How to further enhance XKB configuration

Kamil Toman
Ivan U. Pascal

X Version 11, Release 7.7

25 November 2002

Abstract

This guide is aimed to relieve one's labour to create a new (internationalized) keyboard layout. Unlike other documents this guide accents the keymap developer's point of view.

Table of Contents

verview .....
The asics .....
nhancing on guration .....
evels nd roups .....
e ning ew ayouts .....
rede ned ymbol ets .....
ey Types .....
ules .....
escriptive iles of ules 1

Overview

The developer of a new layout should read the kb protocol speci cation (The eyboard tension rotocol peci cation https www. .org docs proto.pdf) at least to clarify for himself some kb-specific terms used in this document and elsewhere in kb con guration. lso it shows wise to understand how the server and a client digest their keyboard inputs (with and without kb).

useful source is also van ascal's te t about kb con guration http www.tsu.ru pascal en kb often referenced throughout this document.

ote that this document covers only enhancements which are to be made to ree version . and 11 . . and above.

The Basics

t the startup (or at later at user's command) server starts its kb keyboard module e tension and reads data from a compiled con guration le.

This compiled con guration le is prepared by the program which behaves altogether as an ordinary compiler (see **man xkbcomp**). ts input are human readable kb con guration les which are veri ed and then composed into a useful kb con guration. Users don't need to mess with themselves for them it is invisible. Usually it is started upon server startup.

s you probably already know the kb con guration consists of ve main modules

eycodes Tables that de nes translation from keyboard scan codes into reasonable symbolic names ma imum minimum legal keycodes symbolic aliases and description of physically present -indicators. The primary sense of this component is to allow de nitions of maps of symbols (see below) to be independent of physical keyboard scancodes. There are two main naming conventions for symbolic names (always four bytes long)

names which e press some traditional meaning like (stands for space bar) or

names which e press some relative positioning on a keyboard for e ample 1 (an e clamation mark on U keyboards) on the right there are keys etc.

Types Types describe how the produced key is changed by active modi ers (like hift ontrol lt ...). There are several prede ned types which cover most of used combinations.

ompat ompatibility component de nes internal behaviour of modi ers. Using compat component you can assign various actions (elaborately described in kb speci cation) to key events. This is also the place where -indicators behaviour is de ned.

ymbols or i1 n purposes this is the most important table. t de nes what values (symbols) are assigned to what keycodes (represented by their symbolic name see above). There may be de ned more than one value

for each key and then it depends on a key type and on modi ers state (respective compat component) which value will be the resulting one.

eometry eometry les aren't used by kb itself but they may be used by some e ternal programs to depict a keyboard image.

ll these components have the les located in kb con guration tree in subdirectories with the same names (usually in `/usr/lib/X11/xkb`).

Enhancing XKB Configuration

ost of kb enhancements concerns a need to de ne new output symbols for the some input key events. n other words a need to de ne a new symbol map (for a new language standard or ust to feel more comfortable when typing te t).

hat do you need to do enerally you have to de ne following things

the map of symbols itself

the rules to allow users to select the new mapping

the description of the new layout

irst of all it is good to go through e isting layouts and to e amine them if there is something you could easily ad ust to t your needs. ven if there is nothing similar you may get some ideas about basic concepts and used tricks.

Levels And Groups

ince ree . . and 11 . . you can use concept of kb con guration. Though it is still in boundaries of kb protocol and general ideas the keymap designer must obey new rules when creating new maps. n e change we get a more powerful and cleaner con guration system.

emember that it is the application which must decide which symbol matches which keycode according to e ective modi er state. The server itself sends only an input event message to. f course usually the general interpretation is processed by lib aw otif t tk and similar libraries. The server only supplies its mapping table (usually upon an application startup).

ou can think of the server's symbol table as of a irregular table where each keycode has its row and where each combination of modi ers determines e actly one column. The resulting cell then gives the proper symbolic value. ot all keycodes need to bind di erent values for di erent combination of modi ers. T key for instance usually doesn't depend on any modi ers so it its row has only one column de ned.

ote that in there is no prior assumption that certain modi ers are bound to certain columns. y editing proper les (see ey Types) this mapping can be changed as well.

