

SKK Manual

This edition is for SKK version 14.2.50
\$Date: 2011/04/26 13:27:03 \$

Copyright ©1991-2007 Masahiko Sato(佐藤雅彦),
Yukiyoshi Kameyama(亀山幸義), NAKAJIMA Mikio(中島幹夫), IRIE Tetsuya (入江),
Kitamoto Tsuyoshi (北本剛), Teika Kazura(定家), Tsukamoto Tetsuo(塚本徹雄) and
Tsuyoshi AKIHO (秋保強). Revised by Kiyotaka Sakai(酒井清隆) and Satoshi Harauchi(原内
聡).

Permission is granted to make and distribute verbatim copies of this manual provided the copy-
right notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the condi-
tions for verbatim copying, provided that the entire resulting derived work is distributed under
the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language,
under the above conditions for modified versions, except that this permission notice may be
stated in a translation approved by the author.

1 はじめに

1.1 このバージョンの SKK について

Daredevil SKK (以下 ‘DDSKK’ といいます) は、動作が早くて効率的な日本語入力環境を提供するソフトウェアです。GNU General Public License を適用したフリー・ソフトウェアとして配布されています。

DDSKK 14.2.50 が動作すると思われる Emacsen のバージョンは下記の通りです。

- GNU Emacs 21.4
- GNU Emacs 22.1 以降 (推奨)
- GNU Emacs 23.1 以降 (推奨)
- Mule 機能付きでコンパイルされた XEmacs 21.4 の最新版
- Mule 機能付きでコンパイルされた XEmacs 21.5 の最新版

なお、XEmacs に関しては本体と別配布のパッケージ群は最新版が要求されます。少なくとも xemacs-base パッケージは最新でなければなりません。また、fsf-compat パッケージが動作のために必須です。

総論として、現在は XEmacs よりも GNU Emacs での動作がよくテストされており、最近では XEmacs でのテストは充分行われていません。GNU Emacs 21 では動作テストが充分とは言えません。GNU Emacs 22 以上での利用が最も推奨されます。

現時点で Emacs のバージョンごとに少なくとも以下の制限があります。

GNU Emacs 20.x

DDSKK 14.2 は GNU Emacs 20 をサポートしていません。GNU Emacs 20 のユーザーは DDSKK 14.1 をお使いください。

GNU Emacs 21.4

- JIS X 0213 対応には Mule-UCS の導入が必要です。
- X Window System 上での日本語メニュー表示に対応しません。
- アノテーションとして Wikipedia を表示するには ‘html2text.el’ と ‘w3-url-e21’ の導入が必要です。

GNU Emacs 22.3

- JIS X 0213 対応には Mule-UCS の導入が必要です。
- X Window System 上での日本語メニュー表示は GTK 対応版のみです。

GNU Emacs 23.2

- X Window System 上での日本語メニュー表示は GTK 対応版のみです。

XEmacs 21.4

- ‘skk-kcode.el’ の機能を含む JIS X 0213 対応が機能しません。
- インライン候補表示は機能しません。
- 動的補完における複数候補表示は機能しません。
- ツールチップ表示が機能しません。
- 日本語メニュー表示は X リソースによる方法のみテストされています。

XEmacs 21.5 (beta)

- ‘skk-kcode.el’ の機能を含む JIS X 0213 対応が機能しません。
- インライン候補表示は機能しません。
- 動的補完における複数候補表示は機能しません。
- 日本語メニュー表示は X リソースによる方法のみテストされています。

1.2 SKK とはなにか

SKK はかな漢字変換プログラムであり、Simple Kana to Kanji conversion program にちなんで名付けられました。また、その名は Combinatory Logic での有名な等式 ‘SKK = I’ にも由来しています¹。

Daredevil SKK (DDSKK) は SKK の更なる拡張版として位置付けられています。

ただし、‘SKK モード’、‘SKK 辞書’、‘SKK サーバ’ といった歴史的な用語は引き続き使用しており、DDSKK と呼ばない場合もあります。また、SKK 方式の入力方法を採用したプログラムなど、広く SKK family を意味する場合も同様です。

DDSKK の主な特徴は、次の通りです。

- 多彩な入力方式サポート。ローマ/かな 両対応のかな入力のほか、AZIK、ACT、TUT-code による入力も可能。
- 文法的知識を用いない高速な「かな 漢字」変換。
- 辞書に登録されていない単語のシームレスな登録。
- 確定語を個人辞書へ自動登録することによる効率的な変換候補の表示。
- マイナーモードとして実装されているので、メジャーモードにほとんど影響を与えない。つまり、Emacs との親和性が高い。
- DDSKK 本体 (Emacs Lisp) と辞書ファイルのみで動作 (辞書サーバは必須ではない)。辞書サーバがダウンしていても使用できる。
- 辞書サーバを使うことで、使用メモリの削減が可能。
- ディスク容量に応じて選べる辞書ファイル。
- Emacs のオリジナルと同様の操作で行える日本語インクリメンタル・サーチ。
- Emacs Lisp で書かれたプログラムが返す値を変換候補に挙げるができる。

‘Daredevil’ の名の由来については [Q1-1 Daredevil SKK って SKK とは違うのですか?], page 101 を参照して下さい。

¹ ‘SKK = I’ について詳しくは <http://openlab.jp/skk/SKK.html> をご参照下さい。

2 インストール

2.1 APEL のインストール

DDSKK 14.2 より、GNU Emacs 22 以上において APEL を別途インストールする必要がなくなりました。他に APEL 依存のプログラムを使用していなければ、インストール済の APEL は削除することが可能です。

GNU Emacs 21 及び XEmacs をお使いの場合は DDSKK をインストールする前に APEL (APEL 10.8 以上を推奨) をインストールして下さい。

APEL は次のサイトから入手できます。

<http://kanji.zinbun.kyoto-u.ac.jp/%7Etomo/elisp/APEL/>

2.2 DDSKK のインストール

ここでは、UNIX 上で make コマンドが用意されている場合を想定しています¹。

まず、DDSKK のアーカイブ ‘ddskk-VERSION.tar.gz’ を tar コマンドと gzip コマンドを使用して展開します。

```
% gzip -cd ddskk-14.2.50.tar.gz | tar xvf -
```

次に、DDSKK のトップディレクトリ² をカレントディレクトリにします。

```
% cd ddskk-14.2.50
```

2.2.1 GNU Emacs へのインストール

まずは、DDSKK がどのディレクトリにインストールされるのか、‘what-where’ を引数に make コマンドを実行して確認しましょう。

```
% make what-where
+SKK modules:
+ skk-cursor, skk-viper, ...
+ -> /usr/local/share/emacs/23.2/site-lisp/skk

+SKK infos:
+ skk.info
+ -> /usr/local/share/info

+SKK tutorials:
+ SKK.tut, SKK.tut.E, NICOLA-SKK.tut, skk.xpm
+ -> /usr/local/share/skk
```

この段階でインストール先ディレクトリが表示されない場合は APEL の path を ‘SKK-CFG’ に記入します。‘SKK-CFG’ に設定方法が書いてあります。また、DDSKK のインストール先ディレクトリを変更したい場合も、‘SKK-CFG’ ファイルを編集することにより指定できます。

つぎにスーパーユーザになって、

¹ Microsoft Windows 環境では、makeit.bat を使用することで同様の操作でインストールできます。次の資料を参照してください。

<http://openlab.jp/skk/skk/main/READMEs/README.w32.ja>

cygwin 環境をインストールされている方は make コマンドが使用できるので、ここの説明がそのまま当てはまります。

また、Apple Mac OS X 環境の方は以下の資料を参照してください。

<http://openlab.jp/skk/skk/main/READMEs/README.MacOSX.ja>

² ‘ChangeLog’ や ‘Makefile’ が置かれているディレクトリです。

```
% make install
```

と実行すると、実際に DDSKK がインストールされます。

あるいは、一般ユーザが自分の home directory を root directory としてインストールするには、PREFIX を

```
% make install PREFIX=~/
```

と指定します。

特定の Emacs を指定する場合は、

```
% make install EMACS=mule
```

と指定します。

2.2.2 XEmacs へのインストール

XEmacs でパッケージとしてインストールする場合は、まず、‘what-where-package’ を引数に make コマンドを実行してパッケージのインストール先を確認しましょう。

```
% make what-where-package EMACS=/usr/bin/xemacs
+ xemacs -batch -q -no-site-file -l SKK-MK -f SKK-MK-what-where-package
+ Loading /home/user/temp/ddskk-14.1.50/SKK-CFG...
+ SKK modules:
+ skk-cursor, skk-viper, ...
+ -> /usr/share/xemacs/site-packages/lisp/skk

+ SKK infos:
+ skk.info
+ -> /usr/share/xemacs/site-packages/info

+ SKK tutorials:
+ SKK.tut, SKK.tut.E, NICOLA-SKK.tut, skk.xpm
+ -> /usr/share/xemacs/site-packages/etc/skk
```

インストール先ディレクトリの確認が済みましたら、スーパーユーザになって ‘install-package’ を引数に make を実行すると実際にインストールされます。

```
% make install-package EMACS=/usr/bin/xemacs
+ xemacs -batch -q -no-site-file -l SKK-MK -f SKK-MK-install-package
+ Loading /home/user/temp/ddskk-14.1.91/SKK-CFG...
...
```

2.2.3 対話的なインストール

DDSKK 14.3 で対話的インストーラが追加されました。

まず、*M-x dired* とタイプして dired を起動してください。このとき、ディレクトリを問われますので「DDSKK のアーカイブを展開したディレクトリ」を指定してください。

次に、表示されたディレクトリ一覧の ‘SKK-MK’ にカーソルをあわせて L をタイプしてください。プロンプト ‘Load SKK-MK?’ には y をタイプしてください。

以降、インストーラからの質問に答えてインストール作業を進めてください。

2.3 辞書について

DDSKK を使用するには、いわゆる辞書（主にかなと漢字の対応を記述したデータ）が必要です。

従来の DDSKK を利用するには、後述する SKK 辞書の別途インストールが必要でした。DDSKK 14.2 より、GNU Emacs との組み合わせにおいて Emacs 同梱の辞書データ (ja-dic) を利用した変換に対応し、別途辞書ファイルをインストールしなくても最低限の使用ができるようになりました。(XEmacs では利用できません)

ja-dic は Emacs の入力メソッド LEIM のために SKK-JISYO.L から変換して生成されたものです。英数変換や数値変換などのエントリは削除されているものの、通常のかな漢字変換については SKK-JISYO.L と同等にできます。

とはいえ、有志の知恵を結集して作られている各種 SKK 辞書は便利ですから、是非入手してインストールしましょう。

2.4 辞書の入手

DDSKK を使用する SKK 辞書群の入手について解説します。

SKK 各辞書の解説とダウンロード

<http://openlab.jp/skk/wiki/wiki.cgi?page=SKK%BC%AD%BD%F1>

上記のサイトには様々な辞書が用意されています。以下は一例です。

SKK-JISYO.S

S 辞書（主に単漢字が登録。最小限必要な語を収録）

SKK-JISYO.M

M 辞書（普通に使う分には足りる程度）

SKK-JISYO.ML

M 辞書と L 辞書の間サイズの辞書。L 辞書収録語の内、EPWING 辞書やオンライン辞書で正しいと判別された語をベースにして加除。

SKK-JISYO.L

L 辞書（あらゆる単語を収録）

zipcode 郵便番号辞書

SKK-JISYO.JIS2

JIS X 0208 で定められている第 2 水準の文字を、部首の読みを見出し語として単漢字を収録した辞書

SKK-JISYO.JIS3_4

JIS 第 3 水準、第 4 水準の文字に代表される、JIS X 0208 には含まれないが JIS X 0213 には含まれる文字及びそれらを含む語録を収録した辞書

SKK-JISYO.public+

public+ 辞書

SKK-JISYO.edict

edict 辞書（英和辞書）

SKK-JISYO.wrong

間違い辞書（S, M, L 辞書に既に登録されていたが、間違いであったので削除された単語を収録）

一部の辞書の著作権は GNU GPL v2 とは異なるものがありますのでご注意ください。詳細は、次の資料を参照して下さい。

<http://openlab.jp/skk/skk/dic/READMEs/committers.txt>

2.5 辞書を DDSKK と同時にインストールする

DDSKK のソースを展開すると、中に 'dic' というディレクトリが存在します。'SKK-JISYO.L' などをごこのディレクトリにコピーしてから

```
% make install
```

を実行すると、辞書ファイルがチュートリアル（'SKK.tut'）と同じディレクトリ（'/usr/share/skk' や 'c:/emacs-23.2/etc/skk' など）にインストールされます。

さらに、'SKK-JISYO.L' を DDSKK ソースの 'dic' ディレクトリにコピーしてから

```
% make cdb
```

を実行すると、CDB 形式辞書 ‘SKK-JISYO.L.cdb’ が生成されます。この後で上記 `make install` を実行すると ‘SKK-JISYO.L.cdb’ も ‘SKK-JISYO.L’ と共にインストールされます³。

2.6 辞書サーバの入手

辞書サーバはオプションですので無くても DDSKK は動作しますが、特に辞書のサイズが大きな場合は辞書サーバを利用することで省メモリ効果を得られます。また、辞書サーバによっては複数辞書の検索、EPWING 辞書の検索ができたりするものもあります。

DDSKK は特定の辞書サーバの実装に依存していませんので、下記の辞書サーバのいずれでも動作可能です。ソースやバイナリの入手、インストールについてはそれぞれのウェブサイトをご参照下さい。

辞書サーバの説明とリンク

<http://openlab.jp/skk/skkserv-ja.html>

³ CDB 形式辞書の生成には python スクリプト ‘etc/skk2cdb.py’ が実行されるので、あらかじめ python をインストールしておく必要があります。

3 はじめの設定

標準的にインストールした場合は、特段の設定なしに Emacs を起動するだけで DDSKK が使える状態になります。自動的に 'skk-setup.el' というファイルが読み込まれ、設定されます¹。

この自動設定によらずに手動で設定したい場合は、以下の説明を参照してください。

3.1 最も基本的な設定

自動設定によらず、手動で設定する場合は、以下のように '~/.emacs' に書きます²。

```
(require 'skk-autoloads)
(global-set-key "\C-x\C-j" 'skk-mode)
(global-set-key "\C-xj" 'skk-auto-fill-mode)
(global-set-key "\C-xt" 'skk-tutorial)
```

ただし XEmacs でパッケージとしてインストールをした場合は、

```
(require 'skk-autoloads)
```

の記述はしません。

辞書サーバを使わない場合は、更に辞書ファイルを指定する必要があります。

```
(setq skk-large-jisyo "/your/path/to/SKK-JISYO.L")
```

この場合 skk-large-jisyo は Emacs のバッファに取り込んで使用します。これによりメモリが圧迫されすぎる場合は、上記の 'SKK-JISYO.L' を 'SKK-JISYO.M'、'SKK-JISYO.ML' 又は 'SKK-JISYO.S' に変更します。

なお、DDSKK 14 より辞書サーバを経由せずとも CDB 形式の辞書ファイルを直接利用できるようになりました。CDB 形式辞書ファイルを利用する場合は以下のように指定してください。

```
(setq skk-cdb-large-jisyo "/your/path/to/SKK-JISYO.L.cdb")
```

skk-large-jisyo と skk-cdb-large-jisyo を共に指定した場合には、標準では CDB 辞書の方が先に検索されます。これに関しては Section 5.10.3.1 [辞書検索の設定の具体例], page 69 も参照してください。

3.2 インクリメント検索の設定

以下のように '~/.emacs' に書きます。

```
(add-hook 'isearch-mode-hook
  #'(lambda ()
      (when (and (boundp 'skk-mode)
                 skk-mode
                 skk-isearch-mode-enable)
            (skk-isearch-mode-setup))))
(add-hook 'isearch-mode-end-hook
  #'(lambda ()
      (when (and (featurep 'skk-isearch)
                 skk-isearch-mode-enable)
            (skk-isearch-mode-cleanup))))
```

¹ Emacs が起動する過程で 'SKK_LISPDIR/leim-list.el' が読み込まれます。

'leim-list.el' は 'skk-autoloads.el' と 'skk-setup.el' を require します。

'skk-autoloads.el' はインストール時に自動的に生成されるファイルであり、各関数を自動ロード (autoload) するように定義しています。

'skk-setup.el' はキーバインド ($C-x C-j \Rightarrow 'skk-mode$)、変数 skk-tut-file の定義及びインクリメンタル・サーチの定義を行っています。

² サンプルとして、配布物に 'etc/dot.emacs'、'etc/dot.skk' ファイルがあります。参考にして下さい。

上記の設定では、DDSKK が起動されているときのみ、付属のインクリメンタル・サーチを使用します。常に付属のインクリメンタル・サーチを使用したかったら上記の `isearch-mode-hook`、`isearch-mode-end-hook` の設定を下記の設定に変更します。

```
(add-hook 'isearch-mode-hook 'skk-isearch-mode-setup)
(add-hook 'isearch-mode-end-hook 'skk-isearch-mode-cleanup)
```

3.3 辞書サーバを使いたいときの設定

辞書サーバを使いたいときは、`~/ .skk` で以下のように設定します。

```
(setq skk-server-host "example.org")
(setq skk-server-portnum 1178)
```

`skk-server-host` [ユーザ変数]
辞書サーバが起動しているホスト名又は IP アドレス。

`skk-server-portnum` [ユーザ変数]
辞書サーバが使うポート番号。'/etc/services' に `skkserv` のエントリがあれば指定する必要は無い。

辞書サーバが起動していなかったときに Emacs から `skkserv` プロセスを立ち上げる事もできます。`skk-server-inhibit-startup-server` を `nil` にする事でこの機能が有効になります。Section 5.10.5 [サーバ関連], page 71 も参照してください。

Emacs から立ち上げて利用する事ができる辞書サーバは、

```
skkserv [-p port] [jisyo]
```

のようなオプションを受け付け、`inetd` などを経由せず直接起動するものに限られます。

辞書サーバプログラムと辞書ファイルは、次のように設定します。

```
(setq skk-server-prog "/your/path/to/skkserv")
(setq skk-server-jisyo "/your/path/to/SKK-JISYO.L")
```

`skk-server-prog` [ユーザ変数]
辞書サーバプログラムをフルパスで指定する。

`skk-server-jisyo` [ユーザ変数]
辞書サーバに渡す辞書をフルパスで指定する。辞書サーバによっては辞書ファイルの指定を独自の方法で行ない `emacs` からの指定を無視するものもあります。詳しくは各辞書サーバの説明書を読んで下さい。

これらの設定は、環境変数を利用して下記のようにすることもできます。

B シェルの場合 (sh, bash, ksh, zsh など)

```
export SKKSERVER=example.org
export SKKSERV=/your/path/to/skkserv
export SKK_JISYO=/your/path/to/SKK-JISYO.L
```

C シェルの場合 (csh, tcsh など)

```
setenv SKKSERVER example.org
setenv SKKSERV /your/path/to/skkserv
setenv SKK_JISYO /your/path/to/SKK-JISYO.L
```

関連項目: Section 2.6 [辞書サーバの入手], page 6、Section 5.10.5 [サーバ関連], page 71

4 基本的な使い方

本章では、DDSKK の基本的な使用方法を説明します。これを読めば、とりあえず DDSKK を使ってみるには充分です。

DDSKK を使った入力方法に慣れるには、付属のチュートリアルが最適なので、お試しください。

See Section 4.5 [チュートリアル], page 18.

なお、次章の「便利な応用機能」は、興味のある個所のみをピックアップしてお読みになるのがいいでしょう。

4.1 起動と終了

SKK モードに入るには `C-x C-j`、もしくは `C-x j` とタイプします。モードラインの左端には、下記のように ‘--かな:’ が追加されます。

```
--かな:MULE/7bit----- Buffer-name (Major-mode)---
```

また、カーソルの色が変化します¹。

`C-x C-j`、もしくは `C-x j` を再度タイプすることで、SKK モードに入る前のモードに戻り、カーソル色も元に戻ります²。

`skk-status-indicator` [ユーザ変数]

デフォルトはシンボル `'left` です。この変数をシンボル `'minor-mode` と設定すれば、インジケータはモードラインのマイナーモードの位置に表示されます。

```
-MULE/7bit----- Buffer-name (Major-mode かな)---
```

`skk-preload` [ユーザ変数]

`'~/emac`s' にて変数 `skk-preload` を `non-nil` と設定することにより、DDSKK の初回起動を速くすることができます。

```
(setq skk-preload t)
```

これは、SKK 本体プログラムの読み込みと変数 `skk-search-prog-list` に指定された辞書の読み込みを Emacs の起動時に済ませてしまうことにより実現しています。そのため、Emacs の起動そのものは遅くなりますが、DDSKK を使い始めるときのレスポンスが軽快になります。

コマンド `skk-restart` [Function]

`M-x skk-restart` と実行すると SKK を再起動します。`'~/skk`' は再ロードしますが、`'~/emac`s' は再ロードしません。

4.1.1 SKK オートフィルモード

`C-x j` とタイプすれば、SKK モードに入ると同時にオートフィルモード (see Section “Auto Fill” in *GNU Emacs Manual*) をオンにします。

既にオートフィルモードがオンになっているバッファで `C-x j` をタイプすると、オートフィルモードは逆にオフになるので注意してください。

バッファの状態にかかわらず強制的にオートフィルモード付で SKK モードに入りたい場合は、`M-1 C-x j` や `C-u C-x j` などとタイプし、このコマンドに正の引数を渡します³。

オートフィルモードをオフにし、かつ SKK モードも終了したい場合には `M-- C-x j` や `C-u -1 C-x j` などとタイプし、このコマンドに負の引数を渡します。

¹ カラーディスプレイを使用し、カラー表示をサポートしている Window System 下で対応する Emacs を使用している場合。

See Section 5.14.3 [入力モードを示すカーソル色に関する設定], page 87.

² ただし、「アスキーモード」を利用すれば SKK モードから抜ける必要はほとんどありません。

See Section 4.2 [アスキーモード], page 10.

³ 「引数」については、

Section “Arguments” in *GNU Emacs Manual*.

を参照のこと。

4.1.2 辞書の保存

Emacs を終了するときは、保存前の個人辞書を ‘~/skk-jisyo.BAK’ に退避してから、個人辞書 (see Section 5.10.1 [個人辞書], page 67) の内容を ‘~/skk-jisyo’ に保存します。

‘~/skk-jisyo’ や ‘~/skk-jisyo.BAK’ のファイル名を変更したければ、それぞれ `skk-jisyo` や `skk-backup-jisyo` の値を変更して下さい。

個人辞書を保存せずに Emacs を終了させたい場合には、

```
M-x skk-kill-emacs-without-saving-jisyo
```

とタイプします。

個人辞書の保存動作について更に詳しくは、Section 5.10.10 [個人辞書の保存動作], page 74 を参照してください。

4.2 入力モード

SKK モードは、文字種類による 4 種類の「入力モード」と、辞書を用いた変換の状態により 3 つの「変換モード」を持ちます。

4.2.1 入力モードの説明

「かなモード」

アスキー小文字をひらがなに変換するモード。

マイナーモードの表示: ‘かな’

カーソル色: 赤系

「カナモード」

アスキー小文字をカタカナに変換するモード。

マイナーモードの表示: ‘カナ’

カーソル色: 緑系

「全英モード」

アスキー小文字 / 大文字を全角アルファベットに変換するモード。

マイナーモードの表示: ‘全英’

カーソル色: 黄系

「アスキーモード」

文字変換を行わないモード。入力されたキーは `C-j` を除いて通常の Emacs のコマンドとして解釈される。

マイナーモードの表示: ‘SKK’

カーソル色: 背景によりアイボリーかグレイ。

入力モードに伴うカーソル色の変更方法については、Section 5.14.3 [入力モードを示すカーソル色に関する設定], page 87 を参照してください。

4.2.2 入力モードを切り替えるキー

- `q` 「かなモード」, 「カナモード」間をトグルする。
- `l` 「かなモード」又は「カナモード」から「アスキーモード」へ。
- `L` 「かなモード」又は「カナモード」から「全英モード」へ。
- `C-j` 「アスキーモード」又は「全英モード」から「かなモード」へ。

実際にはカナモードや全英モードで長時間入力を続けることはほとんどないので、かなモードのままカナ文字や全英文字を入力する便法が用意されています。(see Section 5.3.1 [かなモードからカタカナを入力], page 24, see Section 5.3.2 [全英文字の入力], page 25)

4.3 変換モード

変換モードは、次の 3 種類のいずれかです。

「モード (確定入力モード)」

あるキー入力に対応する文字列を、辞書を用いた文字変換を行わずに直接バッファへ入力するモード。入力モードに応じてローマ字からひらがな、ローマ字からカタカナ、あるいはアスキー文字から全角アルファベットへの文字変換を行う。

「モード」

辞書変換の対象となる文字列「見出し語」を入力するモード。

「モード」

見出し語について、辞書変換を行うモード。

また、モードの変種として *SKK abbrev mode* があり、モードのサブモードとして、「辞書登録モード」があります。

4.3.1 モード

確定入力モードを「モード」と呼びます。モードでは、あるキー入力に対応した特定の文字列への変換を行うだけで、辞書変換は行いません。アスキー文字列から、入力モードに応じて、ひらがな、カタカナ、あるいは全角アルファベットへの文字変換を行います。変換を行うカレントバッファにこのモード特有のマークは表示されません。

かなモード、カナモードで、かつモードである場合、デフォルトの入力方法はいわゆるローマ字入力です。訓令式、ヘボン式のどちらによっても入力することができます。主な注意点は以下のとおりです。

- ‘ん’ は *nn* 又は *n'* で入力する。直後に ‘n’、‘y’ 以外の子音が続くときは *n* だけで入力できる。
- 促音は、*chotto* ⇒ ‘ちよつと’、*moppara* ⇒ ‘もっぱら’ のように次の子音を重ねて入力する。
- 促音や拗音 (ひらがなの小文字) を単独で入力するときは、*xa* ⇒ ‘あ’、*xya* ⇒ ‘や’ などのように *x* を用いる。
- 長音は、*-* で入力する。 *-* ⇒ ‘ー’。

4.3.2 モード

「モード」では、辞書変換の対象となる文字列を入力します。かなモード、もしくはカナモード⁴で、かつ、モードであるときに、キー入力を大文字で開始することで、モードに入ります。例えば、

```
K a n j i
----- Buffer: foo -----
  かんじ*
----- Buffer: foo -----
```

のようにタイプすることでモードに入り、続けて辞書変換の対象となる文字列「見出し語」を入力します。‘ ’マークは「モードである」という表示ですが、見出し語の開始点を示す表示でもあります。

4.3.2.1 後からモードに入る方法

辞書変換の対象としたい文字列であったにも関わらず、先頭の文字を大文字で入力し忘れた場合は、その位置までポイント⁵を戻してから *q* をタイプすることで、モードに入ることができます。例えば、下記のように操作します (* の地点にカーソルがあります)。

⁴ See Section 4.2 [かなモード、カナモード], page 10.

⁵ See Section “ポイント” in *GNU Emacs Manual*.

```
k a n j i
```

```
----- Buffer: foo -----
```

```
かんじ*
```

```
----- Buffer: foo -----
```

```
C-u 3 C-b
```

```
----- Buffer: foo -----
```

```
*かんじ
```

```
----- Buffer: foo -----
```

```
Q
```

```
----- Buffer: foo -----
```

```
*かんじ
```

```
----- Buffer: foo -----
```

```
C-e
```

```
----- Buffer: foo -----
```

```
かんじ*
```

```
----- Buffer: foo -----
```

‘7 がつ 24 にち’ のように大文字から始めることができない文字列を見出し語としたい場合は、*Q* をタイプして モードにしてから ‘7 がつ 24 にち’ の文字列を入力します。

なお、モードでは、文字列の間に空白を含めることはできません。これは、辞書エントリの見出し語に空白を含めることができない制限からきています。

4.3.2.2 モードを抜ける方法

誤って モードに入ってしまったときは、*C-j* とタイプして モードに戻るか、*C-g* とタイプして見出し語を消去するかの 2 通りの処理の方法があります。具体例を下記に示します。

```
K a n j i
```

```
----- Buffer: foo -----
```

```
かんじ*
```

```
----- Buffer: foo -----
```

```
C-j
```

```
----- Buffer: foo -----
```

```
かんじ*
```

```
----- Buffer: foo -----
```

あるいは、

```
K a n j i
```

```
----- Buffer: foo -----
```

```
かんじ*
```

```
----- Buffer: foo -----
```

```
C-g
```

```

----- Buffer: foo -----
*
----- Buffer: foo -----

```

4.3.3 モード

「モード」では、モードで入力した見出し語を、辞書に従って変換する作業を行います。モードで見出し語を入力した後に SPC をタイプすることでモードに入ります。‘ ’ マークから SPC をタイプしたときのポイントまでの文字列が見出し語として確定され、‘ ’ マークは‘ ’ マークで置き換えられ、この見出し語が辞書の中で検索されます。

4.3.3.1 送り仮名が無い場合

仮に、辞書に

```
かんじ /漢字/幹事/
```

というエントリ⁶を含むとして、例を示します。

```

K a n j i

----- Buffer: foo -----
かんじ*
----- Buffer: foo -----

SPC

----- Buffer: foo -----
漢字*
----- Buffer: foo -----

```

この例では、モードにおける‘ ’ マークからポイントまでの間の文字列‘かんじ’を辞書変換の対象文字列として確定し、それについて辞書内での検索を行っています。実際の変換動作では、候補部分がハイライト表示されます⁷。

‘漢字’が求める語であれば C-j をタイプしてこの変換を確定します。ハイライト表示も‘ ’ マークも消えます。

また、C-j をタイプせずに新たな確定入力が続けて行うか又は新たな変換を開始すると、直前の変換は自動的に確定されます。これを「暗黙の確定」と呼んでいます。副作用として確定を伴うキーは、印字可能な文字全てと RET です。ただし、Section 5.7.4 [暗黙の確定のタイミング], page 58 を参照してください。

4.3.3.2 次候補・前候補

求める語がすぐに表示されなければ、更に続けて SPC をタイプすることで、次候補を検索します。

```

----- Buffer: foo -----
漢字*
----- Buffer: foo -----

SPC

----- Buffer: foo -----
幹事*
----- Buffer: foo -----

```

⁶ 本マニュアルでは、見出し語と候補群を合わせた一行を「エントリ」と呼びます。詳細は、Section 5.10.6.1 [送りありエントリと送りなしエントリ], page 72 を参照してください。

⁷ ハイライト表示は FSF Emacs の Overlays、XEmacs の extent の機能を使用しています。

候補が5つ以上あるときは、5番目以降の候補は7つずつまとめてエコーエリアに表示されません。例えば、辞書が

きょ /距/巨/居/裾/嘘/拒/拠/虚/挙/許/渠/据/去/

というエントリを含むときに *Kyo* の後に SPC を5回続けてタイプすれば

```
----- Echo Area -----
A:嘘 S:拒 D:拠 F:虚 J:挙 K:許 L:渠 [残り 2]
----- Echo Area -----
```

がエコーエリア⁸に表示されます。ここで仮に‘許’を選択したければ *k* を入力します。

‘A’, ‘S’, ‘D’, ‘F’, ‘J’, ‘K’, ‘L’ の各文字は、押し易さを考慮してキーボードのホームポジションから横方向に一直線に配置されているキーが選ばれています。また、候補の選択のために押すキーは、大文字、小文字のいずれでも構いません。候補の選択に用いるキーの変更については、Section 5.6.5 [候補の選択に用いるキー], page 51 を参照してください。

SPC を連打してしまい求める候補を誤って通過してしまったときは *x* をタイプすれば前候補 / 前候補群に戻ることができます⁹。

候補を次々と探しても求める語がなければ、自動的に辞書登録モードになります (辞書登録モードは モードのサブモードです)。Section 4.3.4 [辞書登録モード], page 15 にて説明します。

`skk-previous-candidate-keys` [ユーザ変数]
 前候補 / 前候補群に戻る関数 `skk-previous-candidate` を割り当てるオブジェクトのリストを指定する。オブジェクトにはキーを表す文字列または `event vector` が指定できます。
 デフォルトは `(list "x" "\C-p")` です。

4.3.3.3 送り仮名が有る場合

次に送り仮名のある単語について説明します。

‘動く’を変換により求めたいときは *UgoKu* のように、まず モード に入るために *U* を大文字で入力し、次に送り仮名の開始を `DDSKK` に教えるために *K* を大文字で入力します。送り仮名の *K* をタイプした時点で SPC をタイプすることなく、モード に入り辞書変換が行われます。

送り仮名の入力時 (ローマ字プレフィックスが挿入された瞬間) プレフィックスの直前に ‘*’ を一瞬挿入し、送り仮名の開始時点を示します。プレフィックスに続くキー入力で、かな文字が完成した時点で ‘*’ は消えます。

キー入力を分解して追いながらもう少し詳しく説明します。

```
Ugo
----- Buffer: foo -----
うご*
----- Buffer: foo -----

K
----- Buffer: foo -----
うご*k*
----- Buffer: foo -----

u
```

⁸ エコーエリアとミニバッファは視覚的には同一の場所にありますが、エコーエリアが単にユーザへのメッセージを表示するのみであるのに対し、ミニバッファは独立したバッファとして機能する点が異なります。

⁹ ‘x’ は小文字で入力する必要があります


```
----- Buffer: foo -----
 動く*
----- Buffer: foo -----
```

このように、DDSKK では送り仮名の開始地点をユーザが明示的に入力するので、システム側で送り仮名を分解する必要がありません。これにより、高速でヒット効率が高い変換が可能になります。See Section 5.8.3 [送り仮名の自動処理], page 62.

ただし、サ変動詞の変換では、サ変動詞の語幹となる名詞を「送りなし変換」¹⁰ として変換し、その後‘する’をモードで入力した方が効率が良くなります。See Section 4.3.4.3 [サ変動詞の入力], page 16.

4.3.4 辞書登録モード

DDSKK には独立した辞書登録モードはありません。その代わりに、辞書にない単語に関して変換を行った場合に、自動的に辞書登録モードに入ります。例えば辞書に

```
へんかんちゅう /変換中/
```

のエントリがない場合に、‘変換中’を入力しようとして、*Henkantyuuspc* とタイプすると、下記のように、カレントバッファはモードのまま‘へんかんちゅう’に対して変換ができない状態で休止し、同時にミニバッファに‘へんかんちゅう’というプロンプトが表示されます。

```
----- Buffer: foo -----
 へんかんちゅう
----- Buffer: foo -----

----- Minibuffer -----
[辞書登録] へんかんちゅう: *
----- Minibuffer -----
```

もちろん、誤って登録した単語は削除できます。(see Section 5.10.8 [誤った登録の削除], page 73, see Section 5.10.9 [個人辞書ファイルの編集], page 74)

4.3.4.1 送り仮名が無い場合の辞書登録

辞書登録モードでは、キー入力はミニバッファに対して行われます。仮に辞書に

```
へんかん /変換/
ちゅう /中/
```

のようなエントリがあるとして、ミニバッファで‘変換中’の文字列を‘変換’と‘中’とに分けて作ります。

```
Henkan SPC T y u u SPC

----- Minibuffer -----
[辞書登録] へんかんちゅう: 変換 中*
----- Minibuffer -----
```

ここでRETをタイプすれば‘変換中’が個人辞書に登録され、辞書登録モードは終了します¹¹。同時に、変換を行っているカレントバッファには‘変換中’が挿入され確定されます。See Section 5.10.1 [個人辞書], page 67.

