立っている。これらの項目の多くにはアイコンがないので、ユーザーが好きなアイコンを読み出すこともできる。これは、設定ダイアログのカスタムツールバーリストで、項目のコンテキストメニューから「別のアイコンの読み出し」をすることで可能となる。

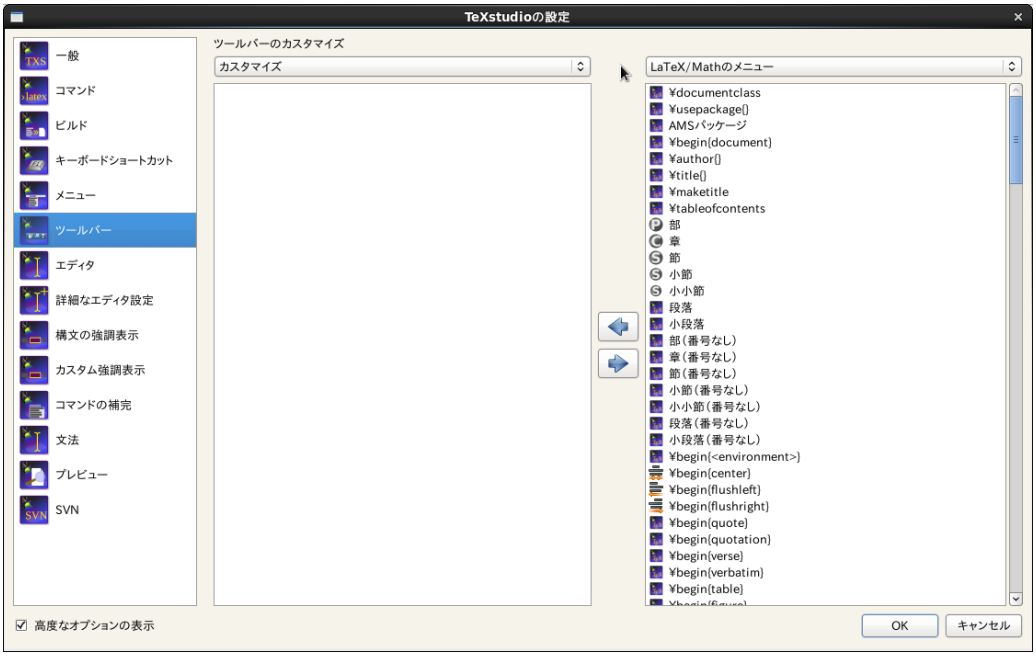

図 1.11　ツールバーのカスタマイズ

1.10　SVN サポートの設定

文章のバージョン管理を提供するため、TeXstudio では SVN(Subversion) を使用している。これを利用するには、SVN コマンドラインツールがインストールされている必要がある。Linux と Mac OSX では通常 SVN ツールが（パッケージとして）すでに提供されており、Windows では「SlikSVN」のインストールが推奨される。

TeXstudio のオプションのコマンドページの適切な欄に、「svn」と「svnadmin」のコマンドへの完全なパスが設定されている必要がある。SVN ページでは、自動化の度合いや WSVN を選択できる（下記参照）。

注：TeXstudio を通じてリポジトリをチェックアウトすることはできない。これを行うには通常のツールを使用するように（コマンドラインか選んだ GUI での svn checkout）。一度作業コピーができると、TeXstudio でそれを操作できる。

「保存後に自動的にチェックインする」を選択すると、TeXstudio が文書を保存するたびに SVN チェックインを行う。従って文書作成の非常に完全な履歴が残ることとなる。テキスト文書はディスクの空きスペースに比べて小さいので、SVN のデータベースのサイズは問題にならない。更に、ディレクトリがすでに SVN の制御下にある場合、（「名前をつけて保存」で）新たに保存したファイルは自動的に SVN の制御下に追加される。そうでない場合、TeXstudio は現在のディレクトリ上での「SVN ディレクトリの検索深度」のディレクトリ内で SVN の制御下のディレクトリを検索し、サブディレクトリと TeX 文書を SVN の制御下に追加する。適切なディレクトリが見つからない場合、リポジトリが「./repo」というディレクトリに自動的に作成され、文書が追加される。従って、ユーザーはリポジトリ設定のための必要なコマンドを探す必要はない。「自動チェックイン」を有効化した場合にのみこの機能は有効化される！

「最後の保存以前に戻すために SVN リビジョンを用いる」を使うと、TeXstudio はアンドゥを通常のように行う

が、内部記憶領域にアンドゥ可能なコマンドがそれ以上ない場合に、文書を SVN 履歴での一つ前のバージョンに変更する。さらなるアンドゥコマンドでより古いリビジョンへ戻すことが可能である一方で、「元に戻す」コマンドでより新しいバージョンへ変更できる。これは直接メニューコマンドを通して SVN のリビジョンを選択することよりもインタラクティブな手法である（4.4 章 SVN サポート参照）。

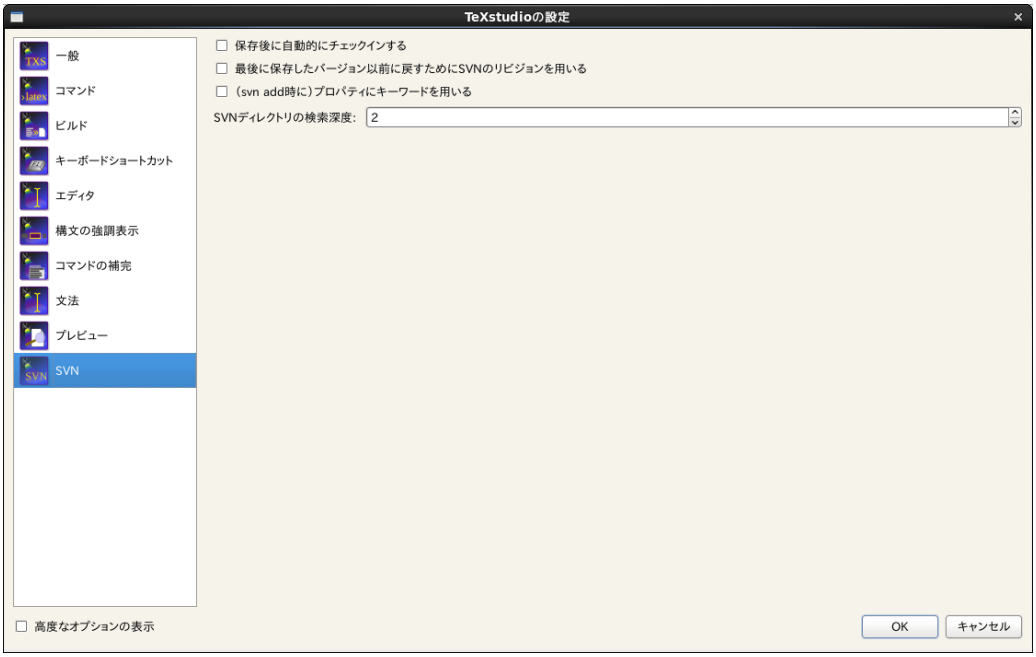

図 1.12　SVN の設定

第 2 章

TeX 文書の編集

2.1　通常のコマンド

標準的なコマンド（切り取り、コピー、検索…）は「編集」メニューと「編集」ツールバーを通して実行できる。

図 2.1　標準的なコマンド

2.2　新しい文書の作成

新しく文書を作成するには、次の小節で説明されるように 2 つの異なった方法がある。

2.2.1　TeX 文書のプリアンブルの設定

文書のプリアンブルを定義するには、(「ウィザード」メニューの)「簡単テンプレート」ウィザードが利用できる。

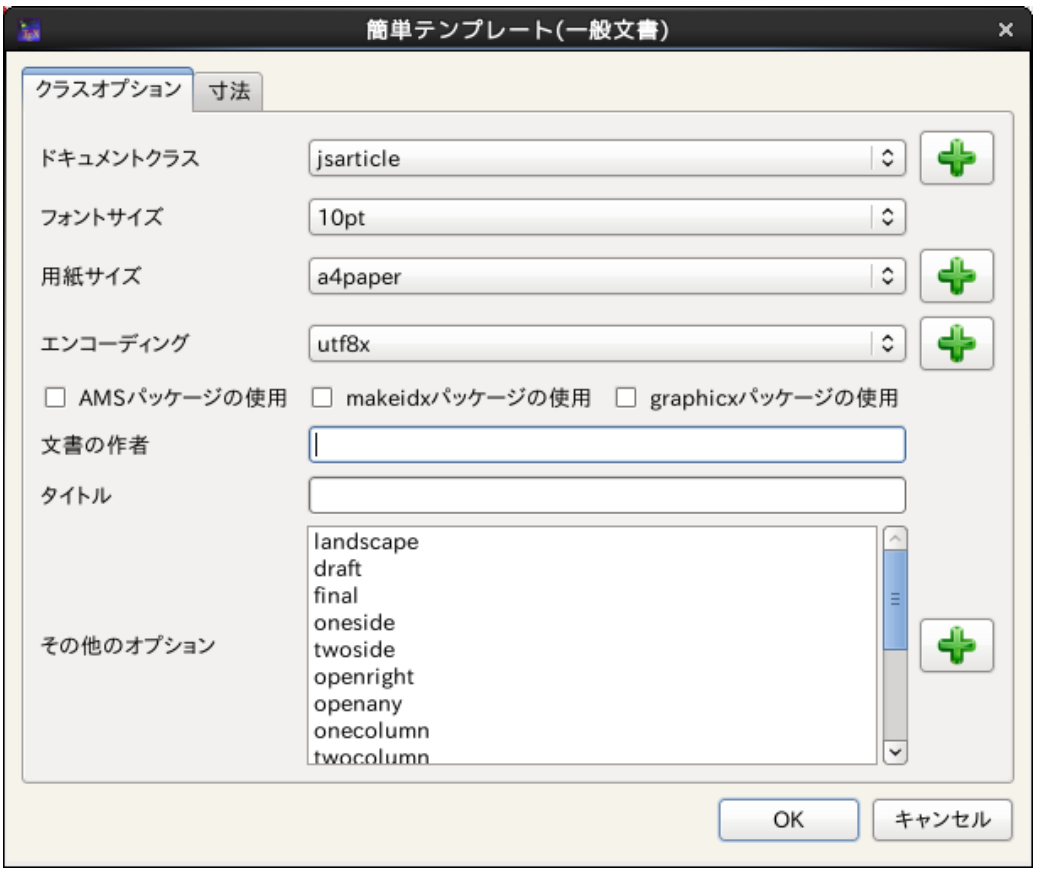

図 2.2　簡単テンプレート

このダイアログでは文書の主な仕様（文書クラス、用紙サイズ、エンコーディング、…）を設定することができる。

注意：“+” ボタンをクリックすることで他のオプションを追加することができる。また、設定全ては記録される。

また、エディターで自分自身のプリアンブルモデルを入力することもできる：「コピー／貼り付け」や「名前をつけて保存」コマンドで、新規文書にそれを利用できる。

2.2.2　新規文書作成時のテンプレート使用

新規文書に対して、「ファイル／テンプレートから新規作成」コマンドを用いてテンプレートを適応できる。ダイアログでは利用するテンプレートを選択できる。

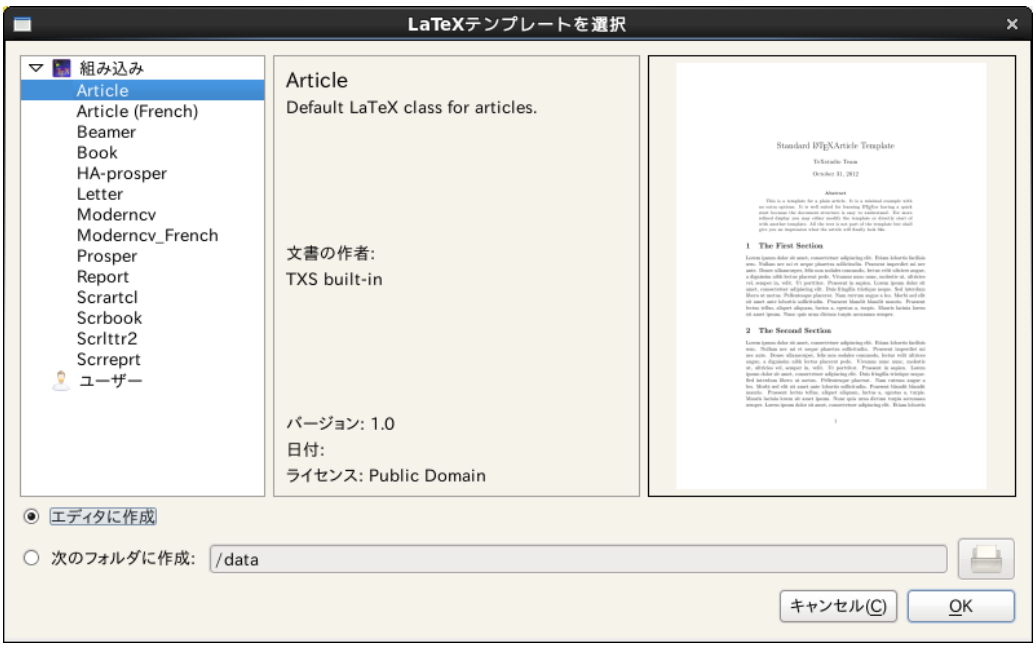

図 2.3　テンプレート

テンプレートから新しいエディター文書を作成するか、テンプレートをファイルとしてディスク上に作成してそれをエディターで開くか選択することができる。前者のオプションは複数のファイルのテンプレートに対して利用できない。

テンプレートとして利用したい開いている文書に対して、「ファイル／テンプレートを作成」コマンドを用いて新規テンプレートを作成することができる。このダイアログは現在テンプレートシステムの機能すべてをサポートしてはいないことに注意するように。特に、プレビュー画像を提供したり、画像つきの複数ファイルテンプレートを作成することはできない。このようなことは手動で行う必要がある（テンプレートの書式参照）。

ユーザーが追加したテンプレートは、テンプレート選択ダイアログでコンテキストメニューから編集や削除が可能である。しかし、組み込みのテンプレートは変更できない。

ユーザーテンプレートは、設定ディレクトリの`/templates/user/`サブディレクトリに保存される。

テンプレートの書式

最も単純な形式では、テンプレートは`.tex`ファイルのみである。複数ファイルテンプレートはすべての`.tex`ファイルを zip アーカイブにまとめることで作成することができる。追加で、同名だが拡張子が“.tex”や“.zip”の代わりに“.json”である別ファイルにメタデータが JSON 形式で保存される。現在次の項目がメタデータ内でサポートされている：

```
{
"Name"        : "Book",
"Author"      : "TXS built-in",
"Date"        : "04.01.2013",
"Version"     : "1.1",
"Description" : "Default LaTeX class for books using separate files for each
    chapter.",
"License"     : "Public Domain",
"FilesToOpen" : "./TeX_files/chapter01.tex;main.tex"
}
```

FilesToOpen は複数ファイルの文書に対してのみ影響がある。テンプレートファイルの隣にプレビュー画像を付け加えてもよい。ただ、ファイル名は同名で拡張子が ".png" でなければいけない。

2.3　文書構造

TeXstudio で文書に新しい部分（節、小節、……）を定義するには、ツールバーのこのコンボボックスボタンを使用すれば良い：

図 2.4　区分化

2.4　文書の閲覧

「構造ビュー」（左側パネル）を用いると、文書のあらゆる部分にすぐに移動できる。何らかの項目（ラベル、節、……）をクリックしさえすれば、エディタの対応する場所の先頭へ移動する。ある行へ移動する仕組みでは、もはや行数を考慮するだけでなく実際のテキスト行をも記憶している。従って、行を追加や削除しても間違った位置へ移動することはない。

灰色の背景はテキストと構造ビューでの現在のカーソル位置を示している。緑の背景は付録での節を表している。

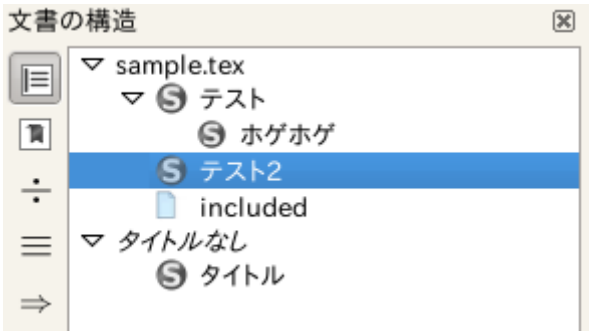

図 2.5　構造ビュー

「構造ビュー」はタイプしたとおりに自動的に更新される。また、いつでも（「Idefix」メニューの）「文書構造の更新」コマンドを用いることができる。

ラベル、節、`include` や `beamer` ブロックの他に、`%TODO` で始まるコメントもスキャンされて構造ビューに項目として表示される。これはテキストに一種の永久ブックマークを作成したり、変更が依然として必要な場所を書き留めるために使用することができる。

また構造ビューでは、（小節を含む）節に属するテキストすべてをコピー／切り取りして別の節の前または後ろに貼り付けるコンテキストメニューが利用できる。更に節のインデント／インデントの解除をすることが可能である。それは階層構造のレベルを変更することを意味している。つまり`\section` が`\subsection` に変更され、全ての小節がそれに応じて扱われる事になる。

それぞれのファイルに対して、移動の高速化のため 3 つのブックマークが利用できる：ブックマークに追加または削除するには行番号をクリックすれば良い。すでに 3 つのブックマークが定義されている場合、新たなブックマークを追加するためそのうち 1 つを削除しなければならない。エディタ上でブックマークに対応する行へ移動するには、ステータスバーのブックマークボタンをクリックすれば良い。

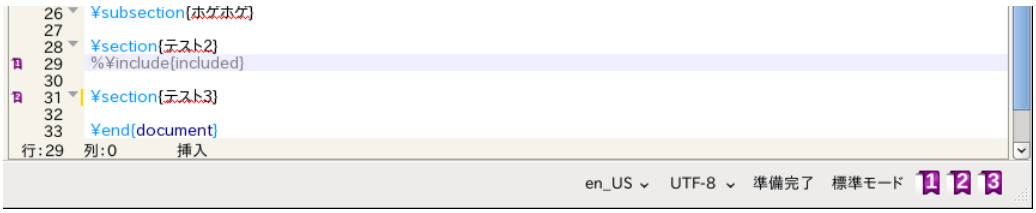

図 2.6　ブックマーク

2.5　テキストの整形

このツールバーでテキストの一部の書式を簡単に設定することができる：

図 2.7　書式ツールバー

追加のオプション：選択したテキストを特定の環境で直接囲うこともできる。例：“Hello” という単語を選択した後「ボールド体」のボタンをクリックすることで、次のコードが入力される：`\textbf{Hello}`。

このオプションは「LaTeX」メニューの “[選択]” で示されている環境全てに対して利用できる。

2.5.1　大文字使用

メニュー「編集」 -> 「テキスト操作」には選択したテキストの大文字使用を変更する方法がいくつか含まれている：

- 小文字化
- 大文字化
- 厳密なタイトルケース（先頭は大文字で他は小文字）化
- スマートなタイトルケース（先頭は大文字で他は小文字）化

「タイトルケース化」の両方共、短い単語（a, the, of など）は小文字のままにする。加えて、「スマートなタイトルケース化」は、大文字を含む単語を大文字使用の固定を必要とする頭字語（例：「TeXstudio」）と仮定して変換しない。

2.5.2　予約語のエスケープ

もし、TeX の予約語を含むテキストがあり、文書中でその文字どおりにテキストを表示したい場合、LaTeX が解釈するのを防ぐためその予約語をエスケープする必要がある。次の機能でそれを扱うことができる（メニュー：Idefix）

- LaTeX として貼り付け：クリップボードからテキストを取得して、予約語をエスケープしてエディタに貼り付ける。

- LaTeX に変換：現在の選択部内の予約語をエスケープする。

例：“Less than 10% of computer users know the meaning of $PATH.” は次のように変換される：“Less than 10\% of computer users know the meaning of \$PATH.”

