The Answer Girl

Keyboard Wizardry

Is caps lock getting on your nerves? Still looking for a good way to put your Windows keys back to work? Along with other definitive questions... BY PATRICIA JUNG

A new job, a new computer or just a keyboard that bites the dust after years of faithful service – this is a situation that everyone has to face some time – getting on top of a keyboard that will surely be supplied with Windows keys nowadays. Even if you swap the Windows keys, that will not affect the functionality, or lack of it, depending on your distribution.

The Answer Girl

In the world of everyday computing, even under Linux, there are often surprises: Time and again things do not work, or at least not as they're supposed to. The Answer Girl shows how to deal elegantly with such little problems.

An unused Windows key might be regarded as a slight blemish, but the caps lock key is a downright nuisance that will no doubt cause you to inadvertently SHOUT at your innocent computer from time to time.

So why not put the loudmouth to work? You might consider converting the key to a second left shift key, as Caps Lock is often hit by mistake instead of left shift. Some users simply shift the caps lock functions from the Caps Lock key to the left control key. And you can also consider an individual option – but let's first look at how you can accomplish that.

Unfortunately we are not looking at a single solution because the two user interfaces common to Linux, i.e. the character-based console and the X Window GUI have separate methods of defining keys. If you have previously been required to install a Linux distribution with foreign keyboard mappings or had to add non-standard characters, you will no doubt already have guessed that this task involves tinkering with two different sets of control levers.

Keyboard Assignments without X

Most systems load a keyboard definition file shortly after booting. So, if we can find the corresponding command, that should provide us with an answer to this problem. There are at least three different ways to do this:

- Feed your favourite search engine with keywords such as *keyboard*, *Linux*,

assignment and *keytable*.

- Search the boot scripts in the *init.d* directory (normally */etc/rc.d/init.d* or */etc/init.d*) for a command that should include the word *key*.
- Search your *whatis* database for the corresponding command.

The first option is a question of personal preference. Whether the second option will be successful or not depends on your current distribution. Take SuSE 7.2 for example, with its penchant for scripts so complex that normal users have no idea what to look for. You might try the following command:

```
trish@linux:~ > grep keys /etc/init.d/*
[...]

/etc/init.d/kbd: rc_status && retmsg="`loadkeys $KEYMAP 2>&1`"
[...]
```

and quickly conclude that the */etc/init.d/kbd* file (that is "keyboard") is responsible for loading the keytable. But if you just happen to look at this script without really knowing what you are looking for, you will probably feel slightly lost. Depending on which you use, Debian (*/etc/init.d/keymap.sh*), Red Hat (*/etc/init.d/keytable*) or Caldera OpenLinux (*/etc/rc.d/init.d/keytable*), the search results should be far more clear and indicate that the command *loadkeys* is what you are looking for.

You can then search the *whatis* database using the *apropos* command or type the following:

```
trish@linux:~ > man -k keys
loadkeys (1)         - load keyboard translation tables
```

GLOSSARY

Shouting: *Whether in email or IRC dialog, most people intuitively view text passages in CAPITALS as the visual counterpart of shouting.*

Keyboard Layout: *This refers to the way the keys are organized on the keyboard. If you look at an English, Scandinavian, Polish, ... keyboard, you see that the top row of letters begins with the keys [Q], [W], [E], [R], [T], and [Y] (this layout is thus referred to as "qwerty"). French keyboards have have the [A], [Z], [E], [R], [T], and [Y] keys ("azerty") in the top row instead.*

Listing 1: Excerpt from *uk.map*

```
# uk.map
[...]
include "qwerty-layout"
[...]
#       Normal   Shift   AltGr   Ctrl
keycode   1 = Escape    Escape
[...]
keycode 54 = Shift
keycode 56 = Alt
keycode 57 = space
keycode 58 = Caps_Lock
keycode 86 = backslash bar   bar
Control_backslash
keycode 97 = Control
```

Listing 2: Keycodes for Windows Keys

```
trish@linux:~ > showkey
kb mode was XLATE

press any key (program terminates
10s after last keypress)...
keycode  28 release
[Right_Win_Key]
keycode 125 press
keycode 125 release
[Left_Win_Key]
keycode 126 press
keycode 126 release
[Menu Key]
keycode 127 press
keycode 127 release
```

and view the corresponding man page to confirm your suspicions. We find that this is the console command for changing the keyboard assignments, which are stored in the so-called map files under */usr/lib/kbd/keymaps* (SuSE 7.2, Red Hat), */usr/share/keymaps* (Debian) or */usr/share/kbd/keymaps* (Caldera, SuSE 8.0). But don't expect to find the map files in this subdirectory – instead they are nicely organized by the computer architecture (*i386*, *sun*, *mac* etc.) and **keyboard layout** (*qwerty*, *azerty* etc.).