辞書登録モードを抜けたいときはC-gをタイプするか、または何も登録せずRETをタイプするとモードに戻ります。

¹⁰ 詳細は、Section 4.3.3.1 [送り仮名が無い場合], page 13 を参照してください。

¹¹ ここでは暗黙の確定が行われるのでC-jをタイプする必要はありません。ただし、Section 5.6.6 [モードでのRET], page 52 を参照してください。

4.3.4.2 送り仮名が有る場合の辞書登録

送り仮名のある単語の登録では、ミニバッファで作る候補に送り仮名そのものを登録しないように注意しなければいけません。仮に辞書に

うごk /動/

というエントリがないとして、例を挙げて説明します。

U g o K u

```
----- Buffer: foo -----
```

うごく

```
----- Buffer: foo -----
```

```
----- Minibuffer -----
```

[辞書登録] うご*k*: *

```
----- Minibuffer -----
```

ミニバッファで辞書登録すべき文字列は‘動’だけであり、送り仮名の‘く’は含めてはいけません。‘動く’と登録してしまうと、次に *U g o K u* とタイプしたときに出力される候補が‘動くく’になってしまいます¹²。

D o u S P C

```
----- Minibuffer -----
```

[辞書登録] うご*k*: 動*

```
----- Minibuffer -----
```

RET

```
----- Buffer: foo -----
```

動く*

```
----- Buffer: foo -----
```

4.3.4.3 サ変動詞の辞書登録に関する注意、

サ変動詞（名詞の後に‘する’を付けた形で構成される動詞）については‘する’を送り仮名とした送りあり変換（see Section 4.3.3.3 [送り仮名が有る場合], page 14）をしないで、‘運動’と‘する’と分けて入力することを前提としています。¹³

例えば‘運動する’は *U n d o u S P C s u r u* とタイプすることにより入力できます。名詞から作られる形容詞等も同様です。

4.3.4.4 再帰的辞書登録

ミニバッファを再帰的に使って辞書登録を再帰的に行うことができます。

仮に辞書に

さいきてき /再帰的/

さいき /再帰/

のようなエントリがなく、かつ

¹² 変数 *skk-check-okurigana-on-touroku* が *non-nil* であれば、辞書登録時に送り仮名のチェックを行います。この変数に、シンボル *ask* を設定すれば、ユーザに確認を求め、送り仮名と認められれば送り仮名を取り除いてから登録します。シンボル *auto* を設定すれば、ユーザに確認を求めず、勝手に送り仮名を判断して削除してから登録します。*skk-check-okurigana-on-touroku* のデフォルトは *nil* です。

¹³ *SKK-JISYO.L* など共有辞書のメンテナンス上、原則としてサ変動詞を送りありエントリに追加していません。そのため、‘する’を送り仮名とした送りあり変換では、辞書に候補がなく辞書登録モードに入ってしまうので、名詞として分解して入力することが一般的です。ただし、*DDSKK 13* 以降では暫定的にサ変動詞の送りあり変換を可能にする機能を用意しました。（see Section 5.5.10 [サ変動詞変換], page 45）

```

さい /再/
き /帰/
てき /的/

```

のようなエントリがあるとします。

ここで *Saikitekispc* とタイプすると、文字列 ‘さいきてき’ に対する候補を見つけられないので、ミニバッファに ‘さいきてき’ というプロンプトを表示して辞書登録モードに入ります。

‘さいきてき’ に対する辞書エントリを作るため *Saikispc* とタイプすると、更にこの候補も見つけられないので、ミニバッファに ‘さいき’ というプロンプトを表示して、再帰的に ‘さいき’ の辞書登録モードに入ります。

Saispc Kispc とタイプすると、ミニバッファは、

```

----- Minibuffer -----
[[辞書登録]] さいき: 再 帰
----- Minibuffer -----

```

となります。プロンプトが ‘[[辞書登録]]’ となり、‘[]’ がひとつ増えていますが、この ‘[]’ の数が再帰的な登録モードの深さを表わしています。ここで RET をタイプすると、個人辞書には

```

さいき /再帰/

```

というエントリが登録され、ミニバッファは ‘さいきてき’ の辞書登録モードに戻り、プロンプトは ‘さいきてき’ となります。

今度は ‘再帰’ が変換可能なので *Saikitekispctekispc* とタイプすると、

```

----- Minibuffer -----
[[辞書登録]] さいきてき: 再帰 的
----- Minibuffer -----

```

となります。ここで RET をタイプすることで、‘さいきてき’ の辞書登録モードから抜け、個人辞書に

```

さいきてき /再帰的/

```

というエントリが登録されます。カレントバッファのポイントには、‘再帰的’ が挿入されます。

4.3.4.5 改行文字を含む辞書登録

改行文字を含む文字列を辞書に登録するには、辞書登録モードで改行文字を *C-q C-j* により入力します。例えば、

```

〒 980
仙台市青葉区片平 2-1-1
東北大学電気通信研究所

```

を辞書に登録するには、辞書登録モードで、

```

‘〒 980’,
C-q C-j,
‘仙台市青葉区片平 2-1-1’,
C-q C-j,
‘東北大学電気通信研究所’,
RET

```

と入力します。

4.4 インクリメンタル・サーチ

DDSKK では、専用のインクリメンタル・サーチプログラムを Emacs 添付の ‘isearch.el’ のラッパーとして実装しているため、日本語文字列のインクリメンタル・サーチをアスキー文字と同様の操作で行うことができます。

4.4.1 skk-isearch の操作性

大部分の動作は、Emacs オリジナルのインクリメンタル・サーチのままですから、Emacs オリジナルのインクリメンタル・サーチのコマンド¹⁴ やユーザ変数でのカスタマイズ¹⁵ もそのまま利用できます。

Emacs オリジナルのインクリメンタル・サーチについては、Section “Incremental Search” in *GNU Emacs Manual* を参照してください。

インクリメンタル・サーチ中の入力方法は、通常のバッファにおける各入力モード、変換モードでの入力方法と同一です。

C-s や *C-r*、あるいは *M-C-s* や *M-C-r* でインクリメンタル・サーチを起動すると、インクリメンタル・サーチを起動したバッファの入力モードと同一の入力モードで、キーとなる文字の入力が可能となります。

4.4.2 skk-isearch と入力モード

入力モードに合わせて、インクリメンタル・サーチのプロンプトが表示されます。プロンプトの種類は、以下の 5 つです。

‘I-search: [aa]’
アスキーモード

‘I-search: [か]’
かなモード

‘I-search: [カ]’
カナモード

‘I-search: [英]’
全英モード

‘I-search: [--]’
インクリメンタル・サーチモードで *C-x C-j* などをタイプして DDSKK を終了した場合は、このプロンプトが表示されます。

4.5 チュートリアル

DDSKK には、基本的な操作方法を学習できるチュートリアルが附属しています。日本語版チュートリアルは *M-x skk-tutorial* で、英語版チュートリアルは *C-u M-x skk-tutorial RET English RET* で実行します。

`skk-tut-file` [ユーザ変数]

チュートリアルファイルが標準の場所に置かれていない場合は、‘~/emacs’で

```
(setq skk-tut-file "/usr/local/share/skk/SKK.tut")
```

と書くことにより、指定したチュートリアルファイルを使用させることができます。英語版のチュートリアルファイルは、‘skk-tut-file’に‘.E’が付いたファイル名です。この場合であれば、‘/usr/local/share/skk/SKK.tut.E’になります。

¹⁴ *M-y* の *isearch-yank-kill* や *M-p* の *isearch-ring-retreat*, *M-n* の *isearch-ring-advance* など

¹⁵ *search-highlight* など

5 便利な応用機能

5.1 ファイル構成

SKK の基本的な機能は、`'skk.el'` に収められています。一方、DDSKK 14 で応用機能を提供するプログラムのほとんどは `'skk.el'` とは別のファイルに収めています。これらは、必要に応じてオートロードするように設計されています。各応用機能の概略と該当のファイル名について説明します。

また、DDSKK の変数は `'skk-vars.el'` に集約されています。カスタマイズしたい場合などにこのファイルを見ると参考になるかもしれません。

`'context-skk.el'`

編集の文脈に応じて自動的に skk のモードを切り替えたり、SKK の各種設定を変更する機能を提供します。

See Section 5.3.5 [文脈に応じた自動モード切り替え], page 27.

`'skk-abbrev.el'`

SKK abbrev モードの機能を提供するプログラムを集めたファイル。

See Section 5.5.5 [SKK abbrev mode], page 41.

`'skk-act.el'`

dvorak 配列での拡張ローマ字入力 "ACT" を SKK で使うための設定を提供します。

See Section 6.2 [ACT], page 94.

`'skk-annotation.el'`

個人辞書に付けたアノテーション (注釈) を活用するプログラムを集めたファイル。

See Section 5.11 [注釈 (アノテーション)], page 77.

`'skk-auto.el'`

送り仮名の自動処理を行うプログラムを集めたファイル。

See Section 5.8.3 [送り仮名の自動処理], page 62.

`'skk-autoloads.el'`

オートロードの設定を記述したファイル。XEmacs で DDSKK をパッケージとしてインストールした場合は `'auto-autoloads.el'` というファイルがこれに相当します。

`'skk-azik.el'`

拡張ローマ字入力 "AZIK" の設定を提供します。

See Section 6.1 [AZIK], page 94.

`'skk-bayesian.el'`

SKK の学習機能のひとつで、ユーザの過去の入力から変換候補を予測します。

<http://openlab.jp/skk/skk/main/bayesian/README.ja>

`'skk-cdb.el'`

CDB 形式辞書を辞書サーバなしに直接利用できるプログラム。

See Section 3.1 [最も基本的な設定], page 7.

`'skk-comp.el'`

見出し語の補完を行うプログラムを集めたファイル。

See Section 5.4 [補完], page 28.

`'skk-cursor.el'`

カーソルの色を制御するプログラムを集めたファイル。

See Section 5.14.3 [入力モードを示すカーソル色に関する設定], page 87.

- ‘skk-cus.el’
M-x customize-group による対話的な設定変更機能の簡易版を提供します。
See Section 5.2.3 [Customize による設定変更], page 24.
- ‘skk-dcomp.el’
skk-comp による補完を自動的に実行して見出し語入力を支援します。
See Section 5.4.3 [動的補完], page 30.
- ‘skk-develop.el’
バグレポートのメールバッファを用意するプログラムファイル。
- ‘skk-emacs.el (DDSKK 14.1 以前のファイル名: skk-e21.el)’
GNU Emacs 21 以降の拡張機能を利用するプログラムを集めたファイル。インジケータのカラー化や画像表示、ツールティップ利用など。
- ‘skk-gadget.el’
プログラム実行変換を行うプログラムを集めたファイル。
See Section 5.5.7 [プログラム実行変換], page 42.
- ‘skk-hint.el’
SKK の変換候補が多いときにヒントを与えて絞りこむ機能を提供します。
See Section 5.5.2 [候補の絞り込み], page 36.
- ‘skk-inline.el’
変換候補のインライン表示機能を集めたファイル。
See Section 5.14.4 [変換候補一覧の表示方法], page 87.
- ‘skk-isearch.el’
DDSKK を併用したインクリメンタル・サーチ機能を提供します。
See Section 5.17 [I-search 関連], page 92.
- ‘skk-jisx0201.el’
いわゆる半角カナを利用する機能を提供します。
- ‘skk-jisx0213.el’
JIS X 0213 文字集合を扱うプログラムです。この機能は Emacs 23 では標準で利用できます。それ未満のバージョンの Emacs においては、Mule-UCS を導入するなどの対策が必要です。Mule-UCS については Mule-UCS Unofficial Site (<http://tats.haun.org/mule-ucs/>) を参照してください。
- ‘skk-jisyo-edit-mode.el’
SKK 辞書を編集するためのメジャーモードを提供します。
- ‘skk-kakasi.el’
KAKASI インターフェイスプログラムを集めたファイル。
See Section 5.3.3 [領域の操作], page 26.
- ‘skk-kanagaki.el’
キーボードのかな配列などに対応する枠組みを提供します。現段階では旧 JIS 配列のかなキーボード及び NICOLA 規格の親指シフト配列に対応しています。
See Section 6.4 [かな入力と親指シフト], page 94.
- ‘skk-kcode.el’
文字コードまたはメニューによる文字入力を行うプログラムを集めたファイル。
See Section 5.12.1 [文字コードまたはメニューによる文字入力], page 80.
- ‘skk-leim.el’
LEIM 関連プログラムファイル。DDSKK を Emacs の input method として利用できるようにします。

- ‘skk-look.el’
look コマンドとのインターフェイスプログラムを集めたファイル。
See Section 5.13.2 [skk-look], page 82.
- ‘skk-lookup.el’
Lookup で検索できる辞書を使って単語の候補を出力するプログラム。
See Section 5.13.1 [skk-lookup], page 82.
- ‘skk-macs.el’
他のファイルで共通して使用するマクロなどを中心にまとめたファイル。
- ‘skk-num.el’
数値変換を行うプログラムを集めたファイル。
See Section 5.5.4 [数値変換], page 39.
- ‘skk-server-completion.el’
拡張された辞書サーバによる見出し語補完機能を利用できます。
- ‘skk-server.el’
辞書サーバと通信して変換する機能を提供します。
See Section 5.10.5 [サーバ関連], page 71.
- ‘skk-setup.el’
自動的に個人設定を行うためのファイル。
- ‘skk-sticky.el’
変換開始位置及び送り開始位置の指定方法を変更可能にする。
See Section 5.6.9 [変換位置の指定方法], page 54.
- ‘skk-study.el’
直前に確定したいくつかの語との関連性を確認し、候補順を操作する学習効果を提供するプログラム。
See Section 5.9.1 [変換の学習], page 65.
- ‘skk-tankan.el’
SKK を使って単漢字変換を行うプログラムです。
See Section 5.5.1 [単漢字変換], page 33.
- ‘skk-tut.el’
SKK チュートリアルプログラム。
See Section 4.5 [チュートリアル], page 18.
- ‘skk-tutcode.el’
SKK で TUT-code 入力を実現します。
See Section 6.3 [TUT-code], page 94.
- ‘skk-version.el’
DDSKK のバージョン情報を提供するプログラムファイル。
- ‘skk-viper.el’
VIPER インターフェイスプログラムを集めたファイル。
See Section 5.16 [VIP/VIPER との併用], page 92.
- ‘skk-xemacs.el’
XEmacs の拡張機能を利用するプログラムを集めたファイル。インジケータのカラー化や画像表示、ツールティップ利用など。

なお、Elib より ‘queue-m.el’ ファイルを収録しています。

5.2 ユーザオプションの設定方法

DDSKK のカスタマイズは、`~/ .emacs` あるいは `~/ .skk` に記述します。また、各ファイルの提供するフックも利用します。上記のファイルやフックを利用した設定がいつ有効になるのか、という点についてここで説明します¹。

5.2.1 設定ファイル

`~/ .emacs`

`~/ .emacs.d/init.el`

`~/ .xemacs/init.el`

Emacs を起動したときに一度だけ読み込まれます。このマニュアルは `~/ .emacs` という記述で統一しています。

See Section “.emacs” in *GNU Emacs Manual*.

`~/ .skk`

DDSKK を起動した最初の一度だけ読み込まれます。ファイル名のデフォルトは、OS の種類により異なりますが、実際は Emacs の関数 `convert-standard-filename` により加工されます。`~/ .skk` のファイル名は変数 `skk-init-file` で変更することができます。また、DDSKK にはこのファイルを自動的にバイトコンパイルする機能があります。

See Section 5.2.1.1 [skk-init-file の自動コンパイル], page 23.

`skk-user-directory`

[ユーザ変数]

DDSKK は、`~/ .skk` や `~/ .skk-jisyo` といった複数のファイルを使用します。これらのファイルをひとつのディレクトリにまとめて置きたい場合は、変数 `skk-user-directory` にそのディレクトリ名を設定します。

この変数のデフォルトは `nil` です。この変数を設定する場合は `~/ .emacs` で行ってください。DDSKK 起動時に `skk-user-directory` が指すディレクトリが存在しない場合は、自動的に作られます。

```
(setq skk-user-directory "~/ .ddskk")
```

この変数を設定した場合（例えば上記 `~/ .ddskk`）、以下に挙げる各変数のデフォルト値が変更されます。

影響を受ける変数	デフォルト値	変更後のデフォルト値
<code>skk-init-file</code>	<code>~/ .skk</code>	<code>~/ .ddskk/init</code>
<code>skk-jisyo</code>	<code>~/ .skk-jisyo</code>	<code>~/ .ddskk/jisyo</code>
<code>skk-backup-jisyo</code>	<code>~/ .skk-jisyo.BAK</code>	<code>~/ .ddskk/jisyo.bak</code>
<code>skk-emacs-id-file</code>	<code>~/ .skk-emacs-id</code>	<code>~/ .ddskk/emacs-id</code>
<code>skk-record-file</code>	<code>~/ .skk-record</code>	<code>~/ .ddskk/record</code>
<code>skk-study-file</code>	<code>~/ .skk-study</code>	<code>~/ .ddskk/study</code>
<code>skk-study-backup-file</code>	<code>~/ .skk-study.BAK</code>	<code>~/ .ddskk/study.bak</code>
<code>skk-bayesian-history-file</code>	<code>~/ .skk-bayesian</code>	<code>~/ .ddskk/bayesian</code>

なお、`skk-user-directory` を設定した場合でも、各変数を個別に設定している場合はその個別の設定が優先されます。

¹ 以前のバージョンでは、どのようなカスタマイズが行われている場合でもチュートリアル使用時に標準設定に戻す目的で、変数の宣言に `defconst` を使用していたので、全てのカスタマイズは `~/ .skk` で行う必要がありましたが、DDSKK 11 以降では、`defcustom` を使用した宣言を行っているため、この制限はなくなりました。

5.2.1.1 skk-init-fileの自動コンパイル

skk-byte-compile-init-file

[ユーザ変数]

ここでは

- 「DDSKK の設定ファイル」を e1 と、
 - 「DDSKK の設定ファイルをバイトコンパイルしたファイル」を e1c と
- それぞれ呼ぶこととします。

DDSKK の起動時に、

- この変数の値が non-nil であれば、
 - e1c が存在しないか、又は
 - e1c よりも e1 が新しいとき
 は、e1 をバイトコンパイルした e1c を生成します。
- この変数の値が nil であれば、
 - e1c よりも e1 が新しいときは、e1c を消去します。

以上の機能を有効にしたい場合は、‘~/ .emacs’ に

```
(setq skk-byte-compile-init-file t)
```

と記述します。²

5.2.2 フック

skk-mode-hook

C-x C-j と入力して SKK モードに入る度に呼ばれます。主にバッファローカルの設定などを行います。

skk-auto-fill-mode-hook

C-x j と入力してオートフィルモード付きで SKK モードに入る度に呼ばれます。主にバッファローカルの設定などを行います。

skk-load-hook

‘skk.el’ の読み込みを完了した時点で呼ばれます。‘~/ .skk’ は SKK モードを起動しなければ読み込まれないのに対し、このフックは、‘skk.el’ を読み込んだら SKK モードを起動しなくとも呼ばれます。

skk-act-load-hook

skk-auto-load-hook

skk-azik-load-hook

skk-comp-load-hook

skk-gadget-load-hook

skk-kakasi-load-hook

skk-kcode-load-hook

skk-num-load-hook

skk-server-load-hook

‘skk-act.el’, ‘skk-auto.el’, ‘skk-azik.el’, ‘skk-comp.el’, ‘skk-gadget.el’, ‘skk-kakasi.el’, ‘skk-kcode.el’, ‘skk-num.el’, ‘skk-server.el’ の各ファイルの読み込みが完了した直後に呼ばれるフック。

load-hook が提供されていないプログラムであっても、ロード完了後に何らかの設定を行いたい場合は、関数 eval-after-load を使用します。例えば、

² この変数は ‘~/ .skk’ が読み込まれる前に調べられるため、‘~/ .skk’ に上記の設定を記述してもこの機能は有効になりません。

```
(eval-after-load "skk-look"
  '(
    ...
  ))
```

のように記述します。

5.2.3 Customize による設定変更

SKK の設定は、Emacs 標準の Customize 機能を使って行うこともできます。ただし、Customize での設定は ‘~/ .emacs’ での設定と同様、‘~/ .skk’ による設定で上書きされてしまいますので注意してください。

M-x customize-group を実行すると skk の設定を対話的に変更することができます。ミニバッファに “Customize group:” とプロンプトが表示されます。

```
----- Minibuffer -----
Customize group: (default emacs) *
----- Minibuffer -----
```

ここで “skk” と答えると、SKK グループの画面へ展開します。

skk で設定できる変数の中には、まだこのマニュアルで記述されていないものもあります。Customize を使うと、それらについても知ることができます。

あるいは、モードラインの SKK インジケータをマウスの右ボタン（第 3 ボタン）でクリックすると表示されるメニューから “SKK をカスタマイズ” を選んでも同じ画面となります。

カスタマイズの使い方は以下を参照してください。

See Section “Easy Customization” in *GNU Emacs Manual*.

M-x customize-group の設定が複雑すぎると感じるユーザのために、簡易版として *M-x skk-customize* を用意しています。これは SKK グループのユーザオプションのうち、よく使うものだけ抜粋して設定できるようにしたものです。

これは、モードラインの SKK インジケータをマウスの右ボタン（第 3 ボタン）でクリックして表示されるメニューから “SKK をカスタマイズ (簡易版)” を選んで呼び出すこともできます。

5.3 カタカナ、英字入力の便法

この節では、カタカナや全英文字を入力するための、便利な方法を説明します。単純に各モードを用いる方法については前述しました。(see Section 4.2 [カナモード、全英モード], page 10)

5.3.1 かなモードからカタカナを入力

まず、かなモードに入ります。q キーでいったん モードにして何かひらがなを入力し、最後に q をタイプすると、カタカナに変換され確定されます。

実際には、ひらがな以外にも変換できます。以下ようになります。

- カタカナはひらがなへ
- ひらがなはカタカナへ
- 全英文字はアスキー文字へ
- アスキー文字は全英文字へ

細かく言えば、‘ ’ とポイント間の文字列の種類³ をキーとして変換が行われます。かなモード、カナモード、どちらでも同じです。

このような変換を、トグル変換と呼びます。以下はトグル変換の例です。

³ 正確には ‘ ’ の次の位置にある文字列によって文字種の判別を行っているので、途中で文字種類の違う文字が混在していても無視されます。

```

K a t a k a n a

----- Buffer: foo -----
  かたかな *
----- Buffer: foo -----

q

----- Buffer: foo -----
  カタカナ *
----- Buffer: foo -----

```

このトグル変換を上手く利用することにより、かなモードのまま一時的にカタカナを入力したり、またその逆を行うことができます。こうすると、例えばひらがな/カタカナ混じり文を書くときに、その都度 `q` キーを押して入力モードを切り換える必要がありません。

全英文字の入力には、次節の方法を用いるのが簡単でしょう。(see Section 5.3.2 [全英文字の入力], page 25)⁴

トグル変換は、領域を対象としたコマンドによって行うこともできます。(see Section 5.3.3 [領域の操作], page 26)

5.3.2 全英文字の入力

まず、かなモードに入ります。次に `/` をタイプすると SKK abbrev モードに入ります。これにより、アルファベット (アスキー文字) を入力できます。なお、SKK abbrev モードでは、

`'is'` ⇒ `'インクリメンタル・サーチ'`

のような変換を行うことができます。他の変換と同様、`SPC` を押すと変換モードに入ってしまうので、SKK abbrev モードからアスキー文字を入力するのは、一語のみの場合以外は不便です。(see Section 5.5.5 [アスキー文字を見出し語とした変換], page 41)

かなモードから、全角アルファベットを入力することもできます。上述のようにアスキー文字を入力し、最後に `C-q`⁵ を入力することで、`'` マークから `C-q` を入力した位置までの文字列が全角アルファベットに変換され確定されます。

```

/ f i l e

----- Buffer: foo -----
  file*
----- Buffer: foo -----

C-q

----- Buffer: foo -----
  f i l e *
----- Buffer: foo -----

```

なお、この変換を行うために、

```
file / f i l e /
```

のような辞書エントリを持つ必要はありません。なぜなら、辞書を参照せず、アスキー文字を 1 文字ずつ全英文字に変換しているからです。

⁴ 全英文字とアスキー文字のトグルでの変換を行うこともできます。ただし、全英モードやアスキーモードでは `q` やその他の大文字によりモードに入ることができないので、かな カナ のときと同様にトグル変換できるわけではありません。かなモード/カナモードにおいて、既に入力された全英文字、アスキー文字に対してトグル変換をするような設計になっています。

⁵ `C-q` は `skk-abbrev-mode-map` にて特別な動作をするように定義されています。See Section 5.5.5 [アスキー文字を見出し語とした変換], page 41.

5.3.3 領域の操作

以下のコマンドを *M-x* により呼ぶことで領域内の文字列を一括変換することができます⁶。

M-x skk-hiragana-region
カタカナをひらがなへ変換。

M-x skk-katakana-region
ひらがなをカタカナへ変換。

M-x skk-latin-region
全英文字をアスキー文字へ変換。

M-x skk-jisx0208-latin-region
アスキー文字を全英文字へ変換。

以下に紹介する「漢字から読みを求めるコマンド」は、外部プログラム KAKASI⁷ が必要です。KAKASI がインストールされていない場合は使用することができません。

M-x skk-gyakubiki-region
漢字をひらがなへ変換。具体的な変換例をあげると、
‘漢字をひらがなへ変換。’ → ‘かんじをひらがなへへんかん。’

のようになります。引数を渡して、

C-u M-x skk-gyakubiki-region

のようにすると、複数の候補がある場合に、‘{ }’ で囲って表示します。例えば

‘中島’ → ‘{なかしま|なかじま}’

のようになります。

送り仮名がある語は、送り仮名まで含めて領域に指定します (さもないと誤変換の原因となります)。例えば、‘五月蠅い’ について、送り仮名 ‘い’ を含めずにこのコマンドを実行すると、‘ごがつはえ’ に変換されてしまいます。

M-x skk-gyakubiki-and-henkan
領域の漢字をひらがなへ変換し、これで得たひらがなを見出し語として漢字変換を実行します。

M-x skk-gyakubiki-katakana-region
漢字をカタカナへ変換。
引数を渡して、*C-u M-x skk-gyakubiki-katakana-region* のようにすると、複数の候補がある場合に、‘{ }’ で囲って表示します。

M-x skk-hurigana-region
漢字にふりがなを付ける。例えば、
‘漢字の脇に’ → ‘漢字 [かんじ] の脇 [わき] に’
のようになります。引数を渡して *C-u M-x skk-hurigana-region* のようにすると、複数の候補がある場合に、‘{ }’ で囲って表示します。

M-x skk-hurigana-katakana-region
漢字にカタカナのふりがなを付ける。
引数を渡して、*C-u M-x skk-hurigana-katakana-region* のようにすると、複数の候補がある場合に、‘{ }’ で囲って表示します。

⁶ メニューバーが使用できる環境では、メニューバーを使ってこれらの一括変換コマンドを呼び出すことができます。ただし *kakasi* がインストールされていない場合は *kakasi* を利用する機能が灰色になり使用できません。See Section “メニューバー” in *GNU Emacs Manual*.

⁷ KAKASI - 漢字 かな (ローマ字) 変換プログラム (<http://kakasi.namazu.org/>)

M-x skk-romaji-region

漢字、ひらがな、カタカナをローマ字へ、全英文字をアスキー文字へ変換。標準では、ローマ字への変換様式はヘボン式です。例えば、

```
‘‘し’’ ↦ ‘‘shi’’
```

となります。変数 *skk-romaji-*-by-hepburn* の値を *nil* に設定すると、訓令式を用います。例えば、

```
‘‘し’’ ↦ ‘‘si’’
```

のようになります⁸。

以下のコマンドは、領域内の文字列を置き換える代わりに、変換結果をエコーエリアに表示します。

- *M-x skk-gyakubiki-message*
- *M-x skk-gyakubiki-katakana-message*
- *M-x skk-hurigana-message*
- *M-x skk-hurigana-katakana-message*
- *M-x skk-romaji-message*

skk-gyakubiki-jisyo-list

[ユーザ変数]

関数 *skk-gyakubiki-region* はコマンド *kakasi* を呼び出しています。*kakasi* には漢字をひらがなへ変換する機能があり、この変換には環境変数 *KANWADICTPATH* で指定されている辞書を用いています。

変数 *skk-gyakubiki-jisyo-list* を設定することにより *kakasi* へ与える辞書を任意に追加することができます。以下のように個人辞書 *skk-jisyo* を指定することにより、辞書登録モードで登録したばかりの単語も *kakasi* による逆引き変換の対象とすることができます。

```
(setq skk-gyakubiki-jisyo-list (list skk-jisyo))
```

5.3.4 カタカナの見出し語

q のタイプでかなモード、カナモードを度々切り替えて入力が続いていると、カナモードで誤ってモードに入ってしまうことがあります。そのため、カナモードでモードに入った場合は、まず見出し語をひらがなに変換してから辞書の検索に入るよう設計されています。なお、この場合の送りあり変換での送り仮名は、カタカナになります。

5.3.5 文脈に応じた自動モード切り替え

‘*context-skk.el*’ は、編集の文脈に応じて自動的に *skk* のモードをアスキーモードに切り替える等の機能を提供します。

‘*context-skk.el*’ をロードするには、‘*~/ .emacs*’ に

```
(add-hook 'skk-load-hook
  (lambda ()
    (require 'context-skk)))
```

と書いてください。

あるプログラミング言語のプログラムを書いているとき、日本語入力の必要があるのは一般に、そのプログラミング言語の文字列中かコメント中に限られます。文字列・コメントの「外」を編集するときは、多くの場合日本語入力は必要ありません。たとえば *emacs lisp* では、「*~*」や「*;; ~*」といった個所でだけ日本語入力が必要となります。

現在の文字列・コメントの「外」で編集開始と同時に (*skk* がオンであれば) *skk* の入力モードをアスキーモードに切り替えます。エコーエリアに

⁸ 昭和 29 年 12 月 9 日付内閣告示第一号によれば、原則的に訓令式 (日本式) を用いるかのように記載されていますが、今日一般的な記載方法は、むしろヘボン式であるようです。

```
----- Echo Area -----
[context-skk] 日本語入力 off
----- Echo Area -----
```

と表示され、アスキーモードに切り替わったことが分かります。これにより、文字列・コメントの「外」での編集を開始するにあたって、日本語入力が on になっていたために発生する入力誤りとその修正操作を回避することができます。

上記の機能は `context-skk-mode` というマイナーモードとして実装されており `M-x context-skk-mode` でオン/オフを制御できます。オンの場合、モードラインのメジャーモード名の隣に「;」と表示されます。

`context-skk` にて「プログラミングモード」とみなすメジャーモードは、変数 `context-skk-programming-mode` で定義されています。

5.4 補完

読みの前半だけを入力して TAB を押せば残りを自動的に補ってくれる、これが補完です。Emacs ユーザにはおなじみの機能が DDSKK でも使えます。

よく使う長い語を効率良く入力するには、アルファベットの略語を登録する方法もあります。(see Section 5.5.5 [アスキー文字を見出し語とした変換], page 41)

5.4.1 読みの補完

モードで TAB を押すと、見出し語（マークから、ポイントまでの文字列）に対する補完が行われます⁹。見出し語補完は、個人辞書の内、送りなしエントリに対して行われます。個人辞書に限っているのは、共有辞書では先頭の文字を共通にする見出し語が多すぎて、望みの補完が行える確率が低いからです。

次の読みの候補を表示するには、. (ピリオド) を、戻る時には、, (コンマ) を押します。その読みで別の語を出すには、いつものように `SPC` を押します。

例を見てみましょう。実際の動作は、個人辞書の内容によって異なります。

```
S a
----- Buffer: foo -----
  さ*
----- Buffer: foo -----

TAB
----- Buffer: foo -----
  さとう*
----- Buffer: foo -----

.
----- Buffer: foo -----
  さいとう*
----- Buffer: foo -----

,
```

⁹ 細かい説明です。TAB を押す直前にモードで入力された文字列を X と呼ぶことにします。このとき、個人辞書の送りなしエントリの中で、先頭が X と一致し長さが X よりも長い見出し語を捜して、そのような語がもしあれば、X の代わりにその語が表示されます。

```
----- Buffer: foo -----
  さとう*
----- Buffer: foo -----
```

SPC

```
----- Buffer: foo -----
  佐藤*
----- Buffer: foo -----
```

C-j

```
----- Buffer: foo -----
  佐藤*
----- Buffer: foo -----
```

補完される見出し語がどのような順で表示されるかと言うと「最近使われた語から」となります。例えば、‘斉藤’、‘佐藤’の順で変換した後、‘さ’をキーにして見出し語の補完を行うと、最初に‘さとう’が、その次に‘さいとう’が補完されます。これは、個人辞書では、最近使われたエントリほど上位に来るようになっているためです。¹⁰

いったん SPC を入力して モードに入ると、以後は見出し語補完は行われません。

また、. の代わりに C-u TAB を入力すると、現在の候補に対して補完をします。上の例では‘さ’に対し、‘さとう’が補完された時に C-u TAB を押すと、以後の補完は、‘さとう’を含む語 (例えば、‘さとうせんせい’など) について行われます。

`skk-comp-circulate` [ユーザ変数]
 . (ピリオド) で次の見出し語候補を、, (コンマ) で前の見出し語候補を表示するところ、候補が尽きていればデフォルト `nil` では「 で補完すべき見出し語は他にありません」とエコーエリアに表示して動作が止まります。この変数が `non-nil` であれば当初の見出し語を再び表示して見出し語補完を再開します。

`skk-try-completion-char` [ユーザ変数]
 見出し語補完を開始するキーキャラクタです。デフォルトは TAB です。

`skk-next-completion-char` [ユーザ変数]
 次の見出し語候補へ移るキーキャラクタです。デフォルトはピリオド . です。

`skk-previous-completion-char` [ユーザ変数]
 前の見出し語候補へ戻るキーキャラクタです。デフォルトはコンマ , です。

`skk-previous-completion-use-backtab` [ユーザ変数]
 Non-nil であれば、前の見出し語候補へ戻る動作を SHIFT + TAB でも可能とします。デフォルトは t です。この機能の有効化 / 無効化の切り替えは、ファイル ‘~/ .skk’ を書き換えて Emacs を再起動してください。

`skk-previous-completion-backtab-key` [ユーザ変数]
 SHIFT + TAB が発行する key event です。Emacs の種類 / 実行環境によって異なります。

5.4.2 補完しながら変換

前節で見出し語の補完について述べました。本節では、見出し語の補完動作を行った後、SPC を入力し、モードに入るまでの動作を一回の操作で行う方法について説明します。

¹⁰ Section 5.10.6 [辞書の書式], page 71

やり方は簡単。TAB・SPC と打鍵していたところを、*M-SPC* に換えると、見出し語を補完した上で変換を開始します。

この方法によると、補完される見出し語があらかじめ分かっている状況では、キー入力を一回分省略できるので、長い見出し語の専門用語を連続して入力する場合などに威力を発揮します。

```
K a s i t a n n p o s e k i n i n n
```

```
----- Buffer: foo -----
```

```
かしたんぼせきにん *
```

```
----- Buffer: foo -----
```

