Developer Tutorial

Moidfy a Desktop Keyboard

Session 2

This session we will modify a desktop keyboard for the Dagbani language of Ghana.

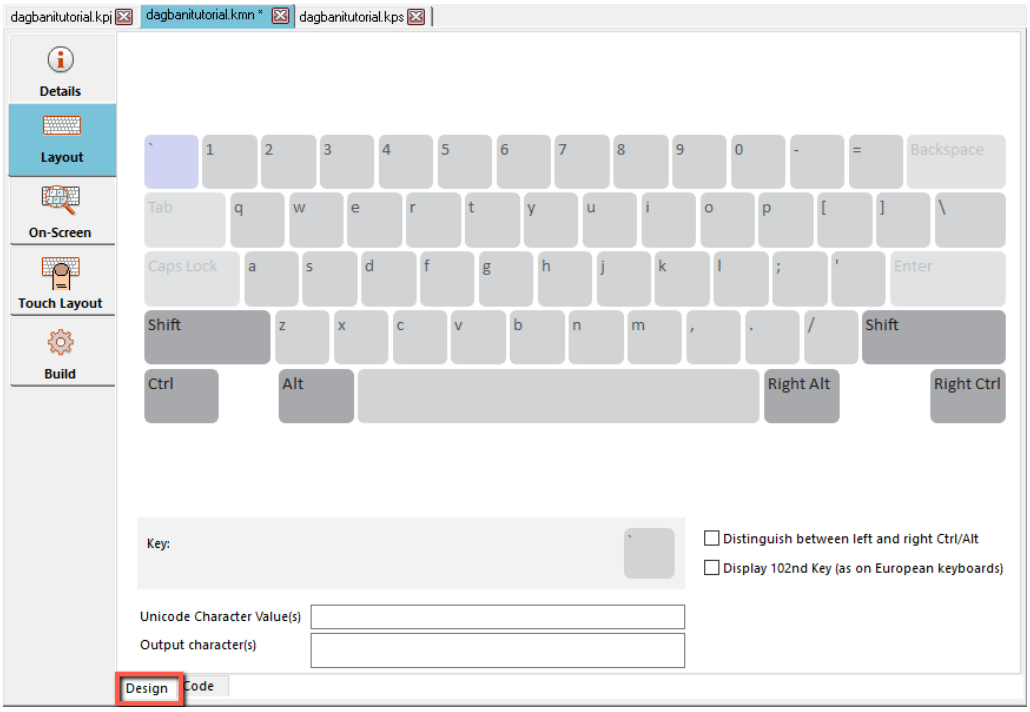
1. Start **Keyman Developer**.

2. In the **Project** menu, point to **Recent Projects**, click **DagbaniTutorial.kpj**.

3. In the **Project - Keyboard** dialog box, click **Keyboards**. Then click **dagbanitutorial.kmn**. The **Details** page appears.

4. Click **Layout**. The **Layout** page appears. There is a **Design** tab and a **Code** tab. The **Code** tab shows us the code that Keyman compiles to build the keyboard. The **Design** tab shows a pictorial picture of the keyboard.

5. We will show how to make changes using the **Design** tab. Click **Design** tab.

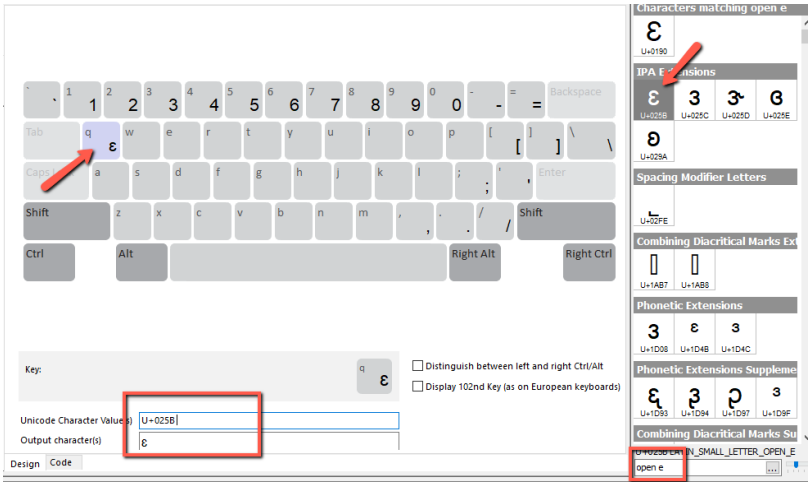


6. In this tab, we see a visual representation of the keyboard. We can change what the keyboard will do when we press a certain key.

