vim-combining2

1. Combining Characters

Combining characters are used for all manner of things. Three examples:

1. Adding an accent to a character, e.g., ā (U+0061,U+0304)
2. Enabling character variations, e.g., ✋ versus ✋🏻 (U+1F590,U+FE0F)
3. HTML5 two code point characters: e.g., ≧̸ (*aka* ≩︀ U+2269,U+FE00)

In Unicode 15, there are 2,450 combining characters categorised as M{char}. Refer: UnicodeData.txt.

An example (noting the "Mn" — Mark, Nonspacing):

```
0300;COMBINING GRAVE ACCENT;Mn;230;NSM;;;;;N;NON-SPACING GRAVE;;;;
```

Some Unicode blocks have descriptions of "Combining…". Examples are "0300..036F; Combining Diacritical Marks" and "FE20..FE2F; Combining Half Marks". However, there are many other combining characters outside of those blocks. For example, U+0483 to U+0486 are combining Cyrillic characters. Further, all the variation selectors, which are used for varying the visual presentation of characters, such as the emoji example (#2), above, are category Mn (Mark, Nonspacing).

2. Searching for / Substituting Combining Characters

One challenge with Vim is that when *using classes* you can neither search for nor substitute characters that are combined with combining characters.

For example, suppose you have the word "Māori", where the ā is U+0061,U+0304. Some searches or substitutions will work, e.g., literals like:

```
/\va%u304
```

or

```
/\v(a|e|i|o|u)[\u300-\u36F]
```

However, the following will not find the character with the combining character U+0304:

```
/\v[a]%u304
```

or

```
/\v.[\u300-\u36F]
```

I raised issue 12361, which was not considered a bug, though there was one comment at least partially in support:

> I would also think it's a bug, at least [e][\u0305] should work the same as e[\u305] I think.

With the issue going quiet / nowhere, I decided to find my own "solution" — or, probably more accurately/fairly, workaround — to it.

3. So, "What's the problem?"

Two simple examples illustrate some of the challenges caused by the current way search and substitution work.

First, a searching scenario:

```
/a.
```

This will find words such as "an" and "at".

However, it will also find:

- ā (a with macron, U+0061,U+0304)
- á (a with acute, U+0061,U+0301)

That *may* not be what you want because humans read those characters as one glyph, so one or both *characters* may be found (or substituted) unintentionally. (And it is, in part, why some precomposed characters like á, i.e., U+00E1, exist. Precomposed characters like that provide a one-code point character, and, as the following shows, it is a combined U+0061,U+0301 - "0061 0301").

```
00E1;LATIN SMALL LETTER A WITH ACUTE;Ll;0;L;0061 0301;;;;N;LATIN SMALL LETTER
A ACUTE;;00C1;;00C1
```

Second, performing some types of substitutions is impossible. Take a scenario such as where you want to substitute all characters of a class where each character has a particular variation selector to another variation selector (or remove the selector altogeher). You cannot use `.[\uFE0F]` to do so; you would have to use an *or* with all of the characters you wanted to find.

So, for example, with dozens of emoji where the variation selector 16 (U+FE0F) has been used, you would have to list them all out in a big *or* search pattern. That obviously would be inefficent, annoying, and impractical. It would require something like this:

```
/\v(⊗|✋)[\uFE0F]
```

... to find ☹️ and ✋️.

A solution/workaround for this is to pre-process the combining characters. Once substituted with decimal character references, hexadecimal character references, or Python Unicode references, searches and substitutions may then be constructed using classes of characters.

For example, using the example `/\v.[\u300-\u36F]`, once all combining characters are substituted with hexadecimal character references, finding all characters that use them is straightforward:

```
/\v\c.\&#x3[0-6][0-6a-f];
```

Similarly, now ⊛️, ✋️, a̅, etc., may be found with a search like:

```
/\v.\&#x[[:alnum:]]+;
```

4. What this plugin does

This plugin provides a means of substituting all M{char} category characters with either a decimal character reference (`'&#' [0-9]+ ';'`), hexadecimal character reference (`'&#x' [0-9A-F]+ ';'`), or a Python Unicode character (`"\u" [0-9a-z]{4}` *or, where necessary* `"\U" [0-9a-z]{8}`).

Three commands have been created to do this:

- C2d — Combining to decimal
- C2h — Combining to hexadecimal
- C2p — Combining to Python

The demonstrations of C2h and C2d, which follow, show two of these commands in action. The input buffer is left as-is, with a new buffer created alongside it for the user to determine whether they want to use it or otherwise discard it.