SPC, RET

```
----- Buffer: foo -----
```

```
瑕疵担保責任 *
```

```
----- Buffer: foo -----
```

K a

```
----- Buffer: foo -----
```

```
か *
```

```
----- Buffer: foo -----
```

M-SPC

```
----- Buffer: foo -----
```

```
瑕疵担保責任 *
```

```
----- Buffer: foo -----
```

5.4.3 動的補完

モードでは、TAB を押さなくとも、文字を入力する都度、自動的に見出し語補完の読みを表示させる事ができます。この機能を以下「動的補完」と呼びます。動的補完も、個人辞書の送りなしエントリに対してのみ行なわれます。

類似の機能としては、ブラウザの URL の入力や、Microsoft Excel のセル入力の自動補完（同じ列に既に入力している文字列があったときにそれを参照して補完しようとする機能）をイメージすると分かりやすいかも知れません。

この機能を利用するには、`~/ .skk` に次の式を書きましょう。

```
(setq skk-dcomp-activate t)
```

例を見てみましょう。実際の動作は、個人辞書の内容によって左右されます。* はポイント位置を表します。

H o

```
----- Buffer: foo -----
```

```
ほ * んとう
```

```
----- Buffer: foo -----
```

face が使える環境では、`んとう`の部分のみ異なる face で表示され、動的補完により補完された部分であることを示します。

自動的に補完された見出し語が、自分の意図したものであれば TAB を押すことでポイント位置を動かして、補完された見出し語を選択することができます。

TAB

```
----- Buffer: foo -----
 ほんとう*
----- Buffer: foo -----
```

この状態から SPC を押して変換するなり、q を押してカタカナにするなり、DDSKK 本来の動作を何でも行うことができます。

補完された見出し語が自分の意図したものでない場合は、かまわず次の入力をして下さい。補完された部分を無視したかのように動作します。

H o

```
----- Buffer: foo -----
  ほ*んとう
----- Buffer: foo -----
```

k a

```
----- Buffer: foo -----
  ほか*ん
----- Buffer: foo -----
```

補完されない状態が自分の意図したものである場合も、補完された部分を単に無視するだけで OK です。下記の例では、‘ほ’を見出し語とした変換を行っています。

H o

```
----- Buffer: foo -----
  ほ*んとう
----- Buffer: foo -----
```

SPC

```
----- Buffer: foo -----
  保
----- Buffer: foo -----
```

補完された状態から BS を押すと、消された補完前の見出し語から再度補完動作を行います。

```

H o
----- Buffer: foo -----
  ほ*んとう
----- Buffer: foo -----

k a
----- Buffer: foo -----
  ほか*ん
----- Buffer: foo -----

BS
----- Buffer: foo -----
  ほ*んとう
----- Buffer: foo -----

```

skk-dcomp-activate [ユーザ変数]

この変数の値が Non-nil であれば、カーソル位置に関わらず常に動的補完が有効となります。値がシンボル eolp であれば、カーソルが行末にあるときに限って動的補完が有効となります。値が nil であれば、動的補完機能は無効となります。

skk-dcomp-face [ユーザ変数]

この変数の値はフェイスであり、このフェイスによって動的に補完された部分が装飾されます。標準は “DarkKhaki” です。

skk-dcomp-multiple-activate [ユーザ変数]

Non-nil であれば、動的補完の候補をインラインに複数表示します¹¹。XEmacs では動作しません。

```

----- Buffer: foo -----
  ほ*んとう
  ほんとう
  ほかん
  ほっかいどう
  ほうほう
  ...
----- Buffer: foo -----

```

候補の選択には TAB 又は SHIFT + TAB を押します。また、普通の補完と同様に . (ピリオド) と , (コンマ) も利用できます。Section 5.4.1 [読みの補完], page 28

skk-dcomp-multiple-rows [ユーザ変数]

動的補完の候補を複数表示する場合の表示行数。標準は 7。

skk-dcomp-multiple-face [ユーザ変数]

動的補完の複数表示群のフェイス。上記例では「ほ」のフェイス。

skk-dcomp-multiple-trailing-face [ユーザ変数]

動的補完の複数表示群の補完部分のフェイス。上記例では「んとう」、「かん」、「っかいどう」、「うほう」のフェイス。

¹¹ 現在は候補群の右側 1 カラムのフェイスがデフォルトに戻る、という制約があります。

skk-dcomp-multiple-selected-face [ユーザ変数]
 動的補完の複数表示郡の選択対象のフェイス。上記例では TAB を押すたびに「ほんとう」、「ほかん」、「ほっかいどう」と選択位置が移ります。その現在選択位置に適用するフェイスです。

5.5 便利な変換、その他の変換

5.5.1 単漢字変換

ファイル 'skk-tankan.el' を読み込むことによって単漢字変換が可能となります。候補は総画数の昇順でソートして表示します。

単漢字変換を使うには設定が必要ですが、先に例を見てみましょう。

モードの最後の文字に @ を付加してから変換を開始してください。

T a n @

```
----- Buffer: foo -----
  たん@*
----- Buffer: foo -----
```

SPC

```
----- Buffer: foo -----
  丹*
----- Buffer: foo -----
```

```
----- Echo Area -----
4 画 ( 部 3 画)
----- Echo Area -----
```

SPC

```
----- Buffer: foo -----
  反*
----- Buffer: foo -----
```

```
----- Echo Area -----
4 画 (又部 2 画)
----- Echo Area -----
```

SPC

```
----- Buffer: foo -----
  旦*
----- Buffer: foo -----
```

```
----- Echo Area -----
5 画 (日部 1 画)
----- Echo Area -----
```

SPC

```

----- Buffer: foo -----
  但 *
----- Buffer: foo -----

----- Echo Area -----
7画(人部5画)
----- Echo Area -----

SPC

----- Buffer: foo -----
  *
----- Buffer: foo -----

----- Buffer: *候補* -----
A: 坦;8画(土部5画)
S: 担;8画(手部5画)
D: 単;9画(十部7画)
F: 彖;9画(彡部6画)
J: 炭;9画(火部5画)
K: 眈;9画(目部4画)
L: 胆;9画(肉部5画)
[残り 50+++++]
----- Buffer: *候補* -----

```

以上のとおり、総画数の昇順でソートされた候補が次々に表示されます。

5.5.1.1 検索キーの設定

デフォルトの検索キーは @ です。DDSKK の標準設定ではキー @ は関数 `skk-today` の実行に割り当てられていますが、DDSKK 14.2 からは特段の設定なしにモードで @ のタイプが可能となりました。

`skk-tankan-search-key` [ユーザ変数]
 単漢字変換の検索キーは、変数 `skk-tankan-search-key` で変更できます。以下は、検索キーを ! へと変更する例です。

```
(setq skk-tankan-search-key ?!)
```

5.5.1.2 辞書の設定

DDSKK 14.2 からは、標準で変数 `skk-search-prog-list` に `skk-tankan-search` が含まれています。DDSKK 14.1 を利用の方、ご自身で `skk-search-prog-list` を設定する方は以下の解説を参考にしてください。

‘`skk-tankan.el`’ には、漢字の部首と其中での画数のデータのみが入っています。読みのデータは、普通の辞書ファイルを使います。

単漢字変換の辞書の設定は、変数 `skk-search-prog-list` に以下の形式で要素を追加します。

```
(skk-tankan-search 'function . args)
```

「確定変換」を併用する場合は、`skk-search-prog-list` の先頭の要素は `skk-search-kakutei-jisyo-file` でなければいけませんので、`skk-search-prog-list` の 2 番目の要素に `skk-tankan-search` を追加します。

```
;; skk-search-prog-list の2番目の要素に skk-tankan-search を追加する
(setq skk-search-prog-list
  (cons (car skk-search-prog-list)
        (cons '(skk-tankan-search 'skk-search-jisyo-file
                    skk-large-jisyo 10000)
              (cdr skk-search-prog-list))))
```

なお、「確定変換」を使用しない場合は、skk-search-prog-list の要素の先頭が skk-tankan-search でも大丈夫です。

```
(add-to-list 'skk-search-prog-list
  '(skk-tankan-search 'skk-search-jisyo-file
    skk-large-jisyo 10000))
```

See Section 5.10.3 [辞書の検索方法の設定], page 69.

5.5.1.3 総画数による単漢字変換

モードで総画数を入力して最後に @ を付加してから変換を開始します¹²。

```
Q 1 0 @

----- Buffer: foo -----
  10@*
----- Buffer: foo -----

SPC

----- Buffer: *候補* -----
A: 俚;10画(人部8画)
S: 倦;10画(人部8画)
D: 個;10画(人部8画)
F: 候;10画(人部8画)
J: 倅;10画(人部8画)
K: 借;10画(人部8画)
L: 修;10画(人部8画)
[残り] 532+++++++
----- Buffer: *候補* -----
```

5.5.1.4 部首による単漢字変換

モードで @ を2つ重ねて変換を開始すると、部首による単漢字変換ができます¹³。

```
Q @ @

----- Buffer: foo -----
  @@*
----- Buffer: foo -----

SPC

----- Minibuffer -----
部首を番号で選択(TABで一覧表示): *
----- Minibuffer -----
```

¹² C-u 総画数 M-x skk-tankan でも可。

¹³ M-x skk-tankan でも可。

TAB

----- Completion List -----

Click <mouse-2> on a completion to select it.

In this buffer, type RET to select the completion near point.

Possible completions are:

001 一 002 二 003 丶 004 丿 005 乙 006 丨 007 二 008 亅 009 人
010 儿 011 入 012 八 013 冂 014 冂 015 丩 016 几 017 凵 018 刀
...

----- Completion List -----

0 1 8 RET¹⁴

----- Buffer: *候補* -----

A:切;4画(刀部2画)

S:刈;4画(刀部2画)

D:刊;5画(刀部3画)

F:刊;5画(刀部3画)

J:刎;6画(刀部4画)

K:刑;6画(刀部4画)

L:刳;6画(刀部4画)

[残り 51+++++++]

----- Buffer: *候補* -----

skk-tankan-face

[ユーザ変数]

M-x skk-tankan を実行したときに表示される「単漢字バッファ」で使用するフェイスです。

5.5.2 候補の絞り込み

‘skk-hint.el’ は、2つの読みの積集合みたいなものを取ることによって候補の絞り込みを行うプログラムです。インストールは ‘~/skk’ に以下を記入します。

(require 'skk-hint)

例えば、読み“かんだう”に対する変換は L 辞書によると

感動、勘当、完動、問道、官道、貫道

と複数の候補があります。

一方、これに“あいだ”という「他の読み」(ヒント)を与えると候補は“問道”に一意に決まります。ヒントは ; に続けて入力します。

K a n d o u ; a i d a

; 自体は表示されません。

----- Buffer: foo -----

かんだうあいだ

----- Buffer: foo -----

SPC

¹⁴ M-v の打鍵で、カーソルを Completion List へ移すこともできます。

```
----- Buffer: foo -----
```

```
  間道
```

```
----- Buffer: foo -----
```

‘skk-hint.el’ は、2つの読みの厳密な積集合を取っているわけではなく、通常の変換候補のなかでヒントとして与えられた読みを含んだ漢字を持つものに候補を絞ります。この実例として“感動”と“感圧”を挙げます。

```
K a n d o u ; k a n n a t u
```

```
----- Buffer: foo -----
```

```
  かんどうかんあつ
```

```
----- Buffer: foo -----
```

SPC

```
----- Buffer: foo -----
```

```
  感動
```

```
----- Buffer: foo -----
```

‘skk-hint.el’ は単漢字の候補がたくさんある場合に、そこから候補を絞りこむ手段としても非常に有効です。例えば

```
  わ
```

を変換すると、輪、環、話、和、羽、...と大量に候補が出てきます。この中から“和”を選びたいとします。普通に変換していてもそのうち“和”が表示されますが、これを *Wa ; heiwa* と入力し変換すると、「へいわ」の候補である「平和」に含まれる

```
  和
```

が唯一の候補となります。

```
W a ; h e i w a
```

```
----- Buffer: foo -----
```

```
  わへいわ
```

```
----- Buffer: foo -----
```

SPC

```
----- Buffer: foo -----
```

```
  和
```

```
----- Buffer: foo -----
```

5.5.3 接頭辞・接尾辞

接頭辞 (prefix)、接尾辞 (suffix) の入力のために特別な方法が用意されています。たとえば、「し」の候補は沢山あり、「し」から「氏」を変換するのは、そのままでは効率が悪いです。接尾辞の「し」ならば、「氏」や「市」が優先されるでしょう。

接頭辞・接尾辞は辞書の中では、‘>’などで示されます。

```
>し /氏/
```

というエントリがあるとき、‘小林氏’を接尾辞入力を用いて、以下のように入力することができます。

```
K o b a y a s h i
```

```
----- Buffer: foo -----
```

```
  こばやし*
```

```
----- Buffer: foo -----
```

SPC

```
----- Buffer: foo -----
  小林 *
```

>

```
----- Buffer: foo -----
  小林 >*
```

s i

```
----- Buffer: foo -----
  小林 >し *
```

SPC

```
----- Buffer: foo -----
  小林 氏 *
```

C-j

```
----- Buffer: foo -----
  小林氏 *
```

接頭辞も同様です。辞書に

ちょう> /超/

というエントリがあるとき、‘超大型’を接頭辞入力を用いて、以下のように入力することができます。

T y o u

```
----- Buffer: foo -----
  ちょう *
```

>

```
----- Buffer: foo -----
  超 *
```

O o g a t a

```
----- Buffer: foo -----
  超 おおがた *
```



```
SPC
----- Buffer: foo -----
超 大型*
----- Buffer: foo -----
```

```
C-j
----- Buffer: foo -----
超大型*
----- Buffer: foo -----
```

キー > を押しただけで、SPC が押されたかのように変換されます。他の接頭辞を選びたいときは、SPC を押して下さい。

skk-special-midashi-char-list [ユーザ変数]

モードまたは モードにおいて、この変数の値に含まれる文字の入力があった場合、接頭辞・接尾辞の入力を開始します。この変数のデフォルトは、

```
(?> ?< ??)
```

です。つまり、> と < と ? を入力した時に接頭辞・接尾辞入力を行います。? を入力したときに接頭辞・接尾辞入力を行わない場合は ? を外して

```
(setq skk-special-midashi-char-list '(?> ?<))
```

とします。

L 辞書の接頭・接尾辞は、昔は <, ? も使われていましたが、現在は > に統一されています。

5.5.4 数値変換

DDSKK は、'だい12かい' を変換すると '第12回'、'第一二回'、'第十二回' などの候補を挙げるなど数字を含む見出し語を様々に変換できます。

この節では、このような候補を辞書に登録する方法を説明します。基本は、数字の部分を '#' で置き換えることです。SKK の L 辞書から、具体例を見てみましょう。

```
だい#かい /第#1回/第#0回/第#2回/第 #0回/第#3回/
```

'だい12かい' のような見出し語で変換しようとした場合、見出し語の数字の部分は自動的に '#' に置き換えられるようになっているので、上記の例にマッチします。

右辺の '#1'、'#2' などは「どのように数字を加工するか」のタイプを表します。以下、各タイプについて説明します。

'#0'

タイプ 0。無変換。入力されたアスキー文字をそのまま出力します。例えば、'第12回' のような変換を得るために使われます。

'#1'

タイプ 1。全角文字での数字。'12' を '12' に変換します。

'#2'

タイプ 2。漢数字で位取りあり。'5500' を '五五〇〇' に変換します。

'#3'

タイプ 3。漢数字で位取りなし。'5500' を '五千五百' に変換します。

'#4'

タイプ 4. 数値再変換。見出し語中の数字そのもの¹⁵ をキーとして辞書を再検索し、‘#4’ の部分を再検索の結果の文字列で入れ替えます。これについては後で例を挙げて説明します。

‘#5’

タイプ 5. 小切手や手形の金額記入の際用いられる表記で変換します。例えば、‘1995’ を ‘壹仟九百九拾伍’ に変換します。(これを大字と言います。)

‘#9’

タイプ 9. 将棋の棋譜の入力用。‘全角数字 + 漢数字’ に変換します。これについては後で例を挙げて説明します。

以下にいくつか例を示します。辞書に

/#3/

というエントリがあるときに、

*Q 1 0 0 2 0 0 3 0 0 4 0 0 5 0 0 SPC*¹⁶

と入力すれば、‘百兆二千三億四十万五百’ と変換されます¹⁷。

辞書に

#m#d /#0 月#0 日/

というエントリがあるときに、*/ 2 m 2 5 d SPC* と入力すれば、‘2 月 25 日’ と変換されます¹⁸。

辞書に

#kin /#9 金/

というエントリがあるときに、*/ 3 4 k i n SPC* と入力すれば、‘3 四金’ と変換されます。

辞書に

p# /#4/

125 /東京都葛飾区/

というエントリがあるときに、*/ p 1 2 5 SPC* と入力すれば、見出し語 ‘p125’ の候補が ‘#4’ なので、見出し語の数字部分の ‘125’ に対し辞書が再検索され、‘東京都葛飾区’ と変換されます。

最後に、実際に登録する例を 1 つ挙げます。‘2 月 2 5 日’ を得るために、

Q 2 g a t u 2 5 n i t i SPC

と入力したときに、辞書に見出し語

#がつ#にち /#1 月#1 日/

がないときは、辞書登録時のプロンプトは、‘#がつ#にち’ となります。全角数字のタイプは、‘#1’ なので、‘#1 月#1 日’ をミニバッファで作り登録します。

タイプを覚えている必要はありません。ちゃんと、ウィンドウが開かれて説明が表示されます。

`skk-num-convert-float`

[ユーザ変数]

この変数の値を `non-nil` に設定すると、浮動小数点数を使った見出し語に対応して数値変換を行います。ただし、辞書において

#.# /#1 . #1/#0 月#0 日/

などの見出し語が使用できなくなります。

`skk-use-numeric-conversion`

[ユーザ変数]

この変数を `nil` に設定すると、本節で説明した数値変換の機能を全て無効にします。

¹⁵ ‘p125’ という見出し語であれば、その数値部分である ‘125’ が再変換の見出し語となります。

¹⁶ または */ 1 0 0 2 0 0 3 0 0 4 0 0 5 0 0 SPC*。

¹⁷ 数字を大文字で入力し始めることはできないので、*q* または */* で モードに入る必要があります。

¹⁸ ‘m’ や ‘d’ などアスキー文字を見出し語として入力する場合は */* キーを最初に入力して `SKK abbrev` モードに入ってから入力する必要があります。See Section 5.5.5 [SKK abbrev mode], page 41.

5.5.5 アスキー文字を見出し語とした変換

かなモードで / をタイプすると *SKK abbrev mode* に入り、以後の入力はアスキー文字になります。普通に SPC を押すと、その見出し語に係る変換が得られます。

仮に、辞書に

```
is /インクリメンタル・サーチ/
```

というエントリがあるとして、以下に例を示します。

```
/
```

```
----- Buffer: foo -----
```

```
  *
```

```
----- Buffer: foo -----
```

```
i s
```

```
----- Buffer: foo -----
```

```
  is*
```

```
----- Buffer: foo -----
```

```
SPC
```

```
----- Buffer: foo -----
```

```
  インクリメンタル・サーチ *
```

```
----- Buffer: foo -----
```

```
C-j
```

```
----- Buffer: foo -----
```

```
  インクリメンタル・サーチ *
```

```
----- Buffer: foo -----
```

変換が終わると *SKK abbrev* モードを抜けてかなモードに戻ります。

SKK abbrev モードで使われる辞書は、普通の変換と同じです。エントリがアスキー文字で書かれているだけで、特殊な点はありません。

入力したアスキー文字をそのまま全角アルファベットに変換する方法については、Section 5.3.2 [全英文字の入力], page 25 を参照してください。

なお、*SKK abbrev* モードにおいても見出し語の補完を行うことができます。(see Section 5.4 [補完], page 28)

5.5.6 今日の日付の入力

かな/カナモードで @ を入力すれば、今日の日付が入力されます。

日付の形式は以下の変数により決定されます。

`skk-date-ad` [ユーザ変数]

この変数の値が `non-nil` であれば西暦で、`nil` であれば元号で表示します。デフォルトは `nil` です。

`skk-number-style` [ユーザ変数]

この変数の値は以下のように解釈されます。

0

`nil`

半角数字。‘1996年7月21日(日)’ のようになります。

1
t

全角数字。‘1996年7月21日(日)’のようになります。

2

漢数字。‘一九九六年七月二日(日)’のようになります。

3

漢数字。‘千九百九十六年七月二十一日(日)’のようになります。

上記の‘1996年’、‘1996年’、‘一九九六年’の部分は、変数 `skk-date-ad` の値が `nil` であれば‘平成8年’のように元号で表示されます。 `skk-number-style` のデフォルトは、‘1’です。

L 辞書¹⁹には、見出し語‘today’の候補として、`skk-date-ad` と `skk-number-style` の全ての組み合わせがプログラム実行変換機能²⁰を用いて登録されています。従って、`/today SPC` と入力すると、今日の日付が上記の形式で順次候補として表示されます。

@によって日付を挿入するのではなく、文字通り@を挿入したい場合には、

```
(setq skk-rom-kana-rule-list
      (append skk-rom-kana-rule-list
              '(("@" nil "@"))))
```

のようにします。全角文字の‘@’を挿入したい場合は、

```
(setq skk-rom-kana-rule-list
      (append skk-rom-kana-rule-list
              '(("@" nil "@"))))
```

のようにします。

5.5.7 プログラム実行変換

辞書の候補に Emacs Lisp のプログラムが書いてあれば、そのプログラムを Emacs に実行させ、戻り値をカレントバッファに挿入します。これを「プログラム実行変換」と呼んでいます。例えば、辞書に

```
now /(current-time-string)/
```

というエントリがあるとします。このとき `/now SPC` と入力すれば、現在のバッファに `current-time-string` の戻り値である

```
Sun Jul 21 06:40:34 1996
```

のような文字列が挿入されます。

ここで、プログラムの戻り値は文字列である必要があります。また、プログラム実行変換の辞書登録は通常の単語と同様に行うことができますが、その中に改行を含まないように書く必要があります²¹。

今日の日付の入力²²で説明した‘today’の辞書エントリは、実際は下記のようなプログラムを候補にもっています。

```
today /(let ((skk-date-ad) (skk-number-style t)) (skk-today))/.../
```

‘skk-gadget.el’には、西暦/元号変換プログラムや、簡単な計算プログラムなど、実行変換用のプログラムが集められています。以下、2つの例を掲げます。

¹⁹ Section 5.10.1 [L 辞書], page 67.

²⁰ Section 5.5.7 [プログラム実行変換], page 42.

²¹ 通常の単語では、改行を含むことが可能です。それは、評価するとその位置に改行を挿入するような実行変換プログラムに変換して辞書に書き込んでいるからです。See Section 5.10.1 [辞書の種類], page 67.

しかし、実行変換されるプログラムを辞書登録するにはこの機能を利用できないため、改行を含むことができません。

²² See Section 5.5.6 [今日の日付の入力], page 41.

skk-calc operator [Function]

skk-calc は、引数を 1 つ取り、見出し語の数字に対しその演算を行う簡単な計算プログラムです。

```
(defun skk-calc (operator)
  ;;2つの引数を取って operator の計算をする。
  ;;注意: '/' は引数として渡せないので (defalias 'div '/') などとし、別の形で
  ;;skk-calc に渡す。
  ;;辞書見出し例; #*# /(skk-calc '*)/
  (number-to-string (apply operator
                            (mapcar 'string-to-number
                                    skk-num-list))))
```

この関数を実際にプログラム実行変換で利用するには、辞書に以下のようなエントリを追加します²³。

```
### /(skk-calc '*)/
```

Q 1 1 1 * 4 5 SPC と入力します。ここで、'111' と '45' の 2 つの数字は、変換時に ("111" "45") のような文字列のリストにまとめられ、変数 skk-num-list の値として保存されます。次に関数 skk-calc が呼ばれます。この中で、skk-num-list の各要素に対し演算を行うため、各要素は数に変換されます。その上で、skk-calc に与えられた引数 (この場合は '*') を演算子として演算を行います。

skk-gadget-units-conversion 基準単位 数値 変換単位 [Function]

数値について、基準単位から変換単位への変換を行います。

```
/ 1 3 m i l e

----- Buffer: foo -----
13mile*
----- Buffer: foo -----

SPC

----- Buffer: foo -----
20.9209km*
----- Buffer: foo -----

RET

----- Buffer: foo -----
20.9209km*
----- Buffer: foo -----
```

単位変換の情報は、変数 skk-units-alist で定義されています。

skk-units-alist [ユーザ変数]

この変数は以下の形式の連想リストです。

```
(基準となる単位 (変換する単位 . 変換時の倍率)
 (... . ...))
```

関数 skk-gadget-units-conversion で利用されています。デフォルトでは、以下の単位変換の情報を定義しています。

²³ Section 5.5.4 [数値変換], page 39.

```
("mile" ("km" . 1.6093)
      ("yard" . 1760))
```

```
("yard" ("feet" . 3)
      ("cm" . 91.44))
```

```
("feet" ("inch" . 12)
      ("cm" . 30.48))
```

```
("inch" ("feet" . 0.5)
      ("cm" . 2.54))
```

5.5.8 空白・改行・タブを含んだ見出し語の変換

変換の際、見出し語の中の空白、改行、タブは無視されます。

```
----- Buffer: foo -----
   じんじょうしょ
   うがっこう*
----- Buffer: foo -----
```

SPC

```
----- Buffer: foo -----
   尋常小学校*
----- Buffer: foo -----
```

オートフィルモードで折り返された文字列に対し、折り返された状態のまま変換を行うこともできます。

```
----- Buffer: foo -----
   仮名漢字変換プログラムをさ
   くせいしました。*
----- Buffer: foo -----
```

C-u 10 C-b Q

```
----- Buffer: foo -----
   仮名漢字変換プログラムを  *さ
   くせいしました。
----- Buffer: foo -----
```

C-u 5 C-f

```
----- Buffer: foo -----
   仮名漢字変換プログラムを さ
   くせい*しました。
----- Buffer: foo -----
```

SPC

```
----- Buffer: foo -----
   仮名漢字変換プログラムを 作成*しました。
----- Buffer: foo -----
```

ここでは改行を越えて見出し語を探し、変換する例を示しました。同様に、空白、タブ文字を中間に含む文字列に対しても変換を行うことができます。

`skk-allow-spaces-newlines-and-tabs` [ユーザ変数]
この変数を `nil` に設定すると、本節で説明したような 2 行以上にまたがる文字列に対する変換を禁止します。

5.5.9 カタカナ変換

`skk-search-katakana` [ユーザ変数]
通常、SKK でカタカナ語を入力するには、`q` でカナモードに移ってからカタカナを入力するか、`Q` モードで `q` によりカタカナへ変換するか、です。これらの方法は手軽ですが、個人辞書に登録されないため補完候補にも現れず、何度でも入力しなければいけません。

しかし、ここに紹介する方法ではカタカナ語が普通の変換候補として現れるため、個人辞書に登録することができます。設定するには以下を `'~/ .skk'` に記述します。²⁴

```
(setq skk-search-katakana t)
```

また、値を以下のようにするとカタカナ候補に加え半角カタカナ候補も変換候補に現れます。

```
(setq skk-search-katakana 'jisx0201-kana)
```

5.5.10 サ変動詞変換

`skk-search-sagyo-henkaku` [ユーザ変数]
通常、SKK では諸般の事情によりサ行変格活用の動詞は送りなし変換をする前提になっています。このことは共有辞書のメンテナンスにおける便宜上やむをえないのですが、個人辞書が育たない（サ変動詞と名詞の区別ができない）という弱点もあります。(see Section 4.3.4.3 [サ変動詞の辞書登録に関する注意], page 16)

しかし、ここに紹介する方法では任意の送りなし候補を利用してサ行の送りプレフィックスに限定して送りあり変換が可能になり、個人辞書を育てることが可能になります。設定するには以下を `'~/ .skk'` に記述します。²⁵

```
(setq skk-search-sagyo-henkaku t)
```

例えば‘お茶する’の変換は以下のように変化します。

- 従来 ... `O c h a S P C s u r u`
- サ変 ... `O c h a S u r u`

変数の値を `anything` に設定すると、サ行に限らず任意の送り仮名を許可し、送りあり変換をします。これにより、送りあり変換の利用範囲を形容詞・動詞の変換のみならず、あらゆるひらがな開始点の指定に拡張することができます。

このサ変動詞送りあり変換機能は、カタカナ変換機能と組み合わせるとさらに有効です。(see Section 5.5.9 [カタカナ変換], page 45)

²⁴ `skk-search-prog-list` の設定をユーザが変更している場合は期待通りに動作しない場合があります。その場合は `skk-search-prog-list` の設定に関数 `skk-search-katakana` の呼び出しがあることを確認してください。またこの機能の設定は DDSKK 14.1 以前では異なります。詳しくはソースに付属のドキュメント、設定例をご覧ください。

²⁵ `skk-search-prog-list` の設定をユーザが変更している場合は期待通りに動作しない場合があります。その場合は `skk-search-prog-list` の設定に関数 `skk-search-sagyo-henkaku` の呼び出しがあることを確認してください。またこの機能の設定は DDSKK 14.1 以前では異なります。詳しくはソースに付属のドキュメント、設定例をご覧ください。

5.5.11 ファンクションキーの使い方

skk-j-mode-function-key-usage [ユーザ変数]

シンボル `conversion` ならば、`skk-search-prog-list-1` ~ `skk-search-prog-list-9`, `skk-search-prog-list-0` を実行するよう自動設定します。これらのプログラムはモード限定でファンクションキー ([F1] ~ [F10]) に割り当てられます。[F5] ~ [F10] については本オプションの設定により自動的に割り当てられます。これらの割り当てはユーザオプション `skk-verbose` を設定するとエコーエリアに表示されるようになります。(see Section 5.15.2 [助言的案内メッセージの表示], page 91)

- [F5] ... 単漢字
- [F6] ... 無変換
- [F7] ... カタカナ
- [F8] ... 半角カナ
- [F9] ... 全角ローマ
- [F10] ... ローマ

一方シンボル `kanagaki` に設定すると、かなキーボード入力用に自動設定します。`nil` ならば、自動設定しません。

5.6 キー設定

関連項目: Chapter 6 [ローマ字入力以外の入力方式], page 94

5.6.1 かなモード/カナモードのキー設定

5.6.1.1 ローマ字のルールの設定

`skk-rom-kana-base-rule-list`
`skk-rom-kana-rule-list`

DDSKK のモードにおける文字変換は、これら2つの変数を用いて行われます。`skk-rom-kana-base-rule-list` には基本的なローマ字かな変換のルールが定められています。一方、`skk-rom-kana-rule-list` はユーザが独自の規則を定めるために用意されており、`skk-rom-kana-base-rule-list` より優先されます。

これらは「入出力の状態がいかに移り変わるべきか」を決定します。その内容は、

(入力される文字列 出力後に自動的に入力に追加される文字列 出力)

という形のリストを列挙したものです。

‘入力される文字列’とは変換される前のアスキー文字の文字列をいいます。

‘出力’は次の入力状態に移るときにバッファに挿入される文字列の組み合わせであり、("ア" . "あ") のようなコンスセルです。

`skk-rom-kana-base-rule-list` の一部を見てみましょう。

```
("a" nil ("ア" . "あ"))
("ki" nil ("キ" . "き"))
("tt" "t" ("ッ" . "っ"))
("nn" nil ("ン" . "ん"))
("n'" nil ("ン" . "ん"))
```

のような規則があります。これによると

```
a ↦ あ
ki ↦ き
tt ↦ っ t
nn ↦ ん
```


n' ↦ん

のようになります。

5.6.1.2 ローマ字ルールの変更例

skk-rom-kana-base-rule-list の規則に従うと

hannou ↦はんおう
han'ou ↦はんおう
hannnou ↦はんのう

のようになります。ここで

```
(setq skk-rom-kana-rule-list
      (append skk-rom-kana-rule-list
              '(("nn" "n" ("ン" . "ん")))))
```

のような設定にすることで

hannou ↦はんのう

のようにローマ字かな変換が行われるようになります。

他の例として、略号を設定することもできます。

tp ↦東北大学
skk ↦skk
skK ↦SKK

といった変換は、

```
("tp" nil ("東北大学" . "東北大学"))
("sk" nil (" " . " "))
("skk" nil ("skk" . "skk"))
("skK" nil ("SKK" . "SKK"))
```

のような規則を追加することで実現されます。自分の名前を入力することはよくあるので、適当な省略形を用いて、このリストに追加しておく、といった利用をお勧めします。

更に skk-rom-kana-rule-list を用いれば TUT-code による日本語入力を実現することもできます。例えば TUT-code による入力についてはソースアーカイブの 'tut-code' ディレクトリに収録されているソースコードを参照してください。(see Chapter 6 [ローマ字入力以外の入力方式], page 94)

5.6.1.3 モードに関連するその他の変数

skk-kana-input-search-function [ユーザ変数]

ルールリストの中に記せない変換ルールを処理する関数。これは skk-rom-kana-base-rule-list と skk-rom-kana-rule-list の要素を全て検索した後にコールされます。引数はありません。パツファの文字を、直接 preceding-char など調べて下さい。

初期設定では h で、長音を表すために使われています。次の例を見て下さい。

ohsaka ↦ おおさか
ohta ↦ おおた

一方で、hh は「っ」になります。

ohhonn ↦ おっほん
ohhira ↦ おっひら

これは skk-rom-kana-rule-list のデフォルトに

```
("hh" "h" ("ッ" . "っ"))
```

が入っているためです。これを削除すれば

ohhonn ↦ おおほん
ohhira ↦ おおひら

となります。

skk-kutouten-type [ユーザ変数]

モードの標準では、キーボードの . をタイプすると「。」が、, をタイプすると「、」がバッファに入力されます。変数 skk-kutouten-type に適切なシンボルを設定することにより、この組み合わせを変更することができます²⁶。そのシンボルとは、次の4つです。

```
'jp    ↪ 「。」「、」 (デフォルト)
'en    ↪ 「。」「、」
'jp-en ↪ 「。」「、」
'en-jp ↪ 「。」「、」
```

または、変数 skk-kutouten-type にはコンセルを指定することも可能です。その場合は、

```
(句点を示す文字列 . 読点を示す文字列)
```

のように指定します。例として、次のように設定するとキーボードの . で abc が、, で def がバッファに入力されます。

```
(setq skk-kutouten-type '("abc" . "def"))
```

なお、変数 skk-kutouten-type はバッファローカル変数です。すべてのバッファで統一した設定としたい場合は、

```
(setq-default skk-kutouten-type 'en)
```

のように関数setq-defaultを用いてください。

5.6.1.4 数字や記号文字の入力

かなモード/カナモードにおける次のキーは、かつては skk-self-insert 関数にバインドされていました。現在は他のキーと同様に skk-insert 関数にバインドされています。

```
! # % & ' * +
- 0 1 2 3 4 5
6 7 8 9 : ; <
= > ? " ( ) [
] { } ^ _ ` |
~
```

これらの数字や記号文字のキーに対応し挿入される文字は、変数 skk-input-vector の値により決定されていましたが、DDSKK 10 にてこの変数は廃止されました。カスタマイズするためには、代わりに変数 skk-rom-kana-rule-list を利用します。例えば、SKK 9.6 以前において、

```
(aset skk-input-vector ?! nil)
(aset skk-input-vector ?, nil)
(aset skk-input-vector ?. nil)
(aset skk-input-vector ?: nil)
(aset skk-input-vector ?; nil)
(aset skk-input-vector ?? nil)
```