The map files which are stored in these subdirectories (*i386/qwerty/uk.map.gz*) are *gzip*ped text files that can be viewed with the *zless* command (Listing 1 shows an excerpt).

As you would expect, the # character at the start of a line indicates a comment that has no effect on the functionality. The line reading *include "qwerty-layout"* is interesting – instead of defining all from scratch, you can include pre-defined keymaps. Individual key assignments include a so-called *keycode* on the left of the equals sign, and up to four function values on the right: for the key itself or in combination with the Shift, AltGr and Ctrl keys.

How to Stop Your Console Shouting

As we already know the keycode for caps lock (keycode 58) and so can quickly redefine its function. To do so, we create a file called *personal.map*, and add an entry to assign the *Shift* function to the key with the keycode *58*, leaving the other assignments as defined in *uk.map*:

```
include "/usr/share/kbd/keymaps/i386/qwerty/uk.map.gz"
keycode 58 = Shift
```

(You may need to change the path to *uk.map.gz* on your machine.) We can now test the new keyboard assignment on the console by typing:

```
trish@linux:~ > loadkeys personal.map
Loading personal.map
```

As you can see, the keyboard mapping seems to be working perfectly: Caps lock now performs exactly like the Shift key.

Making Use of those Windows Keys

Before we can use the Windows keys on the console, we need to find out their keycodes. Luckily, the *loadkeys* man page contains an example of the corresponding command *showkey* in the *LOAD KERNEL KEYMAP* section. If you now type...

```
trish@linux:~ > showkey
kb mode was RAW
[ if you are trying this under X, it might not work
since the X server is also reading /dev/console ]

KDSKBMODE: This operation is not permitted
```

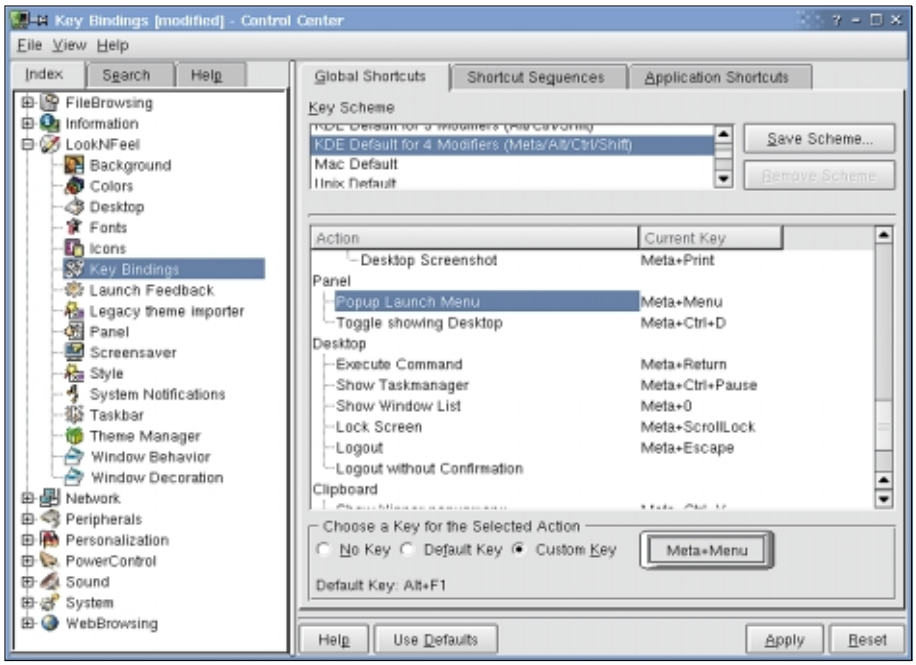

Figure 1: Assigning keyboard shortcuts in KDE 2.1.2

... the command will not run inside an X terminal: This shows that the keyboard assignments for X are independent of the console. If you try the same command on the console, however (Listing 2), then the keycode for the return key (*28*) will be displayed before we can press any other keys. *showkey* not only shows us the keys we press, but also the keys we release. After compiling the required codes, you need to be patient: Pressing Ctrl-C or Ctrl-D will not quit the program, you just have to wait for 10 seconds.

