Supporting APL keyboards on Linux

Producing a clean, integrated, satisfying APL keyboard

Geoff Streeter

Dyalog Ltd

geoff@dyalog.com

Main Topic: keyboards

Abstract

All of the GUI interfaces for Linux are built on X-windows. X-windows is a client-server design. The screen-keyboard-mouse side is the server. Programs running on any machine use the services provided to receive input and draw output.

There is an extension to X-windows servers called xkb which provides additional keyboard support.

This paper describes how Dyalog have used this feature to implement an APL keyboard that overlays the original keyboard. The APL character set is available in all applications, whether the underlying keyboard is US, Russian – or any other language.

Objective

The main objective of this work was to create an APL keyboard that would overlay the users existing keyboard. This APL keyboard would be usable in any program be it a word processor, text editor, email client … anything.

X and Xkb

There are three desktop operating systems in common use - various versions of Windows, Linux and Mac OS/X. This work focuses on Linux. Specifically Linux running one of the several environments that run on top of X windows. Thus KDE, Gnome, XFCE etc.

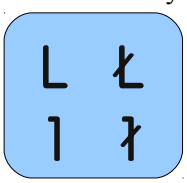
This work builds on the work of Erik Fortune of Silicon Graphics Inc. In 1996 he created the X keyboard extension now normally known as "xkb"[1]. This was driven by a desire to implement the ISO9995 standard for keyboards[2].

X windows is a protocol (communication layer) defined between an X server and an X client. The X server is the display, keyboard, mouse end. It provides services to the program. So if the program is running on an AIX server in a rack somewhere and the user is sitting at a Linux desktop the Linux end is the X server and the AIX end is the X client. Of course both client and server may be on the same machine and for running applications like word processing that would normal.

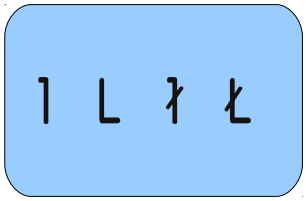
The X protocol itself provides for eight modifier keys which showed a lot of foresight. Examples of modifier keys are "shift", "control" and "alt". Xkb adds a possibility of virtual modifier keys so that some other keystroke is interpreted as producing a particular combination of modifier keys. This work has not needed to use that capability to any extent additional to normal Linux.

The characters generated by an Xkb keyboard definition are Unicode characters.

Consider a key :

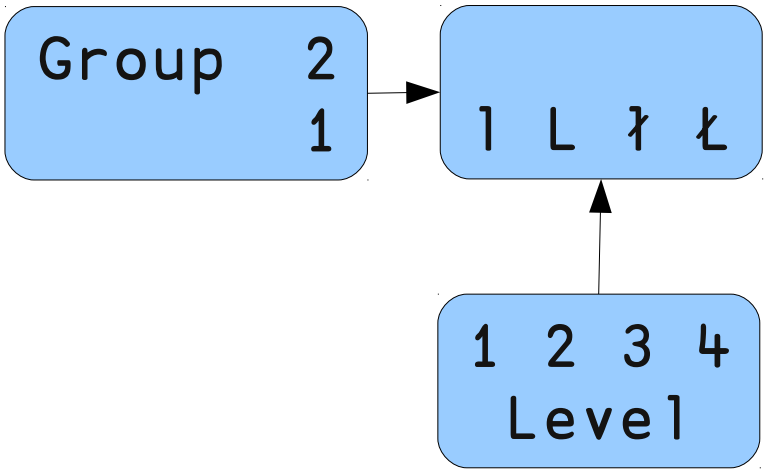


This has four glyphs. However, the arrangement as a matrix is just a convenience for the keyboard designer. From an xkb point of view it looks like:



Characters are selected using “level”. Level is determined by the values of the eight modifier keys. Thus there can be 256 levels which can be used for any one keypress. In ISO9995 terms the state of these modifiers are known as “qualifiers”.

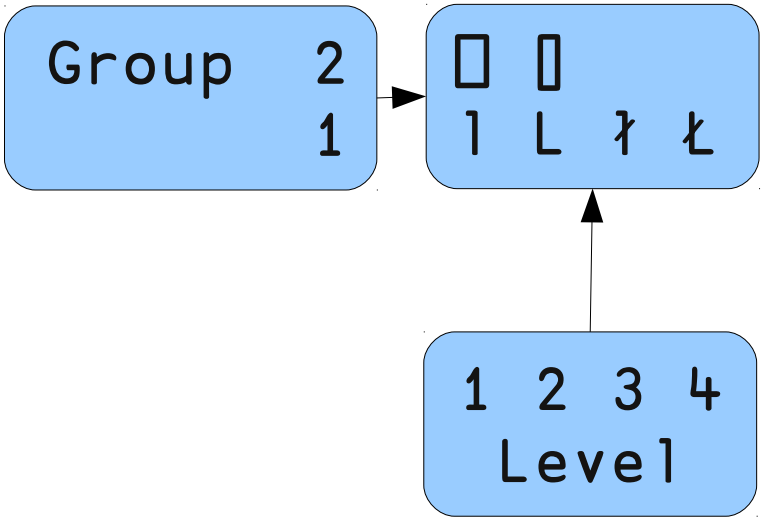
ISO9995 added another concept which Xkb adopted - “group”. There can be four groups. Thus there can be 1024 glyphs on an individual key.