このような設定をしていた場合、DDSKK 10.x 以降で同様の入力をするためには、

```
(setq skk-rom-kana-rule-list
      (append skk-rom-kana-rule-list
              '(("!" nil "!")
                (", " nil ",")
                (". " nil ".")
```

²⁶ 変数 skk-use-kana-keyboard が non-nil ならば無効である。

```
( ":" nil ":" )
( ";" nil ";" )
( "?" nil "?" )))
```

のような設定に変更する必要があります。

skk-insert は、Emacs のオリジナル関数 self-insert-command をエミュレートしています。具体的には、引数を渡すことによって同じ文字を複数、一度に挿入することが可能です²⁷。

```
C-u 2 !

----- Buffer: foo -----
!!
----- Buffer: foo -----
```

5.6.2 全英モードのキー設定

全英モードにおける印字可能な全てのキーはコマンド skk-jisx0208-latin-insert に割り付けられています。また、変数 skk-jisx0208-latin-vector の値により挿入される文字が決定されます。skk-jisx0208-latin-vector のデフォルトは以下のようになっています。

```
[nil nil nil nil nil nil nil nil
 nil nil nil nil nil nil nil nil
 nil nil nil nil nil nil nil nil
 nil nil nil nil nil nil nil nil
 " " "!" " " " " "#" "$" "%" "&" " '"
 "(" ")" "*" "+" " ," - " ." "/"
 "0" "1" "2" "3" "4" "5" "6" "7"
 "8" "9" ":" ";" "<" "=" ">" "?"
 "@" "A" "B" "C" "D" "E" "F" "G"
 "H" "I" "J" "K" "L" "M" "N" "O"
 "P" "Q" "R" "S" "T" "U" "V" "W"
 "X" "Y" "Z" "[" "\ " "]" "^" "_"
 " " "a" "b" "c" "d" "e" "f" "g"
 "h" "i" "j" "k" "l" "m" "n" "o"
 "p" "q" "r" "s" "t" "u" "v" "w"
 "x" "y" "z" "{" "|" "}" "~" nil]
```

挿入される文字の変更方法を知りたい場合は、Section 5.6.1.4 [数字や記号文字の入力], page 48 を参照してください。

skk-jisx0208-latin-insert も Emacs オリジナルの関数 self-insert-command をエミュレートしています。つまり、引数を渡すことにより同じ文字を複数、一度に挿入することができます。skk-insert における動作と同じですから、Section 5.6.1.4 [数字や記号文字の入力], page 48 における例を参考にしてください。

5.6.3 閉じ括弧の自動入力

通常、‘(’を入力したら、‘)’を後で入力する必要があります。‘(’の入力時点で、対になる文字を自動挿入してくれると打鍵数を減らすことができますし、同時に入力忘れの防止にもなるでしょう。

そのために変数 skk-auto-insert-paren が用意されています。この値を non-nil にすると、上記の自動挿入を行います。

²⁷ かつては、‘ あ’などを入力する場合には引数渡しができなかったのですが、本バージョンではこれもできるように拡張されています。

```
----- Buffer: foo -----
彼はこう言った*
----- Buffer: foo -----
```

```
[
```

```
----- Buffer: foo -----
彼はこう言った「*」
----- Buffer: foo -----
```

上記のように ‘ ’ の入力時点で対となる ‘ ’ を自動挿入し、 ‘ ’ と ‘ ’ の間にポイントを再配置するので、その位置からかぎかっこに囲まれた文字列を即始めることができます。

自動挿入すべきペアの文字列は、変数 `skk-auto-paren-string-alist` で指定します。デフォルトは下記のようになっています。

```
((("「" . "」") ("『" . "』") ("(" . ")") ("(" . ")") ("{" . "}")
 ("{" . "}") (" " . " ") ("《" . "》") ("[" . "]") ("[" . "]")
 ("[" . "]") ("【" . "】") ("\" . "\"") ("“" . "”") ("‘" . "’”))
```

これは、ひと言でまとめると、「開き括弧と閉じ括弧とのコンセルを集めたリスト」です。各コンセルの `car` にある文字列を挿入したときに、`cdr` にある文字列が自動挿入されます。²⁸²⁹

キーとなる文字の挿入が行われても、その挿入後のポイントに自動挿入すべき文字が既に存在している場合には、自動挿入は行われないように設計されています。

```
----- Buffer: foo -----
*」
----- Buffer: foo -----
```

```
[
```

```
----- Buffer: foo -----
「*」
----- Buffer: foo -----
```

対になる文字を複数挿入したい場合は、引数を渡して文字を指定します。

```
C-u 2 [
```

```
----- Buffer: foo -----
「「*」」
----- Buffer: foo -----
```

`yatex-mode` など、既に同様の機能が付いているモードがあります。そのようなモードにおいてもこの自動挿入の機能が邪魔になることはないでしょうが、特定のモードに限って自動入力機能をオフにしたい場合は、当該モードに入ったときにコールされるフック変数を利用して設定を行うことができます。

²⁸ このリストの各要素の `car` の文字列は、必ず変数 `skk-rom-kana-rule-list` の規則によって入力されなければなりません。例えば、 ‘ ’ に対する ‘ ’ の自動挿入をするには

```
(setq skk-rom-kana-rule-list
      (append skk-rom-kana-rule-list
              '((" nil "("))))
```

のような設定をする必要があります。

²⁹ 既に SKK モードになっているバッファで変数 `skk-auto-paren-string-alist` を変更した場合は、`C-x C-j` もしくは `C-x j` を 2 度タイプして `skk-mode` もしくは `skk-auto-fill-mode` を起動し直す必要があります。

```
(add-hook 'yatex-mode-hook
  #'(lambda ()
    (when skk-auto-insert-paren
      (make-local-variable 'skk-auto-insert-paren)
      (setq skk-auto-insert-paren nil))))
```

特定のモードにおいて、自動挿入すべき文字を変更したい場合にも同様にフック変数を用いて操作できます。

```
(add-hook 'tex-mode-hook
  #'(lambda ()
    (when skk-auto-insert-paren
      (make-local-variable 'skk-auto-paren-string-alist)
      (setq skk-auto-paren-string-alist
        (cons '("$" . "$") skk-auto-paren-string-alist))))))
```

同様に、特定のペアを削除したい場合は、例えば下記のように設定します。

```
(add-hook 'tex-mode-hook
  #'(lambda ()
    (when skk-auto-insert-paren
      (make-local-variable 'skk-auto-paren-string-alist)
      (setq skk-auto-paren-string-alist
        (delete
          '("$" . "$")
          (copy-sequence skk-auto-paren-string-alist))))))
```

5.6.4 確定するキー

skk-kakutei-key [ユーザ変数]

この変数の値は、明示的な確定動作を行うキーを指定します。標準設定では `\C-j` となっています。

関連事項: Section 5.7.4 [暗黙の確定のタイミング], page 58

5.6.5 候補の選択に用いるキー

変換において、候補が5つ以上あるときは、5番目以降の候補は7つずつまとめてエコーエリアに下記のように表示されます³⁰。

```
----- Echo Area -----
A:嘘 S:拒 D:抛 F:虚 J:拳 K:許 L:渠 [残り 2]
----- Echo Area -----
```

この際、候補の選択に用いるキーは、次の変数によって決定されます。

skk-henkan-show-candidates-keys [ユーザ変数]

7つの異なる文字のリスト。文字は必ず小文字とする³¹。デフォルトは、以下の通り。

```
(?a ?s ?d ?f ?j ?k ?l)
```

skk-henkan-show-candidates-keys-face [ユーザ変数]

選択キーを表示する際のフェイスを指定します。

skk-henkan-rest-indicator [ユーザ変数]

デフォルトは nil。Non-nil であれば '[残り 99++]' の表示を右寄せ配置する。

³⁰ Section 4.3.3 [モード], page 13.

³¹ x, SPC 及び C-g は、それぞれ候補選択中における前候補群の表示、次候補群の表示、取り止めのために割り付けられているので、skk-henkan-show-candidates-keys の中に含めてはいけません。

skk-henkan-rest-indicator-face

[ユーザ変数]

‘[残り 99++]’ の face 属性。デフォルトは default。

5.6.6 モードでの RET

標準設定では、

```

K a k u t e i S P C

----- Buffer: foo -----
  確定 *
----- Buffer: foo -----

RET

----- Buffer: foo -----
  確定
  *
----- Buffer: foo -----

```

のように、モードで RET を入力すると、確定し、かつ改行を行います。この挙動を変えるためのユーザオプションが用意されています。

skk-egg-like-newline

[ユーザ変数]

この変数の値を non-nil にすると、モードで RET を入力したときに確定のみ行い、改行はしません³²。

```

K a k u t e i S P C

----- Buffer: foo -----
  確定 *
----- Buffer: foo -----

RET

----- Buffer: foo -----
  確定 *
----- Buffer: foo -----

```

5.6.7 モードでの BS

標準設定では、モードで BS を押すと、前の一文字を削除した上で確定します。

```

D e n k i y a S P C

----- Buffer: foo -----
  電気屋 *
----- Buffer: foo -----

BS

----- Buffer: foo -----
  電気 *
----- Buffer: foo -----

```

³² 従って、辞書登録モードにおいてモードであるときの RET 入力時の挙動も変化します。標準の確定、登録の動作については、Section 4.3.4 [辞書登録モード], page 15 を参照してください。

skk-delete-implies-kakutei [ユーザ変数]

この変数の値を nil に設定すると、モードで BS を押した時に一つ前の候補を表示します。
例えば、

でんき /電気/伝記/

という辞書エントリがあるとき、以下のようになります。

D e n k i

----- Buffer: foo -----

でんき *

----- Buffer: foo -----

SPC

----- Buffer: foo -----

電気 *

----- Buffer: foo -----

SPC

----- Buffer: foo -----

伝記 *

----- Buffer: foo -----

BS

----- Buffer: foo -----

電気 *

----- Buffer: foo -----

BS

----- Buffer: foo -----

でんき *

----- Buffer: foo -----

変数 skk-delete-implies-kakutei がシンボル dont-update であれば、non-nil 時と同じ動作のうえで個人辞書を更新しません。

なお、変数 skk-delete-implies-kakutei の値にかかわらず、*候補*バッファを表示している場合は一つ前の候補表示に戻る動作となります。

5.6.8 送りあり変換中の C-g

送りありの変換中に C-g を入力すると、モードを抜け、その見出し語と送り仮名を現在のバッファに挿入し、モードに入ります。

N a K u

----- Buffer: foo -----

泣く *

----- Buffer: foo -----

C-g

```
----- Buffer: foo -----
なく*
----- Buffer: foo -----
```

skk-delete-okuri-when-quit [ユーザ変数]

この変数の値を non-nil に設定すると、送りありの変換中に C-g を入力したときの挙動が変化します。モードに入るのは同じですが、同時に送り仮名を消します。送り仮名の入力間違いを修正するには便利です。例えば、以下のようになります。

```
N a K u

----- Buffer: foo -----
泣く*
----- Buffer: foo -----

C-g

----- Buffer: foo -----
な*
----- Buffer: foo -----
```

5.6.9 変換位置の指定方法

SKK では通常、漢字変換の開始と送り仮名開始の位置を大文字で指定しますが、これを任意のキーで指定することで sticky-shift ライクな操作³³ も可能です。

```
(setq skk-sticky-key ";")
```

と設定すると ; キーで変換位置が指定できるようになります。³⁴ 例えば ‘有る’ が

```
; a ; r u
```

でも入力でき、シフトキーを押す必要がなくなります。操作上は see [Q3-4 左手の小指を SHIFT で酷使したくありません。], page 104 などにある通常の sticky-shift と変わりませんが、画面表示は

打鍵	通常の sticky	skk-sticky
;	変化なし	
a	あ	あ
;	あ	あ*
r	あ*r	あ*r

と遷移します。skk-sticky は ; を押した時点で表示が変化するので若干分かり易くなるかと思えます。

キーの設定方法は割当てるキーの種類によって異なります。

1. 表示可能なキー

‘;’ などの表示が可能なキーの場合は

³³ あくまでも「任意のキーで変換位置を指定する」ものであり、sticky-shift そのものではありません。したがって、アスキーモードや abbrev モード、また SKK 以外でも sticky-shift を使いたい場合は前述のような設定を併用する必要があります。

³⁴ ‘skk-hint.el’ を併用する場合は skk-hint-start-char のデフォルトも ‘;’ なのでどちらかを別のキーに割当てる必要があります。


```
(setq skk-sticky-key ";")
```

のように string を設定して下さい。skk-sticky-key に設定した文字そのものを入力したい場合は 2 回続けて打つと入力できます。

2. 表示されないキー

無変換 のような表示を伴わないキーの場合は

```
(setq skk-sticky-key [muhenkan]) ; Microsoft Windows 環境ならば [noconvert]
```

のようにそのキーを表わす vector を設定して下さい。

3. 同時打鍵

2 つのキーを同時に打鍵することでも変換位置を指定できます。例えば *f* と *j* の同時打鍵で指定する場合は

```
(setq skk-sticky-key '(?f ?j))
```

のように character のリストを設定して下さい。

Dvorak 配列のような、押しやすい場所に適当なキーがない環境でもこの機能を使いたい場合に便利かもしれません。

skk-sticky-double-interval [ユーザ変数]
この変数が指定する秒数以内に打鍵されたものを同時打鍵と判定する。デフォルトは 0.1 秒。

5.6.10 1 回の取り消し操作 (undo) の対象

Emacs では本来、連続する 20 文字の挿入が一回の取り消し操作 (アンドゥ) の対象となっています。そこで DDSKK のかな・カナ・全英モードにおける入力も、これと同様の動作をするように設計されています³⁵。正確に言えば、skk-insert, skk-set-henkan-point, skk-jisx0208-latin-insert³⁶ の各関数にバインドされたキー入力については、連続して入力された 20 文字を 1 つのアンドゥの対象としています³⁷。

ただし、これらの DDSKK のコマンドと Emacs 本来の self-insert-command を織り混ぜてキー入力した場合³⁸ は、このエミュレーションは正常に動作しませんが、これは現在の仕様です。

```
a i u e o k a k i k u k e k o s a s i s u s e s o t a t i t u t e t o
```

```
----- Buffer: foo -----
```

```
あいうえおかきくけこさしすせそたちつと ; 連続する 20 文字。
```

```
----- Buffer: foo -----
```

```
C-_
```

```
----- Buffer: foo -----
```

```
;20 文字全てがアンドゥの対象となる。
```

```
----- Buffer: foo -----
```

³⁵ buffer-undo-list に Emacs が挿入したアンドゥの境目の目印を取り除く方法でエミュレートしています。

³⁶ SKK abbrev モードでは、アスキー文字入力が Emacs 本来の self-insert-command により行われているので、エミュレーションのための内部変数である skk-self-insert-non-undo-count をインクリメントすることができず、アンドゥをエミュレートできません。しかも、カンマやピリオドを挿入した時点で、コマンド skk-abbrev-comma や skk-abbrev-period を使うことになるので、本来のアンドゥの機能も損なってしまいます。ただし、現実問題として、元来 SKK abbrev モードは省略形としての見出し語を挿入するためのモードですから、長い見出し語を挿入することはあまりないと考えられます。

³⁷ '20' は Emacs のソースファイルの一部である 'keyboard.c' に定められたマジックナンバーと一致します。

³⁸ かなモードでの入力中、アスキーモードに移行して入力した場合などがこれにあたります。

```
a i u e o k a k i k u k e k o s a s i s u s e s o t a t i t u t e t o n a
```

```
----- Buffer: foo -----
```

```
あいうえおかきくけこさしすせそたちつてとな ; 連続する 21 文字。
```

```
----- Buffer: foo -----
```

```
C- 
```

```
----- Buffer: foo -----
```

```
あいうえおかきくけこさしすせそたちつてと ; 最後の 1 文字のみがアンドウの対象となる。
```

```
----- Buffer: foo -----
```

5.7 変換、確定の前後

関連事項: Section 5.8.4 [送りあり変換の変換開始のタイミング], page 65、Section 5.6.9 [変換位置の指定方法], page 54 (大文字以外で変換位置を指定する方法を説明)

5.7.1 ポイントを戻して モードへ

モードに入り忘れた場合に、手動で マークを付ける方法については、前述しました³⁹。ここで述べる方法では、遡って マークを付ける位置を自動的に選び、しかもカーソルは動きません。

M-Q (大文字の ‘Q’ です。) とタイプすると現在位置の直前の文字列について走査し、同種の文字⁴⁰ が続く限り後方に戻り、 マークを付けます。ポイントは動きません。

```
k a n j i
```

```
----- Buffer: foo -----
```

```
かんじ*
```

```
----- Buffer: foo -----
```

```
M-Q
```

```
----- Buffer: foo -----
```

```
かんじ*
```

```
----- Buffer: foo -----
```

変換開始位置を決定するとき、スペース文字、タブ文字、長音を表わす ‘ー’は無条件に無視されます。ただし、ひらがなの場合は ‘を’ が、カタカナの場合は ‘ヲ’ が見つかった時点で変換開始位置の走査を止め、モードに入ります。変換開始ポイントを ‘を’、‘ヲ’ の直前で止めるのは、たいていその直後から単語が始まるからです。

以上は *M-Q* を引数を与えないで実行した場合です。一方で、*C-u 5 M-Q* のように引数を渡して実行すると、変換開始位置から現在位置までの文字数を指定することができます。この場合は文字種別を問わず、与えられた文字数だけ無条件にポイントを戻します。

後方にポイントを戻す途中で行頭に到達した場合は、更に上の行について、行末の文字列から同様の走査を行い、必要があれば更にポイントを戻します。こうした「行を超えての走査」をやめるためには、変数 `skk-allow-spaces-newlines-and-tabs` の値を `nil` に設定します。

5.7.2 直前の確定を再変換

一番最後に行った変換についての確定を取り消して、再変換を行うことができます。これを「確定アンドウ」と呼びます。

³⁹ See Section 4.3.2.1 [後から モードに入る方法], page 11.

⁴⁰ ひらがな、カタカナ、全角アルファベット、アルファベットの 4 種類のいずれか。

例えば、辞書エントリが

こうこう /高校/孝行/航行/
のようになっているとします。

K o u k o u S P C

----- Buffer: foo -----

高校*

----- Buffer: foo -----

s u r u

----- Buffer: foo -----

高校する*

----- Buffer: foo -----

M-x skk-undo-kakutei

----- Buffer: foo -----

孝行*する

----- Buffer: foo -----

この例では、‘高校’の確定について取り消しています。すると、辞書の第一候補である‘高校’をとばして、次候補である‘孝行’が現れます。ここで更に SPC を押せば次候補である‘航行’が現れ、更にもう一度 SPC を押せば辞書登録モードに入ります。

この例に見られるように、確定アンドゥは、確定した直後でなくとも有効です。より正確には、次の新たな確定⁴¹を行うまでは確定に関する情報が保持されているので、確定アンドゥすることができます。

また、変換、確定に関連しない文字列は、確定アンドゥを行っても削除されないように設計されています。上記の例では、‘する’がそのままカレントバッファに残っています。

skk-undo-kakutei-return-previous-point [ユーザ変数]

この値が non-nil であれば確定アンドゥ後に以前の位置にカーソルが戻ります。

上の例の場合、確定した後のカーソル位置はデフォルトでは‘孝行’の直後のままですが、non-nil であれば‘する’の直後に復帰します。

5.7.3 自動変換開始

モードで見出し語の入力中に「を」や「。」などの文字を打鍵すると、SPC を押したかのように変換を開始⁴²し、モードに入るようになっています。

K a n j i

----- Buffer: foo -----

かんじ*

----- Buffer: foo -----

w o

----- Buffer: foo -----

漢字を*

----- Buffer: foo -----

⁴¹ c-j をタイプして明示的に確定した場合は勿論、暗黙の確定を行った場合も同様です。

⁴² マークからポイントの直前の文字までを見出し語とします。最後に入力された文字は見出し語には含まれません。

変数 `skk-auto-okuri-process` の値を `non-nil` に設定して 送り仮名の自動処理 (see Section 5.8.3 [送り仮名の自動処理], page 62) を行っている場合は、以下のような変換も可能です。ただし、個人辞書に ‘`でき r /出来/[る/出来/]`’ というようなエントリがあると仮定します。

```
D e k i r u n n d e s u

----- Buffer: foo -----
    できるんです
----- Buffer: foo -----

.
----- Buffer: foo -----
    出来るんです。
----- Buffer: foo -----
```

`skk-auto-start-henkan-keyword-list` [ユーザ変数]

この変数の値は、単語や文節の区切りとなるような文字列のリストです。デフォルトは以下のようになっています。

```
("を" "、" "。" "。" "。" "、" " " ? " " " " " ! " " " ; " " : " " " ) " " ;"
 ":" " " ) " " " " " "】" " "】" " "》" " " " " " }" " " ]" " " )" " " }"
 "]" " " ?" " " ." " " ," " " !" )
```

`skk-auto-start-henkan` [ユーザ変数]

この変数の値を `nil` に設定すると、本節で説明した自動変換開始機能を無効にします。

5.7.4 暗黙の確定のタイミング

標準の設定では、確定が済む前に次の文字を入力すると、直ちに確定されます⁴³。これを「暗黙の確定」と呼んでいます。具体的には以下ようになります。

```
K a k u t e i

----- Buffer: foo -----
    かくてい
----- Buffer: foo -----

SPC

----- Buffer: foo -----
    確定
----- Buffer: foo -----

s

----- Buffer: foo -----
    確定 s
----- Buffer: foo -----

u

----- Buffer: foo -----
    確定す
----- Buffer: foo -----
```

⁴³ 正確には、印字可能な文字または `RET` が入力されたときです。

skk-kakutei-early [ユーザ変数]

この変数の値を nil にすると、暗黙の確定を遅らせます。具体的には、括弧 ((,), [,]) または句読点 (, , .) の入力時、次の変換開始 (A から Z までの大文字の入力) 時、あるいは RET 入力時まで暗黙の確定が遅延されます⁴⁴。

```

K a k u t e i

----- Buffer: foo -----
  かくてい
----- Buffer: foo -----

SPC

----- Buffer: foo -----
  確定
----- Buffer: foo -----

s

----- Buffer: foo -----
  確定 s
----- Buffer: foo -----

u r u

----- Buffer: foo -----
  確定する
----- Buffer: foo -----

.

----- Buffer: foo -----
  確定する。
----- Buffer: foo -----

```

5.7.5 積極的な確定

変換候補が一つしか見つからない場合は自動的に確定する、という設定ができます。

skk-kakutei-when-unique-candidate [ユーザ変数]

この値が non-nil の場合、この機能が有効になります。

t であれば送りあり変換、送りなし変換、abbrev モードでの変換、全てでこの機能が有効になります。

また、'okuri-ari', 'okuri-nasi', 'abbrev' を要素とするリストであることもできます。その場合は変換対象がその条件に合致した場合のみ確定変換が機能します。

例: '(okuri-nasi abbrev)'

skk-kakutei-search-prog-limit [ユーザ変数]

この機能は、全ての辞書を検索して、候補が唯一かどうかを調べます。そのため、skk-search-prog-list の内容によってはレスポンスが悪くなる可能性があります。(see Section 5.10.3 [辞書の検索方法の設定], page 69

⁴⁴ skk-kakutei-early の機能と skk-process-okuri-early の機能を同時に有効にすることはできません。skk-kakutei-early の値を non-nil にする場合は skk-process-okuri-early の値を nil にする必要があります。

この値が数値であった場合、検索対象を `skk-search-prog-list` の先頭から数えてこの個数までの辞書に制限します。

それ以外であれば無制限に全ての辞書を検索対象とします。

5.7.6 確定辞書

特定の語は、変換したら即座に確定させる事ができます。これを確定変換と呼び、利用するには「確定辞書」を用意します。例えば、

```
じしょ /辞書/
```

というエントリが確定辞書にあったとします。このとき、

```
Z i s h o
----- Buffer: foo -----
  じしょ
----- Buffer: foo -----

SPC
----- Buffer: foo -----
  辞書
----- Buffer: foo -----
```

のように、SPC を押しただけでいきなり確定します。

確定辞書に登録された同音異義語を得るには、確定直後に `x` をタイプします。すると、モードに戻り次の候補を検索することができます。

次の例では、確定辞書に「辞書」、個人辞書か共有辞書に「自署」が登録されているとします。

```
Z i s y o SPC
----- Buffer: foo -----
  辞書
----- Buffer: foo -----

x
----- Buffer: foo -----
  自署
----- Buffer: foo -----
```

確定辞書の単語は、優先的に変換されます。

`skk-kakutei-jisyo` [ユーザ変数]

確定変換用の辞書ファイルを指定します⁴⁵。この辞書は、標準の配布パッケージには含まれていないので、使用するのであればユーザ側で用意する必要があります。(see Section 5.10.6 [辞書の書式], page 71)

`nil` であれば、確定変換は行われません。

⁴⁵ 確定変換用辞書の見出し語の配列については、サイズが大きい場合は、共有辞書と同様、ソートして二分検索を行い、サイズが小さければ適当な配置で直線的検索を行うことをお勧めします。(Section 5.10.3.2 [辞書検索のための関数], page 69. 及び Section 5.10.6.3 [エントリの配列], page 73) を参照してください。

5.8 送り仮名関連

SKK の送り仮名の処理は、好みが分かれるところです。色々な対策が用意されていますので、試してみてください。

5.8.1 送り仮名の厳密なマッチ

今、個人辞書に

```
おおk /大/多/[く/多]/[き/大]/
```

という送りありエントリがあると仮定します。

ここで `o o K i i S P C` と入力した場合、普通は‘大きい’と‘多きい’という2通りの候補が出力されますが、このうち‘多きい’は現代の日本語として正しくありません。このような場合に、出力される候補を正しい表現のみに絞りこむ方法について、説明します。

`skk-henkan-okuri-strictly` [ユーザ変数]

この変数の値を `non-nil` に設定すると、見出し語がマッチするかどうかのチェックの上に、送り仮名がマッチするかどうかのチェックが行われます。結果として送り仮名がマッチしない候補は出力されません。上記の例では、送り仮名‘き’がマッチする‘大きい’は出力されますが、‘多きい’は出力されません⁴⁶。

個人辞書の送りありエントリが充実していれば、標準の設定よりも候補が絞り込まれるので変換効率がアップしますが、さもないと、すぐに辞書登録モードに入ってしまうため逆に不便になります。

変数 `skk-henkan-okuri-strictly` の値を `non-nil` にすると、辞書登録モードに入っても送り仮名のマッチが厳密に行われます。これは辞書登録の際希望する候補を得るためには障害となります。そのような障害を避けるためには、下記のようにフック変数を設定します。これにより、辞書登録時だけは、一時的に送り仮名の厳密なマッチをしないようになります⁴⁷。

```
(add-hook 'minibuffer-setup-hook
  #'(lambda ()
    (when (and (boundp 'skk-henkan-okuri-strictly)
              skk-henkan-okuri-strictly
              (not (eq last-command 'skk-purge-jisyo)))
      (setq skk-henkan-okuri-strictly nil)
      (put 'skk-henkan-okuri-strictly 'temporary-nil t))))
```

```
(add-hook 'minibuffer-exit-hook
  #'(lambda ()
    (when (and (get 'skk-henkan-okuri-strictly 'temporary-nil)
              (<= (minibuffer-depth) 1))
      (put 'skk-henkan-okuri-strictly 'temporary-nil nil)
      (setq skk-henkan-okuri-strictly t))))
```

5.8.2 送り仮名の優先的なマッチ

Section 5.8.1 [送り仮名の厳密なマッチ], page 61 では、見出し語と送り仮名が一致した場合のみ候補を表示します。ここでは、その条件を緩めて優先的に表示する方法を紹介します⁴⁸。

今、個人辞書に

⁴⁶ この機能は、変数 `skk-process-okuri-early` の値を `non-nil` に設定した状態と共存できません。この理由を知りたい場合は Section 5.8.4 [送りあり変換の変換開始のタイミング], page 65 を参照してください。

⁴⁷ 実は変数 `skk-henkan-okuri-strictly` の値は辞書バッファで参照されるので、ミニバッファのバッファローカル値を変更してもうまくいきません。将来のバージョンでは、これを改良し、辞書バッファでの動作に影響するユーザ変数をバッファローカル化できるようにする予定です。See Section 7.1 [最新情報], page 96。

⁴⁸ ‘大きく’などの候補は鬱陶しいが、すぐに単語登録に入ってしまうのも嫌な人におすすめです。

おお k /大/多/[く/多/]/[き/大/]/

という送りありエントリがあると仮定します。

ここで *o o K i i S P C* と入力した場合、普通は‘大きい’と‘多きい’という2通りの候補が出力されますが、このうち‘多きい’は現代の日本語として正しくありません。このような場合に、出力される候補を正しい表現が優先的にする設定を紹介します。

`skk-henkan-strict-okuri-precedence` [ユーザ変数]

この変数の値を `non-nil` に設定すると、見出し語と送り仮名がマッチした候補を優先して表示します。

上記の例では‘ おお*く ’を変換したとき、まず‘多く’を出力し、次に‘大く’を出力します。

この変数の値が `non-nil` の時は、変数 `skk-process-okuri-early` の値は `nil` でなければなりません⁴⁹。また変数 `skk-henkan-okuri-strictly` が `non-nil` のときは、この変数は無視されます。

5.8.3 送り仮名の自動処理

「あげる」と入力してから、SPC を押しても、「上げる」と変換する機能を、この節では紹介します。

5.8.3.1 どのように変換されるか

`skk-auto-okuri-process` [ユーザ変数]

この変数の値を `non-nil` に設定すると、送り仮名の自動処理が行われます。

例えば、*T a t i a g e r u S P C* と入力した場合を考えます。このとき、検索される見出し語の変化を追うと、

‘たちあげる’ ⇒ ‘たちあげr’ ⇒ ‘たちあg’
⇒ ‘たちa’ ⇒ ‘たt’

のようになります。仮に個人辞書エントリが、

たちあ g /立ち上/[げ/立ち上/]/[が/立ち上/]/
た t /建/断/経/立/[つ/建/断/経/立/]/[ち/建/断/経/立/]/[て/経/立/建/]/

の2つのエントリを含むとすると、見出し語を後方から順に切り詰める過程で‘たちあg’と‘たt’の2つの見出し語の検索時にこれらの辞書エントリがマッチします。

つまり、‘たちあげる’という見出し語に対し、見出し語を最後尾から1文字ずつ切り詰め、「切り詰めの結果残った文字列」と、「切り捨てられた先頭の文字のローマ字プレフィックス」を連結した文字列を送りあり変換の見出し語として、検索します。⁵⁰

次に、マッチしたエントリの各候補に対し、切り捨てられた先頭の文字を送り仮名として取るかどうかをチェックします。この判断には、個人辞書の送り仮名ブロック部分⁵¹を利用します。

‘たちあg’の場合の送り仮名チェックの対象は、切り捨てられた最初の文字の‘げ’です。個人辞書に

[げ/立ち上/]

の部分があることから、送り仮名として取るべきと判断します。また、‘たt’の場合の送り仮名チェックの対象は、‘ち’です。個人辞書に

[ち/建/断/経/立/]

の部分があることから、送り仮名として取るべきと判断します。

⁴⁹ 理由を知りたい場合は、Section 5.8.4 [送りあり変換の変換開始のタイミング], page 65 を参照してください。

⁵⁰ 実際には、普通の送りなし変換として最初は検索されます。個人辞書まで調べて候補が見つからないときは、その後、送り仮名の自動処理の検索に移ります。

⁵¹ Section 5.10.6.2 [送りありエントリのブロック形式], page 72.

こうして、送り仮名がマッチする候補が‘立ち上’、‘建’、‘断’、‘経’、‘立’の5つに絞られます。これらは文字列の長さ順に昇順にソートされ⁵²、それぞれの候補と該当の見出し語から切り捨てられた文字列と連結したもの⁵³を、送り仮名の自動処理の最終候補として返します。上記の例は、‘立ち上げる’、‘建ちあげる’、‘断ちあげる’、‘経ちあげる’、‘立ちあげる’の5つが最終候補になります。

自動送り機能は、個人辞書のみを検索します。

ここで、自動送り機能の長所を考えてみると、

- 送り仮名の最初のローマ字表現を大文字で始める必要がない。
- 送り仮名を正確に思い出せない場合に送り仮名を指定しなくとも変換できる。

などがあります。一方短所としては、

- 意図しない変換をされる割合が増える。
- 個人辞書の送りありエントリが貧弱な場合は、自動処理ができない可能性が高い。

などが考えられます。変数 `skk-auto-okuri-process` の値を `non-nil` に設定しても、従来通りの送りあり変換も同時にできますから、一度この機能を試してみることをお勧めします⁵⁴。

5.8.3.2 辞書登録の際に注意すべきこと

送り仮名の自動処理を行っている場合⁵⁵には、辞書登録の際に注意すべきことがあります。

個人辞書に見出し語‘わたs’についてのエントリが全くない場合、あるいは個人辞書のエントリが

```
わたs /渡/[し/渡]/
```

のような送り仮名のブロックを持たない場合を考えてみます。ここで、`Watasita SPC`と入力すると、送り仮名の自動処理においては送り仮名がマッチしないので、候補が見つからずに辞書登録モードに入ります。

```
W a t a s i t a S P C
----- Buffer: foo -----
  わたした
----- Buffer: foo -----

----- Minibuffer -----
[辞書登録] わたした★
----- Minibuffer -----
```

辞書登録モードで `Watasita RET` と送り仮名を明示的に入力し、‘渡した’と変換して登録したとします。この場合、登録する語の最後が平仮名で終わるので、その最後の平仮名の文字列（上記の例では、‘した’）が見出し語の最後と一致するかを調べます。一致する場合には、辞書の登録を送りありエントリとして行うのかどうかの確認を求めます。

```
W a t a S i t a
----- Minibuffer -----
[辞書登録] わたした 渡した★
----- Minibuffer -----
```

⁵² 長さ順にソートするのは、変換された部分がより長い候補を先順位として出力するためです。

⁵³ ‘該当の見出し語から切り捨てられた文字列’を送り仮名とみなして処理しています。

⁵⁴ 専ら補完的に自動送り処理を利用するのであれば、`(skk-okuri-search)`を `skk-search-prog-list`の最後にもつてくるという手もあります。(see Section 5.10.3 [辞書の検索方法の設定], page 69)

⁵⁵ 変数 `skk-auto-okuri-process` の値を `non-nil` に設定している。

RET

```
----- Echo Area -----
Shall I register this as okuri-ari word: わた s /渡/ ? (y or n)
----- Echo Area -----
```

この確認に対し、‘y’ と回答した場合は、

わた s /渡/[し/渡/] /

という辞書エントリが個人辞書の送りありエントリに書き込まれます。一方 ‘n’ と回答した場合は、個人辞書の送りなしエントリに

わたした /渡した/

というエントリが書き込まれます。本例の場合は、‘y’ と回答するのが正解です。

skk-kana-rom-vector

この変数は、送り仮名部分をローマ字プレフィックスに分解する際に、参照されます。

変数 skk-kana-rom-vector のデフォルトは以下のようになっています。

```
["x" "a" "x" "i" "x" "u" "x" "e" "x" "o" "k" "g" "k" "g" "k" "g"
 "k" "g" "k" "g" "s" "z" "s" "j" "s" "z" "s" "z" "s" "z" "t" "d"
 "t" "d" "x" "t" "d" "t" "d" "t" "d" "n" "n" "n" "n" "n" "h" "b"
 "p" "h" "b" "p" "h" "b" "p" "h" "b" "p" "h" "b" "p" "m" "m" "m"
 "m" "m" "x" "y" "x" "y" "x" "y" "r" "r" "r" "r" "r" "x" "w" "x"
 "x" "w" "n"]
```

このベクトルは、それぞれ下記のかな文字をそのローマ字プレフィックスで現したものです。

```
あ あ い い う う え え お お か が き ぎ く ぐ
け げ こ ご さ ざ し じ す ず せ ぜ そ ぞ た た
ち ち っ っ っ っ っ っ と ど な に ん ね の は ば
ぱ ひ び び ぶ ぶ ぶ へ べ ぺ ほ ぼ ぼ ま み む
め も ちゃ や ゆ ゆ よ よ ら り る れ ろ わ わ ゐ
ゑ を ん
```

これに従うと、見出し語中の送り仮名がローマ字プレフィックスに分解される際、例えば ‘じ’ は ‘j’ に、‘ち’ は ‘t’ に、‘ぶ’ は ‘h’ に、それぞれ分解されます。これらをそれぞれ ‘z’、‘c’、‘f’ に変更することもできます。それには変数 skk-kana-rom-vector の該当部分を “z”、”c”、”f” に変更します。

```
(setq skk-rom-kana-vector
 ["x" "a" "x" "i" "x" "u" "x" "e" "x" "o" "k" "g" "k" "g" "k" "g"
 "k" "g" "k" "g" "s" "z" "s" "z" "s" "z" "s" "z" "s" "z" "t" "d"
 "c" "d" "x" "t" "d" "t" "d" "t" "d" "n" "n" "n" "n" "n" "h" "b"
 "p" "h" "b" "p" "f" "b" "p" "h" "b" "p" "h" "b" "p" "m" "m" "m"
 "m" "m" "x" "y" "x" "y" "x" "y" "r" "r" "r" "r" "r" "x" "w" "x"
 "x" "w" "n"])
```

次にもうひとつ例を挙げます。‘ありがさつき’ に対し ‘有賀さつき’ を登録したい場合は、上記と同様に辞書登録をし、

Shall I register this as okuri-ari entry: ありが s /有賀/ ? (y or n)

の確認に対し ‘n’ と回答します。この結果、個人辞書の送りなしエントリには、

ありがさつき /有賀さつき/

というエントリが書き込まれます。

5.8.4 送りあり変換の変換開始のタイミング

skk-process-okuri-early [ユーザ変数]

この変数の値を non-nil に設定すると、送りあり変換の変換開始のタイミングが早められます。つまり、送り仮名のローマ字プレフィックスの入力時点で変換を開始します。

U g o K

----- Buffer: foo -----

動 k

----- Buffer: foo -----

送り仮名が分からないまま変換しているため、個人辞書が送り仮名に対応した形に成長しません。つまり‘うご k /動/’のような形態のままとなります。ただし、

うご k /動/[</動/]/[か/動/]/[け/動/]/[き/動/]/[こ/動/]/

のようなエントリが既に個人辞書にある場合、それを破壊することはありません⁵⁶。

このユーザオプションを non-nil に設定して SKK モードを起動すると、両立できないオプションである下記オプションは自動的に nil に設定されます。

- skk-kakutei-early
- skk-auto-okuri-process
- skk-henkan-okuri-strictly

既に SKK モードに入った後でこの変数の設定を変更した場合は、カレントバッファで C-x C-j もしくは C-x j を 2 回タイプして SKK モードを起動し直すことで、これらの変数間の衝突を調整します。

See Section 5.7.4 [skk-kakutei-early], page 58.

