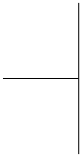


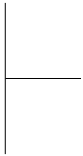


# AutoHotKey、 Karabiner-Elements ではじめ るキーバインド設定



1

ごまなつ 著



2019-12-14 第1版 発行

# はじめに

この本を手にとっていただきありがとうございます。

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、Esc キーは小指を伸ばさないと押せない
- 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

これを解決するために、自作キーボード！と言いたいところですが、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。その時、キーバインド設定をす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。

筆者は押しやすい位置にある無変換キー、変換キー、カタカナひらがなキーをトリガーにして、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ると考え実装しました。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 以降に対応しています。Sierra 未満の方は Karabiner を使ってください。

現時点でも開発が進められているため、2019/12/14 時点における機能の紹介となります。では、快適キーボード操作するための方法を見ていきましょう。

2019 年 12 月ごまなつ

## 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。

# 目次

はじめに	2
免責事項	3
<b>第 1 章 AutoHotkey</b>	<b>5</b>
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
ScrollLock、Pause/Break	10
1.4 便利なキーバインド設定の例	10
1.5 ホットストリング	15
1.6 AHK-Studio	16
1.7 終わりに	17
<b>第 2 章 Karabiner-Elements</b>	<b>18</b>
2.1 Karabiner-Elements とは	18
2.2 実際にキーバインド設定してみよう	18
2.2.1 既存設定のインポート	18
2.2.2 独自設定	19
2.3 終わりに	35
あとがき	36
著者紹介	37

## 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、別ソフトの起動を行う方法を紹介します。文字列操作やファイル操作、GUI 操作などをしたい方は英語ですが 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。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

ここでは、AutoHotKey を用いた設定例を示していきます。最終的には、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、特にこだわらなければ「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。エディタを開き、次の内容を書きます。

## 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。上から順番に実行していき、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設定した処理に変更されるので、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、出てくる画面で 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 を使います。

Send は、キーストロークやマウスクリックを発生させたり、文字列を送信するコマンドです。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 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 (Send, {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、キー操作のキーバインドは実現できます。（「よろしく願いいたします。」「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）

次に、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、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、キーの名前では指定できず、スキャンコードで指定します。スキャン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、スキャン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。

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

## 第 1 章 AutoHotkey

VK の値、SC の値、もしくはその両方でもキーを指定することができます。たとえば、「a」は「sc02A」、変換キーは「sc079」になります。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 がキーを離したか、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。「;」「:」「,」「\」の 4 つのキーはエスケープシーケンスのため、スキャンコードで指定することをお勧めします。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スキャンコードをうまく AutoHotKey が認識できないことがあるからです。動作しなかった方、Send 関数で最初にし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であれば 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

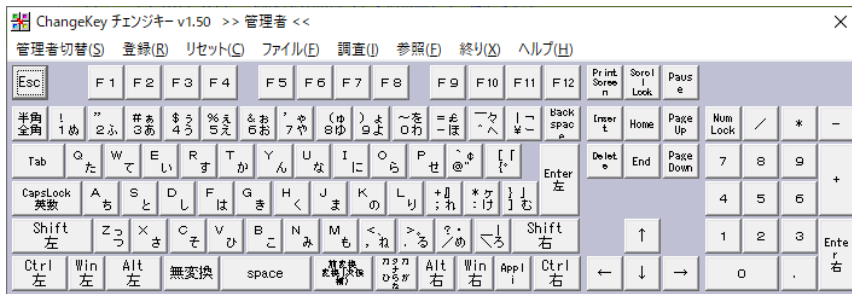
ChangeKey は、常駐せずにキー割り当てを変更できるフリーソフトです。Windows の機能として、レジストリを書き換えることでキー割り当てを変更することができます。レジストリ書き換えの場所を間違えると PC のシステムを破壊することがあるので、このソフトでレジストリ書き換えを代行してもらおうと安全だと思います。

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>）圧縮形式が LZH なので、LZH ファイルを解凍できる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。

