



Keyboard Layout Language Spec

0.3d

HaaTa - Jacob Alexander
haata@kiibohd.com
July 5, 2016

Revision History

Revision	Date	Author(s)	Description
0.1	2014-05-10	HaaTa	Initial Draft
0.2	2014-05-11	HaaTa	Finished trigger section
0.2a	2014-05-15	HaaTa	Finished result section
0.2b	2014-05-17	HaaTa	Initial pass at USB Code Table
0.2c	2014-05-18	HaaTa	Finished USB Code Table
0.3	2014-09-07	HaaTa	Initial KLL implementation
0.3a	2014-11-21	HaaTa	Adding Defines
0.3b	2015-05-02	HaaTa	Adding Media Keys and Null Key
0.3c	2015-09-29	HaaTa	Adding soft replacement
0.3d	2015-10-17	HaaTa	Adding more consumer control codes

Contents

1 Summary	1
2 Scope	1
3 Variables	2
3.1 Defines	2
4 Capabilities	3
5 Keymapping	4
5.1 Trigger	5
5.1.1 Scan Code	6
5.1.2 USB Code	6
5.1.3 Analog	7
5.1.4 Range	8
5.1.5 Sequence	9
5.1.6 Combination	10
5.2 Result	10
5.2.1 USB Code	10
5.2.2 None	11
5.2.3 Consumer Control Code	11
5.2.4 System Control Code	11

5.2.5 Sequence	12
5.2.6 Combination	12
5.2.7 Capability	12
6 File Format	13
6.1 Comments	13
6.2 Required Variables	14
6.3 Layer Control	14
6.4 Example Keymap	15
7 Layering	16
7.1 Scan Map	16
7.2 Combined Map	17
7.3 Partial Map	17
8 Compilation	17
9 USB Code Table	18
10 System Control Code Table	24
11 Consumer Control Code Table	25
12 Capabilities Table	34
13 Glossary	35
14 Future Considerations	35

1 Summary

This document outlines a complex keymapping scheme for custom keyboard firmware. It is based upon the concept of trigger:result pairs and ways to configure both parts of the pair (input and output). A variety of macros are supported along with analog keyswitches.

In addition, this keymapping scheme supports static and dynamic layering as well as supporting arbitrary keyboard features that may or may not be available on a given keyboard.

2 Scope

Hardware keymappings for keyboards traditionally has been cumbersome. At best, each layer can be defined as a set of predefined trigger:result pairs. A trigger being a key and a result being a key press (e.g. USB Code). Some keyboards allow for macros; however, these macros usually have to be custom programmed to the microcontroller (most, if not all custom keyboard firmware), or have very limited scope (Kinesis Advantage Pro and some Cherry POS keyboards such as the G80-8200).

Secondly, keymap layering (e.g. FN Layer) almost universally supports distinct layers (in the firmware that support this). However, no current firmware support "partial layering". Partial layering is only modifying a part of the current layout, similar to how XKB handles remapping Ctrl and CapsLock.

To reduce size constraints, EEPROM storage of keyboard layouts will not be considered. It may be possible to do post compilation keymap modifications, but this is considered optional for an implementation.

Finally, as of current (2015-05-10), no keymapping mechanism supports analog keyswitches for use in typing. This is mostly lack of availability of analog keyboard switches.

3 Variables

Variables serve two purposes. The first, to give information to the compiler both for informational and controlling internal keyboard firmware features. The second is for keymapping convenience. In most cases variables will only be used for compiler information.

Variables used for non-compiler purposes must be integers. The compiler will enforce this.

Variables are defined as follows.

```
<variable name> = <contents>;
```

Each command ends with a semi colon, there are no exceptions to this. Every keymap file will have the following variable.

```
Name = myKeymapFile;
```

If spaces are required for either the variable name or variable contents, doubles quotes may be used. The spaces will be internally converted to underscores if the variable is used for non-informational purposes. Variable names can only use A-Z, a-z, 0-9 and underscores and must not start with a number.

```
"space containing variable" = "space-ful variable";  
mix = "And Match";
```

3.1 Defines

Defines are a special case of variables that are used to influence static configuration options of the keyboard. This allows information to be given to the keyboard firmware before it is compiled. For example, defining the word size of the macro definition (i.e. using 8 bit instead of 32 bit when only 256 positions are required).

Defines are set the same way as variables are. However, and additional configuration is required to indicate which variables set defines for the compiled firmware. This help save the user from not setting critical defines that the firmware requires as warning messages are generated by the compiler when they are missing.

```
<capability name> => <corresponding C/C++ define>;
```

Each keyboard will likely have a different set of configurable defines which will be exposed. By default, they will have set values; however, these values can be overridden.

```
# Define declaration
myDefine => myCDefine;

# Default value
myDefine = 23;

# Override
myDefine = 144;
```

In order to pass strings, the desired quotes must be double-quoted.

```
myDefine => myCDefine;

# Correct
myDefine = '"This is a good string"';

# Possibly incorrect
myDefine = "This is an iffy string";
```

The second variable assignment will not pass the double quotes to the generated file so the result will not be interpreted as a C-String by the C/C++ compiler.

4 Capabilities

Capabilities define what the keyboard can do. At basic level, each keyboard has the capability to send USB Codes. Some keyboards have solenoids that can be fired, others have clickers. Capabilities are read by the compiler to cross-reference the functions that control that capability to the designated key press.

If a capability is specified as a result to a keypress and is not defined for the specified keyboard, it is ignored and removed from the compiled keyboard mappings (for each of the layers it is defined on).

In general, capabilities specific to each keyboard are defined in the keyboard specific Scan Code to USB Code keymap. Capabilities such as clickers and solenoids are defined using a standard so that their functionality

does not have to be redefined for each keymap (i.e. keymaps are transferable between keyboards even with special functionality that does not apply to every keyboard).

Capabilities are defined as follows.

```
<capability name> => <corresponding C/C++ function>;
```

The arguments of the C/C++ function must be specified, but not defined. Capabilities are used to describe to both the compiler and user, what the special functionality is supposed to do. The argument names themselves should be descriptive names containing no spaces, start with a letter and may use a-z, A-Z, 0-9 and underscore. The colon after each argument specifies the byte length of the argument. This number must be a positive integer.

Capabilities have two implied arguments, state and state type. These arguments are set by the TriggerMacro that signaled this Capability.

