ddskk Documentation

リリース *17.1*

foo

2022 年 05 月 07 日

Contents:

Version

- This edition is for SKK version 17.1

Authors

- Masahiko Sato <masahiko@kuis.kyoto-u.ac.jp>
- Yukiyoshi Kameyama <kameyama@kuis.kyoto-u.ac.jp>
- NAKAJIMA Mikio <minakaji@namazu.org>
- IRIE Tetsuya <irie@t.email.ne.jp>
- Kitamoto Tsuyoshi <tsuyoshi.kitamoto@gmail.com>
- Teika Kazura <teika@lavabit.com>
- Tsukamoto Tetsuo <czkmt@remus.dti.ne.jp>
- Tsuyoshi AKIHO <akiho@sky.email.ne.jp>
- SAKAI Kiyotaka <ksakai@kso.netwk.ntt-at.co.jp>
- Satoshi Harauchi <satoshi@sys.sdl.melco.co.jp>

第 1 章

Copying

第 2 章

はじめに

2.1 このバージョンの SKK について

Daredevil SKK （以下、このマニュアルにおいて DDSKK と呼びます。）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 に基づいて配布されているフリー・ソフトウェアです。DDSKK 17.1 が動作すると思われる Emacsen のバージョンは、次のとおりです。

- GNU Emacs 24.3 以降
- GNU Emacs 25.1 以降
- GNU Emacs 26.1 以降

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
: DDSKK 14.2 以降は GNU Emacs 20 はサポート対象外です。GNU Emacs 20 のユーザは DDSKK 14.1 をお使いください。

GNU Emacs 21
: DDSKK 15.1 以降は GNU Emacs 21 はサポート対象外です。GNU Emacs 21 のユーザは DDSKK 14.4 をお使いください。

GNU Emacs 22
: DDSKK 16.2 以降は GNU Emacs 22 はサポート対象外です。GNU Emacs 22 のユーザは DDSKK 16.1 をお使いください。

GNU Emacs 23
: DDSKK 17.1 以降は GNU Emacs 23 はサポート対象外です。GNU Emacs 23 のユーザは DDSKK 16.3 をお使いください。

GNU Emacs 24.1, 24.2

: DDSKK 17.1 以降は GNU Emacs 24.1 と 24.2 はサポート対象外です。これら GNU Emacs ユーザは DDSKK 16.3 をお使いください。

GNU Emacs 24.3　GNU Emacs 24.3 と DDSKK 14 の組み合わせで isearch 使用時の不具合が発見されています。GNU Emacs 24.3 のユーザは DDSKK 15 以降をお使いください。

- http://mail.ring.gr.jp/skk/201211/msg00000.html
- http://mail.ring.gr.jp/skk/201212/msg00000.html

GNU Emacs 24.4

- coding tag を明示していないファイルは utf-8 と取り扱われます[*1] 。DDSKK 15.2 で対策済みです。
- NTEmacs は 24.3 と比べてディレクトリ構成 が異なります[*2] 。DDSKK 15.2 で対策済みです。

GNU Emacs 25.1　DDSKK 15.2 以降をお使いください（DDSKK 16 を推奨します）。

GNU Emacs 27.1

cl が正式に廃止され、cl-lib が採用されました。DDSKK 17 で対応しています。

XEmacs　DDSKK 17.1 以降は XEmacs はサポート対象外です。XEmacs のユーザは DDSKK 16.3 をお使いください。

2.2 SKK とはなにか

SKK は、かな漢字変換プログラムです。

Simple Kana to Kanji conversion program にちなんで名付けられ、その名は Combinatory Logic での有名な等式 $SKK = I$ にも由来[*3] しています。

Daredevil SKK は、SKK の更なる拡張版です[*4] 。

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、次のとおりです。

- **多彩な入力方式** をサポート。
  ローマ／かな 両対応のかな入力のほか、AZIK、ACT、TUT-code の各方式による入力も可能。
- 文法的知識を用いない高速な「かな → 漢字」変換。
- シームレスかつ再帰的な **辞書登録モード** 。

[*1] 2013-06-11 international/mule-conf.el (file-coding-system-alist).
[*2] Emacs News: Changes in Emacs 24.4 on Non-Free Operating Systems.
[*3] $SKK = I$ について詳しくは https://github.com/skk-dev/ddskk/blob/master/READMEs/history.md をご参照下さい。
[*4] Daredevil の名の由来は [Q1-1 Daredevil SKK って SKK とは違うのですか?].

- 確定語を個人辞書へ自動登録することによって、変換候補を効率的に表示する。
- 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可能。
  つまり、辞書サーバは必須ではなく、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。
- 辞書ファイルの一括ダウンロード機能。
- Emacs のオリジナル操作と同様に行える **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** ことができる。
- **入力モードの自動切り替え** `context-skk.el`
- 多彩な **アノテーション表示**
  - ユーザ・アノテーション
  - EPWING 辞書
  - Apple macOS 辞書
  - Wikipedia/Wiktionary
- 「見出し語」の **動的補完**
- **単漢字変換** （総画数、部首、部首の読み）
- 文字コード入力

脚注

第 3 章

インストール

3.1 DDSKK のインストール

ここでは、UNIX 上で **make** が利用できる環境[*1] を想定します。

まず、DDSKK のアーカイブ ddskk-VERSION.tar.gz を **tar** と **gzip** を使用して展開します。

```
% gzip -cd ddskk-VERSION.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-VERSION
```

3.1.1 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確認するために what-where を引数に **make** を実行しましょう。

```
% make what-where
-| emacs -batch -q -no-site-file -l SKK-MK -f SKK-MK-what-where
-| Loading /home/USER/temp/ddskk-VERSION/SKK-CFG...

-| Running in:
-|   GNU Emacs 26.0.50 (build1, x86_64-pc-linux-gnu, GTK+ Version ...

-| SKK modules:
-|   skk-cursor, skk-viper, ...
```

(次のページに続く)

[*1] Microsoft Windows 環境では **makeit.bat** を使用することで、UNIX と同様の操作でインストールできます。ファイル READMEs/README.w32.ja を参照してください。cygwin 環境をインストールされている方は **make** が使用できるので、本文の解説がそのまま当てはまります。Apple macOS 環境の方はファイル READMEs/README.MacOSX.ja を参照してください。

[*2] ファイル ChangeLog やファイル Makefile が置かれているディレクトリです。

(前のページからの続き)

```
-|   -> /path/to/emacs/site-lisp/skk

-| SKK infos:
-|   skk.info
-|   -> /path/to/share/info

-| SKK tutorials:
-|   SKK.tut, SKK.tut.E, NICOLA-SKK.tut, skk.xpm
-|   -> /path/to/share/skk
```

emacs の実体ファイルを特定することもできます。

```
$ make what-where EMACS=/Applications/Emacs.app/Contents/MacOS/Emacs
```

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は、ファイル SKK-CFG を編集してください。編集後は必ず **make** what-where を実行して表示内容を確認してください。

次にスーパーユーザになって、

```
$ su
% 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 DDSKK をインストールするには、

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

と、 PREFIX を指定して **make** を実行します。

特定の Emacs を指定する場合は、

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

と指定します。

3.1.2 対話的なインストール

DDSKK 14.3 では「対話的インストーラ」が追加されました。

まず M-x dired とキー入力して 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、先に述べた「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。

```
------ Minibuffer -------
Dired (directory): ~/temp/ddskk-VERSION RET
------ Minibuffer -------
```

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（アルファベットのエルの大文字）を打鍵してください。

```
------ Dired -------
-rw-r--r-- 1 user user  99999 2011-00-00 00:00 SKK-CFG
-rw-r--r-- 1 user user  99999 2011-00-00 00:00*SKK-MK    "L"
drwxr-xr-x 1 user user  99999 2011-00-00 00:00 bayesian
------ Dired -------
```

プロンプト Load SKK-MK? には y を打鍵してください。

以降、インストーラが表示する質問に答えながら DDSKK のインストールを進めてください。なお、パーミッションは一切考慮していませんので、インストール先は書き込み権限を有するディレクトリを指定してください。

3.1.3 MELPA によるインストール

2014 年 12 月、 MELPA[*3] に DDSKK が登録されたことにより、 GNU Emacs でも package.el[*4] によるインストールが可能となりました。

詳細については、次のドキュメントを参照してください。

https://github.com/skk-dev/ddskk/blob/master/READMEs/INSTALL.MELPA.md

3.2 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。

DDSKK 14.2 からは、 GNU Emacs 同梱の辞書データ ja-dic を利用したかな漢字変換に対応しましたので、SKK 辞書ファイルを別途インストールしなくても最低限の使用ができます。

しかし、 ja-dic は、 GNU Emacs の入力メソッド LEIM のためにファイル SKK-JISYO.L から変換して生成されたものであり、英数変換や数値変換などのエントリ、および「大丈夫」など複合語とみなし得る語が大幅に削除されています。そのため、ファイル SKK-JISYO.L を利用したかな漢字変換と同等の結果は得られません。

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。

[*3] Milkypostman's Emacs Lisp Package Archive.

[*4] GNU Emacs 24 以降で標準で搭載されています。GNU Emacs 23 以前では手動でイ ンストールする必要があります。 http://wikemacs.org/wiki/Package.el

3.3 辞書の入手

次のサイトには、様々な辞書が用意されています。

SKK dictionary files gh-pages

以下は、その一例です。

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 辞書の中間のサイズの辞書。 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.lisp	候補に Emacs Lisp 関数を含むエントリーを集めた辞書。 見出し語を変換する過程で Emacs Lisp 関数を評価し、その値を候補として表示します。 **プログラム実行変換**
SKK-JISYO.wrong	S, M, L 辞書に既に登録されていたが、間違いであったことが判明したために削除された単語を収録

注釈: 一部の辞書は、著作権が GNU GPL v2 ではありませんのでご注意下さい。詳細は、次の資料を参照して下さい。

https://github.com/skk-dev/dict/blob/master/committers.txt

M-x skk-get
: Emacs の使用中に M-x skk-get と実行すると、辞書ファイルを一括ダウンロードすることができます。プロンプトが表示されるので、ダウンロード先のディレクトリを入力してください。

defun skk-get &optional DIRECTORY
: *skk-get* を関数として使用することで、ユーザプログラムの中からでも辞書ファイルを一括ダウンロードすることができます。

```
(skk-get "~/jisyofiles")
```

3.4 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。ファイル SKK-JISYO.L などを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[*5] にインストールされます。

具体的なインストール先は **make** what-where を実行すると表示されます。

```
-| SKK dictionaries:
-|   SKK-JISYO.lisp, SKK-JISYO.zipcode, SKK-JISYO.office.zipcode, ...
-|   -> c:/emacs-24.5/share/emacs/24.5/etc/skk
```

dic ディレクトリに辞書ファイルを置くためには **make** get と実行する[*6] のが簡単です。

3.5 辞書サーバの入手

辞書サーバはオプションです。辞書サーバが無くても DDSKK は動作しますが、特に辞書のサイズが大きい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

脚注

[*5] /usr/share/skk や c:/emacs-24.5/etc/skk など
[*6] Microsoft Windows 環境では **makeit.bat** get と実行します。

第 4 章

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。自動的にファイル skk-setup.el 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。

4.1 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、次の内容をファイル ~/.emacs.d/init.el 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)
```

辞書サーバを使わない場合は、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")
```

辞書サーバを使わない場合は Emacs のバッファに変数 *skk-large-jisyo* が指すファイルを取り込んで使用するためメモリ使用量が増加します。これが支障となる場合は、上記のファイル SKK-JISYO.L を SKK-JISYO.M 、 SKK-JISYO.ML 又は SKK-JISYO.S に変更してください。

DDSKK 14.1 以降は辞書サーバを経由せずとも CDB 形式[*3] 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[*4] を利用する場合は、以下のように指定してください。

[*1] Emacs が起動する過程の関数 normal-top-level でファイル SKK_LISPDIR/leim-list.el が読み込まれます。ファイル leim-list.el はファイル skk-autoloads.el とファイル skk-setup.el を require します。ファイル skk-autoloads.el は DDSKK の **make** 時に自動的に生成されるファイルであり、各関数を autoload するよう定義するほか関数 register-input-method も行います。ファイル skk-setup.el はキーバインド（ C-x C-j → skk-mode ）の定義、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] 配布物にサンプルとしてファイル etc/dot.emacs とファイル etc/dot.skk があります。参考にして下さい。

[*3] constant database のこと。詳しくは http://cr.yp.to/cdb.html 又は http://ja.wikipedia.org/wiki/Cdb を参照のこと。

[*4] SKK 辞書のファイル Makefile 中の cdb ターゲットを実行することでファイル SKK-JISYO.L に基づくファイル SKK-JISYO.L.

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

変数 *skk-large-jisyo* と変数 *skk-cdb-large-jisyo* を同時に指定した場合は、標準では CDB 形式辞書ファイルの方が先に検索[*5] されます。

4.2 インクリメント検索の設定

基本的な設定はファイル skk-setup.el が読み込まれた時点で完了しています[*6] 。

defvar skk-isearch-mode-enable

この変数はファイル ~/.emacs.d/init.el か M-x customize-variable で設定してください。

Non-nil (標準設定は t)	SKK が ON になっているバッファで skk-isearch を有効にします。
nil	skk-isearch を無効にすることができます。
シンボル 'always	SKK が ON になっていないバッファでも skk-isearch を有効にします。

4.3 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、ファイル ~/.skk で以下のように設定します。

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

defvar skk-server-host

辞書サーバが起動しているホスト名又は IP アドレス。

defvar skk-server-portnum

辞書サーバが使うポート番号。ファイル /etc/services に skkserv のエントリが記述されていれば、この変数を指定する必要は無い。

defvar skk-server-prog

辞書サーバプログラムをフルパスで指定する。

defvar skk-server-jisyo

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独自の方法で辞書ファイルを指定して 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。

cdb を生成することができます。

[*5] **辞書検索の設定の具体例**

[*6] ファイル skk-setup.el では、isearch-mode-hook に関数 skk-isearch-setup-maybe を、isearch-mode-end-hook に関数 skk-isearch-cleanup-maybe をそれぞれ追加しています。関数 skk-isearch-{setup|cleanup}-maybe もファイル skk-setup.el で定義されており、その実体は、関数 skk-isearch-mode-{setup|cleanup} です。

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

- **辞書サーバの入手**
- **サーバ関連**

4.4 DDSKK を Emacs の Input Method とする

Emacs の標準キーバインドでは `C-\` を打鍵すると、関数 `toggle-input-method` を実行します。この関数は、変数 *`default-input-method`* が指す input method をトグル切り替えします。

変数 *`default-input-method`* の値はおそらく `Japanese` であり、結果として `C-\` の打鍵で LEIM (Library of Emacs Input Method) を on / off します。

使用可能な input method は `M-x list-input-methods` で確認することができ、コマンド `M-x set-input-method` 又は `C-x RET C-\` を実行することで input method を切り替えることができます。

ファイル:file:*skk-leim.el* から生成されるファイル `skk-autoloads.el` で input method をふたつ追加しています。

input method	内容
"japanese-skk"	関数 `(skk-mode 1)`
"japanese-skk-auto-fill"	:関数 el:defun:*(skk-auto-fill-mode 1)*

`defvar default-input-method`

Emacs 起動時の input method を DDSKK とするには、ファイル `~/.emacs.d/init.el` に

```
(setq default-input-method "japanese-skk")
```

と記述してください。

脚注

第 5 章

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。

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。

5.1 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 とキー入力します。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[*1] 。また、カーソルの色が変化します。

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

再び C-x C-j もしくは C-x j をキー入力することで、SKK モードに入る前のモードに戻り[*2] 、カーソル色も元に戻ります。

defvar skk-status-indicator

標準設定はシンボル '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。

defvar skk-preload

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)
```

[*1] ファイル skk.el の関数 skk-setup-modeline にて、変数 mode-line-format に変数 *skk-icon* と変数 skk-modeline-input-mode を追加しています。

[*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。

> **注釈:** ファイル ~/.emacs.d/init.el にて設定すること

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。

M-x skk-restart
SKK を再起動します。ファイル ~/.skk は再ロードしますが、ファイル ~/.emacs.d/init.el は再ロードしません。