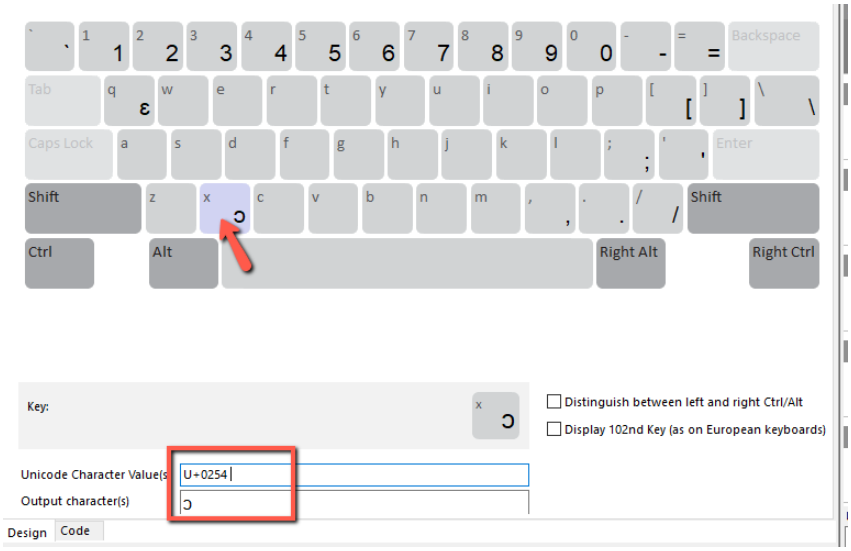
> Since the q and x characters are not used in the Dagbani language, we could reassign the keys to other characters in the language like open e and open o, the other two vowels in their language. First we want to see the character map, so in the **View** menu, set the **Character Map** menu option.
>
> To change the q key to open e key, click on the **q** key in the layout to select it. In the **Character Map** pane, enter **open e** in the textbox at the bottom of the pane. Highlight the Latin small letter open e character by clicking it and then drag it to the q key. Then double-click the Latin small letter open e character in the character map. Note that the **Unicode Character Value** and the **Output character** box are updated with open e Unicode codepoint

and character respectively.

Note that we can enter in the textbox at the bottom of the pane the Unicode value (e.g. 025B) to bring up the character. Or we could enter a range of Unicode values (e.g. 0620-064F) if we are working with a range of characters.



To change the x key to open o key, click on the **x** key in the layout to select it. In the **Character Map** pane, enter **open o** in the textbox at the bottom of the pane. Highlight the Latin small letter open o character by clicking it. Then double-click it or drag it to the x key. Note that the **Unicode Character Value** and the **Output character** box are updated with open o Unicode codepoint and character respectively.

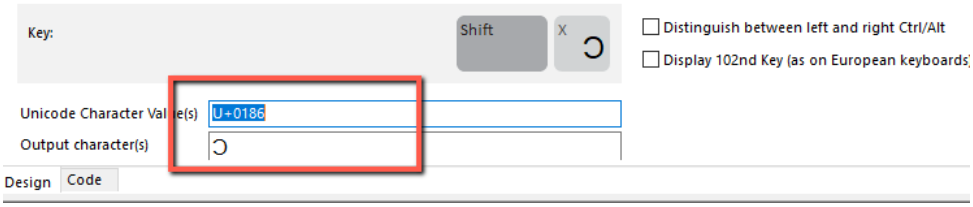


7. We would also need to do this for the capital letters. So first we need to click the Shift key on the keyboard. The Shift key will change colors and the keyboard will change to reflect what happens when we use the Shift key.

To change the Shift-q key to the upper-case open e key, select the Q key. In the **Character Map** pane, enter **open e** in the textbox at the bottom of the pane. Highlight the Latin capital letter open e character by clicking it. Then double-click it. Note that the **Unicode Character Value** and the **Output character** box are updated with the upper-case open e Unicode codepoint and character respectively.

To change the Shift-x key to the upper-case open o key, select the X key. In the **Character Map** pane, enter **open o** in the textbox at the bottom of the pane. Highlight the Latin capital letter

open o character by clicking it. Then double-click it. Note that the **Unicode Character Value** and the **Output character** box are updated with the upper-case open o Unicode codepoint and character respectively.



Let's look at the code we generated by clicking **Code**. Note that we now have four new rules under the **group (main)**.

8. In this case, we do not really to replace the q and x keys, since we may want to use them for borrowed foreign words. Also, we would need more keys for the consonants not on the keyboard. So, we will delete the keys that we just added, by deleting the text in the **Unicode Character Value(s)** text boxes for the four keys in the **Design** tab. Or we could delete the keys, by deleting the fours line of generated code in the **Code** tab

9. We will use a different method for adding those keys. First click the **Code** tab.

We will want to paste back the NCAPs lines that we save in a text file to their original position.