This would be a challenge to any keyboard engraver. The group information is passed differently to the modifier information when a key stroke is passed from X server to X client. However, from a user perspective it is just more shift keys.

Adding APL

The use of a group enables us to overlay the group 1 keyboard with a completely different keyboard. So we can add APL characters.



The keys used to select groups are independent of the modifier keys. So we can code the APL keyboard as group 2 and add it to any existing keyboard that doesn't use group 2. In 2007 this is exactly what was done.

Integrating with current practice

Recently, X.org and Xfree which are the suppliers of X to Linux re-thought the way that groups are handled. This has helped this project. Sergey Udaltsov is the maintainer for the X.org project. He is a Russian who also uses Ukrainian and English. This means that he was constantly switching languages. Instead of coding all of the group information in the tables that map the keystrokes to the symbols, he mapped languages to groups. Each individual language keyboard is defined only for group 1. When languages are added that definition is changed to a new group. Thus up to four languages can be used with what X regards as a single keyboard.

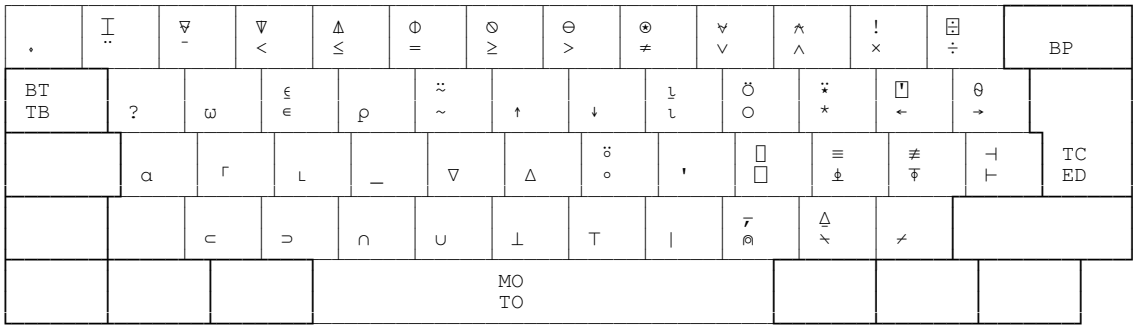
APL can be one of those languages just like any other. This is important. APL is not special in the Linux environment. It is just another language. The normal environment is not being hi-jacked in any way.

Linux xkb comes pre-configured with some group selection techniques. There are two strategies for this. Firstly a "latching" strategy which selects a group only whilst its activation key is kept pressed. Secondly, a "locking" strategy that selects the group until some other activation selects a different group. For APL programmers a latching strategy is ideal because only a single, or very few, characters are required before the keyboard reverts to normal. For real language switching, like from Russian to English, a locking strategy is better because the language is going to be in use for a sentence or a paragraph. Linux provides a number of activation choices for latching to the next group. The most useful is, probably, the otherwise unused "windows" keys that are present on most modern keyboards - even laptop ones. Locking shift selections can be chosen to increase group, decrease group, select the first group, select the last group … .

From an xkb point of view the concept is "group". From a KDE or Gnome configuration point of view the concept is "layout".

It is going to help if APL is placed as the next language after the one in which the text part of APL code is typed. So a Russian would probably want to allocate languages in the order Russian, US, APL. Then when writing APL code he would do a locking group switch to US[1] and use the latching shift to obtain the additional APL characters. He might do a locking shift back to Russian to comment his code.

Dyalog has defined an APL keyboard.



For the usual APL glyphs it matches the keyboard that we have had engraved. This is a very traditional allocation of APL symbols to keys so most programmers should be comfortable using it. Some line drawing characters have been placed on the numeric keypad.

	RD	TG	LN
┌	┬	┐	TL
├	┼	┤	
└	┴	┘	
─		│	

In addition the keyboard supplies the codes that Dyalog have supported for many years to do things like "open the editor". These are largely where a Dyalog Windows user would expect them to be. Except this

[1]Most, possibly all, current APLs restrict the character set that can be used for variable or function names to the upper and lower case alphabet, the digits and a few other characters. Since names tend to be longer than function or operator sequences, the base keyboard when typing APL code is easier to use if it is Latin based. This may change as APL develops in a Unicode environment.

keyboard is a true overlay so, for example, the “ED” key, which a Windows user would expect to be on Shift+Enter is on APL+Enter, where “APL” is the key the user has chosen to latch the next group. These special codes have been mapped into the “Private Use” part of the Unicode standard. Their presence should not interfere with any other application.