2.6　スペース

通常の「スペース」コマンドは「LaTeX」と「数式」メニューで利用できる。「強制改行」の LaTeX コマンドを簡単に挿入するには、ツールバーの対応するコマンドを利用できる（ショートカットキー：Ctrl+return）。

2.7　リストの挿入

通常の箇条書き環境コードは「LaTeX->箇条書き」メニューから簡単に挿入できる。
注：`\item` コマンドのショートカットキーは Ctrl+Shift+I。

2.8　表の挿入

「表作成」ウィザード（「ウィザード」メニュー）で、表環境に対応する LaTeX コードを簡単に挿入することができる：

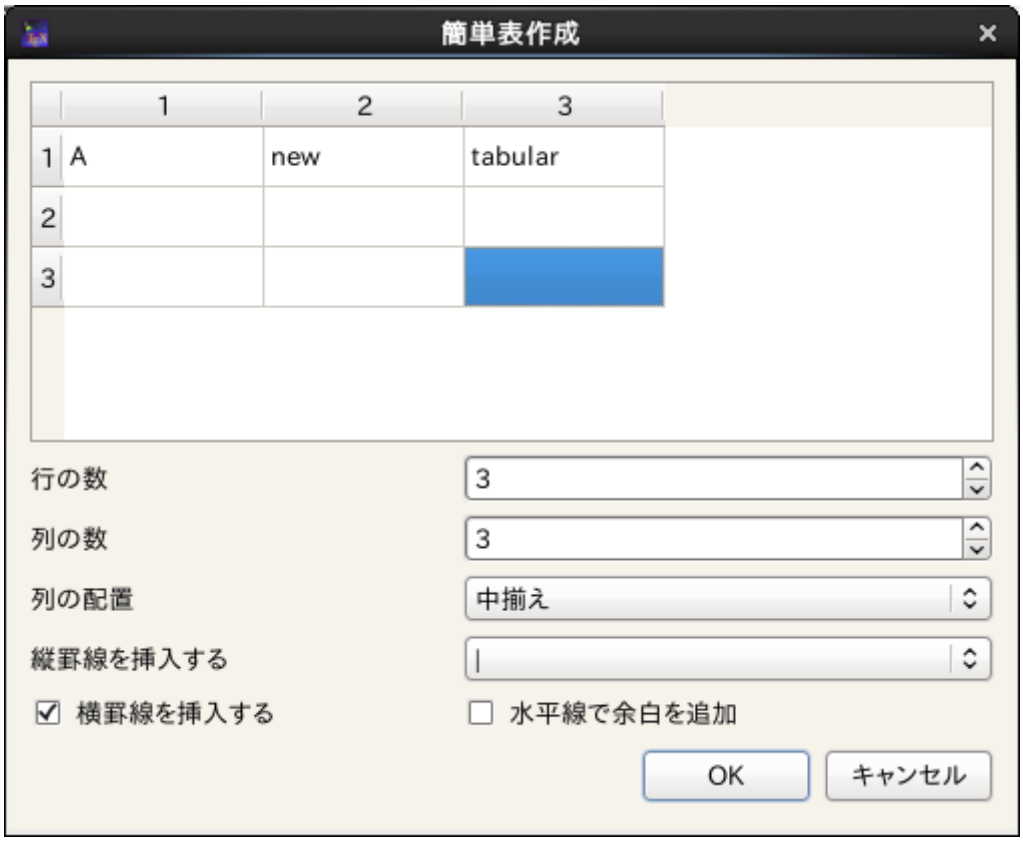

図 2.8　表ウィザード

このウィザードで表の主な特徴を設定することができる。
注：このダイアログでセルにコードを直接入力することができる。
対応する LaTeX コードが自動的にエディタに挿入される。

2.8.1　表の操作

TeXstudio では表を簡単に操作できるコマンドが提供されている。そのコマンドは LaTeX → 表の操作と表ツールバーにある。表構築のコマンドが複雑になりすぎると予期しない結果となるかもしれないことを認識しておくこと。次のコマンドが提供されている：

- 現在の行の後ろに行を追加
- 行を削除：カーソルがある行を削除する
- 列の追加：現在のカーソル位置の後ろに列を追加する。カーソルが行頭（第一列）にある場合、列は新しい第一列として追加される。
- 列の削除：現在の列を削除する
- `\hline` の追加／削除：現在の行以降の全ての行に対して`\hline` を追加／削除する。すでに`\hline` のコマンドが存在している場合、二番目のコマンドが置かれることはない。
- 列の整列：列区切り記号（&）を空白記号を挿入することで揃える。セルのテキストは表のヘッダーの指定に従って揃えられる。これは表のソースを読むのに役立つ。
- テンプレートを用いて表を再構築する。これによって文書で同一の表の構築を行うことができる。いくつかのテンプレートが予め定義されており、java script のプログラミングを通して更にテンプレートを追加することができる。このコマンドはメニューにしか存在しない。

TeXstudio ではブロックカーソルも利用できる。<Ctrl>+<Alt>+<Shift>を押してマウスでカーソルをドラッグすることで利用できる。ブロックカーソルは通常のカーソルの組のように機能する。通常と同じくテキストのコピーや貼り付けができる。また新たなテキストを入力すると、全ての行にそのテキストが追加される。

```
\begin{tabular}{lcr}
    Left aligned & centred & right aligned \\
    0  cm        &    1    &              1 \\
    2  cm        &    3    &              8 \\
    13 cm        &   21    &             34 \\
\end{tabular}
```

図 2.9　ブロック選択

2.9　“tabbing” 環境の挿入

“tabbing” コードを挿入するのを容易に行うため、(「ウィザード」メニューの) “Tabbing” ウィザードを利用できる：

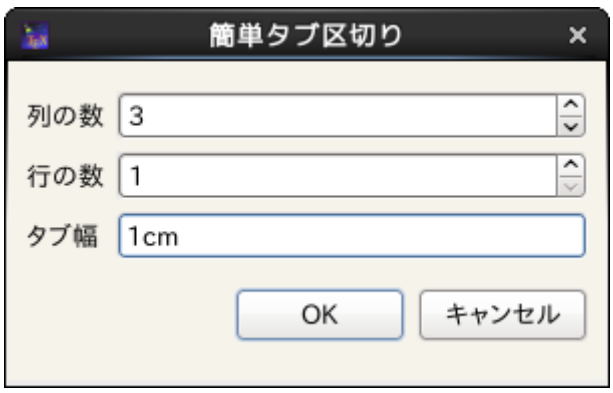

図 2.10　Tabbing ウィザード

2.10　図の挿入

文書中に図を挿入するには、「LaTeX」メニューの “`\includegraphics`” コマンドを使用すれば良い。そして画像ファイルを選択するためダイアログの「ブラウザ」ボタンをクリックすれば良い。

注：図の挿入前に（「LaTeX - 環境」メニューの）“figure” LaTeX 環境を挿入してもよい。

図 2.11　図環境

2.10.1　「ウィザード」を用いた図の挿入

図の適切な挿入は、LaTeX 初心者には挑戦であり、熟練者にはほんの僅かのテキストをタイプすることである。従って TeXstudio では文書への画像挿入コードを扱うウィザードを提供している。「画像オプション」では `\includegraphics[options]{file}`のオプションパラメータを定義する。最も使用される幅／高さの属性は容易に設定できる一方で、ユーザー定義の設定で完全に制御することもできる。

画像をテキスト中のまさにその位置に配置する必要がない場合、画像を `figure` 環境内に置くべきだ。そしたら LaTeX の方でページ上の最適な位置を決定してくれる。

「既定として保存」ボタンを押すことで、現在の設定（ファイル、図見出し、ラベルを除く）が保存される。そしてウィザードを開いた際その設定を既定として使用することができるようになる。

画像ファイルを文書にドラッグ＆ドロップしたり、エクスプローラーでコピーして TeXstudio で貼り付けたりした時にも、画像挿入ウィザードが起動する。これによって、調整可能な既定パラメータを伴って新しい画像を非常に素早く挿入できる。更に図のコード上にカーソルがあるときに画像挿入ウィザードを開始すると、既存の図の設定をそのウィザードで操作することができる。

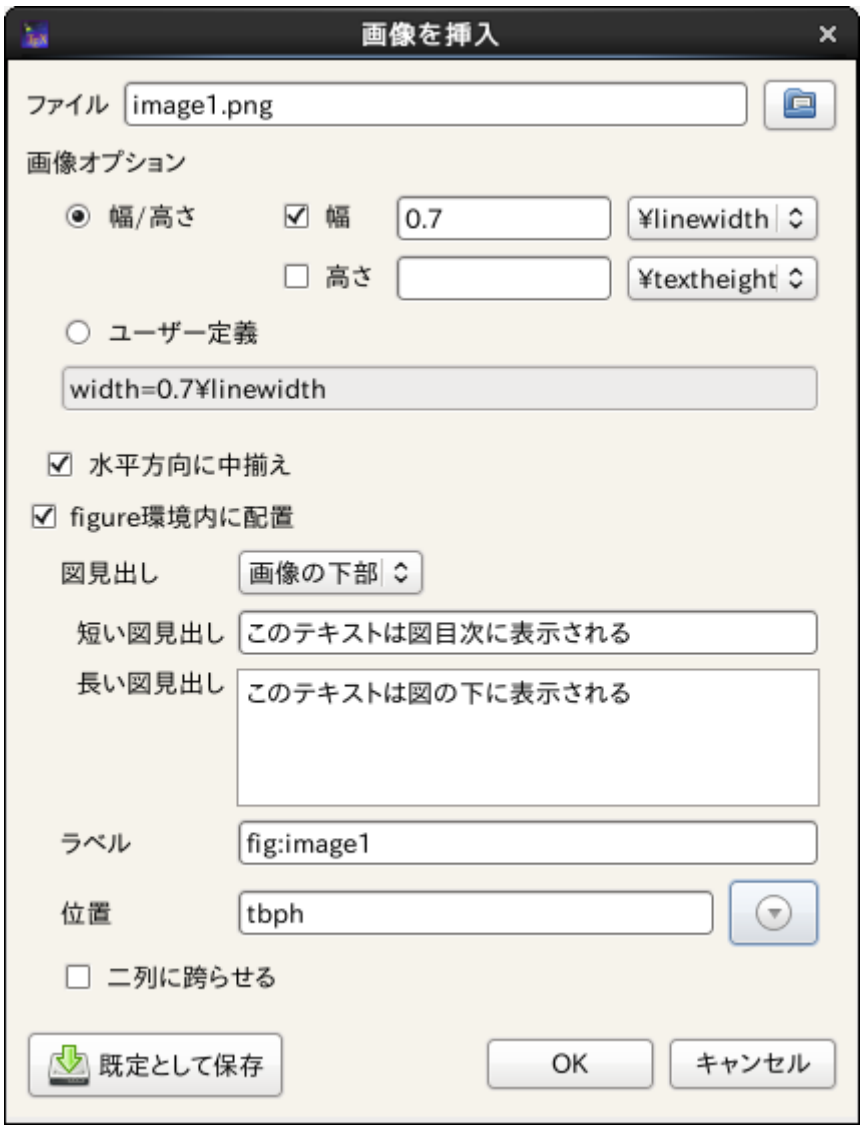

図 2.12　図ウィザード

2.11　相互参照及び注釈

ツールバーのこのツールボックスでラベルや引用、参照、脚注などのコードをすぐに挿入できる。
注：文書中で用いられているラベルは「構造ビュー」に表示される。

図 2.13　ラベルなどのツールボックス

追加のオプション：`\ref` コマンドに対しては、ダイアログボックスで直接ラベルを選択することができる。

2.12　数式の挿入

ツールバーの「インライン数式」ボタン（ショートカット：Ctrl+Shift+M）または「数式」メニューで、「インライン数式」環境内の状態へと切り替えることができる。「ディスプレイ数式」環境のショートカットキーは次である：Alt+Shift+M。
「数式」ツールバーで`\left` や`\right` タグのような最も使用される数学的な形 (frac, sqrt...) を挿入できる。

図 2.14　数式ツールバー

また、「文書の構造」ビューの「記号パネル」を用いて、400 種類の数学記号のコードを挿入することができる。

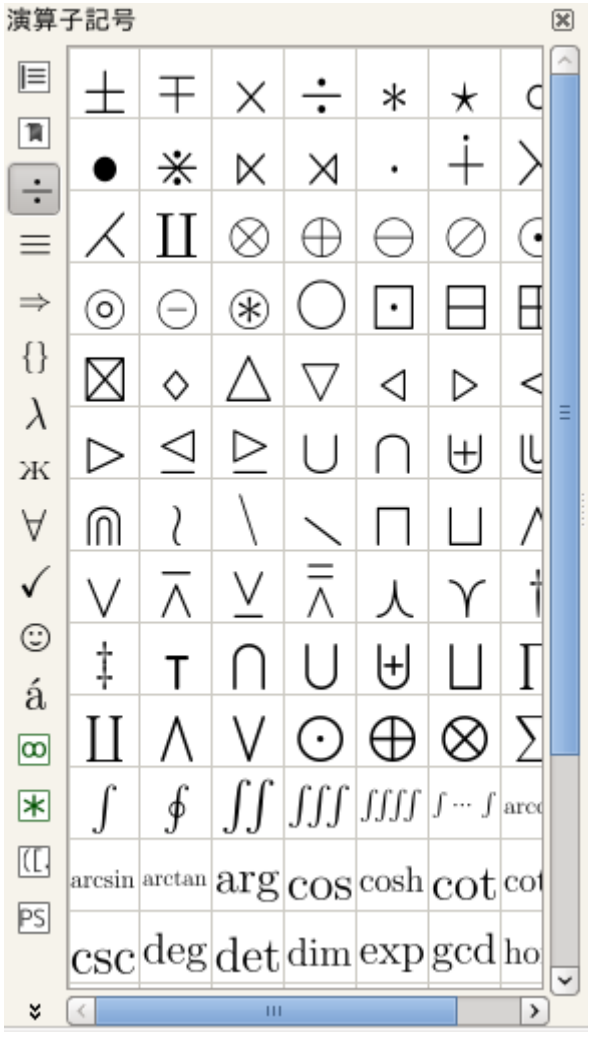

図 2.15　数学記号パネル

「数式」メニューを通して数学的テキストの書式を決めることもできる。

“array” 環境に対しては、(「表」ウィザードのように)「ウィザード」メニューからウィザードが利用できる。このウィザードでは、使用する環境（array, matrix, pmatrix など）を選択することができる。更にセルを直接埋めることもできる。

図 2.16　行列ウィザード

2.13　自動補完

“\” に続いて文字を打つと常に、考えられる LaTeX タグのリストが表示され、正しいものを選択することができる。追加の文字を打った場合、LaTeX タグリストはフィルター処理されて、すでに書かれたテキストで始まるタグのみが表示される。タグリストが同じ文字の組み合わせで始まる単語だけを含む場合、Tab キーを押して全てに共通する文字を補完することができる。もしタグリストに一つしか要素がない場合、Enter キーのように Tab キーでこれを選択して補完することができる。この振る舞いは bash シェルでの tab 補完に似ている。また、望むときに Ctrl+Space を押してこのタグリストを開くことも可能である。

タグに異なるオプションがある場合、短い説明的なテキストが挿入され、それぞれのオプションの意味を教える。更に、Ctrl+Left と Ctrl+Right を押してあらゆる位置を選択することができる。