Correct

```
myCapability => myCFunction( arg1:1, arg2:2 );
```

Incorrect, defines the arg2 as 25

```
yourCapability => myCFunction( arg1:1, 25:2 );
```

Refer to Section 12 for a list of common capabilities.

5 Keymapping

Keymapping is the main purpose of KLL. KLL is designed to deal with most kinds of macros and layering (in addition to keyboard specific functions).

Keymapping is defined in two parts. The first part is called the trigger. The trigger defines what the signal of a keypress (e.g. press a) is that will be used to generate a result (e.g. send USB Code a). Triggers can be simple or very complex. Defined as Scan Codes (non-portable) and USB Codes (portable).

The second part is called a result. The result is what the keyboard firmware is designated to do once the corresponding trigger is received. Results vary from sending single USB Codes, to USB Code Macros, to enabling keyboard specific capabilities and toggling other keymapped layers. Defined as USB Codes and Capabilities. If an undefined Capability is used it

is ignored and the trigger:result pair is removed from the keymap.

As a note, time based key sequences (e.g. entering keys within a specified period of time) are not supported for both the trigger and result mechanisms. These may be added in a future version of KLL if enough demand (and a sensible/scalable implementation is proposed).

The following is the general syntax of the trigger:result pair.

```
<trigger> : <result>;
```

There are four variants of the trigger:result pair. Depending on which variant is used, assignment of the result changes.

```
<trigger> : <result>; # Replaces results on trigger
<trigger> :: <result>; # Soft layer replacement
<trigger> :+ <result>; # Adds results to trigger
<trigger> :- <result>; # Removes result from trigger
```

In general, the normal trigger:result pair will be used to replace/change keymapping. The trigger::result pair is a soft layer replacement which is used to only do replacement if two conditions are met: not the default layer and the trigger being replaced is not the same as defined in the base map. This is useful for making broad replacement rules for all layers but only come into effect if actually being used. An adding trigger:+result pair is useful when adding an extra action to a key of macro of keys (e.g. solenoid press). A subtraction trigger:-result pair is useful when compiling in many layouts and there are certain additions that are undesirable.

5.1 Trigger

The trigger defines the conditions required for a specific keymapping. These conditions can be as simple as a single key press or as complex as sequences of key combinations.

At a basic level, there are two types of triggers: Scan Codes and USB Codes. Scan Codes are native identifiers to the keyboard firmware. Whereas USB Codes are the identifiers used to send over USB to the OS. After compilation, all keymappings are mapped to Scan Codes; however, using Scan Codes in keymap files is not portable between different keyboards so their use is discouraged outside of defining the default Scan Code to USB Code keymap.

Beyond this, it is possible to specify ranges of keys to do the same function. Require a sequence of keys entered to enable a function. Use a

combination of pressed keys to trigger a capability.

Finally, it is possible to define percentage pressed values for triggering functions. This is useful for analog keyboard switches (not currently common).

Precedence is evaluated as follows: Scan Code/USB Code then range, then combination and finally sequence. Ranges of combinations does not make sense, nor does combinations of sequences.

5.1.1 Scan Code

Scan Codes are the native identifier for keypresses on keyboard firmware. In general, Scan Codes should not be used for defining keymaps; however, they are required for defining the initial Scan Code to USB Code mapping for each keyboard.

Each trigger must identify whether it is a Scan Code or USB Code. For Scan Codes it is prefixed with an S.

```
S<Scan Code> : <result>;
```

Scan Codes can be defined as either hex or decimal numbers.

```
S0x2A : <result>;
S124  : <result>;
```

The compiler will error if it is specified in the keymap explicitly.

5.1.2 USB Code

USB Codes define how USB understands keyboard output press/release events and are defined by the USB HID Spec. USB Codes are the recommended identifier when defining triggers for KLL keymaps.

Each trigger must identify whether it is a Scan Code or USB Code. For USB Codes it is prefixed with a U.

```
U<USB Code> : <result>;
```

USB Codes can be defined in a number of ways. Like Scan Codes, the USB Code can be defined directly using hex or decimal numbers (not recommended). The recommended way is to use the descriptive names for the USB Codes. Descriptive names are always strings, and must be enclosed with double quotes.

```
U0x2A : <result>;
U124  : <result>;
U"A"  : <result>;
U"a"  : <result>; # Same as previous
```

Refer to Section 9 for the complete list of USB Codes.

There is an important difference when using a USB Code as the the Trigger as compared to a Scan Code. When an assignment is done using a USB Code, a lookup must occur to find the original Scan Code that the USB Code was assigned to. Since assignment order matters this can become confusing.

```
U"s" : U"r";
U"r" : U"p";
```

This has an unintended side-effect. All R's will be assigned to P. If the user is aware, the problem isn't too bad to deal with. However, this is a very tricky problem to deal with for GUI keymap programs as it is not intuitive to enforce order.

Therefore, to resolve this issue, all assignments using a USB Code trigger must be cached until the current kll file has finished parsing. Then the cached assignments can be applied to the current layer. This will allow mass assignment of keys without having to worry about the order of assignment. If re-assignment is needed, using another kll file or Scan Code triggers can be used.

5.1.3 Analog

For analog keyboard switches, it is useful to specify a trigger at a press percentage as press/release behaviour needs to be defined. The expected implementation defines a press when the key stops moving or begins to move back up. If a stopped key moves further down (past the next trigger), stops or begins to move back up the next trigger is used and the previous on is released. Again, if the stopped key moves back up going back over a past trigger, stops (or with a slight delay before going back up further) this past trigger will activate, releasing the previous trigger.

Analog triggers are specified using parenthesis.

```
<Code>(<Analog Value>) : <result>;
```

Both USB Codes and Scan Codes can be used with analog keyboards

signals. Mapping analog triggers gets more tricky, so it is recommended that they are only used with USB Codes. Analog triggers are specified from 0 to 100 using integers only. 0 is a special case and is only pulsed when it is reached rather than held.

```
S0x2A(10) : <result>; # 10% press
S0x2A(80) : <result>; # 80% press
U124 (50) : <result>; # 50% press
U"A" (0) : <result>; # 0% pulse
U"a" (52) : <result>; # 52% press
```

If the keyboard is not analog, any trigger with an analog trigger will be ignored. And inversely, if the Scan Code/USB Code maps to an analog key, a normal press/release trigger will be ignored.