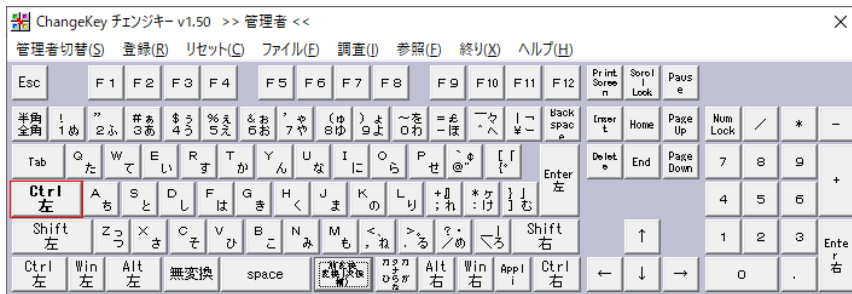
変更したいキーをクリックして、クリック後の画面で設定したいキーをクリックします。この例では、CapsLock を Ctrl に変更しています。登録→現在の設定内容で登録しますをクリックして、PC を再起動するとキーが変更されています。戻したい場合は、またこのソフトで変更してください。



### 1.3 うまいかないキーボードは ChangeKey を使おう



▲ 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、キーボードには存在しないが 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。仮想キーコード、スキャンコードは前節での内容を参考に調べてください。ChangeKey の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スキャンコードを設定します。

## 第 1 章 AutoHotkey

### 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。Send で送信するキー指定で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。また、BackSpace、Space など文字列で指定する場合は、{} で囲まないとその文字列が送信されてしまいます。

## 1.4 便利なキーバインド設定の例

キーバインド設定を増やしていくなかで、1 つのファイルにすべて書いていると巨大なファイルになりメンテナンスしにくくなります。そのため、いくつかのファイルに分割したくなります。分割した後、ahk ファイルをひとつずつ実行しているとインジケーターが雑多になるし、設定 OFF にしたいときにひとつずつ終了するのは手間

## 1.4 便利なキーバインド設定の例

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起動したいファイルを、このファイルと同じ場所に置くのが分かりやすいです。

さて、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。遠くにあるキーをホームポジションから押しやすい場所に変更しています。



▲ 図 1.4 変更後

AutoHotKey ではマウス操作、単語移動、カーソルキーを設定しています。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。

### ▼ 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 が左クリック、right が右クリックです。left のほかにも条件を記述できます。

## 第 1 章 AutoHotkey

### ▼ mouseclick

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があると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

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があると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。細かい移動の設定のみだ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します。違和感を感じない速度に設定します。

### ▼ 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}
```

Enter 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。この設定は2種類の例を示します。

### ▼ 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 は今押したキーが記録されます。よって、この2つが同じキーだと連打しているとわかります。TimeSincePriorHotkey はこの2つのキーが押下された時間の差です。連打が200ミリ秒以下だったら、入力された「、」を消すため BackSpace を一回押して「。」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。連打を感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。もう一つの例は次です。

## 第 1 章 AutoHotkey

### ▼ 2 連打設定（待ちが発生する）

```
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}
        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「、」を消すため BackSpace を一回押し、「。」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。しかし、3 行目で指定した待つ時間待たないと「、」が入力されないので、タイピングが速い方は違和感を感じるかもしれません。

どちらの設定例でも、3 回以上の連打に同様の動作をします。好みのほうを使ってください。筆者は前者の設定を使っています。

外部プログラムの起動も設定できます。アプリケーションの起動は次のように書きます。

### ▼ Run

```
Run, Target, WorkingDir, Max|Min|Hide

* Targetはファイル名、ファイルパス、コマンドラインの文字列、URLを指定
* WorkingDir, Max|Min|Hideは省略可
* WorkingDirは、起動するプログラムの作業ディレクトリを指定。省略時は起動するプログラムの場所
* Max|Min|Hideは起動状態を指定。最大化・最小化・非表示。
```

### ▼ メモ帳起動

```
Pause::
If (A_PriorHotKey == A_ThisHotKey and A_TimeSincePriorHotkey < 300)
{
    Process,Exist,notepad.exe
    If ErrorLevel <> 0
        WinActivate,ahk_pid %ErrorLevel%
    else

```

## 1.5 ホットストリング

```
Run, notepad
}
else
{
    Send, ScrollLock
}
Return
```

Run の後にアプリケーションのファイルパスを指定すれば任意のエディタが起動できます。前半部分の Process, Exist, notepad は notepad というプロセスがあれば最前面に表示し、なければ notepad を起動するというものです。