また通常のテキストも単語をタイプし始めて Ctrl+Space を押すことで補完を行うことができる。現在の文書の適切な単語全てが考えられる候補として用いられる。

環境を挿入するつもりであれば、環境名の最初をタイプして Ctrl+Alt+Space を押すことで、適切な環境の候補が表示されて`\begin{env}..\end{env}`で補完挿入される。

そして最後に、ユーザータグを補完で使用することもできる略語へ割り当てることが可能である。略語の最初の部分をタイプして Ctrl+Space で補完を開始するだけでよい。そうした略語は、特に “略語（テンプレート）” でしるし付けされて補完リストに表示される。

もし新たなコマンドを補完することでとあるコマンドを変える場合、コマンド名のみが取り替えられる。同じことが環境に対しても当てはまり、`\begin`- & `\end`-コマンド内の環境名が変化する。

2.14　類語辞典

TeXstudio は単純な類語辞典を統合している。これには OpenOffice 2.x のデータベースを使用している。単語上にカーソルを置いて類語辞典をアクティブにする（Ctrl+Shift+F8 またはツール／類語辞典）ことで、この単語に対する類義語を見つけようとする。類語辞典を初めて起動する場合には、データベースの読み込みが生じて少し時間がかかりうるので我慢すること。

図 2.17　類語辞典

左側の最初の行は類義語を探索する対象単語を含む。その下のリストは単語の種類のリストである。それらのいずれかを選択して候補の数を減らすことができる。右側の欄は提案された類義語のリストを含む。このリストから選択した単語が、そのテキストの代わりに対する提案として右側の最初の行に表示される。この単語は手動で変更できる。その単語は、それ「で始まる」またはそれ「を含む」単語や類義語に対するさらなる調査にも用いられる。「調べる」でその単語の類義語を探すために直接利用することもできる。

2.15　特殊コマンド

2.15.1　単語／コマンド／環境の削除

Alt+Del のショートカットでカーソル位置の単語が削除される。それがコマンドならば、そのコマンドが開き括弧と閉括弧を含んで削除される。例 : “`\textbf{text}`” は “text” が残ることになる。もしも環境だった場合、取り囲む begin/end が削除される。

2.15.2　環境名の付け替え

カーソルが環境名もしくは対応する begin-/end-コマンド上にある場合、少ししてミラーカーソルがアクティブになる。これで begin-&end-コマンドの環境名を同時に変更することができる。もし “`\begin{tabular}...\end{tabular}`” 構文を “`\begin{tabularx}...\end{tabularx}`” へ変更したい場合、テキストカーソルを “tabular” の上に置き「環境名の付け替え」を実行して少し待てばよい。すると、ミラーカーソルが現れて “tabular” を “tabularx” に変更できるようになる。

2.15.3　バッファの切り取り

何かを選択してコマンドを入力し始めて補完を行う場合、選択したものが最初の引数として配置される。例 : “text” があり、それを選択して “`\textbf`” を補完入力する。すると結果として得られるテキストは “`\textbf{text}`” となる。

第 3 章

文書のコンパイル

3.1　コンパイル

文書をコンパイルする最も簡単な方法は、「コンパイル」コマンドまたは「ビルド＆表示」コマンド（「コンパイル」ボタン - ショートカット：F6、「ビルド＆表示」ボタン - ショートカット：F1）を使用することである。「TeXstudio の設定」ダイアログを通して既定のコマンドを選択することができる。

（また、「ツール」メニューで各々のコマンドを一つ一つ起動することも可能である。）

注：「ツールメニュー」の「補助ファイルの削除」で LaTeX のコンパイル時に生成されるファイル (dvi, toc, aux...) を削除することができる（ps & pdf ファイルを除く）。

図 3.1　コンパイルツールバー

注意：すべてのファイルには拡張子がなければならず、「タイトルなし」ファイルや名前に空白のあるファイルはコンパイルできない。

3.2　ログファイル

「ビルド」コマンドで、ログファイルが「メッセージ／ログファイル」パネルに自動的に表示される。もしコンパイル時にエラーが生じれば「エラー」パネルにそのエラーが表示される。「エラー」パネル上の「行」列の数字をクリックすれば、エディタ上で対応する行にカーソルが移動してそのエラーが「ログファイル」パネル上でも表示される。

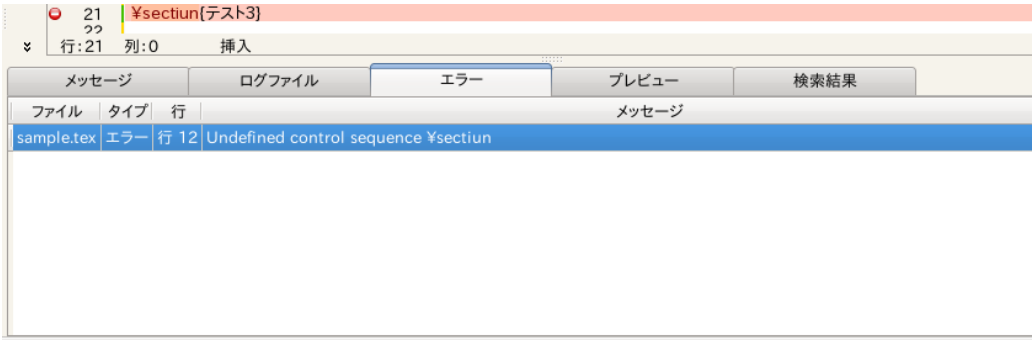

図 3.2　ログファイル

「次のエラー」と「前のエラー」コマンドでコンパイル中に検出されたエラーの間を移動できる。

エラーや警告、良くないボックスのある行は背景がそれぞれ赤、黄色、青で強調表示される。また、Ctrl+Up/Down でそれらの間を移動できる（エラーのみに対しては Ctrl+Shift、警告のみに対しては Ctrl+Alt、良くないボックスのみに対しては Alt+Shift を用いる）。

更に、それらの行へ移動するとツールチップで間違いのさらなる詳細が表示される（行番号の左側の印にマウスを合わせた場合にも表示される）。

第 4 章

その他の機能

4.1 複数のファイルに分割してある文章について

LaTeX 文書は複数のファイルに渡っても良い。TeXstudio は自動的に読み込んだ文書の親／子関係を把握する。このことには、ルートドキュメントの検出と定義されているラベルとコマンドの把握が含まれる。

4.1.1 ルートドキュメント

ルートドキュメントは、複数ファイルの文書における一番上のファイルである。単一ファイルの文書では、これはそのファイル自身である。既定では、LaTeX を呼び出す全ての動作はルートドキュメントに対して行われる。

TeXstudio は自動的にルートドキュメントを検出する。もしこの機能がうまくいかない場合は、ファイルの一番最初にマジックコメント `% !TeX root = root-filename` を置いても良い。

最後の手段として、「オプション -> ルートドキュメント -> 明示的に現在の文書をルートとして設定」を使って**明示的なルートドキュメント**を設定しても良い。この設定は絶対的なものである。エディタ上でどの文書を編集していても、「ツール」メニューのコマンド全てはこのルートドキュメントに対して呼び出される（より正確には、ビルドシステムはプレースホルダ % をルートドキュメントに展開する）。さらに、あらゆる開いてある文書内で定義されているラベルとユーザーコマンドは、あらゆる開いている文書で補完の為に利用される。

以前のバージョンでは、**明示的なルートドキュメント**は**マスターファイル**と幾分誤解を生む呼び方だった。

4.1.2 読み込まれた文書

明らかに、TeXstudio が利用できるのは認識している情報（定義されたコマンド、ラベル、文書構造など）のみである。我々は全ての開いているファイルでその情報を利用するのだが、複数ファイルからなる文書内のラベルが読み込まれていないファイル内で定義されている場合、TeXstudio はそれを認識できず参照の欠損として印付けする。これに対処するには、対応するファイルも開くだけでよい。

より新しいバージョンの TeXstudio には高度なオプション「エディタ -> 含まれるファイルを自動的に読み込む」がある。これは、古いシステムでのパフォーマンスの理由から既定では無効になっている。このオプションを有効にすると、複数ファイルからなる文書のうち一つのファイルを開くと、即座に TeXstudio は文書に関わるファイルすべてを自動的に読み込み解析する。 ルートドキュメントを開いた状態でない場合、マジックコメント `% !TeX root = root-filename` を設定する必要があるかもしれない。このオプションを有効にしておくと、TeXstudio は常に文書全体を認識して、強調表示や補完を行うときにそれに応じて振る舞う。

4.2 構文チェック

LaTeX 構文チェッカーは、コマンドが正しいかどうか決めるため考えられる補完コマンドのリストを採用している。どの文脈でコマンドが有効であるか、例えばコマンドが数式モードでのみまたは表モードでのみ有効なのかどうかを決めるため、補完リストは部分的に追加情報を含んでいる。

更に表の正確さはより詳細に確認される。列の数が続く行で解析・確認される。もしある行で列の数が多かったり少なかったりすると、警告印が表示される。

4.3　参考文献

“bib” ファイルに対して、「文献」メニューで文書の標準型に対応する項目を直接挿入できる。
注：「文献」メニューの「削除」コマンドでオプション項目が自動的に削除される。

図 4.1　参考文献

4.4　SVN サポート

第 1.10 節（SVN サポートの設定）ですでに述べたサポートされている svn 機能とは別に、TeXstudio はさらに 2 つのコマンドをサポートしている。

「ファイル／ SVN ／チェックイン」で、svn 履歴に保存されるチェックインメッセージを求める入力ダイアログとともに明示的に保存、チェックインが行われる。

「ファイル／ SVN ／古いリビジョンを表示」で、全ての利用可能なリビジョンを表示するダイアログが表示される。古いリビジョンを選択することで、その古いリビジョンへ現在の文書が即座に変更される。また、古い部分をコピーして最新のバージョンへ戻ることで、その部分を文書の最新バージョンにすることができる。もしその文書を直接編集し始めると、ダイアログは閉じて現在のテキストが未保存だが新しい最新バージョンとなる。

4.5　私的マクロ

TeXstudio では自分のマクロを挿入できる（ショートカット：Shift+F1...Shift+F10）。これらのマクロは「マクロ - マクロを編集」メニューで定義できる。マクロは TXS に直接配置される単純なテキストからなる。また、それは begin/end で自動的に展開される「環境」でも、java script でも良い。必要な機能はチェックボックスで選択することができる。

「略語」は LaTeX 補完に対する擬似コマンドである。擬似コマンドが補完されると代わりにそのマクロが挿入される。擬似コマンドはバックスラッシュ (“\”) で始まる必要があることに注意すること。

「トリガー」はマクロを含むものを自動実行する正規表現である：最後に入力された文字がこの表現に一致すると、それは削除されてマクロが挿入／実行される（詳細はトリガーを見ること）。

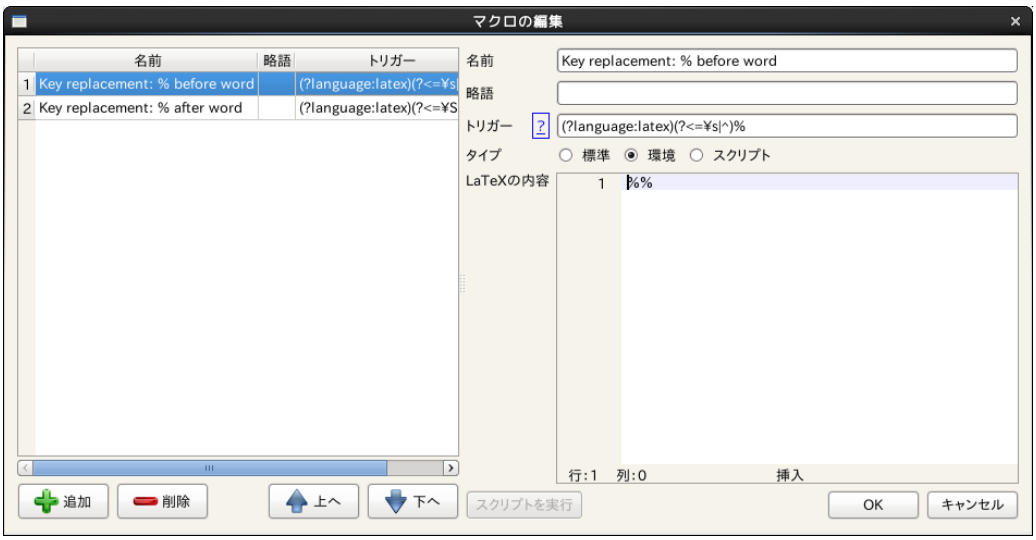

図 4.2　マクロの編集

4.5.1　テキストマクロ

通常のテキストとは別に、いくつかの特別なコードが認識されて挿入時に置換される。

もしどこかに `%|` を書いた場合、カーソルは挿入されたテキストのその場所に配置される（2 番めの `%|` はそれらの間全ての選択になる）。

`%<something%>` を書くと、Ctrl+Left/Right で選択できる説明的テキストとしてしるし付けされる。

オプション `%(filefilter%)` はファイルダイアログで尋ねられるファイル名で置換される。ファイルフィルター *filefilter* は標準的な Qt ファイルフィルター形式である。

例：“Images (*.png *.xpm *.jpg);;Text files (*.txt);;XML files (*.xml)”。Qt-Doc も参照すること。

4.5.2　環境マクロ

テキストは環境名として用いられる。従って “`%environment`” は次のように挿入される：

```
\begin{environment}

\end{environment}
```

注：TeXstudio では、挿入時に配置されないものの環境名が “%” で始まる必要がある。

4.5.3　Javascript マクロ

コード片を用いる代わりに、javascript を使用することもできる。そのためには最初の行に “`%SCRIPT`” を置けばよい。続くコードは javascript として解釈される。言語は ECMAScript に基づく。文書にアクセスするために次のオブジェクトが導入されている：

- “editor” で検索／保存／読み込みといった最上位の操作を現在の文書で行うことができる。
- “cursor” で移動、テキストの挿入や削除といったカーソル操作にアクセスすることができる。
- “fileChooser” で非常に単純なファイル選択ダイアログにアクセスすることができる。
- “app” でクリップボードやメニューといったアプリケーションの幅広いものへアクセスすることができる。

次の表は考えられるコマンドの概要である。

表 4.1　グローバルスコープのコマンド一覧

グローバルスコープ	
コマンド	説明
alert(str), information(str), warning(str) または critical(str)	文字列 str をあるアイコンとともにメッセージボックスに表示
confirm(str) または confirmWarning(str)	文字列 str をはい／いいえ (yes/no) の質問としてメッセージボックスに表示
debug(str)	文字列 str を標準出力 stdout に表示
writeFile(name, value)	ファイル name に値 value を書き込む（書き込み権限を要求）
readFile(name)	ファイル name 全体を読み込む（読み込み権限を要求）
system(cmd)	コマンド cmd を呼び出し、次のメソッドをもつ ProcessX オブジェクトを返す： • waitForFinished: プロセスが終了するまで待つ • readAllStandardOutputStr: 標準出力 stdout を返す • readAllStandardErrorStr: 標準エラー出力 stderr を返す • exitCode: 終了コード • exitStatus: Qt 終了ステータス • terminate または kill: プロセスを停止する
setGlobal(name, value)	一時的なグローバル変数を設定する
getGlobal(name)	グローバル変数を読み込む
hasGlobal(name)	グローバル変数の存在を確認する
setPersistent(name, value)	グローバル設定変数を設定する（ini ファイルの値を変更することができる。書き込み権限を要求）
getPersistent(name)	グローバル設定変数を読み込む（ini ファイルの値全てを読み込むことができる。読み込み権限を要求）
hasPersistent(name)	グローバル設定変数が存在するかを確認する（読み込み権限を要求）
hasReadPrivileges()	スクリプトに読み込み権限があるかを確認する
hasWritePrivileges()	スクリプトに書き込み権限があるかを確認する
registerAsBackgroundScript([id])	スクリプトがバックグラウンドで実行できるようにする（スクリプトがイベント／シグナルを扱う場合には必要）
triggerMatches	スクリプトがエディタトリガーで呼び出された場合、通常のトリガー表現に一致する
triggerId	スクリプトがイベントトリガーで呼び出された場合、トリガーの id の数字を表す
include(script)	別のスクリプトを読み込む。script はファイル名やマクロ名である。
pdfs	全ての開いている内部 pdf ビューワーのリストである

表 4.2: エディタオブジェクトの一覧

エディタオブジェクト	
コマンド	説明
`editor.search(searchFor, [options], [scope], [callback])`	エディタ上で検索を行う。 • `searchFor` は検索対象テキストである。文字列（例：“..”）または正規表現（例：`/[.]{2}/`）のいずれかである。 • `options` は文字列で、“i”、“g”、“w” の組み合わせである。これにより大文字・小文字の区別なし (case-*i*nsensitive) 検索かグローバル (*g*lobal) 検索（最初の一致発見後も続ける）、あるいは単語単位での (whole-*w*ord-only) 検索かを指定する。 • `scope` は検索範囲を制限するカーソルである（`editor.document().cursor` を見よ）。 • `callback` は一致ごとに呼び出される関数である。一致位置を表すカーソルが最初の引数として渡される。 `searchFor` 以外の引数は全て省略可能であり、順番は変更してもよい（将来の互換性はないかもしれない）。関数は見つかった一致したものの数を返す。
`editor.replace(searchFor, [options], [scope], [replaceWith])`	エディタ上で検索と置換を行う。これは、単純な文字列またはコールバック関数の `replaceWith` 引数を別として `editor.search` のように振る舞う。`replaceWith` が関数の場合、その返り値は `replaceWith` に渡されたカーソルで表される一致物を置換するのに用いられる。