M-x skk-version
と実行するとエコーエリアに SKK のバージョンを表示[*3] します。

```
----------- Echo Area -----------
Daredevil SKK/16.2.50 (CODENAME)
----------- Echo Area -----------
```

5.1.1 SKK オートフィルモード

C-x j とキー入力すれば、SKK モードに入ると同時にオートフィルモードをオンにします。

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 をキー入力すると、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 などとキー入力し、このコマンドに正の引数を渡します。

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 などとキー入力し、このコマンドに負の引数を渡します。

- *Auto Fill Mode in GNU Emacs Manual*
- *Arguments in GNU Emacs Manual*

5.1.2 辞書の保存

Emacs を終了するときは、保存前の個人辞書をファイル ~/.skk-jisyo.BAK に退避してから**個人辞書** の内容をファイル ~/.skk-jisyo に保存[*4] します。

[*3] **エラーなどの日本語表示**
[*4] **個人辞書の保存動作**

ファイル ~/.skk-jisyo やファイル ~/.skk-jisyo.BAK の名称を変更したければ、それぞれ変数 *skk-jisyo* や変数 *skk-backup-jisyo* の値を変更して下さい。

M-x skk-kill-emacs-without-saving-jisyo
個人辞書を保存せずに Emacs を終了させたい場合には、このコマンドをキー入力します。

5.2 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。

5.2.1 入力モードの説明

モード名称	説明	マイナーモードの表示	カーソル色
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	かな	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	カナ	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[*5]に変換するモード	全英	黄系
アスキーモード	文字を変換しないモード。 打鍵は C-j を除いて通常の Emacs のコマンドとして解釈される。	SKK	背景によりアイボリーかグレイ

入力モードを示すカーソル色に関する設定

[*5] JIS X 0208 英字のこと。このマニュアルでは「全角アルファベット」と表記する。

5.2.2 入力モードを切り替えるキー

Key	Bind	説明
q	skk-toggle-kana	「かなモード」と「カナモード」間をトグル切り替えする
l	skk-latin-mode	「かなモード」又は「カナモード」から「アスキーモード」へ
L	skk-jisx0208-latin-mode	「かなモード」又は「カナモード」から「全英モード」へ
C-j	skk-kakutei	「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、かなモードのままで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。

- **かなモードからカタカナを入力**
- **全英文字の入力**

defvar skk-show-mode-show
: Non-nil であれば、入力モードを切り替えたときに、入力モードをカーソル付近にも一瞬表示します。

M-x skk-show-mode
: 変数 *skk-show-mode-show* の値をトグル切り替えします。

defvar skk-show-mode-style
: 標準設定は、シンボル 'inline です。シンボル 'tooltip を指定することも可能です。

defface skk-show-mode-inline-face
: シンボル 'inline 利用時の face です。

5.3 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へ、ローマ字からカタカナへ、あるいはアスキー文字から全角アルファベットへ文字を変換する。

▽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

▽モードの変種として「SKK abbrev モード」があります。

▼モード　見出し語について、辞書変換を行うモード

▼モードのサブモードとして **辞書登録モード** があります。

5.3.1 ■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文字を変換します。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、標準設定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 ｎ ｎ 又は ｎ ' で入力する。直後に ｎ 及び ｙ 以外の子音が続くときは ｎ だけで入力できる。
- 促音は ｃ ｈ ｏ ｔ ｔ ｏ⇒ 「ちょっと」 や ｍ ｏ ｐ ｐ ａ ｒ ａ⇒ 「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 ｘ ａ⇒ 「ぁ」 や ｘ ｙ ａ⇒ 「ゃ」 などのように ｘ を用いる。
- 長音（ー）は - で入力する。

5.3.2 ▽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。

```
K a n j i

  ------ Buffer: foo ------
  ▽かんじ*
  ------ Buffer: foo ------
```

Ｋ ａ ｎ ｊ ｉ のように打鍵することで▽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。

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を戻してから Ｑ を打鍵することで、▽モードに入ることができます。

```
k a n j i

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

C-u 3 C-b
```

(次のページに続く)

(前のページからの続き)

```
  ------ 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 を打鍵して▽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。

- *Point in GNU Emacs Manual*

▽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、次のどちらかの方法で復帰します。

- C-j を打鍵して、■モードに戻る
- C-g を打鍵して、見出し語を消去する

```
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 ------
```

5.3.3 ▼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 を打鍵することで▼モードに入ります。▽マークから SPC を打鍵したポイントまでの文字列が「見出し語」として確定され、検索されます。同時に、▽マークは▼マークで置き換えられます。

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** が含まれるとして、以下に例を示します。

```
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（見出し語）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[*6] されます。

[*6] ハイライト表示は GNU Emacs の Overlays 機能を使用しています。

「漢字」が求める語であれば C-j を打鍵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 を打鍵せずに新たな確定入力を続ける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。打鍵することによる副作用として暗黙の確定を伴うキーは、印字可能な文字全てと RET です。

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 を打鍵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 ------
```

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す。

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 K y o の後に SPC を５回[*7] 続けて打鍵すれば

```
-------------------- 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 を打鍵します。

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。

SPC を連打してしまって求める候補を誤って通過してしまったときは x を打鍵[*9] すれば、前候補／前候補群に戻ることができます。

次々と候補を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）。

[*7] 変数 *skk-show-candidates-nth-henkan-char*

[*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。

defvar skk-previous-candidate-keys

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。

標準設定は (list "x" "\C-p") です。

defvar skk-show-candidates-nth-henkan-char

候補一覧を表示する関数 skk-henkan-show-candidates を呼び出すまでの変数 skk-start-henkan-char を打鍵する回数。2以上の整数である必要。

defvar skk-henkan-number-to-display-candidates

いちどに表示する候補の数。

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 U g o K u 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 を打鍵した時点で▼モードに入って辞書変換が行われます（ SPC 打鍵は要さない)。

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）にプレフィックスの直前に一瞬だけ * が表示されることで送り仮名の開始時点を明示します。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。

```
U g o

  ------ Buffer: foo ------
  ▽うご*
  ------ Buffer: foo ------

K

  ------ Buffer: foo ------
  ▽うご*k
  ------ Buffer: foo ------

u

  ------ 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[*10] 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。

ただし、サ変動詞の変換[*11] では、サ変動詞の語幹となる名詞を **送りなし変換**[*12] 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。

5.3.4 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、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 H e n k a n t y u u SPC とキー入力すると、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 -------
```

注釈: もちろん、誤って登録してしまった単語を削除することができます。

- **誤った登録の削除**
- **個人辞書ファイルの編集**

---

defvar skk-read-from-minibuffer-function

この変数に「文字列を返す関数」を収めると、その文字列を辞書登録モードに入ったときのプロンプトに初期表示します。関数 read-from-minibuffer の引数 INITIAL-CONTENTS に相当します。

```
(setq skk-read-from-minibuffer-function
      (lambda () skk-henkan-key))
```

defface skk-jisyo-registration-badge-face

[*10] **送り仮名の自動処理**
[*11] **サ変動詞の辞書登録に関する注意**
[*12] **送り仮名が無い場合**

変数 *skk-show-inline* が non-nil であれば、辞書登録モードに移ったことを明示するためにカレントバッファに「↓辞書登録中↓」とインライン表示します。この「↓辞書登録中↓」に適用するフェイスです。

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。

```
H e n k a n 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 を打鍵すれば「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[*13] 。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。

辞書登録モードを抜けたいときは C-g を打鍵するか、または何も登録せず RET を打鍵すると▽モードに戻ります。

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/
```

というエントリが無いとして、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*く: *
  ------ 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 とキー入力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。

[*13] ここでは「暗黙の確定」が行われるので C-j を打鍵する必要はありません。

```
D o u SPC

  ------ Minibuffer -------
  [辞書登録] うご*く: 動*
  ------ Minibuffer -------

RET

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

defvar skk-check-okurigana-on-touroku

標準設定は nil です。

non-nil	辞書登録時に送り仮名のチェックを行います。
シンボル 'ask	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	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。

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[*14] をしないで、「運動」と「する」とに分けて入力することを前提としています[*15] 。

例えば「運動する」は U n d o u SPC s u r u とキー入力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。

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

[*14] 送り仮名が有る場合

[*15] ファイル 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。 サ変動詞変換

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Ｓａｉｋｉｔｅｋｉ SPC とキー入力すると、見出し語「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Ｓａｉｋｉ SPC とキー入力すると、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。

Ｓａｉ SPC Ｋｉ SPC とキー入力すると、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［ ］の数が再帰的な辞書登録モードの深さを表わしています。

ここで RET を打鍵すると、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Ｓａｉｋｉ SPC Ｔｅｋｉ SPC とキー入力すると、

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

となります。ここで RET を打鍵することで「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。

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
```

と入力します。

5.4 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 添付のファイル `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。

5.4.1 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[*16] やユーザ変数でのカスタマイズ[*17] もそのまま利用できます。

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。

5.4.2 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 6 つです。

[*16] `M-y` の関数 `isearch-yank-kill` 、`M-p` の関数 `isearch-ring-retreat` 又は `M-n` の関数 `isearch-ring-advance` など *Incremental Search in GNU Emacs Manual*

[*17] 変数 `search-highlight` など

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: [aa]	アスキーモード
I-search: [a あ]	Abbrev モード
I-search: [–]	インクリメンタル・サーチモードで C-x C-j などを打鍵して DDSKK を終了した場合は、このプロンプト が表示されます。

defvar skk-isearch-mode-string-alist

プロンプトとして表示される文字列

5.5 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。

defvar skk-tut-file

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、ファイル ~/.emacs.d/init.el 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、ファイル /usr/local/share/skk/SKK.tut.E になります。

defvar skk-tut-lang

チュートリアルで用いる言語を文字列 Japanese 又は English で指定します。この変数よりも C-u M-x skk-tutorial による言語指定が優先されます。

defvar skk-tut-use-face

Non-nil であれば、チュートリアルで face を利用して表示します。

脚注

第 6 章

便利な応用機能

6.1 ファイル構成

SKK の基本的な機能はファイル `skk.el` に収められています。一方、DDSKK で応用機能を提供するプログラムのほとんどはファイル `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 の変数はファイル `skk-vars.el` に集約されていますので、カスタマイズしたい場合などには、このファイルを見ると参考になるかもしれません。

FILE	説明
ccc.el	buffer local cursor color control library
cdb.el	constant database (cdb) reader for Emacs Lisp
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 。
ddskk-pkg.el	*Multi-file Packages in GNU Emacs Lisp Reference Manual*
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
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
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

次のページに続く

表 1 – 前のページからの続き

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
skk-autoloads.el	コマンド **make** 時に自動生成されるファイル。 オートロードの設定のほか関数 `register-input-method` も行う。
skk-azik.el	拡張ローマ字入力 AZIK の設定を提供します
skk-bayesian.el	学習効果を提供するプログラム ユーザの過去の入力から変換候補を予測します
skk-cdb.el	CDB 形式辞書ファイルを辞書サーバなしに直接利用できるプログラム
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
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
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
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
skk-develop.el	font-lock 関係のほか、おもに開発者向けのプログラムを集めたファイル
skk-emacs.el	GNU Emacs 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
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
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
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
skk-jisx0201.el	JIS X 0201 カナ[*1] 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

次のページに続く

表 1 – 前のページからの続き

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
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。 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
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
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	コマンド **look** とのインターフェイスプログラムを集めたファイル
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
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
skk-search-web.el	Google CGI API for Japanese Input を利用したかな漢字変換 **辞書登録モード** に Google サジェストを初期表示する
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
skk-setup.el	自動的に個人設定を行うためのファイル
skk-show-mode.el	カーソル付近に入力モードを表示する機能を提供します
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

次のページに続く

表 1 – 前のページからの続き

skk-study.el	学習効果を提供するプログラム 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
skk-tankan.el	単漢字変換を行うプログラム
skk-tut.el	SKK チュートリアルプログラム
skk-tutcode.el	TUT-code 入力を実現します
skk-vars.el	DDSKK で使われる変数を集約したファイル
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
tar-util.el	utility for tar archive

6.2 ユーザオプションの設定方法

DDSKK のカスタマイズは、ファイル `~/.emacs.d/init.el` あるいはファイル `~/.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。

6.2.1 設定ファイル

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

Emacs を起動したときに一度だけ読み込まれます。

The Emacs Initialization File in GNU Emacs Manual

このマニュアルではファイル `~/.emacs.d/init.el` という記述で統一しています。

`~/.skk`

DDSKK を起動した最初の一度だけ読み込まれます。ファイル名の標準設定は 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。

ファイル `~/.skk` の名称は、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。

[*1] いわゆる半角カナ。以下、このマニュアルでは「半角カナ」と記述します

defvar skk-user-directory

DDSKK はファイル ~/.skk やファイル ~/.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、この変数にそのディレクトリ名を設定します。標準設定は nil です。

この変数はファイル ~/.emacs.d/init.el で設定してください。DDSKK 起動時にこの変数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（例えば上記ファイル ~/.ddskk ）、以下に挙げる各変数の標準設定値が変更されます。

影響を受ける変数	標準の値	変数 *skk-user-directory* を設定した場合の値
skk-init-file	~/.skk	~/.ddskk/init
skk-jisyo	~/.skk-jisyo	~/.ddskk/jisyo
skk-backup-jisyo	~/.skk-jisyo.BAK	~/.ddskk/jisyo.bak
skk-emacs-id-file	~/.skk-emacs-id	~/.ddskk/emacs-id
skk-record-file	~/.skk-record	~/.ddskk/record
skk-study-file	~/.skk-study	~/.ddskk/study
skk-study-backup-file	~/.skk-study.BAK	~/.ddskk/study.bak
skk-bayesian-history-file	~/.skk-bayesian	~/.ddskk/bayesian
skk-bayesian-corpus-file	~/.skk-corpus	~/.ddskk/corpus

なお、 変数 *skk-user-directory* を設定した場合でも、上記「影響を受ける変数」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。

skk-init-file の自動コンパイル

ここでは、「DDSKK の設定ファイル」を el 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 elc とそれぞれ呼びます。

変数 *skk-byte-compile-init-file* を適切に設定することによって、DDSKK の起動時に自動的に el をバイトコンパイルすることができます。

skk-byte-compile-init-file の値	DDSKK の起動時
non-nil	「elc が存在しない」又は「elc よりも el が新しい」ときは、 el をバイトコンパイルした elc を生成します。
nil	elc よりも el が新しいときは、 elc を消去します。

defvar skk-byte-compile-init-file

設定ファイルの自動バイトコンパイル機能を有効にしたい場合は、ファイル ~/.emacs.d/init.el に

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

と記述します。この変数はファイル ~/.skk が読み込まれる前に調べられるため、ファイル ~/.skk に上記の設定を記述しても無効です。

6.2.2 フック

defvar 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。

defvar 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。

defvar skk-load-hook

ファイル skk.el の読み込みを完了した時点で呼ばれます。ファイル ~/.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、このフックはファイル 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。

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は以下のとおり。

ファイル	フック
skk-act.el	skk-act-load-hook
skk-auto.el	skk-auto-load-hook
skk-azik.el	skk-azik-load-hook
skk-comp.el	skk-comp-load-hook
skk-gadget.el	skk-gadget-load-hook
skk-kakasi.el	skk-kakasi-load-hook
skk-kcode.el	skk-kcode-load-hook
skk-num.el	skk-num-load-hook
skk-server.el	skk-server-load-hook

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。

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

6.2.3 Customize による設定変更

Emacs 標準の Customize 機能を使って SKK を設定することもできます。ただし、Customize での設定はファイル ~/.emacs.d/init.el での設定と同様に、ファイル /.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。M-x skk-emacs-customize と実行するのも同様です。

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 Info を参照してください。

Easy Customization in GNU Emacs Manual

skk で設定できる変数の中には、まだこのマニュアルで解説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。

6.2.4 skk-customize による設定変更

M-x skk-customize

前述の「Emacs 標準の Customize 機能 M-x customize-group 」による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。

6.3 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。

6.3.1 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 を打鍵すると、カタカナに変換され確定されます。

実際には、ひらがな以外からも変換できます。以下のようになります。

- 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、▽マークとポイント間の文字列の種類[*2] 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。

```
K a t a k a n a

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

q
```