2 連打の設定で注意しなければならないのは、単押しでそのキーを送信して 2 連打で他の処理を割り当てるという設定は難しいことです。例えば、Esc キーに 2 連打でメモ帳起動、単押しで Esc 送信を割り当てたとします。Esc キーが押された時の処理はこの一連の処理に設定されており、Esc キーの処理ではありません。よって、単押しとして送信された Esc キーをトリガーにまた一連の処理を行い、同時押しした処理が実行されてしまいます。「,」2 連打で「。」は、違うキーを押しているので問題ありません。例では、普段使わない Pause キーをトリガーにし、単押しの場合は Scrolllock キーを送信しています。

## 1.5 ホットストリング

ホットストリングは、指定した文字列が入力されたあと、終了文字が入力されるとその文字列を削除して設定した文字列を入力するというものです。つまり、置換ですね。デフォルトの終了文字は、-()[]{}';"/\,.,?!{Enter}{Space}{Tab} です。追加することもできますし、終了文字を押さなくとも置換を実行することもできます。

### ▼ ホットストリングの例

```
#Hotstring EndChars ;この後に関した文字が終了文字になる。全てのホットストリングに  
影響する  
::btw::by the way
```

英語圏のテキスト入力省力化を想定されているのか、置換後の文字列に 2 バイト文字を使うことができません。日本語環境では、クリップボードを介して置換することで実装できます。

### ▼ ホットストリングの例（日本語）

```
::kita-::  
Clipboard = ｷﾀ━━━━━━(´▽`)━━━━━━ !!!!!  
Send, ^v
```

## 第 1 章 AutoHotkey

### Return

終了文字を押さずとも置換を実行するなどの設定は、最初の::の間にオプションを設定します。\*を書くことで終了文字なしで置換を実行します。日本語環境でホットストリングを使用する注意点としては、次のものがあります。

- テキスト入力欄以外でも置換を実行するので予期せぬ動作をする可能性がある
- IME が有効でも動作し、ひらがなに変換されおかしくなる。また、未確定文字列の数が英語と異なるため、余分に自動削除される。そのため、Ktのように先頭を大文字で母音を含まない文字列にし、オプションに C を入れ大文字小文字を区別するようにすると安全
- 文字列の上限は 40 文字
- 複数のホットストリングにマッチした場合は先に指定されたものが優先される

### ▼ ホットストリングの例（終了文字なし、母音なし）

```
:*:Kt-::  
Clipboard = ｷｰ————(´▽´)———— !!!!!  
Send,~v  
Return
```

このホットストリング機能でスクリプトを書くこともできます。

### ▼ Web ブラウザをページ開いた状態で起動

```
:*:exmprun::  
Run, http://example.com/  
Return
```

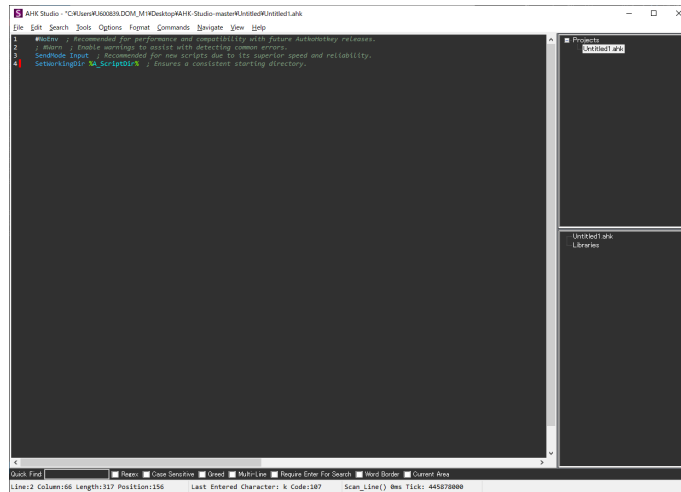
ブラウザを起動して、指定した URL を開けます。http://example.com/は例として用いた URL なので、使うときは自分が開きたい Web ページの URL を設定してください。テキスト入力欄でなくても実行することを逆手に取ります。このように、ショートカットキー代わりに使うこともできます。

## 1.6 AHK-Studio

コーディングをするときは統合開発環境（IDE）を使うことが増えてきました。IDE の利点の一つに、オートコンプリート（自動補完）があります。これがあると便利ですね。AutoHotKey にも、オートコンプリートができるソフトがあります。それが、AHK-Studio です。全ての ahk ファイルを切り替えて編集できますし、自分で設定を複数作ってみようと考えた方は是非使ってください。



## 1.7 終わりに



▲ 図 1.5 AHK-Studio

## 1.7 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、特定のウィンドウでのみ動作する設定をしたり、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、自分好みの設定を作り出してください。