5. Demonstrations

C2h using gvim (my preferred Vim flavour 😎🖼):

Demonstration of C2h in gvim

And to show it works in Neovim too, C2d:

Demonstration of C2h in Neovim

6. How it works

Although it is possible to substitute combining characters with a reverse loop using Vimscript, I decided to use the Python Unicode Character Database (UCD) module, unicodedata in this plugin. That was as much for my own learning, i.e., to see how to use Python "within" Vim, which I had done very little of before.

6.1. The combining2.vim

Consequently, the only vimscript is combining2.vim, which has just four lines:

```
let s:path = substitute(expand('<sfile>:p:h'), '\\', '/', 'g')
command! C2d silent execute ":py3file " .. s:path .. "/combining2dec.py"
command! C2h silent execute ":py3file " .. s:path .. "/combining2hex.py"
command! C2p silent execute ":py3file " .. s:path .. "/combining2py.py"
```

1. The first line determines the path to the script, which is where the Python scripts are similarly located.
2. The `command!` lines define the three commands, which, when used, execute, using `py3file`, the applicable Python script on the contents of the current buffer.

6.2. The Python Scripts

The code in the `.py` files has a few comments, though with only a dozen substantive lines of code (in, e.g., combining2hex.py) not much explanation is necessary. Key points are:

1. `import unicodedata, vim` is used to import the required modules
2. A `result` variable is created
3. The lines in the current buffer are looped through
    a. Initialise the `sline` variable (it's used to store the replacement line)
    b. Where a character is in category Mc, Me, or Mn, it is replaced with the applicable decimal, hexadecimal, or Python reference, and added to `sline`
    c. Other non-M? characters are passed to `sline` as-is
    d. At the end of each line, add a NewLine character to `sline`
    e. Add `sline` to `result`
4. Add the `result` to the * register
5. Split the window, create a new buffer, and put the * register into it.

7. Installation

7.1. Using the in-built Vim package management

Refer Vim's in-built package management. This example is for Windows, so adjust it accordingly if you are using a Linux distro, create any necessary directories, etc.:

```
git clone https://github.com/kennypete/vim-combining2
~\vimfiles\pack\plugins\start\vim-combining2
```

Alternatively, download the .zip from https://github.com/kennypete/vim-combining2 and unzip the contents within the folder vim-combining2-main to `~\vimfiles\pack\plugins\start\vim-combining2`.

7.2. Using a plugin manager

If you use a plugin manager, you probably already know how to use it. Nonetheless, here are simple steps explaining how to do so with vim-plug (using "shorthand notation"):

- In the vim-plug section of your `_vimrc`, add `Plug 'kennypete/vim-combining2'` between `call plug#begin()` and `call plug#end()`.
- Reload your `_vimrc` and `:PlugInstall` to install plugins, which should install `vim-combining2`.

7.3. Running the Python Scripts Independently

You don't need to install a plugin. If you want to only use one or more of the Python scripts, just download it/them and run them from Vim with `py3file`, e.g.:

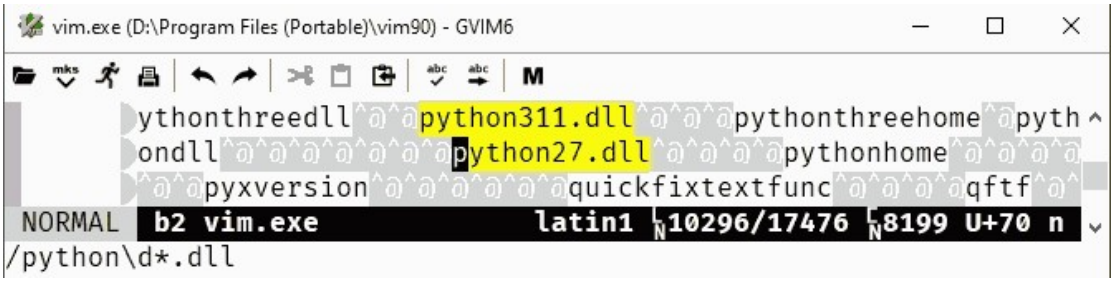
```
:py3file {path}combining2hex.py
```

8. Python 3

You also need Python 3 installed, of course. For example, if you are using the latest gvim (at the time of writing version 9.0 with patch 1677) then you will need Python 3.11. As explained at python-dynamic:

> The name of the DLL should match the Python version Vim was compiled with. ... For Python 3 ... edit "gvim.exe" and search for "python\d*.dll\c".

This is *literal*. So, for example, drag the vim.exe into a gvim window and then execute the search. It will find **python311.dll** (or whatever the version the vim.exe was compiled with).



Neovim's requirements are different. As I don't use it, other than when testing whether things also work with it, all I will say is that I had to read the Neovim documentation and run `.\python.exe -m pip install --user --upgrade pynvim` from my Python 3.11 installation directory from PowerShell.