(次のページに続く)

[*2] 正確には、▽マークの次の位置にある文字列によって文字種を判別しているので、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。

(前のページからの続き)

```
------ 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／カタカナが混在した文章を書くときに、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[*3] 。

領域を対象としたコマンド でも「かな← → カナ」のトグル変換を行うことができます。

6.3.2 全英文字の入力

まず、かなモードに入ります。次に / を打鍵すると SKK abbrev モード[*4] に入りますのでアルファベット（アスキー文字）を入力します。アルファベットの入力後に C-q を打鍵する[*5] ことで、▽マークから C-q を打鍵した位置までの間にあるアルファベットが全角アルファベットに変換されて確定されます。

```
/ f i l e

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

C-q

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

なお、この変換を行うために、

```
file /ｆｉｌｅ/
```

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、辞書を参照せずにアスキー文字を 1 文字ずつ全英文字に変換しているからです。

[*3] 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。

[*4] 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。他の変換と同様に SPC を押すと変換モードに入ってしまいますので、 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。

[*5] *:kbd:C-q* は変数 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。

6.3.3 領域の操作

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

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、外部のコマンド **KAKASI**[*7] が必要です。コマンド **KAKASI** がインストールされていなければ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「五月蝿い」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。

[*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 コマンド **kakasi** がインストールされていない場合は コマンド **kakasi** を利用する機能が灰色になり使用できません。
[*7] KAKASI - 漢字 → かな（ローマ字）変換プログラム http://kakasi.namazu.org/

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。

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。

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

defvar skk-gyakubiki-jisyo-list

関数 *skk-gyakubiki-region* は、 コマンド **kakasi** を呼び出しています。 コマンド **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利用しています。

変数 *skk-gyakubiki-jisyo-list* を設定することによって コマンド **kakasi** へ与える辞書を任意に追加することができます。以下のように設定して コマンド **kakasi** へ個人辞書 *skk-jisyo* を与えることによって **辞書登録モード** で登録したばかりの単語も コマンド **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))
```

defvar skk-romaji-*-by-hepburn

この変数の値を nil に設定すると、関数 skk-romaji-{region|message} によるローマ字への変換様式に訓令式[*8] を用います。標準設定は t です。

し → si

6.3.4 カタカナの見出し語

q の打鍵でかなモード、カナモードを度々切り替えて入力を続けていると、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。

6.3.5 文脈に応じた自動モード切り替え

ファイル context-skk.el は、編集中の文脈に応じて SKK の入力モードを自動的にアスキーモードに切り替える等の機能を提供します。

ファイル context-skk.el をロードするにはファイル ~/.emacs.d/init.el 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。たとえば Emacs Lisp で日本語入力の必要があるのは

```
"文字列"
;; コメント
```

といった個所だけでしょう。文字列・コメントの **外** を編集するときは、多くの場合は日本語入力は必要ありません。

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

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

[*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。

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。

defvar context-skk-programming-mode

context-skk が「プログラミングモード」と見做すメジャーモード。

defvar context-skk-mode-off-message

アスキーモードに切り替わった瞬間にエコーエリアに表示するメッセージ。

6.4 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。

アスキー文字を見出し語とした変換

6.4.1 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[*9] が行われます。見出し語補完は、個人辞書のうち「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ためです。

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
```

(次のページに続く)

[*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」の中から「先頭が X と一致し」かつ「長さが X よりも長い見出し語」を検索して、そのような語が該当すれば X の代わりに表示します。

(前のページからの続き)

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

.

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

,

  ------ 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。

辞書の書式

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。

defvar skk-completion-prog-list

補完関数、補完対象の辞書を決定するためのリスト。標準設定は以下のとおり。

```
'((skk-comp-by-history)
  (skk-comp-from-jisyo skk-jisyo)
  (skk-look-completion))
```

defvar skk-comp-circulate

．（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標準設定 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。

defvar skk-try-completion-char

見出し語補完を開始するキーキャラクタです。標準設定は TAB です。

defvar skk-next-completion-char

次の見出し語候補へ移るキーキャラクタです。標準設定はピリオド．です。

defvar skk-previous-completion-char

前の見出し語候補へ戻るキーキャラクタです。標準設定はコンマ，です。

defvar skk-previous-completion-use-backtab

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。標準設定は 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。

defvar skk-previous-completion-backtab-key

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。

defun skk-comp-lisp-symbol &optional PREDICATE

この関数をリスト *skk-completion-prog-list* へ追加すると、Lisp symbol 名の補完を行います。

```
(add-to-list 'skk-completion-prog-list
             '(skk-comp-lisp-symbol) t)
```

6.4.2 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 を打鍵し、▼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。

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、読みが長い見出し語の単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 ------
  ▽かしたんぽせきにん*
  ------ Buffer: foo ------

SPC RET
```

(次のページに続く)

(前のページからの続き)

```
  ------ 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 ------
```

defvar skk-start-henkan-with-completion-char

標準設定は M-SPC です。

6.4.3 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。類似の機能としては、ウェブブラウザの URL の入力や、Microsoft Excel のセル入力の自動補完[*10] をイメージすると分かりやすいかも知れません。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。

動的補完を利用するにはファイル ~/.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」の部分が異なる face で表示され、動的補完機能によって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、補完され

[*10] 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

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続けて下さい。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 ------------------
```

(次のページに続く)

(前のページからの続き)

```
k a

  ---------------- Buffer: foo ------------------
  ▽ほか*ん
  ---------------- Buffer: foo ------------------

BS

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

defvar skk-dcomp-activate

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。

defface skk-dcomp-face

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。

defvar skk-dcomp-multiple-activate

Non-nil であれば、動的補完の候補をインラインに複数表示[*11] します。

```
---------------- 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。

defvar skk-dcomp-multiple-rows

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。

defface skk-dcomp-multiple-face

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。

defface skk-dcomp-multiple-trailing-face

[*11] 現在は候補群の右側 1 カラムのフェイスが標準設定に戻る、という制約があります。

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。

defface skk-dcomp-multiple-selected-face

動的補完の複数表示群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。

6.5 便利な変換、その他の変換

6.5.1 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 @ を付して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 (丶部 3 画)
  ----- Echo Area -----

SPC

  ----- Buffer: foo -----
  ▼反*
  ----- Buffer: foo -----

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

SPC

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

(次のページに続く)

(前のページからの続き)

```
  ▼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 (彑部 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。

検索キーの設定

標準設定の検索キーは @ 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 @ の打鍵が可能となりました。

defvar skk-tankan-search-key

単漢字変換の検索キー。以下は、検索キーを！へと変更する例です。

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

辞書の設定

DDSKK 14.2 からは標準で変数 skk-search-prog-list に関数 skk-tankan-search が含まれています。DDSKK 14.1 を利用の方、ご自身で変数 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。

ファイル skk-tankan.el には、漢字の部首とその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、変数 skk-search-prog-list の先頭の要素は関数 *skk-search-kakutei-jisyo-file* でなければいけませんので、変数 skk-search-prog-list の 2 番目の要素に関数 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、変数 skk-search-prog-list の要素の先頭が関数 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))
```

辞書の検索方法の設定

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 @ を付してから変換を開始します。C-u **総画数** M-x skk-tankan でも可能です。

```
Q 1 0 @

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

SPC
```

(次のページに続く)

(前のページからの続き)

```
  ----- Buffer: *候補* -----
  A:倹;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:倖;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* -----
```

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。M-x skk-tankan でも可能です。

```
Q @ @

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

SPC

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

TAB

  ------ *Completions* -------
  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 (いち)                    002 | (ぼう、たてぼう)
  003 丶 (てん)                    004 丿 (の)
  005 乙 (おつ)                    006 亅 (はねぼう)
   :                                :
  ------ *Completions* -------

0 1 8 RET
注) M-v の打鍵で、カーソルを *Completions* バッファへ移すこともできます。

  ----- Buffer: *候補* -----
  A:切;4 画 (刀部 2 画)
```

(次のページに続く)

(前のページからの続き)

```
  S:刈;4 画 (刀部 2 画)
  D:刊;5 画 (刀部 3 画)
  F:刋;5 画 (刀部 3 画)
  J:刎;6 画 (刀部 4 画)
  K:刑;6 画 (刀部 4 画)
  L:刻;6 画 (刀部 4 画)
  [残り 51+++++++]
  ----- Buffer: *候補* -----
```

defface skk-tankan-face

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。

defface skk-tankan-radical-name-face

部首の読みに適用するフェイスです。

部首の読みによる単漢字変換

直前の小々節「部首による単漢字変換」にて、部首番号を入力するプロンプトで単に RET を打鍵すると、部首の読みを入力するプロンプトに替わります。

```
  ------ Minibuffer -------
  部首を読みで選択（TAB で一覧表示）: *
  ------ Minibuffer -------

TAB

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

  Possible completions are:
  あいくち          (021) 匕          あお              (174) 青
  あか              (155) 赤          あくび            (076) 欠
  あさ              (200) 麻          あさかんむり      (200) 麻
   :                                    :
  ------ Completion List -------
```

6.5.2 候補の絞り込み

ファイル 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。インストールはファイル ~/.skk に以下を記入します。

```
(require 'skk-hint)
```

例えば、読み「かんどう」に対する変換は L 辞書によると

感動、勘当、完動、間道、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「間道」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 -----
  ▽かんどうあいだ
  ----- Buffer: foo -----

SPC

  ----- Buffer: foo -----
  ▼間道
  ----- Buffer: foo -----
```

ファイル 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 -----
```

ファイル 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 W a ; h e i w a 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 -----
```

defvar skk-hint-start-char

ヒント変換を開始するキーを character で指定します。

6.5.3 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 **辞書エントリ** 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 ------

>

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

(次のページに続く)

(前のページからの続き)

```
s i

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

SPC

  ------ Buffer: foo ------
  小林▼氏*
  ------ Buffer: foo ------


C-j

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

接頭辞も同様です。辞書に

```
ちょう> /超/
```

という **辞書エントリ** 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 ------

>

  ------ Buffer: foo ------
  ▼超*
  ------ Buffer: foo ------

O o g a t a

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

SPC

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

(次のページに続く)

(前のページからの続き)

```
  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。

defvar skk-special-midashi-char-list

▽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。この変数の標準設定は、

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

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 < と ? も使われていましたが、 現在は > に統一されています。

6.5.4 数値変換

DDSKK は **数字を含む見出し語** を様々な候補に変換することができます。例えば、見出し語「だい 12 かい」を変換すると「第１２回」、「第一二回」、「第十二回」といった候補を挙げます。

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# で置き替えることです。辞書ファイル SKK-JISYO.L の **辞書エントリ**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#3 回/第 #0 回/
```

「だい 12 かい」のような数字を含む見出し語を変換した場合、見出し語の中の数字の部分は自動的に # に置き換えられますので、 **辞書エントリ** の左辺（つまり見出し語）である「だい#かい」にマッチします。

辞書エントリ の右辺の #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。

各タイプ	説明
#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います。
#1	全角文字の数字。 12 を「１２」に変換します。
#2	漢数字で位取りあり。1024 を「一〇二四」に変換します。
#3	漢数字で位取りなし。1024 を「千二十四」に変換します。
#4	数値再変換。 見出し語中の数字そのもの[*12] 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	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「壱阡九百九拾伍」に変換します。これを大字と言います。
#8	桁区切り。1234567 を 1,234,567 に変換します。
#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

[*12] `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。

```
# /#3/
```

という **辞書エントリ** があるときに、Ｑ １ ０ ０ ２ ０ ０ ３ ０ ０ ４ ０ ０ ５ ０ ０ SPC または/ １ ０ ０ ２ ０ ０ ３ ０ ０ ４ ０ ０ ５ ０ ０ SPC とキー入力[*13] すれば、「百兆二千三億四十万五百」と変換されます。

辞書に

```
#m#d /#0月#0日/
```

という **辞書エントリ** があるときに/ ２ ｍ ２ ５ ｄ SPC と入力[*14] すれば、「2 月 25 日」と変換されます。

辞書に

```
#kin /#9金/
```

という **辞書エントリ** 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 **辞書エントリ** 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 に対して辞書が再検索され、「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ひとつ挙げます。「２月２５日」を得るために、Ｑ ２ ｇ ａ ｔ ｕ ２ ５ ｎ ｉ ｔ ｉ SPC とキー入力したときに、辞書に見出し語

```
#がつ#にち /#1月#1日/
```

がないときは、 **辞書登録モード** 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。

defvar skk-num-convert-float

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。

[*13] SHIFT キーを伴って数字を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。

[*14] 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。
アスキー文字を見出し語とした変換

defvar skk-show-num-type-info

Non-nil であれば、 **辞書登録モード** に入るのと同時に変換タイプの案内を表示します。標準設定は t です。

defvar skk-num-grouping-separator

タイプ #8 で使用する記号。標準設定は , 。

defvar skk-num-grouping-places

タイプ #8 について、何桁毎に区切るのかを数値で指定する。標準設定は 3。

defvar skk-use-numeric-conversion

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。

6.5.5 アスキー文字を見出し語とした変換

かなモードで / を打鍵すると **SKK abbrev モード** 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 **辞書エントリ** 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 ------
```

候補を確定すると SKK abbrev モードを抜けてかなモードに戻ります。

SKK abbrve モードで使われる辞書は、普通のかな漢字変換と同じです。見出し語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。

上記の例において SPC の代わりに C-q を打鍵することで、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こともできます。

全英文字の入力

なお、SKK abbrev モードにおいても TAB による **見出し語の補完**を行うことができます。

6.5.6 今日の日付の入力

かなモード／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。

defvar skk-date-ad
: 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。標準設定は nil です。

defvar skk-number-style
: この変数の値は以下のように解釈されます。標準設定は 1 です。

設定値	出力結果
0 or nil	ASCII 数字 「1996 年 7 月 21 日 (日)」のようになります。
1 or t	全角数字 「１９９６年７月２１日 (日)」のようになります。
2	漢数字（位取） 「一九九六年七月二一日 (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。

辞書ファイル SKK-JISYO.lisp には、見出し語 today の候補として関数 skk-date-ad と変数 *skk-number-style* の全ての組み合わせが **プログラム実行変換**機能を用いて登録されています。従って、/ t o d a y 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。

関数 *skk-relative-date* を利用すると、昨日、一昨日、明後日など任意の日付を求めることができます。詳細はファイル skk-gadget.el のコメントを参照してください。

なお、@ の打鍵で日付を挿入するのではなく、文字どおり @ を挿入したい場合の設定は次のとおり。

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

6.5.7 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、返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 **辞書エントリ** があるとします。このとき / n o w SPC とキー入力すれば、現在のバッファに関数 current-time-string の返り値である

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

のような文字列が挿入されます。

ここで、プログラムの返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[*15] があります。

今日の日付の入力 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持っています。

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

ファイル skk-gadget.el には、西暦／元号変換や簡単な計算など**プログラム実行変換** 用の関数が集められています。

defun skk-calc operator

関数 *skk-calc* は、引数をひとつ取り、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。
  ;; 辞書エントリの例 -> #*# /(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 **辞書エントリ** を追加します。

数値変換

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

[*15] 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。

辞書の種類

しかし、実行変換されるプログラムを辞書登録する際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。

Q 1 1 1 * 4 5 SPC とキー入力します。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。この中で変数 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。その上で関数 *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。

defun skk-gadget-units-conversion 基準単位 数値 変換単位

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。

defvar skk-units-alist

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。標準設定では、以下の単位変換の情報を定義しています。

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

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

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

(次のページに続く)

(前のページからの続き)

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

defun skk-relative-date pp-function format and-time **&key** (yy 0) (mm 0) (dd 0)
関数 skk-current-date の拡張版。引数 PP-FUNCTION, FORMAT, AND-TIME の意味は関数 skk-current-date の docstring を参照のこと。キーワード変数 :yy, :mm, :dd に正または負の数値を指定することで明日、明後日、一昨日などの日付を求めることができる。詳細はファイル skk-gadget.el のコメントを参照のこと。

6.5.8 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することもできます。

```
  ---------------- 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 ------------------
```

(次のページに続く)

(前のページからの続き)

