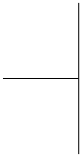


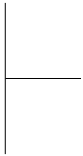


AutoHotKey、 Karabiner-Elements ではじめ るキーバインド設定



1

ごまなつ 著



2019-12-14 第1版 発行

はじめに

皆さんはキーボードに対して不満はありませんか？

- ローマ字入力なら無変換キー、変換キー、カタカナひらがなキーは押しやすい位置にあるのになかなか使わない
- IME キー、Enter キー、BackSpace キーは小指を伸ばさないと押せない
- 各種記号はほぼ小指。最上段まで指を伸ばすのは大変
- Delete キー、PageUp キー、PageDown キー、カーソルキーはホームポジションでは押せない

このような不満が筆者にはありました。これを解決するために、自作キーボード！と言いたところですが、自作キーボードまで手を出せないという方もいると思います。筆者も最初はそうでした。よって、キーバインド設定をできるソフトを導入しました。キーバインドとは、ホットキー、ショートカットキーと同じような意味で特定のキーの組み合わせである処理を実行する設定です。押しやすい無変換キー、変換キー、カタカナひらがなキーのいずれかをトリガーにして、ホームポジションから押せない位置にあるキーを入力できるようにすればほとんどホームポジションから手を動かさずに済むようになります。また、紹介するソフトは、マウス移動、クリックも設定できます。

この本で紹介するのは AutoHotKey と Karabiner-Elements です。Windows ユーザは AutoHotKey、Mac ユーザは Karabiner-Elements を使って下さい。Karabiner-Elements は macOS Sierra 以降に対応しており、Sierra 未満の方は Karabiner を使ってください。

では、快適キーボード操作にするための方法を見ていきましょう。

免責事項

本書に記載された内容は、情報の提供のみを目的としています。したがって、本書を用いた開発、製作、運用は、必ずご自身の責任と判断によって行ってください。これらの情報による開発、製作、運用の結果について、著者はいかなる責任も負いま



せん。



3

3



目次

はじめに	2
免責事項	2
第 1 章 AutoHotkey	5
1.1 AutoHotkey とは	5
1.2 実際にキーバインド設定してみよう	5
1.3 うまくいかないキーボードは ChangeKey を使おう	8
1.3.1 [column]ScrollLock、Pause/Break	10
1.4 便利なキーバインド設定の例	11
1.5 終わりに	14
第 2 章 Karabiner (-Elements)	15
2.1 Karabiner (-Elements) とは	15
2.2 実際にキーバインド設定してみよう	15
2.2.1 既存設定のインポート	15
2.2.2 独自設定	16
2.2.3 to	21
2.2.4 2 連打に処理を割り当てる方法	28
2.3 終わりに	30
著者紹介	31

第 1 章

AutoHotkey

1.1 AutoHotkey とは

AutoHotkey は、ショートカットキーを自分で作成するなど、キーボードをカスタマイズできるスクリプトエンジンです（。トリガーとなるキーの組み合わせや単打、トリガーした後の処理を簡単な文法で設定できます。キーボードのキーの入力を別のキーに変更するだけでなく、文字列操作やファイル操作、マウス操作、GUI プログラムの作成、特定ウィンドウでのみ動作、タイマーなど多彩なコマンドが用意されています。今回の本では、簡単に設定できるトリガーのキーの組み合わせでキー入力、マウス操作を行う方法を紹介します。文字列操作やファイル操作、GUI 作成は英語ですが AutoHotKey の公式リファレンス、日本語がいいなら AutoHotKey Wiki を参照してください。すべてのトリガー、処理が掲載されています。ただし、AutoHotKey Wiki は 2014 年で更新が終わっているため、注意が必要です。

1.2 実際にキーバインド設定してみよう

今回のゴールは、自作キーボードのようにあるキーを押していると他のキーの動作が変わるという処理を実装します。あるキーを押しながら他のキーを押して上下左右カーソル移動、Delete キー、Enter キー、IME 切り替えキー、マウス移動ができるようにしていきます。まずはじめに、AutoHotKey をインストールしましょう。公式サイト (<https://autohotkey.com/download/>) の Download Autohotkey Installer をクリックしてインストーラをダウンロードします。インストーラを起動して、特にこだわりがなければ「Express Installation」をクリックしてください。

次にスクリプトを書いていきます。エディタを開くか、テキストファイルを作成して、以下を書きます。

第 1 章 AutoHotkey

▼ ex1.txt

```
a::b
Return
```

これを保存し、拡張子を.ahk に変更します。すると、緑背景に H が描かれたアイコンに変更され、ダブルクリックで実行できます。何も起こっていないように思えますが、Windows 画面右下のインジケータに AutoHotKey のアイコンが表示されています。つまり、バックグラウンドで実行されています。テキストファイルで「a」キーを入力してみてください。「b」が入力されます。複数行の場合上から順番に実行していきますが、Return で処理を終了します。この例は、「a」キーを「b」に変換するというものです。インジケータから AutoHotKey のアイコンの ex1.ahk を右クリックして Exit をクリックするとこのスクリプトが終了します。スクリプトが動作しているときのみ操作されるので、テキストファイルで「a」キーを入力すると「a」が入力されます。

これで単打のキー入力変更ができました。複数キー組み合わせの時はどうするのでしょうか。ex1.ahk を右クリックして、「Edit This Script」をクリックすると、編集できます。次のように変更します。変更を反映するには右クリックメニューの Reload This Script をクリックするか、もう一度ファイルをダブルクリックして次の画面で OK を押します。