See Section 5.8.3 [skk-auto-okuri-process], page 62.

See Section 5.8.1 [skk-henkan-okuri-strictly], page 61.

5.9 候補の順序

skk の初期設定では、変換、確定された語が、次回の変換では最初に表示されます。これを変更して、効率良く変換する方法があります。

ベイズ統計を用いた学習のプログラム ‘skk-bayesian.el’ もあります。(see Section 5.1 [ファイル構成], page 19)

確定辞書 (see Section 5.7.6 [確定辞書], page 60) を用いた変換も、候補の順序に影響を与えます。

5.9.1 変換の学習

‘skk-study.el’ は、ある語 A を確定した場合に、A 及びその見出し語 A’ に対して、直前に変換した語 B とその見出し語 B’ を関連語として登録しておき、再度見出し語 A’ の変換を行ったときに、B 及び B’ のペアが直前の何回かに確定した語の中に見つかれば、A を優先して出力する単純な学習効果を提供するプログラムです。

‘~/ .skk’ に (require ‘skk-study) と書いて DDSKK を起動して下さい。以降変換についての学習を始めます。

例えば、‘梅雨には雨が降る’ と変換した場合、

1. ‘雨’ (‘あめ’) の関連語は ‘梅雨’ (‘つゆ’)、
2. ‘降る’ (‘ふ r’) の関連語は ‘雨’ (‘あめ’)、

という風に直前に確定した語を関連語として、語と語の関連性を学習します。

⁵⁶ Section 5.10.6 [辞書の書式], page 71 を参照してください。

ここで続けて、‘傘を振る’ と変換すると、個人辞書がアップデートされてしまい、見出し語 ‘ふ r’ の第一候補は ‘振る’ になってしまいます。

しかし、更に続けて *Ame SPC gaHuRu* と type すると、*HuRu* (‘ふ r’) に対して ‘雨’ (‘あめ’) が関連語になっているため、‘ふ r’ と対で記憶されている ‘降る’ に変換されるというわけです。

では、またここで ‘傘を振る’ と変換し、個人辞書の第一候補が ‘振る’ になった状態で、

```
Ame SPC gaTairyō SPC niHuRu
```

と変換すれば ‘ふ r’ はどう変換されるでしょうか？ 今度は ‘雨’(‘あめ’) と ‘ふ r’ の間に ‘大量’(‘たいりょう’) が入っています⁵⁷ 実はちゃんと

```
‘雨が大量に降る’
```

と変換されます。何故なら (‘ふ r’) の関連語を探す際、*skk-study-search-times*⁵⁸ に指定された回数分だけ遡って、以前に確定した語の中に関連語がないか探すのです。従って、この場合だと、2 つ前の確定情報を探した際に ‘雨’(‘あめ’) を見つけ、これを関連語として、(‘ふ r’) の値を決めようとするのです。

‘*skk-study.el*’ に関するその他のオプションを説明します。

skk-study-max-distance [ユーザ変数]

この変数には *integer* を指定します。直前に確定したポイントと今回の変換ポイントがこの距離以上離れていると学習データを蓄積しないようにします。この変数は、必ずしも文章がバッファの *point-min* から *point-max* へ流れるように書かれるものではなく、ポイントを前に戻したり後へ移動したりして書かれることを想定しています。この変数に *integer* を設定すると、直前の変換よりも前のポイントで変換した場合に学習データを蓄積しないようにします。この変数に *nil* を指定すると直前に確定したポイントとの距離を考慮せずに学習します。この変数のデフォルト値は 30 です。

なお、この変数の値にかかわらず、直前の変換バッファと現在変換を行っているバッファが異なる場合は学習データを蓄積しません。

skk-study-first-candidate [ユーザ変数]

この変数が *non-nil* であれば、第一候補で確定した際も学習します。*nil* であれば、第一候補で確定したときのみ学習データを蓄積しません。学習データをできるだけ小さくしたい場合、この変数を *nil* にすると効果があるかもしれません。この変数のデフォルト値は *t* です。

skk-study-file [ユーザ変数]

学習結果を保存するファイル名です。この変数のデフォルト値は ‘~/skk-study’ です。変数 *skk-user-directory* からも設定ができます。(see Section 5.2.1 [設定ファイル], page 22)

skk-study-backup-file [ユーザ変数]

‘~/skk-study’ のバックアップファイルです。この変数のデフォルト値は ‘~/skk-study.BAK’ です。

skk-study-sort-saving [ユーザ変数]

学習データのデータ構造に関するものです。この変数の値が *non-nil* であれば学習結果をソートしてセーブします。この変数が影響を及ぼすのは学習データの単なる見映えの問題だけです。この変数のデフォルト値は *nil* です。

skk-study-check-alist-format [ユーザ変数]

学習データのデータ構造に関するものです。この変数の値が *non-nil* であれば、学習結果の読み込み時に連想リストのフォーマットをチェックします。これは主に *debug* の目的で使います。この変数のデフォルト値は *nil* です。

⁵⁷ ‘ふ r’ に対して ‘大量’(‘たいりょう’) が関連語として保存されます。勿論 (‘ふ r’) に対する ‘雨’(‘あめ’) の学習もまだ生きています。

⁵⁸ デフォルト値は 5 です。

`M-x skk-study-switch-current-theme`

そのバッファで利用する学習テーマを切り替えます。プロンプト ‘Theme of current buffer: ’ に対して学習テーマ名を入力してください。例えば、科学の話題を書くバッファでは `science` と、法律の話題を書くバッファでは `law` などと入力してください。

`M-x skk-study-remove-theme`

不要な学習テーマを消去します。

`M-x skk-study-copy-theme`

学習テーマを複製します。

5.9.2 候補の順序の固定

`skk` の初期設定では、変換、選択された候補は、次回の変換では最初に表示されます。これに対し、毎回同じ順序で候補を表示させることができます。

`skk-jisyo-fix-order`

[ユーザ変数]

`non-nil` であれば、確定の際に個人辞書の同音語の順序を変更せず、個人辞書に新規追加する際は既出語の後に追加する。

これは、個人辞書のエントリの中の、各候補の順序を変更しないことで、実現されていますから、‘`skk-study.el`’ を用いた学習 (see Section 5.9.1 [変換の学習], page 65) と併用できます。

`skk-jisyo-fix-order` が `non-nil` の時、個人辞書の候補を手軽に並べ替える方法は、現時点ではありません。個人辞書ファイルを直接編集するか、コマンド `M-x skk-edit-private-jisyo` を実行して下さい。(see Section 5.10.9 [個人辞書ファイルの編集], page 74)

直前に変換したばかりの単語は、個人辞書の送りあり / なしエントリの一番上にありますので、すぐに見つけることができます。

5.10 辞書関連

本節では、辞書の種別と形式、設定方法、その他辞書にまつわる動作や設定を説明します。

5.10.1 辞書の種類

共有辞書

‘`SKK-JISYO.S`’ (S 辞書)、‘`SKK-JISYO.M`’ (M 辞書)、‘`SKK-JISYO.ML`’ (ML 辞書)、‘`SKK-JISYO.L`’ (L 辞書) などがあります。通常、個人辞書よりもサイズが大きく、省資源の面からユーザ間で共有して参照されます。

ユーザの変換操作によって内容が書き替えられることはありません。

これら以外にも、共有辞書として使えるファイルが配布されています。それぞれの辞書の詳細については <http://openlab.jp/skk/dic.html> をご参照下さい。

個人辞書

変数 `skk-jisyo` で指定されるファイル。DDSKK を一番最初に使い始めたときにホームディレクトリに自動的に作られます。その後の使用により日々刻々とエントリが追加され、更新されていきます。

なお、最初の個人辞書として S 辞書をリネームして使用するのも良いかもしれません。

`skk-initial-search-jisyo`

`skk-kakutei-jisyo`

これらは共有辞書、個人辞書という区別のいずれにも属しません。これらは個人毎に持つものを使用するか、ユーザ間で共有しているものを使用します。その性格から、辞書内容の更新は行われず、参照のみ行われます。また使用目的から、通常は小さい辞書を使用します。

個人辞書、`skk-initial-search-jisyo`、`skk-kakutei-jisyo` は Emacs のバッファに読み込んで検索を行います。共有辞書は設定により Emacs のバッファに読み込んで使用するか、または辞書サーバ経由で使用します。

5.10.2 辞書ファイルの指定

この節では、辞書ファイルを指定する変数を説明します。個人辞書とバックアップのディレクトリは、変数 `skk-user-directory` でも変更できます。(see Section 5.2.1 [設定ファイル], page 22)

`skk-kakutei-jisyo` [ユーザ変数]
 確定変換 (see Section 5.7.6 [確定辞書], page 60) のための辞書です。一番最初に参照されます。確定変換をしない時は、初期設定の `nil` のままで良いです。

`skk-initial-search-jisyo` [ユーザ変数]
 確定辞書の後、かつ、個人辞書の前に検索を行う辞書です。
 この辞書を適当に指定することにより、最初に出てくる候補を操作することができます。例えば、複数の専門用語毎の辞書を用意しておいて `skk-initial-search-jisyo` の値を切り替えることにより、専門分野毎の専門用語を切り替えて入力することができます。
 この辞書は、標準の配布パッケージには含まれていないので、使用するのであればユーザ側で用意する必要があります。
 不要ならば、初期設定の `nil` のままで良いです。

`skk-jisyo` [ユーザ変数]
 個人辞書。DDSKK を一番最初に起動したとき、変数 `skk-jisyo` が指すファイルが存在しなければ自動的に作られます。

`skk-backup-jisyo` [ユーザ変数]
 個人辞書の予備 (バックアップ) です。検索の対象ではなく、あくまで個人辞書のバックアップとして指定してください。

`skk-large-jisyo` [ユーザ変数]
 共有辞書のひとつ。バッファに読み込んで検索を行います。
 例えば `skk-large-jisyo` に S 辞書か M 辞書を指定し、`skk-aux-large-jisyo` に L 辞書を指定する、という選択肢もあります。
 また、辞書サーバ経由のアクセスも決して遅くはないので、共有辞書はバッファには読み込まない、という設定も自然です。そのためには、`skk-large-jisyo` を `nil` に設定します。

`skk-aux-large-jisyo` [ユーザ変数]
 共有辞書のひとつ。辞書サーバに接続できない時にバッファに読み込んで検索を行う辞書です。

`skk-cdb-large-jisyo` [ユーザ変数]
 共有辞書のうち CDB 形式に変換した辞書。指定した場合は `skk-large-jisyo` より先に検索されます。DDSKK 14 より CDB 形式辞書を辞書サーバを経由せずに直接検索できるようになりました。

`skk-extra-jisyo-file-list` [ユーザ変数]
 SKK では個人辞書の他に、メイン辞書 (`skk-large-jisyo`、`skk-cdb-large-jisyo`) または辞書サーバの設定をして利用するのが一般的ですが、ほかに多彩な辞書がメンテナンスされており、利用可能です。そのためには変数 `skk-search-prog-list` を手動で編集することもできますが、これは厳密にはユーザ変数に分類されていないため、予期しない問題が起こることもあります。
 DDSKK 14.2 以降では追加の辞書を簡単に設定する方法を提供します。以下の例を参考に '`~/skk`' に記述します。

```
(setq skk-extra-jisyo-file-list
      (list '("/usr/share/skk/SKK-JISYO.JIS3_4" . euc-jisx0213)
            "/usr/share/skk/SKK-JISYO.zipcode"))
```

このように、辞書のファイル名のリストを指定します。ただし、変数 `skk-jisyo-code` (see Section 5.10.13 [辞書バッファの文字コードの設定], page 76) とは異なる文字コードのファイルについては、上記の例中の `'SKK-JISYO.JIS3_4'` のように「ファイル名と文字コードのペア」を記述します。

これらの変数の意味するところは初期設定でのものですが、`skk-search-prog-list` の設定で変更することもできます。(see Section 5.10.3.2 [辞書検索のための関数], page 69)

5.10.3 辞書の検索方法の設定

辞書の検索方法の指定は、変数 `skk-search-prog-list` で行われます。特に必要が無ければ、読み飛ばして下さい。

5.10.3.1 辞書検索の設定の具体例

この節では、`skk-search-prog-list` の具体例として DDSKK 14 での初期設定を示し、大体の流れを説明します。

DDSKK では、複数の辞書を扱うことが可能です。複数の辞書が同時に検索されるのではなく、指定した順番に検索します。`skk-search-prog-list` はリストであり、大雑把に言えば、確定されるまで、先頭の要素から順に `lisp` として評価されます。

```
((skk-search-kakutei-jisyo-file skk-kakutei-jisyo 10000 t)
 (skk-search-jisyo-file skk-initial-search-jisyo 10000 t)
 (skk-search-jisyo-file skk-jisyo 0 t)
 (skk-okuri-search)
 (skk-search-cdb-jisyo skk-cdb-large-jisyo)
 (skk-search-jisyo-file skk-large-jisyo 10000)
 (skk-search-server skk-aux-large-jisyo 10000))
```

この例では、

1. `skk-kakutei-jisyo` (see Section 5.7.6 [確定辞書], page 60), `skk-initial-search-jisyo`, `skk-jisyo` (個人辞書) の順に検索を行い、
2. 次に送り仮名の自動処理を行い、(see Section 5.8.3 [送り仮名の自動処理], page 62)
3. その後、`skk-cdb-large-jisyo` と `skk-large-jisyo` の検索を順に行い、
4. 最後に `skk-aux-large-jisyo` に辞書サーバ経由でアクセスしています。

これらの辞書の意味については、see Section 5.10.2 [辞書ファイルの指定], page 68 参照。

もし確定辞書で候補が見つかったらそのまま自動的に確定されます。1 回 SPC を押す動作に対し、プログラム側では新たな候補を見つけるまで上記の動作を進めます。例えば、

1. 確定辞書では候補は見つけれなかったが `skk-initial-search-jisyo` に候補がある場合、そこでいったん止まりユーザにその候補を表示します。
2. 更に SPC が押されると、次は個人辞書を検索します。そこで候補が見つかり、しかもその候補が `skk-initial-search-jisyo` で見つけた候補とは異なるものであったときは、そこでまた止まりその候補をユーザに表示します。

以降、共有辞書についても同様の繰り返しを行います。最後まで候補が見つからなかった時は、辞書登録モードに入ります。

5.10.3.2 辞書検索のための関数

前節で見た通り、変数 `skk-search-prog-list` を適切に定義することによって辞書の検索方法を指定します。そこで使われる辞書検索のための関数を使いこなすことで、より細かい辞書検索の方法を指定することができます。

`skk-search-jisyo-file` *FILE LIMIT &optional NOMSG* [Function]

通常の検索を行うプログラム。変数 `skk-henkan-key` を見出し語として変換検索を実施します。個人辞書、共有辞書又は辞書サーバを使わずに検索を行いたい場合はこの関数を使用します。FILE で指定した辞書ファイルは Emacs のバッファに読み込まれます。

第 1 引数 FILE は、検索対象となる辞書ファイルを指定します。nil を指定したときは、検索を行いません。

第 2 引数 LIMIT は二分検索が行なわれる領域の大きさを指定します。一つの見出し語に対する変換動作に対し、検索対象の領域の大きさが第 2 引数に指定された大きさより小さくなるまでは二分検索が行われ、最後に直線的検索が 1 回行われます。第 2 引数に 0 を指定すると、常に直線的検索のみが行われます。個人辞書に対しては LIMIT を 0 にして下さい。配列がソートされておらず二分検索ができないからです。

第 3 引数 NOMSG が nil ならば、辞書ファイルをバッファに読み込む関数 `skk-get-jisyo-buffer` のメッセージをミニバッファに出力します。non-nil を与えると出力しません。

`skk-search-cdb-jisyo` *CDB-PATH* [Function]
not documented

`skk-search-kakutei-jisyo-file` *FILE LIMIT &optional NOMSG* [Function]

「確定変換」を行う検索プログラム。検索対象の辞書ファイルは Emacs のバッファに読み込まれます。検索対象のファイルから候補を見つけると、内部変数 `skk-kakutei-henkan-flag` を立てて、いきなり確定します。このためユーザーが確定する必要はありません。

引数の意味はいずれも `skk-search-jisyo-file` の場合と同様です。

See Section 5.7.6 [確定辞書], page 60.

`skk-okuri-search` [Function]

形式: (`skk-okuri-search`)

自動送り処理を行うプログラム。変数 `skk-auto-okuri-process` の値が non-nil のときだけ機能します。

個人辞書の送りありエントリを検索対象としているので、個人辞書のバッファを流用します。そのため、専用の辞書バッファは作りません。

See Section 5.8.3 [送り仮名の自動処理], page 62.

`skk-search-server` *FILE LIMIT &optional NOMSG* [Function]

辞書サーバ経由で検索するプログラム。

辞書サーバが使用不能になると辞書ファイルを Emacs のバッファに読み込んで検索を行います。引数の意味はいずれも `skk-search-jisyo-file` と同じですが、これらは辞書を Emacs のバッファに読み込んだときのみ利用されます。

辞書サーバが使う辞書ファイルの設定については、see Section 3.3 [辞書サーバを使いたいときの設定], page 8 及び see Section 5.10.5 [サーバ関連], page 71 をご覧下さい。

5.10.4 Emacs 付属の辞書

GNU Emacs には 'SKK-JISYO.L' を元に変換された 'ja-dic.el' という辞書が付属する。これを用いて通常のかな漢字変換 (送りあり、送りなし、接頭辞、接尾辞) が可能である (ただし 'SKK-JISYO.L' による英数変換、数値変換などはできない)。

DDSKK 14.2 より「ja-dic.el 検索機能」(`skk-search-ja-dic`) が追加された。この `skk-search-ja-dic` は `skk-large-jisyo`、`skk-aux-large-jisyo`、`skk-cdb-large-jisyo` 及び `skk-server-host` の全てが無効な場合に有効となる。

`skk-inhibit-ja-dic-search` [ユーザ変数]

この変数を Non-nil に設定すると、あらゆる場面で `skk-search-ja-dic` を無効とする。

skk-search-ja-dic [Function]

GNU Emacs に付属するかな漢字変換辞書 (ja-dic.el) を用いて検索する。現在の Emacs には SKK-JISYO.L を基に変換された ja-dic.el が付属している。この辞書データを用いて送りあり、送りなし、接頭辞、接尾辞の変換を行う。ただし、SKK-JISYO.L のような英数変換、数値変換などはできない。

5.10.5 サーバ関連

辞書サーバの基本的な設定は、see Section 3.3 [辞書サーバを使いたいときの設定], page 8 をご覧下さい。

skk-servers-list [ユーザ変数]

この変数を使うと、複数のホスト上の辞書サーバを使い分けることができます。

この変数の値は、辞書サーバ毎の情報リストです。各リストは 4 つの要素から成ります。4 つの要素はそれぞれ、ホスト名、辞書サーバ名 (フルパス)、辞書サーバが読み込む辞書ファイル名、辞書サーバが使用するポート番号、となります。ただし、辞書及びポート番号は、辞書サーバ自身が決定することもあるため、そのような場合は nil として構いません。

例えば、以下のように設定します。

```
(setq skk-servers-list
      '(("host1" "/your/path/to/skkserv" nil nil)
        ("host2" "/your/path/to/skkserv" nil nil)))
```

上記の設定の場合、まず host1 上の辞書サーバと接続します。接続できなくなると、次に host2 上の辞書サーバと接続します。

skk-server-report-response [ユーザ変数]

この変数の値が non-nil であれば、変換時に、辞書サーバの送出する文字を受け取るまでに関数 accept-process-output が実行された回数を報告します。

skk-server-inhibit-startup-server [ユーザ変数]

この変数を nil に設定すると、辞書サーバと接続できない場合に、辞書サーバプログラムを call-process して立ち上げようと試みます (デフォルト値は t です)。inetd 経由で立ち上げられる多くの辞書サーバは、call-process で立ち上げることができませんが、'skkserv' のように立ち上げることができる辞書サーバを利用している場合には、この変数を nil に設定するのが良いかもしれません。

skk-server-remote-shell-program [ユーザ変数]

この変数には、リモートシェルのプログラム名を指定します。デフォルトは、システム依存性を考慮する必要があるため、以下の Emacs Lisp コードを評価することにより決定されています。

```
(or (getenv "REMOTESHELL")
    (and (boundp 'remote-shell-program) remote-shell-program)
    (cond
     ((eq system-type 'berkeley-unix)
      (if (file-exists-p "/usr/ucb/rsh") "/usr/ucb/rsh" "/usr/bin/rsh"))
     ((eq system-type 'usg-unix-v)
      (if (file-exists-p "/usr/ucb/remsh") "/usr/ucb/remsh" "/bin/rsh"))
     ((eq system-type 'hpux) "/usr/bin/remsh")
     ((eq system-type 'EWS-UX/V) "/usr/ucb/remsh")
     ((eq system-type 'pcux) "/usr/bin/rcmd")
     (t "rsh"))))
```

5.10.6 辞書の書式

5.10.6.1 送りありエントリと送りなしエントリ

以下は個人辞書の一例です。

```
;; okuri-ari entries.
たと e /例/[え/例]/
も t /持/[つ/持]/[って/持]/[た/持]/[て/持]/[ち/持]/[と/持]/
たす k /助/[け/助]/
うご k /動/[く/動]/[か/動]/[け/動]/[き/動]/[こ/動]/
ふく m /含/[め/含]/[む/含]/[ま/含]/[み/含]/[も/含]/
...
;; okuri-nasi entries.
てん /点/・/天/
ひつよう /必要/
さくじょ /削除/
へんこう /変更/
じゅんじょ /順序/
ぐん /群/郡/
こうほ /候補/
いち /位置/一/壱/
...
```

‘てん /点/・/天/’ を例にして説明します。これは‘てん’が見出し語であり、その候補が、‘点’、‘・’、‘天’です。候補はそれぞれ、‘/’によって区切られています。SKK では、見出し語と候補群を合わせた‘てん /点/・/天/’の一行を「エントリ」と呼びます。

辞書は単純なテキストファイルで、必ず下記の 2 つの行を持っています。

```
;; okuri-ari entries.
;; okuri-nasi entries.
```

この 2 つの行は、それぞれ送り仮名あり、送り仮名なしのエントリの開始地点を示すマークです。‘;; okuri-ari entries.’ までの行で‘;’を行頭に持つ行はコメント行として無視されます。‘;; okuri-ari entries.’ 以降にコメント行を含むことはできません。

‘;; okuri-ari entries.’ と ‘;; okuri-nasi entries.’ の間に囲まれた上半分の部分が送り仮名ありのエントリです。これを「送りありエントリ」と呼びます。‘;; okuri-nasi entries.’ 以下の下半分部分が送り仮名なしのエントリです。これを「送りなしエントリ」と呼びます。

送りありエントリを検索する変換を「送りあり変換」、送りなしエントリを検索する変換を「送りなし変換」と呼びます。SKK では送り仮名の有無が変換方法の 1 つの種別となっています。送り仮名がある変換では送りありエントリのみが検索され、送り仮名がない変換では送りなしエントリのみが検索されます。

1 つの見出し語についてのエントリは 1 行内に書かれます。2 行以上にまたがることはできません。改行を含む候補については、(concat "改\n行") のように、評価すると改行を該当個所に挿入するような Lisp プログラム (see Section 5.5.7 [プログラム実行変換], page 42) に候補を変換して辞書に収めています。

送りありエントリは、基本的には‘も t /持/’ のようになっています。送り仮名部分は、送り仮名をローマ字表現したときの 1 文字目⁵⁹ で表現されています。この 1 エントリで‘持た’、‘持ち’、‘持つ’、‘持て’、‘持と’の 5 つの候補に対応します。その 5 つの候補の送り仮名をローマ字プレフィックスで表現すれば、いずれも‘t’になります。

5.10.6.2 送りありエントリのブロック形式

個人辞書の送りありエントリには‘[’ と ‘]’ に囲まれたブロックがあります。これは、そのブロックの先頭にある平仮名を送り仮名に取る候補群です。

⁵⁹ あるかな文字をローマ字表現したときの 1 文字目を「ローマ字プレフィックス」と呼びます。

たと e /例/[え/例]/

...

ふく m /含/[め/含]/[む/含]/[ま/含]/[み/含]/[も/含]/

この例で見ると、見出し語‘たと e’の場合は‘え’を送り仮名とする 1 つのブロックから構成されています。見出し語‘ふく m’の場合は、‘ま’、‘み’、‘む’、‘め’、‘も’を送り仮名とする 5 ブロックに分けられています。

この送り仮名毎のブロック部分は、`skk-henkan-okuri-strictly` あるいは `skk-auto-okuri-process` のいずれかの変数が `non-nil` である場合に使用されます。その場合、検索において、見出し語の一致に加えて、更に送り仮名もマッチするかどうかをテストします。例えば、

おお k /大/多/[く/多]/[き/大]/

というエントリがあるとします。同じ見出し語‘おお k’であっても、送り仮名が‘き’であれば、候補は‘大’のみで‘多’は無視されます。⁶⁰

現在 <http://openlab.jp/skk/dic.html> で配布されている共有辞書では、‘[’と‘]’を使用した送り仮名毎のブロックの形式に対応していません。個人辞書のみがこの形式で書き込まれています。skk-henkan-okuri-strictly が `nil` であっても送り仮名のブロック形式で書き込まれます。⁶¹

5.10.6.3 エントリの配列

共有辞書では、送りありエントリは‘; ; okuri-ari entries.’から順に下方向に、見出し語をキーとして降順に配置されています。送りなしエントリは、‘; ; okuri-nasi entries.’から順に下方向に、見出し語をキーとして昇順に配置されています。

降順、昇順に配置されているのは、辞書サイズが大きいことに配慮して二分検索を行うためです。⁶²

一方個人辞書は、一番最後に変換された語が最も手前に置かれています。つまり、送りなし、あり、それぞれのエントリが‘; ; okuri-ari entries.’、‘; ; okuri-nasi entries.’を基点として最小ポイントに挿入され辞書が更新されます。⁶³ 個人辞書は通常は共有辞書程はサイズが大きくないので、検索時にはそれぞれの基点から直線的に検索が行われます。

最後に確定された語は、一つのエントリの中の最初の位置に置かれます。

5.10.7 強制的に辞書登録モードへ入る

モードにてエコーエリアで候補表示中に . をタイプすると強制的に辞書登録モードへ入ります。

`skk-force-registration-mode-char` [ユーザ変数]

強制的に辞書登録モードへ入るためのキーキャラクタをこの変数で定義します。標準設定は . です。

5.10.8 誤った登録の削除

誤って個人辞書に登録した単語は削除できます。

削除したい単語を変換により求め、その単語が得られた時点で *x* を入力します。ミニバッファに確認が出るので、それに対し *y e s* と答えます。すると、個人辞書の対応するエントリが削除されます。現在のバッファに先程入力した誤りの変換結果も削除されます。

⁶⁰ See Section 5.8.3 [skk-henkan-okuri-strictly], page 62. See Section 5.8.1 [skk-auto-okuri-process], page 61. See Section 5.8.2 [skk-henkan-strict-okuri-precedence], page 61.

⁶¹ ただし `skk-process-okuri-early` の値が `non-nil` であれば、送り仮名を決定する前に変換を開始することになるので、送り仮名を明示的に入力していても個人辞書にはブロック形式は作られません。

⁶² ソートする際には、見出し語を `unsigned-char` と見なしています。この順序は Emacs が関数 `string<` で文字列を比較するときの順序であり、UNIX の `sort` コマンドでの標準の順序とは異なります。Emacs のコマンド `sort-lines` を用いればファイルをこの順序でソートすることができます。Emacs のコマンド `sort-columns` は内部的に UNIX コマンドの `sort` を使っているので、辞書のソートには使えません。

⁶³ 正確に言えば、送りあり変換では `skk-okuri-ari-min+1` の位置、送りなし変換では `skk-okuri-nasi-min+1` の位置。

さいきてき /再起的/

というエントリを誤って登録したという仮定で、その誤登録を削除する場合を例にとって説明します。

```
S a i k i t e k i SPC

----- Buffer: foo -----
  再起的
----- Buffer: foo -----

X

----- MiniBuffer -----
Really purge ‘さいきてき /再起的/’?(yes or no)
----- MiniBuffer -----

y e s RET

----- Buffer: foo -----

----- Buffer: foo -----
```

5.10.9 個人辞書ファイルの編集

コマンド `M-x skk-edit-private-jisyo` を使うと、個人辞書ファイルが開かれます。

個人辞書ファイルを開いて編集している最中も `skk` を使えますが、`skk` からの単語の登録、削除はできません。(他にも少し制限がありますが、気にならないでしょう。)

編集が終わったら、`C-c C-c` と押すと個人辞書ファイルをセーブしてバッファを閉じます。

5.10.10 個人辞書の保存動作

個人辞書の保存動作について説明します。

個人辞書の保存が行われる場合として、次の4通りがあります。

1. `C-x C-c` (または `M-x save-buffers-kill-emacs`) により Emacs を終了させようとした場合。
2. `M-x skk-save-jisyo` と入力するか、メニューバーより ‘Save Jisyo’ を選択した場合。
3. 変数 `skk-jisyo-save-count` で指定された回数、辞書の更新を行った結果として、自動保存 (オートセーブ) 機能が働くとき。
4. 変数 `skk-save-jisyo-instantly` が `non-nil` であれば、単語登録 (単語削除) のたびに個人辞書を保存する。

保存動作を分析して考えます。まず、Emacs に読み込んだ個人辞書が更新されているかどうかを調べます。更新されていたら保存動作に入ります。Emacs の個人辞書バッファを一時ファイルに保存して、そのファイルサイズが現存の (セーブ前の) 個人辞書より小さくないかどうかをチェックします。個人辞書より小さいときは、保存動作を継続するかどうか、確認のための質問がされます。⁶⁴

```
----- Minibuffer -----
New ~/.skk-jisyo will be 11bytes smaller.  Save anyway?(yes or no)
----- Minibuffer -----
```

⁶⁴ 通常の使用の範囲では `M-x skk-purge-from-jisyo` した場合、あるいは個人辞書をユーザが意図的に編集した場合、複数の Emacs で DDSKK を使用した場合などに、個人辞書が小さくなる場合があります。他の場合はバグの可能性もあります。

ここで `no RET` と答えた場合は、そこで保存動作が中止され、個人辞書は以前の状態のままになります。 `yes RET` と答えた場合は元の個人辞書を退避用の辞書に退避し、一時ファイルに保存した新しい個人辞書を `skk-jisyo` に保存します。

もし、一時ファイルのサイズが 0 である場合は、なんらかの異常と考えられるので、保存動作は直ちに中止されます。その場合は

```
M-x skk-kill-emacs-without-saving-jisyo
```

で Emacs を終了させ、個人辞書 (`skk-jisyo`) 及び個人辞書の退避用辞書 (`skk-backup-jisyo`) をチェックするよう強くお勧めします⁶⁵。

`skk-compare-jisyo-size-when-saving` [ユーザ変数]
この変数の値を `nil` に設定すると、保存前の個人辞書とのサイズ比較を行いません。

`skk-jisyo-save-count` [ユーザ変数]
この変数で指定された回数、個人辞書の更新を行った場合に個人辞書の自動保存が行われます。デフォルトは 50 です。また、この値を `nil` にすると、個人辞書の自動保存機能が無効になります。

ここで、個人辞書の更新回数は確定回数と一致します。また、同じ候補について確定した場合でもそれぞれ 1 回と数えられます⁶⁶。

`skk-save-jisyo-instantly` [ユーザ変数]
この変数が `non-nil` であれば、単語を登録するたび(削除するたび)に個人辞書を保存します。

`skk-share-private-jisyo` [ユーザ変数]
`Non-nil` であれば、複数の SKK による個人辞書の共有を考慮して辞書を更新する。SKK 起動後にこの変数を変更した場合は `M-x skk-restart` で反映させること。

5.10.11 変換及び個人辞書に関する統計

DDSKK では、Emacs の終了時に変換及び個人辞書に関する統計を取っています。その結果はファイル `~/ .skk-record` に保存されます。保存される内容は以下のような形式です。

```
Sun Jul 28 09:38:59 1996 登録: 4 確定: 285 確定率: 98% 語数: 3042
```

上記の「語数」の欄は 1 行を 1 語として数えているので、1 つの見出し語に対し複数の候補を持っている場合、2 つ目以降の候補を無視しています。

`skk-record-file` [ユーザ変数]
統計の結果を保存するファイル名を指定します。変数 `skk-user-directory` から設定ができます。(see Section 5.2.1 [設定ファイル], page 22)

`skk-keep-record` [ユーザ変数]
この変数の値を `nil` に設定すると、本節で説明した統計機能を無効にします。ただし数値であれば、`skk-record-file` を指定数値の行数より大きくしません。

`skk-count-private-jisyo-candidates-exactly` [ユーザ変数]
この変数の値を `non-nil` に設定すると、「語数」の数え方を変更します。具体的には、1 行を 1 語として数えるのではなく、正確に語数を数えます。なお、その分時間がかかります。また、この場合でも '[' と ']' に囲まれた送り仮名毎のブロック形式内は数えません。

```
M-x skk-count-jisyo-candidates
```

このコマンドを使うと、辞書の候補数を数えることができます。

⁶⁵ `skk-jisyo` が既に壊れていても、変数 `skk-backup-jisyo` が指し示すファイルにそれ以前の個人辞書が残っている可能性があります。

⁶⁶ これは、個人辞書の最小ポイントに、常に最後に変換を行ったエントリを移動させるために、エントリ数、候補数が全く増えていなくとも、確定により個人辞書が更新されているからです。

```

M-x skk-count-jisyo-candidates

----- MiniBuffer -----
Jisyo file: (default: /your/home/.skk-jisyo) ~/
----- MiniBuffer -----

. s k k - j i s y o RET

----- Echo Area -----
Counting jisyo candidates... 100% done
----- Echo Area -----

----- Echo Area -----
3530 candidates
----- Echo Area -----

```

ただし、‘[’ と ‘]’ に囲まれた送り仮名毎のブロック形式内は数えません。

また、メニューバーが使用できる環境では、メニューバーを使ってこのコマンドを呼び出すことができます。See Section “メニューバー” in *GNU Emacs Manual*.

