

How to create a new IME in about 15 minutes with SCIM and scim-tables

Author: JanuszPrusaczyk<JanuszPrusaczykSPAMFILTER@unknown.com>

With SCIM, you can create a table based input method without programming. We are going to use scim-tables IME engine.

There are many input methods available in SCIM. But sometimes we may need something special - maybe you want to input cuneiform, Tangut, Jurchen, Rongorongo or Tocharian. You will not find IMEs for these languages (at least now. I guess it will change in the future ;)) Another reason can be your desire to input a supported language more conveniently. If you use English keyboard and are not a native speaker, the existing IMEs may not be convenient for you at all. Yet another example: maybe you came up with a proposition of a new transcription for an existing language (I know this happened with Cantonese) and want to use it in IME on your own computer.

I assume that your system already supports the language you need IME for, ie. there is some standard of encoding, you have fonts and generally programs on your system handle the data in your target language, you just lack suitable IME.

Specification of the problem

Let us assume that we want to modify existing Korean IME to allow input using transcription (similar to Chinese pinyin input). It is more convenient for non-native speaker with English keyboard.

It is easier with a simple project

We can save ourselves effort by creating a simple project with a makefile. I mean, it saves effort if we continue to work on one or more input tables. I did create a project, you do not have to follow my example. It is not worth the effort for one-time job, but if you want to spend some time editing your tables, I suggest you to create a project.

First, we create a directory structure (yours can be different, but then you will have to modify makefile):
`mkdir tables`

```
cd tables
touch Makefile
mkdir bin
mkdir icons
mkdir src
```

My Makefile looks like this (I have three tables in the ./src right now):

```
SCIM_TABLES_DIR=/usr/share/scim/tables
SCIM_ICONS_DIR=/usr/share/scim/icons

SRC_DIR=./src
BIN_DIR=./bin
ICON_DIR=./icons

SCIM_MAKE_TABLE=scim-make-table

compile: compile-korean compile-Viqr compile-akkadian

compile-ko-latin: $(BIN_DIR)/korean.bin
compile-Viqr: $(BIN_DIR)/Viqr.bin
compile-akkadian: $(BIN_DIR)/akkadian.bin

install: install-korean install-Viqr install-akkadian
uninstall: uninstall-korean uninstall-Viqr uninstall-akkadian

install-korean: compile-korean
cp $(BIN_DIR)/korean.bin $(SCIM_TABLES_DIR)
cp $(ICON_DIR)/korean.png $(SCIM_ICONS_DIR)

uninstall-korean:
rm $(SCIM_TABLES_DIR)/korean.bin
rm $(SCIM_ICONS_DIR)/korean.png

install-Viqr: compile-Viqr
cp $(BIN_DIR)/Viqr.bin $(SCIM_TABLES_DIR)
cp $(ICON_DIR)/Viqr.png $(SCIM_ICONS_DIR)

uninstall-Viqr:
rm $(SCIM_TABLES_DIR)/Viqr.bin
rm $(SCIM_ICONS_DIR)/Viqr.png

install-akkadian: compile-akkadian
cp $(BIN_DIR)/akkadian.bin $(SCIM_TABLES_DIR)
cp $(ICON_DIR)/akkadian.png $(SCIM_ICONS_DIR)

uninstall-akkadian:
rm $(SCIM_TABLES_DIR)/akkadian.bin
rm $(SCIM_ICONS_DIR)/akkadian.png

clean:
rm -f $(BIN_DIR)/*

$(BIN_DIR)/korean.bin: $(SRC_DIR)/korean.txt.in
sed -e \\ 's,@SCIM_ICONDIR@,$(SCIM_ICONS_DIR),g\\ '
$(SRC_DIR)/korean.txt.in > $(BIN_DIR)/korean.txt
$(SCIM_MAKE_TABLE) $(BIN_DIR)/korean.txt -b -o
$(BIN_DIR)/korean.bin
```

```

rm $(BIN_DIR)/korean.txt

$(BIN_DIR)/Viqr.bin: $(SRC_DIR)/Viqr.txt.in
    sed -e \\ 's,@SCIM_ICONDIR@,$(SCIM_ICONS_DIR),g\\ '
$(SRC_DIR)/Viqr.txt.in > $(BIN_DIR)/Viqr.txt
    $(SCIM_MAKE_TABLE) $(BIN_DIR)/Viqr.txt -b -o $(BIN_DIR)/Viqr.bin
rm $(BIN_DIR)/Viqr.txt

$(BIN_DIR)/akkadian.bin: $(SRC_DIR)/akkadian.txt.in
    sed -e \\ 's,@SCIM_ICONDIR@,$(SCIM_ICONS_DIR),g\\ '
$(SRC_DIR)/akkadian.txt.in > $(BIN_DIR)/akkadian.txt
    $(SCIM_MAKE_TABLE) $(BIN_DIR)/akkadian.txt -b -o
$(BIN_DIR)/akkadian.bin
rm $(BIN_DIR)/akkadian.txt

