Keyboard Ideas and Samples

Remember Your Users and know your Keyboarding Context.

Before starting a keyboard, pay attention to the existing keyboards in the area. Do they use dead keys, certain modifer keys, rotas, or something else. How do they compose complex characters? Does the Caps Key affect the number row? What other keyboards output a similar orthography? Don't blindly copy the national language layout and stuff in your characters, but you may want to adopt similar features to make transition to your keyboard easier.

Secondly, discuss with users how they would like to use their keyboard, and what modifications need to be made to implement their orthography.

Choosing to create a multilingual or national keyboard should not be done lightly. This can be much more work, and the extra characters may confuse the users. When possible, target simple monolingual keyboards.

Choosing Natural Input Methods for Physical Keyboards

Modifier Keys

The most common modifier key is the Shift key, which on most keyboards will allow access to capital letters. Keyman also allows you to use other modifier keys (ALT, RALT, CTRL etc.) and combinations (SHIFT-RALT, etc.) to allow even more combinations.
For infrequently used keys (such as typesetting marks), they may be hidden away behind the RALT key and still accessible. Note that some keyboard combinations are reserved by Keyman or other applications, so be sure to test your Modifier Keys in different programs and operating systems.

□□□□□□□□

> + [CAPS K_M] > U+002C + [CAPS RALT K_M] > U+00D7 + [CAPS SHIFT K_M] > U+003F + [CAPS SHIFT RALT K_M] > beep + [NCAPS K_M] > U+002c + [NCAPS RALT K_M] > beep + [NCAPS SHIFT K_M] > U+003F + [NCAPS SHIFT RALT K_M] > U+00D7

Note: The names that are used for keys can be found here: Virtual Keys.

Any combination of modifier keys that does not exist in your code will fall back to a base keyboard, so it may be helpful to define all combinations in your keyboard to recieve the expected output. One challenging combination is CAPS+SHIFT+K_9 or any other number, which on some keyboards (US English) should produce the shifted character and on others (French France) should produce the unshifted character. (CAPS does not affect the number row on an English keyboard, but it does on a French keyboard.)

Dead Keys:

Dead-keys are a time-tested method for accessing special characters. (Keyboarding systems that could not modify already-typed characters *had* to use deadkeys, so many users are accustomed to this. In languages where the dead-key tradition is not already well-established, it may be better to provide a visual change for every keystroke.) Dead-keys are usually typed before a letter key to access a special or modified output. If the semicolon key ";" is used as a deadkey, you might type ";n" and receive "ŋ". The US international and IPA keyboards rely heavily on dead keys that must be memorized.

SIL Philippines Keyboard (Uses Deadkeys)

Dead Key Documentation

Dead key combinations can be defined and used one at a time:

```
```

group(main) using keys + [K_SLASH] > dk(0021) ... match > use(deadkeys) group(deadkeys) dk(0021) e > ə

Or through stores:

```
```

store(plainvowels) 'a' 'e' 'i' 'o' 'u' 'A' 'E' 'I' 'O' 'U' store(acutevowels) U+00E1 U+00E9 U+00ED U+00F3 U+00FA U+00C1 U+00C9 U+00CD U+00D3 U+00DA + "'" > dk(apostrophe) c U+0027 APOSTROPHE dk(apostrophe) + any(plainvowels) > index(acutevowels, 2)

Note: Since dead keys give no visual feedback until the second key is pressed, and there is no easy way to alert the user which following keys will produce vailid output, they are probably not appropriate for touch keyboards unless the combinations are VERY well known.

Combinations:

It is possible to use special key combinations to access special characters. One might type "ng" and receive "ŋ". For orthographies with tens or thousands of characters (such as Traditional Chinese), this is probably the most efficent method. For example, in the Pinyin input method, the user types "xiatian" (a romanized phonetic form of the word) and receives "夏天" in Chinese characters.

Combinations can be defined and used one at a time:

```
```

"t" + "h" > "θ" "T" + "h" > "Θ"

Or through stores:

```
```

store(vowel) "aeiouAEIOU" store(vMid) "āēīōūĀĒĪŌŪ" ... group(main) using keys any(vowel) + "-" > index(vMid,2)

Rota Groups

A rota is a group of characters that can be cycled through with repeated presses of a key. (This key may or may not be the same key that created the initial character.) While this is a well-known system of accessing special characters in some areas, it is probably more appropriate for physical keyboards rather than touch keyboards (without some custom programming).