```
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。

defvar skk-allow-spaces-newlines-and-tabs

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。

6.5.9 カタカナ変換

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
- ▽モードで q によりカタカナへ変換する **かなモードからカタカナを入力**

のどちらかです。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見出し語の補完候補にも現れず、何度でも入力しなければなりません。

変数 skk-search-katakana を設定することで、カタカナ語が普通の変換候補として現れ、個人辞書にも登録されます。設定するには以下をファイル ~/.skk に記述します[*16] 。

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

また、値をシンボル 'jisx0201-kana とすると、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)
```

6.5.10 サ変動詞変換

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。

サ変動詞の辞書登録に関する注意

[*16] 変数 skk-search-prog-list の設定をユーザが変更している場合は期待どおりに動作しない場合があります。その場合は変数 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。

変数 skk-search-sagyo-henkaku を設定することで、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ファイル ~/.skk に記述します[*17] 。

```
(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 SPC s u r u
サ変	O c h a S u r u

変数の値をシンボル '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。

6.5.11 異体字へ変換する

「辺」（42 区 53 点）の異体字である「邊」（78 区 20 点）や「邉」（78 区 21 点）を入力したいときがあります[*18] 。

```
  ---- Buffer: foo ----
  *辺
  ---- Buffer: foo ----

Q

  ---- Buffer: foo ----
  ▽*辺
  ---- Buffer: foo ----

C-f

  ---- Buffer: foo ----
  ▽辺*
  ---- Buffer: foo ----

SPC
```

(次のページに続く)

[*17] 変数 skk-search-prog-list の設定をユーザが変更している場合は期待どおりに動作しない場合があります。その場合は変数 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。

[*18] 辞書が充実していれば、かな漢字変換で見出し語「へん」から「邊」や「邉」を求めることができます。もちろん、文字コードを指定して「邊」や「邉」を直接挿入することもできます。

(前のページからの続き)

```
  ---- Buffer: foo ----
  ▼邉*
  ---- Buffer: foo ----

SPC

  ---- Buffer: foo ----
  ▼邉*
  ---- Buffer: foo ----
```

defvar skk-itaiji-jisyo

辞書ファイル SKK-JISYO.itaiji 又はファイル SKK-JISYO.itaiji.JIS3_4 へのパスを指定する。他の辞書ファイルと異なり、この２つの辞書ファイルは見出し語が漢字です。

defun skk-search-itaiji

not documented. http://mail.ring.gr.jp/skk/200303/msg00071.html

6.5.12 ファンクションキーの使い方

defvar skk-j-mode-function-key-usage

シンボル 'conversion ならば、変数 skk-search-prog-list-1 〜変数 skk-search-prog-list-9 および変数 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変数 *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。

冗長な案内メッセージの表示

[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

シンボル 'kanagaki ならば、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。

6.6 キー設定

6.6.1 かなモード／カナモードのキー設定

ローマ字のルールの設定

DDSKK の■モードにおける文字変換は、２つの変数

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

を用いて行われます。

変数 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。

変数 skk-rom-kana-rule-list は、ユーザが独自のルールを定めるために用意されており、変数 skk-rom-kana-base-rule-list よりも優先して評価されます。

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	変換される前のアスキー文字の文字列
出力	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 （**"ア" . "あ"**）のようなコンスセル

変数 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。

変数 skk-rom-kana-base-rule-list には、次のような便利な変換ルールも定められています。

z SPC	→ 全角スペース
z*	→ ※
z,	→ ‥
z-	→ 〜
z.	→ …
z/	→ ・
z0	→ ○
z@	→ ◎
z[	→ 『
z]	→ 』
z{	→ 【
z}	→ 】
z(	→ （
z)	→ ）
zh	→ ←
zj	→ ↓
zk	→ ↑
zl	→ →
zL	→ ⇒

ローマ字ルールの変更例

変数 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。

更に変数 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。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各ファイルを参照してください。

ローマ字入力以外の入力方式

■モードに関連するその他の変数

defvar skk-kana-input-search-function

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。これは変数 skk-rom-kana-base-rule-list と変数 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。バッファの文字を、直接 preceding-char などで調べて下さい。

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

これは変数 skk-rom-kana-rule-list の標準設定に

```
("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。

defvar skk-kutouten-type

■モードの標準では、キーボードの . を打鍵すると「。」が、 , を打鍵すると「、」がバッファに挿入されます。変数 *skk-kutouten-type* に適切なシンボルを設定することにより、この組み合せを変更[*19] することができます。そのシンボルとは、次の４つです。

設定するシンボル	バッファに挿入される文字
'jp （標準設定）	「。」「、」
'en	「.」「,」
'jp-en	「。」「,」
'en-jp	「.」「、」

または、変数 *skk-kutouten-type* にはコンス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。

defvar skk-use-auto-kutouten

標準設定は nil 。 Non-nil であれば、カーソル直前の文字種に応じて句読点を動的に変更します。

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、関数 skk-insert にバインドされています。

[*19] 変数 *skk-use-kana-keyboard* が non-nil ならば無効である。

```
!  #  %  &  '  *  +

-  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をカスタマイズするためには、変数 skk-rom-kana-rule-list を利用します。

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

関数 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。

```
C-u 2 !

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

6.6.2 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関数 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、その標準設定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
 " "  "！" "”" "＃" "＄" "％" "＆" "’"
 "（" "）" "＊" "＋" "，" "－" "．" "／"
```

(次のページに続く)

(前のページからの続き)

```
 "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]
```

挿入される文字を変更したい場合: **数字や記号文字の入力**

関数 `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、関数 `skk-insert` における動作と同じく、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。

数字や記号文字の入力

6.6.3 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、なにより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。

```
  ------ 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の入力を即始めることができます。

defvar skk-auto-paren-string-alist

自動挿入すべきペアの文字列を指定します。標準設定は下記のとおり。

```
(("「" . "」") ("『" . "』") ("(" . ")") ("（" . "）") ("{" . "}")
 ("｛" . "｝") ("〈" . "〉") ("《" . "》") ("[" . "]") ("［" . "］")
```

(次のページに続く)

(前のページからの続き)

```
("〔" . "〕") ("【" . "】") ("\"" . "\"") ("“" . "”") ("`" . "'"))
```

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スセルを集めたリスト」です。各コンスセルの関数 car にある文字列を挿入したときに関数 cdr にある文字列が自動挿入されます。

このリストの各要素の関数 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 ")" を自動挿入するには

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

のように設定する必要があります。

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 度キー入力して関数 skk-mode もしくは関数 skk-auto-fill-mode を起動し直す必要があります。

キーとなる文字が挿入されても、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、自動挿入さ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することができます。

```
(add-hook 'yatex-mode-hook
          (lambda ()
```

(次のページに続く)

(前のページからの続き)

```
              (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))))))
```

6.6.4 リージョンを括弧で囲む

「閉じ括弧の自動入力」の応用として、リージョンを括弧で囲むことができます。

```
  ------ Buffer: foo ------
  このマニュアルにおいて*DDSKK*と呼びます
  ------ Buffer: foo ------

`


  ------ Buffer: foo ------
  このマニュアルにおいて`DDSKK'*と呼びます
  ------ Buffer: foo ------
```

defvar skk-use-auto-enclose-pair-of-region

non-nil であれば、上記の機能が有効になります。当然に変数 skk-auto-insert-paren も non-nil である必要があります。なお、 delete-selection-mode の方が優先されます。

6.6.5 確定するキー

defvar skk-kakutei-key

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。

暗黙の確定のタイミング

6.6.6 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。

```
-------------------- Echo Area --------------------
A:嘘  S:拒  D:拠  F:虚  J:挙  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。

defvar skk-henkan-show-candidates-keys

７つの異なる文字のリスト。文字は必ず小文字とする。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、含めてはならない。標準設定は、以下のとおり。

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

defface skk-henkan-show-candidates-keys-face

選択キーを表示する際のフェイスを指定します。

defvar skk-henkan-rest-indicator

標準設定は nil 。Non-nil であれば [**残り** 99++] の表示を右寄せ配置する。

defface skk-henkan-rest-indicator-face

[**残り** 99++] の face 属性。標準設定は default 。

6.6.7 ▼モードでの RET

標準設定では、

```
K a k u t e i SPC

  ------ Buffer: foo ------
  ▼確定*
  ------ Buffer: foo ------
```

(次のページに続く)

(前のページからの続き)

```
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。

defvar skk-egg-like-newline

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。従って、 **辞書登録モード**において▼モードであるときの RET 打鍵時の挙動も変化[*20] します。

```
K a k u t e i SPC

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

RET

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

6.6.8 ▼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 SPC

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

BS

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

[*20] 辞書登録モードの標準の確定、登録の動作は **辞書登録モード**

defvar skk-delete-implies-kakutei

この変数の値を nil に設定すると、▼モードで BS を押した時にひと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ひとつ前の候補表示に戻る動作となります。

6.6.9 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 ------
```

defvar skk-delete-okuri-when-quit

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のには便利です。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 ------
```

6.6.10 変換位置の指定方法

SKK では通常、「漢字変換の開始位置」と「送り仮名の開始位置」を大文字で指定しますが、これらを任意のキーで指定することで sticky-shift ライクな操作[*21] も可能です。

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

と設定すると ; キーで[*22] 漢字変換位置が指定できるようになります。

例えば「有る」という単語を入力するには ; a ; r u というキー入力で可能となり、シフトキーを押す必要がなくなります。

[*21] あくまでも「任意のキーで変換開始位置を指定する」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 SKK 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。

[*22] ファイル skk-hint.el を併用する場合は変数 *skk-hint-start-char* の標準設定も ; であるため、どちらかを別のキーに割り当てる必要があります。

候補の絞り込み

操作上は通常の sticky-shift[*23] 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。通常の sticky と比べて skk-sticky は ; を押した時点で画面表示が変化するので若干分かり易いと思います。

キーの設定方法は、割り当てるキーの種類によって異なります。

- 表示を伴うキー

  ; などの表示を伴うキーの場合は

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

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

- 表示を伴わないキー

  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 では [noconvert]
```

  のようにそのキーを表わす vector を設定して下さい。

- 同時打鍵

  ２つのキーを同時に打鍵することでも漢字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。

defvar skk-sticky-double-interval

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。標準設定は 0.1 秒。

[*23] *Q3-4* **左手の小指を** *SHIFT* **で酷使したくありません。**

6.6.11 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[*24] 。

正確に言えば、関数 skk-insert ， 関数 skk-set-henkan-point ， 関数 skk-jisx0208-latin-insert[*25] 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[*26] をいちどのアンドゥの対象としています。

ただし、これらの DDSKK のコマンドと Emacs 本来の関数 self-insert-command を織り混ぜてキー入力した場合[*27] 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てと*    ※ 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
  ------------------------- Buffer: foo -------------------------

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 文字のみがアンドゥの対象
  -------------------------- Buffer: foo --------------------------
```

[*24] 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。

[*25] SKK abbrev モードでは、アスキー文字入力が Emacs 本来の関数 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変数 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、関数 skk-abbrev-comma や関数 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。

[*26] 20 は Emacs のソースファイルの一部であるファイル keyboard.c に定められたマジックナンバーと一致します。

[*27] 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。

6.7 変換、確定の前後

関連事項:

- **送りあり変換の変換開始のタイミング**
- **変換位置の指定方法**

6.7.1 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は**後から▽モードに入る方法** で説明しました。

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、しかもポイントは動きません。

M-Q

M-x skk-backward-and-set-henkan-point

M-Q（大文字の Q です。）と打鍵すると、現在位置の直前の文字列について走査し、同種の文字（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）が続く限り後方に戻って▽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て▽モードに入ります。変換開始ポイントを「を」又は「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、引数を与えないで M-Q を実行した場合です。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。

defvar skk-allow-spaces-newlines-and-tabs

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、この変数の値を nil に設定します。

6.7.2 直前の確定を再変換

一番最後（直近）の確定を取り消して、再変換することができます。これを **確定アンドゥ** と呼びます。

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 SPC

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

この例では、「高校」の確定を取り消しています。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 を押せば候補が尽きて **辞書登録モード** に入ります。

この例のとおり、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、次の新たな確定を行うまで[*28] 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。

defvar skk-undo-kakutei-return-previous-point

この変数の値が non-nil であれば、確定アンドゥ処理が完了した後に、確定アンドゥ処理の直前の位置にカーソルが戻ります。上の例の場合、確定アンドゥ処理が完了した後のカーソル位置は、標準設定 nil 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。

[*28] C-j を打鍵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。

6.7.3 自動変換開始

▽モードで見出し語を入力しているときに「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[*29] 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 ------
```

defvar skk-auto-okuri-process

この変数を non-nil に設定して **送り仮名の自動処理** 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 **辞書エントリ** 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 ------
```

defvar skk-auto-start-henkan-keyword-list

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。標準設定は以下のようになっています。

```
("を" "、" "。" "." "," "？" "」" "！" "；" "：" ")" ";"
 ":" ")" "”" "】" "』" "》" "〉" "｝" "］" "〕" "｝"
 "]" "?" "." "," "!" )
```

[*29] ▽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。打鍵に入力された文字（「を」や「。」）は見出し語には含まれません。

defvar skk-auto-start-henkan

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。標準設定は t です。

6.7.4 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[*30] を入力すると、直ちに確定されます。これを「暗黙の確定」と呼んでいます。具体的には以下のようになります。

```
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*      ; 暗黙の確定
  ------ Buffer: foo ------

u

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

defvar skk-kakutei-early

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 (,), [,] の入力時
- 句読点 ,, . の入力時
- 次の変換開始時（ A から Z までの大文字の入力時）
- RET 入力時

まで暗黙の確定が遅延されます[*31] 。

[*30] 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。

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

```
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 r u

  ------ Buffer: foo ------
  ▼確定する*
  ------ Buffer: foo ------

.

  ------ Buffer: foo ------
  確定する。*       ; 暗黙の確定
  ------ Buffer: foo ------
```

6.7.5 積極的な確定

変換候補がひと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。

defvar skk-kakutei-when-unique-candidate

この値が non-nil であれば、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、SKK abbrev モードでの変換の全てでこの機能が有効になります。

または、 `okuri-ari` , `okuri-nasi` , `abbrev` を要素とするリストであることもできます。この場合は、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。

```
'(okuri-nasi abbrev)
```

この機能は、全ての辞書を検索した上で変換候補が唯一か否かを調べます。そのため、変数 `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。

辞書の検索方法の設定

defvar skk-kakutei-search-prog-limit

この変数の値が数値であれば、積極的な確定 skk-kakutei-when-unique-candidate における「変換候補が唯一か否か」の判定を変数 skk-search-prog-list の先頭から数えて当該数値の個数までの辞書に制限します。

数値以外であれば、無制限に全ての辞書を検索対象とします。

6.7.6 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 **辞書エントリ** 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。**辞書エントリ** の候補がひとつだけだからです。

確定辞書以外の辞書に登録されているであろう同音異義語を得るには、確定変換の直後に x を打鍵します。すると、▼モードに戻って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」が、個人辞書（や共有辞書）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 ------
```

(次のページに続く)

(前のページからの続き)

```
▼自署*
------ Buffer: foo ------
```

確定辞書の単語は、優先的に変換されます。

defvar skk-kakutei-jisyo

確定変換用の辞書ファイル[*32] を指定します。 nil であれば、確定変換は行われません。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。

辞書の書式

6.8 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、試してみて下さい。

6.8.1 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 SPC 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。

defvar skk-henkan-okuri-strictly

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[*33] 。

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、さもなければ、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。

[*32] 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（バイナリサーチ）を行い、サイズが小さければ適当な配置で直線的検索（リニアサーチ）を行うことをお勧めします。次も参照してください。

- **辞書検索のための関数**
- **エントリの配列**

[*33] 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。
送りあり変換の変換開始のタイミング

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[*34] 。

```
(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))))
```

6.8.2 送り仮名の優先的なマッチ

送り仮名の厳密なマッチ 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[*35] 。

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 SPC 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。

defvar skk-henkan-strict-okuri-precedence

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。

[*34] 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。

[*35] 「大く」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。

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[*36] 。 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。

6.8.3 送り仮名の自動処理

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

どのように変換されるか

defvar skk-auto-okuri-process

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 SPC とキー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[*37] 。