表 4.2: 表の続き

`editor.replaceSelectedText(newText, [options])`	この関数は現在の選択部を newText で置換する。何も選択されていない場合 newText を挿入する。 newText が関数の場合、選択テキストと対応するカーソルを引数としてその関数が呼び出され、戻り値が newText となる。 全テキスト置換／挿入にこの関数を使用することが推奨される。なぜならこれが複数のカーソル／ブロック選択を正しく扱う最も簡単な方法だからだ。ただしこの関数は TXS >= 2.8.5 でのみ利用できる。 オプション (options) は次のプロパティを持つオブジェクトである： • `{"noEmpty": true}` 置換のみ； 選択部が空の場合何も挿入しない • `{"onlyEmpty": true}` カーソル位置に挿入するのみ；空でない選択テキストを変更しない • `{"append": true}` 現在の選択部の末尾に newText を追加し、古いテキストを削除しない • `{"prepend": true}` 現在の選択部の先頭に newText を追加、古いテキストを削除しない • `{"macro": true}` newText を通常のマクロテキストとして扱う（例：`%< %>`プレースホルダを挿入） 例： `editor.replaceSelectedText("world", "append": true )` “world” を現在の選択部の末尾に追加 `editor.replaceSelectedText(function(s)return s.toUpperCase();)` 現在の選択部を大文字へ変換
`editor.undo();`	エディタ上で最後のコマンドを元に戻す
`editor.redo();`	エディタ上で最後のコマンドをやり直す
`editor.cut();`	選択部をクリップボードへ切り取る
`editor.copy();`	選択部をクリップボードへコピーする
`editor.paste();`	クリップボードの内容を貼り付ける
`editor.selectAll();`	全てを選択する
`editor.selectNothing();`	何も選択しない（選択を解除する）
`editor.find();`	「検索パネル」を起動する
`editor.find(QString text, bool highlight, bool regex, bool word=false, bool caseSensitive=false);`	事前に定義された値で「検索パネル」を起動する
`editor.find(QString text, bool highlight, bool regex, bool word, bool caseSensitive, bool fromCursor, bool selection);`	事前に定義された値で「検索パネル」を起動する

表 4.2: 表の続き

`editor.findNext();`	次を検索する
`editor.replacePanel();`	（検索パネルが開いていて何か選択物がある場合）置換する
`editor.gotoLine();`	「指定行へ移動パネル」を起動する
`editor.indentSelection();`	選択部をインデントする
`editor.unindentSelection();`	選択部のインデントを解除する
`editor.commentSelection();`	選択部をコメント化する
`editor.uncommentSelection();`	選択部のコメント化を解除する
`editor.clearPlaceHolders();`	プレースホルダーを削除する
`editor.nextPlaceHolder();`	次のプレースホルダーへ移動する
`editor.previousPlaceHolder()`	前のプレースホルダーへ移動する
`editor.setPlaceHolder(int i, bool selectCursors=true);`	プレースホルダーを設定する
`editor.setFileName(f);`	ファイル名を f に設定する
`editor.write(str)`	現在のカーソル位置に文字列 `str` を挿入する（ミラーカーソルがある場合そのすべてに `str` が挿入される）
`editor.insertText(str)`	現在のカーソル位置に文字列 `str` を挿入する（ミラーカーソルは無視されるので、replaceSelectedText の使用または代わりに書き込むのに好ましい）
`editor.setText(`*`text`*`)`	現在の文書のテキスト全てを *`text`* で置換する
`editor.text()`	完全な文書のテキストを返す
`editor.text(int line)`	行 *`line`* のテキストを返す

表 4.3　文書オブジェクトの一覧

文書オブジェクト	
コマンド	説明
editor.document().lineCount()	行の数を返す
editor.document().visualLineCount()	見かけの行の数を返す（ワードラップした行を数える）
editor.document().cursor(line, [column = 0], [lineTo = -1], [columnTo = length of lineTo])	カーソルオブジェクトを返す。もし lineTo が与えられた場合、カーソルは line:column から lineTo:columnTo までの選択部を持つ。
editor.document().text([removeTrailing = false], [preserveIndent = true])	文書のテキスト全体を返す
editor.document().textLines()	テキスト行全ての配列を返す
editor.document().lineEndingString()	行の末端（\n または\n \r）を含む文字列を返す
editor.document().canUndo()	元に戻す (undo) が可能なら真 (true) を返す
editor.document().canRedo()	やり直す (redo) が可能なら真 (true) を返す
editor.document().expand(lineNr)	行を展開する
editor.document().collapse(lineNr)	行を折りたたむ
editor.document().expandParents(lineNr)	行の全ての親をその行が見えるまで展開する
editor.document().foldBlockAt(bool unFold, lineNr);	lineNr の前の最初のブロックを折りたたむまたは展開する
editor.document().getMasterDocument();	この文章を直接含む開いてある文書を返す
~~editor.document().getTopMasterDocument();~~	**非推奨**：代わりに getRootDocument() を使うこと
editor.document().getRootDocument();	この文章を間接的に含み、自身は他の文書で含まれていない、開いてある文書を返す
editor.document().getMagicComment(name);	存在する場合マジックコメントの内容を返す
editor.document().updateMagicComment(name, value, [create = false]);	マジックコメントを変更する
editor.document().labelItems/refItems/bibItems	全てのラベル／参照または含まれる参考文献ファイルの id を返す
editor.document().getLastEnvName(lineNr)	（その行の終わりでの）現在の環境名を返す

表 4.4　文書管理オブジェクトの一覧

文書管理オブジェクト	
コマンド	説明
documentManager.currentDocument	現在の文書（スクリプトがバックグラウンドモードで起動していない限り、通常は editor.document() と同じである）
documents.masterDocument	マスターファイル
[documentManager.]documents	開いている文書全ての配列
documentManager.findDocument(fileName)	あるファイル名 fileName を持つ開いてある文書を返す
documentManager.singleMode()	明示的なマスターファイルがない場合真 (true) を返す
~~documentManager.getMasterDocumentForDoc(document)~~	**非推奨**：代わりに getRootDocumentForDoc(document) を使うこと
documentManager.getRootDocumentForDoc(document)	与えられた文書 (document) を（おそらく間接的に）含んでいる開いてある文書を返す
documentManager.findFileFromBibId(id)	与えられた id の項目を含む bib ファイルのファイル名を返す

表 4.5: カーソルオブジェクトの一覧

カーソルオブジェクト	
コマンド	説明
cursor.atEnd()	カーソルが文書の終端にあるかどうかを返す
cursor.atStart()	カーソルが文書の先頭にあるかどうかを返す
cursor.atBlockEnd()	カーソルがブロックの終端にあるかどうかを返す
cursor.atBlockStart()	カーソルがブロックの先頭にあるかどうかを返す
cursor.atLineEnd()	カーソルが行の終端にあるかどうかを返す
cursor.atLineStart()	カーソルが行の先頭にあるかどうかを返す
cursor.hasSelection()	カーソルに選択部があるかどうかを返す
cursor.lineNumber()	カーソルの行番号を返す
cursor.columnNumber()	カーソルの列を返す
cursor.anchorLineNumber()	アンカーの行番号を返す
cursor.anchorColumnNumber()	アンカーの列を返す
cursor.shift(int offset)	カーソル位置（テキスト列）を列（文字）の数でずらす
cursor.setPosition(int pos, MoveMode m = MoveAnchor)	文書の先頭から数えて pos 文字後にカーソル位置を設定する（非常に遅い）

表 4.5: 表の続き

`cursor.movePosition(int offset, MoveOperation op = NextCharacter, MoveMode m = MoveAnchor);`	カーソルを *`offset`* 回移動する。MoveOperations は次のものである： • `cursorEnums.NoMove` • `cursorEnums.Up` • `cursorEnums.Down` • `cursorEnums.Left` • texttt`cursorEnums.PreviousCharacter` = Left • `cursorEnums.Right` • `cursorEnums.NextCharacter = Right` • `cursorEnums.Start` • `cursorEnums.StartOfLine` • `cursorEnums.StartOfBlock = StartOfLine` • `cursorEnums.StartOfWord` • `cursorEnums.StartOfWordOrCommand` • `cursorEnums.PreviousBlock` • `cursorEnums.PreviousLine = PreviousBlock` • `cursorEnums.PreviousWord` • `cursorEnums.WordLeft` • `cursorEnums.WordRight` • `cursorEnums.End` • `cursorEnums.EndOfLine` • `cursorEnums.EndOfBlock = EndOfLine` • `cursorEnums.EndOfWord` • `cursorEnums.EndOfWordOrCommand` • `cursorEnums.NextWord` • `cursorEnums.NextBlock` • `cursorEnums.NextLine = NextBlock` MoveMode に対するオプションは次のものである： • `cursorEnums.MoveAnchor` • `cursorEnums.KeepAnchor` • `cursorEnums.ThroughWrap`
`cursor.moveTo(int line, int column);`	カーソルを行 *`line`* と列 *`column`* へ移動する
`cursor.eraseLine();`	現在の行を削除する
`cursor.insertLine(bool keepAnchor = false);`	空行を挿入する
`cursor.insertText(text, bool keepAnchor = false)`	テキスト *`text`* をカーソル位置に挿入する（この関数はインデントとミラーを無視する。`editor.write` と `editor.insertText` を見ること。）
`cursor.selectedText()`	選択したテキストを返す
`cursor.clearSelection();`	選択を削除する
`cursor.removeSelectedText();`	選択したテキストを削除する

表 4.5: 表の続き

`cursor.replaceSelectedText(text);`	選択したテキストをテキスト *`text`* で置換する
`cursor.deleteChar();`	カーソルの右側の文字を削除する
`cursor.deletePreviousChar();`	カーソルの左側の文字を削除する
`cursor.beginEditBlock();`	新規編集ブロックを開始する。編集ブロックに埋め込まれたカーソル操作全ては一度で元に戻す／やり直すことになる。
`cursor.endEditBlock();`	編集ブロックを終了する

表 4.6　アプリケーションオブジェクトの一覧

アプリケーションオブジェクト	
コマンド	説明
`app.getVersion()`	現在のバージョン (0xMMmm00)
`app.clipboard`	クリップボードへの読み書きのための属性
`app.getCurrentFileName()`	現在編集しているファイルの名前
`app.getAbsoluteFilePath(rel, ext = "")`	相対的なファイル名を絶対的なものに変換する
`app.load(file)`	ファイル `file` を読み込む
`app.fileOpen/Save/Close/.../editUndo/.../QuickBuild/...`	メニューコマンド全て（つまり texmaker.h ファイルのスロット全て）。設定ダイアログの「メニュー」ページの現在存在するスロット全てのリストを見ることができる。
`app.newManagedMenu([parent menu,] id, caption)`	新規メニューを作成してそれを返す
`app.getManagedMenu(id)`	ある `id` を持つ QMenu を返す
`app.newManagedAction(menu, id, caption)`	新規アクションを作成してそれを返す • `menu`: 親メニュー • `id`: 新規アクションの ID（最後の唯一の ID は**メニュー** *id* ／**アクション** *id* となる） • `caption`: 表示されるテキスト `action.triggered.connect(function(){...});` を用いて、関数を返りアクションへつなげることができる（詳細は qt signal/slot 文書を見ること）。
`app.getManagedAction([id])`	ある `id` を持つ QAction を返す（全ての id は通常一つのメニューで main/menu1/menu2/.../menuN/action という形を持つ。例：“main/edit/undo”。texmaker.cpp を見ること。）
`app.createUI(file, [parent])`	ある ui ファイルを読み込み、それから QWidget*を作成する
`app.createUIFromString(string, [parent])`	文字列 `string` で記述された QWidget*を作成する
`app.slowOperationStarted()/slowOperationEnded()`	遅い操作の開始／終了に関して一時的に無限ループ検出を無効化するよう TXS に通知する
`app.simulateKeyPress(shortcut)`	与えられたショートカットに対してキー入力イベントを発生させる（例：`app.simulateKeyPress("Shift+Up")`）。**注**：これは主にショートカットと移動の為に用意されたものである。現在キー入力イベントの全ての機能をサポートしているわけではない。特にテキストは全く入力できない。

表 4.7　UniversalInputDialog クラスの一覧

UniversalInputDialog クラス	
コマンド	説明
`new UniversalInputDialog()`	新しいダイアログを作成する
`dialog.add(defaultValue, [description, [id]])`	与えられた規定値 `defaultValue` と省略可能な説明 `description` と `id` を持つ新しい変数をダイアログに追加する。そして対応する Qt コンポーネントを返す。文字列既定値は QLineEdit に、数字は QSpinBox に、配列は QComboBox になる。
`dialog.get(nr/id)`	`nr` 番目に追加された変数またはある `id` を持つ変数の現在値を返す
`dialog.getAll()`	全ての変数の値を組み合わされた数字／連想配列として返す。i 番目の変数を得るために `returnvalue[i]` を、ある id を持つ変数を得るために `returnvalue.id` を用いることができる。
`dialog.exec()`	ダイアログを表示する。ユーザーがダイアログを承諾した場合 1 を、ダイアログが取り消された場合 0 を返す。
`dialog.show()`	ダイアログを非同期的に表示する
`UniversalInputDialog([[defaultValue_0, description_0, id_0], [defaultValue_1, description_1, id_1], ...])`	短い形：新規ダイアログを作成して、配列の全ての変数を追加し、`exec` をその上で呼び出す

表 4.8　FileChooser オブジェクトの一覧

FileChooser オブジェクト	
コマンド	説明
`fileChooser.exec()`	ダイアログを表示して、再度閉じられるまで待機する
`fileChooser.setDir(dir)`	ダイアログ内のディレクトリを ***dir*** に設定する
`fileChooser.setFilter(filter)`	QT-format を用いてファイルフィルターを ***filter*** に設定する。上を参照すること。
`fileChooser.fileName()`	（exec の後）選択したファイル名を返す

いくつかの例：

- 現在のファイル名をクリップボードにコピーする：

```
1 %SCRIPT
2 app.clipboard = editor.fileName();
```

- エディタテキストの実行：

```
1 %SCRIPT
2 eval(editor.text());
```

- オブジェクトの全ての属性を表示する：

```
1 %SCRIPT
```

```
2 obj = editor;                                          //表示するオブジェクト（例：現在のエディタ）
3
4 app.fileNew();                                         //新しい文書を作成
5 newEditor = documentManager.currentDocument.editorView.editor;   //新しく作成された文書にアクセス
6 for (var prop in obj)
7   newEditor.write(prop+"\n");                          //属性を出力
```

- 編集メニューにアクションを追加：

```
1 %SCRIPT
2 var menu = app.getManagedMenu("main/edit");                          //編集メニューを入手
3 var act = app.newManagedAction(menu, "script", "scripttest");        //アクションを追加
4 act.triggered.connect(function(){alert("called");});                 //簡単なハンドラを登録
5 registerAsBackgroundScript("test");                                  //ハンドラを有効化し続ける
```

- 非同期ダイアログ：

```
1 %SCRIPT
2 var ui = createUI("␣...␣path␣to␣your␣ui␣file␣...");   //ダイアログを読み込む
3 ui.accepted.connect(function(){alert("x");})          //ダイアログを閉じることに反応
4 registerAsBackgroundScript("abc");                    //関数の有効化を維持
5 ui.show();                                            //ダイアログの表示
```

ダイアログは Qt Designer で作られる ui ファイルに記述されている。

- 計算機：

```
1 %SCRIPT
2 currentLine=editor.text(cursor.lineNumber());
3 from=currentLine.lastIndexOf("%")+1;
4 to=currentLine.lastIndexOf("=");
5 if (from>=0 && to > from) {
6   toEvaluate = currentLine.substring(from, to);
7   with (Math) { value = eval(toEvaluate);}
8   cursor.eraseLine();
9   cursor.insertText(currentLine.substring(0, from)+toEvaluate+"="+value);
10  cursor.insertLine();
11  cursor.movePosition(1,cursorEnums.Left );
12 }
```

これは % と=の間のもの全てを評価して、=の後ろに結果を書き込む。tex ファイルに %5+3=を書くと計算機のように使用出来る。

- (tikz) 座標の組の移動：

```
1 %SCRIPT
2 var doit = function(){
```

```
3   var mytext=cursor.selectedText();
4   var regExNumberPre = "␣*[0-9]+([.][0-9]*)?␣*";
5   var regExDigit = /[0-9]/;
6   var regExSpace = / /g;
7   var regExPairPre = "␣*(-?"+regExNumberPre+")";
8   var regExPair = new RegExp("()[(]"+regExPairPre+","+regExPairPre+"[)]"); ;
9
10  //最初の座標の組を読み込む
11  var regExFirstPairPre = regExPairPre + "␣*([+-]"+regExNumberPre+")?";
12  var regExFirstPair = new RegExp("()[(]"+regExFirstPairPre+","+
        regExFirstPairPre+"[)]");
13
14  //オフセットを抽出（-x - yを許可するため、最初の数字から正規表現検索を開始）
15  var matches = regExFirstPair.exec(mytext);
16  if (matches == null) throw "missing";
17  //一致物を投げる
18  var offsetXPre = matches[4];
19  var offsetYPre = matches[8];
20  if (offsetXPre == "" && offsetYPre == "") throw "abc";
21  var offsetX = offsetXPre == ""?0.0:offsetXPre.replace(regExSpace, "")*1.0;
22  var offsetY = offsetYPre == ""?0.0:offsetYPre.replace(regExSpace, "")*1.0;
23
24  //最初の組を移動
25  var matchpos = mytext.search(regExFirstPair);
26  editor.write(mytext.slice(0,matchpos));
27  editor.write("("+(matches[2].replace(regExSpace, "")*1.0+offsetX));
28  editor.write(",␣"+(matches[6].replace(regExSpace, "")*1.0+offsetY)+")");
29
30  //他の組を移動
31  var remaining = mytext.slice(matchpos+matches[0].length);
32  while (remaining != ""){
33    matches = regExPair.exec(remaining);
34    if (matches == null) break;
35    matchpos = remaining.search(regExPair);
36    editor.write(remaining.slice(0,matchpos));
37    remaining = remaining.slice(matchpos+matches[0].length);
38    editor.write("(" + ((matches[2].replace(regExSpace, "")*1.0)+offsetX) +
          ",␣"+ ((matches[4].replace(regExSpace, "")*1.0)+offsetY) + ")");
39  }
40 editor.write(remaining);
41 }
42 doit();
```