Xinaliq Keyboard (Uses Rotas)

In the following example, a "post-modifier" function key (BKQUOTE) changes the previous character to rotate between different options in each rota group. In this example, each base character has 1, 2 or 3 dotted forms in addition. (On a touch layout, providing a function button for this would typically only be a secondary, optional method that is provided in addition to the long press method. Providing a meaningful icon for that function key may require modifying the font used to display touch keys.)

```
```

c Use BKQUOTE Key To Add A Dot, Or Cycle Between Different Dotted Forms store (predot) ';039#&ﺗ;039#& ;039#&ﻭ;039#& ;039#&ﺷ;039#& ;039#&ﺴ;039#& ;039#&ﺳ;039#& ;039#&ﺴ;039#& ;039#&ﺭ;039#& ;039#&ﺧ;039#& ;039#&ﺮ;039#& ;039#&ﻗ;039#& ;039#&ﻫ;039#& ;039#&ﻋ;039#& ;039#&ﻋ;039#& ;039#&ﺳ;039#& ;039#&ﺴ;039#& ;039#&ﻧ;039#& ;039#&ﻣ;039#& ;039#&ﻣ;039#& ;039#&ﻟ;039#& 039#&ﺳ;039#& store (postdot) ';039#&ﺳ;039#& ;039#&ﺷ;039#& ;039#&ﺴ;039#& ;039#&ﺴ;039#& ;039#&ﺧ;039#& ;039#&ﻣ;039#& ;039#&ﻗ;039#& ;039#&ﻫ;039#& ;039#&ﻋ;039#& ;039#&ﺩ;039#& ;039#&ﺗ;039#& ;039#&ﺳ;039#& ;039#&ﻣ;039#& ;039#&ﻧ;039#& ;039#&ﻣ;039#& ;039#&ﻟ;039#& ;039#&ﻟ;039#& ;039#&ﻫ;039#& 039#&ﺳ;039#& ;039#&ﺳ;039#& any(predot) + [BKQUOTE] > index(postdot,1)

Rota groups that are based on the same key that created the initial character are best implemented with simple replacement rules. Because the Keyman language does not support timing distinctions (e.g. distinguishing between two single taps and one double tap), this kind of rota is best employed only where it would be extremely rare for the user to actually want two of any of these characters. For example, to use the COLON key to cycle through colon (:), visarga (ः), and avagraha (ऽ), you can do this:

□□□□□□□□

c Use COLON Key To Cycle Through COLON, VISARGA, And AVAGRAHA Characters ':' + ':' > 'ः' 'ः' + ':' > 'ऽ' 'ऽ' + ':' > '::'

Note: With a "manual" rota like this, if you want it to cycle back to the beginning (instead of creating a doubled character sequence, as in the above example) make the first input the same as the last output.

If doubling of the character is possible in the orthography, but the number of combinations is fairly limited, you could add a doubled combination to the rota's cycle.

□□□□□□□□

'T' + 'T' > 'Ṫ' C DOT ABOVE 'Ṫ' + 'T' > 'T̂' C CIRCUMFLEX 'T̂' + 'T' > 'T̂T̂' C DOUBLED CIRCUMFLEX 'T̂T̂' + 'T' > 'TT' C DOUBLED T 'TT' + 'T' > 'T' C REVERTS TO BASE CHARACTER 'T'

Choosing Natural Input Methods For Touch Keyboards

One may be tempted to recreate the complete physical keyboard on the touch keyboard, but this is rarely ideal. Ten-finger typing on a physical keyboard is a very different experience than typing on a touch-screen. While specific key combinations on physical keyboards typically needed to be memorized, touch keyboards open up new possibilities for showing the user what the output will be.

Creating a Simple Roman Touch Keyboard

Creating a Non-Roman Touch Keyboard

Replacing an unneeded key

Sometimes users are familiar with a national language keyboard, and mimicing its general layout may be helpful for transition. As with a physical keyboard, you can replace an unneeded letter from your underlying keyboard that does not exist in your orthography with something useful. For example, if "V" does not exist in your orthography, you can replace the "v" and "V" with something more useful. If users may still need the "V", for example, to access **V**imeo.com, you may choose to add the "v" back as a

longpress (don't forget to add the capital "V" as a longpress on the capital letter).

Adding an Extra Key

One touch option that is not available on a physical keyboard is adding a new key. One may be able to add one or two keys to a row to fit in more letters, but keep in mind that too many more keys will make the individual keys harder to hit. This will likely not be ideal on a tiny phone, though it may be on a tablet. Less than 10 keys per row is ideal, 14 is probably too many.

Long Press

A Long-press is a natural method of accessing characters similar (visually or phonetically) to the letter represented on the keycap.
For example, a long press on the key for "n" might logically propose "ŋ", and a long press on "o" might logically propose "ɔ".