Unlike the original protocol the approach is far more e ible. t is comfortable to add one additional term - group. ou can think of a group as of a vector of columns per each keycode (naturally the dimension of this vector may di er for di erent keycodes). hat is it good for The group is not very useful unless you intend to use more than one logically di erent set of symbols (like more than

one alphabet) de ned in a single mapping table. ut then the group has a natural meaning - each symbol set has its own group and changing it means selecting a di erent one. approach allows up to four di erent groups. The columns inside each group are called (shift) levels. The server knows the current group and reports it together with modi er set and with a keycode in key events.

To sum it up

for each keycode keyboard map contains up to four one-dimensional tables - groups (logically di erent symbol sets)

for each group of a keycode keyboard map contains some columns - shift levels (values reached by combinations of hift trl lt ... modi ers)

di erent keycodes can have di erent number of groups

di erent groups of one keycode can have di erent number of shift levels

the current group number is tracked by server

t is clear that if you sanely de ne levels groups and sanely bind modi ers and associated actions you can have simultaneously loaded up to four di erent symbol sets where each of them would reside in its own group.

The multi-layout concept provides a facility to manipulate kb groups and symbol de nitions in a way that allows almost arbitrary composition of prede ned symbol tables. To keep it fully functional you have to

de ne all symbols only in the rst group

(re)de ne any modi ers with e tra care to avoid strange (anisometric) behaviour

Defining New Layouts

ee ome ords bout internals http www.tsu.ru pascal en kb internals.html for e planation of used kb terms and problems addressed by e tension.

ee ommon notes about con guration les language http www.tsu.ru pascal en kb gram-common.html for more precise e planation of synta of kb con guration les.

Predefined XKB Symbol Sets

f you are about to de ne some uropean symbol map e tension you might want to use on of four prede ned latin alphabet layouts.

kay let's assume you want e tend an e isting keymap and you want to override a few keys. et's take a simple U. . keyboard as an e ample (de ned in `pc/gb`)

```
partial default alphanumeric_keys
xkb_symbols "basic" {
  include "pc/latin"

  name[Group1]="Great Britain";
```

```
  key <AE02>  { [         2,   quotedbl,  twosuperior,    oneeighth ] };
  key <AE03>  { [         3,   sterling, threesuperior,    sterling ] };
  key <AC11>  { [apostrophe,         at, dead_circumflex, dead_caron] };
  key <TLDE>  { [     grave,    notsign,          bar,          bar ] };
  key <BKSL>  { [numbersign, asciitilde,   dead_grave,   dead_breve ] };
  key <RALT>  { type[Group1]="TWO_LEVEL",
                [ ISO_Level3_Shift, Multi_key ]   };

  modifier_map Mod5   { <RALT> };
};
```

t de nes a new layout in `basic` variant as an e tension of common latin alphabet layout. The layout (symbol set) name is set to reat ritain . Then there are rede nitions of a few keycodes and a modi ers binding. s you can see the number of shift levels is the same for 11 T and keys but it di ers from number of shift levels of T .

ote that the T key itself is a binding key for od and that it serves like a shift modi er for evelThree together with hift as a multi-key. t is a good habit to respect this rule in a new similar layout.

kay you could now de ne more variants of your new layout besides `basic` simply by including (augmenting overriding ...) the basic de nition and altering what may be needed.

Key Types

The di erences in the number of columns (shift levels) are caused by a di erent types of keys (see the types de nition in section basics). ost keycodes have implicitly set the keytype in the included `pc/latin` le to `FOUR_LEVEL_ALPHABETIC` . The only e ception is T keycode which is e plicitly set `TWO_LEVEL` keytype.

ll those names refer to pre-de ned shift level schemes. Usually you can choose a suitable shift level scheme from `default` types scheme list in proper kb component's subdirectory.

The most used schemes are

The key does not depend on any modi ers. The symbol from rst level is always chosen.

T The key uses a modi er hift and may have two possible values. The second level may be chosen by hift modi er. f ock modi er (usually aps-lock) applies the symbol is further processed using system-speci c capitalization rules. f both hift ock modi er apply the symbol from the second level is taken and capitalization rules are applied (and usually have no e ect).

T The key uses modi ers hift and ock. t may have two possible values. The second level may be chosen by hift modi er. hen ock modi er applies the symbol from the rst level is taken and further processed using system-speci c capitalization rules. f both hift ock

modi er apply the symbol from the rst level is taken and no capitalization rules applied. This is often called shift-cancels-caps behaviour.