5.1.4 Range

For convenience, it is also possible to define a range of triggers rather than explicitly defining each of the trigger:result pairs. This is useful for defining things such as using the clicker speaker for every single key or just the letter keys.

Ranges are defined as the numerical range of either USB Codes or Scan Codes (use not recommended). Fortunately this means ranges such as A-Z will work as USB HID defines the USB Codes in order. Unfortunately the range 0-9 will not work as the USB Codes are organized: 1, 2, 3, 4, 5, 6, 7, 8, 9, 0 just like on a keyboard. To assist with this problem, the compiler will warn when this case occurred. Forward and reverse ranges are possible. There is no wrap-around. Scan Codes and USB Codes cannot be mixed.

To define a range, two additional pieces of syntax are required: square brackets and range specifier.

```
S[0x2A-0x50] : <result>;
S[12-43]      : <result>;
S[69]         : <result>; # Also a valid definition
U[124-43]     : <result>; # Reverse range
U["A"-"Z"]    : <result>;
U["0"-"9"]    : <result>; # Compiler will warn, only 0 and 9
U["1"-"0"]    : <result>; # Correct definition of 0-9
```

In addition to a specific range, multiple single keys or ranges may also be specified.

```
S[0x2A–0x50, 0x5] : <result>;
S[12, 43] : <result>;
U["A"–"Z", "Enter"] : <result>;
```

When using an analog supporting keyboard, a range is specified to that specific range. Or the analog trigger can be specified for the entire range.

```
S[0x2A–0x50(20), 0x5(56)] : <result>;
S[12, 43](79) : <result>;
U["A"–"Z"(42), "Tab"](30) : <result>; # 30, unless specified
```

5.1.5 Sequence

Sequences are a type of macro. A sequence is a set of expected inputs that must happen in order with no extra inputs of non-defined inputs. If a non-defined input is found, the sequence is reset and waits for the first input in the sequence again. Sequences cannot mix Scan Codes and USB Codes.

Trigger inputs in the sequence are separated by commas. The Scan Codes and USB Codes are evaluated/expanded first, then the sequences are evaluated (Codes have higher precedence).

```
S[0x2A–0x50], S[12], U[5] : <result>;
U["Q"–"Y"], U["Enter"] : <result>;
U["A"–"Z"(42), "Tab"](30), U[12](53) : <result>;
```

Analog triggers cannot be set across multiple inputs in the sequence.

For convenience, a second sequence syntax is available only for specifying ASCII characters. This allows for sentences to be inputted rather than the individual USB Codes. The key difference is the use of single quotes. If a shift is required for the specific key, it will be added as a combination (see Section 5.1.6).

```
'abc' : <result>;
U'T' : <result>;
U['Can you type this?'](23) : <result>;
```

Single quotes are only used for sentence expansion, they cannot be used for strings.

5.1.6 Combination

Combinations are a type of macro. A combination is a set of inputs pressed at the same time. For example: Ctrl+Alt+Delete. In order for a combination trigger to be signalled all of the keys must be pressed. The combination will not end if a key is missing from the combination (as the user may not have pressed it yet). Extra keys not part of the combination also do not cancel the combination. Combinations cannot mix Scan Codes and USB Codes.

Trigger inputs in the combination are separated by pluses. The Scan Codes and USB Codes are evaluated/expanded first, then sequences are evaluated and finally combinations (Codes have the highest precedence, then sequences, then combinations).

U["Q"–"Y"] + U["Enter"]	: <result>;
S[0x2A–0x50] + S[12], U[5]	: <result>;
U["A"–"Z"(42), "Tab"](30) + U[12](53)	: <result>;
'Can you type this?'(23) + 'a'(43)	: <result>;

Analog triggers cannot be set across multiple inputs in the combination.

5.2 Result

The result is the action, or set of actions, after the conditions for the trigger have been satisfied. The result can be as simple as a single USB Code output or as complex as signalling a keyboard specific clicker and outputting a combinational macro. A Scan Code cannot be used as a result.

Like triggers, there are two basic types of results: keyboard specific functionality and USB Codes. Both of these results are referred to as capabilities. A basic capability of every keyboard is to send USB Codes, while other keyboards may have other capabilities such as FN layers or clickers. When a trigger is satisfied each of the results assigned to this trigger are signalled.

Much of the syntax is the same as with triggers. The major difference being, no analog qualifiers and no ranges.

5.2.1 USB Code

USB Codes are the primary use of the result part of the trigger:result pairs. Result USB Codes are defined the same way as trigger USB Codes

(Section 5.1.2).

```
<trigger> : U0x2A;
<trigger> : U124;
<trigger> : U"A";
<trigger> : U"a"; # Same as previous
```

Refer to Section 9 for the complete list of USB Codes.

5.2.2 None

Sometimes it is useful to block the fall-through to a previous layer and have the key do nothing. The **None** keyword is case sensitive.

```
<trigger> : None;
```

5.2.3 Consumer Control Code

Consumer Control Codes, or Media Keys are a USB HID control type that allows you to control media functions. These include global hotkeys such as Play, Next and Previous.

Similar to USB Codes (Section 5.1.2), Consumer Control Codes can be defined as numbers or by its symbolic name. It is important to note that there is no support for sequences, combinations or ranges of Consumer Control Codes.

```
<trigger> : CON0xB0;
<trigger> : CON176;
<trigger> : CON"Play";
<trigger> : CON"play"; # Same as previous
```

Refer to Section 11 for the complete list of Consumer Control Codes.

5.2.4 System Control Code

System Control Codes are a USB HID control type that allows you to system level functions. These include global hotkeys such as Power, Sleep and Eject.

Similar to USB Codes (Section 5.1.2), System Control Codes can be defined as numbers or by its symbolic name. It is important to note that

there is no support for sequences, combinations or ranges of System Control Codes.

```
<trigger> : SYS0x82;
<trigger> : SYS130;
<trigger> : SYS"Sleep";
<trigger> : SYS"sleep"; # Same as previous
```

Refer to Section 10 for the complete list of System Control Codes.