これは、全ての選択した座標の組に最初の組のオフセットを加えて与えられた方向に全ての組を移動させる。
例：(1 + 1, 2 − 1.5) (3, 4) がある場合、(2, 0.5) (4, 2.5) に変更される。

4.5.4　トリガー

正規表現

トリガーの最も単純な形式では、トリガーは単なるテキストであり、マクロで置換される。例：trigger=“eg” macro=“example given” の場合、“the leg” の “eg” は “g” を入力すると “example given” で置換される。

トリガーが正規表現であると、手の込んだトリガーを作成することができる。TXS は後方参照検索を使用している：“`(?<=\s)%`” は、前の文字が空白なら “%” を置換するのに使用される。正規表現についてのさらなるヘルプはインターネット上で見ることができる。

グローバル変数 `triggerMatches` を通してスクリプト中で一致した表現にアクセスすることができる。`triggerMatches` は配列である。その 0 番目の要素は正規表現全体に一致したものである。続く要素は（グループが定義されていれば）グループに一致したものである。

例：

```
1 Trigger: #([a-z])
2 Typed text: #a
3
4 triggerMatches[0] == '#a'
5 triggerMatches[1] == 'a'
```

スコープの制限

マクロが有効なスコープには、パターン (`?[scope-type]:...`) の表現を前に付けることができる。

表 4.9　スコープ制限表現の一覧

スコープ制限表現	意味
`(?language:...)`	文書の強調表示が与えられた言語に一致している場合にのみ、マクロは有効である。例：`(?language:latex)`
`(?highlighted-as:...)`	マクロを特定の強調表示される環境に制限する。可能な値は構文強調表示設定ページのリスト（英語）に対応する。例：`(?highlighted-as:numbers,math-delimiter,math-keyword)`
`(?not-highlighted-as:...)`	`(?highlighted-as:...)` に似ているが、与えられた環境でマクロを無効化する。

`(?language:...)` と `(?highlighted-as:...)` の表現を組み合わせることができる。しかし、`(?highlighted-as:...)` と `(?not-highlighted-as:...)` の組み合わせは論理的に意味を成さず、未定義な振る舞いとなる。

トリガー自体の正規表現が必要なことに注意すること。次は充分で複雑な例である：
`(?language:latex)(?highlighted-as:comment,commentTodo)FIXME`。

このトリガーは “FIXME” を入力することに対応しているが、LaTeX 文書のコメントと todo ノート内でのみ有効である。

イベントトリガー

さらに、対応するイベントが生じた時にスクリプトを実行するために次の特別なトリガー表現（括弧なし）を使うことができる：

表 4.10　特別なトリガー表現の一覧

特別なトリガー	スクリプトが実行されるイベント
`?txs-start`	TeXstudio 開始時
`?new-file`	新規ファイル作成時
`?new-from-template`	テンプレートからの新規ファイル作成時
`?load-file`	ファイル読み込み時
`?load-this-file`	マクロを含むファイルの読み込み時（スクリプトがマジックコメントとして定義されている場合にのみ意味をなす）
`?save-file`	ファイル保存時
`?close-file`	ファイルを閉じた時
`?master-changed`	文書がマスターファイルとして定義解除／定義された時
`?after-typeset`	LaTeX 類似コマンド終了時
`?after-command-run`	コマンド実行終了時（例：latex を 2 回呼び出してビューワーを開くコンパイルコマンドの場合、このイベントが生じるのは 1 回だが `after-typeset` では 2 回である。）

これら特殊トリガーは “|” 記号で複数個組み合わせることができる。

4.6　Pstricks のサポート

主な pstricks コマンドは「構造ビュー」の「Pstricks」パネルで挿入できる。

4.7　Metapost のサポート

Metapost キーワードは「構造ビュー」の「Metapost」パネルで挿入できる。また、「ツール／コマンド」メニューを通じて「mpost」などのコマンドを起動させることができる。

4.8　「HTML へ変換」コマンド

このコマンド（「ツール」メニューにある）は、LaTeX ソースファイルから html ページ各々につき一画像の html ページのセットを生成する。プレゼンテーションスライドの各ページは、LaTeX を実行して得られる postscript ページの一つに対応する。

またこのコマンドでは、LaTeX で得られる目次に対応するページも生成される。このページの各項目は対応する html ページへのリンクを含んでいる。

tex ファイルに`\ttwplink{}{}`コマンドを用いることで、生成される html ページにリンクを作成することができる。

概要：

`\ttwplink{http://www.mylink.com}{my text}` （外部リンク）

`\ttwplink{page3.html}{my text}` （内部リンク）

`\ttwplink{name_of_a_label}{my text}` （内部リンク）

警告：hyperref パッケージ（あるいは他のいくつかのパッケージ）とともにこのコマンドを使用することはできない。このコマンドは「HTML へ変換」ツールとともにしか使用できない。

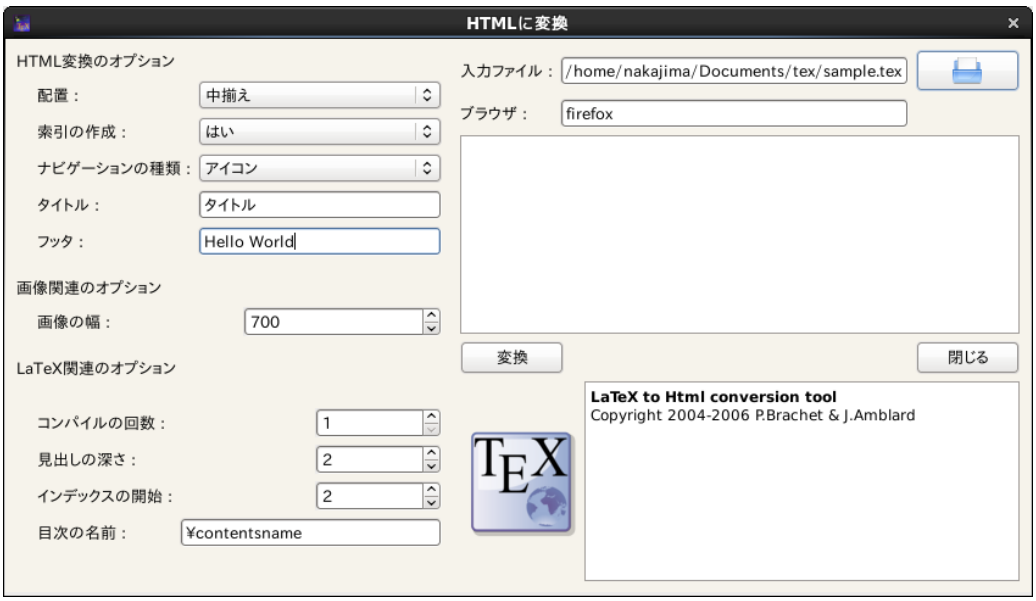

図 4.3　「HTML へ変換」ダイアログ

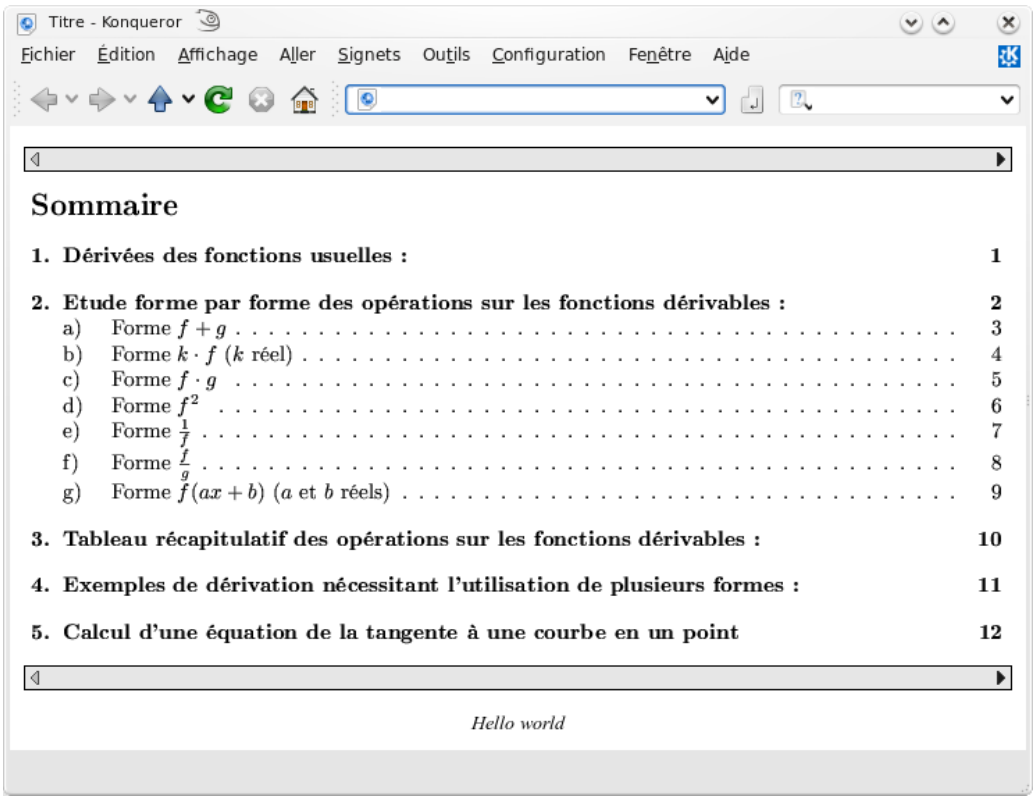

図 4.4　変換された html

4.9　TeXstudio での「順方向／逆方向検索」

4.9.1　統合された PDF ビューワー

TeXstudio では順方向・逆方向検索を提供する統合された PDF ビューワーが提供されている。pdflatex コマンド（または類似コマンド）で synctex が有効化されていることを確認すること（オプション“`-synctex=1`”を追加する必要がある）。もし正しく設定されていない場合、TeXstudio はコマンド自体を修正するかどうかを尋ねる。

順方向検索は PDF ビューワーが開いた時にはいつも自動的に行われる。TeXstudio はカーソルが現在位置している場所へ（PDF 上で）移動する。さらに、テキストエディタ上の単語を CTRL+ 左クリックすることで PDF へ移動したり、コンテキストメニューを用いて「PDF へ移動」を選択して移動することもできる。

逆方向検索は、PDF 上で CTRL+ 左マウスボタンクリックするか、コンテキストメニュー（右マウスボタンクリック）で「ソースへ移動」を選択することで有効になる。さらに、PDF ビューワーの設定で「カーソルに続いてスクロールする」を有効化することができる。これで、PDF ビューワー上の位置をエディタ上のカーソル位置に同期させることができる。同様に、「スクロールに続いてカーソルを移動する」を有効化することでエディタ上の位置を PDF ビューワー上の位置に同期させることができる。

4.9.2　外部ビューワーに対する一般的な設定

いくつかの (dvi) ビューワーでは、(La)TeX ソースファイル上のある行番号に対応する DVI ファイル上での位置へ移動（あるいは視覚的に強調表示）することができる。

この順方向検索を有効にするには、ユーザーツール（「オプション／ TeXstudio の設定」 -> 「ビルド」）のコマンド行または設定ダイアログのビューワーコマンド行（「オプション／ TeXstudio の設定」 -> 「コマンド」）で、対応するビューワーのコマンドを入力すれば良い。ビューワーが起動するときに、@プレースホルダは現在の行番号で置換され、**?c:ame** は現在のファイルの完全な絶対パスでのファイル名で置換される。

Windows では、次の形のコマンドを挿入することで DDE コマンドを実行できる：`dde:///service/control/[commands...]` あるいは必要な場合次の形のコマンドでプログラムを起動することもできる（TeXstudio 1.9.9 から）：`dde:///programpath:service/control/[commands...]`

下にいくつかの一般的なビューワーに対するコマンドのリストを載せている。当然、コマンドを使用したい場合 *(your program path)* をコンピュータ上のそのプログラムのパスで置換する必要がある。

4.9.3　Sumatra

Sumatra を TeXstudio から起動して逆方向検索を設定：`"`*`(your sumatra path)`*`" -reuse-instance -inverse-search "\"`*`(your TeXstudio path)`*`\" \"%%f\" -line %%l" "?am.pdf"`

起動している Sumatra である行へ移動（Windows 限定）：
`dde:///SUMATRA/control/[ForwardSearch("?am.pdf","?c:am.tex",@,0,0,1)]`

起動していなければ Sumatra を実行して、ある行へ移動（Windows 限定）：`dde:///`*`(your sumatra path)`*`:SUMATRA/control/[ForwardSearch("?am.pdf","?c:am.tex",@,0,0,1)]`

Sumatra から TeXstudio を実行：`"`*`(your TeXstudio path)`*`" "%f" -line %l`

(your Sumatra path) の考えられる値は `C:/Program Files/SumatraPDF/SumatraPDF.exe` である。

4.9.4　Foxit Reader

Foxit Reader を TeXstudio から起動：`"`*`(your Reader path)`*`" "?am.pdf"`

4.9.5 Acrobat Reader

TeXstudio から Acrobat Reader を起動：`"(your Reader path)" "?am.pdf"`

移動と閉じることは DDE コマンドを通じて行う。Adobe 製品のバージョン 10 から、DDE サービス名に製品とバージョン番号に対する文字列が含まれる。

表 4.11　製品とサービス名の対応

製品	サービス名
Adobe Reader 9	acroview
Adobe Acrobat 9	acroview
Adobe Reader 10	acroviewR10
Adobe Acrobat 10	acroviewA10
Adobe Reader 11	acroviewR11
Adobe Acrobat 11	acroviewA11

Adobe Reader 11 に対する例が次である：

実行している Acrobat Reader 上のある位置へ移動（Windows 限定）：

```
dde:///acroviewR11/control/[DocOpen("?am.pdf")][FileOpen("?am.pdf")][DocGotoNameDest("?am.pdf",
"jump-position")]
```

jump-position は *hyperref* パッケージで定義できる

実行している Acrobat Reader 上の文書を閉じる（Windows 限定）：

```
dde:///acroviewR11/control/[DocOpen("?am.pdf")][FileOpen("?am.pdf")][DocClose("?am.pdf")]
```

4.9.6 Yap (Yet Another Previewer)

TeXstudio から Yap を起動：`"(your Yap path)" -1 -s @?c:m.tex %.dvi`

Yap から TeXstudio を起動：`"(your TeXstudio path)" "%f" -line %l`

(your Yap path) の考えられる値は `C:/Program Files/MiKTeX 2.7/miktex/bin/yap.exe` である。

4.9.7 xdvi

TeXstudio から xdvi を起動：`xdvi %.dvi -sourceposition @:?c:m.tex`

TeXstudio から xdvi を起動して逆方向検索を有効化：`xdvi -editor "texstudio %f -line" %.dvi -sourceposition @:%.tex`

4.9.8 kdvi

TeXstudio から kdvi を起動：`kdvi "file:%.dvi#src:@ ?c:m.tex"`

4.9.9 Okular

TeXstudio から okular を起動：`okular --unique %.dvi#src:@?c:m.tex`

Okular から TeXstudio を起動： `texstudio %f -line %l`

4.9.10　Skim

TeXstudio から Skim を起動：`(your Skim path)/Contents/SharedSupport/displayline @ ?am.pdf ?c:ame`

Skim から TeXstudio を起動：コマンド：`/applications/texstudio.app/contents/macos/texstudio` 引数：`"%file" -line %line`

(your Skim path) の考えられる値は`/Applications/Skim.app` である。

4.10　高度なヘッダの使用法

いわゆる「マジックコメント」はエディタのオプションを文書ごとのレベルで適応させる方法である。概念は元々 TeXshop に導入されて、以来数多くのエディタに取り入れられている。TeXstudio では次のマジックコメントがサポートされている：