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[*38] 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

[*36] **送りあり変換の変換開始のタイミング**

[*37] 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。

[*38] **送りありエントリのブロック形式**

[**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。

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[*39] 、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[*40] 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、自動送り機能の特徴を考えてみると、

長所	• 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。
短所	• 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。

となります。

変数 *skk-auto-okuri-process* の値を non-nil に設定したとしても、従来どおりの送りあり変換も同時にできますので、一度この機能を試してみることをお勧めします[*41] 。

[*39] 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。

[*40] 「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。

[*41] **専ら補完的に自動送り処理を利用するのであれば関数** *skk-okuri-search* を変数 skk-search-prog-list の最後に設定するという方法もあります。
辞書の検索方法の設定

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[*42] 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 W a t a s i t a 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 SPC

  ------ 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 W a t a S i t a RET と送り仮名を明示的に入力して「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 -------

RET

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

この確認に対して 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。一方で、 n と回答した場合は、個人辞書の「送りなしエントリ」に

[*42] 変数 *skk-auto-okuri-process* の値を non-nil に設定している。

```
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。

defvar 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* の標準設定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。

```
ぁ あ ぃ い ぅ う ぇ え ぉ お か が き ぎ く ぐ
け げ こ ご さ ざ し じ す ず せ ぜ そ ぞ た だ
ち ぢ っ つ づ て で と ど な に ぬ ね の は ば
ぱ ひ び ぴ ふ ぶ ぷ へ べ ぺ ほ ぼ ぽ ま み む
め も ゃ や ゅ ゆ ょ よ ら り る れ ろ ゎ 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、「ふ」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」に対して「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、

```
-------------------------- Echo Area --------------------------
Shall I register this as okuri-ari entry: ありがs /有賀/ ? (y or n)
-------------------------- Echo Area --------------------------
```

の確認に対して 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。

6.8.4 送りあり変換の変換開始のタイミング

defvar skk-process-okuri-early

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/[く/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[*43] 。

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。

- 変数 *skk-kakutei-early*
- 変数 *skk-auto-okuri-process*
- 変数 *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 回打鍵して 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。

- **暗黙の確定のタイミング**
- **送り仮名の自動処理**
- **送り仮名の厳密なマッチ**

[*43] **辞書の書式**

6.9 候補の順序

skk の初期設定では、変換で確定された単語は、次の変換時では最初に表示されます。この動作を変更して、効率良く変換する方法があります。

ここで解説するほか、 **確定辞書を用いた変換** も、候補の順序に影響を与えます。

6.9.1 変換の学習

ファイル 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。

ファイル ~/.skk に (require 'skk-study) と書いて DDSKK を起動して下さい。以降、 かな漢字変換の学習を始めます。

例えば「梅雨には雨が降る」と変換した場合、

- 雨（あめ）の関連語 → 梅雨（つゆ）
- 降る（ふ r）の関連語 → 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。

ここで続けて「傘を振る」と変換すると、個人辞書が更新されてしまい、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 A m e SPC g a H u R u とキー入力すると、H u R u（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」に更新された状態で、

```
A m e SPC g a T a i r y o u SPC n i H u R u
```

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[*44] 。

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、変数 skk-study-search-times 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。

ファイル skk-study.el に関するその他のオプションを説明します。

[*44] 「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。

defvar skk-study-sesearch-times

現在の変換キーに対する関連変換キーをいくつまで遡って検索するか。標準設定は 5 です。

defvar skk-study-max-distance

この変数には integer を指定します。標準設定値は 30 です。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。

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。

defvar skk-study-first-candidate

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。この変数の標準設定値は t です。

defvar skk-study-file

学習結果を保存するファイル名です。この変数の標準設定値は *~/.skk-study* です。変数 *skk-user-directory* からも設定ができます。

設定ファイル

defvar skk-study-backup-file

ファイル `~/.skk-study` **のバックアップファイルです。** この変数の標準設定値は *~/.skk-study.BAK* です。

defvar skk-study-sort-saving

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。この変数の標準設定値は nil です。

defvar skk-study-check-alist-format

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。この変数の標準設定値は nil 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

```
------ Minibuffer -------
Theme of current buffer: *
```

(次のページに続く)

(前のページからの続き)

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

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。

6.9.2 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。

defvar skk-jisyo-fix-order
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。標準は nil 。

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ので、ファイル `skk-study.el` による変換の学習と併用できます。

変換の学習

変数 *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。コマンド `M-x skk-edit-private-jisyo` を実行して**個人辞書ファイルを直接編集** して下さい。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。

6.9.3 ベイズ統計を用いた学習

ファイル `skk-bayesian.el` は、直前の履歴のみ使用するファイル `skk-study.el` に比べて、更に拡張された学習機能です。ベイズ統計を用いて文脈から変換候補が選択される確率を計算して候補順をソートします。なお、機能が重なることからファイル `skk-study.el` と の併用はお勧めできません。

動作の枠組みは

- emacs lisp のファイル `skk-bayesian.el` と
- ruby スクリプト[*45] の コマンド **bskk**[*46]

[*45] http://www.ruby-lang.org
[*46] Ruby 2.4 以降を使用する場合は、DDSKK 16.2 以降に付属するファイル `bayesian/bskk` を使用してください。

が連携することで実現しています。

ファイル skk-bayesian.el のインストールについてはファイル bayesian/README.ja.md を参照してください。

defvar skk-bayesian-debug
non-nil ならば、以下のとおりデバッグ用のメッセージを表示します。

- ファイル skk-bayesian.el が吐き出すメッセージを *Messages* バッファに表示します。
- コマンド **bskk** サブプロセスを -d オプションで起動させます。コマンド **bskk** はファイル $HOME/tmp/bskk.log にメッセージを吐き出します。
- 普段は非表示である *skk-bayesian* バッファを表示するようにします。このバッファには コマンド **bskk** の出力が表示されます。

defvar skk-bayesian-prefer-server
non-nil ならば変数 *skk-bayesian-host* の変数 *skk-bayesian-port* に接続します。nil であれば コマンド **bskk** を emacs のサブプロセスとして起動します。

defvar skk-bayesian-host
コマンド **bskk** サーバが稼動しているホスト名

defvar skk-bayesian-port
コマンド **bskk** サーバのポート番号

defvar skk-bayesian-history-file
not documented

defvar skk-bayesian-corpus-make
not documented

defvar skk-bayesian-corpus-file
not documented

M-x skk-bayesian-kill-process
not documented

6.10 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。

6.10.1 辞書の種類

- 共有辞書

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。

ファイル SKK-JISYO.S (S 辞書)、ファイル SKK-JISYO.M (M 辞書)、ファイル SKK-JISYO.ML (ML 辞書)、ファイル 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。

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 https://skk-dev.github.io/dict/ 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

  これは共有辞書、個人辞書という区分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。

- skk-kakutei-jisyo

  同上

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。

6.10.2 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。

設定ファイル

defvar skk-kakutei-jisyo

確定変換 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。

defvar skk-initial-search-jisyo

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変数

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。

defvar skk-jisyo

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。

```
(setq skk-jisyo "/your/path/to/jisyofile")
```

個人辞書は、変数 @code{skk-jisyo-code} で指定された文字コードで取り扱われます。ところで、変数 @code{skk-jisyo-code} の指定は、個人辞書だけに限らず共有辞書などにも影響しますので、もし、個人辞書だけを変数 *skk-jisyo-code* の指定とは異なる文字コードで取り扱いたいときは、次のとおりコンス・セルを設定します。

```
(setq skk-jisyo (cons "/your/path/to/jisyofile" 'utf-8))
```

この設定は `~/.emacs.d/init.el` で行ってください。同時に、Emacs を起動する前に個人辞書ファイルの文字コードを変換しておく必要があります。配布物の `etc/dot.emacs` に設定例が記載されていますので、参考にしてください。

defvar skk-backup-jisyo

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。

defvar skk-cdb-large-jisyo

共有辞書のうち *CDB 形式に変換した辞書* です。指定した場合は変数 *skk-large-jisyo* よりも先に検索されます。DDSKK 14.1 からは辞書サーバを経由せずとも CDB 形式辞書ファイルを直接検索できるようになりました。

defvar skk-large-jisyo

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。例えば変数 *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あり、これには変数 *skk-large-jisyo* を nil に設定します。

defvar skk-aux-large-jisyo

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。

defvar skk-extra-jisyo-file-list

SKK では個人辞書の他に、共有辞書または辞書サーバを設定して利用するのが一般的ですが、郵便番号辞書ファイル `SKK-JISYO.zipcode` をはじめとした多彩な辞書もメンテナンスされています。

これらの辞書を利用するために変数 skk-search-prog-list を手動で編集することもできますが、この変数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。以下の例を参考に変数 *skk-extra-jisyo-file-list* の設定をファイル ~/.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[*47] 。

ただし、変数 *skk-jisyo-code*[*48] とは異なる文字コードのファイルについては、上記の例中のファイル 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、変数 skk-search-prog-list の設定で変更することもできます。

辞書検索のための関数

6.10.3 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。

辞書検索の設定の具体例

この節では変数 skk-search-prog-list 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。複数の辞書が同時並列に検索されるのではなく、指定した順番に検索します。変数 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)
 (skk-search-ja-dic-maybe)
 (skk-search-extra-jisyo-files)
```

(次のページに続く)

[*47] 変数 skk-search-prog-list に登録されている関数 skk-search-extra-jisyo-files が、変数 *skk-extra-jisyo-file-list* の各要素を逐次処理します。

[*48] 辞書バッファの文字コードの設定

(前のページからの続き)

```
(skk-search-katakana-maybe)
(skk-search-sagyo-henkaku-maybe))
```

この例では、

- skk-kakutei-jisyo （確定辞書）
- skk-initial-search-jisyo
- skk-jisyo （個人辞書）

の順に検索を行い、次に

- **送り仮名の自動処理**

を行い、その後

- skk-cdb-large-jisyo と
- skk-large-jisyo の

検索を順に行い、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。

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、

- 確定辞書では候補は見つけられなかったが 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。
- 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。

辞書検索のための関数

前節で見たとおり、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。

defun skk-search-jisyo-file FILE LIMIT **&optional** NOMSG

通常の検索を行うプログラム。変数 skk-henkan-key を見出し語（検索文字列）として、 FILE を被検

索対象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。

第 1 引数 FILE は、被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。 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 2 引数 LIMIT は二分検索（バイナリ・サーチ）が行なわれる領域の大きさを指定します。ひと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[*49] が第 2 引数に指定された数値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（リニア・サーチ、関数 search-forward ）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 *skk-jisyo* はソートされておらず二分検索が不可能であるため LIMIT を 0 にして下さい。

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 のメッセージをエコーエリアに出力します。non-nil を与えると出力しません。

defun skk-search-cdb-jisyo CDB-PATH

not documented

defun skk-search-kakutei-jisyo-file FILE LIMIT **&optional** NOMSG

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。このためユーザが確定操作を行う必要はありません。引数の意味はいずれも関数 *skk-search-jisyo-file* の場合と同様です。

defun 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。

送り仮名の自動処理

defun skk-search-server FILE LIMIT **&optional** NOMSG

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関数 *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、

- **辞書サーバを使いたいときの設定**
- **サーバ関連**

をご覧下さい。

[*49] 検索領域の先頭ポイントと末尾ポイントの差

6.10.4 Emacs 付属の辞書

GNU Emacs には、ファイル SKK-JISYO.L を元に変換されたファイル leim/ja-dic/ja-dic.el という辞書が付属しています。

DDSKK 14.2 からは、このファイル ja-dic.el を利用したかな漢字変換（送りあり、送りなし、接頭辞、接尾辞）が可能となりました。つまり、ファイル SKK-JISYO.L などの辞書ファイルを別途準備しなくても一応は DDSKK の使用が可能、ということです。

DDSKK 14.2 から追加された「ja-dic.el 検索機能」 (*skk-search-ja-dic*) は、次の設定の **全てが無効** な場合に有効となります。

- skk-large-jisyo
- skk-aux-large-jisyo
- skk-cdb-large-jisyo
- skk-server-host

ただし、「ja-dic.el 検索機能」はファイル SKK-JISYO.L を利用する場合と比べて、英数変換や数値変換などができません。可能な限りファイル SKK-JISYO.L などの辞書を利用することを推奨します。

辞書の入手

defvar skk-inhibit-ja-dic-search

この変数を non-nil に設定すると、変数 *skk-large-jisyo* 等の値にかかわらず、あらゆる場面で関数 *skk-search-ja-dic* を無効とします。

defun skk-search-ja-dic

GNU Emacs に付属するかな漢字変換辞書ファイル ja-dic.el を用いて検索する。現在 の GNU Emacs にはファイル SKK-JISYO.L を基に変換されたファイル 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、ファイル SKK-JISYO.L のような英数変換、数値変換などはできず、また「大丈夫」のように複合語とみなしうる語彙が大幅に削除されている。

6.10.5 サーバ関連

辞書サーバの基本的な設定は、 **辞書サーバを使いたいときの設定**を参照してください。

defvar skk-servers-list

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。各リストは次の４つの要素から成ります。

- 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

ただし、辞書ファイル名及びポート番号は、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。

defvar skk-server-report-response

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エコーエリアに報告します。

```
-------------------- Echo Area --------------------
辞書サーバの応答を 99 回待ちました
-------------------- Echo Area --------------------
```

defvar skk-server-inhibit-startup-server

nil であれば、辞書サーバと接続できない場合に、Emacs から関数 `call-process` で辞書サーバプログラムの起動を試みます。標準設定値は t です。

`inetd` 経由で起動する多くの辞書サーバは関数 `call-process` で起動することができませんが、`skkserv` のように

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

といったオプションを受け付けて関数 `call-process` で起動す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。

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")
```

defvar skk-server-remote-shell-program

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。標準設定は、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")))
```

defvar skk-server-version

辞書サーバから得たバージョン文字列とホスト名文字列を表示する。

```
(skk-server-version)
-| SKK SERVER version (wceSKKSERV) 0.2.0.0 (ホスト名 foo:192.168.0.999: )
```

6.10.6 サーバコンプリージョン

Server completion に対応した辞書サーバであれば、見出し語から始まる全ての語句の検索が可能です。

defun skk-comp-by-server-completion

この関数を変数 *skk-completion-prog-list* の要素に追加すると、▽モードにおいて見出し語補完を実行します。

```
(add-to-list 'skk-completion-prog-list
             '(skk-comp-by-server-completion) t)
```

defun skk-server-completion-search

この関数を変数 skk-search-prog-list の要素に追加すると、変換を実行する際に変数 *skk-server-completion-search-char* を付すことによって見出し語で始まるすべての候補を掲げます。

```
(add-to-list 'skk-search-prog-list
             '(skk-server-completion-search) t)
```

```
   ------ Buffer: foo ------
   ▽おおさか~*
   ------ Buffer: foo ------

SPC

   ------ Buffer: *候補* ------
```

(次のページに続く)

(前のページからの続き)

```
A:おおさかいかだいがく
S:大阪医科大学
D:おおさかいがい
F:大阪以外
J:おおさかいだい
K:大阪医大
L:おおさかいちりつだいがく
------ Buffer: *候補* ------
```

defvar skk-server-completion-search-char

標準設定は ~ （チルダ、 #x7e ）です。

6.10.7 辞書の書式

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。

ファイルの先頭から `;; okuri-ari entries.` までの間の行で、行頭に `;` を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.` の間に囲まれた部分が**送りありエントリ** です

`;; okuri-nasi entries.` 以降のファイル末尾までの部分が **送りなしエントリ** です。

「送りありエントリ」を検索する変換を **送りあり変換** と、「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ひと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。

ひと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 プログラムに変換して辞書に収めています。

プログラム実行変換

「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[*50] 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 になるからです。

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。

```
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」を送り仮名とするひとつ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、変数 *skk-henkan-okuri-strictly* あるいは変数 *skk-auto-okuri-process* のいずれかが non-nil である場合に使用されます。この場合、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[*51] 。

[*50] 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。

[*51]

- 送り仮名の自動処理
- 送り仮名の厳密なマッチ

https://skk-dev.github.io/dict/ で配布し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きます。変数 *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[*52] 。

エントリの配列

共有辞書は「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、「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** に配置されます。