5.10.12 辞書バッファ

辞書検索プログラムを実行すると、必要ならば辞書が Emacs のバッファに読み込まれます。このバッファを辞書バッファと呼びます。

辞書バッファの名前は、

「空白+‘*’+辞書ファイル名(ディレクトリ抜き)+‘*’」

という規則に基づいて付けられます。例えば、変数 `skk-large-jisyo` の値が

`‘/usr/local/share/skk/SKK-JISYO.L’`

であるとき、これに対する辞書バッファ名は、

`‘*SKK-JISYO.L*’`

となります。

このバッファのメジャーモードは `fundamental-mode` です。しかし、諸般の事情により、変数 `major-mode` の値をシンボル `skk-jisyo-mode`、変数 `mode-name` の値を文字列 `‘SKK dic’` としています⁶⁷。

5.10.13 辞書バッファの文字コードの設定

`skk-jisyo-code` [ユーザ変数]

この変数の値は辞書ファイルの文字コードを決定します。この変数は以下のような値を取ります。

- “euc”, “ujis”, “sjis”, “jis” などの文字列。順に `euc-jisx0213`, `euc-jisx0213`, `shift_jisx0213`, `iso-2022-jp-3-strict` の各シンボルへ変換されます (`skk-coding-system-alist`)。
- Emacs の coding system (コード系)⁶⁸

⁶⁷ これは、Emacs の `‘dabbrev.el’` の機能との調和を考えた措置です。Dabbrev においては、現在のバッファと同じモードの他のバッファを検索して abbreviation の展開を行うように設定することができますが、仮に辞書バッファにおける変数 `major-mode` の値が `fundamental-mode` のままだとすると、Dabbrev が辞書バッファを検索してしまう可能性があります。この措置によって、そのような事態を回避しています。

⁶⁸ coding system は GNU Emacs の場合 `euc-jp`, `shift_jis`, `junet` などの symbol で表されます。これらは `M-x describe-coding-system` で調べることができます。なお、XEmacs ではこれらの symbol は coding system そのものではなく、coding system object を指示するための symbol として扱われます。具体的には GNU Emacs では (`coding-system-p ‘euc-jp’`) が `t` を返すのに対し、XEmacs では `nil` を返しますが、代わりに symbol が示す coding system object を返す `find-coding-system` 関数が存在します。

- nil

デフォルトは nil です。この場合、DDSKK 側で辞書の文字コードを設定することはせず、Emacs に任せます。

5.10.14 辞書バッファの buffer-file-name

Emacs には save-some-buffers という関数があります。この関数は、ファイルに関連付けられている各バッファについて、変更があればファイルに保存しますが、実際に保存するかどうかをユーザに質問します。

Emacs のコマンドには *M-x compile* のように、save-some-buffers を呼び出すものがあります。もし、個人辞書の辞書バッファがファイル名と関連付けられていたとしたら、こうしたコマンドを実行するたびに個人辞書を保存するかどうか質問されるので、面倒です。

DDSKK では、このような事態を避けるため、辞書バッファにおける変数 buffer-file-name の値を nil に設定しています。

5.11 注釈 (アノテーション)

変換の際に、候補に注釈 (アノテーション) が登録されていれば、それを表示することができます。

5.11.1 アノテーションの基礎

この節では、辞書の中でのアノテーションの取り扱いを説明します。

アノテーションは、ユーザが登録したものと、共有辞書に元々登録されているものの2つに大別されます。

ユーザが付けたアノテーションを「ユーザアノテーション」と呼びます。ユーザアノテーションは、次の形式で個人辞書に登録されます。

「きかん /期間/機関;*機関投資家/基幹;*基幹業務/」

上記のとおり、; の直後に * が自動的に振られる⁶⁹ ことによってユーザが独自に登録したアノテーションであることが分かります。

一方、共有辞書に元々登録されているアノテーションを「システムアノテーション」と呼び、これは ; の直後に * の文字を伴いません。システムアノテーションは、次の形式で辞書に登録されています。

「いぜん /以前;previous/依然;still/」

システムアノテーションは、L 辞書等に採用されています。

5.11.2 アノテーションの使用

skk-show-annotation [ユーザ変数]

この変数の値を non-nil に設定するとアノテーションを表示します⁷⁰。

辞書の候補にアノテーションが登録されていれば、エコーエリア又は other-window に表示します。

skk-annotation-copy-key [ユーザ変数]

C-w をタイプすると、現在表示されているアノテーションを kill ring に保存します。

タイプするキーを変更するには、変数 skk-annotation-copy-key を適宜設定してください。

⁶⁹ * の文字は変換時には表示されません

⁷⁰ Viper 対策はまだ行われていません。`~/viper` に次のように書いて下さい。

```
(viper-harness-minor-mode "skk-annotation")
```

skk-annotation-show-as-message [ユーザ変数]

この変数の値を nil に設定すると、other-window を一時的に開いてアノテーションを表示します。other-window は、その候補を確定するか、その候補の選択を止める (次の候補の表示又は quit) と自動的に閉じます。

skk-annotation-toggle-display-char [ユーザ変数]

「*候補*バッファ」で変換候補を一覧表示しているときにアノテーションの表示/非表示を動的に切り替えるキーを設定します。デフォルトは ^ です。

```
----- Buffer: *候補* -----
```

```
A:射
```

```
S:亥;[十二支](12)いのしし
```

```
D:夷;夷狄
```

```
F:姨;おば
```

```
J:洩;はな
```

```
K:瘕;満身創痍
```

```
L:維;維持
```

```
----- Buffer: *候補* -----
```

```
^
```

```
^
```

```
----- Buffer: *候補* -----
```

```
A:射
```

```
S:亥;
```

```
D:夷;
```

```
F:姨;
```

```
J:洩;
```

```
K:瘕;
```

```
L:維;
```

```
----- Buffer: *候補* -----
```

skk-annotation-function [ユーザ変数]

ユーザアノテーションとシステムアノテーションを区別することで、ユーザアノテーションだけを表示したり、あるいはその逆を行うことが可能です。

変数 skk-annotation-function に「表示したいアノテーションを non-nil と判定する関数」を定義します。アノテーション文字列を引数にして変数 skk-annotation-function が指し示す関数が funcall されて、戻り値が non-nil である場合に限りアノテーションが表示されます。

```
(setq skk-annotation-function
      #'(lambda (annotation)
          (eq (aref annotation 0) ?*)))
```

上記の例では、アノテーションの先頭が * で始まる「ユーザアノテーション」の場合に t を返す 式を skk-annotation-function に定義しました。これによってユーザアノテーションだけが表示されます。

5.11.3 アノテーションの登録

コマンド *skk-annotation-add* &optional *NO-PREVIOUS-ANNOTATION* [Function]

アノテーションを登録/修正するには、アノテーションを付けたい単語を確定した直後に同じバッファで *M-x skk-annotation-add* と実行します。

アノテーションを編集するバッファ(*SKK annotation*)が開いてカレントバッファになりますので、アノテーションとして表示する文章を編集してください。編集が終わったら *C-c C-c* とタイプします。

その単語に既にアノテーションが付いている場合は、あらかじめ当該アノテーションを挿入して *SKK annotation* を開きます。

コマンド *skk-annotation-kill* [Function]

上記 *M-x skk-annotation-add* を実行したもののアノテーションを付けずに *SKK annotation* を閉じたいときは、*C-c C-k* とタイプするか *M-x skk-annotation-kill* を実行してください。

コマンド *skk-annotation-remove* [Function]

最後に確定した候補からアノテーションを取り去りたいときは *M-x skk-annotation-remove* と実行します。

5.11.4 アノテーションとして Wikipedia を表示する

候補にアノテーションの登録がない場合、アノテーションに代えて Wikipedia (<http://ja.wikipedia.org/>) による解説を表示することができます。

モードで候補を表示しているときに TAB を押すと、*skk-annotation-wikipedia-sources* で指定された順で解説を取得してエコーエリアに表示⁷¹ します。

B o k u j o u

----- Buffer: foo -----

ぼくじょう*

----- Buffer: foo -----

SPC

----- Buffer: foo -----

牧場*

----- Buffer: foo -----

TAB

----- Echo Area -----

牧場(ぼくじょう)とは、ウシ、ウマなどの家畜を飼養する施設。訓読みされてまきばと呼ばれることもある。

----- Echo Area -----

エコーエリアに解説が表示されている最中に *C-o* を押すと、関数 *browse-url* を用いてその解説の元となった URL をブラウズします。

skk-annotation-show-wikipedia-url [ユーザ変数]

デフォルトは *nil*。Non-*nil* とすると、見出し語を *SPC* で変換した際にその見出し語のアノテーションが存在しない場合に限り Wikipedia (日本語版) でその語が参照されるための URL をアノテーションとして表示します。表示中に *C-o* を押すと、関数 *browse-url* を用いてその URL をブラウズします。

skk-annotation-wikipedia-sources [ユーザ変数]

アノテーションとして表示する Wikimedia のソースを指定します。デフォルトは *ja.wikipedia*、*en.wiktionary*、*simple.wikipedia*、*en.wikipedia*、*ja.wiktionary* の順です。

⁷¹ 変数 *skk-show-tooltip* が *non-nil* の場合、ツールチップで表示します。

5.11.5 Wikipedia アノテーション機能を SKK の枠をこえて活用する

前述した Wikipedia アノテーション機能は、実は SKK の変換モードだけでなく、Emacs のあらゆる状況で使うことができます。そのためには、コマンド `skk-annotation-wikipedia-region-or-at-point` を任意のキーに割り当てます。

コマンド `skk-annotation-wikipedia-region-or-at-point` **&optional** [Function]
PREFIX-ARG START END

このコマンドは、領域が指定されていればその領域の文字列をキーワードとして Wikipedia アノテーションを探し、表示します。領域が指定されていなければ、可能な範囲でその位置にある単語 (始点と終点) を推測し、Wikipedia アノテーションを探します。一例として、以下のキー割当を紹介します。

```
(global-set-key "\M-i" 'skk-annotation-wikipedia-region-or-at-point)
```

このようにしておくで、何かの意味が調べたくなったとき、領域選択して `M-i` とタイプすれば立ち所にして Wikipedia を検索できます。

さらに、ユーザオプション `skk-annotation-wikipedia-sources` の 2 番目は標準で `en.wiktionary` になっています。例えば、英文を読んでいて `buffer` という語の正確な意味を参照したくなったとします。そのときは単語 `buffer` にポイントを合わせ、`M-2 M-i` とプレフィックス付でコマンドを実行してみてください。

```
----- Buffer: *scratch* -----  
;; This buffer* is for notes you don't want to save, and for ...  
----- Buffer: *scratch* -----
```

```
M-2 M-i
```

すると SKK モードでのアノテーションと同様、以下のような説明が表示されます。

```
----- Echo Area -----  
English, Noun  
buffer (plural&#160;buffers)  
1: Someone or something that buffs.  
2: (chemistry) A solution used to stabilize the pH (acidity) of a  
liquid.  
3: (computing) A portion of memory set aside to store data, often  
before it is sent to an external device or as it is received from an  
external device.  
[...]  
----- Echo Area -----
```

5.12 文字コード関連

関連項目 See Section 5.10.13 [辞書バッファの文字コードの設定], page 76.

5.12.1 文字コードまたはメニューによる文字入力

かなモードで \ キーを入力すると、変数 `skk-kcode-charset` が指す文字集合に従ってバッファ `*skk-list-chars*` に文字の JIS コード一覧が表示されます。

```

----- *skk-list-chars* -----
variable skk-kcode-charset's value is 'japanese-jisx0208'.

01-#x--- 0-- 1-- 2-- 3-- 4-- 5-- 6-- 7-- 8-- 9-- A-- B-- C-- D-- E-- F
 2120      、 ． ； ？ ！ ` ° ´ ˘ ˙
 2130 ^    _  \  \  >  >  "  全 々 々 〇 ー  /
 2140 \   ~  |  ...  '  '  "  "  (  )  [  ]  [  ]
 2150 {  }      《 》 「 」 『 』 【 】【 + - ± ×
 2160 ÷  =  <  >      °      ¥
 2170 $  ¢  £  %  #  &  *  @  §
----- *skk-list-chars* -----

```

```

f
C-f
l          カーソル移動

b
C-b
h          カーソル移動

n
C-n
j          カーソル移動

p
C-p
k          カーソル移動

C-x C-x   カーソル移動

\
o          文字集合の切り替え

c          文字コード入力

i
RET       文書バッファへ文字を挿入

q          skk-list-chars を抜ける。

$          カーソル位置の文字の文字コードを表示

```

ほか、Emacs のコマンド `M-x list-charset-chars` も有用でしょう。

5.12.2 文字コードを知る方法

かな/カナモードで `$` を入力すると、現在のポイント位置の直後にある文字の文字コードをエコーエリア⁷²に表示します。例えば、カーソルを文字 `‘ ’` の上に置いて `$` を入力すると、

```

----- Echo Area -----
‘ ’,KUTEN:07-01, JIS:#x2721, EUC:#xa7a1, SJIS:#x8440, UNICODE:U+0410
----- Echo Area -----

```

とエコーエリアに表示され、この文字がキリル文字であることがわかります。

プレフィックス付きで `\` (つまり `C-u \`) とタイプすると、ポイント直後の文字について文字コード一覧が表示されます。

⁷² 変数 `skk-show-tooltip` が `non-nil` であればツールティップ

```

----- *skk-list-chars* -----
07-#x--- 0-- 1-- 2-- 3-- 4-- 5-- 6-- 7-- 8-- 9-- A-- B-- C-- D-- E-- F
      2720
      2730
----- *skk-list-chars* -----

```

ほか、 Emacs のコマンド `M-x describe-char` も⁷³ 有用でしょう。

`skk-display-code-prompt-face` [ユーザ変数]
 エコーエリアに表示されるメッセージ中「KUTEN:」, 「JIS:」, 「EUC:」, 「SJIS:」及び「UNICODE:」に適用するフェイスです。

`skk-display-code-char-face` [ユーザ変数]
 エコーエリアに表示されるメッセージ中の当該文字に適用するフェイスです。

5.13 DDSKK 以外のツールを用いた辞書変換

5.13.1 skk-lookup

‘`skk-lookup.el`’ を使用すると、辞書検索ツールの Lookup (<http://openlab.jp/edict/lookup/>)⁷⁴ で検索できる辞書を用いて単語の候補を出すことができるようになります⁷⁴。

DDSKK のインストール過程で (`require 'lookup`) が成功する場合は ‘`skk-lookup.el`’ も自動的にインストールされます。Lookup がインストールされているにも関わらず、うまく ‘`skk-lookup.el`’ がインストールされない場合は、‘`SKK-CFG`’ を編集して ‘`lookup.el`’ があるパスを `ADDITIONAL_LISPDIR` に設定し、再度 DDSKK をインストールして下さい。⁷⁵

‘`~/ .skk`’ に以下のように設定します。

```

(setq skk-search-prog-list
  (append skk-search-prog-list
    (list
      '(skk-lookup-search))))

```

DDSKK が用意している検索プログラムの中で最も遅いため、`skk-search-server` の後ろに持ってきて辞書サーバの検索より後にします。

Lookup の agent で利用するのは、`lookup-search-agents` から `ndkks`, `ndcookie` 及び `ndnmz` を取り去ったものです⁷⁶。

5.13.2 skk-look

‘`skk-look.el`’ は、`look` コマンドを使って次の 3 つの機能を提供します⁷⁷。

5.13.2.1 英単語の補完

`skk-use-look` を `non-nil` に設定すると ‘`skk-look.el`’ が使用できるようになります。

例えば、‘`~/ .skk`’ で以下のように設定します。

```

(setq skk-use-look t)

```

SKK abbrev モードが拡張されて、`look` コマンドを使用した補完が有効になります。

⁷³ Emacs 21 では `M-x describe-char-after` です。

⁷⁴ ‘`skk-lookup.el`’ は ‘`skk-look.el`’ とは別ものです。

⁷⁵ 関数 `skk-lookup-search` が ‘`skk-autoloads.el`’ に追加されます (see Section 5.10.3.2 [辞書検索のための関数], page 69).

⁷⁶ `skk-lookup-search-agents` にセットして検索するようにしています。Lookup とは異なる設定をする場合、この変数の設定を変更すれば可能です

⁷⁷ ‘`skk-look.el`’ は ‘`skk-lookup.el`’ とは名前が似ていますが全くの別ものです

```

/ a b s t r

----- Buffer: foo -----
  abstr*
----- Buffer: foo -----

TAB

----- Buffer: foo -----
  abstract*
----- Buffer: foo -----

```

と補完してくれます。通常の補完と同様に、`.` で次の補完候補に、`,` でひとつ前の補完候補に移動できます。

SKK 形式の英和辞書⁷⁸があれば、ここから SPC を押して英和変換ができます。

5.13.2.2 英単語をあいまいに変換して取り出す

見出し語にアスタリスク `*` を入れて SPC を押すと英単語をあいまいにして変換できます。

```

----- Buffer: foo -----
  abstr*
----- Buffer: foo -----

SPC

----- Buffer: foo -----
  abstract*
----- Buffer: foo -----

```

確定すると、`'abstr*'` を見出し語と、`'abstract'` を候補とするエントリが個人辞書に追加されます。このようなエントリを追加したくない場合、ユーザ変数 `skk-search-excluding-word-pattern-function` を適切に設定します。

例えば次のような設定です。

```

(add-hook 'skk-search-excluding-word-pattern-function
  ;; 戻り値が non-nil の時、個人辞書に取り込まない。
  ;; KAKUTEI-WORD を引数にしてコールされるので、不要でも引数を取る
  ;; 必要あり
  #'(lambda (kakutei-word)
      (and skk-abbrev-mode
           (save-match-data
            ;; SKK-HENKAN-KEY が "*" で終わるとき
            (string-match "\\*$" skk-henkan-key))))))

```

5.13.2.3 英単語をあいまいに変換して取り出した後、更に再帰的な英和変換を行う

SKK 辞書に

```

abstract /アブストラクト/抽象/
abstraction /アブストラクション/

```

⁷⁸ SKK 形式の英和辞書 `edict` が提供されています。See Section 2.4 [辞書の入手], page 5.

というエントリがあるとして解説します⁷⁹。

変数 `skk-look-recursive-search` の値を `non-nil` にセットして下さい。

```

abstr*

SPC

abstract

SPC

アブストラクト

SPC

抽象

SPC

abstraction

SPC

アブストラクション

```

このように英単語 + その英単語を見出し語にした候補の「セット」を変換結果として出力することができます。

`skk-look-expanded-word-only` [ユーザ変数]
 この変数の値が `non-nil` であれば、再帰検索に成功した英単語の「セット」だけを出力することができます。再帰検索で検出されなかった英単語は無視して出力しません。

5.13.3 Lisp シンボル名の補完検索変換

SKK abbrev モードにて、Lisp シンボル名を補完して検索し、検索結果を候補として返すことができます。英文字の後ろに「~」を付加してから変換を開始してください。

まずは動作例を示します。

```

/ d e f i ~

----- Buffer: foo -----
defi~*
----- Buffer: foo -----

SPC

----- Buffer: foo -----
defimage*
----- Buffer: foo -----

```

⁷⁹ edict 辞書 'SKK-JISYO.edict' があれば、例えば、

```

(setq skk-search-prog-list
  (append skk-search-prog-list
    (list
      '(skk-search-jisyo-file "/your-path/SKK-JISYO.edict" 0 t))))

```

のように設定することにより、edict 辞書を使用できます。

SPC

```
----- Buffer: foo -----
      define-abbrev*
----- Buffer: foo -----
```

SPC

```
----- Buffer: foo -----
      define-abbrev-table*
----- Buffer: foo -----
```

SPC

```
----- Buffer: foo -----
      define-abbrevs*
----- Buffer: foo -----
```

SPC

```
----- Buffer: *候補* -----
A:define-auto-insert
S:define-category
D:define-ccl-codepoint-translation-table
F:define-ccl-constant-translation-table
J:define-ccl-identity-translation-table
K:define-ccl-program
L:define-ccl-slide-translation-table
----- Buffer: *候補* -----
```

この機能を有効とするには、リスト `skk-search-prog-list` の要素に関数 `skk-search-lisp-symbol` を加えてください。

```
(add-to-list 'skk-search-prog-list
            '(skk-search-lisp-symbol) t)
```

なお、見出し語に ‘~’ を含む辞書もあります。例えば ‘SKK-JISYO.JIS3_4’ には

```
A~ /チルド付き A(LATIN CAPITAL LETTER A WITH TILDE)/
```

と登録⁸⁰されています。したがって、A~ SPC と変換したときに「チルド付き A」が表示されるか、Lisp シンボル名が補完されるかは、リスト `skk-search-prog-list` 内の要素の順によります。

`skk-search-lisp-symbol` **&optional** *PREDICATE NOT-ABBREV-ONLY* [Function]
WITHOUT-CHAR-MAYBE

オプション *PREDICATE* で補完検索する範囲（関数名、変数名、コマンド名）を限定することができます。詳細は *docstring* を参照してください。

`skk-completion-search-char` [ユーザ変数]
`skk-completion-search` による変換機能を指示するキーキャラクタ。デフォルトは ~ です。

⁸⁰ 実際には JIS X 0213 の 1 面 9 区 26 点の 1 文字が登録されています。この文字を `skk.texi` に直接記載するのは避けました。

5.14 飾りつけ

5.14.1 仮名文字のローマ字プレフィックスのエコー

`skk-echo` [ユーザ変数]
 この変数の値は、仮名文字のローマ字プレフィックス⁸¹のエコーの有無を制御します。

変数 `skk-echo` の値が `non-nil` であれば、仮名文字のローマ字プレフィックスが、入力時点でいったん現在のバッファに挿入され、続く母音の入力の際に、かな文字に変換された時点で現在のバッファから消去されます。

```
t
----- Buffer: foo -----
t
----- Buffer: foo -----

a
----- Buffer: foo -----
た
----- Buffer: foo -----
```

変数 `skk-echo` の値が `nil` であれば、仮名文字のローマ字プレフィックスのエコーは行われません。これを上記の例で考えると、‘t’ が現在のバッファに挿入されず、続く母音 (a) が入力されたとき ‘た’ の文字が挿入されます。

`skk-prefix-hiragana-face` [ユーザ変数]

`skk-prefix-katakana-face` [ユーザ変数]

`skk-prefix-jisx0201-face` [ユーザ変数]

それぞれかなモード、カナモード、JIS X 0201 モードにおけるローマ字プレフィックスのフェイスを指定します。

5.14.2 入力モードを示すモードラインの文字列の変更

下記の変数の値を変更することによって、モードライン上の入力モードを示す文字列を変更することができます。

`skk-latin-mode-string` [ユーザ変数]
 アスキーモードを示す文字列。標準は、“SKK”。

`skk-hiragana-mode-string` [ユーザ変数]
 かなモードを示す文字列。標準は、“かな”。

`skk-katakana-mode-string` [ユーザ変数]
 カナモードを示す文字列。標準は、“カナ”。

`skk-jisx0208-latin-mode-string` [ユーザ変数]
 全英モードを示す文字列。標準は、“全英”。

`skk-abbrev-mode-string` [ユーザ変数]
 SKK abbrev モードを示す文字列。標準は、“a あ”。

⁸¹ See Section 5.10.6.1 [送りありエントリと送りなしエントリ], page 72.

5.14.3 入力モードを示すカーソル色に関する設定

`skk-use-color-cursor` [ユーザ変数]

この変数が `non-nil` ならば、カーソルを色付けします。`nil` ならば、この機能を無効にします。標準では、ウィンドウシステムを使用しているなおかつ色表示が可能な場合に限り、この機能が有効になります。

この機能が有効になっているとき、以下の変数の値を変更することで、各モードにおけるカーソルの色を変更できます。

`skk-cursor-default-color` [ユーザ変数]

SKK モードがオフであることを示すカーソル色。標準では、カーソルのある該当フレームにおける標準のカーソル色を使います。

`skk-cursor-hiragana-color` [ユーザ変数]

かなモードであることを示すカーソル色。標準は、背景の明暗により `coral4` または `pink` です。

`skk-cursor-katakana-color` [ユーザ変数]

カナモードであることを示すカーソル色。標準は、背景の明暗により `forestgreen` または `green` です。

`skk-cursor-jisx0201-color` [ユーザ変数]

JIS X 0201 モードであることを示すカーソル色。標準は、背景の明暗により `blueviolet` または `thistle` です。

`skk-cursor-jisx0208-latin-color` [ユーザ変数]

全英モードであることを示すカーソル色。標準は、`gold` です。

`skk-cursor-latin-color` [ユーザ変数]

アスキーモードであることを示すカーソル色。標準は、背景の明暗により `ivory4` または `gray` です。

`skk-cursor-abbrev-color` [ユーザ変数]

`skk abbrev` モードであることを示すカーソル色。標準は、`royalblue` です。

5.14.4 変換候補一覧の表示方法

変換候補一覧の表示方法は、次の4つに大別されます。

- 現在のウィンドウにインライン表示する
- ツールチップで表示する
- 現在のウィンドウの隣に別なウィンドウを開いて表示する (ポップアップ)
- エコーエリアに表示する

ここではその表示方法の制御について解説します。

`skk-show-inline` [ユーザ変数]

この値が `non-nil` であれば、候補一覧を現在のポイント位置でインライン表示します。

また、値が `vertical` であれば、各候補を縦方向にインライン表示します。

ただし、XEmacs ではこの機能はサポートされません。

`skk-inline-show-face` [ユーザ変数]

インラインで候補を表示する際のフェイスを指定します。

候補文字列のフェイス属性をそのまま使いたい場合は `nil` に設定して下さい。

- `skk-inline-show-background-color` [ユーザ変数]
 インラインで候補を表示する際の背景色を指定します。`skk-inline-show-face` または `skk-treat-candidate-appearance-function` で背景色が指定されていない文字に対してのみ作用します。
- `skk-show-tooltip` [ユーザ変数]
 この値が `non-nil` であれば、候補一覧をツールチップで表示します。同時に、この変数は注釈 (アノテーション) の表示方法も制御します。
 See Section 5.11 [注釈 (アノテーション)], page 77.
- `skk-tooltip-face` [ユーザ変数]
 ツールチップで候補を表示する際のフェイスを指定します。
 候補文字列のフェイス属性をそのまま使いたい場合は `nil` に設定して下さい。
- `skk-tooltip-mouse-behavior` [ユーザ変数]
 ツールチップを表示する位置及びマウスポインタの挙動を指定します。Emacs21 は `'follow` 固定です。下記に掲げるシンボル以外のシンボルを指定した場合は `nil` となります。
- `'follow` マウスポインタをカーソル位置へ移動させてツールチップを表示します。ツールチップの表示を終えるとマウスポインタは元の位置へ戻ります。ただし、元のマウスポインタが Emacs フレーム外であったならばツールチップの表示を終えてもマウスポインタはカーソル位置のままです。
 - `'banish` マウスポインタを Emacs フレーム右上隅へ移動させてツールチップを表示します。ツールチップの表示を終えてもマウスポインタは Emacs フレーム右上隅のままです。
 - `'avoid` マウスポインタを Emacs フレーム右上隅へ移動させてツールチップを表示します。ツールチップの表示を終えるとマウスポインタは元の位置へ戻ります。ただし、元のマウスポインタが Emacs フレーム外であったならばツールチップの表示を終えてもマウスポインタは Emacs フレーム右上隅のままです。
 - `'avoid-maybe` マウスポインタが Emacs フレーム内であれば `'avoid` と同じ動作です。マウスポインタが Emacs フレーム外であればマウスポインタ位置を変更せず、その位置にツールチップを表示します。
 - `nil` マウスポインタを一切移動せず、その位置にツールチップを表示します。ツールチップのテキストとマウスポインタが重なったり、うまくツールチップが表示できなかったりする場合があります。
- `skk-show-candidates-always-pop-to-buffer` [ユーザ変数]
 この値が `non-nil` であれば、画面を上下に分割したうえで、変換一覧を専用の「*候補*バッファ」で表示します。
 候補一覧表示中に、この値を動的に切り換える手段が用意されています。
- `skk-show-candidates-toggle-display-place-char` [ユーザ変数]
 デフォルトは `C-f` です。このキーを候補一覧表示時にタイプすると、候補一覧の表示位置をエコーエリアとバッファとで切り替えます。
- `skk-candidate-buffer-background-color` [ユーザ変数]
 *候補*バッファの背景色を指定します。背景色を付けたくない場合は `nil` を指定すること (デフォルト)。
- デフォルトでは3つの変数 `skk-show-inline`, `skk-show-tooltip`, `skk-show-candidates-always-pop-to-buffer` とも `nil` です。この状態では候補一覧はエコーエリアに表示⁸² します。もしも、これら変数のうち2つ以上が `non-nil` の場合、優先順位は上記の解説の順です。

⁸² `frame-width` が不足する場合は *候補*バッファに表示します。

5.14.5 モードにおける変換候補のハイライト表示

`skk-use-face` [ユーザ変数]

この変数の値が `non-nil` であれば、Emacs のフェイス機能を使って変換候補をハイライト表示します。

このハイライト表示には Emacs のオーバーレイ (overlay) の機能を使います⁸³。

`skk-henkan-face` [ユーザ変数]

この変数の値はフェイスであり、このフェイスによって変換候補がハイライト表示されます。標準では、背景の明暗により “black/darkseagreen2” 又は “white/darkolivegreen” を用います。

変数 `skk-henkan-face` には、既存のフェイス⁸⁴ を指定できますが、新たにフェイスを作ることもできます。そのために、以下の関数が用意されています。

`skk-make-face FACE` [Function]

形式: (`skk-make-face FACE`)

この関数は、引数 `FACE` と同じ名前のフェイスを作成して、そのフェイスを返します。フェイスの前景色・背景色は、引数 `FACE` によって、例えば以下の例のように決定されます。

```
(skk-make-face 'DimGray/PeachPuff1)
(setq skk-henkan-face 'DimGray/PeachPuff1)
```

この場合、前景色は `DimGray` に、背景色は `PeachPuff1` になります。

もうひとつ例を挙げます。

```
(skk-make-face 'RosyBrown1)
(setq skk-henkan-face 'RosyBrown1)
```

この場合、前景色は `RosyBrown1` になります。背景色が無指定の場合はバッファの背景色がそのまま見えます。

5.14.6 変換候補の更なる装飾

変換候補についてユーザの任意の加工を施すための変数を用意してあります。

`skk-treat-candidate-appearance-function` [ユーザ変数]

この変数に適切な形式で関数を収めることによって、変換候補をユーザの任意に加工することができます。

「適切な形式」とは、次のとおりです。

1. 引数を 2 つ取ること。
2. 第 1 引数は文字列として扱うこと。これは加工前の文字列に相当する。
3. 第 2 引数が `nil` の時は通常の変換時、`non-nil` の時は候補一覧表示時を表すものとして扱うこと。
4. 返り値は次のいずれかとする。こと。

‘文字列’

この場合、この文字列は候補と注釈を両方含みうるものとして処理される。

⁸³ 以前のバージョンではテキスト属性 (text property) を使用していました。

オーバーレイ属性はテキスト属性と異なり、テキストの一部とは見なされません。そのため、テキストのコピーの際にオーバーレイ属性は保存されません。その他にも、オーバーレイの移動やその属性の変更はバッファの変更とは見なされないこと、オーバーレイの変更はバッファのアンドゥリストに記録されないこと、などが特徴として挙げられます。

なお、XEmacs にはオーバーレイ機能はありません。代わりに `extent` というものが用意されているのでそれを利用します。

⁸⁴ Emacs 標準では `default`, `modeline`, `region`, `secondary-selection`, `highlight`, `underline`, `bold`, `italic`, `bold-italic` があります。

‘(候補 . 注釈)’

この場合、候補はもう注釈を含まないものとして処理される。注釈については先頭が ‘;’ かどうかを調べた上で処理される。

‘(候補 . (セパレータ . 注釈))’

この場合、候補はもう注釈を含まないものとして処理される。セパレータは通常の ‘;’ の代わりに利用される。注釈はもうセパレータを含まないものとして処理される。

ファイル ‘etc/dot.skk’ に設定例があるほか、サンプルとして関数 `skk-treat-candidate-sample1` と `skk-treat-candidate-sample2` を用意してあります。ファイル ‘~/dot.skk’ に次のいずれかを書いてみて変換候補の装飾を試してください。

```
(setq skk-treat-candidate-appearance-function
      'skk-treat-candidate-sample1)
(setq skk-treat-candidate-appearance-function
      'skk-treat-candidate-sample2)
```

5.14.7 モードラインの装飾

XEmacs 及び Emacs 21 以降では以下の機能が使用できます。

5.14.7.1 インジケータ

`skk-indicator-use-cursor-color` [ユーザ変数]

DDSKK のインジケータをモードラインの左に表示⁸⁵ している場合、インジケータの色がカーソルの色と同期します。インジケータに色を付けたくない場合は、この変数を `nil` にします。

インジケータに独自色を使いたい場合は、以下のフェイス⁸⁶ を設定します。この場合カーソルの色は参照されません。

Emacs 21 以上⁸⁷ の場合

- `skk-emacs-hiragana-face`
- `skk-emacs-katakana-face`
- `skk-emacs-jisx0208-latin-face`
- `skk-emacs-jisx0201-face`
- `skk-emacs-abbrev-face`

XEmacs の場合

- `skk-xemacs-hiragana-face`
- `skk-xemacs-katakana-face`
- `skk-xemacs-jisx0208-latin-face`
- `skk-xemacs-latin-face`
- `skk-xemacs-jisx0201-face`
- `skk-xemacs-abbrev-face`

なお、インジケータを右クリックするとポップアップメニューが表示されます。

5.14.7.2 アイコン

`skk-show-icon` [ユーザ変数]

変数 `skk-show-icon` の値を `non-nil` と設定することにより、モードラインに SKK のアイコンが表示されます。アイコンの画像ファイル ‘`skk.xpm`’ へのパスは、変数 `skk-icon` で定義されています。

⁸⁵ デフォルトでは、左です。See Section 4.1 [起動と終了], page 9.