## 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 が開発されました（それより前は Karabiner が開発されていました）。この本では、Karabiner-Elements を扱います。

既存で用意されている設定をインポートすることができ、その中に主要エディタのショートカットキー（Vim, Emacs, Visual Studio Code）があり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。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。

1  
8

### 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。

#### 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（Vim, Emacs, Visual Studio Code など）といった既存の設定がインポートするだけで使うことができます。独自設定も追加でき、追加する場合は複数キーの組み合わせを扱

## 2.2 実際にキーバインド設定してみよう

え、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 には追加されて、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ています。)

Karabiner-Elements は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。また、json ファイル内容の形式が正しくなかった場合も読み込まれません。

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。

#### ▼ json ファイルの書式例

```
{
  "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。キーと値の組み合わせが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、配列は要素を「,」で区切り、[ ]で囲みます。{ }でも[ ]でも、最後の要素に「,」は要らないです。閉じ括弧の前には「,」はいらないということです。

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。括弧の数を間違えがち(開いた括弧を閉じていない)なので、改行して括弧の中身を一段階インデントし、見やすくすることが多いです。

## 第 2 章 Karabiner-Elements

それでは、json ファイルの中身を見ていきましょう。

### ▼ rules

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

Karabiner-Elements の Rule 追加画面に表示される部分です。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 の設定ツリーの例

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

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。

## 2.2 実際にキーバインド設定してみよう

設定全体の注意点として、**from** に設定した入力は、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。もう、**from** に設定した入力は **to** の処理し  
かしないということです。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。

### ▼リスト 2.1 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"
          ]
        }
      }
    ]
  ]
}
```

リスト 2.1 では、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  
クロールするようになります。

ここからの説明では、**"title"/"rules"/"description"/"manipulators"/"type"** の直  
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。

### ▼リスト 2.2 カーソルキー

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

## 第 2 章 Karabiner-Elements

リスト 2.2 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 ("mandatory":["any"]) 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, 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"は button1 が左クリック、button2 が右クリック、button3 がホイールクリックです。注意点として、これらのボタンの単体押しを from の処理に割り当てると、そのクリックは元々のクリックとして使えなくなります。modifiers を用いて同時押しの一部に使いましょう。(例: from{"pointing\_button":"button1"}) とすると、左クリックは左クリックを検知すると to の処理をするボタンになる。元々の左クリックの機能はなくなる)

"optional"に指定したキーは、"to"の時にも引き継がれます。"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。

### ▼リスト 2.3 any

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

リスト 2.3 は、control と何かのキー入力を受け付けると、command と押したキーを送信するといったものです。この例では、command の同時押しは command を control に置き換えて送信されるという設定です。デフォルトキーバインドは command に割り当てられていますが、control でもできるようになると便利ですね。

## 2.2 実際にキーバインド設定してみよう

"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 の時 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。"modifiers"は両方押されている状態を検知したとき、"simultaneous"は指定した時間内に同時押しを検知したときに発火します。

### ▼リスト 2.4 同時押し

```
{
  "manipulators": [
    {
      "type": "basic",
      "from": {
        "simultaneous": [
          { "key_code": "j" },
          { "key_code": "k" },
          { "key_code": "l" }
        ]
      },
      "to": [
        { "shell_command": "open -a TextEdit" }
      ]
    }
  ]
}
```

リスト 2.4 は、jkl の同時押しでテキストエディタを起動します。simultaneous の下の from 内に、オプションを書けます。

### ▼ simultaneous のオプション

```
"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。

## 第 2 章 Karabiner-Elements

false だと、変換処理に入りません。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。省略した場合は false です。

- "key\_down\_order"
  - "strict", "strict\_inverse", "insensitive" のどれかで指定します。キーが押される順番を明確に決めて発火させる設定です。"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。省略した場合は "insensitive" です。
- "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。省略した場合は "insensitive" です。
- "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、次で指定するような変数を用いた処理に使います。

2  
4

### ▼リスト 2.5 set\_variable

```
"simultaneous_options": {  
  "to_after_key_up": [  
    { "set_variable": { "name": "hogehoge", "value": 0 } }  
  ],  
  "to": [  
    { "set_variable": { "name": "hogehoge", "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、離れたときに 0 にできます。