▼ ex1.txt

```
+a::b
Return
```

「a」キーを押すと「a」が入力されます。+ とは何なのでしょう。これは、Shift キーです。Shift キーと a キーの同時押しで b が入力されます。AutoHotKey では、修飾キーを記号で表します。修飾キーとは単独では文字入力や制御など具体的な機能を持たないですが、他のキーとの組み合わせた時に何らかの機能を発揮させることができるキーのことです。英語ではモディファイアキー、モディファイアと呼ばれます。

▼ 修飾キー

```
^      Ctrlキー
!      Altキー
+      Shiftキー
#      Winキー
```

1.2 実際にキーバインド設定してみよう

しかしこれでは、修飾キーとの同時押ししか定義できません（例：a と b の同時押しが定義できない）。そこで、Send 関数を使います。

Send 関数は、キーストロークやマウスクリックを発生させる関数です。上記の内容も、AutoHotKey 処理内部では Send 関数に変更されています。今回は特定のキーの組み合わせで特定のキーを送信する設定をします。

▼ 特定キー送信

```
a & b::Send, ^a
Return
```

とすると、「a」と「b」の同時押しで「Ctrl+c」が入力されるようになります。それを確認した後、「a」を押してみてください。「a」が無効化されているのではないのでしょうか。このように、Send 関数で最初に設定したキーが無効化されることがありますので、無効化されても問題ないキーを使いましょう（そんなキーがないという人は後述する ChangeKey を使いましょう。）。Send 関数では、文字列も送信できますし、連打も定義できます（{押すキー、回数}）。大文字小文字は区別されます。ここまでできれば、キー操作のショートカットキーは実現できます。（「よろしくお願いします。」「お世話になっております。」を一回のキー操作で入力することも可能）

ここで、無変換キーをトリガーにして、同時押しでカーソルキーが入力されるようにします。キーの指定ですが、今までのようにそのキーを指定すればよかったのですが、AutoHotKey は US 配列基準です。US 配列には、無変換キーと変換キー、カタカナひらがなキーは存在しません。よって、キーコードで指定します。キーコードの確認方法は、インジケータの ahk ファイルを右クリック→ Open をクリックします。開いた画面の View → Key history and script info をクリックし、キーコードを確認したいキーを押した後、F5 を押すと、押したキーの履歴が表示されます。

第 1 章 AutoHotkey

VK	SC	Type	Up/Dn	Elapsed	Key
EB	07B		d	18.97	not found
EB	07B		u	0.11	not found
FF	079	s	d	0.86	not found
FF	079	s	u	0.09	not found
FF	070		d	0.94	not found
FF	070		u	0.08	not found
41	01E		d	1.00	a
41	01E		u	0.06	a
74	03F		d	3.27	F5
Press [F5] to refresh.					

▲図 1.1 キーコード確認

SC の行の値でもキーを指定することができます。たとえば、「a」は「sc02A」になります。US 配列に存在しない無変換キー、変換キー、カタカナひらがなキーはキーコードを確認してください。ちなみに、Type の行では入力された文字、Up/Dn の行は d がキーを押した/u がキーを離れたか、Elapsed がひとつ前との経過時間、Key がスクリプトの処理が反映されて入力された文字、Window がどのウィンドウ上の操作だったかです。では、変換キーと j で←を入力してみましょう。

▼ 変換キーを使う

```
sc079 & j::Send, {Blind}{left}
Return
```

どうですか？ 動作しましたか？ 動作しなかった方もいると思います。これは、ドライバによっては無変換キー、変換キー、カタカナひらがなキーのキーコードをうまく AutoHotKey が認識できないことがあるからです (Key が not found)。動作しなかった方、Send 関数で最初書いたキーが無効化されてしまった方は、ChangeKey というソフトを利用します。(動作した方もキー単押しは ChangeKey は GUI で簡単に設定できるので読むといいかも。)

1.3 うまくいかないキーボードは ChangeKey を使おう

ChangeKey は、常駐せずにキー割り当てを変更できるフリーソフトです。Windows の機能として、レジストリを書き換えることでキー割り当てを変更する

1.3 うまくいかないキーボードは ChangeKey を使おう

ことができます。レジストリ書き換えを失敗すると PC のシステムを破壊することがあるので、このソフトでレジストリ書き換えを代行してもらおうと安全だと思います。まずはじめに。窓の杜から ChangeKey をダウンロードします。(https://forest.watch.impress.co.jp/library/software/changekey/) 圧縮形式が LZH なので、解凍ソフトを別にインストールしておいてください。解凍すると、そのフォルダに ChgKey.exe があります。右クリック→管理者として実行をクリックしてください。

変更したいキーをクリックして、クリック後の画面で設定したいキーをクリックします。この例では、CapsLock を Ctrl に変更しています。登録→現在の設定内容で登録しますをクリックして、PC を再起動するとキーが変更されています。戻したい場合は、またこのソフトで変更してください。



▲図 1.2 変更前



▲図 1.3 変更後

さて、変換キーがうまく動作しなかった場合、AutoHotKey が認識できるキーに

第 1 章 AutoHotkey

変更すればいいと考えられます。普段使わず、操作が邪魔にならず認識できるキーにしたいですね。そんなキーがキーボードに存在しています。それは、ScrollLock、Pause/Break です。今回は、ScrollLock を使用します。他の方法としては、キーボードにはないが PC には設定されている仮想キーコードとして F13~F24 キーが存在しています。これを用いる方法もあります。キーコードは前節での内容を参考に調べてください。変更後のキーを選択する画面で、Scan code をクリックして調べたキーコードを設定するとそのキーに設定されます。