⁸⁶ 変数 `window-system` が `nil` の場合は、これらフェイスは未定義となります。

⁸⁷ 変数 `mule-version` の値が 5.0 以上の Emacs

5.15 ユーザガイド関連

5.15.1 エラーなどの日本語表示

標準では、エラー、メッセージ及びミニバッファでのプロンプトは、英語で表示されます。

`skk-japanese-message-and-error` [ユーザ変数]
この変数の値を `non-nil` に設定すると、エラー、メッセージ及びミニバッファでのプロンプトを日本語で表示します。標準では `nil` です。

`skk-show-japanese-menu` [ユーザ変数]
この変数の値を `non-nil` に設定すると、メニューバーを日本語で表示します。

5.15.2 助言的案内メッセージの表示

`skk-verbose` [ユーザ変数]
この変数の値を `non-nil` に設定すると、SKK が従来表示しなかったような入力・変換中の助言的メッセージが表示されます。
(`setq skk-verbose t`)

このオプションの具体的な効果をみていきましょう。

モード

ファンクションキー (F1 ~ F10) の使い方を表示します。`skk-verbose` の設定とともに `skk-j-mode-function-key-usage` を以下のように設定してみてください。

```
(setq skk-j-mode-function-key-usage 'conversion)
```

すると、モードで一定時間 (標準では 1.5 秒) キー入力がなかったとき、エコーエリアに以下のようなメッセージが表示されます。

```
----- Echo Area -----
[F5] 単漢字 [F6] 無変換 [F7] カタカナ [F8] 半角カナ [F9] 全角ローマ [F10]
ローマ
----- Echo Area -----
```

この案内に従ってファンクションキーを押すことで、一時的に単漢字変換やカタカナ変換を行うことができます。

モード

Wikipedia アノテーション機能の使い方をメッセージで案内します。`skk-verbose` の設定とともに `skk-show-annotation` を以下のように設定してみてください。

```
(setq skk-show-annotation t)
```

すると、モードで一定時間 (標準では 1.5 秒) キー入力がなかったとき、エコーエリアに以下のようなメッセージが表示されます。

```
----- Echo Area -----
{どのWiki?}[C-1 C-i]ja.wikipedia [C-2 C-i]en.wiktory
[C-3 C-i]simple.wikipedia [C-4 C-i]en.wikipedia [C-5 C-i]ja.wiktory
----- Echo Area -----
```

この案内に従って、例えば `C-1 C-i` をタイプすると日本語 Wikipedia の該当記事を調べて、あればその一部をアノテーションとして表示します。

一方、現在の変換候補に対するアノテーションが既に表示されているときは、以下のメッセージが上記のものと交互に表示されます。

```
----- Echo Area -----
{アノテーション}[C-w] コピー [C-o]URL ブラウズ [C-i] デフォルトWiki 参
照
----- Echo Area -----
```

この案内に従って `C-w` をタイプすればアノテーションの全文を `kill ring` に保存して利用することができます。また `C-o` を押した場合には、もし現在のアノテーションが Wikipedia アノテーションであればその出典となる Wikipedia/Wiktionary のページをウェブブラウザで表示します。

`skk-verbose-wait` [ユーザ変数]
助言的メッセージを表示するまでの待ち時間 (秒)。標準は 1.5 秒です。

`skk-verbose-message-interval` [ユーザ変数]
助言的メッセージが複数ある場合の 1 メッセージあたり表示時間を秒で指定する。標準は 5.0 秒です。この時間が経過したら表示を次の助言的メッセージに切り替えます。

5.16 VIP/VIPER との併用

`skk-use-viper` [ユーザ変数]
この変数の値を `non-nil` に設定すると、VIPER に対応します。

VIPER については Section “VIPER” in *VIPER Manual*. を参照してください。

また、VIPER の前身である VIP にも対応します。ただし、正式に対応しているバージョンは 3.5 のみです。これは Mule 2.3 に標準添付します⁸⁸。

5.17 I-search 関連

5.17.1 起動時の入力モードの指定

`skk-isearch-use-previous-mode` [ユーザ変数]
この変数の値が `non-nil` であれば、次のインクリメンタル・サーチ起動時の入力モードは、前回のインクリメンタル・サーチでの入力モードになります。

`skk-isearch-start-mode` [ユーザ変数]
この変数の値は、インクリメンタル・サーチの起動時の入力モードを決定します。ただし変数 `skk-isearch-use-previous-mode` の値が `non-nil` ならば、そちらの機能が優先されます。この変数は以下の値を取り得ます。

```
'latin'
    アスキーモード
'hiragana'
    かなモード
'jisx0208-latin'
    全英モード
```

5.17.2 間に空白等を含む文字列の検索

‘検索’ という文字列をインクリメンタル・サーチにより検索する場合に、バッファが以下のような状態になっていることがあります。

```
----- Buffer: foo -----
この行末から始まる文字列を検
索して下さい。
----- Buffer: foo -----
```

このような場合のために、Emacs は正規表現によるインクリメンタル・サーチを提供しています。DDSKK はこの正規表現によるインクリメンタル・サーチにも対応しているため、空白や改行を含んだ検索も可能です。

⁸⁸ ちなみに、VIP 3.5 の作者は、SKK の原作者でもある佐藤雅彦教授です。VIP 3.5 の発展版である VIPER は現在もメンテナンスされています。Emacs 19, 20 には、VIP、VIPER とともに標準添付します。

M-x isearch-forward-regexp

前方への正規表現によるインクリメンタル・サーチ。C-u C-s または M-C-s で起動します。

M-x isearch-backward-regexp

後方への正規表現によるインクリメンタル・サーチ。C-u C-r または M-C-r で起動します。

以下は空白や改行の処理を制御する変数です。

skk-isearch-whitespace-regexp

この変数の値は正規表現です。この正規表現にマッチする要素は、正規表現によるインクリメンタル・サーチにおいては、単語を区切る要素ではないと判断されます。この変数のデフォルトは以下のようになっています。

```
"\\(\\s \\|[ \\t\\n\\r\\f]\\|\\)*"
```

変数 *skk-isearch-whitespace-regexp* の値を変更することで、正規表現によるインクリメンタル・サーチを拡張することができます。例えば、電子メールの引用部分を検索する場合を考えます。

- > 引用部分も検索できる。

上記のうち、「検索」という語は 2 行にわたっているうえ引用マークが挿入されています。ここで

```
(setq skk-isearch-whitespace-regexp "\\(\\s \\|[ \\t\\n\\r\\f<>|]\\|\\)*")
```

と設定することにより、「検索」を検索できるようになります。

5.18 picture-modeとの併用

SKK モードを *picture-mode* において使用した場合は、以下のような問題点があります。ただし、これらは *picture-mode* の問題なので、現在のところ DDSKK 側では対処していません。

1. SKK モードで全角文字を入力した場合に、BS で全角文字を消すことができません。現状では、後方にある文字を消したい場合は、その文字にポイントを合わせ、C-c C-d で一文字ずつ消す必要があります。
2. コマンド *picture-movement-up* や *picture-movement-down* により上下に全角文字を挿入した場合に、桁がずれる場合があります。

関数 *move-to-column-force* の中で使用されている関数 *move-to-column* の引数として、全角文字を無視した桁数が与えられることがあり、そのときカーソル移動ができないため、これらの問題が生じます。

6 ローマ字入力以外の入力方式

DDSKK は、SKK 旧来のローマ字式かな入力 (訓令式、ヘボン式) 方式のほか、各種キー配列と入力方式に対応しています。

6.1 AZIK

AZIK (エイズィック) は QWERTY 配列をベースとした拡張ローマ字入力です。一般のローマ字入力がそのまま使える上での拡張であることが特徴です。

拡張ローマ字入力『AZIK』 ・ 『ACT』で快適な日本語入力を!
(<http://hp.vector.co.jp/authors/VA002116/azik/azikindx.htm>)

```
skk-use-azik [ユーザ変数]
  この値が non-nil であれば AZIK 拡張が有効となります。 '~/.skk' に
    (setq skk-use-azik t)
  と書きます。
```

```
skk-azik-keyboard-type [ユーザ変数]
  AZIK で使うキーボードのタイプをシンボルで指定する。 'jp106'、'jp-pc98' 又は 'us101'
  を指定してください。
```

6.2 ACT

ACT は AZIK の考え方を Dvorak 配列に適用し、Dvorak 配列でかなを快適にタイプできるように考案された方式です。

ACT (AZIK on Dvorak) (http://www1.vecceed.ne.jp/~bemu/act/act_index.html)

```
skk-use-act [ユーザ変数]
  この値が non-nil であれば ACT 拡張が有効となります。 '~/.skk' に
    (setq skk-use-act t)
  と書きます。
```

6.3 TUT-code

TUT-code は 2 ストローク系の日本語直接入力方式の一つです。

<http://plone.crew.sfc.keio.ac.jp/groups/tut-code>

使用するには、SKK のインストール時にいくつかのファイルをインストールする必要があります。SKK ソースの 'tut-code' ディレクトリにある 'skk-tutcdef.el' と 'skk-tutcode.el' を、SKK ソースのトップディレクトリにコピーして、SKK のインストールを再度行います。

See Section 2.2 [DDSKK のインストール], page 3.

その後、 '~/.skk' に

```
(require 'skk-tutcdef)
```

と書きます。

6.4 かな入力と親指シフト

DDSKK はローマ字式ではない、いわゆるかな入力方式をサポートします。具体的には

- 旧 JIS 配列でのかな入力
- 親指シフト方式でのかな入力

に対応しています。これを使うにはまず、nicola-ddskk 拡張パッケージをインストールする必要があります。SKK ソースディレクトリの 'nicola' ディレクトリに移動し、ドキュメントに従ってインストールしてください。

`http://openlab.jp/skk/skk/main/nicola/README.ja`

続いて設定をします。

`skk-use-kana-keyboard` [ユーザ変数]

この変数を `non-nil` に設定すると、かな入力サポートが SKK 起動時に有効になります。

```
(setq skk-use-kana-keyboard t)
```

`skk-kanagaki-keyboard-type` [ユーザ変数]

この変数で、かな入力サポートの種類を切換えます。

'106-jis'

日本語 106 キーボード (旧 JIS 配列) でのかな入力に対応します。

```
(setq skk-kanagaki-keyboard-type '106-jis)
```

'nicola-jis'

日本語 106 キーボード (旧 JIS 配列) での親指シフトエミュレーションに対応します。

```
(setq skk-kanagaki-keyboard-type 'nicola-jis)
```

'omelet-jis'

`nicola-jis` と同様ですが、より入力しやすい配列が考慮されています。

```
(setq skk-kanagaki-keyboard-type 'omelet-jis)
```

かな入力方式使用時の モードでは以下のコマンドなどが役に立ちます。

F1 1

かな入力方式での特殊キ一定義の一覧を表示します。

F1 2

かな入力方式でのかなキー配列を表示します。

F12

M-x skk-kanagaki-toggle-rom-kana

かな入力方式とローマ字入力方式とを切り換えます。

なお、親指シフト方式については NICOLA 日本語入力コンソーシアム (<http://nicola.sunicom.co.jp/>) が参考になります。

7 SKK に関する情報

7.1 最新情報

DDSKK についての最新情報は、

<http://openlab.jp/skk/>

から得ることができます。

SKK の開発は、CVS によってなされています。CVS を使って最新の SKK を取得する場合、CVS account を取得して開発に参加される場合は、以下を参照してください。

<http://openlab.jp/skk/cvs-ja.html>

最新版 DDSKK の変更内容と更に過去の変更点については以下のリソースを参照してください。

<http://openlab.jp/skk/skk/main/READMEs/NEWS.ja>

また、将来のバージョンにおける拡張アイディアについては、TODO としてまとめられています。

<http://openlab.jp/skk/skk/main/READMEs/TODO.ja>

SKK Openlab では、開発者、文章の整備にご協力いただける方、テスター、よろずものを言う人などなど、常に募集しています。また要望、拡張の具体的アイディアがあれば、メーリングリストに連絡いただけることを期待します。

See Section 7.2 [SKK メーリングリスト], page 96.

7.2 SKK メーリングリスト

SKK Openlab メーリングリストは、統一された一つの ML です。利用者用、開発者用などと分かれていない他、SKK 辞書、DDSKK の開発議論が中心ですが、辞書サーバやフロントエンド、SKK 辞書ツールの話題なども議論の範囲に入ります。

メーリングリストに参加する

アドレス skk-subscribe@ring.gr.jp 宛てに空のメールを送って下さい。確認の為のメッセージが指定されたアドレス宛に送信されます。その確認の為のメッセージに対して返信することで加入手続きは終了します。

メーリングリストから脱会する

アドレス skk-unsubscribe@ring.gr.jp 宛てに空のメールを送って下さい。確認の為のメッセージが指定されたアドレス宛に送信されます。その確認の為のメッセージに対して返信することで脱退手続きは終了します。

登録したアドレスを変更する

古いアドレスについていったん unsubscribe して、新しいアドレスから再度 subscribe して下さい。

記事の投稿

アドレス skk@ring.gr.jp へ送ります。メーリングリストに登録されている人全員にメールが配信されます。

過去ログの閲覧

<http://mail.ring.gr.jp/skk>

<news://news.ring.gr.jp/ring.openlab.skk>

7.3 SKK 関連ソフトウェア

SKK 関連ソフトウェアに関しては、以下の URL にリンクをまとめてありますので参照してください。

SKK 辞書 Wiki におけるリンク集 (<http://openlab.jp/skk/wiki/wiki.cgi?page=%A5%EA%A5%F3%A5%AF%B>)

7.4 SKK 辞書について

SKK 辞書は多くのユーザの方々から提供された辞書によりコピーフリーの辞書としては最大規模の辞書になっています。今後もこの方式により SKK 辞書をより充実したものにしていきたいと思えます。

<http://openlab.jp/skk/registdic.cgi> にて Web/cgi を利用した登録・削除希望フォームを運用しています。SKK 辞書に追加したい単語、誤登録として削除したい単語がありましたら、是非ご利用下さい。

7.5 辞書ツール

SKK 辞書に関するツールには、Perl, C, Ruby の各言語により書かれたツールがありますが、Perl によるツールは現在十分メンテナンスされていません。現在は C, Ruby のツールが開発・メンテナンスされています。

辞書メンテナンスツール (<http://openlab.jp/skk/wiki/wiki.cgi?page=%BC%AD%BD%F1%A5%E1%A5%F3%A5%>)

7.6 SKK の作者

SKK の原作者は、現京都大学の佐藤雅彦教授 (<http://www.sato.kuis.kyoto-u.ac.jp/~masahiko/masahiko->) です。

現在の DDSKK は、大勢のボランティアの貢献により成立しています。以下は確認されている貢献者名です。ここに入っていない貢献者がおられる場合、email address が変更になり credit をこれに合わせたい場合には、追加、修正いたしますので、ご遠慮なく SKK Openlab の管理人 (skk-owner@ring.gr.jp) までご一報ください。

- 中津山 恒 (hisashi@rst.fujixerox.co.jp)
- 小関 吉則 (kose@emailizard.tamra.co.jp)
- 小野 孝男 (takao@hirata.nuee.nagoya-u.ac.jp)
- 松井孝雄/MATUI Takao (mat@nuis.ac.jp)
- 進藤裕志 (shindo@super.ees.saitama-u.ac.jp)
- 沈志勇 (jshen@cas.org)
- 柘植 正大 (ma-tsuge@kdd.co.jp)
- Akihiro Motoki (amotoki@dd.iiij4u.or.jp)
- Chikanobu Toyofuku (unbound@papaya.juice.or.jp)
- Daiki Ueno (ueno@unixuser.org)
- EGUCHI Satoshi (eguchi@kyoto-wu.ac.jp)
- Eiji Obata (obata@suzuki.kuee.kyoto-u.ac.jp)
- FUKANO Masaaki (fukano-m@alp.cs.uec.ac.jp)
- Fukaya Shigeru (SFUKAYA@oracle.co.jp)
- Fumihiko MACHIDA (machida@users.sourceforge.jp)
- FURUE Hideyuki (furue@kke.co.jp)
- GUNJI Takao (gunji@sils.shoin.ac.jp)
- G. Yoshida (grothendieck@max.hi-ho.ne.jp)
- Haru Mizuno (mizu@cs3.cs.oki.co.jp)
- Haru'yasu Ueda (hal@sics.se)
- Hideki Sakurada (sakurada@kuis.kyoto-u.ac.jp)
- Hiroya Murata (lapis-lazuli@pop06.odn.ne.jp)
- Hiroyuki KUROSAKI (noir@st.rim.or.jp)

- Hisao Kuroda (kuroda@msi.co.jp)
- IIDA Yosiaki (iida@ring.gr.jp)
- IRIE Tetsuya (irie@t.email.ne.jp)
- ISHIKAWA Naoya (ml@be.to)
- Jun-ichi Nakamura (nakamura@pluto.ai.kyutech.ac.jp)
- Katuya Tomioka (tomioka@culle.l.chiba-u.ac.jp)
- Kazuo Hirokawa (hirokawa@rics.co.jp)
- Kazushi Marukawa (kazushi@kubota.co.jp)
- Keisuke Nishida (kxn30@po.cwru.edu)
- Kenichi OKADA (okada@opaopa.org)
- Kenji Rikitake (kenji@reseau.toyonaka.osaka.jp)
- Kenji Yamashita (kenji@cs.titech.ac.jp)
- Kentaro Fukuchi (fukuchi@users.sourceforge.net)
- KIHARA Hideto (deton@m1.interq.or.jp)
- Kimura Chikahiro (kimura@oa1.kb.nec.co.jp)
- Kimura Fuyuki (fuyuki@hadaly.org)
- Kitamoto Tsuyoshi (tsuyoshi.kitamoto@gmail.com)
- Koichi MORI (kmori@onsei2.rilp.m.u-tokyo.ac.jp)
- KOIE Hidetaka (鯉江英隆) (hide@koie.org)
- Koji Uchida (uchida@cf.d.tytlabs.co.jp)
- KURIHARA Kenichi (kenichi_kurihara@nifty.com)
- Makoto MATSUSHITA (matusita@ics.es.osaka-u.ac.jp)
- Manabu Kawashima (kaw@lp.nm.fujitsu.co.jp)
- Martin Buchholz (martin@xemacs.org)
- Masahiko Suzuki (suzmasa@sm.sony.co.jp)
- Masahiro Doteguchi (xdote@rp.open.cs.fujitsu.co.jp)
- Masakazu Takahashi (masaka-t@ascii.co.jp)
- Masatake YAMATO (masata-y@is.aist-nara.ac.jp)
- NAKAJIMA Mikio (minakaji@osaka.email.ne.jp)
- MINOURA Itsushi (minoura@uni.zool.s.u-tokyo.ac.jp)
- MITA Yuusuke (clefs@mail.goo.ne.jp)
- MIYOSHI Tsutomu (minkov@fuzzy.or.jp)
- Motohiko Mouri (mouri@jaist.ac.jp)
- Murata Shuuichirou (mrt@network.org)
- Naoki HAMADA (nao@mimo.jaist-east.ac.jp)
- Naoki Wakamatsu (naoki-w@ht-net21.ne.jp)
- NEMOTO Takashi (tnemoto@mvi.biglobe.ne.jp)
- NISHIDA Keisuke (knishida@nn.ij4u.or.jp)
- Norio Maeda (nmaeda@dd.ij4u.or.jp)
- ONODA Arata (onoto@ma.nma.ne.jp)
- Rei FURUKAWA (furukawa@tcp-ip.or.jp)
- Ryoichi Hashimoto (gnu@ipri.go.jp)
- SAKAI Kiyotaka (ksakai@kso.netwk.ntt-at.co.jp)

- Satoshi Harauchi (satoshi@sys.sdl.melco.co.jp)
- Seiichi NAMBA (sn@asahi-net.email.ne.jp)
- Sekita Daigo (sekita@mri.co.jp)
- Shigeru OKUMURA (shigeru@okunet.gr.jp)
- Shuhei KOBAYASHI (shuhei@aqua.ocn.ne.jp)
- Shuji Ashizawa (ashizawa@zuken.co.jp)
- SIMOYAMA Takasi (simo@festa.or.jp)
- SL Baur (steve@altair.xemacs.org)
- sphere (sphere@pop12.odn.ne.jp)
- Stephen Turnbull (turnbull@sk.tsukuba.ac.jp)
- SUZUKI Hitoshi (sigh@kuzirabekon.econ.nagasaki-u.ac.jp)
- TAKAHASHI Kaoru (kaoru@kaisei.org)
- Takao KAWAMURA (kawamura@ike.tottori-u.ac.jp)
- Takeshi OHTANI (ohtani@iiis.flab.fujitsu.co.jp)
- Tatsuya Kinoshita (tats@vega.ocn.ne.jp)
- Teika Kazura (teika@lavabit.com)
- Thierry Emery (Thierry.Emery@nmuc.alcatel.fr)
- TOKUYA Junichi (tokuya@crab.fuji-ric.co.jp)
- Tomotaka SUWA (cooper@saitama.fujimic.fujisankei-g.co.jp)
- Tomoyuki Hiro (hiro@momo.it.okayama-u.ac.jp)
- Toru YANO (toru@maid.ne.jp)
- Toyonobu Yoshida (toyono-y@is.aist-nara.ac.jp)
- Tsugutomo Enami (enami@ptgd.sony.co.jp)
- Tsukamoto Tetsuo (czkmt@remus.dti.ne.jp)
- TSUMURA Tomoaki (tsumura@kuis.kyoto-u.ac.jp)
- Tsuyoshi AKIHO (akiho@sky.email.ne.jp)
- Wataru Matsui (matsui@atr-rd.atr.co.jp)
- Wataru Matsui (matsui@gf.hm.rd.sanyo.co.jp)
- YAGI Tatsuya (ynyaaa@ybb.ne.jp)
- YAMASHITA Junji (ysjj@unixuser.org)
- Yano Keisuke (yano@moon.email.ne.jp)
- Yoshida Toyonobu (toyono-y@is.aist-nara.ac.jp)
- Yukihiro Nakai (ynakai@redhat.com)
- Yoshiki Hayashi (yoshiki@xemacs.org)
- YOSHIZAWA Masahiro (manbou@ceres.dti.ne.jp)
- Yuh Ohmura (yutopia@giga.ocn.ne.jp)
- Yu-ji Hosokawa (yu-ji@hoso.net)
- Yukiyoshi Kameyama (kameyama@kuis.kyoto-u.ac.jp)
- Yusuke Shinyama (yusuke@example.com)
- Kiyoka Nishiyama. (kiyoka@sumibi.org)

7.7 SKK の歴史

SKK の成り立ちと歴史に関しては以下の URL を参照してください。

SKK の誕生秘話 (<http://openlab.jp/skk/born-ja.html>)

“SKK = I” (<http://openlab.jp/skk/SKK.html>)

SKK の歴史 (付 Emacs の歴史の一部) (<http://openlab.jp/skk/history-ja.html>)

7.8 このマニュアルについて

本マニュアルは、SKK オープンラボの有志の貢献により、従来のマニュアルに加筆修正したものです。

7.9 謝辞

DDSKK の開発は、Ring Server Open Laboratory (<http://openlab.jp>) (オープンラボラトリ) に ‘SKK Openlab’ として参加する形で行われています。‘SKK Openlab’ は Ring から共有ディスク、CVS 及び ML の提供を受けています。オープンラボラトリの運営は、完全にボランティアにより行われております。Ring 並びにオープンラボラトリにかかわる皆さんに深く感謝いたします。

(以降の記載は、SKK の原作者、佐藤雅彦教授により記載された旧来のマニュアルのものです。が、歴史的意義を踏まえて、そのまま掲載します。)

SKK の設計方針は TAO/ELIS 上の日本語入力システム Kanzen の影響を受けています。Kanzen のデモを行ってくださり、また Kanzen を使う機会を与えてくださった NTT の竹内郁雄さんに感謝します。

第 1 版の辞書作成のための読みの入力を行ってくださった東北大学電気通信研究所佐藤研究室の安藤大君、猪岡美紀さん、奥川淳一君、佐々木昭彦君、佐藤克志君、山岸信寛君に感謝します。

SKK 辞書第 2, 3, 4, 5, 6, 7, 8 版作成のためのデータを提供してくださった方々に感謝します。

SKK 辞書第 6, 7 版作成にあたり協力してくださった高橋裕信氏に感謝します。

8 よくある質問とその回答 (FAQ)

これは SKK に対するよくある質問と、それに対する回答集です。

8.1 SKK のなぜなに

Q1-1 Daredevil SKK って SKK とは違うのですか？

SKK Openlab で開発、リリースされる SKK は、京大の佐藤先生が中心になって開発していた SKK と区別するために、‘Daredevil SKK’ と呼ぶことにしました。その略称は ‘DDSKK’ で、SKK Openlab で最初に ‘Daredevil SKK’ としてリリースされた version は 11.1 です (オリジナルの version を継承しました)。

なお、‘Daredevil’ の名前の採択は、開発陣の一人が講読している某ラジオ英会話講座の、ある日のスキット名が「Daredevil なんとか」で、その内容は「とにかくやってみよう。うぎゃあああ、やられたあ」というものでした。これがあまりに自分の開発ポリシーに合致していた、ということに由来します。

Q1-2 SKK はシンプルなのが長所だったのでは？

かような議論は 10 年来行われてきており、結論は出ていませんが、事実として現在まで開発が続けられています。

「シンプルな操作性の維持と多機能化・高機能化は両立できる」

というのが現在の開発陣の考えであるようです。

SKK が Simple Kana to Kanji conversion program の略である通り、かなを漢字に変換するルーチンの簡単さが SKK を定義付けています。その周辺の拡張に関する制約は基本的にはありません。

多機能化と言っても多くはユーザオプションによって無効にすることができますし、‘skk.el’ 本体が複雑化しないようにモジュール化されています。

Q1-3 DDSKK はどの Emacs で使えますか？

基本的には、GNU Emacs と Mule 機能付きの XEmacs で使えます。

対応する Emacs のバージョンについては以下をご覧ください。

See Section 1.1 [このバージョンの SKK について], page 1.

Q1-4 DDSKK はどんなオペレーティングシステムで使えますか？

SKK がサポートしている Emacs がその OS で動いているなら、SKK の基本的な機能は動くはずです。Microsoft Windows でも Apple Mac OS X でも使えます。

拡張機能については、UNIX の各種コマンドを前提としているものがいくつかあります (look や ispell など)。これらのコマンドがお使いの OS にも存在すれば該当の拡張機能も基本的には使えるでしょう。

Apple Mac OS X 版 Emacs に特化した情報については、以下のファイルを参照してください。

<http://openlab.jp/skk/skk/main/READMEs/README.MacOSX.ja>

Q1-5 APEL って何？ 必要ですか？

APEL は A Portable Emacs Library の略です。APEL の主な機能は異なる Emacs 間の非互換性を吸収することです。

GNU Emacs 21 及び XEmacs では APEL が必要です。

GNU Emacs 22 以上では APEL は不要となりました。この変更は 2010 年 9 月に CVS に commit され、2011 年 1 月に DDSKK 14.2 としてリリースされました。

8.2 SKK の入手から導入まで

Q2-1 SKK を使うのに何が必要ですか？

SKK 本体と SKK 辞書が必要です。オプションで辞書サーバを用意することができます。GNU Emacs 21 及び XEmacs では事前に APEL をインストールしてください。

See Section 2.1 [APEL のインストール], page 3.

SKK 本体は以下から入手できます。

<http://openlab.jp/skk/maintrunk>

Q2-2 SKK 辞書はどこにありますか？

以下を参照してください。

See Section 7.4 [SKK 辞書について], page 97.

Q2-3 SKK サーバはどこにありますか？

DDSKK 14 は辞書サーバの種類、バージョンには依存していません。

<http://openlab.jp/skk/skkserv-ja.html>

からお好きな辞書サーバを入手して下さい。

8.3 SKK の基本設定からお好みのカスタマイズまで

Q3-1 「`、`」「`、`」が入力できるようにカスタマイズしたいのですが。

3通りの方法を紹介します。

1. 通常 '`、`' で '`、`','`、`' で '`、`」を入力したい場合

'`~/skk`' に以下を設定します。

```
(setq skk-kutouten-type 'en)
```

2. 一時的に '`、`' で '`、`','`、`' で '`、`」を入力したい場合

`M-x skk-toggle-kutouten` を実行すると、その場で '`、`」「`、`」に切り替えることができます。「`、`」「`、`」に戻すには、もう一度 `M-x skk-toggle-kutouten` を実行を実行します。特定のバッファでのみ '`、`」「`、`」に切り替えたい場合は、File Variables (see Section “File Variables” in *GNU Emacs Manual*) を参照下さい。例えば、`tex` モードでのみ '`、`」「`、`」に切り替えたい場合は、つぎの設定を `tex` ファイルの最後に追加します。

```
% Local Variables:
% skk-kutouten-type: en
% end:
```

3. 常に '`、`' で '`、`','`、`' で '`、`」を入力したい場合

`skk-rom-kana-rule-list` を直接変更します。なお、この設定をすると、`M-x skk-toggle-kutouten` での切り替えが効かなくなるので、注意して下さい。

'`~/skk`' に以下を追加します。

```
(setq skk-rom-kana-rule-list
      (append '(("." nil ".") ("," nil ","))
              skk-rom-kana-rule-list))
```

この設定方法は応用が効き、細かく制御することが可能です。「`、`」と '`、`' のところをそれぞれ、「`、`」と '`、`' とすることで、「かなモード」「カナモード」でも、「`、`」と '`、`' を直接入力することができます。

Q3-2 「ゐ」や「𛄁」が入力できるようにカスタマイズしたいのですが。

一つ前の Q の変形問題ですね。かな/カナモードでそれぞれ出力する文字を変えるやり方です。

‘~/skk’ に

```
(setq skk-rom-kana-rule-list
      (append '(("wi" nil ("𛄁" . "ゐ"))
              skk-rom-kana-rule-list))
```

と書いてみましょう。

一番内側の cons cell は car がカナモード、cdr がかなモードでの入力文字を表しています。

一つ前の Q に対する答えのように、カナモード、かなモードともに入力する文字が変わらなければ、cons cell の代わりに文字列を書くことができます。

Q3-3 検索する辞書を増やしたいのですが。

skk-search-prog-list で設定をしましょう。

まず、現在の設定を確認しましょうね。*scratch* バッファに skk-search-prog-list と書いてそのシンボルの末尾にポイントを置いて C-j してみましょう。例えば次のように出力されます。

```
((skk-search-jisyo-file skk-jisyo 0 t)
 (skk-search-server skk-aux-large-jisyo 10000))
```

上記の例は 2 つの要素を持ったリストになっています。設定によりもっと多くの要素があるかもしれません。

各要素は検索する関数と辞書を指定したリストです。要素の順番に検索がなされます。上記の例だとまず最初に skk-jisyo (個人辞書) を skk-search-jisyo という関数を使ってリニアサーチ、次に skk-search-server という関数を使って skk-aux-large-jisyo をサーチします。

変換の際、SPC を押しますよね? 1 回 SPC を押すと、SKK は候補が見つかるまでの間、skk-search-prog-list の要素を前から読んでいって検索を行い、見つければそこでいったん検索を止めてユーザに候補を提示します。

ユーザが SPC を更に押してゆき最初の要素のプログラムが見つけた候補が尽きると、SKK は中断していた個所から再び skk-search-prog-list の次の要素を見つけ、ここで指定されている関数を使って検索する、で新しい候補が見つければまた提示する、というシステムになっています。

では、辞書サーバを使って検索した後に、JIS 第 2 水準の単漢字辞書、‘SKK-JISYO.JIS2’を検索したい場合はどうすれば良いでしょうか? もうわかりますよね? 辞書サーバを使った検索式の次に第 2 水準辞書の検索式を書いたリストを skk-search-prog-list に指定すれば良いのです。‘~/skk’ に次のように書きましょう。

```
(setq skk-search-prog-list
      '((skk-search-jisyo-file skk-jisyo 0 t)
        (skk-search-server skk-aux-large-jisyo 10000)
        (skk-search-jisyo-file "~/dic/SKK-JISYO.JIS2" 0)))
```

skk-search-jisyo-file の第 2 引数、0 の数字はリニアサーチにて検索するよう指定しています。第 2 水準辞書はあまり大きくないので、リニアサーチで十分でしょう。大きな辞書を検索する場合などは、

```
(skk-search-jisyo-file "~/dic/SKK-JISYO.L" 10000)
```

のようにすると良いでしょう。SKK は Emacs のバッファに読み込まれた ‘~/dic/SKK-JISYO.L’ の検索リージョンのポイント差が 10,000 未満になるまではバイナリサーチを行い、その後リニアサーチを行います。大きな辞書ではバイナリサーチを行う方がはるかに効率が良いです。嘘だと思ふなら、‘SKK-JISYO.L’ を読み込んでリニアサーチするような設定にして試してみてください。

ちなみに、‘SKK-JISYO.JIS2’ は、最大でもリージョン間のポイント差が 8,500 程度です。

Q3-4 左手の小指を SHIFT で酷使したくありません。

SKK を標準の状態ですべて使っている場合、変換のためにシフトキーを多用しますので小指への負担が大きくなります。¹

この苦しみを回避するためにここでは 4 つの方法を紹介します。

1. 親指の近くにあるキーを利用してシフトキーの代用とする。

日本語 106 キーボードのように無変換、変換などのキーがある場合は、これらをシフトキーの代用とすることが可能です。こうすると、例えば

SHIFT を押しながら a を押す

というキー操作は

無変換 を押して、その後で a を押す

という操作で置き換えることができるようになります。

それでは具体的なやり方を説明しましょう。まず、使用中の Emacs が無変換キーを何という名前で認識しているか調べます。それには

```
M-x describe-key
```

というコマンドを実行し、続いて 無変換キーを押してみます。XFree86 上でなら、おそらく

```
muhenkan is undefined
```

という答えが返ってくるでしょう。次に、この名前を使って '~/.emacs' に設定を書きこみます。以下は 無変換 = muhenkan の場合の例です。

```
(unless (keymapp key-translation-map)
  (setq key-translation-map (make-sparse-keymap)))

(let ((i ?a))
  (while (<= i ?z)
    (define-key key-translation-map
      (vector 'muhenkan i) (vector (- i 32)))
    (setq i (1+ i))))
```

この設定を終えると、*muhenkan-a* で A が入力できるようになります。続いて SKK を起動してみましょう。*muhenkan-a* で ' あ' となります。送りの開始点も、もちろん同様の操作で指定できます。²

2. xmodmap を使う。

X Window System 上では、xmodmap というプログラムを使ってキー配列を変更できます。例えば、無変換キーをシフトキーとして使いたければ

```
% xmodmap -e 'add Shift = Muhenkan'
```

とします。これで無変換キーは通常のシフトキーと同じような感じで使えるようになります。

3. 'skk-sticky.el' を使う。

See Section 5.6.9 [変換位置の指定方法], page 54.