Now it is simply a matter of giving the keys with keycodes 125–127 something useful to do. And why not have the left Windows key do something similar to its original job, i.e. launch the graphic user interface (although this option does not make much sense for people that use the GUI login as X is already running in this case). The right Windows key could then display the date and time and it might be appropriate for the Menu key to display the last 20 commands. You could then choose a command number, add an exclamation mark (possibly edit) and then launch the command.

These are simple tasks that can be launched from a prompt using the *startx*, *date* and *history 20* commands. The real question is, how can you map all these commands to the appropriate keys? Again the *loadkeys* man page is a big help. The section following *LOAD KERNEL STRING TABLE* details how to assign symbols for the non-existant function keys *F100*, *F101*, and *F102* to the corresponding keycodes:

```
keycode 127 = F102
```

You can now assign a string for this key, for example:

```
string F102 = "history 20"
```

After loading the modified keymap, you can simply hit the Menu key to write the string *history 20* in the command line. Just press Enter to confirm, and launch the command. We would prefer not to have to hit the Enter key; in other words, when we hit the Menu key, we want to actually launch the command *history 20*.

The character that the Enter key writes is an end of line. We could solve this issue by including the character in our string. Thinking about outputting text strings to the command line brings the *echo* command to mind, and we can use *\n* ("*n*ewline") to enter a new line:

```
trish@linux:~ > echo -e
?"foo\nbar"
foo
bar
```

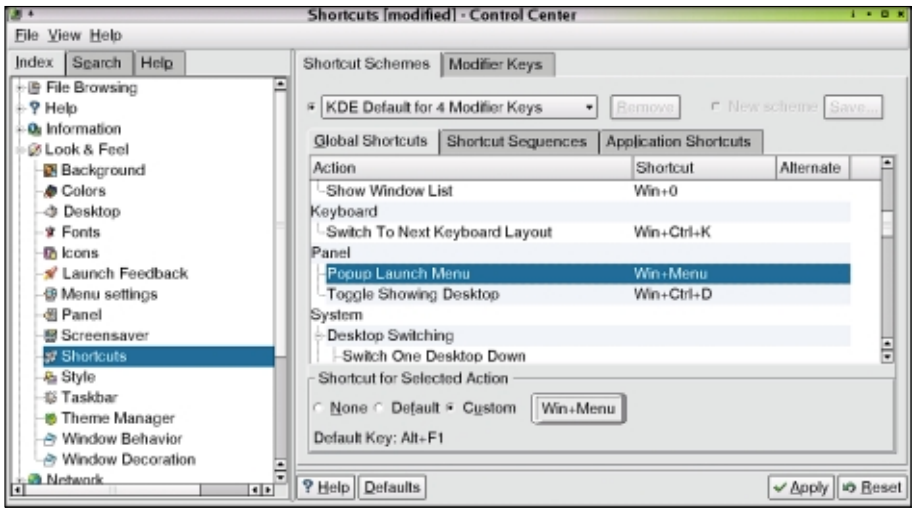

Figure 2: Assigning actions to keyboard shortcuts in KDE 3.0

GLOSSARY

case: *To translate the algorithm "If the content of variable 1 is 'start' or 'reload', do this, and if the variable contains 'stop' do that" to valid Bash syntax, you need the following:*

```
case $1 in
    start|reload) this
    ;;
    stop) that
    ;;
esac
```

Standard Runlevel: *The runlevel that a Linux machine boots to by default is defined by the number in the "id:5:initdefault:" line in /etc/inittab. Runlevel 5, shown in our example, is an operating mode that allows multiple users to work simultaneously (multiuser level), where an X Server is automatically launched and networking is permitted. The machine will shutdown in runlevel 0, and reboot in runlevel 6. Runlevel 1, the single user mode, is reserved for system maintenance – where only basic services are launched and only the user root can perform necessary maintenance tasks. Any other runlevels differ from distribution to distribution and may be individually defined.*

Link: *Another name for an existing file. Symbolic links can be created using the ln -s file secondname syntax.*

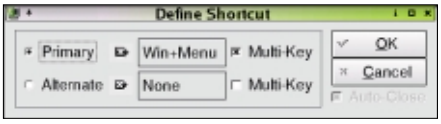

Figure 3: Hotkeys for Actions in KDE 3 are defined here

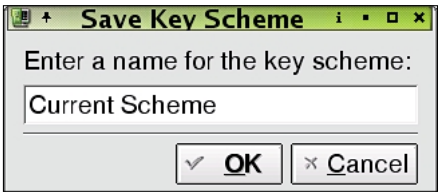

Figure 4: To modify an existing scheme, simply keep the original name

The plan seems to work fine! *string F102 = "history 20\n"* in the keymap (which you will need to reload after editing using *loadkeys*) means that:

```
trish@linux:~ > [Menu]
  926  history 20
[...]
  942  echo -e "foo\nbar"
  943  vi personal.map
  944  loadkeys personal.map
trish@linux:~ > !943
```

really does launch *vi* with the *personal.map* file.