5.2.5 Sequence

A result sequence is a series of outputted USB Codes which are split between each USB output buffer refresh. This is similar to a trigger input sequence. The syntax is the same as trigger sequences (Section 5.1.5).

```
<trigger> : U["Q"-"Y"], U["Enter"];
<trigger> : U["A"-"Z", "Tab"], U[12];
```

Single quoted strings will be literally evaluated. Any characters requiring a Shift key will have it added as a combination as part of the result.

```
<trigger> : 'abc';
<trigger> : U'T';
<trigger> : U['Can you type this?'];
```

5.2.6 Combination

A result combination is a set of keys sent out during the same USB output buffer. The syntax is the same as trigger combinations (Section 5.1.6). Keep in mind, if the keyboard does not support NKRO then some result combinations will not be possible.

```
<trigger> : U["Q"-"Y"] + U["Enter"];
<trigger> : U["A"-"Z", "Tab"] + U[12];
<trigger> : 'Can you type this?' + 'a';
```

5.2.7 Capability

In addition to outputting USB Codes, capabilities can be triggered. Technically, outputting USB Codes is a capability of the keyboard, but it is

always considered to exist. If a capability is specified as a result, but is not available for that keyboard, the trigger:result pair is ignored by the compiler and removed from the keymap.

To specify a capability as the result, define the capability name and any arguments required for the capability. To use the default parameter for an argument, set the argument as NULL.

```
# Defined capability
myCapability => myCFunction( arg1:1, arg2:1 );

# Using default first argument, and 25 for the second
<trigger> : myCapability( NULL, 25 );
```

To specify one of the arguments as the analog/press (press, hold, release) variable from the trigger, specify the argument as "output".

```
# Defined capability
myCapability => myCFunction( arg1:1, arg2:1 );

# output specifies the analog/press result
<trigger> : myCapability( NULL, output );
```

Refer to Section 12 for a list of common capabilities.

6 File Format

The Keyboard Layout Language or KLL uses a series of plain text, readable files to define configurations. This does not preclude GUI keymapping tools as it can generate KLL files first.

Each file represents a single keymapping layer and it is up the firmware configuration to define the precedence of each layer. KLL uses the concept of "Capabilities" to define potential results of a keypress (Section 4). At a basic level these are the keycodes (USB Codes) sent out to USB, but can be things like a clicker speaker or special LED signals.

6.1 Comments

End-of-line comments are defined using a '#'. For example,


```
# This is a comment
Not a comment # But this is a comment
```

defines the two supported styles of comments. Block and inline comments are not supported.

6.2 Required Variables

Each keymap file has a set of required variables that must be defined.

```
# Name of the keymap
Name = myKeymap; # Spaces will be replaced with _'s

# Version of the keymap
Version = 1.32a-HaaTa;

# Modified Date
Date = 2014-05-10;

# Author
# Multiple authors should be comma separated with the year
# range of their contributions
# Quotes can be used if spaces are required.
Author = "HaaTa (Jacob Alexander) 2014";

# KLL Version, used to detect future incompatibilities
KLL = 0.1;
```

The **Name** and **KLL** variables are used by the KLL compiler. Name prefixed to all keymap specific capabilities (e.g. myKeymap_Latch). KLL is for version checking to deal with future potential issues.

The rest of the variables are just informational. But are to be propagated to combined keymaps.

6.3 Layer Control

While up to the compiler how layers are controlled, these are some guidelines on how to control layers. Each layer must support three different activation types.

- Shift - Enables keymap while held

- Latch - Enables keymap until the next key is pressed
- Lock - Enables keymap until Lock is pressed again

6.4 Example Keymap

The following is a QWERTY/US ANSI to Colemak keymap. Only USB Codes are used as this is not a Scan Map.

```
Name = colemak;
Version = 0.1;
Author = "HaaTa (Jacob Alexander) 2014";
KLL = 0.2a;

# Modified Date
Date = 2014-05-17;

# Top Row
'e' : 'f';
'r' : 'p';
't' : 'g';
'y' : 'j';
'u' : 'l';
'i' : 'u';
'o' : 'y';
'p' : ',';

# Middle Row
's' : 'r';
'd' : 's';
'f' : 't';
'g' : 'd';
'j' : 'n';
'k' : 'e';
'l' : 'i';
',' : 'o';

# Bottom Row
'n' : 'k';
```

When remapping ASCII characters, using single quotes is the simplest way to map keys. Remember, all other keys must be remapped using either the USB Code or USB Code identifier.

```
Name = capslock2ctrl;  
Version = 0.1;  
Author = "HaaTa (Jacob Alexander) 2014";  
KLL = 0.2a;  
  
# Modified Date  
Date = 2014-05-17;  
  
U[ "CapsLock" ] : U[ "Ctrl" ];
```

7 Layering

While the KLL files have little to do with the keymap layering implementation, there are a few features that do help with the implementation. In general, keymaps are defined by USB codes and not Scan Codes as Scan Codes are keyboard specific. However, for each keyboard an initial mapping must be defined from Scan Codes to USB Codes.

After compilation, each keymap is reduced down to Scan Codes as they are the unambiguous internal key indications that keyboard firmwares understand.

7.1 Scan Map

The Scan Map is the keymapping used to define each type of keyboard. This keymapping must use Scan Codes to USB Codes, otherwise later compilation process will fail (as it won't be possible to resolve the Scan Codes).

If possible, each Scan Map should define a US ANSI-like keymap. Any keys that are not defined by the general US ANSI standard should be defined to another reasonable USB Code.

The scan map should also define any capabilities that are provided by the keyboard. For example, a clicker speaker.

```
# Clicker speaker with "sound select" and volume args  
myControlableClicker => clickerCFunction( sound:2, volume:1 );  
  
# Use the capability when Scan Code 0x2C is pressed
```

```
# NULL specifies the default option of the capability
```

```
S[0x2C] : myControlableClicker( NULL, 25 );
```

When defining capabilities like solenoids and clickers, follow the naming convention and same number of arguments. If there are extra capabilities for this keyboard (e.g. click volume), add an additional capability to control this functionality.

Refer to Section 12 for a list of common capabilities.

7.2 Combined Map

The Combined Map is a set of keymaps that have been folded (or compiled) down to a single keymap. For example, if there is the default Scan Map and a Colemak keymap is specified, both of these keymaps will be compiled down to Scan Code triggers and the USB code results (as well as any other specified capabilities). For keys that were not specified in the secondary maps (the highest layered map takes precedence) it will use the lower layer keymapping to be included into the Combined Map.