4. 親指シフト入力のエミュレーション機能を利用する。

これは 1, 2 とはかなり違ったアプローチです。SKK 本来のローマ字的入力を捨てて、富士通のワープロ OASYS のような親指シフト入力を修得します。³

¹ このため、ある人々は SKK を小指キラーと呼びます。

² 変数 `key-translation-map` の意味を調べてみてください。

M-x describe-variable RET `key-translation-map`

³ 親指シフト入力の詳細については、ここでは述べません。興味がある場合は、日本語入力コンソーシアムの Web サイト

<http://nicola.sunicom.co.jp/>
を訪れてください。

DDSKK には NICOLA-DDSKK というプログラムが付属しており、これをインストールすると親指シフト入力が可能になります。インストール自体は簡単で、

```
% cd nicola
% make install
```

とした後に、`~/skk` に

```
(setq skk-use-kana-keyboard t)
(setq skk-kanagaki-keyboard-type 'omelet-jis)
```

と書くだけです。詳しいことは、NICOLA-DDSKK 付属のドキュメントを参照してください。NICOLA 配列は、特別に日本語入力のために考えられた配列なので、慣れれば非常に効率的な日本語入力ができるようになりますと期待されます。一方で、ローマ字的入力方式に慣れてしまっている人にとっては、NICOLA 配列に慣れるまでかなり練習を要することは確かです。

Q3-5 全く漢字が出てきません。

恐らく辞書の設定ができていないのでしょう。

`'SKK-JISYO.L'` というファイルがインストールされている場所を確認してください。普通は

```
'/usr/local/share/skk'
'/usr/share/skk'
```

といった場所にインストールされています。XEmacs のパッケージならば

```
'/usr/local/lib/xemacs/mule-packages/etc/skk'
```

などを確認します。その後で `~/skk` に

```
(setq skk-large-jisyo "/usr/local/share/skk/SKK-JISYO.L")
```

のように設定します。

なお、辞書サーバを使っている場合はこの設定は必要ありません。その場合は、辞書サーバの設定や、それがちゃんと起動しているかどうかを確認してください。

また、どこにも辞書がインストールされていない場合は

```
http://openlab.jp/skk/dic/
```

から取得します。

Q3-6 チュートリアルが起動できません。

`'SKK.tut'` というファイルがインストールされている場所を確認してください。普通は

```
'/usr/local/share/skk'
'/usr/share/skk'
```

といった場所にインストールされています。XEmacs のパッケージならば

```
'/usr/local/lib/xemacs/mule-packages/etc/skk'
```

などを確認します。その後で `~/emacsk` に

```
(setq skk-tut-file "/usr/local/share/skk/SKK.tut")
```

のように設定します。

8.4 SKK 辞書関連

Q4-1 SKK には郵便番号辞書がありますか?

CVS から辞書を取得した場合は、`'zipcode'` というディレクトリに入っています。WWW では、

```
http://openlab.jp/skk/dic/
```

より入手できます。使用方法は

```
http://openlab.jp/skk/skk/dic/zipcode/README.ja
```

を御覧下さい。

Q4-2 SKK の辞書には、品詞情報がないんですね。

SKK は漢字とかなとの区切りをユーザが指定する方式により、品詞情報を使った解析を用いることなく効率的入力ができます。

TODO としては、辞書に品詞情報を持たせることで更なる入力の効率化ができるという提案がなされており、そのような辞書の作成が既に試みられています。興味のある方は

<http://openlab.jp/skk/wiki/wiki.cgi?page=SKK%BC%AD%BD%F1>

における ‘SKK-JISY0.notes’ の項目をご覧ください。

Q4-3 複数の SKK 辞書を結合できますか？

SKK 本体のパッケージには同封されていませんが、skk-tools という別パッケージがあります。以下をご覧ください。

See Section 7.5 [辞書ツール], page 97.

Q4-4 SKK 形式の英和辞書があると聞いたのですが。

edict は和英辞書ですが、これを SKK 辞書形式の英和辞書に変換したものを

<http://openlab.jp/skk/dic/SKK-JISY0.edict>

として置いています。これは edict を単純に機械的に変換した後、バグの修正や、エントリ・候補の追加が SKK Openlab で独自に行われているものです。

edict を自分で加工して上記と同等のものを作成することもできます。edict は

<ftp://ftp.u-aizu.ac.jp:/pub/SciEng/nihongo/ftp.cc.monash.edu.au/>

などから入手できます。

加工には日本語の通る gawk と skk-tools 中のプログラムを使い、下記のように行います。

```
% jgawk -f edict2skk.awk edict > temp
% skkdic-expr temp | skkdic-sort > SKK-JISY0.E2J
% rm temp
```

できた ‘SKK-JISY0.E2J’ の利用方法は色々ありますが、

```
% skkdic-expr SKK-JISY0.E2J + /usr/local/share/skk/SKK-JISY0.L | \
skkdic-sort > SKK-JISY0.L
```

などとして、‘SKK-JISY0.L’ とマージして使うのが手軽です。

なお、edict の配布条件は GNU GPL (General Public License) ではありません。

<http://www.csse.monash.edu.au/groups/edrdg/newlic.html>

をご覧ください。‘SKK-JISY0.edict’ のヘッダー部分にもそのダイジェストが記載されています。

8.5 SKK の活用法その他**Q5-1 SKK abbrev モードでもっと英単語を利用した変換ができませんか？**

UNIX look コマンドと ‘skk-look.el’ を利用すると、色々できますよ。まず、‘~/skk’ で skk-use-look を t にセットして Emacs/SKK を立ち上げ直して下さい。

さあ、下記のような芸当が可能になりました。

1. 英単語の補完ができます。

```
abstr(TAB) ⇨ abstract
```

通常の補完機能と同様に、. で次の補完候補に、, でひとつ前の補完候補に移動できます。SKK 形式の英和辞書があれば、ここから SPC を押して英和変換ができますね。また、skk-look-use-ispell の値が non-nil であれば、look で検索する前に ispell でスペルチェック・修正をします。

2. 英単語をあいまいに変換して取り出すことができます。上記同様、`skk-look-use-ispell` の値が `non-nil` であれば、`look` で検索する前に `ispell` でスペルチェック・修正をします。

```
abstr* (SPC) ⇨ abstract
```

見出し語に ‘*’ を入れるのをお忘れなく。

3. あいまいに変換した後、更に再帰的な英和変換を行うことができます。

まず、`skk-look-recursive-search` の値を `non-nil` にセットして下さい。Emacs/SKK を再起動する必要はありません。すると、例えば、

```
abstr* (SPC)
⇨ abstract (SPC)
⇨ アブストラクト (SPC)
⇨ 抽象 (SPC)
⇨ abstraction (SPC)
⇨ アブストラクション
```

このように英単語 + その英単語を見出し語にした候補の「セット」を変換結果として出力することができます。

この際、`skk-look-expanded-word-only` の値が `non-nil` であれば、再帰検索に成功した英単語の「セット」だけを出力することができます (再帰検索で検出されなかった英単語は無視して出力しません)。

もちろん、SKK 辞書に

```
abstract /アブストラクト/抽象/
abstraction /アブストラクション/
```

というエントリがあることを前提としています。edict を SKK 辞書形式に変換すると良いですね。

なお、‘`skk-look.el`’ を使った補完・変換が期待するスピードよりも遅い、補完・変換で余分な候補が出る、とお感じの貴方は、`skk-look-use-ispell` の値を `nil` にして `ispell` によるスペルチェック・修正をオフにしてお試し下さい。

Q5-2 市販の CD-ROM 辞書やネットワークの辞書サーバが利用できますか？

Lookup が扱える辞書はほとんど使えます。Lookup がインストールされている状態で SKK をインストールすると、SKK と Lookup のゲートウェイプログラム ‘`skk-lookup.el`’ がインストールされます。

インストールで注意すべきは、`make` で呼び出される Emacs は ‘`-q -no-site-file`’ フラグ付きで呼ばれるので、‘`~/emacs`’ や ‘`site-start.el`’ などは読み込まれないことです。デフォルトで `load-path` の通っているディレクトリに `lookup` をインストールするか、‘`SKK-CFG`’ の中で `VERSION_SPECIFIC_LISPDIR` などにディレクトリを明示することで解決できます。

さあ、‘`~/skk`’ で `skk-search-prog-list` の要素に (`skk-lookup-search`) を追加しましょう。他の検索エンジンよりも検索は比較的遅いので、最後の方が良いと思います。

こんな感じです。

```
(setq skk-search-prog-list
      '((skk-search-jisyo-file skk-jisyo 0 t)
        (skk-search-server skk-aux-large-jisyo 10000)
        (skk-lookup-search)))
```

Lookup については、

<http://openlab.jp/edict/lookup/>

をご参照下さい。

Q5-3 他の FEP を使用中にも SHIFT を押してしまいます。

治すには SKK をやめるしかありません :-)

Emacs 上以外でも SKK みたいな操作性を実現するソフトウェアがあります。Section 7.3 [SKK 関連ソフトウェア], page 96 をご覧になってください。

事項索引

‘*SKK-JISYO.L*’ 76

#

#0 39
 #1 39
 #2 39
 #3 39
 #4 39
 #5 39
 #9 39

;

;; okuri-ari entries 72
 ;; okuri-nasi entries 72

~

‘~/emacs’ 22
 ‘~/emacs.d/init.el’ 22
 ‘~/skk’ 22, 92
 ‘~/xemacs/init.el’ 22

A

ACT 94
 APEL 3
 Auto Fill 9
 AZIK 94

B

BS 93

C

‘canna.el’ 89
 ‘ccc.el’ 19
 ‘context-skk.el’ 19, 27

D

‘dabbrev.el’ 76

E

edict 106
 ‘edict2skk.awk’ 106
 EUC コード 80, 81
 Extents 89

I

I-search 17, 92
 Incremental regexp search 92
 Incremental search 17, 92
 ‘isearch.el’ 17

J

JIS コード 80, 81

K

‘keyboard.c’ 55

L

L 辞書 67
 ‘leim-list.el’ 19
 look 106
 Lookup 82, 107

M

M 辞書 67
 Menu Bars 26, 75
 move-to-column 93
 move-to-column-force 93

N

NICOLA 94, 104

O

OASYS 104
 Overlays 13, 89

P

picture-mode 93
 ‘picture.el’ 93

Q

‘queue-m.el’ 21

R

‘rgb.txt’ 89

S

S 辞書 67
 SKK abbrev mode 41
 ‘skk-abbrev.el’ 19
 ‘skk-act.el’ 19
 ‘skk-annotation.el’ 19
 ‘skk-auto.el’ 19
 ‘skk-autoloads.el’ 19
 ‘skk-azik.el’ 19
 ‘skk-bayesian.el’ 19
 ‘skk-cdb.el’ 19
 ‘skk-comp.el’ 19
 ‘skk-cursor.el’ 19
 ‘skk-cus.el’ 20
 ‘skk-dcomp.el’ 20, 30

'skk-develop.el' 20
 'skk-emacs.el' 20
 'skk-gadget.el' 20, 42
 'skk-hint.el' 20, 36
 'skk-icon' 90
 'skk-inline.el' 20
 'skk-isearch.el' 20
 'skk-jisx0201.el' 20
 'skk-jisx0213.el' 20
 'skk-jisyo-edit-mode.el' 20
 'SKK-JISYO.L' 67
 'SKK-JISYO.M' 67
 'SKK-JISYO.ML' 67
 'SKK-JISYO.S' 67
 'skk-kakasi.el' 20
 'skk-kanagaki.el' 20
 'skk-kcode.el' 20
 'skk-leim.el' 20
 'skk-look.el' 21, 82, 106
 'skk-lookup.el' 21, 82, 107
 'skk-macs.el' 21
 'skk-num.el' 21
 'skk-server-completion.el' 21
 'skk-server.el' 21
 'skk-setup.el' 21
 'skk-sticky.el' 21, 54
 'skk-study.el' 21, 65
 'skk-tankan.el' 21, 33
 'skk-tut.el' 21
 'skk-tutcode.el' 21
 'skk-vars.el' 19
 'skk-version.el' 21
 'skk-viper.el' 21
 'skk-xemacs.el' 21
 'skk.el' 19
 SKK_JISYO 8
 skkdic-expr 106
 skkdic-sort 106
 SKKSERV 8
 SKKSERVER 8

T

Text Properties 89
 TUT-code 94

V

VIP 92
 'vip.el' 92
 VIPER 92
 'viper.el' 92

X

xmodmap 104

モード 11

モード 11

あ

アスキーモード 10

い

インストール 3

え

エントリ 72

お

オートフィル 9

か

かなモード 10
 カナモード 10
 かな入力 94

き

キー設定 46

さ

サ変動詞の辞書登録に関する注意 16

ち

チュートリアル 18, 105

で

トグル変換 24

は

ハイライト 13
 はじめに 1
 パッケージ 105

ふ

プログラム実行変換 41

め

メニューバー 26, 75

ろ

ローマ字プレフィックス 72
 ローマ字入力 11

暗

暗黙の確定 13, 15, 58, 60

英

英単語の検索	106
英和辞書	106

改

改行文字を含む文字列の辞書登録	17
-----------------	----

確

確定アンドウ	56
確定辞書	60
確定入力	11
確定入力モード	11
確定変換	60, 70

漢

漢数字	39
-----	----

逆

逆引き	26
-----	----

共

共有辞書	67
------	----

金

金額	39
----	----

見

見出し語の補完	28
---------	----

個

個人辞書	67, 74
個人辞書エントリの削除	73
個人辞書のオートセーブ	74

後

後から モードに入る方法	11, 56
--------------	--------

誤

誤登録	73
-----	----

再

再帰的辞書登録	16
再変換	56

辞

辞書のソート方法	73
辞書のマージ	106
辞書パッファの名付け規則	76
辞書登録	15

辞書変換対象の文字列の決定	11
---------------	----

親

親指シフト	94
親指シフト入力	104

数

数をパラメータとする語の変換	39
数字から始まる見出し語の入力	12
数値再変換	39

接

接頭辞	37
接尾辞	37

全

全英モード	10
-------	----

送

送りありエントリ	72
送りあり変換	72
送りなしエントリ	72
送りなし変換	72

大

大字	39
----	----

単

単漢字	33
-----	----

直

直線的検索	70
-------	----

読

読みの補完	28
-------	----

二

二分検索	70
------	----

日

日本語入力コンソーシアム	104
--------------	-----

品

品詞情報	106
------	-----

文

文脈に応じた自動モード切り替え	27
-----------------	----

变

变换開始 13

補

補完 28

和

和英变换 106

变数索引

B

buffer-file-name 77

I

isearch-mode-end-hook 7
isearch-mode-hook 7

K

key-translation-map 104

M

major-mode 76
minibuffer-exit-hook 61
minibuffer-setup-hook 61
mode-name 76

S

skk-abbrev-mode-string 86
skk-act-load-hook 23
skk-allow-spaces-newlines-and-tabs 45, 56
skk-annotation-copy-key 77
skk-annotation-function 78
skk-annotation-show-as-message 78
skk-annotation-show-wikipedia-url 79
skk-annotation-toggle-display-char 78
skk-annotation-wikipedia-sources 79
skk-auto-fill-mode-hook 23
skk-auto-insert-paren 49
skk-auto-load-hook 23
skk-auto-okuri-process 57, 62, 65, 73
skk-auto-paren-string-alist 50
skk-auto-start-henkan 58
skk-auto-start-henkan-keyword-list 58
skk-aux-large-jisyo 68
skk-azik-keyboard-type 94
skk-azik-load-hook 23
skk-backup-jisyo 10, 68
skk-byte-compile-init-file 23
skk-candidate-buffer-background-color 88
skk-cdb-large-jisyo 7, 68
skk-check-okurigana-on-touroku 16
skk-comp-circulate 29
skk-comp-load-hook 23
skk-comp-are-jisyo-size-when-saving 75
skk-completion-search-char 85
skk-count-private-jisyo-candidates-exactly
..... 75
skk-cursor-abbrev-color 87
skk-cursor-default-color 87
skk-cursor-hiragana-color 87
skk-cursor-jisx0201-color 87
skk-cursor-jisx0208-latin-color 87
skk-cursor-katakana-color 87
skk-cursor-latin-color 87
skk-date-ad 41
skk-dcomp-activate 32skk-dcomp-face 32
skk-dcomp-multiple-activate 32
skk-dcomp-multiple-face 32
skk-dcomp-multiple-rows 32
skk-dcomp-multiple-selected-face 33
skk-dcomp-multiple-trailing-face 32
skk-delete-implies-kakutei 53
skk-delete-okuri-when-quit 54
skk-display-code-char-face 82
skk-display-code-prompt-face 82
skk-echo 86
skk-egg-like-newline 52
skk-emacs-abbrev-face 90
skk-emacs-hiragana-face 90
skk-emacs-jisx0201-face 90
skk-emacs-jisx0208-latin-face 90
skk-emacs-katakana-face 90
skk-extra-jisyo-file-list 68
skk-force-registration-mode-char 73
skk-gadget-load-hook 23
skk-gyakubiki-jisyo-list 27
skk-henkan-face 89
skk-henkan-okuri-strictly 61, 62, 65, 73
skk-henkan-rest-indicator 51
skk-henkan-rest-indicator-face 52
skk-henkan-show-candidates-keys 51
skk-henkan-show-candidates-keys-face 51
skk-henkan-strict-okuri-precedence 62
skk-hiragana-mode-string 86
skk-indicator-use-cursor-color 90
skk-inhibit-ja-dic-search 70
skk-initial-search-jisyo 67, 68
skk-inline-show-background-color 88
skk-inline-show-face 87
skk-isearch-start-mode 92
skk-isearch-use-previous-mode 92
skk-isearch-whitespace-regexp 93
skk-j-mode-function-key-usage 46, 91
skk-japanese-message-and-error 91
skk-jisx0208-latin-mode-string 86
skk-jisyo 10, 67, 68
skk-jisyo-code 76
skk-jisyo-fix-order 67
skk-jisyo-save-count 75
skk-kakasi-load-hook 23
skk-kakutei-early 59, 65
skk-kakutei-henkan-flag 70
skk-kakutei-jisyo 60, 67, 68
skk-kakutei-key 51
skk-kakutei-search-prog-limit 59
skk-kakutei-when-unique-candidate 59
skk-kana-input-search-function 47
skk-kana-rom-vector 64
skk-kanagaki-keyboard-type 95
skk-katakana-mode-string 86
skk-kcode-load-hook 23
skk-keep-record 75
skk-kutouten-type 48
skk-large-jisyo 68, 76
skk-latin-mode-string 86
skk-load-hook 23

- skk-look-expanded-word-only 83, 84, 106
 skk-look-recursive-search 83, 106
 skk-look-use-ispell 106
 skk-lookup-search-agents 82
 skk-mode-hook 23
 skk-next-completion-char 29
 skk-num-convert-float 40
 skk-num-load-hook 23
 skk-number-style 41
 skk-prefix-hiragana-face 86
 skk-prefix-jisx0201-face 86
 skk-prefix-katakana-face 86
 skk-preload 9
 skk-previous-candidate-keys 14
 skk-previous-completion-backtab-key 29
 skk-previous-completion-char 29
 skk-previous-completion-use-backtab 29
 skk-process-okuri-early 58, 61, 62, 65, 73
 skk-record-file 75
 skk-rom-kana-base-rule-list 46
 skk-rom-kana-rule-list 46, 50
 skk-romaji-*-by-hepburn 27
 skk-save-jisyo-instantly 74, 75
 skk-search-excluding-word-pattern-function
 83
 skk-search-katakana 45
 skk-search-prog-list 69, 103, 107
 skk-search-sagyō-henkaku 45
 skk-self-insert-non-undo-count 55
 skk-server-host 8
 skk-server-inhibit-startup-server 8, 71
 skk-server-jisyo 8
 skk-server-load-hook 23
 skk-server-portnum 8
 skk-server-prog 8
 skk-server-remote-shell-program 71
 skk-server-report-response 71
 skk-servers-list 71
 skk-share-private-jisyo 75
 skk-show-annotation 77, 91
 skk-show-candidates-always-pop-to-buffer 88
 skk-show-candidates-toggle-display-place-char
 88
 skk-show-icon 90
 skk-show-inline 87
 skk-show-japanese-menu 91
 skk-show-tooltip 88
 skk-special-midashi-char-list 39
 skk-status-indicator 9
 skk-sticky-double-interval 55
 skk-study-associates-number 65
 skk-study-backup-file 66
 skk-study-check-alist-format 66
 skk-study-file 66
 skk-study-first-candidate 66
 skk-study-max-distance 66
 skk-study-search-times 65
 skk-study-sort-saving 66
 skk-tankan-face 36
 skk-tankan-search-key 34
 skk-tooltip-face 88
 skk-tooltip-mouse-behavior 88
 skk-treat-candidate-appearance-function 89
 skk-try-completion-char 29
 skk-tut-file 18, 105
 skk-undo-kakutei-return-previous-point 57
 skk-units-alist 43
 skk-use-act 94
 skk-use-azik 94
 skk-use-color-cursor 87
 skk-use-face 89
 skk-use-kana-keyboard 95
 skk-use-look 82, 106
 skk-use-numeric-conversion 40
 skk-use-viper 92
 skk-user-directory 22
 skk-verbose 91
 skk-verbose-message-interval 92
 skk-verbose-wait 92
 skk-xemacs-abbrev-face 90
 skk-xemacs-hiragana-face 90
 skk-xemacs-jisx0201-face 90
 skk-xemacs-jisx0208-latin-face 90
 skk-xemacs-katakana-face 90
 skk-xemacs-latin-face 90
- T**
- tex-mode-hook 51
- Y**
- yatex-mode 50

関数索引

C

convert-standard-filename 22

E

eval-after-load 23

F

fundamental-mode 76

I

isearch-backward-regexp 93

isearch-forward-regexp 93

S

save-some-buffers 77

self-insert-command 55

skk-abbrev-comma 55

skk-abbrev-period 55

skk-annotation-add 78

skk-annotation-kill 78

skk-annotation-remove 78

skk-backward-and-set-henkan-point 56

skk-calc 43

skk-count-jisyo-candidates 75

skk-gadget-units-conversion 43

skk-gyakubiki-and-henkan 26

skk-gyakubiki-katakana-message 27

skk-gyakubiki-katakana-region 26

skk-gyakubiki-message 27

skk-gyakubiki-region 26

skk-hiragana-region 26

skk-hurigana-katakana-message 27

skk-hurigana-katakana-region 26

skk-hurigana-message 27

skk-hurigana-region 26

skk-insert 55

skk-jisx0208-latin-insert 55

skk-jisx0208-latin-region 26

skk-kana-input 55

skk-katakana-region 26

skk-kill-emacs-without-saving-jisyo 10

skk-latin-region 26

skk-lookup-search 82, 107

skk-make-face 89

skk-okuri-search 70

skk-restart 9

skk-romaji-message 27

skk-romaji-region 27

skk-search-cdb-jisyo 70

skk-search-ja-dic 71

skk-search-jisyo-file 70

skk-search-kakutei-jisyo-file 60, 70

skk-search-lisp-symbol 85

skk-search-server 70

skk-set-henkan-point 55

skk-tankan-search 34

skk-tutorial 18

コ

コマンド 9, 78, 79, 80

キー索引

\$

\$ 81

,

',' 28, 32
 , 106

.

.' 28, 32
 106

/

/ 25

@

@ 41

\

\ 80

B

BS 52

C

C-f 88

C-g 12, 53

C-j 12, 51

C-q 25

C-q C-j 17

C-r 18

C-s 18

C-u -1 C-x j 9

C-u C-r 93

C-u C-s 93

C-u C-x j 9

C-u TAB 29

C-x C-c 74

C-x C-j 9, 65

C-x j 9, 65

F

F1 1 95

F1 2 95

F12 95

M

M-- C-x j 9

M-1 C-x j 9

M-C-r 18, 93

M-C-s 18, 93

M-Q 56

M-SPC 29

M-x customize-group 24

M-x isearch-backward-regexp 93

M-x isearch-forward-regexp 93

M-x skk-count-jisyo-candidates 75

M-x skk-customize 24

M-x skk-gyakubiki-and-henkan 26

M-x skk-gyakubiki-katakana-message 27

M-x skk-gyakubiki-katakana-region 26

M-x skk-gyakubiki-message 27

M-x skk-gyakubiki-region 26

M-x skk-hiragana-region 26

M-x skk-hurigana-katakana-message 27

M-x skk-hurigana-katakana-region 26

M-x skk-hurigana-message 27

M-x skk-hurigana-region 26, 27

M-x skk-jisx0208-latin-region 26

M-x skk-kanagaki-toggle-rom-kana 95

M-x skk-katakana-region 26

M-x skk-latin-region 26

M-x skk-romaji-message 27

M-x skk-study-copy-theme 67

M-x skk-study-remove-theme 67

M-x skk-study-switch-current-theme 67

M-x skk-tutorial 18

Q

Q 11

q 24

T

TAB 28, 32

Short Contents

1	はじめに.....	1
2	インストール.....	3
3	はじめの設定.....	7
4	基本的な使い方.....	9
5	便利な応用機能.....	19
6	ローマ字入力以外の入力方式.....	94
7	SKKに関する情報.....	96
8	よくある質問とその回答 (FAQ).....	101
	事項索引.....	109
	変数索引.....	113
	関数索引.....	115
	キー索引.....	116

Table of Contents

1	はじめに	1
1.1	このバージョンの SKK について	1
1.2	SKK とはなにか	2
2	インストール	3
2.1	APEL のインストール	3
2.2	DDSKK のインストール	3
2.2.1	GNU Emacs へのインストール	3
2.2.2	XEmacs へのインストール	4
2.2.3	対話的なインストール	4
2.3	辞書について	4
2.4	辞書の入手	5
2.5	辞書を DDSKK と同時にインストールする	5
2.6	辞書サーバの入手	6
3	はじめの設定	7
3.1	最も基本的な設定	7
3.2	インクリメント検索の設定	7
3.3	辞書サーバを使いたいときの設定	8
4	基本的な使い方	9
4.1	起動と終了	9
4.1.1	SKK オートフィルモード	9
4.1.2	辞書の保存	10
4.2	入力モード	10
4.2.1	入力モードの説明	10
4.2.2	入力モードを切り替えるキー	10
4.3	変換モード	11
4.3.1	モード	11
4.3.2	モード	11
4.3.2.1	後から モードに入る方法	11
4.3.2.2	モードを抜ける方法	12
4.3.3	モード	13
4.3.3.1	送り仮名が無い場合	13
4.3.3.2	次候補・前候補	13
4.3.3.3	送り仮名が有る場合	14
4.3.4	辞書登録モード	15
4.3.4.1	送り仮名が無い場合の辞書登録	15
4.3.4.2	送り仮名が有る場合の辞書登録	16
4.3.4.3	サ変動詞の辞書登録に関する注意	16
4.3.4.4	再帰的辞書登録	16
4.3.4.5	改行文字を含む辞書登録	17
4.4	インクリメンタル・サーチ	17
4.4.1	skk-isearch の操作性	18
4.4.2	skk-isearch と入力モード	18
4.5	チュートリアル	18

5	便利な応用機能	19
5.1	ファイル構成	19
5.2	ユーザオプションの設定方法	22
5.2.1	設定ファイル	22
5.2.1.1	skk-init-fileの自動コンパイル	23
5.2.2	フック	23
5.2.3	Customize による設定変更	24
5.3	カタカナ、英字入力の便法	24
5.3.1	かなモードからカタカナを入力	24
5.3.2	全英文字の入力	25
5.3.3	領域の操作	26
5.3.4	カタカナの見出し語	27
5.3.5	文脈に応じた自動モード切り替え	27
5.4	補完	28
5.4.1	読みの補完	28
5.4.2	補完しながら変換	29
5.4.3	動的補完	30
5.5	便利な変換、その他の変換	33
5.5.1	単漢字変換	33
5.5.1.1	検索キーの設定	34
5.5.1.2	辞書の設定	34
5.5.1.3	総画数による単漢字変換	35
5.5.1.4	部首による単漢字変換	35
5.5.2	候補の絞り込み	36
5.5.3	接頭辞・接尾辞	37
5.5.4	数値変換	39
5.5.5	アスキー文字を見出し語とした変換	41
5.5.6	今日の日付の入力	41
5.5.7	プログラム実行変換	42
5.5.8	空白・改行・タブを含んだ見出し語の変換	44
5.5.9	カタカナ変換	45
5.5.10	サ変動詞変換	45
5.5.11	ファンクションキーの使い方	46
5.6	キー設定	46
5.6.1	かなモード/カナモードのキー設定	46
5.6.1.1	ローマ字のルールの設定	46
5.6.1.2	ローマ字ルールの変更例	47
5.6.1.3	モードに関連するその他の変数	47
5.6.1.4	数字や記号文字の入力	48
5.6.2	全英モードのキー設定	49
5.6.3	閉じ括弧の自動入力	49
5.6.4	確定するキー	51
5.6.5	候補の選択に用いるキー	51
5.6.6	モードでの RET	52
5.6.7	モードでの BS	52
5.6.8	送りあり変換中の C-g	53
5.6.9	変換位置の指定方法	54
5.6.10	1回の取り消し操作 (undo) の対象	55
5.7	変換、確定の前後	56
5.7.1	ポイントを戻して モードへ	56
5.7.2	直前の確定を再変換	56
5.7.3	自動変換開始	57
5.7.4	暗黙の確定のタイミング	58
5.7.5	積極的な確定	59

5.7.6	確定辞書	60
5.8	送り仮名関連	61
5.8.1	送り仮名の厳密なマッチ	61
5.8.2	送り仮名の優先的なマッチ	61
5.8.3	送り仮名の自動処理	62
5.8.3.1	どのように変換されるか	62
5.8.3.2	辞書登録の際に注意すべきこと	63
5.8.4	送りあり変換の変換開始のタイミング	65
5.9	候補の順序	65
5.9.1	変換の学習	65
5.9.2	候補の順序の固定	67
5.10	辞書関連	67
5.10.1	辞書の種類	67
5.10.2	辞書ファイルの指定	68
5.10.3	辞書の検索方法の設定	69
5.10.3.1	辞書検索の設定の具体例	69
5.10.3.2	辞書検索のための関数	69
5.10.4	Emacs 付属の辞書	70
5.10.5	サーバ関連	71
5.10.6	辞書の書式	71
5.10.6.1	送りありエントリと送りなしエントリ	72
5.10.6.2	送りありエントリのブロック形式	72
5.10.6.3	エントリの配列	73
5.10.7	強制的に辞書登録モードへ入る	73
5.10.8	誤った登録の削除	73
5.10.9	個人辞書ファイルの編集	74
5.10.10	個人辞書の保存動作	74
5.10.11	変換及び個人辞書に関する統計	75
5.10.12	辞書バッファ	76
5.10.13	辞書バッファの文字コードの設定	76
5.10.14	辞書バッファの buffer-file-name	77
5.11	注釈 (アノテーション)	77
5.11.1	アノテーションの基礎	77
5.11.2	アノテーションの使用	77
5.11.3	アノテーションの登録	78
5.11.4	アノテーションとして Wikipedia を表示する	79
5.11.5	Wikipedia アノテーション機能を SKK の枠をこえて活用する	80
5.12	文字コード関連	80
5.12.1	文字コードまたはメニューによる文字入力	80
5.12.2	文字コードを知る方法	81
5.13	DDSKK 以外のツールを用いた辞書変換	82
5.13.1	skk-lookup	82
5.13.2	skk-look	82
5.13.2.1	英単語の補完	82
5.13.2.2	英単語をあいまいに変換して取り出す	83
5.13.2.3	英単語をあいまいに変換して取り出した後、更に再帰的な英和変換を行う	83
5.13.3	Lisp シンボル名の補完検索変換	84
5.14	飾りつけ	86
5.14.1	仮名文字のローマ字プレフィックスのエコー	86
5.14.2	入力モードを示すモードラインの文字列の変更	86
5.14.3	入力モードを示すカーソル色に関する設定	87
5.14.4	変換候補一覧の表示方法	87
5.14.5	モードにおける変換候補のハイライト表示	89

5.14.6	変換候補の更なる装飾	89
5.14.7	モードラインの装飾	90
5.14.7.1	インジケータ	90
5.14.7.2	アイコン	90
5.15	ユーザガイダンス関連	91
5.15.1	エラーなどの日本語表示	91
5.15.2	助言的案内メッセージの表示	91
5.16	VIP/VIPER との併用	92
5.17	I-search 関連	92
5.17.1	起動時の入力モードの指定	92
5.17.2	間に空白等を含む文字列の検索	92
5.18	picture-mode との併用	93
6	ローマ字入力以外の入力方式	94
6.1	AZIK	94
6.2	ACT	94
6.3	TUT-code	94
6.4	かな入力と親指シフト	94
7	SKK に関する情報	96
7.1	最新情報	96
7.2	SKK メーリングリスト	96
7.3	SKK 関連ソフトウェア	96
7.4	SKK 辞書について	97
7.5	辞書ツール	97
7.6	SKK の作者	97
7.7	SKK の歴史	100
7.8	このマニュアルについて	100
7.9	謝辞	100
8	よくある質問とその回答 (FAQ)	101
8.1	SKK のなぜなに	101
Q1-1	Daredevil SKK って SKK とは違うのですか?	101
Q1-2	SKK はシンプルなのが長所だったのでは?	101
Q1-3	DDSKK はどの Emacs で使えますか?	101
Q1-4	DDSKK はどんなオペレーティングシステムで使えますか?	101
Q1-5	APEL って何? 必要ですか?	101
8.2	SKK の入手から導入まで	102
Q2-1	SKK を使うのに何が必要ですか?	102
Q2-2	SKK 辞書はどこにありますか?	102
Q2-3	SKK サーバはどこにありますか?	102
8.3	SKK の基本設定からお好みのカスタマイズまで	102
Q3-1	「 <code>ふ</code> 」「 <code>じ</code> 」が入力できるようにカスタマイズしたいのですが。	102
Q3-2	「 <code>ゐ</code> 」や「 <code>ぢ</code> 」が入力できるようにカスタマイズしたいのですが。	103
Q3-3	検索する辞書を増やしたいのですが。	103
Q3-4	左手の小指を SHIFT で酷使したくありません。	104
Q3-5	全く漢字が出てきません。	105
Q3-6	チュートリアルが起動できません。	105
8.4	SKK 辞書関連	105
Q4-1	SKK には郵便番号辞書がありますか?	105
Q4-2	SKK の辞書には、品詞情報がないんですね。	106
Q4-3	複数の SKK 辞書を結合できますか?	106
Q4-4	SKK 形式の英和辞書があると聞いたのですが。	106

8.5 SKK の活用法その他	106
Q5-1 SKK abbrev モードでもっと英単語を利用した変換ができませんか?	106
Q5-2 市販の CD-ROM 辞書やネットワークの辞書サーバが利用できますか?	107
Q5-3 他の FEP を使用中にも SHIFT を押してしまいます。	108
事項索引	109
変数索引	113
関数索引	115
キー索引	116