降順／昇順に配置されるのは、辞書サイズが大きいことに配慮して二分検索（バイナリサーチ）を行うためです[*53] 。

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ます。つまり、「送りなしエントリ」は `;; okuri-ari entries.` を、「送りありエントリ」は `;; okuri-nasi entries.` を基点として最小ポイントに挿入されて辞書が更新されます[*54] 。

個人辞書は、通常は共有辞書ほど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検索（リニアサーチ）が行われます。最後に確定された語は、ひとつのエントリの中の最初の位置に置かれます。

6.10.8 強制的に辞書登録モードへ入る

▼モードにて、エコーエリアに変換候補が表示されているときに ． を打鍵すると、強制的に **辞書登録モード** へ入ります。

defvar skk-force-registration-mode-char

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 ．（ピリオド、0x2E）です。

6.10.9 誤った登録の削除

誤って個人辞書に登録してしまった単語は、あとからでも削除できます。

いったん、削除したい単語を変換により求めて、その単語が表示された時点（確定する前の▼モードの状態）で X（大文字のエックス）を打鍵します。

- **送り仮名の優先的なマッチ**

[*52] ただし、変数 *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。

[*53] ソートする際には、見出し語を unsigned-char と見なします。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 のコマンド **sort** での標準の順序とは異なります。Emacs の関数 `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 の関数 `sort-columns` は 内部的に UNIX のコマンド **sort** を使っているので、辞書のソートには使えません。

[*54] 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

ミニバッファに確認プロンプトが出るので y e s と答えると、個人辞書の対応するエントリが削除されます。先程、現在のバッファにいったん入力した「誤りの変換結果」も削除されます。

例えば、

さいきてき /再起的/

という **辞書エントリ** を誤って登録してしまったという仮定で、この誤登録を削除する場合を説明します。

```
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 ------
```

6.10.10 個人辞書ファイルの編集

警告: 構文チェックが十分ではありませんので、個人辞書ファイルの編集は、自己責任のもと行ってください。

M-x skk-edit-private-jisyo

このコマンドを使うと、個人辞書ファイルが開かれます[*55] 。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 とキー入力しましょう。個人辞書ファイルを保存してバッファを閉じます。

[*55] 前置引数 C-u を伴って実行する C-u M-x skk-edit-private-jisyo ことで、コーディングシステムを指定して個人辞書を開くことができます。

6.10.11 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。

- C-x C-c または M-x save-buffers-kill-emacs によって Emacs を終了する場合
- M-x skk-save-jisyo と入力したか、メニューバーの Save Jisyo を選択した場合
- 個人辞書の更新回数が、変数 *skk-jisyo-save-count* で指定された値に達した結果として、自動保存（オートセーブ）機能が働くとき。
- 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[*56] 。

```
--------------------------- Minibuffer -----------------------------
New ~/.skk-jisyo will be 11bytes smaller.  Save anyway?(yes or no)
--------------------------- Minibuffer -----------------------------
```

ここで n o 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。 y e s RET と答えた場合は、元の個人辞書を退避用の辞書ファイル ~/.skk-jisyo.BAK 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ため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[*57] 。

defvar skk-compare-jisyo-size-when-saving

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を比較しません。

defvar skk-jisyo-save-count

この変数で指定された回数、個人辞書が更新された場合に個人辞書が自動保存されます。標準設定は 50 です。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[*58] 。

[*56] 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ことがあります。他の場合はバグの可能性があります。

[*57] *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。

[*58] 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。

defvar skk-save-jisyo-instantly

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。

defvar skk-share-private-jisyo

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。

6.10.12 変換及び個人辞書に関する統計

DDSKK は、かな漢字変換及び個人辞書に関する統計を取っており、Emacs の終了時にファイル `~/.skk-record` に保存します。保存する内容は、以下の形式です。

```
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:」の数は個人辞書 *skk-jisyo* に登録されている候補数ですが、ここでは１行を１語として数えています。そのため、ひとつの見出し語に対して複数の候補を持っている場合は、２つ目以降の候補を無視しています。

defvar skk-record-file

統計情報を保存するファイル名を指定します。

設定ファイル

defvar skk-keep-record

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。数値を設定すると、変数 *skk-record-file* を指定数値の行数より大きくしません。

defvar skk-count-private-jisyo-candidates-exactly

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。

```
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 ----------------
```

(次のページに続く)

(前のページからの続き)

```
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。

Menu Bars in GNU Emacs Manual

6.10.13 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。辞書バッファの命名規則は、

空白 ＋ ＊ ＋ 辞書ファイル名（ディレクトリ抜き） ＋ ＊

です。

例えば、変数 *skk-large-jisyo* の値がファイル /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* （アンダーバーは SPACE の 意）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 と、変数 mode-name の値を文字列 SKK dic としています[*59] 。

6.10.14 辞書バッファの文字コードの設定

defvar skk-jisyo-code

この変数は、辞書ファイルの文字コードを決定し、以下のような値を取ります。

- nil （標準設定）この場合、シンボル 'euc-jis-2004 が使われます。詳細は、関数 skk-find-coding-system を参照のこと。
- Emacs の coding system （コード系）[*60]

[*59] これは、Emacs のファイル dabbrev.el の機能との調和を考えての措置です。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るのですが、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。

[*60] coding system は、 'euc-jp, 'shift_jis, 'junet などのシンボルで表され、 M-x describe-coding-system や M-x list-coding-systems で 調べることができます。

- euc, ujis, sjis, jis の文字列。skk-coding-system-alist に従って、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。

6.10.15 辞書バッファの buffer-file-name

Emacs には関数 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 のように関数 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。

6.11 注釈（アノテーション）

かな漢字変換の際に、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。

6.11.1 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、

1. ユーザが登録したもの、
2. 共有辞書に元々登録されているもの、
3. それ以外の情報源から取得されるもの

の 3 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[*61] 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。

[*61] * の文字は変換時には表示されません。

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。

上記のいずれでもなく、外部の辞典その他の情報源から得られるものを「外部アノテーション」といいます。外部アノテーションは Emacs Lisp パッケージである lookup.el、Apple macOS 付属の辞書、Wiktionary/Wikipedia などから取得可能です。

6.11.2 アノテーションの使用

C-w

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。

The Kill Ring in GNU Emacs Manual

^

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。標準設定は ^ 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:洟; はな
 K:痍; 満身創痍
 L:維; 維持
 ----- Buffer: *候補* -----

^


 ----- Buffer: *候補* -----
 A:射
 S:亥;
 D:夷;
 F:姨;
 J:洟;
 K:痍;
 L:維;
 ----- Buffer: *候補* -----
```

defvar skk-show-annotation

設定例	動作
t	アノテーションを常に表示します。
'(not list)	候補バッファではアノテーションを表示しません。
'(not minibuf)	ミニバッファにおけるかな漢字変換（単語登録時）ではアノテーションを表示しません。
'(not list minibuf)	候補バッファ及びミニバッファではアノテーションを表示しません。
nil	いかなる場合もアノテーションを表示しません。

defvar skk-annotation-delay

アノテーションを表示するまでの遅延を秒で指定する。標準設定は 1.0 秒。

defvar skk-annotation-show-as-message

Non-nil （標準設定）	アノテーションをエコーエリアに表示します。
nil	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	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。

この変数の値にかかわらず、変数 *skk-show-tooltip* が non-nil の場合はアノテーションをツールティップで表示します。

defvar skk-annotation-function

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* が指し示す関数が関数 `funcall` されて、戻り値が non-nil である場合に限って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、アノテーションがユーザアノテーション（先頭が * で始まる）の場合に t を返すラムダ式を変数 *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を表示することができます。

defvar interprogram-cut-function

保存した内容を Emacs 以外のアプリケーションで利用したい場合に設定してください。

6.11.3 アノテーションの登録

M-x skk-annotation-add

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* バッファを開きます。

M-x skk-annotation-kill

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* バッファを閉じたいときは 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。

M-x skk-annotation-remove

特定の語からアノテーションを取り去りたいときは、まず、かな漢字変換で当該語を確定し、続けて M-x skk-annotation-remove と実行します。

6.11.4 アノテーションとして EPWING 辞書を表示する

ファイル skk-lookup.el に含まれる関数 *skk-lookup-get-content* を活用することにより、EPWING 辞書から得た内容をアノテーション表示することが可能です。辞書検索ツールの Lookup[*62] が正常にインストールされていることが前提です。Lookup を新規にインストールした場合は、SKK をインストールし直す必要があります。

EPWING 辞書の内容をアノテーション表示するには、２つの方法があります。

skk-treat-candidate-appearance-function を設定する方法

候補の表示を装飾する関数を指定する変数 *skk-treat-candidate-appearance-function* を設定する場合は、ファイル etc/dot.skk に示されている設定例を以下のように変更してください。

```
+ (require 'skk-lookup)
  (setq skk-treat-candidate-appearance-function
        #'(lambda (candidate listing-p)
            (let* ((value (skk-treat-strip-note-from-word candidate))
                   (cand (car value))      ; 候補
-                  (note (cdr value))      ; 注釈
```

(次のページに続く)

[*62] 変数 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。

(前のページからの続き)

```
+                   (note (skk-lookup-get-content cand listing-p))
                    (sep (if note           ; セパレータ
                    :
```

defun skk-lookup-get-content 単語 `listing-p`

「単語」の意味を EPWING 辞書から取得します。オプション引数 `listing-p` が non-nil ならば、候補一覧用に一行の短い文字列を返しますが、 nil ならば全体を返します。

defvar skk-lookup-get-content-nth-dic

関数 *skk-lookup-get-content* が「どの EPWING 辞書から単語の意味を取得するのか」を、ゼロを起点とした数値で指定します。docstring に例示した S 式 を評価してみてください。

M-x skk-lookup-get-content-setup-dic

DDSKK の起動後に変数 *skk-lookup-get-content-nth-dic* の数値を変更した場合は、このコマンドを必ず実行してください。

skk-annotation-lookup-lookup を設定する方法

次に変数 *skk-annotation-lookup-lookup* について説明します。この変数は EPWING 経由アノテーションの設定を簡単にします。

defvar skk-annotation-lookup-lookup

Non-nil ならばファイル `lookup.el` を利用してアノテーションを取得する。

```
(setq skk-annotation-lookup-lookup t)
```

この値をシンボル 'always に設定[*63] すると、候補一覧でも辞書サー ビスを引く。

```
(setq skk-annotation-lookup-lookup 'always)
```

6.11.5 Apple macOS 「辞書」サービスからアノテーションを取得する

Apple Mac OS X 10.5 (Leopard) 以降に標準で入っている国語辞典などからアノテーションが取得できます。この機能を利用するには、python の拡張機能として `readline` と `pyobject-framework-DictionaryServices` が必要です。

`pyobject-framework-DictionaryServices` については Apple Mac OS X 10.5 (Leopard) 以降の OS 標準の python にあらかじめインストールされています。

[*63] この設定は、変数 *skk-treat-candidate-appearance-function* の値を上書きします。変数 *skk-treat-candidate-appearance-function* を自分で設定する場合は変数 *skk-annotation-lookup-lookup* には t または nil を必要に応じて設定します。

readline については Apple Mac OS X 10.7 (Lion) 標準の python ではインストールする必要がありません。Apple Mac OS X 10.6 (Snow Leopard) 以前の場合は

```
% easy_install readline
```

などの方法でインストールします。

今のところ、アノテーションを取得する辞典を選択することはできません。Apple macOS の「辞書」アプリ (Dictionary.app) を起動し、環境設定から辞書の検索順を指定してください。国語辞典を上位に指定すれば使いやすくなります。

defvar skk-annotation-lookup-DictionaryServices

Non-nil ならば Apple macOS の辞書サービスを利用してアノテーションを取得する。

```
(setq skk-annotation-lookup-DictionaryServices t)
```

この値をシンボル 'always に設定すると、候補一覧でも辞書サービスを引く[*64] 。

```
(setq skk-annotation-lookup-DictionaryServices 'always)
```

defvar skk-annotation-python-program

アノテーション取得のために呼びだす python のプログラム名。

```
(setq skk-annotation-python-program "/usr/bin/python")
```

6.11.6 Wikipedia/Wiktionary からアノテーションを取得する

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

- Wiktionary
- Wikipedia

による解説を表示することができます。他のアノテーションが変換時に自動的に表示されるのに対し、Wikipedia/Wiktionary アノテーションは基本的にユーザの指示によって取得される点が異なります。

▼モードで候補を表示しているときに C-i を押すと、変数 *skk-annotation-other-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[*65] します。

[*64] この設定は、変数 *skk-treat-candidate-appearance-function* の値を上書きします。変数 *skk-treat-candidate-appearance-function* を自分で設定したい場合は変数 *skk-annotation-lookup-DictionaryServices* には t または nil を必要に応じて設定します。

[*65] 変数 *skk-show-tooltip* が non-nil の場合、ツールティップで表示します。

```
B o k u j o u

   ----- Buffer: foo -----
   ▽ぼくじょう*
   ----- Buffer: foo -----

SPC

   ----- Buffer: foo -----
   ▼牧場*
   ----- Buffer: foo -----

C-i

   ------------------------------ 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。

defvar skk-annotation-wikipedia-key

標準設定は C-i です。

defvar skk-annotation-browse-key

標準設定は C-o です。EWW (Emacs Web Wowser) で閲覧したい場合は、次のとおり設定してください。

```
(setq browse-url-browser-function 'eww-browse-url)
```

Emacs Web Wowser Manual

defvar skk-annotation-other-sources

アノテーションを取得する SKK 辞書以外のソースを指定します。

6.11.7 外部コマンドからアノテーションを取得する

外部コマンドからアノテーションを取得できます。

defvar skk-annotation-lookup-dict

Non-nil ならば、変数 *skk-annotation-dict-program* に指定された外部コマンドからアノテーションを指定します。

defvar skk-annotation-dict-program

アノテーションを取得するための外部コマンド名を指定します。

defvar skk-annotation-dict-program-arguments
　アノテーションを取得に使う外部コマンドに渡す引数を指定します。

6.11.8 各種アノテーション機能を SKK の枠をこえて活用する

これまでに解説した各種の外部アノテーション

- lookup.el + EPWING 辞書
- Apple macOS 辞書
- Wikipedia / Wiktionary

は、SKK の変換モードだけでなく Emacs のあらゆる状況で辞書引き機能として使うことができます。そのためには、関数 *skk-annotation-lookup-region-or-at-point* を任意にキー定義します。

defun skk-annotation-lookup-region-or-at-point &optional 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 Lookup.el, Apple macOS 辞書サービス、または Wikipedia/Wiktionary 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-lookup-region-or-at-point)
```

このようにしておくと、何かの意味が調べたくなったとき、領域選択して M-i と打鍵すれば、その場で辞書を引くことができます。

さらに、変数 *skk-annotation-other-sources* の 3 番目 (Apple macOS では 4 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て M-3 M-i (Apple macOS では M-4 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[*66] 。

```
----- Buffer: *scratch* -----
;; This buffer* is for notes you don't want to save, and for ...
----- Buffer: *scratch* -----

M-3 M-i (Apple macOS では M-4 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
```

(次のページに続く)

[*66] 変数 *skk-annotation-other-sources* の標準の値は環境によって異なります。lookup.el と skk-lookup.el の設定が有効になっている場合は en.wiktionary は 4 番目 (Apple macOS では 5 番目) になります。

(前のページからの続き)

```
buffer (plural 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 --------------------
```

6.12 文字コード関連

6.12.1 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 \ キーを打鍵すると、ミニバッファに