7.3 Partial Map

A Partial Map is a keymap that may not define every possible Scan Code. Some uses for Partial Maps are to swap Caps Lock and Control, or define Colemak, where not all of the keys change from QWERTY/US ANSI. Partial Maps can be used to generate a Combined Map or be compiled individually to be layered on the keyboard firmware dynamically for FN layers.

8 Compilation

Compilation is relatively simple. Reduce all triggers down to Scan Codes and all results into capabilities and USB Codes. To combine keymaps, they must be stacked during compilation. The first layout specified is the default. Each layout layered on top takes precedence over the layout underneath of it.

There is no requirement to support online key remapping, EEPROM stor-

age or post compilation key remapping (pre microcontroller flashing).

Keys that are not mapped should be ignored and not present. Capabilities specified, but do not exist for a given keyboard are also ignored.

Warnings should be given about ambiguous ranges or no keymappings/trigger:result pairs are mapped.

Compilation should not succeed if there are missing mandatory variables or syntax problems. Additionally, there cannot be any duplicate capabilities defined, whether they are ignored or not.

9 USB Code Table

Following is a table of USB Codes. For each USB Code there is at least a long and short name. The names are case-insensitive. Some USB Codes have multiple names for convenience. Spaces and underscores both translate to underscore internally.

Please refer to the USB Keyboard HID spec for more details. All event based key codes that do not correspond to a physical key are not included in the table. For example, USB Code 0x01, which is a rollover error indicator.

Many USB Codes are recognized/implemented on the OS side. Do not expect lesser known USB Codes to "just work".

Table 9.1: Table of USB Codes

USB Code	Name	Alternate Name(s)
0x04 (4)	A	
0x05 (5)	B	
0x06 (6)	C	
0x07 (7)	D	
0x08 (8)	E	
0x09 (9)	F	
0x0A (10)	G	
0x0B (11)	H	
0x0C (12)	I	
0x0D (13)	J	
0x0E (14)	K	
0x0F (15)	L	

0x10 (16)	M
0x11 (17)	N
0x12 (18)	O
0x13 (19)	P
0x14 (20)	Q
0x15 (21)	R
0x16 (22)	S
0x17 (23)	T
0x18 (24)	U
0x19 (25)	V
0x1A (26)	W
0x1B (27)	X
0x1C (28)	Y
0x1D (29)	Z
0x1E (30)	1
0x1F (31)	2
0x20 (32)	3
0x21 (33)	4
0x22 (34)	5
0x23 (35)	6
0x24 (36)	7
0x25 (37)	8
0x26 (38)	9
0x27 (39)	0
0x28 (40)	Enter
0x29 (41)	Esc
0x2A (42)	Backspace
0x2B (43)	Tab
0x2C (44)	Space
0x2D (45)	- Minus
0x2E (46)	= Equals, Equal
0x2F (47)	[Left Bracket, LBracket
0x30 (48)] Right Bracket, RBracket
0x31 (49)	\ Backslash
0x32 (50)	# Number, Hash
0x33 (51)	; Semicolon
0x34 (52)	' Quote
0x35 (53)	` Backtick
0x36 (54)	, Comma
0x37 (55)	. Period
0x38 (56)	/ Slash
0x39 (57)	CapsLock

0x3A (58)	F1	
0x3B (59)	F2	
0x3C (60)	F3	
0x3D (61)	F4	
0x3E (62)	F5	
0x3F (63)	F6	
0x40 (64)	F7	
0x41 (65)	F8	
0x42 (66)	F9	
0x43 (67)	F10	
0x44 (68)	F11	
0x45 (69)	F12	
0x46 (70)	PrintScreen	
0x47 (71)	ScrollLock	
0x48 (72)	Pause	
0x49 (73)	Insert	
0x4A (74)	Home	
0x4B (75)	PageUp	
0x4C (76)	Delete	
0x4D (77)	End	
0x4E (78)	PageDown	
0x4F (79)	Right	
0x50 (80)	Left	
0x51 (81)	Down	
0x52 (82)	Up	
0x53 (83)	NumLock	
0x54 (84)	P/	Keypad Slash
0x55 (85)	P*	Keypad Asterix
0x56 (86)	P-	Keypad Minus
0x57 (87)	P+	Keypad Plus
0x58 (88)	PEnter	Keypad Enter
0x59 (89)	P1	Keypad 1
0x5A (90)	P2	Keypad 2
0x5B (91)	P3	Keypad 3
0x5C (92)	P4	Keypad 4
0x5D (93)	P5	Keypad 5
0x5E (94)	P6	Keypad 6
0x5F (95)	P7	Keypad 7
0x60 (96)	P8	Keypad 8
0x61 (97)	P9	Keypad 9
0x62 (98)	P0	Keypad 0
0x63 (99)	P.	Keypad Decimal

0x64 (100)	ISO/	ISO Slash
0x65 (101)	App	
0x67 (103)	P=	Keypad Equals
0x68 (104)	F13	
0x69 (105)	F14	
0x6A (106)	F15	
0x6B (107)	F16	
0x6C (108)	F17	
0x6D (109)	F18	
0x6E (110)	F19	
0x6F (111)	F20	
0x70 (112)	F21	
0x71 (113)	F22	
0x72 (114)	F23	
0x73 (115)	F24	
0x74 (116)	Exec	
0x75 (117)	Help	
0x76 (118)	Menu	
0x77 (119)	Select	
0x78 (120)	Stop	
0x79 (121)	Again	
0x7A (122)	Undo	
0x7B (123)	Cut	
0x7C (124)	Copy	
0x7D (125)	Paste	
0x7E (126)	Find	
0x7F (127)	Mute	
0x80 (128)	VolumeUp	
0x81 (129)	VolumeDown	
0x82 (130)	CapsToggleLock	
0x83 (131)	NumToggleLock	
0x84 (132)	ScrollToggleLock	
0x85 (133)	P,	Keypad Comma
0x86 (134)	Keypad AS400 Equals	
0x87 (135)	Inter1	Kanji1
0x88 (136)	Inter2	Kanji2, Kana
0x89 (137)	Inter3	Kanji3, Yen
0x8A (138)	Inter4	Kanji4, Henkan
0x8B (139)	Inter5	Kanji5, Muhenkan
0x8C (140)	Inter6	Kanji6
0x8D (141)	Inter7	Kanji7, ByteToggle
0x8E (142)	Inter8	Kanji8