1.3.1 [column]ScrollLock、Pause/Break

ScrollLock、Pause/Break の機能を知っていますか？ Excel などの表計算ソフトでカーソルキーを押すと選択セルが動きますが ScrollLock を ON にすると選択セル位置は固定のままスクロールバーが動きます。Excel の動きがおかしくなった、という原因のひとつにこれがあります。Pause/Break キーは、処理を途中で止めるものです。コマンドプロンプトで処理停止に使えます (ctrl+c と同じ処理)。使う場面はほとんどないキーですが、キーボードに残っています。===

AutoHotKey で ScrollLock を使ってキーバインド設定し、ChangeKey で変換キーを ScrollLock に割り当てするとよさそうです。すべてのカーソルキーを実装しましょう。(sc079 で動作した方は ScrollLock は sc079 でも可) ScrollLock はキー名で使えます。

▼ カーソル移動

```
ScrollLock & j::Send, {Blind}{left}    ;←  
ScrollLock & i::Send, {Blind}{up}      ;↑  
ScrollLock & k::Send, {Blind}{down}    ;↓  
ScrollLock & l::Send, {Blind}{right}   ;→
```

筆者はゲームで WASD 移動に慣れているため、右手も同じ形にしました。横一列のほうが慣れているなら、HJKL に設定すると良いですね。これでカーソルキーのキーバインド設定ができました。他のキーを使ってもやってみましょう。他のキーの組み合わせでもやってみよう、となったとき、気を付けることがあります。キーコードの指定は特殊キーに当たるものはキー名を {} で囲む必要があります。既に意味を持っている記号や単語を区別するためです。^! +# {} は囲む必要がある記号です。すでに意味を持っていますよね。

1.4 便利なキーバインド設定の例

筆者が使っているキーバインド設定を紹介します。ChangeKey での設定は図 1.4 に示します。



▲ 図 1.4 変更後

設定を増やしていくと 1 つのファイルにすべて書くのではなく分割したいです。その時、ahk ファイルをひとつずつ実行しているとインジケータが雑多になるし、設定 OFF にしたいときにひとつずつ終了するのは手間です。1 つの ahk ファイルに、他の ahk ファイルを起動する設定を書きましょう。

▼ 一括起動

```
#Include %A_ScriptDir%/hoge.ahk
#include %A_ScriptDir%/huga.ahk
Return
}
```

%A_ScriptDir%はこのファイルがあるファイルパスです。このファイルに書いたファイルを、このファイルと同じ場所に置くのが分かりやすいです。
AutoHotKeyではマウス操作、単語移動、遠いキーを近くに設定しています。まず、マウス操作を設定しましょう。マウスのクリックはMouseClickで実現できます。

```
//emlist[マウスクリック]{
ScrollLock & w::MouseClick, left      ;左クリック
ScrollLock & r::MouseClick, right     ;右クリック
```

left が左、right が右クリックです。left のほかにも条件を記述できます。MouseClick の記述法は、

▼ mouseclick

第 1 章 AutoHotkey

```
MouseClick [, WhichButton, X, Y, ClickCount, Speed, D|U, R]
```

- * WhichButton(left,right,middle,wheelup,wheeldown)
- * X,Yはクリックする座標。ディスプレイ左上の座標が (0, 0) で、マウスカーソルも移動する (省略時は現在位置)
- * ClickCountはクリックする回数 (省略時は1回)
- * Speedは座標指定時に移動する速度。0~100を設定 (0が即時) (マウス移動が速すぎると不都合が発生するソフトが存在)
- * D|UはDが押しっぱなし、Uが押下を離す、省略時はクリック
- * Rは座標指定がカーソル位置からの相対座標になる

例では、WhichButton 以降すべて省略していますが、他にも記述する場合は、指定したい値が記述法での位置に合うように、を足してください。,, と複数並ぶこともあります。マウスの移動は、MouseMove を使います。

▼ マウス移動

```
ScrollLock & s:: MouseMove, -11,0,0,R ;左  
ScrollLock & f:: MouseMove, 11,0,0,R ;右  
ScrollLock & e:: MouseMove, 0,-11,0,R ;上  
ScrollLock & d:: MouseMove, 0,11,0,R ;下
```

というように書きます。

MouseMove の記述法は、

▼ Mousemove

```
MouseMove, [X,Y, Speed, R]
```

- * X, Yは移動先の座標。ディスプレイ左上の座標が (0, 0)
- * Speedは座標指定時に移動する速度。0~100を設定 (0が即時)
- * Rは座標指定がカーソル位置からの相対座標になる

R を書いていない場合はその座標に移動します。R を書くと今の座標から指定した座標だけ移動します。細かい移動で設定していると大きく移動するときに不便なので、大きく移動する場合も設定すると便利です。

▼ 大きなマウス移動

```
ScrollLock & z:: MouseMove, -400,0,5,R  
ScrollLock & v:: MouseMove, 400,0,5,R  
ScrollLock & c:: MouseMove, 0,-400,5,R  
ScrollLock & x:: MouseMove, 0,400,5,R
```

単語移動を設定できます。

1.4 便利なキーバインド設定の例

▼ 単語移動

```
ScrollLock & o::Send, {Blind}~{right}  
ScrollLock & u::Send, {Blind}~{left}
```