### to

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。



## 2.2 実際にキーバインド設定してみよう

### ▼リスト 2.6 to の例

```
"to": [
  {"key_code": "a"}
  {"key_code": "japanese_eisu"}
  {"key_code": "a"}
]
```

リスト 2.6 で、from の設定が発火したら a を 2 回と japanese\_eisu を押すことができます。"pointing\_button"も指定できます。

### ▼リスト 2.7 ダブルクリック

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

リスト 2.7 で、button1 は左クリックです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。

### ▼リスト 2.8 シェルコマンド

```
"to": [
  { "shell_command": "open -a TextEdit" }
]
```

リスト 2.8 で、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。

### ▼リスト 2.9 to の場合の同時押し

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

リスト 2.9 で同時押しできます。"modifiers"は、"from"の場合とは異なり"mandatory"、"optional"の設定は不要です。"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。

## 第 2 章 Karabiner-Elements

### ▼リスト 2.10 IME 変更

```
"to": [
  {
    "select_input_source": {
      "input_source_id": "^com\\.apple\\.inputmethod\\.Kotoeri\\.Japanese$"
    }
  }
]
"to": [
  {
    "select_input_source": {
      "input_source_id": "^com\\.apple\\.inputmethod\\.Kotoeri\\.Roman$"
    }
  }
]
]
```

リスト 2.10 は ime 操作です。"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や、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。この例ではアップル日本語入力です。

この方法では、アップル日本語入力や Google 日本語入力といった入力方式まで扱いますが、英数入力とかな入力を切り替えるだけであればもっと簡単にできます。(後述する特定 IME を参照。) "mouse\_key"を使うと、マウス移動、ホイールスクロールを割り当てられます。クリックは"from"の時と同様に"pointing\_button"で割り当てます。

### ▼リスト 2.11 マウス移動

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

### ▼リスト 2.12 ホイールスクロール

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

リスト 2.11 とリスト 2.12 でマウス操作です。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

## 2.2 実際にキーバインド設定してみよう

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。クリックはリスト 2.7 を参考にしてください。

### ▼リスト 2.13 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" }
]
```

"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。リスト 2.13 は、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て押すと左 Command-押したキーとして認識する設定です。この例では、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。

### ▼リスト 2.14 to\_delayed\_action

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

"to\_delayed\_action"は、一定時間待った後の処理や、コマンドがキャンセルされた時の処理を設定できます。リスト 2.14 は、ctrl-x を押すと"ctrl-x"に 1 を代入します。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と、

## 第2章 Karabiner-Elements

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。

### ▼リスト 2.15 to\_after\_key\_up

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

"to\_after\_key\_up" は、"from" で設定したキーから手を離したときに実行する処理を書きます。設定した変数の値を変更するのによく使います。リスト 2.15 でも変数の値を変更していますね。

### ▼リスト 2.16 to\_if\_held\_down

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

"to\_if\_held\_down" は、"from" で設定したキーを指定した時間押し続けたときに実行する処理を書きます。この時間はデフォルトでは 500 ミリ秒です。設定画面の Complex Modifications → parameters で変更できますが、これだと全てのルールに適用されます。"description" ごとに設定する場合は、"parameters"/"basic.to\_if\_held\_down\_threshold\_milliseconds" に設定します。"repeat" を true に指定すると、処理が実行され続けます。command-q は楽ですが、押し続けると閉じる必要のないアプリケーションを閉じてしまいます。リスト 2.16 は、1 秒間押し続けないと command-q の処理を実行しないようにしています。

"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。条件分岐、すなわち 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。

## 2.2 実際にキーバインド設定してみよう

### ▼リスト 2.17 特定デバイス

```
"conditions": [
  {
    "type": "device_if",
    "identifiers": [
      {
        "vendor_id": 12345,
        "product_id": 12345
      }
    ]
  }
]
```

- `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`"で調べるには"`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。

### ▼リスト 2.18 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。

### ▼リスト 2.19 特定変数

```
"from": {
  "key_code": "japanese_eisuu",
},
"to": [
  {
```

## 第 2 章 Karabiner-Elements

```
    "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
    }
  ]
}
```

リスト 2.19 の設定は、英数キーを押したときに 1、離れたときに 0 を設定し、H を押したとき mode 変数が 1 ということは英数キーが押しっぱなしなので英数キーを押しながら H を押したときに "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。

### ▼リスト 2.20 特定 IME

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

## 2.2 実際にキーバインド設定してみよう

},

英数入力になっているときは、英数キーでかな入力に変更するという設定にすれば、英数キーで IME をトグル設定にできます。リスト 2.20 の設定はとても便利です。

### タッチパッドの利用

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、上下半分か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
  - ー タッチパッドに触れている指の本数
- multitouch\_extension\_finger\_count\_left\_half\_area
  - ー 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
  - ー 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
  - ー 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
  - ー 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。

### ▼リスト 2.21 タッチパッド