0x8F (143)	Inter9	Kanji9
0x90 (144)	Lang1	HangulEnglish
0x91 (145)	Lang2	Hanja, Eisu
0x92 (146)	Lang3	Katakana
0x93 (147)	Lang4	Hiragana
0x94 (148)	Lang5	ZenkakuHankaku
0x95 (149)	Lang6	
0x96 (150)	Lang7	
0x97 (151)	Lang8	
0x98 (152)	Lang9	
0x99 (153)	AltErase	
0x9A (154)	SysReq	
0x9B (155)	Cancel	
0x9C (156)	Clear	
0x9D (157)	Prior	
0x9E (158)	Return	
0x9F (159)	Sep	Separator
0xA0 (160)	Out	
0xA1 (161)	Oper	
0xA2 (162)	Clear	
0xA3 (163)	CrSel	
0xA4 (164)	ExSel	
0xB0 (176)	P00	Keypad 00
0xB1 (177)	P000	Keypad 000
0xB2 (178)	1000Sep	ThousandSeparator
0xB3 (179)	DecimalSep	DecimalSeparator
0xB4 (180)	Currency	CurrencyUnit
0xB5 (181)	CurrencySub	CurrencySubUnit
0xB6 (182)	P(Keypad Left Parentheses
0xB7 (183)	P)	Keypad Right Parentheses
0xB8 (184)	P{	Keypad Left Brace
0xB9 (185)	P}	Keypad Right Brace
0xBA (186)	PTab	Keypad Tab
0xBB (187)	PBackspace	Keypad Backspace
0xBC (188)	PA	Keypad A
0xBD (189)	PB	Keypad B
0xBE (190)	PC	Keypad C
0xBF (191)	PD	Keypad D
0xC0 (192)	PE	Keypad E
0xC1 (193)	PF	Keypad F
0xC2 (194)	PXOR	Keypad XOR
0xC3 (195)	P^	Keypad Chevron

0xC4 (196)	P%	Keypad Percent
0xC5 (197)	P<	Keypad LessThan
0xC6 (198)	P>	Keypad GreaterThan
0xC7 (199)	P&	Keypad BitAnd
0xC8 (200)	P&&	Keypad And
0xC9 (201)	P	Keypad BitOr
0xCA (202)	P	Keypad Or
0xCB (203)	P:	Keypad Colon
0xCC (204)	P#	Keypad Number, Keypad Hash
0xCD (205)	PSpace	Keypad Space
0xCE (206)	P@	Keypad At
0xCF (207)	P!	Keypad Exclaim
0xD0 (208)	PMemStore	Keypad MemStore
0xD1 (209)	PMemRecall	Keypad MemRecall
0xD2 (210)	PMemClear	Keypad MemClear
0xD3 (211)	PMemAdd	Keypad MemAdd
0xD4 (212)	PMemSub	Keypad MemSub
0xD5 (213)	PMemMult	Keypad MemMult
0xD6 (214)	PMemDiv	Keypad MemDiv
0xD7 (215)	P+/-	Keypad PlusMinus
0xD8 (216)	PClear	Keypad Clear
0xD9 (217)	PClearEntry	Keypad ClearEntry
0xDA (218)	PBinary	Keypad Binary
0xDB (219)	POctal	Keypad Octal
0xDC (220)	PDecimal	Keypad Decimal
0xDD (221)	PHex	Keypad Hex
0xE0 (224)	LCtrl	Left Ctrl, Ctrl
0xE1 (225)	LShift	Left Shift, Shift
0xE2 (226)	LAlt	Left Alt, Alt
0xE3 (227)	LGui	Left Gui, Gui
0xE4 (228)	RCtrl	Right Ctrl
0xE5 (229)	RShift	Right Shift
0xE6 (230)	RAlt	Right Alt
0xE7 (231)	RGui	Right Gui
0xF0 (240)	Function1	Fun1, Fun
0xF1 (241)	Function2	Fun2
0xF2 (242)	Function3	Fun3
0xF3 (243)	Function4	Fun4
0xF4 (244)	Function5	Fun5
0xF5 (245)	Function6	Fun6
0xF6 (246)	Function7	Fun7
0xF7 (247)	Function8	Fun8

0xF8 (248)	Function9	Fun9
0xF9 (249)	Function10	Fun10
0xFA (250)	Function11	Fun11
0xFB (251)	Function12	Fun12
0xFC (252)	Function13	Fun13
0xFD (253)	Function14	Fun14
0xFE (254)	Function15	Fun15
0xFF (255)	Function16	Fun16

As function keys are not part of the USB HID spec, they are difficult to reference using USB Codes. To get around this, KLL reserves USB Codes 0xF0 to 0xFF for function keys. It is possible (though unlikely) that a new USB HID spec may use this reserved space. In that case, a new revision of KLL will be made to address this issue. USB Codes 0xF0 to 0xFF are never sent via USB and are for internal processing only.

It is possible to use any arbitrary USB Code as a function key. However, using something like a SysReq as your function key (in the default map) makes it unclear to other readers unless you explicitly comment this case in the .kll file.

10 System Control Code Table

Following is a table of USB System Control Codes. These are useful for defining keys to signal the OS to power off, sleep or eject something.

Please refer to the USB HID spec for more details. Do not expect your OS to recognize these codes. Many of them are system specific.

Table 10.1: Table of System Control Codes

Sys Code	Name	Alternate Name(s)
0x81	PowerDown	
0x82	Sleep	
0x83	WakeUp	
0x84	ContextMenu	
0x85	MainMenu	
0x86	AppMenu	

0x87	MenuHelp
0x88	MenuExit
0x89	MenuSelect
0x8A	MenuRight
0x8B	MenuLeft
0x8C	MenuUp
0x8D	MenuDown
0x8E	ColdRestart
0x8F	WarmRestart
0x90	DpadUp
0x91	DpadDown
0x92	DpadRight
0x93	DpadLeft
0xA0	Dock
0xA1	Undock
0xA2	Setup
0xA3	Break
0xA4	DebuggerBreak
0xA5	AppBreak
0xA6	AppDebuggerBreak
0xA7	SpeakerMute
0xA8	Hibernate
0xB0	DisplInvert
0xB1	DisplInternal
0xB2	DispExternal
0xB3	DispBoth
0xB4	DispDual
0xB5	DispToggleIntExt
0xB6	DispSwapPriSec
0xB7	DispLcdAutoscale

11 Consumer Control Code Table

Following is a table of USB Consumer Control Codes. These are useful for defining media keys such as Next, Previous, Pause, etc.