エンターキーまで小指を伸ばすのがつらく、親指で押しやすいキーはすべて使っているので同時押しに Enter キーを割り当てました。

▼ エンターキー

```
ScrollLock & Space::Send, {Blind}{Enter}
```

「、」の2連打で「。」が入力できたら楽だと思い、実装しています。

▼ 2 連打設定（長押しでも反応）

```
sc033::  
If (A_PriorHotKey == A_ThisHotKey and A_TimeSincePriorHotKey < 200)  
{  
    Send,{BS}  
    Send,{sc034}  
}  
else{  
    Send,{sc033}  
}  
Return
```

A_PriorHotKey は一回前に押したキー、A_ThisHotKey は今押したキーなので同じキーだと連打しているとわかります。TimeSincePriorHotKey はこの2つのキーの差です。連打が200ミリ秒以下だったら、入力された「、」を消して、「。」を入力し、そうでなければそのまま「、」を入力するというものです。連打を検知する間隔は変更できます。この設定だと、連打でなく長押しにも反応します。別の設定方法だと、

▼ ahkwiki

```
sc033::  
    Keywait, sc033, U ;1回目のキーが押し上げられるのを待つ  
    Keywait, sc033, D T0.2 ;0.2秒待つ。この間に「、」が押されればErrorLevelに0、そうでないなら1が代入  
    If (ErrorLevel=1) ;直前のコマンド=Keywaitがタイムアウトで失敗=1なら  
    {  
        Send,{sc033}  
    }  
    else  
    {  
        Send,{BS}
```

第 1 章 AutoHotkey

```
        Send,{sc034}  
    }  
    return
```

Keywait を使って、押下と離す状態を感知して、2 回目が押されたら入力された「、」を消して、「。」を入力しています。連打を感知する間隔は変更できます。こちらの設定方法だと、長押しには反応しません。

もちろん、キー入力だけではなくアプリケーションの起動も設定できます。Esc2 連打でエディタ起動、Alt2 連打で Alt-F4 も設定しています。

1.5 終わりに

この章を読むことによって、ChangeKey によるキー割り当て変更、AutoHotKey によるキー割り当て変更、キーバインド設定、マウス操作が可能になりました。筆者が行っている設定を紹介しましたが、使用しているキーボードは変換キーがちょうど親指の位置に来るスペースキーが小さいタイプなのでこのような設定になっています。使っているキーボードだけでなく使用者の好み・タイピング方法によって使いやすい設定は異なるので、ぜひ自分にあった設定をみつけてください。他にも AutoHotKey では普通に押した場合と長押しで処理を変えたり、タイマー設定、ループもできるので調べればいろいろなことができます。ぜひ、自分好みの設定に挑戦しましょう。(AutoHotKey wiki:<http://ahkwiki.net>)

第 2 章

Karabiner (-Elements)

2.1 Karabiner (-Elements) とは

Karabiner-Elements とは、Sierra 以降の macOS のキーボードをカスタマイズするためのツールです。Sierra からキーボードドライバの構成が変更されたため、Karabiner-Elements となり、それより前は Karabiner という名前で開発されていました。既存で用意されている設定に、主要エディタのショートカットキーがあり (Vim, emacs, vscode)、とても簡単に設定できます。キーの組み合わせで他のキー入力や、マウス操作、キーバインド設定が独自で定義できます。この本では、Karabiner-Elements を扱います。設定のインポート方法、独自設定の設定方法を紹介します。現在でも開発が進められており、新機能が追加されていますので、現時点での設定です。

2.2 実際にキーバインド設定してみよう

まず、公式サイト (<https://pqrs.org/osx/karabiner/>) からインストールします。設定は変更せずインストール完了まで進めてください。Karabiner-Elements の設定は ~/.config/karabiner/karabiner.json に保存されます。Karabiner-Elements を起動すると、このような画面が出ます。(画像入れる) Simple Modifications では、From key の入力を To key の入力に変換できます。Add item をクリックして、新たなルールを設定してください。消したい場合は、右側の Remove をクリックしてください。Target Device では、

2.2.1 既存設定のインポート

Complex Modifications では、主要エディタのショートカットキー設定 (emacs、vim、vscode など) が既存で設定されています。独自設定を追加する場合は複数キー

第2章 Karabiner (-Elements)

の組み合わせを扱うことができ、押しっぱなしといった複雑なルールも扱うことができます。Add rule をクリックして、Import more rules from the internet(open a web browser) をクリックすると、インポート可能なキーバインド設定の一覧が表示されているサイトが立ち上がります。追加したいキーバインド設定の Import をクリックして、インポートが完了すると、Karabiner-Elements の画面に適用可能なキーバインドの一覧が表示され、Enable をクリックすると適用されます。Rules に追加され、Enable になっているキーバインドの設定が表示されます。

2.2.2 独自設定

キーバインド設定をするなら、自分に合った独自のキーバインド設定をしたいですよね。Karabiner-Elements でも独自キーバインド設定ができます。キーバインド設定は、`~/config/karabiner/assets/complex_modifications` に保存されます。インポートした設定は数列.json というファイル名で保存されます。このディレクトリの json ファイルを読み込んでいるため、自分で json ファイルを作成すると読み込まれます。設定ファイル名は英数字内で自由なのですが、ごくたまに読み込まれないことがあります。その際はファイル名を数列.json としてください。json ファイルが分からなくても、Karabiner-Elements においては簡単な書式なので真似すれば問題ないです。=== [column]json ファイルとは JSON とは JavaScript Object Notation の略で、テキストベースのデータフォーマットです。主要なプログラミング言語には json の生成や読み込みを行うライブラリが存在しているため、データ交換のためのデータフォーマットとして利用されます。書式は