Reassigning Keys on Booting

Since keyboard mappings assigned via *loadkeys* are retained after logging out, you will want to define a single keymap for all users of a system. It also makes sense to load a modified version of a tried and tested map that suits your individual needs (use the examples in Listing 3 for reference) when you boot your machine. To this end *root* zips the the individual *personal.map* using *gzip* and stores it in the systems keymap directory.

However, if you are using SuSE (up to and including 7.3) and modify the *KEYTABLE* entry in */etc/rc.config*, then the distributor has a surprise for you. After performing standard mapping, SuSE loads a few additional key assignments including some for the Windows keys (as you can ascertain by typing *dumpkeys | less*).

As the SuSE init script, */etc/init.d/kbd*, is the only script that performs keyboard assignments, we recommend entering the *loadkeys personal.map.gz* command here. Although this file is somewhat cryptic, it

Listing 3: *personal.map*

```
# Our Code Map is based on
include "/usr/lib/kbd/keymaps/i386/qwerty/uk.map"
# (this path refers to SuSE 7.2 and may need editing)

# ReassignCapsLock to Shift
keycode 58 = Shift
# Right Windows key launches X
# (path for startx may need editing!)
keycode 125 = F100
string F100 = "/usr/X11R6/bin/startx\n"
# Left Windows key outputs the date and time
keycode 126 = F101
string F101 = "date\n"
# Context Menu Key outputs the last 20 commands
# Shift or CapsLock and Menu Key outputs the current directory
# content. Type q to quit.
keycode 127 = F102 F103
string F102 = "history 20\n"
string F103 = "ls -Al | less \n"
```

Box 1: *.Xmodmap* by GUI

Using *xmodmap* to approach the goal of a "generic" keyboard layout for X will certainly not appeal to everyone. If you are lucky, you may find a program called *xkeycaps* pre-installed on your machine that helps add some GUI to the grind. If not, you can download it from [1] or from the CD which accompanies the subscription magazine, use *tar -xzvf xkeycaps-2.46.tar.Z* to unzip it and then *cd xkeycaps-2.46* to change to the directory containing the sources.

Since the archive offers neither a *configure* script, nor a *makefile*, but simply an *Imakefile*, you will find the *README* file quite useful. You can use *xmkmf* to create a *makefile* from the latter, which you can then compile using *make*. The *make* option *-n* (which incidentally has the same meaning in *xmodmap*) does not mean what it says, so we do not need to use *root* privileges to check what would happen if we installed the tool:

```
pjung@chekov:~/software/xkeycaps-2.46$ make -n install
if [ -d /usr/X11R6/bin ]; then set +x; \
else (set -x; mkdir -p /usr/X11R6/bin); fi
install -c -s  xkeycaps /usr/X11R6/bin/xkeycaps
echo "install in . done"
```

If we really want to copy the new *xkeycaps* binary to the */usr/X11R6/bin* directory (we can create, if needed), *root* must issue the install command *make install* (and to additionally install the man page *make install.man*).

Launching the program *xkeycaps &* in an X session displays a virtual keyboard (Figure 7) that can be more closely specified using the "*Select Keyboard*" option. The miniature keyboard image in the *Keyboards* column is provided for comparison. The standard US keyboard is best described by the entry *105 key, wide delete, tall Enter*, and the appropriate layout is defined in the *Layouts:* column as *XFree86; US*. You are then required to confirm your selection by clicking on *ok*, and the miniature keyboard is then modified to match.

If you now right click on a key image, you can select *Edit KeySyms of Key* and then define the required mapping interactively. It is useful to know that the middle key on your mouse can be used for speed scrolling in the dialog box shown in Figure 8. An entry for *KeySym 1* maps a single key; *KeySym 2* assumes that the Shift key is pressed simultaneously, and *KeySym 3* refers to the [AltGr] key. When selecting options you should be aware that you can only output the characters, if the corresponding character sets are available: Characters from *Character Set Latin 1* are safe enough, but Cyrillic or Arabic characters may lead to weird substitutions.