```
---------------------------- Minibuffer -----------------------------
○○の文字を指定します。7/8 ビット JIS コード (00nn), 区点コード (00-00),
UNICODE (U+00nn), または [RET] (文字一覧): *
---------------------------- Minibuffer -----------------------------
```

というプロンプトが表示され、文字コード（JIS コード、EUC コードまたは区点番号）またはメニューによる文字入力が促されます。

プロンプト中の○○部分は、変数 skk-kcode-charset の値であり、その初期値は"japanese-jisx0208" 又は "japanese-jisx0213-1" です。初期値は環境によって自動的に設定されます。キー \ の代わりに C-u \ と入力すると、異なる文字集合 (charset) を指定する事ができます。

ここで、文字コードがあらかじめ分かっている場合には、その文字コードを入力します。例えば「℃」の文字コードは、JIS コードでは 216e 、EUC コードでは a1ee なので、いずれかの文字コードを入力すれば ℃ が現在のバッファに挿入されます。

区点番号で入力するには 01-78 のように区と点の間にハイフン "-" を入れる必要があります。ハイフン "-" で区切った 3 組の数字は JIS X 0213 の 2 面を指定したとみなします。例えば 2-93-44 で「ðľ'ÿ¡」（ほっけ、U+29e3d）が入力できます。

6.12.2 メニューによる文字入力

文字コードが不明の文字を入力するには、文字コードを入力せずにそのまま RET キーを入力します。するとミニバッファに以下のような表示が現れます。

```
---------------------------- Minibuffer -----------------------------
A:    S:¯   D:〜   F:}   G:＝   H:¢   Q:◆   W:    E:∩   R:    T:≡   Y:
---------------------------- Minibuffer -----------------------------
```

これを **第１段階のメニュー** と呼びます。第１段階のメニューでは、JIS 漢字をコードの順に 16 文字毎に１文字抽出し、ミニバッファに一度に 12 文字ずつ表示しています。

上記の例では、JIS コード 2121（全角スペース）、2131、2141、 2151 … の文字がそれぞれ表示されています。

ここで SPC を打鍵すると、次の候補群を表示します（文字コードの値を 16 * 12 = 192 ずつ増やします）。キー x を打鍵すると、ひとつ前の候補群に戻ります。

キー a, s, d, f, g, h, q, w, e, r, t, y のいずれかを打鍵する[*67] と、そのキーに対応する文字から始まる 16 個の文字が文字コード順に表示されます。これを **第２段階のメニュー**と呼びます。

例えば、第１段階のメニューが上記の状態のときに d を打鍵すると、第２段階のメニューは以下のようになります。

```
--------------------------------- Minibuffer -----------------------------
A:〜 S:‖ D:｜ F:… G:‥ H:‘ J:’ K:“ L:” Q:（ W:） E:｛ R:｝ T:［ Y:］ U:｛
--------------------------------- Minibuffer -----------------------------
```

ここで、キー a, s, d, f, g, h, j, k, l, q, w, e, r, t, y, u のいずれかを打鍵すると、対応する文字がカレントバッファに挿入されてメニューによる入力が終了します。

第２段階のメニューが表示されているときも SPC と x キーにより第２段階のメニューが前進、後退します。

また、キー < 及び > により、メニューを１文字分だけ移動します。例えば、第２段階のメニューが上記の状態のときにキー < を打鍵すると、メニューは以下のようになります。

```
--------------------------------- Minibuffer -----------------------------
A:＼ S:〜 D:‖ F:｜ G:… H:‥ J:‘ K:’ L:“ Q:” W:（ E:） R:｛ T:｝ Y:［ U:］
--------------------------------- Minibuffer -----------------------------
```

第１段階あるいは第２段階のメニューが表示されているときにキー ? を打鍵すると、そのときのキー A に対応する文字（上記の例では "＼"）の文字コードが表示されます。

defvar skk-kcode-method

キー \ の打鍵で起動する関数 skk-input-by-code-or-menu の挙動を調節します。

[*67] 大文字でも小文字でも構いません。なお、第１段階・第２段階ともに、メニューのキーを変更することができます。
候補の選択に用いるキー

設定値	挙動
シンボル 'char-list	キー \ の打鍵で、文字コード一覧関数 skk-list-chars を起動します。
シンボル 'code-or-char-list	キー \ の打鍵で、文字コード関数 skk-input-by-code を起動します。 JIS コード／区点コード入力プロンプトの表示に対して単に RET を打鍵した場合、 文字コード一覧関数 skk-list-chars を起動します。
シンボル 'this-key	キー \ の打鍵で \ を挿入します。
上記シンボル以外	キー \ の打鍵で、文字コード関数 skk-input-by-code を起動します。 JIS コード／区点コード入力プロンプトの表示に対して単に RET を打鍵した場合、 「メニュー入力」を起動します。

6.12.3 文字コード一覧

M-x skk-list-chars と実行すると、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。プレフィックス付きで、つまり C-u M-x skk-list-chars と実行すると、カーソル位置の文字に照準をあわすようコード一覧を表示します。

```
-------------------- *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 ＾  ￣  ＿  ヽ  ヾ  ゝ  ゞ  〃  仝  々  〆  〇  ー  ―  ‐  ／
  2140 ＼  ～  ∥  ｜  …  ‥  ‘  ’  “  ”  （  ）  〔  〕  ［  ］
  2150 ｛  ｝  〈  〉  《  》  「  」  『  』  【  】  ＋  －  ±  ×
  2160 ÷  ＝  ≠  ＜  ＞  ≦  ≧  ∞  ∴  ♂  ♀  °  ′  ″  ℃  ￥
-------------------- *skk-list-chars* --------------------
```

Key	挙動
C-f, f, l	カーソル移動
C-b, b, h	カーソル移動
C-n, n, j	カーソル移動
C-p, p, k	カーソル移動
C-x C-x	カーソル移動
RET, i	文書バッファへ文字を挿入
g	aa
\, o	文字集合の切り替え
c	文字コード入力
$	カーソル位置の文字の文字コードを表示
w	aa
q	skk-list-chars を抜ける

ほか、Emacs のコマンド M-x list-charset-chars や C-x 8 RET も有用でしょう。

defface skk-list-chars-table-header-face
コード一覧の枠線などに適用するフェイス

defface skk-list-chars-face
プレフィックス付きで実行したときの照準のフェイス

6.12.4 文字コードを知る方法

かなモード／カナモードでキー $ を打鍵する[*68] 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に表示[*69] します。

例えば、カーソルを文字 **A** の上に置いて $ を打鍵すると、

```
-------------------- Echo Area --------------------
`A',KUTEN:07-01, JIS:#x2721, EUC:#xa7a1, SJIS:#x8440, UNICODE:U+0410,
キリール大文字 A,CYRILLIC CAPITAL LETTER A
-------------------- 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分かります。

ほか、 Emacs のコマンド M-x describe-char も有用でしょう。

[*68] リードオンリーなバッファでは M-x skk-display-code-for-char-at-point を実行してください。
[*69] 変数 *skk-show-tooltip* が non-nil であればツールティップで表示します。変数 *skk-show-candidates-always-pop-to-buffer* が non-nil であれば other-window に表示します。変数 *skk-show-tooltip* が優先します。

defface skk-display-code-prompt-face
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。

defface skk-display-code-char-face
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。

defface skk-display-code-tankan-radical-face
エコーエリアに表示されるメッセージ中の総画数表示に適用するフェイスです。

defface skk-display-code-tankan-annotation-face
エコーエリアに表示されるメッセージ中の文字名表示に適用するフェイスです。

6.13 DDSKK 以外のツールを用いた辞書変換

6.13.1 skk-lookup

ファイル skk-lookup.el を使用すると、辞書検索ツールの Lookup 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。

DDSKK のインストール過程で (require 'lookup) が成功する場合はファイル skk-lookup.el も自動的にインストールされます。まずはコマンド **make** what-where を実行して SKK modules: 欄に skk-lookup が含まれていることを確認してください。

Lookup がインストールされているにも関わらず、うまくファイル skk-lookup.el がインストールされない場合は、ファイル SKK-CFG を編集して（変数 ADDITIONAL_LISPDIR にファイル lookup.el が置かれているパスを設定する)、再度 DDSKK をインストールして下さい[*70] 。

ファイル ~/.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))))
```

関数 skk-lookup-search は、 DDSKK が用意している検索プログラムの中で最も遅いものです。したがって、変数 skk-search-prog-list の設定にあっては辞書サーバの検索関数 *skk-search-server* よりも後方に置くよう設定します。

Lookup の agent で利用するのは、変数 lookup-search-agents から ndkks, ndcookie 及び ndnmz を取り去ったものです[*71] 。

[*70] 関数 skk-lookup-search がファイル skk-autoloads.el に追加されます。
[*71] 変数 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。

6.13.2 skk-look

ファイル skk-look.el は、 コマンド **look** コマンドを使って 3 つの機能を提供します。

英単語の補完

defvar skk-use-look

non-nil に設定すると、ファイル skk-look.el が使用できるようになります。例えばファイル ~/.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 コマンド **look** コマンドを使用した補完が有効になります。

```
/ 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 . （ピリオド）で次の補完候補に、, （コンマ）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 *edict* があれば、ここから SPC を押して英和変換ができます。

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。

defvar skk-search-excluding-word-pattern-function

```
(add-hook 'skk-search-excluding-word-pattern-function
          ;; 返り値が 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)))))
```

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/
```

というエントリがあるとして解説します[*72] 。

defvar skk-look-recursive-search

non-nil であれば、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。

```
  ▽ abstr*

SPC

  ▼ abstract

SPC

  ▼アブストラクト
```

(次のページに続く)

[*72] edict 辞書ファイル 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。

(前のページからの続き)

```
SPC
  ▼抽象
SPC
  ▼ abstraction
SPC
  ▼アブストラクション
```

defvar skk-look-expanded-word-only

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。

6.13.3 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 -----

SPC

  ----- Buffer: foo -----
  ▽ define-abbrev*
  ----- Buffer: foo -----

SPC

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

(次のページに続く)

(前のページからの続き)

```
  ▽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。例えばファイル SKK-JISYO.JIS3_4 には

```
A~ /チルド付き A(LATIN CAPITAL LETTER A WITH TILDE)/
```

と登録されています。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。

defun skk-search-lisp-symbol &optional PREDICATE NOT-ABBREV-ONLY WITHOUT-CHAR-MAYBE
: 引数 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。

defvar skk-completion-search-char
: 関数 skk-completion-search による変換機能を指示するキーキャラクタ。標準設定は ~ です。

6.13.4 Google CGI API for Japanese Input を利用したかな漢字変換

かな漢字変換に Google CGI API for Japanese Input を利用することができます。 連文節変換も可能となります。

まず、ファイル ~/.skk にて、変数 skk-use-search-web を non-nil に設定します。これにより、skk-mode を起動した際にファイル skk-search-web.el を require するようになります。

同じくファイル ~/.skk にて、リスト skk-search-prog-list の一番最後の要素として、関数 skk-search-web を追加します。

```
(add-to-list 'skk-search-prog-list
             '(skk-search-web 'skk-google-cgi-api-for-japanese-input)
             t)
```

以上の設定によって、通常のかな漢字変換の **候補が尽きたとき** に関数 skk-search-web が実行され、Google CGI API for Japanese Input による変換結果が表示されます。

そのほか、変数 *skk-read-from-minibuffer-function* を以下のように設定することで、辞書登録モードへの突入時の初期値に Google サジェストを表示することもできます。

```
(setq skk-read-from-minibuffer-function
      (lambda ()
        (car (skk-google-suggest skk-henkan-key))))
```

6.14 飾りつけ

6.14.1 仮名文字のローマ字プレフィックスのエコー

defvar skk-echo

この変数の値は、仮名文字の **ローマ字プレフィックス** 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 が入力された瞬間に「た」の文字が挿入されます。

defface skk-prefix-hiragana-face

かなモードにおける **ローマ字プレフィックス** のフェイスを指定します。

defface skk-prefix-katakana-face

カナモードにおける **ローマ字プレフィックス** のフェイスを指定します。

defface skk-prefix-jisx0201-face

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。

6.14.2 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-show-mode の表示も連動します。

変数	モードライン
skk-latin-mode-string	アスキーモードを示す文字列。標準は SKK
skk-hiragana-mode-string	かなモードを示す文字列。標準は **かな**
skk-katakana-mode-string	カナモードを示す文字列。標準は **カナ**
skk-jisx0208-latin-mode-string	全英モードを示す文字列。標準は **全英**
skk-abbrev-mode-string	SKK abbrev モードを示す文字列。標準は a **あ**

6.14.3 入力モードを示すカーソル色に関する設定

defvar skk-use-color-cursor

この変数が non-nil ならば、カーソルを色付けします。

標準では、ウィンドウシステムを使用して、かつ、色表示が可能な場合に限って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。

変数	カーソルの色
skk-cursor-default-color	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	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	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	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	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	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	SKK abbrev モードであることを示すカーソル色。 標準は royalblue です。

6.14.4 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。

1. 現在のウィンドウにインライン表示する
2. ツールティップで表示する
3. 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）
4. エコーエリアに表示する

現在のウィンドウにインライン表示する

defvar skk-show-inline

この変数の値が 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。値がシンボル 'vertical であれば、各候補を縦方向にインライン表示します。

defface skk-inline-show-face

インライン表示する変換候補を装飾するフェイスを指定します。標準設定は `underline` です。

```
(setq skk-inline-show-face 'font-lock-doc-face)
```

変数 *skk-treat-candidate-appearance-function* による装飾を優先するには nil に設定して下さい。

defvar skk-inline-show-background-color

インライン表示する変換候補の背景色を指定します。

skk-inline-show-face または変数 *skk-treat-candidate-appearance-function* にて、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。

defvar skk-inline-show-background-color-odd

インライン表示する変換候補の背景色（奇数ライン）を指定します。

ツールティップで表示する

defvar skk-show-tooltip

この変数の値が non-nil であれば、候補一覧をツールティップで表示します。同時に、

- **注釈（アノテーション）の表示方法** と
- **文字コードの表示方法**

も制御します。

defface skk-tooltip-face

ツールティップ表示する文字列に適用するフェイスのシンボルを指定する変数です。

```
(setq skk-tooltip-face 'font-lock-doc-face)
;; (make-face 'skk-tooltip-face) ではないことに注意
```

候補文字列のフェイス属性（ 変数 *skk-treat-candidate-appearance-function* による加工など）をそのまま使いたい場合は nil に設定して下さい。

defvar skk-tooltip-mouse-behavior

ツールティ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。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。

シンボル 'follow　マウスポインタをカーソル位置へ移動させてツールティップを表示します。ツールティ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ティ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。

シンボル '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ティップを表示します。ツールティップの表示を終えもてマウスポインタは Emacs フレーム 右上隅のままです。

シンボル '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ティップを表示します。ツールティ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ティ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。

シンボル '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、その位置にツールティップを表示します。

nil　マウスポインタを一切移動せず、その位置にツールティップを表示します。ツールティップのテキストとマウスポインタが重なったり、うまくツールティップが表示できなかったりする場合があります。

defvar skk-tooltip-hide-delay

ツールティップを表示する秒数（標準設定は 1000 秒）。この時間が経過すると、ツールティップは自動的に消える。

defvar skk-tooltip-parameters

SKK 独自のフレームパラメータを設定する。標準設定 nil の場合、変数 `tooltip-frame-parameters` が適用される。

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）

defvar skk-show-candidates-always-pop-to-buffer

この値が non-nil であれば、画面を上下に分割したうえで、候補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。

defvar skk-show-candidates-toggle-display-place-char

候補一覧表示中に、候補一覧の表示位置をエコーエリアとバッファとで動的に切り換えることができます。標準設定は `C-f` です。

defvar skk-candidate-buffer-background-color

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（標準設定）。

defvar skk-candidate-buffer-background-color-odd

候補バッファの背景色（奇数ライン）を指定します。

エコーエリアに表示する

標準設定では３つの変数

- *skk-show-inline*
- *skk-show-tooltip*
- *skk-show-candidates-always-pop-to-buffer*

とも nil であり、この状態では候補一覧はエコーエリアに表示[*73] 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。

6.14.5 ▼モードにおける変換候補のハイライト表示

defvar skk-use-face

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 GNU Emacs のオーバーレイ (overlay) の機能を使います[*74] 。

defface skk-henkan-face

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。

なお、この変数よりも変数 *skk-treat-candidate-appearance-function* の設定が優先されます。

[*73] ただし、変数 `frame-width` が不足する場合は、候補バッファに表示します。

[*74] 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持されません。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。

変数 skk-henkan-face には、既存のフェイス[*75] 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。そのために次の関数が用意されています。

defun 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 にスラッシュ / を含めることよって、例えば以下の例のように決定されます。

```
(setq skk-henkan-face (skk-make-face 'DimGray/PeachPuff1))
```