▼ ex1

```
{
  "key": "value",
  "key2": "value2",
  "key3": [true, 123, "value3"]
}
```

まず、全体を {} で囲む必要があります。キーと値を : で区切って並べて書きます。複数にわたる場合は、で区切って並べます。値は、文字列は"で囲み、数値と bool 値はそのまま、配列は [] で囲み要素を、で区切ります。見やすいように改行することが多いですが、すべてを 1 行で書いてもいいです。===

それでは、json ファイルの中身を見ていきましょう。

▼ rule

2.2 実際にキーバインド設定してみよう

```
{
  "title": "Add rulesに表示させる",
  "rules": [
    {
      "description": "descriptionを表示させる"
    }
  ]
}
```

まず、Karabiner-Elements の Rule 追加画面に表示させる設定です。title に設定した文字列が表示され、その中に description が各項目の設定として表示されています。(画像入れる)

設定ファイルを編集後、変更を反映させるには「Complex Modifications」で対象のルールを Enable して、Add rules で再度ルールを適用してください。

コメントは、キーを増やしても問題ないので"comment"のキーを作成する方法があります。

▼ コメントの付け方

```
"description": "descriptionを表示させる",
"comment": "このキーを増やしてここにコメントを書いてもよい"
```

rules は配列になっているので、1 つの rules の中に複数の description 以下の設定を記述できます。title の中に複数の設定が表示されていましたよね。また、1 つの description の中に複数の設定をすることもできます。

▼ karabiner-Elements の設定ツリー

```
"title"
"rules"
  "description"
  "manipulators"
    "type"      //1つ目の設定
    "from"
    "to"
    "type"      //2つ目の設定
    "from"
    "to"
```

実際にキーバインド設定の記述に入ります。基本としては、from で設定したキー入力を受け付けたら、to で設定したキー入力に変更するというものです。設定全体の注意点として、from に設定した入力、通常の動作をせずここで設定した入力をすることに注意してください。ちなみに、to になにも設定しないと何もしないので from に設定したキーを無効化します。また、単純に from、to に 1 つのキーを設定した場合は、キーマップの変更になります。type には、基本的には"basic"を指定しま

第2章 Karabiner (-Elements)

す。マウスの動きをスクロールに変換したい場合のみ"mouse_motion_to_scroll"を指定します。この機能を使う場合は、Karabiner-Elements の Devices で、自分が使っているマウスにチェックを入れておく必要があります。

▼ マウススクロール

```
{
  "description": "Change control + mouse motion to scroll wheel (rev 1)",
  "available_since": "12.3.0",
  "manipulators": [
    {
      "type": "mouse_motion_to_scroll",
      "from": {
        "modifiers": {
          "mandatory": [
            "control"
          ]
        }
      }
    }
  ]
}
```

Control キーを押しながらマウスを動かすと、動かした方向にスクロールようになります。ここからの説明では、"title"/"rules"/"description"/"manipulators"/"type"の部分、直接関係ない部分は省略してサンプルを載せていきます。ここからの設定は、全て type は"basic"です。

▼ 十字キー

```
"type":"basic"
"from":{"key_code":"j"
"modifiers":{"mandatory":["control"]
}
"to":[
{"key_code":"left_arrow"}
]
```

上記のコードは、control-j の組み合わせを指定しています。例の"control"部分に any を指定すると、全てのキー入力を指定します。キー入力ではなく、マウスのボタンを指定する時は key_code ではなく、"pointing_button":"button3"のように指定してください。modifiers はキーの組み合わせを指定します。ここでは、mandatory と optional があります。

mandatory はキーの組み合わせを指定します。この組み合わせが入力されたときに"to"の処理を行います。mandatory には、修飾キーの command, control, shift,

2.2 実際にキーバインド設定してみよう

option, fn, caps_lock, any のどれかを設定することをお勧めします。これら以外だと、mandatory に指定したキーを押しながら key_code に指定したキーを押した場合のみ動作するので、mandatory に指定したキーの入力が少なくとも 1 回入ってしまいます。"pointing_button"の注意点として、button1 は左クリックですので、ここに割り当てると左クリックとして使えなくなります。modifiers を使って同時押しに使いましょう。

"optional"は、"from"の時に受け付けたキーを、"to"の時にも引き継ぐキーを指定します。"any"だと、全てのキー入力を引き継ぎます。ちなみに、Shift、control など左右に存在しているキーは、left_control のようにすると左右区別できます。left、right を書いていない場合は、どちらでも受け付けます。

▼ any

```
"from":{
  "key_code":"j"
  "modifiers":{
    "mandatory":["control"]
    "optional":["any"]
  }
"to":[
  {"key_code":"command"}
]
```

control-A のキー入力を受け付けると、command-A にするといったものです。any の部分には、一つのキーを指定することができます。"caps_lock"を指定すると、CapsLock が ON 状態でも変換を実行するようになります。Karabiner-Elements は、CapsLock が ON だと、CapsLock が押されていると判断しています。注意点として、"mandatory"と"optional"に同じキーを指定すると引き継ぎできなくなります。

"simultaneous"は、複数キーの同時押しに何らかの処理を割り当てたいときに使います。同時押しの許容時間は、Complex Modifications/Parameters/simultaneous_threshold_milliseconds で設定できます。デフォルトでは、50 ミリ秒になっています。

▼ 同時押し

```
{
  "manipulators": [
    {
      "type": "basic",
      "from": {
        "simultaneous": [
          { "key_code": "j" },
          { "key_code": "k" },