Please refer to the USB HID spec for more details. Do not expect your OS

to recognize these codes. Many of them are system specific.

Table 11.1: Table of Consumer Control Codes

Cons Code	Name	Alternate Name(s)
0x020	10	
0x021	100	
0x022	AmPm	
0x030	Power	
0x031	Reset	
0x032	Sleep	
0x033	SleepAfter	
0x034	SleepMode	
0x035	Illumination	
0x040	Menu	
0x041	MenuPick	
0x042	MenuUp	
0x043	MenuDown	
0x044	MenuLeft	
0x045	MenuRight	
0x046	MenuEscape	
0x047	MenuValueIncrease	
0x048	MenuValueDecrease	
0x060	DataOnScreen	
0x061	ClosedCaption	
0x062	ClosedCaptionSelect	
0x063	VcrTv	
0x064	BroadcastMode	
0x065	Snapshot	
0x066	Still	
0x06F	BrightnessIncrement	
0x070	BrightnessDecrement	
0x072	BacklightToggle	
0x073	BrightnessMin	
0x074	BrightnessMax	
0x075	BrightnessAuto	
0x081	AssignSelection	
0x082	ModeStep	
0x083	RecallLast	
0x084	EnterChannel	
0x085	OrderMovie	
0x088	MediaComputer	

0x089	MediaTv	
0x08A	MediaWww	
0x08B	MediaDvd	
0x08C	MediaTelephone	
0x08D	MediaProgramGuide	
0x08E	MediaVideoPhone	
0x08F	MediaSelectGames	
0x090	MediaSelectMessages	
0x091	MediaSelectCd	
0x092	MediaSelectVcr	
0x093	MediaSelectTuner	
0x094	Quit	
0x095	Help	
0x096	MediaSelectTape	
0x097	MediaSelectCable	
0x098	MediaSelectSatellite	
0x099	MediaSelectSecurity	
0x09A	MediaSelectHome	
0x09B	MediaSelectCall	
0x09C	ChannelIncrement	
0x09D	CahnnelDecrement	
0x09E	MediaSelectSap	
0x0A0	VcrPlus	
0x0A1	Once	
0x0A2	Daily	
0x0A3	Weekly	
0x0A4	Monthly	
0x0B0	Play	
0x0B1	Pause	
0x0B2	Record	
0x0B3	FastForward	
0x0B4	Rewind	
0x0B5	ScanNextTrack	
0x0B6	ScanPreviousTrack	
0x0B7	Stop	
0x0B8	Eject	
0x0B9	RandomPlay	
0x0BC	Repeat	
0x0BE	TrackNormal	
0x0C0	FrameForward	
0x0C1	FrameBack	
0x0C2	Mark	

0x0C3	ClearMark	
0x0C4	RepeatFromMark	
0x0C5	ReturnToMark	
0x0C6	SearchMarkForwards	
0x0C7	SearchMarkBackwards	
0x0C8	CounterReset	
0x0C9	ShowCounter	
0x0CA	TrackingIncrement	
0x0CB	TrackingDecrement	
0x0CC	StopEject	
0x0CD	PausePlay	
0x0CE	PlaySkip	
0x0E2	Mute	
0x0E5	BassBoost	
0x0E6	SurroundMode	
0x0E7	Loudness	
0x0E8	Mpx	
0x0E9	VolumeUp	
0x0EA	VolumeDown	
0x0F0	SpeedSelect	
0x0F2	StandardPlay	
0x0F3	LongPlay	
0x0F4	ExtendedPlay	
0x0F5	Slow	
0x100	FanEnable	
0x102	LightEnable	
0x104	ClimateControlEnable	
0x106	SecurityEnable	
0x107	FireAlarm	
0x10A	Motion	
0x10B	DuressAlarm	
0x10C	HoldupAlarm	
0x10D	MedicalAlarm	
0x150	BalanceRight	
0x151	BalanceLeft	
0x152	BassIncr	
0x153	BassDecr	
0x154	TrebleIncr	
0x155	TrebleDecr	
0x171	SubChannelIncrement	
0x172	SubChannelDecrement	
0x173	AltAudioIncrement	

0x174	AltAudioDecrement
0x181	LaunchButtonConfigTool
0x182	ProgrammableButtonConfig
0x183	ConsumerControlConfig
0x184	WordProcessor
0x185	TextEditor
0x186	Spreadsheet
0x187	GraphicsEditor
0x188	PresentationApp
0x189	DatabaseApp
0x18A	EmailReader
0x18B	Newsreader
0x18C	Voicemail
0x18D	ContactsAddressBook
0x18E	CalendarSchedule
0x18F	TaskProjectManager
0x190	LogJournalTimecard
0x191	CheckbookFinance
0x192	Calculator
0x193	aVCapturePlayback
0x194	LocalMachineBrowser
0x195	LanWanBrowser
0x196	InternetBrowser
0x197	RemoteNetworkingIspConnect
0x198	NetworkConference
0x199	NetworkChat
0x19A	TelephonyDialer
0x19B	Logon
0x19C	Logoff
0x19D	LogonLogoff
0x19E	TerminalLockScreensaver
0x19F	ControlPanel
0x1A0	CommandLineProcessorRun
0x1A1	ProcessTaskManager
0x1A2	SelectTastApp
0x1A3	NextTaskApp
0x1A4	PreviousTaskApp
0x1A5	PreemptiveHaltTaskApp
0x1A6	IntegratedHelpCenter
0x1A7	Documents
0x1A8	Thesaurus
0x1A9	Dictionary