Now we are ready to add some code to add our special characters. We are going use a semicolon plus a character to enter our special characters. We could add a line for each special character as below, using the keyman rule structure Content + Keystroke gives Output.

Begin Unicode > use(main)

group(main) using keys

";" + "e" > "ɛ"

";" + "o" > "ɔ"

";" + "n" > "ŋ"

";" + "g" > "ɣ"

";"+ "z" > "ʒ"

";" + "E" > "Ɛ"

";" + "O" > "Ɔ"

";" + "N" > "ŋ"

";" + "G" > "ɣ"

";" + "Z" > "ʒ"

By pressing the semicolon key followed by the e key we will get the open e. This same principle applies to the rest of the special characters as above.

```
begin Unicode > use(main)

group(main) using keys

";" + "e" > "ɛ"
";" + "o" > "ɔ"
";" + "n" > "ŋ"
";" + "g" > "ɣ"
";"+ "z" > "ʒ"
";" + "E" > "Ɛ"
";" + "O" > "Ɔ"
";" + "N" > "Ŋ"
";" + "G" > "Ɣ"
";" + "Z" > "Ʒ"
```

But there is a more efficient way to write this code. We can create variables that hold a group of characters. We will create two groups of equal size, and then in the rule, we will tell Keyman to replace a member of the one group with the corresponding member of the other group.

begin Unicode > use(main)

store(basekey) "eongzEONGZ"

store(output_char) "ɛɔŋɣʒƐƆŊƔƷ"

group(main) using keys

";" + any(basekey) > index(output_char,2)

Let's look at this rule.

';' + any(basekey) > index(output_char,2).

This says, “When there is a semicolon in the context and one of the characters in basekey is typed, outputting the character at the same index point for the second argument on the left side of the expression.”

The number 2 in the index is known as the offset. It is not referring to the second character in the index. Instead, it is saying that the index is based on the second argument on the left.

This example will help clarify. If we wanted to produce a special character every time the base character was typed (swapping a ɛ for every e), then we would remove the semicolon context and write the rule this way:

+ any(basekey) > index(output_char).

Now the index defaults to the first argument on the left side of the >. We don't need to include the offset at the end of the index since 1 is the default.

Now we can enter above into the **Code** pane as indicated below.

10. Click the **Save** icon to save our work.

11. We could have use a deadkey for producing our special characters. A deadkey is like a character that is used in the context or output but never appears on the screen. We would use deadkeys like this:

c semi-colon becomes a deadkey

+ ";" > deadkey(semicolon)

c Handle semi-colon

deadkey(semicolon) + any(basekey) > index(output_char, 2)

c Handle a single semi-colon

deadkey(semicolon) + ";'" > ";'"

Note that for the sake of convenience, a deadkey can also be written in a short form:

dk(semicolon) c This is identical to deadkey(semicolon)

Typing the three rules above in place of the existing rule %22;%22 + any(basekey) > index(output_char,2) for the semi-colon. If we test the keyboard now, we would find that we would be able to produce the special characters in this way, too. But we've introduced another difference to the keyboard now: the semi-colon is no longer displayed before we type the vowel. This is because we are converting the semi-colon to a deadkey.

Notes

A keyboard file is divided into two sections: the header and the rules section. The header section defines the name of the keyboard, its bitmap, and other general settings. The rules are used to define how the keyboard responds to keystrokes from the user, and are divided into groups.

The keyboard header is the first part of a keyboard; it consists of statements that help Keyman identify the keyboard and set default options for it. Each statement in the header must be on a separate line. While there is no technical requirement to put header statements at the start of a keyboard source file, keeping them there helps us to identify them easily, and keeps them consistent with keyboard programs other people might write.

The keyboard rules section is the most important part: it determines the behavior of the keyboard. The section consists of groups, which in turn contain one or more rules which define the responses of the keyboard to certain keystrokes.

There are two types of groups: groups that process the keys pressed and the context, and groups that process the context only. For simple keyboards, the latter type of group will not be required. A group begins with a group statement, and ends either at the start of another group, or at the end of the keyboard file. An example group statement is given below.

group(Main) using keys

The **using keys** clause tells Keyman that this group will process keystroke

Below the group statement there is a series of rule statements. A rule tells Keyman the output to produce for a certain input. A rule consists of three parts: the context, the key, and the output. The context specifies the conditions under which a rule will act. If what is shown in the document to the left of the cursor matches the context of a rule, the rule will be processed. The key specifies which keystroke the rule will act upon. The output determines the characters that are produced by a rule. The output replaces the matched context in the document.