```

第2章 Karabiner (-Elements)

```
        { "key_code": "l" }
      ],
      "to": [
        { "shell_command": "open -a TextEdit" } //動作未確認
      ]
    }
  ]
}
```

jkl の同時押しでテキストエディタを起動します。simultaneous の下に、オプションを書けます。

▼ simultaneous のオプション

```
"simultaneous_options":{
  "detect_key_down_uninterruptedly",
  "key_down_order",
  "key_up_order",
  "key_up_when",
  "to_after_key_up"
}
```

この5つがあります。* "detect_key_down_uninterruptedly" true か false で指定します。true だと、同時押しの途中で違うキーを押しても、同時押しに設定された操作の変換を行った後、違うキーを押します。上の例だと、jksl と押した場合、jkl を押した動作をした後、s が入力されます。* "key_down_order" "strict", "strict_inverse", "insensitive" のどれかで指定します。発火するキーを押す順番を明確に決める設定です。"strict" が記述した順番、"strict_inverse" が記述と反対の順番です。"insensitive" は、順番を問わなくなります。* "key_up_order" "strict", "strict_inverse", "insensitive" のどれかで指定します。key_up_order を指定すると、指定キーを押した時点では変換されず、いずれかのキーを一つ離れた時点で発火します。"strict" だと "simultaneous" で最初に記述したキー、"strict_inverse" だと最後に記述したキー、"insensitive" だと記述したどのキーを離しても発火します。* "key_up_when" "all" か "any" を指定します。同時押しした全てのキーを離れたことにするタイミングを設定します。"all" だとすべて離れたとき、"any" だとどれか一つを離れた時点で記述した全てのキーを離れたことにします。* "to_after_key_up" 同時押ししたキーを離れた時点で処理を行いたい場合に設定します。例えば、フラグ処理に使えます。

▼ set_variable

2.2 実際にキーバインド設定してみよう

```
"simultaneous_options": {
  "to_after_key_up": [
    { "set_variable": { "name": "hoge", "value": 0 } }
  ],
  "to": [
    { "set_variable": { "name": "hoge", "value": 1 } }
  ]
}
```

"set_variable"で代入ができます。このようにすれば、押したときに"value"を1、離したときに0にできます。

2.2.3 to

"to"についてですが、キーの指定方法に関しては"from"と同様です。配列なので、複数の処理を設定できます。

▼ to の例

```
"to": [
  { "key_code": "control" },
  { "key_code": "japanese_eisu" },
  { "key_code": "control" }
]
```

これで、from の設定が発火したら control を 2 回と japanese_eisu を押すことができます。"pointing_button"もあります。

▼ ダブルクリック

```
"to": [
  { "pointing_button": "button1" },
  { "pointing_button": "button1" }
]
```

button1 は左クリックなので、キー操作にダブルクリックを割り当てることができます。

▼ シェルコマンド 1

```
"to": [
  { "shell_command": "open -a TextEdit" } //動作未確認
]
```

▼ シェルコマンド 2

第2章 Karabiner (-Elements)

```
"to": [
  { "shell_command": "open \"aaa\"" } //動作未確認
]
```

キーの組み合わせにシェルコマンドを割り当てることができます。打ち込むコマンドにエスケープシーケンスが含まれる場合は\でエスケープ処理をする必要があります。

"modifiers"は、"from"の場合とは異なり、"mandatory"、"optional"の設定は不要です。

▼ to の場合の同時押し

```
"to": [
  {
    "key_code": "down_arrow",
    "modifiers": ["command", "shift"]
    "lazy": true
  }
]
```

"lazy":true とすると、変換が実行されるタイミングが"from"が発火したタイミングではなく"from"が発火したキーを押しながら他のキーを押したタイミングになります。

"select_input_source"の"input_source_id"に特定の値を指定することでIMEの状態を指定することができます。IMEは特殊キーなので、値を確認します。Karabiner-ElementsのEventViewer → variablesでキーを押したときの値を見ることができます。アップル日本語入力か、Google日本語入力など入力方式で値が変わります。input_source_identifiersの項目で"input_source_id"の値を確認してください。"language"が"ja"の時はIMEオンです。

▼ IME 変更

```
//標準IMEでオンにする
"to": [
  {
    "select_input_source": {
      "input_source_id": "~com\\.apple\\.inputmethod\\.Kotoeri\\.Japanese$"
    }
  }
]
```

"mouse_key"を使うと、マウス移動、ホイールスクロールが割り当てられます。クリックは前述の"pointing_button"で割り当てます。

2.2 実際にキーバインド設定してみよう

▼ マウス移動

```
"to": [ { "mouse_key": { "x": -10 } } ,  
        { "mouse_key": { "y": -10 } }  
]
```

▼ ホイールスクロール

```
"to": [  
  { "mouse_key": { "horizontal_wheel": 64 } }  
]
```

x,y は座標のことなので、x が横方向（正で右、負で左）、y が縦方向（正で下、負で上）移動です。ホイールは、"horizontal_wheel"は左右のスクロール（正で右、負で左）、"vertical_wheel"は上下のスクロール（正で下、負で上）です。

"to_if_alone"は、単独で押したときのみ、設定に変換するというものです。

▼ 単独押しの設定

```
"from": {  
  "key_code": "left_command",  
  "modifiers": {  
    "optional": [ "any" ]  
  }  
},  
"to": [  
  { "key_code": "left_command" }  
],  
"parameters": {  
  "basic.to_if_alone_timeout_milliseconds": 2000,  
},  
"to_if_alone": [  
  { "key_code": "japanese_eisu" }  
]
```