```

It generally means that if I am in the tables directory and run make command, it will generate source files with correct icon path in `./bin` directory (the `.in` extension gets cut off), and compile it afterwards. For example in case of Korean table the compile command looks like this: `scim-make-table ./bin/korean.txt -b -o ./bin/korean.bin`. The binary table will be written to `./bin` directory. If I update one file, for example `korean.txt.in`, I do not need to compile and reinstall everything. In this case I use command `make install-korean`. This command will first compile the updated table, and install it afterwards.

The installation is equal to copying the binary table and icon to correct locations:

```

cp ./bin/korean.bin /usr/share/scim/tables
cp ./icons/korean.png /usr/share/scim/icons

```

I can also do `make uninstall-korean`. Another command, `make clean` clears the `./bin` directory.

Let us finally begin

The first thing that we need is an icon. Mine is `korean.png` and I put it in `./icons` directory. Remember, it will be copied to `/usr/share/scim/icons`. You may as well put it directly in this location.

Now we need the source of the table. As we just modify existing table, we copy the source of the table that we modify. It is in `scim-tables` sources.

```

cp /dat/src/cpp/scim/scim-tables-0.4.3/ko/Hangul.txt.in ./src
mv ./src/Hangul.txt.in ./src/korean.txt.in

```

We need UUID for our new input:

```
$ uuidgen  
ec1343c3-cd84-421c-b653-fd4793759e6c
```

We open ./src/korean.txt.in in our favourite text editor and make necessary corrections.

First, we change UUID line to: `UUID = ec1343c3-cd84-421c-b653-fd4793759e6c`

```
We also update serial number, icon location and input name:  
SERIAL_NUMBER = 20041005  
ICON = @SCIM_ICONDIR@/korean.png  
NAME = Korean
```

This is enough to compile and install the table (`make && make install`). Now is your part - modify the table data and do not forget to update `MAX KEY LENGTH` value after that.

Sometimes we start the table from scratch. In these cases it may help to generate the dummy table of characters to begin your work with. I do it with Perl - it saves me time. For example, I am going to start work on an latin prefix notation table to allow input of latin accented characters. The script I used to generate the raw table to begin my work with was: `#!/usr/bin/perl`

```
# Basic  
for my (0x0021 .. 0x007e) {  
    print "x", sprintf("%x",), " ",  
    pack(\\'U\\',), " 0  
";  
}  
  
# Supplement  
  
for my (0x00A1 .. 0x00ff) {  
    print "x", sprintf("%x",), " ",  
    pack(\\'U\\',), " 0  
";  
}  
  
# Extended A  
  
for my (0x0100 .. 0x017f) {  
    print "x", sprintf("%x",), " ",  
    pack(\\'U\\',), " 0  
";  
}
```

```

# Extended B
for my (0x0180 .. 0x0236) {
    print "x", sprintf("%x",), " ",
    pack('\U',), " 0
";
}

# Extended Additional
for my (0x1e00 .. 0x1ef9) {
    print "x", sprintf("%x",), " ",
    pack('\U',), " 0
";
}

```

This script prints the table to STDIN. We save it by redirecting it's output to a file, like this:
`./latintab.pl > latin.txt.`

When you are ready with your input tables, type this command in your project directory:`make install`

Final word

You have to restart SCIM to load (or reload) your table. It makes testing difficult.

I did not test much this approach. I just describe what worked for me. Good luck!