上記の場合、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。

```
(setq skk-henkan-face (skk-make-face 'RosyBrown1))
```

上記の場合、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。

6.14.6 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。

defvar skk-treat-candidate-appearance-function

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。

- 引数を 2 つ取ること。
- 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。
- 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。
- 返り値は次のいずれかとすること。

[*75] Emacs 標準では default, modeline, region, secondary-selection, highlight, underline, bold, italic, bold-italic があります。

返り値	説明
文字列	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。
（**候補　.　注釈**）	候補は、もう注釈を含まないものとして処理される。 注釈は、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**））	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` と関数 `skk-treat-candidate-sample2` を用意してあります。

ファイル `~/.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)
```

6.14.7 モードラインの装飾

インジケータ

defvar skk-indicator-use-cursor-color

モードラインの左に DDSKK のインジケータを表示（標準設定）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。

入力モードを示すカーソル色に関する設定

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[*76] を設定します。この場合カーソルの色は参照されません。

- GNU Emacs 21 以上（変数 mule-version の値が 5.0 以上の GNU Emacs）の場合
  - skk-emacs-hiragana-face
  - skk-emacs-katakana-face
  - skk-emacs-jisx0208-latin-face
  - skk-emacs-jisx0201-face
  - skk-emacs-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。

インジケータの装飾

インジケータを装飾することができます。

defvar skk-indicator-prefix
インジケータの接頭辞とする文字列を指定します。

defvar skk-indicator-suffix-func
インジケータの接尾語とする文字列を返す関数を指定します。

アイコン

defvar skk-show-icon
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。なお、アイコン表示は関数 (image-type-available-p 'xpm) が t を返す必要があるため、Emacs の種類／実行環境に依存します。

defvar skk-icon
アイコンの画像ファイル skk.xpm へのパス。関数 skk-emacs-prepare-modeline-properties で定義しています。

6.15 ユーザガイダンス関連

6.15.1 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。

[*76] 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。

defvar skk-japanese-message-and-error

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。

defvar skk-show-japanese-menu

non-nil に設定すると、メニューバーを日本語で表示します。

defvar skk-version-codename-ja

non-nil に設定すると、関数 *skk-version* を評価したときのコードネームを日本語で表示します。

6.15.2 冗長な案内メッセージの表示

defvar skk-verbose

non-nil に設定すると、入力中／変換中に冗長なメッセージを表示します。

```
(setq skk-verbose t)
```

▽モード ファンクションキー F1 〜 F10 に割り当てられている機能を表示します。変数 *skk-verbose* の設定と同時に変数 *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)
```

▽モードにおいてキー入力が一定時間（標準では 1.5 秒）なされなかったとき、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。変数 *skk-verbose* の設定と同時に変数 *skk-show-annotation* を non-nil に設定してみてください。

```
(setq skk-show-annotation t)
```

▼モードにおいてキー入力が一定時間 (標準では 1.5 秒) なされなかったとき、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 --------------------
どれを参照?[C-1 C-i]ja.wikipedia [C-2 C-i]en.wiktionary
[C-3 C-i]simple.wikipedia [C-4 C-i]en.wikipedia [C-5 C-i]ja.wiktionary
-------------------- Echo Area --------------------
```

この案内に従って、例えば C-1 C-i を打鍵すると、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] 標準設定のソースを参照
-------------------- Echo Area --------------------
```

この案内に従って C-w を打鍵すれば、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、キー 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。

defvar skk-verbose-wait

冗長な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。

defvar skk-verbose-message-interval

冗長な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冗長なメッセージに切り替えます。

defface skk-verbose-intention-face

「どれを参照?」と「アノテーション」に適用するフェイスです。

defface skk-verbose-kbd-face

[F5] や [C-1 C-i] に適用するフェイスです。

6.16 I-search 関連

6.16.1 起動時の入力モードの指定

defvar skk-isearch-start-mode

インクリメンタル・サーチを起動したときの入力モードをこの変数で指定できます。以下のいずれかのシンボルを指定できますが、変数 *skk-isearch-use-previous-mode* の設定が優先されます。

指定できるシンボル	インクリメンタル・サーチを起動したときの入力モード
nil	カレントバッファで SKK モードが起動されていれば、そのモードを。 起動されていなければアスキーモード。
シンボル 'hiragana	かなモード
シンボル 'jisx0208-latin	全英モード
シンボル 'latin	アスキーモード

defvar skk-isearch-use-previous-mode

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。nil であれば、変数 *skk-isearch-start-mode* の設定が優先されます。

6.16.2 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。

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。

defvar 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。この変数の標準設定は以下のようになっています。

```
"\\(\\s \\|[ \t\n\r\f]\\)*"
```

この変数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 行に渡っている上、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。

6.17 VIP/VIPER との併用

VIPER については Info を参照してください。

VIPER Manual

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[*77] 。

defvar skk-use-viper

non-nil に設定すると、VIPER に対応します。

6.18 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。

- 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。
- 関数 picture-movement-up や関数 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。

脚注

[*77] ちなみに、VIP 3.5 の作者は、SKK の原作者でもある佐藤雅彦氏（京都大学名誉教授）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。GNU Emacs 19, 20 には、VIP 、VIPER とも標準添付します。

第 7 章

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。

7.1 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。

defvar skk-use-azik

non-nil であれば AZIK 拡張が有効となります。ファイル ~/.skk に (setq skk-use-azik t) と書きます。

defvar skk-azik-keyboard-type

AZIK で使うときのキーボードのタイプをシンボルで指定する。

シンボル	キーボードのタイプ
シンボル 'jp106	日本語 106 キーボード（標準設定）
シンボル 'jp-pc98	NEC PC-98 キーボード
シンボル 'us101	英語キーボード
nil	キーボード依存処理を無効にする

azik と skk で仕様が重なる部分があるため、ファイル skk-azik.el では以下のとおり対応しています。

q

AZIK では、撥音「ん」を入力するには q を使うこととされていますが、skk では既に q に関数 skk-toggle-kana を割り当てています。

そのため、ファイル skk-azik.el では関数 skk-toggle-kana の実行を

- 日本語キーボードであれば @ を
- 英語キーボードであれば [を

それぞれ使用します。

@

上記のとおり、関数 skk-toggle-kana の実行には @ （日本語キーボード）や [（英語キーボード）を使用しますが、skk では既に @ には「今日の日付の入力」(**プログラム実行変換**) を割り当てています。

そのため、skk 本来の動作には x を付けて、それぞれ x@ と x[で代用できるようにしてあります。

l

xx

AZIK では、単独の拗音「ゃゅょぁぃぅぇぉゎ」を入力するには l を前置することとされていますが、skk では既に l に「アスキーモードへの切り替え」を割り当てています。

そのため、ファイル skk-azik.el では、拗音のうち「ぁぃぅぇぉ」 の入力については xx を前置することとしています。

- xxa → ぁ
- xxi → ぃ
- xxu → ぅ
- xxe → ぇ
- xxo → ぉ

なお、拗音のうち「ゃゅょゎ」の単独入力は、AZIK 拡張ファイル skk-azik.el では なく、標準ファイル skk-vars.el です。

- xya → ゃ
- xyu → ゅ
- xyo → ょ
- xwa → ゎ

x

skk では、▼モードでの X は関数 skk-purge-from-jisyo を実行しますが、AZIK では X は「シャ行」の入力に使われます。

そのため、ファイル skk-azik.el での **誤った登録の削除** は、▼モードで M-x skk-purge-from-jisyo を実行してください。

7.2 ACT

ACT (AZIK on Dvorak) 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。

defvar skk-use-act

non-nil であれば、 ACT 拡張が有効となります。ファイル `~/.skk` に `(setq skk-use-act t)` と書きます。

7.3 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。

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ファイル `skk-tutcdef.el` とファイル `skk-tutcode.el` を SKK ソースのトップディレクトリにコピーしてから、あらためて SKK をインストールします。

その後、ファイル `~/.skk` に `(require 'skk-tutcdef)` と書きます。

7.4 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 ソースのディレクトリ `nicola` 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。

https://github.com/skk-dev/ddskk/blob/master/nicola/README.ja

defvar skk-use-kana-keyboard

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)
```

defvar skk-kanagaki-keyboard-type

適切なシンボルを設定することで、かな入力サポートの種類を切り換えます。

シンボル '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)`
シンボル '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)`
シンボル 'nicola-us	
シンボル 'nicola-dvorak	
シンボル 'nicola-colemak	
シンボル '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)`
シンボル 'omelet-us	
シンボル 'omelet-dvorak	
シンボル 'omelet-colemak	
シンボル 'oasys	

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ー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
: 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を参照してください。

第 8 章

そのほかの拡張機能

十分にテストされていない等の理由がありますが、便利・有益と思われる拡張機能を紹介します。

8.1 交ぜ書き変換

ファイル `skk-mazegaki.el` をインストールすると、交ぜ書き変換が可能となります。

- き車 → 汽車
- き者 → 記者
- き社 → 貴社

インストール方法などは、次の投稿を参考にしてください。

http://mail.ring.gr.jp/skk/201111/msg00037.html

第 9 章

SKK に関する情報

9.1 最新情報

DDSKK についての情報は http://openlab.jp/skk/ から得ることができます。

SKK の開発は、 GitHub を利用して行われています。

- https://github.com/skk-dev/ddskk

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。

- https://github.com/skk-dev/ddskk/blob/master/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。

- https://github.com/skk-dev/ddskk/blob/master/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。

9.2 SKK メーリングリスト

SKK Openlab メーリングリストは、統一されたひとつの 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

9.3 SKK 関連ソフトウェア

SKK 関連ソフトウェアに関しては、次の 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

9.4 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います。

注釈: 2018.12 から、SKK 辞書の更新は github のみとしました。 openlab cvs への同期は行いません。

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、

- https://github.com/skk-dev/dict

あて Pull Request をお願います。

9.5 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

9.6 SKK の作者

SKK の原作者は、現京都大学名誉教授の佐藤雅彦氏です。

- http://www.ist.i.kyoto-u.ac.jp/organization/ex-professor.html#Sato

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。ファイル READMEs/Contributors に貢献者名の一覧がありますので、ご覧ください。

9.7 SKK の歴史

SKK の成り立ちと歴史に関しては、次の資料があります。

- SKK の誕生秘話
- SKK = I
- SKK の 25 年
- SKK の歴史（付 Emacs の歴史の一部）

https://github.com/skk-dev/ddskk/blob/master/READMEs/history.md を参照してください。

9.8 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。

9.9 謝辞

DDSKK の開発は、 Ring Server Open Laboratory （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。

(次のページに続く)

(前のページからの続き)

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。

第 10 章

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。

10.1 Introduction

10.1.1 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。うぎゃぁぁぁぁ、やられたぁ」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。

10.1.2 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 の略であるとおり、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、ファイル `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。

10.1.3 Q1-3 DDSKK はどの Emacs で使えますか?

対応する Emacs のバージョンについては **このバージョンの *SKK* について** をご覧ください。

10.1.4 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 macOS でも使えます。

拡張機能については、UNIX の各種コマンド（ **look** や **ispell** など）を前提としているものがいくつかあります。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 macOS 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。

- https://github.com/skk-dev/ddskk/blob/master/READMEs/README.MacOSX.ja

10.1.5 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 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。

10.2 Installation

10.2.1 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。

10.2.2 Q2-2 SKK 辞書はどこにありますか?

SKK **辞書について**

10.2.3 Q2-3 SKK サーバはどこにありますか?

DDSKK は辞書サーバの種類、バージョンには依存していません。

- **辞書サーバの入手**
- http://openlab.jp/skk/skkserv-ja.html

10.3 Customization

10.3.1 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。

通常 . で「.」を、 , で「,」を入力したい場合

■モードに関連するその他の変数 をご覧ください。

一時的に . で「.」を、 , 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` を実行します。

特定のバッファ（例えば tex モード）でのみ「,」「.」に切り替えたい場合は、次の設定を tex 文書ファイルの最後に追加します。

```
% Local Variables:
%   skk-kutouten-type: en
% end:
```

常に . で「.」を、 , で「,」を入力したい場合

変数 `skk-rom-kana-rule-list` を直接変更します。

> **警告:** 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。

ファイル `~/.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。

10.3.2 Q3-2 「ゐ」や「ヰ」 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。かなモード／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。ファイル ~/.skk に

```
(setq skk-rom-kana-rule-list
      (append '(("wi" nil ("ヰ" . "ゐ")))
              skk-rom-kana-rule-list))
```

と書いてみましょう。

一番内側の cons cell は

- 関数 car の評価、つまり「ヰ」が、カナモード
- 関数 cdr の評価、つまり「ゐ」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 の代りに文字列を書くことができます。

10.3.3 Q3-3 検索する辞書を増やしたいのですが。

変数 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）を関数 skk-search-jisyo を使ってリニアサーチし、
- 次に関数 *skk-search-server* 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、見つか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かればまた提示する、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水準の単漢字辞書ファイル SKK-JISYO.JIS2 を検索したい場合はどうすれば良いでしょう？　もう分かりますよね？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。ファイル ~/.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引数である 0 の数字で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辞書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。

ちなみに、ファイル 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。

10.3.4 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。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。

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** 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** を押してみます。X Window System 上であれば、おそらく

```
muhenkan is undefined
```

という答えが返ってくるでしょう。

次に、この名前を使ってファイル ~/.emacs.d/init.el 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。

xmodmap を使う。

X Window System 上では、 **xmodmap** 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。

skk-sticky.el を使う。

変換位置の指定方法

親指シフト入力のエミュレーション機能を利用する。

これは前述した方法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[*1] を修得します 。

[*1] 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、Web サイトを訪れてください。
日本語入力コンソーシアム

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、ファイル ~/.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ると期待されます。

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。

10.3.5 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。

ファイル 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。

その後でファイル ~/.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。

どこにも辞書がインストールされていない場合は

- https://skk-dev.github.io/dict/

から取得します。

10.3.6 Q3-6 チュートリアルが起動できません。

ファイル 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。

その後でファイル ~/.emacs.d/init.el に

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

のように設定します。

10.3.7 Q3-7 C-x C-j で dired が起動してしまいます。

dired-x を読み込むと C-x C-j が関数 dired-jump にバインドされます。この状態でも SKK を C-x C-j で起動したいときは、変数 dired-bind-jump に nil を設定します。

```
(setq dired-bind-jump nil)
```

なお、この設定は dired-x を読み込む前である必要があります。

10.4 Dictionaries

10.4.1 Q4-1 SKK には郵便番号辞書がありますか?

zipcode というディレクトリに入っています。

- https://skk-dev.github.io/dict/

使用方法は

- https://github.com/skk-dev/dict/blob/master/zipcode/README.md

を御覧下さい。

10.4.2 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。

興味のある方は次の url をご覧ください。

- SKK-JISYO.notes

10.4.3 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。

辞書ツール

10.4.4 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

- https://skk-dev.github.io/dict/SKK-JISYO.edict.tar.gz

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` の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-JISYO.E2J
% rm temp
```

できたファイル `SKK-JISYO.E2J` の利用方法は色々ありますが、

```
% skkdic-expr SKK-JISYO.E2J + /usr/local/share/skk/SKK-JISYO.L | \
  skkdic-sort > SKK-JISYO.L
```

などとして、ファイル `SKK-JISYO.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

をご覧下さい。ファイル `SKK-JISYO.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。

10.5 Miscellaneous

10.5.1 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** とファイル skk-look.el を利用すると、色々できますよ。

まず、ファイル ~/.skk で変数 *skk-use-look* を t にセットして Emacs/SKK を立ち上げ直して下さい。

注釈: 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、変数 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。

さぁ、下記のような芸当が可能になりました。

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、変数 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。

英単語をあいまいに変換して取り出す

上記同様、変数 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。

あいまいに変換した後、更に再帰的な英和変換を行う

まず、変数 *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、変数 *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。

10.5.2 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ファイル `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、ファイル `~/.emacs.d/init.el` やファイル `site-start.el` などは読み込まれないことです。標準設

定で変数 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、ファイル 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。

さぁ、ファイル ~/.skk で変数 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。

10.5.3 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。

SKK 関連ソフトウェア

脚注

第 11 章

Indices and tables

- genindex
- search

索引