左 Command キーを単独で押した場合は英数キーとして認識し、他のキーと組み合わせると左 Command +押したキーとして認識する設定です。2000 ミリ秒押し続けると、英数への変換はキャンセルされます。

"to_delayed_action"はコードで説明します。

▼ to_delayed_action

```
"from": {  
  "key_code": "x",  
  "modifiers": { "mandatory": [ "control" ] }  
},
```

第2章 Karabiner (-Elements)

```
"to": [ { "set_variable": { "name": "ctrl-x", "value": 1 } } ],
"to_delayed_action": {
  "to_if_invoked": [
    { "set_variable": { "name": "ctrl-x", "value": 2 } }
  ],
  "to_if_canceled": [
    { "set_variable": { "name": "ctrl-x", "value": 0 } }
  ]
}
```

ctrl-x を押すと"ctrl-x"に 1 を代入し"to"の内容を実行します。そのまま何も押さないと 2 を代入します。ctrl-x に続けて何か別のキーを押すと 0 が代入されます。"to_if_invoked"が実行されるまでの時間は Complex Modifications → Parameters で指定できます。これをどのように使うかということ、emacs などにある"ctrl-x ctrl-s"のキーバインド設定をするために使います。後述する"condition"を使って実装します。

"to_after_key_up"は、"from"で設定したキーから手を離れたときに実行する処理を書きます。設定した変数の値を初期化するのによく使います。

▼ to_after_key_up

```
"to_after_key_up": [
  { "set_variable": { "name": "enthumble_mode", "value": 0 } }
]
```

"to_if_held_down"は、"from"で設定したキーを押し続けたときに実行する処理を書きます。"repeat"を true に指定すると、処理が実行され続けます。

▼ to_if_held_down

```
"parameters": { "basic.to_if_held_down_threshold_milliseconds": 1000 },
"from": {
  "key_code": "q",
  "modifiers": { "mandatory": [ "command" ], "optional": [ "caps_lock" ] }
},
"parameters": { "basic.to_if_held_down_threshold_milliseconds": 1000 },
"to_if_held_down": [
  {
    "key_code": "q",
    "modifiers": [ "command" ],
    "repeat": false
  }
]
```

command-q は楽ですが、閉じる必要のないアプリケーションを閉じないために 1 秒間押し続けしないと処理を実行しないようにしています。

2.2 実際にキーバインド設定してみよう

"conditions"は、一定の条件を満たすときのみに処理を行いたいときに指定する項目です。いわゆる if 文です。

- device_if、device_unless

特定のキーボードの時、特定のキーボードでないときを条件にしたい場合は、"identifiers"に、"vendor_id"と"product_id"を指定します。"vendor_id"と"product_id"は、Karabiner-Elements の Devices タブで確認できます。

▼ 特定デバイス

```
"conditions": [
  {
    "type": "device_if",
    "identifiers": [
      {
        "vendor_id": 1278,
        "product_id": 515
      }
    ]
  }
]
```

- frontmost_application_if、frontmost_application_unless

特定のアプリの時、特定のアプリでないときを条件にしたい場合は、アプリを指定します。アプリの名称は"karabiner-EventViewer"で調べるか、シェルで `osascript -e 'id of app "(調べたいアプリの名称)"` コマンドを実行して調べます。"karabiner-EventViewer"を立ち上げて、調べたいアプリを開くと Frontmost Application 画面にアプリの名称が表示されます。アプリの名称は、正規表現でも指定できます。

▼ 特定アプリケーション

```
"conditions": [
  {
    "type": "frontmost_application_if",
    "bundle_identifiers": [ "^com\\.apple\\.Safari" ]
  }
]
```

- variable_if、variable_unless

変数が特定の値の時、特定の値以外を条件にしたい場合は、押した時と離れたときに異なる数値を"set_variable"で設定して条件に使います。

▼ 特定変数

第 2 章 Karabiner (-Elements)

```
"from": {
  "key_code": "s",
},
"to": [
  {
    "set_variable": {
      "name": "mode", "value": 1
    }
  }
],
"to_after_key_up": [
  {
    "set_variable": {
      "name": "mode", "value": 0
    }
  }
],
},
{
  "type": "basic",
  "from": {
    "key_code": "h",
  },
  "to": [
    { "key_code": "left_arrow" }
  ],
  "conditions": [
    {
      "type": "variable_if",
      "name": "mode",
      "value": 1
    }
  ]
}
```

S を押したときに 1、離したときに 0 を設定しています。"name"は何でも良いです。H を押したとき、mode 変数が 1 ということは S が押しっぱなしなので、S を押しながら H を推したときに"left_arrow"に変換します。

* input_source_if、input_source_unless IME が指定した状態の時、指定した状態でないときを条件にしたいときに使います。同じキーを押したときに IME の状態によって処理内容を変更したい場合に使います。IME 変更をトグル処理にしたいときに使うことが多いです。

▼ 特定 IME

```
{
  "type": "basic",
  "from": { "key_code": "Japanese_eisuu" },
  "to": [{"key_code": "Japanese_kana"}],
  "conditions": [
    {
      "type": "input_source_if",
```

2.2 実際にキーバインド設定してみよう

```
    "input_sources": [{ "language": "en" }]
  }
},
```

キーボードに含めていいのかは意見が分かれるでしょうが、タッチパッドに何本触れているかという情報を取得できます。全体だけでなく、左右半分、上下半分を分けて取得できます。キーを押さずにタッチパッドに触れるだけで良いので、押しながらの処理がもっと簡単な操作でできるようになります。この機能を使う際は、まず Karabiner-Elements の設定画面の「Misc」から「Open Karabiner-MultitouchExtension app」を起動しておく必要があります。((画像入れる))