- % !TeX spellcheck = de_DE
  文書のスペルチェックに用いられる言語を定義する。これは全体的なスペルチェックの設定を上書きする。しかし、適切な辞書がインストールされている必要がある。
- % !TeX encoding = utf8
  文書の文字エンコーディングを定義する。
- % !TeX root = filename
  このファイルに対するルートドキュメント（つまり、ビルドする際に LaTeX コンパイラに渡されるファイル）を定義する。この設定は TeXstudio のルートドキュメントの自動検出を上書きする。続いて、**明示的なルートドキュメント**が「オプション -> ルートドキュメント」で設定されている場合それも上書きされる。
- % !TeX program = pdflatex
  文書に対して利用するコンパイラを定義する。正確には、「ビルド＆表示」と「コンパイル」の動作で使用する既定のコンパイラ (コマンド `txs:///compile`) を上書きする。
- % !TeX TXS−program:bibliography = txs:///biber
  これは TeXstudio 特有の設定である。左側で指定されたビルドシステムコマンドを右側のコマンドで上書きする。例では、TXS に一般的な「参考文献コマンド (`txs:///bibliography`)」に対して biber コマンド (`txs:///biber`) を使用することを伝えている。ビルドシステムの高度な設定も参照のこと。
- ```
  % !TeX TXS-SCRIPT = foobar
  % //Trigger = ?load-this-file
  % app.load("/tmp/test/test.tex");
  % app.load("/tmp/test/a.tex");
  % TXS-SCRIPT-END
  ```
  このコードは一時的な javascript マクロを定義している。このマクロはファイルが読み込まれた時に実行され、順番に/tmp/test にある 2 つのファイルを読み込む。
  `TXS-SCRIPT` を通じて定義されたマクロは文書のファイル全て（例：取り込まれるファイル）で有効である。これらを手動で起動することはできない。これらはトリガー（正規表現または特定のトリガー、トリガーについての節を見よ）を通じて起動する。マクロはファイルを開く際に一度だけ読み込まれる。編集セッションの間の変更はファイルを開き直した時だけ効果がある。
- % !BIB program = biber
  ```

特殊コマンド % !BIB program は TeXShop と TeXWorks との互換性のために用いられる（% !BIB TS-program もである）。これは % !TeX TXS-program:bibliography = txs:///biber と同じである。

4.11　TeXstudio コマンドの概要

texstudio file [-master] [-line xx[:cc]] [-insert-cite citation] [-start-always] [-pdf-viewer-only] [-page yy]

表 4.12　コマンドオプションの一覧

-master	文書を明示的なルートドキュメント（以前はマスターファイルと呼ばれていた）として定義する。
-line xx[:cc]	TeXstudio は文書読み込み後に xx 行へ移動する。さらにコロンで区切って目標の列を追加できる（例："-line 2:5" は 2 行目の 5 列目に移動する）。
-insert-cite citation	カーソル位置へ挿入する bibtex キーを TeXstudio へ通知する。これは外部参考文献管理アプリケーションに対するインターフェースとして、引用を TeXstudio に通知するために用いられる。\mycite {key}のようなコマンド（カスタマイズされていても良い）または、キーだけを渡してもよい。後者の場合、\cite {key}に展開される。また、カンマ区切りのキーリストもサポートされている。カーソルがすでに引用マクロ内にある場合、TeXstudio はそれを認識する。その場合キーのみが適切な位置に挿入され、そうでない場合完全な引用コマンドが挿入される。
-start-always	TeXstudio の別のインスタンスがすでに起動していても、TeXstudio を起動する。これで複数のインスタンスを利用できる。
-pdf-viewer-only	TeXstudio を、エディタなしの単体の PDF ビューワーとして開く。
-page	オプションであり、PDF ビューワーとして用いた場合に TeXstudio で特定のページを表示する。

次の追加のオプションは TeXstudio のデバッグ版でのみ利用できる：

表 4.13　デバッグ版でのみ有効なコマンドオプションの一覧

-disable-tests	あらゆるテストを実行しない。
-execute-tests	最も一般的なテストを強制的に実行する。
-execute-all-tests	全てのテストを強制的に実行する。

注：実行ファイルに変更がある場合（つまり、TXS が最後の実行以来コンパイルされてきた）、最も一般的なテストは自動的に実行される。さらに全てのテストは週一で実行される。

4.12　キーボードショートカット

キーボードショートカットは「オプション -> TeXstudio の設定 -> キーボードショートカット」で変更できる。

次のリストは既定のキーボードショートカットの大まかな概要である。オペレーティングシステム (OS) によって、OS 特有のショートカット規定に適合するためばらつきがあるかもしれない。

- 「ファイル」メニュー：
  - 新規作成：Ctrl+N
  - 開く：Ctrl+O
  - 保存：Ctrl+S
  - 名前をつけて保存：Ctrl+Alt+S

  - 全て保存：Ctrl+Shift+Alt+S
  - 閉じる：Ctrl+W
  - ソースコードの印刷：Ctrl+P
  - 終了：Ctrl+Q
- 「編集」メニュー：
  - 元に戻す：Ctrl+Z
  - やり直す：Ctrl+Y
  - コピー：Ctrl+C
  - 切り取り：Ctrl+X
  - 貼り付け：Ctrl+V
  - 全て選択：Ctrl+A
  - 行を削除：Ctrl+K
  - 行末まで削除：Alt+K
  - 検索：Ctrl+F
  - 次を検索／検索を続ける：F3 / Ctrl+M
  - 置換：Ctrl+R
  - 特定の行番号へ移動：Ctrl+G
  - 前の変更へ移動：Ctrl+H
  - 次の変更へ移動：Ctrl+Shift+H
  - ブックマーク 0..9 へ移動：Ctrl+0..9
  - ブックマーク 0..9 の切り替え：Ctrl+Shift+0..9
  - 名前なしブックマークの設定：Ctrl+Shift+B
  - 次のマークへ移動：Ctrl+Down
  - 前のマークへ移動：Ctrl+Up
  - 前へ移動：Alt+Left
  - 後ろへ移動：Alt+Right
- 「Idefix」メニュー：
  - 単語／コマンド／環境の削除：Alt+Del
  - LaTeX として貼り付け：Ctrl+Shift+V
  - プレビューの表示：Alt+P
  - コメントアウト：Ctrl+T
  - コメントアウトの解除：Ctrl+U
  - 次の LaTeX エラーへ移動：Ctrl+Shift+Down
  - 前の LaTeX エラーへ移動：Ctrl+Shift+Up
  - 次の LaTeX の良くないボックスへ移動：Shift+Alt+Down
  - 前の LaTeX の良くないボックスへ移動：Shift+Alt+Up
  - 定義へ移動：Ctrl+Alt+F
  - 標準的な補完：Ctrl+Space
  - `\begin{`の補完：Ctrl+Alt+Space
  - 通常のテキストの補完：Alt+Shift+Space
  - 最後に開いていた環境を閉じる：Alt+Return
  - プレースホルダーを削除：Ctrl+Shift+K
- 「ツール」メニュー：
  - ビルド＆表示：F1
  - コンパイル：F6
  - PDF を表示：F7

  - 参考文献 (Bibtex)：F8
  - 用語集：F9
  - （カーソル位置から）スペルチェック：Ctrl+:
  - 類語辞典を開く：Ctrl+Shift+F8
- 「LaTeX」メニュー：
  - 箇条書きの項目 (`\item`)：Ctrl+Shift+I
  - イタリック体：Ctrl+I
  - スラント体：Ctrl+Shift+S
  - ボールド体：Ctrl+B
  - タイプライター体：Ctrl+Shift+T
  - スモールキャップス体：Ctrl+Shift+C
  - 強調：Ctrl+Shift+E
  - 強制改行：Ctrl+Return
  - `begin{environment}`：Ctrl+E
  - 次のラベルに参照 (`\ref`) を挿入：Ctrl+Alt+R
- 「数式」メニュー：
  - インライン数式：Ctrl+Shift+M
  - ディスプレイ数式：Alt+Shift+M
  - 番号付き数式：Ctrl+Shift+N
  - 下付き添字：Ctrl+Shift+D
  - 上付き添字：Ctrl+Shift+U
  - `\frac`：Alt+Shift+F
  - `\dfrac`：Ctrl+Shift+F
  - `\sqrt`：Ctrl+Shift+Q
  - `\left`：Ctrl+Shift+L
  - `\right`：Ctrl+Shift+R
- 「表示」メニュー：
  - 前の文書：Ctrl+PgUp
  - 次の文書：Ctrl+PgDown
  - エディタにフォーカスを移動：Ctrl+Alt+Left
  - ビューワーにフォーカスを移動：Ctrl+Alt+Right
  - 閉じる：Esc
  - 全画面表示モード：F11

4.13　cwl 形式の解説

cwl は completion word list（補完単語リスト）の略であり、もともとは補完の際に並べられるコマンドを定義するため Kile で使用されていたファイル形式である。TeXstudio では cwl の拡張書式を用いて、追加の意味情報を含ませたりカーソルやプレースホルダの配置を行えるようにしている。TeXstudio では次の目的のために cwl を利用している：

- 自動補完の事前設定
- 現在の文書における有効なコマンドの認識（`\usepackage` 文に基づく）
- エディタ上の追加コンテキストを提供する意味情報; 例：`\ref` のようなコマンドは参照されるラベルが存在するかどうかを確認する。

4.13.1　cwl ファイルの書式

cwl ファイルの各行ではコマンドが定義されている。コメント行も可能であり、#で始めればよい。コマンド構文は次の形で表される。

```
<command>[#classification]
```

分類 classification がない場合、そのコマンドは LaTeX 文書のあらゆる位置で有効であるとみなされる。文字#は、次のような特別な意味があるので、command 内で使用することはできない：

- `#include:<packagename>`（行の最初で）：packagename.cwl も読み込む。これはあるパッケージが他のパッケージに依存していることを示すのに使用する。
- `#repl:<search> <replacement>`（行の最初で）：文字の置換を定義する（例：ドイツ語に対して"a -> ä)。スペルチェック (babel) における文字置換に対してのみ用いられる。
- `#keyvals:<command>`（行の最初で）：command に対する keyvals の定義を開始する（ソースコードの graphicx.cwl を参照)。キーに対する可能な値を指定するため、#の後ろにそれらを追加する（例：`mode=#text,math`)。
  単一のキー／値の代わりに、完全な特殊リストを与えることができる（例：`color=#%color`、tikz.cwl も参照のこと)。
  command は 2 つの部分からなる（例：\documentclass/thesis はコマンド\documentclass が引数として thesis を使用する場合にのみ有効である)。
  #c が追加されている場合、keyvals は補完にのみ用いられて構文チェックには用いられない。
- `#endkeyvals`（行の最初で）：keyvals の定義の終わりを示す（ソースコードの graphicx.cwl を参照)。
- `#ifOption:<option>`（行の最初で）：以降のブロックは、`<option>`が usepackage コマンドで使われた場合にのみ読み込まれる（例：`\usepackage[svgnames]{color}` -> option=svgnames）
- `#endif`（行の最初で）： 条件ブロックの終わりを示す
- `#`（`#include,#repl,#keyvals,#endkeyvals` を除いた行の最初で）：この行はコメントであり、無視される。
- `#`（行中で）： コマンド command と分類 classification を分ける。

cwl ファイルは UTF-8 としてエンコードされていなければならない。

4.13.2　コマンド書式

最も単純な形式では、文書で見られるようにコマンドは単なる有効な LaTeX 表記である（例：`\section{title}`)。既定では全てのオプションはプレースホルダとして扱われる。かわりに、`%|`でカーソルの停止位置を定義（例：`\left(%|\right)`）するか、`%< %>`を用いてオプションの一部のみをプレースホルダとしてしるし付け（例：`\includegraphics[scale=%<1%>]{file}`）してもよい。改行は `%n` でコマンドに含めることができる。

引数の名前

引数の名前は、補完ウィンドウに表示され、補完後はエディタ上にプレースホルダとして表示される。一般的に、引数の名前は好きなように自由に名付けてよい。我々は意味ある名前を与えることを勧める（例：`\parbox[arg1]{arg2}{arg3}`ではなく`\parbox[position]{width}{text}`)。

特別な意味を持つ引数名がいくつかある：

- `text` あるいは `%text` で終わる：スペルチェッカーはこの引数内で作動する（既定では引数はスペルチェックされない)。
- `title` あるいは `short title`：スペルチェッカーはこの引数内で作動する（既定では引数はスペルチェックさ

れない)。さらに、引数は（section 内のように）ボールド体で設定される。

- `bibid` と `cite key`：コマンド内で使用される場合、“C” と分類される。下の分類子解説を見よ。
- `cmd` と `command` あるいは `%cmd` で終わる：コマンドに対する定義（例：`\newcommand{cmd}`）。この例の “cmd” は引数は無く、何も機能性を伝えないとみなされる。
- `def` と `definition`：コマンドの実際の定義（例：`\newcommand{cmd}{definition}`）。この例の “definition” は構文チェックに対しては無視される。
- `args`：コマンドに対する引数の数（例：`\newcommand{cmd}[args]{definition}`）。
- `package`：パッケージ名（例：`\usepackage{package}`）
- `title` と `short title`：section 名（例：`\section{title}`）
- `color`：色名（例：`\textcolor{color}`）
- `width`、`length`、`height` あるいは `%l` で終わる：幅または長さのオプション（例：`\abc{size%l}`）
- `cols` と `preamble`：tabular などでの列定義（例：`\begin{tabular}{cols}`）
- `file`：ファイル名
- `URL`：URL
- `options`：パッケージオプション（例：`\usepackage[options]`）
- `imagefile`：画像のファイル名
- `key`：ラベル／参照のキー
- オプション`#r` 付きの `label` または `%ref` で終わるキー：参照キー
- `labellist`：cleveref で用いられるラベルのリスト
- `bib file` と `bib files`：参考文献ファイル
- `class`：ドキュメントクラス
- `placement` と `position`：環境の位置
- `beamertheme`：beamer のテーマ（例：`\usebeamertheme{beamertheme}`）
- `keys`,`keyvals` そして `%<options%>`：キー／値のリスト

4.13.3　分類の書式

次の分類が TXS では有効である：

表 4.14　cwl 形式の分類の書式の一覧

分類子	意味
*	「全て」タブでのみ補完に用いられる珍しいコマンド。この印は他の分類子とともに用いてもよい。
S	補完の際に全く表示されない。 この印は他の分類子とともに用いてもよい。
m	数式環境でのみ有効
t	tabular 環境（または同様の環境）でのみ有効
T	tabbing 環境でのみ有効
n	テキスト環境（つまり数式環境でない）でのみ有効
r	このコマンドは “`\ref{key}`” のような参照を表すことを示す。
c	このコマンドは “`\cite{key}`” のような引用を表すことを示す。
C	このコマンドは “`\textcquote{bibid}{text}`” のような複雑な引用を表すことを示す。キーは`bibid` として与えられる必要がある。
l	このコマンドは “`\label{key}`” のようなラベルを表すことを示す。
d	このコマンドは “`\newcommand{cmd}{def}`” のような定義コマンドを表すことを示す。
g	このコマンドは “`\includegraphics{file}`” のような画像読み込みコマンドを表すことを示す。
i	このコマンドは “`\include{file}`” のようなファイル読み込みコマンドを表すことを示す。
u	このコマンドは “`\usepackage{package}`” のようなパッケージ使用コマンドを表すことを示す。
b	このコマンドは “`\bibliography{bib}`” のような参考文献を表すことを示す。
U	このコマンドは “`\url{URL}`” のような url コマンドを表すことを示す（ただし URL はチェックされない）
D	このコマンドは todo 項目を表すことを示す（サイドパネルの todo リストに追加される）
B	このコマンドは色を表すことを示す（色補完に対してのみ利用され、構文チェックには用いられない）
s	このコマンドは特別な定義を表すことを示す。定義クラスは “#” の後に与えられる。クラス名は % に続ける必要がある（例：%color）。以下の例も参照のこと。
V	このコマンドは verbatim のような環境 “`\begin{Verbatim}`” を表すことを示す。
/env1,env2,...	環境 env1、env2... 内でのみ有効
`\env`	環境の別名であることを表し、“env” 環境のようにその環境が扱われることを意味する。これは env=math, tabular に対して有用である。

引数の意味を指定する分類子（c や i のような）は常に最初の非オプションパラメータに適用される。これは cwl 書式と TXS の LaTeX パーサーの現在の限界によるものである。例えば、`\ref{label}#r` と `\ref[option]{label}#r` は期待通りに働くが、`\ref{arg}{label}#r` は `arg` を参照として解釈する。我々はこのような場合にはどの分類も指定しないことを勧める。

例：

表 4.15　分類の書式の例

行	解説
`# test`	コメント
`\typein{msg}#*`	「全て」タブの補完にのみ表示される珍しいコマンド
`\sqrt{arg}#m`	数式モードでのみ有効
`\pageref{key}#r`	補完に正しく使用される参照コマンドを表す。
`\vector(xslope,yslope){length}#*/picture`	picture 環境でのみ有効な珍しいコマンド
`\begin{align}#\math`	コマンドの妥当性や構文強調に関しては、“align” 環境が数式環境のように扱われることを表す！
`\definecolor{name}{model}{color-spec}#s#%color`	`name` を特殊リスト `%color` に追加する

4.13.4　cwl ガイドライン

TeXstudio はパッケージから cwl を自動的に作成するが、こうした自動生成された cwl は意味ある引数名やコマンドの分類を含んでいない。従って多くのパッケージに対して手動調整した cwl ファイルが同梱されている。我々はユーザーの新しい cwl ファイルへの寄与を奨励する。これらは次の属性を持つ：

- **package-based:** それぞれの cwl は一つのパッケージに対応している。例外は基本的な (La)TeX コマンドを含む cwl であるが、それらは我々がすでに書いているのでユーザーが悩む必要はない。cwl ファイルは、自動読み込みが機能するようにパッケージと同じ名前をつけるべきである。もし`\usepackage{mypackage}`という記述があれば、TeXstudio は利用可能であれば mypackage.cwl を読み込む。
- **complete:** cwl はパッケージの全てのコマンドを含むべきである。もしエディタで未指定のコマンドを使用すると、構文チェッカーはそれを unknown（未知）としるし付けする。
- **specific:** コマンドは可能であれば分類すべきである。これにより TeXstudio はコマンドに追加のコンテキストを与えることができる（例：数式コマンドが数式環境の外で使用されている場合に警告したり、参照や引用を確認したりする）。
- **priorized:** いくつかのパッケージでは非常に多くのコマンドが指定されているかもしれない。珍しいものを* 分類子でしるし付けして、滅多に使わないコマンドで補完が重くなるのを防ぐ。

4.14　表テンプレートの使用

Texstudio では、表テンプレートに従って既存の LaTeX 形式の表を書式変更できるようにしている。

例えば、TXS で次の表を入力したとする：

```
1 \begin{tabular}{ll}
2 a&b\\
3 c&d\\
4 \end{tabular}
```

カーソルをその表の内部に置き、メニュー「LaTeX/表の操作/テンプレートを用いて表を再構築する」を選択する。

すると表の書式を定義するテンプレートを選択することができる。多数のテンプレートが TXS で予め定義されている：

- fullyframed_firstBold
- fullyframed_longtable
- plain_tabular

- plain_tabularx
- rowcolors_tabular

最初の項目を選択すると、表は次のように書式変更される：