T s the same as T but it considers an e tra modi er - evelThree which can be used to gain the symbol value from the third level. f both hift evelThree modi ers apply the value from the third level is also taken. s in T the ock modi er doesn't in uence the resulting level. nly hift and evelThree are taken into that consideration. f the ock modi er is active capitalization rules are applied on the resulting symbol.

U s the same as T but unlike T if both hift evelThree modi ers apply the symbol is taken from the fourth level.

U T s similar to U but also de n cancels-caps behaviour as in T . f ock evelThree apply the symbol from the third level is taken and the capitalization rules are applied. f ock hift evelThree apply the symbol from the third level is taken and no capitalization rules are applied.

s the name suggest this scheme is primarily used for numeric keypads. The scheme considers two modi ers - hift and um ock. f none of modi ers applies the symbol from the rst level is taken. f either hift or um ock modi ers apply the symbol from the second level is taken. f both hift um ock modi ers apply the symbol from the rst level is taken. gain shift-cancels-caps variant.

U s similar to scheme but considers evelThree modi er. f evelThree modi er applies the symbol from the third level is taken. f hift evelThree or um ock evelThree apply the symbol from the fourth level is taken. f all hift um ock evelThree modi ers apply the symbol from the third level is taken. This also shift-cancels-caps variant.

esides that there are several schemes for special purposes

t is similar to T scheme but it considers the ontrol modi er rather than hift. That means the symbol from the second level is chosen by ontrol rather than by hift.

t is similar to T scheme but it considers the lt modi er rather than hift. That means the symbol from the second level is chosen by lt rather than by hift.

T The key uses modi ers lt and ontrol. t may have two possible values.
T f only one modi er (lt or ontrol) applies the symbol from the rst level is chosen. nly if both lt ontrol modi ers apply the symbol from the second level is chosen.

T The key uses modi ers hift and lt. t may have two possible values.
T f only one modi er (lt or hift) applies the symbol from the rst level is chosen. nly if both lt hift modi ers apply the symbol from the second level is chosen.

f needed specialcaps schemes may be used. They rede ne the standard behaviour of all *ALPHABETIC types. The layouts (maps of symbols) with keys de ned in respective types then automatically change their behaviour accordingly. ossible rede nitions are

internal

internal nocancel

shift

shift nocancel

one of these schemes should be used directly. They are de ned merely for 'caps:' kb options (used to globally change the layouts behaviour).

on't alter any of e isting key types. f you need a di erent behaviour create a new one.

More On Definitions Of Types

hen the software deals with a separate type description it gets a complete list of modi ers that should be taken into account from the 'modifiers=<list of modifiers>' list and e pects that a set of 'map[<combination of modifiers>]=<list of modifiers>' instructions that contain the mapping for each combination of modi ers mentioned in that list. odi ers that are not e plicitly listed are T taken into account when the resulting shift level is computed. f some combination is omitted the program (subroutine) should choose the rst level for this combination (a uite reasonable behavior).

ets consider an e ample with two modi ers od ne and odTwo

```
type "..." {
    modifiers = ModOne+ModTwo;
    map[None] = Level1;
    map[ModOne] = Level2;
};
```

n this case the map statements for odTwo only and od ne odTwo are omitted. t means that if the odTwo is active the subroutine can't found e plicit mapping for such combination an will use the i.e. evel1.

ut in the case the type described as

```
type "..." {
    modifiers = ModOne;
    map[None] = Level1;
    map[ModOne] = Level2;
```

```
};
```

the odTwo will not be taken into account and the resulting level depends on the od ne state only. That means odTwo alone produces the evel1 but the combination od ne odTwo produces the evel as well as od ne alone.

hat does it mean if the second modi er is the ock t means that in the rst case (the ock itself is included in the list of modi ers but combinations with this modi er aren't mentioned in the map statements) the internal capitalization rules will be applied to the symbol from the rst level. ut in the second case the capitalization will be applied to the symbol chosen accordingly to the rst modi er - and this can be the symbol from the rst as well as from the second level.