```
{
  "type": "basic",
  "from": {
    "key_code": "j",
    "modifiers": {
      "optional": ["any"]
    }
  },
  "to": [
    { "key_code": "up_arrow" }
  ],
}
```

## 第 2 章 Karabiner-Elements

```
"conditions": [
  {
    "type": "variable_if",
    "name": "multitouch_extension_finger_count_total",
    "value": 1
  }
]
```

リスト 2.21 は、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。

### 特定キーボードのみ動作

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 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リスト 2.22 です。

#### ▼リスト 2.22 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 } },

```



## 2.2 実際にキーバインド設定してみよう

```
{ "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。

### ▼リスト 2.23 「、」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 } }
        ],

```

## 第2章 Karabiner-Elements

```
    "to_if_canceled": [
      { "set_variable": { "name": "press_period_key", "value": 0 } }
    ],
    "conditions": [
      { "type": "variable_if", "name": "press_period_key", "value": 0 }
    ]
  }
}
```

リスト 2.23 は、大体はリスト 2.22 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。

### JIS キーボードを使っていて、記号の割り当てがおかしいとき

Karabiner-Elements が、JIS キーボードを US キーボードと認識することがあります。そのため、JIS キーボードに見えている記号を入力しても Karabiner-Elements は US キーボードのその位置の記号だと認識することがあります。この時、キーボードの認識を合わせるよう頑張るか、Karabiner-Elements の認識に合わせて設定をすることになります。JIS キーボードと US キーボードの記号の対応表を掲載しておくので、参考にしてください。

▼表 2.1 対応表

JIS キーボード	US キーボード	指定すべきキーコード
^	=	equal_sign
¥	なし	international3
@	[	open_bracket
[	]	close_bracket
:	'	quote
]	\	backslash
\	.	international1
;	;	semicolon
-	-	hyphen
,	,	comma
.	なし	period
/	/	slash

JIS キーボードの見た目のキーを打つ設定をするときに、指定すべきキーコードで指定してください。たとえば、「:」キーで何か処理をするときは、`{key_code:quote}`

にするということです。

### 便利なキーバインド設定の例

筆者が行っている設定方法を紹介します。紹介しているものもあるので、設定方法は割愛させていただきます。変換キーを押しながら `ikjl` で上下左右カーソル、`edsf` で上下左右マウス移動、`wr` で左右クリック、`Space` キーで `Enter` にします。変換キーの単押しは `BackSpace` です。英数キーをトグル設定にし、`Command` の同時押しは `Control` に変換しています。「`,`」の2回押しで「`.`」を入力しています。筆者はメインが `windowsPC` のため、`windowsPC` で `AutoHotkey` によって設定していた設定をそのまま移植しました。

## 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、マウス操作がいろいろな条件の時に設定可能になりました。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。

## あとがき

この本を手にとっていただきありがとうございます。

キーバインド設定の方法を紹介しました。ぜひ、いろいろな自分好みの設定を考えて実装してください。

筆者は、キーバインド設定を考えて実装していくうちに指をなるべく速くに動かさないようにしたくなりました。そのため、キーの数が減っている自作キーボードに移行しました。自作キーボードでは、`qmk_firmware` というファームウェアでキーマップ、キーバインドが設定できます。左右分離したり、キーの数を変えたり、押し心地を変えたり、ぼくのかんがえたさいきょうのキーボードを探しています。自作キーボードも面白いですよ。

話がそれました。キーバインド設定を楽しく考え、楽に、効率的に入力していきましょう！

2019 年 12 月ごまなつ

## 著者紹介

- ごまなつ



メーカ子会社勤務 2 年目。業務は C# で windows アプリを作っています。趣味の PC ゲームからキーボードに興味を持ち、いろいろなキーボードを買った結果、自作キーボードに行きついた人。自作キーボードまでは手が出ないという人にも、キーボード設定で自分好みのキーボードにできることを伝えていきたい。他の趣味はボードゲーム、カードゲーム、ゲームセンター。ゲームセンターの同人誌も出してます。

Twitter @akrolayer <https://twitter.com/akrolayer>

## 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