```
\begin{tabular}{|l|l|}
\hline
\textbf{a}&\textbf{b}\\ \hline
c&d\\ \hline
\end{tabular}
```

これらのテンプレートで予め定義された様式にちなんで容易に表を書式変更できるようになる。従って、文書中の表が非常に単純な形式で入力されている場合でもその表の形式を同一のものにすることができる。

4.14.1　表テンプレートの作成

テンプレートはユーザーが定義することもできる。テンプレートは設定ディレクトリ (Linux:`~/.config/texstudio`) に置いて、設定 tabletemplate_*name*.js にちなんで名前をつける必要がある。

メタデータを用いてテンプレートの追加の情報を提供できる。メタデータはソースコードの `metaData` に保存される。コード `var metaData = {`はファイルの最初の行で始めなければならない。現在文字列値のみが利用できる。また、整形に html タグを使用することができる。 例：

```
var metaData = {
"Name"        : "Colored rows",
"Description" : "Formats the table using alternate colors for rows. <br> <code>\
    usepackage[table]{xcolor}</code> is necessary.",
"Author"      : "Jan Sundermeyer",
"Date"        : "4.9.2011",
"Version"     : "1.0"
}
```

テンプレート自体は、表全体を含むいくつかの予め定義された変数を持つ javascript（上記参照）である。新しい表は古いものの置換として配置されるだけであり、その変数から情報を使用する。4 つの変数が与えられている：

- def あらゆる書式なしの単純な表定義（つまり、|l|l|の代わりに ll を使用）
- defSplit　列で分割された表定義 (配列 array=l,l,p{2cm})
- env　“tabular” や “longtable” のような古い表の実際の環境名
- tab　実際の表。これは行のリストであり、各行はセルの内容を文字列として含む列のリストである。

行われることの本質を見るために、“plain_tabular” テンプレートのソースをここに載せておく。

```
function print(str){ //ソースをより見やすくするためこの関数を定義
cursor.insertText(str)
}
function println(str){ //ソースをより見やすくするためこの関数を定義
cursor.insertText(str+"\n")
}
println("\\begin{tabular}{"+defSplit.join("")+"}") //表環境を出力
for(var i=0;i<tab.length;i++){   //表の全ての行に対してループ
    var line=tab[i];   //lineは行row[i]の全ての列のリスト
    for(var j=0;j<line.length;j++){ //行の全ての列に対してループ
```

```
11          print(line[j]) //セルを出力
12          if(j<line.length-1) //最後の列で無い場合
13              print("&") //&を出力
14      }
15      println("\\\\") // "\\"で行を終える。文字列中にバックスラッシュが必要なことに注意。
16  }
17  println("\\end{tabular}") //環境を閉じる
```

例で見たように、表は完全に再構築される必要があり、それから新しい書式が適用可能となる。2 番めの例は幾分手の込んだ表を出力する (fullyframed_firstBold) :

```
1   function print(str){
2   cursor.insertText(str)
3   }
4   function println(str){
5   cursor.insertText(str+"\n")
6   }
7   if(env=="tabularx"){
8     println("\\begin{tabularx}{\\linewidth}{|"+defSplit.join("|")+"|}")
9   }else{
10      println("\\begin{"+env+"}{|"+defSplit.join("|")+"|}")
11  }
12  println("\\hline")
13  for(var i=0;i<tab.length;i++){
14      var line=tab[i];
15      for(var j=0;j<line.length;j++){
16                  var col=line[j];
17                  var mt=col.match(/^\\textbf/);
18                  if(i==0 && !mt)
19                    print("\\textbf{")
20          print(line[j])
21                  if(i==0 && !mt)
22                    print("}")
23          if(j<line.length-1)
24              print("&")
25      }
26      println("\\\\␣\\hline")
27  }
28  println("\\end{"+env+"}")
```

4.15　独自の言語定義の書き方

TeXstudio はエディタ部品として QCodeEdit を利用している。そこでは（プログラミング）言語は QNFA という特別な xml 書式で指定されている。これには強調表示、（一致する）括弧、コードの折りたたみが含まれている。通常の TeXstudio インストールでは.qnfa ファイルは見当たらない。なぜなら含まれる言語のファイルはコンパイルさ

れてバイナリに含まれているからだ。設定ディレクトリ内の languages フォルダに適切な.qnfa ファイルを置くことで、独自の言語を追加したり既定のものを上書きすることができる。ここでの定義は組み込みのものより優先される。

.qnfa ファイルでは言語の構文が指定されている。実際の書式情報は.qxf ファイルで指定されている。default-Formats.qxf で指定された書式を利用してもよいし、.qnfa ファイルと一緒に独自の.qxf ファイルを提供してもよい。

構文書式指定を読み、TeXstudio 同梱の書式を見るべきである。

注：TeXstudio をより柔軟にニーズに合わせるため、我々はエンドユーザーに言語指定を公開する。しかし、それをそのまま受け取るべきである。なぜなら、我々はここでサポートを提供することはできないからだ。それは強力な API であるが、洗練されているわけでも完全な機能を提供しているわけでもない。同梱されている.qnfa ファイルにいくつかの構造物が見られるかもしれないが、それは構文書式指定では文書化されていない。加えて、QNFA の書式に基づく正規表現は LaTeX に望む強調表示全てを定義するには十分ではない。従って、さらなる強調表示機能を“(La)TeX”言語に対してソースコード中で直接補っている（例：\begin と\end の括弧内の強調表示）。このためユーザーはこれを変更したり別の言語へ追加したりすることはできない。

4.15.1　例

次は python コードのある強調表示を指定するちょっとした例である：

python.qnfa

```
<!DOCTYPE QNFA>
<QNFA language="Python" extensions="py" defaultLineMark="">
    <sequence parenthesis="round:open" parenthesisWeight="00">\(</sequence>
    <sequence parenthesis="round:close" parenthesisWeight="00">\)</sequence>

    <!-- highlight def and function name -->
    <sequence id="python/definition" format="python:definition">def$s?$w*</
        sequence>

    <sequence id="python/number" format="python:number">[0-9]+</sequence>

    <list id="python/keyword" format="python:keyword">
        <word>return</word>
        <word>if</word>
        <word>elif</word>
        <word>else</word>
    </list>
</QNFA>
```

python.qxf

```
<!DOCTYPE QXF>
<QXF version="1.0" >
    <!-- full specification -->
    <format id="python:keyword" >
        <bold>false</bold>
        <italic>false</italic>
        <overline>false</overline>
        <underline>false</underline>
```

```
9           <strikeout>false</strikeout>
10          <waveUnderline>false</waveUnderline>
11          <foreground>#B200FF</foreground>
12      </format>
13      <!-- but it is sufficient to specify deviations from default -->
14      <format id="python:number" >
15          <italic>true</italic>
16          <overline>false</overline>
17          <foreground>#007F0E</foreground>
18      </format>
19      <format id="python:definition" >
20          <bold>true</bold>
21      </format>
22  </QXF>
```

結果は次のような強調表示となる：