Usually all modi ers introduced in'`modifiers=<list of modifiers>`' list are used for shift level calculation and then discarded. ometimes this is not desirable. f you want to use a modi er for shift level calculation but you don't want to discard it you may list in '`preserve[<combination of modifiers>]=<list of modifiers>`'. That means for a given combination all listed modi ers will be preserved. f the ock modi er is preserved then the resulting symbol is passed to internal capitalization routine regardless whether it has been used for a shift level calculation or not.

ny key type description can use both real and virtual modi ers. ince real modi ers always have standard names it is not necessary to e plicitly declare them. irtual modi ers can have arbitrary names and can be declared (prior using them) directly in key type de nition

```
virtual_modifiers <comma-separated list of modifiers>  ;
```

as seen in for e ample `basic pc` or `mousekeys` key type de nitions.

Rules

nce you are nished with your symbol map you need to add it to rules le. The rules le describes how all the ve basic keycodes types compat symbols and geometry components should be composed to give a sensible resulting kb con guration.

The main advantage of rules over formerly used keymaps is a possibility to simply parameterize (once) ed patterns of con gurations and thus to elegantly allow substitutions of various local con gurations into prede ned templates.

pattern in a rules le (often located in `/usr/lib/X11/xkb/rules`) can be parameterized with four other arguments `Model Layout Variant` and `Options`. or most cases parameters `model` and `layout` should be su icient for choosing a functional keyboard mapping.

The rules le itself is composed of pattern lines and lines with rules. The pattern line starts with an e clamation mark ('`!`') and describes how will the kb interpret the following lines (rules). sample rules le looks like this

```
! model                 = keycodes
  macintosh_old                 = macintosh
  ...
```

```
  *                  = xorg

! model                   = symbols
  hp                   = +inet(%m)
  microsoftpro                  = +inet(%m)
  geniuscomfy                  = +inet(%m)

! model     layout[1]          = symbols
  macintosh us          = macintosh/us%(v[1])
  *             *               =       pc/pc(%m)+pc/%l[1]%(v[1])

! model     layout[2]          = symbols
  macintosh     us              = +macintosh/us[2]%(v[2]):2
  *  *               = +pc/%l[2]%(v[2]):2

! option   = types
  caps:internal   = +caps(internal)
  caps:internal_nocancel = +caps(internal_nocancel)
```

ach rule de nes what certain combination of values on the left side of e ual sign ('=') results in. or e ample a (keyboard) model `macintosh_old` instructs kb to take de nitions of keycodes from le `keycodes/macintosh` while the rest of models (represented by a wild card '`*`') instructs it to take them from le `keycodes/xorg`. The wild card represents all possible values on the left side which were not found in any of the previous rules. The more specialized (more complete) rules have higher precedence than general ones i.e. the more general rules supply reasonable default values.

s you can see some lines contain substitution parameters - the parameters preceded by the percent sign ('`%`'). The rst alphabetical character after the percent sign e pands to the value which has been found on the left side. or e ample `+%l%(v)` e pands into `+cz(bksl)` if the respective values on the left side were `cz` layout in its `bksl` variant. ore if the layout resp. variant parameter is followed by a pair of brackets ('`[` `]`') it means that kb should
. f the brackets are omitted the rst group is the default value.

o the second block of rules enhances symbol de nitions for some particular keyboard models with e tra keys (for internet multimedia ...) . ther models are left intact. imilarly the last block overrides some key type de nitions so the common global behaviour ''shift cancels caps'' or ''shift doesn't cancel caps'' can be selected. The rest of rules produces special symbols for each variant `us` layout of `macintosh` keyboard and standard pc symbols in appropriate variants as a default.

Descriptive Files of Rules

ow you ust need to add a detailed description to `<rules>.xml` description le so the other users (and e ternal programs which often parse this le) know what is your work about.

Old Descriptive Files

The formerly used descriptive les were named `<rules>.lst` ts structure is very simple and uite self descriptive but such simplicity had also some cavities for

e ample there was no way how to describe local variants of layouts and there were problems with the localization of descriptions. To preserve compatibility with some older programs new descriptive les can be converted to old format '.lst'.

or each parameter of rules le should be described its meaning. or the rules le described above the `.lst` le could look like

```
! model
  pc104  Generic 104-key PC
  microsoft Microsoft Natural
  pc98  PC-98xx Series
  macintosh     Original Macintosh
  ...

! layout
  us  U.S. English
  cz  Czech
  de  German
  ...

! option
  caps:internal   uses internal capitalization. Shift cancels Caps
  caps:internal_nocancel uses internal capitalization. Shift doesn't cancel Caps
```

nd that should be it. n oy creating your own kb mapping.