The keyboard mapping is specified in /usr/share/X11/xkb/symbols/apl. It contains stanzas like[2] :

```
key <AC09> {
     type[Group1] = "TWO_LEVEL",
     symbols[Group1] = [ U2395, U2337 ] // ⌕, ⍷
};
```

If, in the future, it is desired to add more characters, then using three levels would only require APL+AltGr+character. This is from an Xkb point of view. From a Windows perspective putting three APL characters on a key would be more difficult[3].

It should be noted that the APL is strictly an overlay[4]. So the alpha is on the key at the left edge of the middle row. Key <AC01> in Xkb parlance. This is not where, say, a French APL user might expect it to be. On a French keyboard the “A” is at the left edge of the top row. A French APL keyboard would place the alpha there. However, the French also relocate the “W” but the omega does not move with it. By using a true overlay all of the APL characters are fixed. Xkb allows a “variant”. A layout which is a modification of an existing layout. So a French variant, if desired, is easy to code and reasonably easy to select.

If it is desired to add a new APL glyph, or indeed a mathematics glyph, to the keyboard only one file needs to be changed on the X server. The change is easy, doesn't need a reboot and works everywhere that understands Unicode[5]. So, for example, to add, say, ⍠ to the L key would change the above stanza to:

```
key <AC09> {
     type[Group1] = "THREE_LEVEL",
     symbols[Group1] = [ U2395, U2337, U2360 ] // ⌕, ⍷, ⍠
};
```

The additional character would challenge the keyboard engraver but is otherwise available with just a restart of the X server[6].

Fonts

This paper is really about keyboards but keyboards produce glyphs. Glyphs need to be displayed. So fonts are an issue. Linux has a concept of a virtual font which is a list of fonts and a particular glyph will be taken from the first font in the list that supports it. The default “monospace” font works well for the APL characters, the only exceptions being base and top (decode, encode) which are not well rendered. Of course the APL385-Unicode font can be added to the virtual font “monospace”. The font used to display the keyboard above was “FreeMono” and has all of the characters. The only one which is rendered particularly dubiously is the diamond which is very small.

---

[2]The syntax for this seems to be evolving. The latest xkb versions will accept:

```
key <AC09> { [ U2395, U2337 ] };
```

The <AC09> locates the key on the physical keyboard. U2395 specifies the Unicode location of the ⌕ glyph.

Documentation for XKB is hard to come by. However, the translation of Ivan Pascal's Russian documentation at http://pascal.tsu.ru/en/ is a starting point.

[3]Windows has fewer qualifier keys. In addition keyboard layout switching is more difficult. A latching accelerator key combination to temporarily switch layout is not possible.

[4]Not in the ISO9995 sense of an overlay

[5]The same job for Dyalog on Windows would require changes to the .din files for the IME, the Cntl keyboard and the AltGr keyboard for each language.

[6]The compiler that reads these stanzas is both uncommunicative and unforgiving. So errors simply mean that the language at fault, in this case APL, is unavailable following the X server restart.

Some programs will not use virtual fonts – usually ones for which font selection is important like word processors and desktop publishing software. For these, you obviously need to pick a suitable font that contains the characters you wish to use in the style you require. Nevertheless using APL on Linux is possible without additional fonts being installed and this is true whether the APL is running on the same box or on something remote like an AIX server.

Communicating with APL itself

This paper concentrates on keyboarding for APL characters for any application. It concentrates on the desktop applications running on Linux. From an APL developer's perspective the most important of those applications is the one that drives APL itself.

APL applications developed and run on servers in racks running Unix are largely based on sockets. The user interfaces are implemented in applications running on desktop operating systems. Nevertheless, such socket based applications need developing and maintaining. In the absence of a suitable GUI client[7] character based clients based on terminal emulators are used. Some applications still use terminal emulators as the user interface.

There are several terminal emulators in the Linux environment. KDE provides "Konsole" and Gnome provides "gnome-terminal". Both can, and do by default, transmit and receive UTF-8 streams. Both are built on top of "xterm" which is similar to a vt420. Any APL that supports character based terminals needs mechanisms to adjust to various terminals. Dyalog is no different and has had these mechanisms for many years. Dyalog's introduction of Unicode support induced some enhancement but no real change. It was easy to provide support both for the Unicode products and the Classic products. Indeed, support was easy to provide for the old products. It is straightforward to run very old Dyalog versions hosted by Konsole or gnome-terminal. The oldest version Dyalog has run like this is v6.2.

The same mechanisms are used to support the "PuTTy" terminal emulator running on Windows. However, keyboarding for that environment works[8] but is frustrating to use. The Linux environment is much more pleasant.

Bibliography

[1] Fortune, Erik,The X Keyboard Extension: Protocol Specification

[2] ISO 9995-1:2006 General Principles Governing Keyboard Layouts[9]

[7] John Daintree is giving a presentation in the Dyalog specific part of this conference showing some work on this.

[8] It has to use the 32 bit IME (Input Method Editor) which needs turning on to get APL characters but turning off for other aspects – like sending interrupts using ctrl+c. PuTTy does not support layouts from the "Microsoft keyboard layout creator".

[9] Erik Fortune was using the 1994 version of ISO9995. There are eight sections to ISO9995.