このアプリが起動していると、以下の変数が格納されます。

- multitouch_extension_finger_count_total

タッチパッドに触れている指の本数* multitouch_extension_finger_count_left_half_area
タッチパッドの左半分に触れている指の本数* multitouch_extension_finger_count_right_half_area
タッチパッドの右半分に触れている指の本数* multitouch_extension_finger_count_upper_half_area
タッチパッドの上半分に触れている指の本数* multitouch_extension_finger_count_lower_half_area
タッチパッドの下半分に触れている指の本数

そのうえで、"variable_if"を使って変数の値に応じた処理を記述していきます。

▼ タッチパッド

```
{
  "type": "basic",
  "from": {
    "key_code": "j",
    "modifiers": {
      "optional": ["any"]
    }
  },
  "to": [
    { "key_code": "up_arrow" }
  ],
  "conditions": [
    {
      "type": "variable_if",
      "name": "multitouch_extension_finger_count_total",
      "value": 1
    }
  ]
}
```

タッチパッドに1本の指で触れているとJキーを↑に変換する設定です。実際に操作するときは、どちらかの親指で触れるとやりやすいです。

第2章 Karabiner (-Elements)

キーバインド設定とは直接関係ないですが、特定のキーボードが接続されている場合に内蔵キーボードを無効化する機能があります。ですが、タッチパッドは無効化できません。この機能を使うときは、Karabiner-Elements の設定画面の「Device」から「Advanced」にある「Disable the built-in keyboard while one of the following selected device is connected.」で、キーボードを指定します。

2.2.4 2 連打に処理を割り当てる方法

ここまでの設定の紹介では思いつきにくいですが便利な設定である、キーの 2 連打に処理を割り当てる方法を紹介します。"set_variable"による変数の値変更で"condition"の条件判定を用いて実現します。アプリケーションを起動する設定は以下ようになります。

▼ アプリ起動

```
"manipulators": [
  {
    "type": "basic",
    "from": { "key_code": "left_control" },
    "to": [
      { "shell_command": "open -a 'safari'" }
    ],
    "conditions": [
      { "type": "variable_if", "name": "left_control_key", "value": 1 }
    ]
  },
  {
    "type": "basic",
    "from": {
      "key_code": "left_control",
      "modifiers": { "optional": [ "any" ] }
    },
    "to": [
      { "set_variable": { "name": "left_control_key", "value": 1 } },
      { "key_code": "left_control" }
    ],
    "to_delayed_action": {
      "to_if_invoked": [
        { "set_variable": { "name": "left_control_key", "value": 0 } }
      ],
      "to_if_canceled": [
        { "set_variable": { "name": "left_control_key", "value": 0 } }
      ]
    },
    "conditions": [
      { "type": "variable_if", "name": "left_control_key", "value": 0 }
    ]
  }
]
```

前半部分は value が 1 のときに実行するので、1 回目に押したときは読み飛ばしま

2.2 実際にキーバインド設定してみよう

す。後半部分は、左のコントロールと何かのキーが押されたときに value を 1 にして、指定した時間何もしなければ 0、他のキーが押されたら 0 にします。2 連打したときは value が 1 のため、前半部分を実行します。という仕組みです。これを文章の入力に使用するときには工夫が必要です。たとえば、「、」キーを 2 回押すと「。」を入力する設定は、「、」が入力されてしまいます。なので、消す必要があります。

▼ 「、」2 回で「。」を入力

```
{
  "description": "Double tap 'keypad_period' to 'comma'",
  "manipulators": [
    {
      "type": "basic",
      "from": { "key_code": "keypad_period" },
      "to": [
        { "key_code": "delete_or_backspace" },
        { "key_code": "comma" },
        { "set_variable": { "name": "press_period_key", "value": 0 } }
      ],
      "conditions": [
        { "type": "variable_if", "name": "press_period_key", "value": 1 }
      ]
    },
    {
      "type": "basic",
      "from": { "key_code": "keypad_period" },
      "to": [
        { "set_variable": { "name": "press_period_key", "value": 1 } },
        { "key_code": "keypad_period" }
      ],
      "to_delayed_action": {
        "to_if_invoked": [
          { "set_variable": { "name": "press_period_key", "value": 0 } }
        ],
        "to_if_canceled": [
          { "set_variable": { "name": "press_period_key", "value": 0 } }
        ]
      },
      "conditions": [
        { "type": "variable_if", "name": "press_period_key", "value": 0 }
      ]
    }
  ]
}
```

大体は一つ前に挙げたものと同じです。ですが、前半部分に BackSpace を一回押す処理を入れたことで 2 連打するとピリオドが打てたように見えるというものです。

第2章 Karabiner (-Elements)

2.3 終わりに

この章を読むことによって、Karabiner-Elements によるキー割り当て変更、キーバインド設定、マウス操作が可能になりました使っているキーボードだけでなく使用者の好み・タイピング方法によって使いやすい設定は異なるので、ぜひ自分にあった設定を見つけてください。

著者紹介

第 1 章 ひつじ / @mhidaka

ひつじだよ～

AutoHotKey、Karabiner-Elements ではじめるキーバインド設定

2019 年 12 月 14 日 技書博

著 者 ごまなつ
印刷所 日光企画

(C) 2019 ごまなつぷろじえくと