```
def sqrt(a):
    if a < 0:
        return 0
    else:
        return a ** 0.5
```

図 4.5　Python コードの強調表示例

付録 A

変更点

A.1 Version 2.10.2 での変更点

- \newtheorem を通して定義された環境の認識を修正
- 構文チェックの無効化を修正
- ショートカット Ctrl+C を用いたメッセージパネルからのコピー機能の修正
- 2 つのクラッシュの可能性の修正
- フランス語、ドイツ語、スペイン語の翻訳の更新

A.2 Version 2.10.0 での変更点

- TXS は複数行に渡る引数を解釈できるようになった（しかし対応するコマンドのその複数行引数の後ろの引数全ては無視される）
- 引数解釈がより柔軟になり、引数位置と引数の種類の対応がより緩くなった。
- 補完フィルターはもう少し文脈に即したリストを並べるようになった。引数が長さを要求する場合、「典型的／最も使用される」に対して長さのコマンドのみ提示される。同じ事が keyval 補完に対しても当てはまる。
- ラベルに対して検索と置換が可能になった。
- item が description 環境内に置かれているかどうかによってオプション引数を挿入する
- cwl 形式は少し変更・適応した。 改行文字が %n から %\に変更された。これは、自分自身の cwl ファイルを使用する場合にのみ影響する。
- include のファイル名が補完されるようになった。
- 修正：限定されたスコープ内でのすべての置換はスコープを変更しない
- 色が補完のツールチップを通じてと色引数の上にカーソルを乗せることでプレビューできるようになった

A.3 Version 2.9.4 での変更点

- 修正：Linux と OS X での上書きされたショートカットに関するバグ

A.4 Version 2.9.2 での変更点

- すべての CJK 文字で単語回り込みが可能になった
- latexmk 呼び出しの単純化（YoungFrog のおかげである）
- 修正：import コマンドでのディレクトリ／ファイル引数の間違った解釈
- 修正：隠れた暗示的マスタードキュメントが削除された時にクラッシュする
- 修正：追加の検索パスの設定時に絶対パスでのファイルが正しく統合されない

- 修正：OS X で pkg-config を機能させる
- 修正：singleDocMode で隠れた文書を再度開くとクラッシュする
- 修正：特定の言語（日本語、イタリア語、.....）で TXS を使用すると基本的なショートカット（"Left"のような）が上書きされ得る
- 修正：複数スクリーン使用時の補完ツールチップの位置

A.5　Version 2.9.0 での変更点

- URL に対してリンクを重ねるように変更
- マウスの中ボタンを押すことでエディタタブを閉じるように変更
- より標準的なショートカットに変更（特に OS X に対して）
- `txs:///view-pdf-internal` でオプションファイル名引数をサポート
- 構造ビューのコンテキストメニューに「すべての文書を表示／非表示」を追加
- tabulary サポートの追加
- `"%& -job-name=targetfile"`のため出力ファイル名が変わっている場合に正しいログファイルを開くよう変更（注：PDF ファイル名の変更はまだサポートされていない）
- ユーザー定義のアイコンは今やポータブルである：可能ならアイコンパスは設定ディレクトリかアプリケーションディレクトリに対して相対的に保存される
- 別枠 PDF ビューワーで`--no-focus` 引数をサポート
- 「次のラベルに`\ref` を挿入」に対するラベル名の検出の改良
- プレビューに対するコンテキスト検出の改良（複数行数式、複数文字の区切り記号内のカーソル）
- texdoc 位置に対する検索の改良
- LaTeX3 の警告とエラーに対するサポートを含むログ解析の改良
- 緩やかな行の回り込みでのリサイズの変更：垂直方向のカーソル位置を不変にする
- .tikz ファイルのサポート（.tex ファイルのように扱う）
- 新しいオプションの追加：「詳細なエディタ設定」->「文書の構造パネル」（「コメント内の文書構造要素を表示」、「`\end{document}`以降の文書構造要素を印付けする」、「付録 (appendix) 内の文書構造要素を印付けする」）
- 新しいオプションの追加：マウスホイールによる拡大縮小でログエディタも制御（Paulo Silva によるパッチ）
- 新しいオプションの追加：bib ファイルの文字エンコーディング
- 新しいオプションの追加：いくつかの GUI 要素が拡大縮小可能になった（より良い高解像度ディスプレイサポートのため）
- オプションダイアログ内のいくつかを整理
- 変更：プレースホルダーが解釈される場合 LaTeX テンプレートは最初の行に`"% !TXS template"`が必要で、そうでない場合ファイルは単なる LaTeX として読み込まれる
- 修正：より多くの状況で構造ビューの展開を維持する
- 修正：全体検索に対する大文字子文字の区別
- 修正：拡大した埋め込み PDF ビューワーでのスクロールの同期
- 修正：`% !BIB program = biber` の解釈
- 修正：PDF ビューワーの複数ページグリッドでの「テキスト幅に合わせる」
- 修正：セッション保存時に未保存の文書で誤った項目が作成される
- 修正：忘れられたエディタショートカット
- 修正：Retina ディスプレイでの行キャッシュ
- 修正：（OS X 上） 辞書の既定のパスを app に対して相対的に
- 修正： FreeBSD でのコンパイル（Abilio によるパッチ）

- 修正：ユーザーコマンドアイコンの表示（Paulo Silva によるパッチ）
- 修正：svn 自動 checkin がいくつかの保存操作後に実行されなかった
- 修正：bib ファイルでのリンククリック時に追加の bib パス内も検索
- 修正：いくつかの節 (section) が「文書の終わりを越えている」と正しく印付けされなかった
- 修正：ファイルが削除され暗黙の再読み込みが有効化されている場合にクラッシュする
- 修正：行末検出のバグのせいで"`! TeX encoding`"検出に失敗する
- 修正：hunspell 辞書を探す際に hyph_*.dic ファイルが無視される
- 修正：起動時にセッションと共に開かれたファイルに対するトリガー`?load-file`
- 修正：`% !TeX encoding = ...` マジックコメントが入力された時にステータスバーのエンコーディングを更新
- 修正：QuaZip を 0.7.1 へ更新することによるいくつかのテンプレート zip ファイルを開くことに伴う問題
- 修正：入れ子のシートの synctex ファイルの解析
- 修正：マジックコメントの挿入の「元に戻す」操作時にクラッシュする
- 修正：暗いテーマのデスクトップで記号グリッドを見えるようにした
- 修正：ショートカットの変更の確認時の複数の既存の割り当ての扱い
- 修正：特別な数式終了コメント `%$`が折りたたまれた領域内で使用された場合 `%BEGIN_FOLD` の折りたたみが壊れる
- 修正：中国語の句読点文字が行にある場合の回り込みの間違い
- 上書きモードで補完を無効化（上書き中は補完は正しく機能しないため）
- 修正：前方選択でインプットメソッドの振る舞いが間違っている
- 修正：OS X での texdoc 使用
- 修正：Retina ディスプレイでの 1x1 の連続していない PDF ページの描画
- 修正：いくつかの記号の欠損
- 削除：大文字小文字の区別を無視した補完（複雑性／性能問題のため）

A.6　Version 2.8.8 での変更点

- 基本的な Asymptote 強調表示の追加
- コマンドオプションの構文解析の改良
- 「名前をつけて保存」後 PDF と TeX ソースファイル (.tex) 間の同期が機能しない（再コンパイルが必要）なことを通知するように変更
- 2.8.6 よりも起動スピードが向上
- 修正：/Applications... 以下に TXS がインストールされていない場合、アプリケーション内のリソースを検索
- 修正：前後関係からカンマが示唆される場合にのみ補完を起動してカンマを入力
- 修正： 表の構文解析の列操作でのクラッシュ
- 修正： 選択肢がある場合、tabOrIndent に対してのみタブ置換を実行
- 修正： 追加したショートカット（エディタ）の保存
- 修正： retina ノートパソコンの非 retina スクリーン上の大きすぎるシンボル
- 修正：Shift+Backspace ショートカットが Backspace のように機能する (Win+Linux)
- 修正：カーソルが行頭で改行で終了しているものを貼り付ける場合にインデントを増加
- 修正：選択部の自身での置換時に何も変更しない（選択部自己置換のいくつかの場合のインデント問題の修正）
- 修正：行のキャッシュを使用するかどうかで描画結果が少し異なるかもしれない。マニュアルにいくつかのアップデートがある

A.7　Version 2.8.6 での変更点

- カーソル選択によりタブキーでタブの挿入またはインデントがなされるように変更
- エラー表でフィルター処理と並び替えができるよう変更
- パッケージ名上のツールチップでそのパッケージの短い説明を表示するよう変更
- コマンドでの [txs-app-dir] と [txs-settings-dir] のサポートを追加（USB スティックで MikTeX ポータブル版と TXS ポータブル版を使用する際にポータブルパスが使用できる）
- qeditor からショートカットを削除できるように変更
- 操作名順でエディタのショートカットの並び替えができるよう変更
- PDF でより厳密にテキスト幅を計算するように変更
- 文書の終わりを超えた段落の背景をオレンジでしるし付けるように変更
- PDF 印刷機能の削除（うまく機能することはない。印刷する際には外部 PDF ビューワーを使用するように。）
- 合字を含むフォントを検出するように変更
- 表に対して「列を揃える」を使用する際にカーソル位置（行と列）を保持するように変更
- 修正： 削除したショートカットを記憶。例：insertSelection に対するタブ
- 修正： keyval 補完に対する特別な場合で補完機能が働く
- 修正： PDF ビューワーでの「次を検索」
- 修正： verbatim 中の % をコメントとして解釈しない
- 修正： ユーザーコマンド名の消失問題
- 修正： PDF ビューワーのモノクロ／カラー設定の更新
- 修正： OS X でヘルプが機能しない問題
- 修正： 古い Qt 4.6 のシステムでも大文字小文字変換が機能する
- 修正： カーソルミラーに対しても大文字小文字変換が機能する
- 修正： 回り込んだ RTL テキストに対してカーソル位置がおかしくなる問題
- 修正： Qt と通常の描画を切り替える際にクラッシュする問題
- 修正： テキスト背景が必ずしも適切な背景色で描画されない問題
- cwls ファイルの更新： yathesis, marvosym, microtype, pifont, glossaries

A.8　Version 2.8.4 での変更点

- PDF との同期の改良: 強調表示されている領域がすでに見えている場合 PDF をページのトップへスクロールしない
- `\DeclareRobustCommand` に対するコマンド検出のサポートの追加
- プレビューパネルのフィットや中心揃えのオプションを永続的に保存するように変更
- 表の自動整形に対してより多くの環境をサポート
- 補助ファイルを削除する際にスコープを記憶するように変更
- 構文チェックでの不完全なオプションの扱いを改良
- 新規追加／改良した cwl ファイル: mathtools, circuitikz
- 修正：RTL（右から左方向へ表示する）テキストの入力でクラッシュする問題
- 修正：いくつかのショートカットが OSX で割り当てられない問題
- 変更：複数カーソル編集を一回のアンドゥ行動にまとめた
- 修正：大きな画像をプレビューする際にスクロール位置がわからなくなる問題
- 変更：エディタ操作に複数のショートカットを割り当てられるようにした
- 修正：選択なしのショートカットでのインデント解除の動作

- 修正：PDF ビューワーでの PgUp/PgDown の動作をより一貫したものへ修正
- 修正：メニューを通じて挿入動作を行った場合にすぐにカーソルミラーを作成するように修正
- 修正：複数の文書で読み込まれる一つのファイルで定義されているコマンドはそれら文書の補完リストでのみ既知となるように修正
- 修正：poppler なしのコンパイルオプション
- 修正：`\newcommand{xyz}{123456789}`でクラッシュする問題
- 修正：`%BEGIN_FOLD` で折りたたまれない問題
- 修正：グレースケールと色反転の設定も拡大鏡へ適用されるよう修正
- 修正：Qt5 でコンパイルした PDF ビューワーでの正しくない検索強調表示領域
- 修正：境界から +/-5 行すでに開始したテキストのドラッグ＆ドロップの動作（現在は 1 行に変更）

A.9　Version 2.8.2 での変更点

- 色補完の追加
- tikz に対する事前に定義された補完をさらに追加
- pdflatex と埋め込みビューワーでの領域プレビューの追加（オプションで有効化した場合）
- 検索／領域プレビューの範囲に対するビジュアルをより一貫したものへ修正
- （グレースケール印刷でどう見えるかのプレビューとして）PDF のグレースケール表示を追加
- インプットメソッドサポートの改良
- `\todo` 強調表示の改良
- texstudio.ini に GUI/ToobarIconSize と GUI/SecondaryToobarIconSize の設定を追加
- ポータブル版でディレクトリへの相対パスを使用するように変更（ポータブル版の現在配置にかかわらず使用可能になった）
- いくつかのバグ修正

A.10　Version 2.8.0 での変更点

- keyval オプションの補完と構文チェックの追加。例：\includegraphics (graphicx!)
- 複数の文書に渡る検索と置換の改良
- ログファイルのパーサーの改良（ファイル名の検出の向上）
- todo スタイルコマンド (例：\todo{}) が todo リストに追加されるように変更
- OSX での標準的なショートカットにより追従するように変更
- コンテキストメニュー（スペルチェックの抑制）に対するキーボード修飾キーを、texstudio.ini の “Editor/-ContextMenuKeyboardModifiers” を通じて設定可能に変更
- 利用可能な場合、Windows での既定のエディタフォントに Consolas を使用するように変更
- 埋め込みビューワーでのツールバーの自動隠蔽機能の改良
- 既定の言語定義の上書きや独自の言語定義の追加をユーザーができるように変更
- makeglossaries のサポートの追加
- include のようなコマンドでのクォートで囲まれたファイル名として、スペースを含むファイル名やディレクトリが使えるように変更
- Windows バージョンは Qt5 でコンパイルするように変更
- LaTeX リファレンスマニュアルの更新
- いくつかのバグを修正

A.11　Version 2.7.0 での変更点

- 「編集」 -> 「テキスト操作」（小文字化 / 大文字化 / タイトルケース化）の追加
- 最近使用したセッションのリストを追加
- 大きな文書の保存の高速化
- 読み込み（import subimport importfrom subimportfrom）に対してファイルツリー上で認識されるように改良（Steven Vandekerckhove のおかげである）
- 実行中のコンパイルの停止ボタンを追加
- 辞書検索のパスに複数のディレクトリを含めることができるように改良
- OpenOffice/LibreOffice 拡張フォーマット (*.oxt) の辞書も利用できるように改良
- LaTeX リファレンスマニュアルを新しいバージョンのものに更新
- 画像ツールチップの最大幅に対するオプションを新しく追加
- 「コンテキストメニューでの参照コマンド」オプションを新しく追加
- エディタロジックの追加検索パスに対するオプションを新しく追加
- エンコーディングの自動検出に対するオプションの新規追加：LaTeX ベースと文字ベースの検出を別個に選択可能に変更
- PDF の強調表示の色と持続時間に対するオプションを新しく追加
- 検索ダイアログ：読み込まれている文書全て、つまり隠れている文書も検索するように変更
- PDF -> ソースの同期の改良
- TeXShop と TeXWorks との互換性のため “`% !BIB = biber`” 構文のサポートを追加
- cwl ファイルをいくつか新しく追加
- いくつかのバグを修正

A.12　Version 2.6.6 での変更点

- 埋め込み PDF ビューワーが開いている場合に home/end キーが正しく働かないバグを修正
- マクロの略語を修正
- エディタ上でログ項目の位置が更新されないバグを修正
- Windows 版インストーラーに署名付加

A.13　Version 2.6.4 での変更点

- パッケージスキャナー：インストールされたパッケージを TeX システムに問い合わせ、存在しないパッケージを強調表示する
- パッケージ補完
- 組み込み PDF ビューワーで基本的な注釈機能をサポート
- 描画速度の向上（特に Mac）
- 隠れた文書の読み込みの高速化（オプション：含まれるファイルの自動読み込み）
- コマンド補完ウィンドウ（特に引用）の高速化
- ログパネルの改良
- dtx 強調表示の改良
- LilyPond book (.lytex) のサポートを追加
- 「編集 -> 行操作」が選択部でも動作するよう改良
- hunspell library を 1.3.2 へ更新

- 修正：インプットメソッドのバグ
- 修正：日本語で矢印キーを含むショートカットが機能していなかった問題
- さらなるバグ修正、例：ツールチップがすぐに消えてしまう問題の修正

A.14　Version 2.6.2 での変更点

- poppler と Windows での Qt(4.8.5) を更新
- 構造ツリービュー：再帰的に構造を閉じる／展開するためのコンテキストメニュー項目を追加
- 結合行での厳密な行のワードラップを改良
- 「表示 -> ビューワーにフォーカスを移動」が別枠ビューワーに対しても機能するように改良
- LanguageTool と辞書の検出が向上
- 「列を揃える」が tabu/longtabu に対しても機能するように改良
- 多数のバグを修正：編集可能なユーザーテンプレート、右から左向きに読む言語での}、Mac での pinyin 入力メソッドの問題、……

A.15　Version 2.6 での変更点

- パッケージの解説文書 PDF が内部 PDF ビューワーで表示されるように変更
- モダン形式に対する Mac での完全な retina サポートを実施
- 画面全体でより読みやすくするため、組み込みビューワーを拡大／縮小できるように改良
- 最後に開いた（そしてまだ閉じていない）環境を alt+enter で閉じることができるように改良
- 「列を揃える」がより多くの環境で機能するように改良
- テンプレートリソースが template_resources.xml を通して設定し、utf-8 で書かれるように変更
- 基本的な Pweave 強調表示を追加
- `% !TeX spellcheck = ...` マジックコメントを追加
- 現在の構文強調に依存する新しいマクロトリガーを追加
- 右から左方向に読む言語に対する双方向表記サポートの改良
- いくつかの小さな修正

A.16　Version 2.5.2 での変更点

- 任意の領域を折りたたみ可能としてしるし付けするための `%BEGIN_FOLD ... %END_FOLD` コメントを新規追加
- PDF ビューワーで CJK とキリル文字の表示のサポートを追加
- タブ幅の最大値を 32 へ増加
- 基本的なインプットメソッドのサポートを修正
- Linux と Mac OS X でのテンプレートの欠損を修正
- Mac OS X でメニューバーが消える問題を修正
- すでに開いているファイルとして保存する際にクラッシュする点を修正
- 長いステータスメッセージのせいでビューワーのサイズが変化しうる問題を修正
- 「次／前の文章」に対するショートカットを Ctrl+PgDown/Up に変更
- いくつかの小さな修正

A.17　Version 2.5.1 での変更点

- 新しいテンプレートシステム
- 折りたたみパネルの改良
- エディターとビューワーでの前方向／後方向マウスボタンのサポートを追加
- インラインプレビューのコンテキストメニューの追加（プレビュー画像のコピー可能に）
- 参照／コマンドの概要を完全にするためすべての含まれるファイルの読み込みのオプションを追加
- `\bibliography{}`コマンドに対する「開く」コンテキストメニュー項目とリンクの重ねあわせの追加
- 図の名前の上に来た時に図のプレビューを表示
- いくつかのバグ修正（PDF のスクロール範囲、ユーザーテンプレートパス、OSX 関連のバグ、……）

A.18　Version 2.5 での変更点

- カーソル履歴の追加（後退／前進）
- 参照、パッケージ、インクルードされるファイルの名前を Ctrl+MouseOver 時にリンクになるよう変更
- 手書きの数式の挿入機能を追加（Windows 7 のみ、TexTablet 使用）
- 好みの書式を指定するオプションを含む、表コード書式の改良
- LaTeX テンプレートと表テンプレートでのメタデータのサポートを追加
- ランタイムライブラリのバージョンが正しいか確認する機能を追加
- コンテキストメニューをさらに追加（折りたたみパネル、ブックマークパネル）
- より見やすくするためもっと太いカーソルをオプションとして追加
- 行操作の追加：上／下へ移動、重複行への操作
- Windows インストーラー：.tex ファイルを TXS へ関連付ける選択肢を追加
- いくつかのバグ修正（クラッシュ、コンパイル、焦点移動、……）

A.19　Version 2.4 での変更点

- いくつかのコマンドを容易に組み合わせることができるビルドシステムに刷新
- 多数の新ツールのサポート：xelatex, lualatex, biber, latexmk, texindy
- 埋め込み PDF ビューワーを追加
- ブックマークマネージャと永続的ブックマークを追加
- LanguageTool を用いたインライン文法チェックを追加
- lua と dtx ファイルの構文強調表示を追加
- biblatex サポートを追加
- 他のアプリケーションから引用を挿入する引用 API を追加（JabRef プラグインを利用可能）
- 表の自動整形
- 外観の改良
- アップデートチェッカーを追加
- スクリプトの拡張：GUI ／ダイアログの作成、他の文章／プログラム／メニューへの接続、バックグラウンドモードとイベント
- クラッシュからの保護機能を追加
- 多数のちょっとした改良
- いくつかのバグ修正

A.20　Version 2.3 での変更点

- \ref/\cite の参照を変更可能なコマンドのリストを追加
- 検索履歴の記録機能を追加
- 文章ごとに異なる辞書を使用する機能のサポートを追加
- 無効な括弧を見つける機能を追加
- ほぼ単語レベルでの逆方向 PDF 検索機能を追加
- 図の挿入マクロでのファイル名の補完機能を追加
- BibTeX の自動呼び出し機能を改良
- スクリプトで利用可能な更に多くの手法を追加
- いくつかのバグ修正（特に PDF ビューワー／構文チェック／構造ビューでのクラッシュ）と細かい改良

A.21　Version 2.2 での変更点

- プレビューの改良：
  - PDF ビューワーで複数のページを連続して表示可能に改良
  - PDF ビューワーを（マルチスレッドで）非停止で機能するように改良
  - プレビューがインクルードされたファイルで機能するように改良
- 任意のユーザーマクロを実行できるようにキーを置換
- ダブルクォートの置換を予め定義しておいたリストから容易に選択できるよう改良
- 補完で通常のコマンド、最も頻繁に使用されるもの、すべての選択可能なものの区別をするように改良
- プロファイルの保存／読み込みが機能するように改良
- 構文強調される環境を増加
- バグ修正と細かい改良

A.22　Version 2.1 での変更点

- オンライン LaTeX の構文チェックの拡張
  - 表中の列数のチェック
  - 文章中でどのコマンドが有効か決めるために\usepackage と\documentclass を利用するように改良
  - 新規コマンドの追加
- TXS が読み込んだ文章の親／子関係を自動検出しそれに応じて振る舞うように変更。従ってマスターモードはもう必要ない。
- プレビューの改良：
  - PDF ビューワーで複数のページを開けるように改良
  - PDF ビューワーでのプレゼンテーションモードと複数ビューのサポートを追加
  - PDF ビューワーの外観と時計ドックを更新
  - 選択部プレビューの高速化とテキスト中表示への対応
- 括弧の選択の容易化
- バグ修正と細かい改良

A.23　Version 2.0 での変更点

- PDF ビューワーと順方向／逆方向検索を結合

- （単純なエラーに対する）オンライン LaTeX の構文チェックを追加
- 表の操作をサポート（行、列、あるいは`\hline` の追加／削除）
- 挿入された括弧を自動的に閉じるように変更
- 厳密な折り返しを伴う行の長さを制限するオプションを追加
- 単語の繰り返しを潜在的なスタイルの間違いとしてしるし付けするように改良
- バグ修正と細かい改良

A.24　Version 1.9.9a での変更点

- Mac でのいくつかのパフォーマンス問題への取り組み。Mac での長い行を高速化。
- ひとつ以上の重ね書きを同時に表示可能に改良（例：構文強調とスペルチェック）
- 補完されたコマンドのコマンド置換を追加
- 切り取りバッファを追加。選択されたテキストが補完を通じてコマンドで置換された場合、除去されたテキストが挿入されたコマンドの引数として使用される（適用できる場合）。
- 補完でのツールチップで選択された参照の示すラベルの周囲が表示されるように改良
- 予め定義しておいたショートカット、メニューの再定義、エディターの設定を含むプロファイルのファイルからの取り込みを追加
- 環境名上でテキストカーソルが待機しているときに、その環境名を（`\begin` と`\end` 同時に）変更できるミラーカーソルを生成するように改良
- ALT-del のタイプで単語やコマンド、環境を削除するように改良
- 既知のテキストコマンドでのみスペルチェックを行うように変更
- いくつかのダイアログを小さな画面サイズでもよりうまく対処できるように修正
- ユーザーフィードバック後多数のバグを除去

A.25　Version 1.9.9 での変更点

- 現在の文書を操作するための java スクリプトがユーザータグで使用可能になる。直接のカーソル処理を通じて操作。さらなる機能が必要な場合、自由に機能要望を上げてよい。
- Mac でのいくつかのパフォーマンス問題への取り組み。まだ完全ではないが、Mac 上でずっと速く感じるはず。
- 式の境界（$、`begin{equation}`、……）上にマウスを合わせると数式構造物をプレビューできるように変更
- ツールバーをカスタマイズ可能に変更
- Math/Latex メニューでの LaTeX の式をユーザーの好みのバージョンに変更可能に改良
- 開いている文書すべてに対する全検索の改良
- SVN を通じたテキスト文書の透過的なバージョン管理のサポート
- 構造ビューとカスタムコマンド補完をタイプした時に更新されるように改良
- 構造ビューで付録の一部やインクルードされたファイルの欠如のようなところを色付けするように変更
- マスター文書が定義してある場合、開いてある（！）サブ文章から参照とラベルを対話式のラベルチェックと補完に用いることができるように改良
- dde コマンドを対応するプログラムが起動していない場合に起動するよう改良
- 起動した LaTeX がフリーズした場合に 2 秒後にエスケープキーを押すことで停止できるよう改良
- 折りたたみは今や本当に有益である：不一致な括弧があってももはや折りたたみに支障ないし、折りたたまれたブロックを編集することも可能
- ユーザーフィードバック後多数のバグを除去

A.26　Version 1.9 での変更点

TeXstudio は前の月の間にこつこつと拡張されてきた。次のリストは新しく追加された機能や変更されたもののおそらく不完全な概要である：

- 動的な構文強調の最初のステップを実装してきた。例えば、`\label` や`\ref` のようなコマンド中の参照がチェックされ、（参照の場合）その参照が存在しない場合や複数回定義されている場合にはしるし付けられる。
- 単語補完システムを拡張してきた。今や既知のコマンドを相当数に拡張している “kile” の単語リストを使用している。bash シェルのように現在の候補リスト中で共通の単語ベースの補完を Tab キーで行える。更に、以前に使用されたテキスト部分を提示することで通常のテキストをも補完できる。この 2 つのモードはバックスラッシュが開始文字かどうかで区別される。そして最後に、補完プロセス時に置換されたユーザー定義の略語を用いて “ユーザータグ”（ユーザー定義のテキストブロック）を挿入できる。キーシーケンスを用いてユーザータグを挿入する以前の方法はまだ利用可能である。終わりに、ユーザー定義の LaTeX コマンドは自動的にスキャンされ、コマンド補完で使用出来る。
- ウィザードを使う場合は別として、テンプレートを用いて新しく文章を作成できる。ユーザーは必要に応じてあとで編集したり削除したりできる自分のテンプレートを追加できる。
- 記号パネルが拡張された。“kile” の記号リストでも拡張された。また原文そのままのリストから “タグ” を挿入できる。そして最後に、列のカウントが利用可能な水平方向のスペースに自動的に適応する。不必要な記号リストを隠せることは言うまでもない。
- 記号リストセレクタをより空きを増やすため texmaker のように左端に移動した。
- カーソルを合わせた際のヘルプを実装した。カーソルを標準の LaTeX コマンドの上に合わせるとツールチップヘルプが表示される。参照の上に合わせた場合、そのラベルを含む対応するテキストメッセージがツールチップとして表示される。
- 選択したテキストのプレビューをステータスパネルかツールチップとして表示するようにした。
- 望むならステータス／ログ／エラーパネルをタブで使用出来る。
- 今やオンラインスペルチェッカーは`‘‘a` や`\’’{a}`のようなエスケープ文字を正しく扱える。また、`\ref{label}`などといった（いくつかの）LaTeX コマンドオプションのスペルチェックを控える。
- 構造ビューのコンテキストメニューで節全体の選択や節のインデントといった有益なオプションを実行できる。これは`\section` を`\subsection` に変更し、それに応じてすべての含まれる見出しも変更することができることを意味している。
- 類語辞典を追加した。これで単語の一部でも検索できるようになった。
- 確かに多数のバグを潰してきた！