For languages with a small inventory of diacritics or letters that take diacritics, the long press might also propose accented versions of the letter on the keycap and the similar letter. Thus, a long press on "o" might propose "o,ó,ò,ɔ,ɔ́,ɔ̀" as options. If the number of possible combinations (including stacked diacritics) often exceeds 9, you might consider adding a diacritic row (see below).

Diacritic/Modifier Row:

in situations where diacritics can be stacked, such as diacritics representing tone and nasalization, the possible combinations with base characters expand exponentially. Where the writing system has a large number of diacritics or diacritics are allowed to be stacked, you may consider adding a diacritic row. If outputting decomposed characters, you only have to add the unicode value of the diacritic to the touch key to output it, and it will combine visually in the interface. If you need to output composed characters, you may need to add some code later to reorder and compose the combinations. In this case, the number row (1, 2,3, etc) may be placed on the same layer as the punctuation.

Example: Cameroon Keyboard (Uses Diacritic Row)

Note: Adding rows of keys will make the keyboard take up more space on the screen. Test the keyboard with phones in use in your area.

Special Layer:

If many characters are accessed through the same Modifier Key or Dead Key, you may consider using a special layer to lay them out. If a large portion of you users are already accustomed to Dead or Modifier keys, consider laying them out in the same location "behind" the simple character. This method can be used as well as long-presses, giving the user multiple options to produce the same character.

Example: Cameroon Keyboard (Uses Special Layers)

Hint: There are some advantages in Keyman Developer to using the RALT and SHIFT-RALT layers instead of creating custom layers. You can add lower-case special characters to the RALT layer and upper-case special characters to the SHIFT-RALT layer.

Note: If you need some keys to be "disabled" on your special layers, but want to leave them to maintain your structure, you have two options. Selecting "Spacer" as the Key Type in developer will leave an empty "untouchable" space the same size as the original key. Selecting "Blank" will create an "untouchable" key that looks greyed out.

Special Features:

You may want to add some special features to your keyboard to improve usability.

Allow Diacritics ONLY on letters:

This code is intended to block combining diacritics from being placed on non-letters. Note: This code lists all characters produced by the keyboard that do not take a diacritic, which in this case is shorter than the list of letters and diacritics. Alternately, one could choose to invert this code with a similar effect.

```
c This store should contain any character (number, letter, or punctuation) that cannot take a diacritic.
store(diablock) &quot; 0123456789?!;:&amp;#039;-
_=&amp;lt;&amp;copy;&amp;reg;&amp;gt;.,[]{}\\/@&amp;deg;#$%^&amp;amp;*()&amp;laquo;&amp;raquo;&amp;lsaquo;&amp;rsaquo;&amp;lsquo;&amp;ldquo;&amp;rsquo;&amp;rdquo;&amp;euro;&amp;yen;&amp;pound;&amp;hellip;&amp;dagger;&amp;quot; U+0022 U+007C c Handle Touch Diacritics (ignoring on non-letters) c This section should be repeated once for each diacritic your keyboard produces. c This line cancels the diacritic if the cursor is preceded by a character that does not accept diacritics. any(diablock) + [T_0300] &amp;gt; context c This line outputs the diacritic U0300 in any other situation. + [T_0300] &amp;gt; U+0300
```

Double Space after word-forming character outputs a period, and mimic automatic capitalization.

Note: These features may later become options enabled through the Language Model or other means. Once this feature is officially added to Keyman, remove these temporary workarounds from the code.

```
c This store should list all letters, diacritics and word-medial punctuation that the keyboard can generate. Numbers should NOT be included. store(word) &quot; a&amp;aelig;ɛbƀcdɗeəfg’'hiɨjklmnŋoɔ&amp;oslash;&amp;oelig;pqrstuʉvwŵxyyzA&amp;AElig;ƐBƁBCDƊDEƏFG’'IHIɨJKLMNŊ]
```

OↃØŒPQRSTUᵾVWẄXYYZ◌-'" U+0300 U+0304 U+0301 U+030C U+0302 U+0303 U+0308 U+0327 U+03B1 U+030D U+0330 c This store should list all phrase-final punctuation. store(final) "!?." c Pressing space after phrase-final punctuation wil activate the Shift layer. platform('touch') any(word) any(final) + [K_SPACE] > index(word,2) index(final,3) " " layer('shift') c Pressing Space twice after a word-final character will add a period, a space, and enable the shift layer. platform('touch') any(word) U+0020 + [K_SPACE] > index(word,2) U+002E " " layer('shift')