0x1AA	Desktop
0x1AB	SpellCheck
0x1AC	GrammarCheck
0x1AD	WirelessStatus
0x1AE	KeyboardLayout
0x1AF	VirusProtection
0x1B0	Encryption
0x1B1	ScreenSaver
0x1B2	Alarms
0x1B3	Clock
0x1B4	FileBrowser
0x1B5	PowerStatus
0x1B6	ImageBrowser
0x1B7	AudioBrowser
0x1B8	MovieBrowser
0x1B9	DigitalRightsManager
0x1BA	DigitalWallet
0x1BC	InstantMessaging
0x1BD	OemFeaturesTipsTutorial
0x1BE	OemHelp
0x1BF	OnlineCommunity
0x1C0	EntertainmentContent
0x1C1	OnlineShopping
0x1C2	SmartcardInfoHelp
0x1C3	MarketMonitor
0x1C4	CustomizedCorpNews
0x1C5	OnlineActivity
0x1C6	SearchBrowser
0x1C7	AudioPlayer
0x201	New
0x202	Open
0x203	Close
0x204	Exit
0x205	Maximize
0x206	Minimize
0x207	Save
0x208	Print
0x209	Properties
0x21A	Undo
0x21B	Copy
0x21C	Cut
0x21D	Paste

0x21E	SelectAll
0x21F	Find
0x220	FindAndReplace
0x221	Search
0x222	GoTo
0x223	Home
0x224	Back
0x225	Forward
0x226	Stop
0x227	Refresh
0x228	PreviousLink
0x229	NextLink
0x22A	Bookmarks
0x22B	History
0x22C	Subscriptions
0x22D	ZoomIn
0x22E	ZoomOut
0x22F	Zoom
0x230	FullScreenView
0x231	NormalView
0x232	ViewToggle
0x233	ScrollUp
0x234	ScrollDown
0x235	Scroll
0x236	PanLeft
0x237	PanRight
0x238	Pan
0x239	NewWindow
0x23A	TileHorizontally
0x23B	TileVertically
0x23C	Format
0x23D	Edit
0x23E	Bold
0x23F	Italics
0x240	Underline
0x241	Strikethrough
0x242	Subscript
0x243	Superscript
0x244	AllCaps
0x245	Rotate
0x246	Resize
0x247	FilpHorizontal

0x248	FilpVertical
0x249	MirrorHorizontal
0x24A	MirrorVertical
0x24B	FontSelect
0x24C	FontColor
0x24D	FontSize
0x24E	JustifyLeft
0x24F	JustifyCenterH
0x250	JustifyRight
0x251	JustifyBlockH
0x252	JustifyTop
0x253	JustifyCenterV
0x254	JustifyBottom
0x255	JustifyBlockV
0x256	IndentDecrease
0x257	IndentIncrease
0x258	NumberedList
0x259	RestartNumbering
0x25A	BulletedList
0x25B	Promote
0x25C	Demote
0x25D	Yes
0x25E	No
0x25F	Cancel
0x260	Catalog
0x261	BuyCheckout
0x262	AddToCart
0x263	Expand
0x264	ExpandAll
0x265	Collapse
0x266	CollapseAll
0x267	PrintPreview
0x268	PasteSpecial
0x269	InsertMode
0x26A	Delete
0x26B	Lock
0x26C	Unlock
0x26D	Protect
0x26E	Unprotect
0x26F	AttachComment
0x270	DeleteComment
0x271	ViewComment

0x272	SelectWord
0x273	SelectSentence
0x274	SelectParagraph
0x275	SelectColumn
0x276	SelectRow
0x277	SelectTable
0x278	SelectObject
0x279	RedoRepeat
0x27A	Sort
0x27B	SortAscending
0x27C	SortDescending
0x27D	Filter
0x27E	SetClock
0x27F	ViewClock
0x280	SelectTimeZone
0x281	EditTimeZone
0x282	SetAlarm
0x283	ClearAlarm
0x284	SnoozeAlarm
0x285	ResetAlarm
0x286	Synchronize
0x287	SendReceive
0x288	SendTo
0x289	Reply
0x28A	ReplyAll
0x28B	ForwardMsg
0x28C	Send
0x28D	AttachFile
0x28E	Upload
0x28F	Download
0x290	SetBorders
0x291	InsertRow
0x292	InsertColumn
0x293	InsertFile
0x294	InsertPicture
0x295	InsertObject
0x296	InsertSymbol
0x297	SaveAndClose
0x298	Rename
0x299	Merge
0x29A	Split
0x29B	DistributeHorizontally

0x29C	DistributeVertically
0x29D	NextKeyboardLayoutSelect

12 Capabilities Table

Following is a table of common capabilities. Whenever defining new capabilities, make sure to check this table to make sure there are no namespace clashes. The number beside each of the arguments specifies the size of the argument in bytes (e.g. 2 \Rightarrow 16 bits).

Table 12.1: Table of Common Capabilities

Capability	Function	Argument(s)
consCtrlOut	Output_consCtrlSend_capability	consCode : 2
kbdProtocolBoot	Output_kbdProtocolBoot_capability	
kbdProtocolNKRO	Output_kbdProtocolNKRO_capability	
layerLatch	Macro_layerLatch_capability	layer : 2
layerLock	Macro_layerLock_capability	layer : 2
layerShift	Macro_layerShift_capability	layer : 2
layerState	Macro_layerState_capability	layer : 2, state : 1
noneOut	Output_noneSend_capability	
sysCtrlOut	Output_sysCtrlSend_capability	sysCode : 1
usbKeyOut	Output_usbCodeSend_capability	usbCode : 1

13 Glossary

Fall-through When a key is undefined on a particular layer, the key definition on the previously stacked layer will be used. Eventually the key definition will be set to using the default layer. If the None keyword is used, then the fall-through will stop and no action will take place.

Latch When referring to keyboards, a key function that is only enabled until the release of the next keypress.

Lock When referring to keyboards, a key function that is enabled until that key is pressed again (e.g. Caps Lock).

NKRO N-Key Rollover is the capability to press N number of keys at the same time on a keyboard and have them all register on the OS simultaneously.

Scan Code Row x Column code or native protocol code used by the keyboard.

Shift When referring to keyboards, a key function that is enabled while that key is held.

USB Code Keyboard Press/Release codes as defined by the USB HID Spec.

14 Future Considerations

- Sequence timing specifier syntax and function
- Alternate analog trigger mechanisms
- More concise way to indicate capability defaults
- Press/release key, hold key internally until signal
- Add field to change USB country code
- Use USB LEDs as a input trigger condition
- Analog velocity and acceleration detection (optional, depends on hardware)