When you have finished your configuration tasks, click on *Write Output* (Figure 7 upper left) to write the complete layout or on (*Changed Keys*) to write just the reassigned key mappings to a file called *~/.xmodmap-machinename* that you can call via *xmodmap* in your X initialization file.

Listing 4: Keyboard Mappings in *XF86Config*

```
# Definition for a PC-Keyboard with 104 keys and British
# layout.

# XFree 3                          # XFree 4
Section "Keyboard"                 Section "InputDevice"
                                       Driver      "keyboard"
                                       Identifier  "Keyboard[0]"
    Protocol "Standard"                Option      "Protocol"     "Standard"
    XkbRules "xfree86"                 Option      "XkbKeyCodes"  "xfree86"
    XkbModel "pc104"                   Option      "XkbModel"     "pc104"
    XkbLayout "gb"                     Option      "XkbLayout"    "gb"
    XkbVariant "nodeadkeys"            Option      "XkbVariant"   "nodeadkeys"
EndSection                         EndSection
```

does adhere to the syntax of a start stop script, where a ***case*** construct defines the action to perform when the script is called using the *start stop* argument, or something similar. In our case, we can just ignore the mess and insert our *loadkeys* line before the ;;, that marks the end of the *start* branch.

Other distributions, like Red Hat, are easier on the system administrator and offer a script named */etc/rc.d/rc.local* that is called right at the end of the boot procedure after loading all other system settings. The end of this file is the ideal place to load an individual keymap.

And of course you can save an init script of your own with the following content:

```
#!/bin/sh
loadkeys personal
```

(the suffix *map.gz* is normally optional and some distributions require you to omit it)in the appropriate *init.d* directory, find the appropriate **standard runlevel** in */etc/inittab* and create a **Link** for it in the appropriate *rc?.d* directory (i.e. *rc2.d* for runlevel 2). The name of the link must start with an *S* (for "start") and a fairly high number. Thus, a link called *S100keymap* that points to the init script will be called after any links starting with *S01* through *S99*.

Keyboard Mappings for KDE

No matter what settings you choose for the console, they will not affect they way the keyboard reacts in an X window session. If you are simply interested in the three Windows keys, the first place to look would be the KDE Control Centre that allows you plenty of leeway to assign your specific actions to keyboard shortcuts via *LookNFeel / Key Bindings* (KDE 2.1.2, see Figure 1) or *Look & Feel / Shortcuts* (KDE 3.0, see Figure 2).

It makes sense to use one of the pre-defined schemes, preferably the one that most closely complies with your way of working. You then mark the action that you want to assign to a hotkey (or the keyboard shortcut). Then select *Custom* (KDE 3.0) or *Custom_Key* (KDE 2.1.2) in the lower frame of the dialog box *Choose a Key for the Shortcut Action*.

KDE 2 does impose a few limits on your creativity. If you are not satisfied with a single key (click on the stylized key button in Figure 1 and press the desired key on your keyboard), you can only use combinations with the Alt, Ctrl and/or Shift keys.

Users who tend to hit the Menu key by mistake will not want to assign the action *Dropdown Menu* to the Menu key, as shown in Figure 1, but can continue to strain their fingers by typing Shift-Ctrl-Alt-Menu to open the *K* Menu.

Now click on *Save scheme...* to assign a name to the new configuration and avoid modifying one of the existing schemes by mistake. Now, click on the *Apply* button to make your new keyboard mappings available for use.

KDE 3 is somewhat more flexible. If you click on one of the stylized keys in the dialog box shown in Figure 2, a slightly cryptic dialog box appears (Figure 3). If you intend to assign a single key for the selected action, then remove the checkmark in *Multi Key*, and you will only be allowed to press a single key on your keyboard (such as the Menu key for example) to perform an action such as *Dropdown Menu*).

If you are defining a combination of keys, you must first select *Multi Key* and then secondly put some thought into the keys you intend to press in order to avoid finger strain. You can use the *Alternate* option to define a second hotkey for the same action.

If you change the keyboard shortcuts for a pre-defined scheme, you will notice that KDE 3 immediately activates and selects the *New Scheme* option. To assign a name to the new scheme, simply click on *Save* (Figure 2). However, you still have to take the roundabout route via *New Scheme* and the *Save* button, even if you want

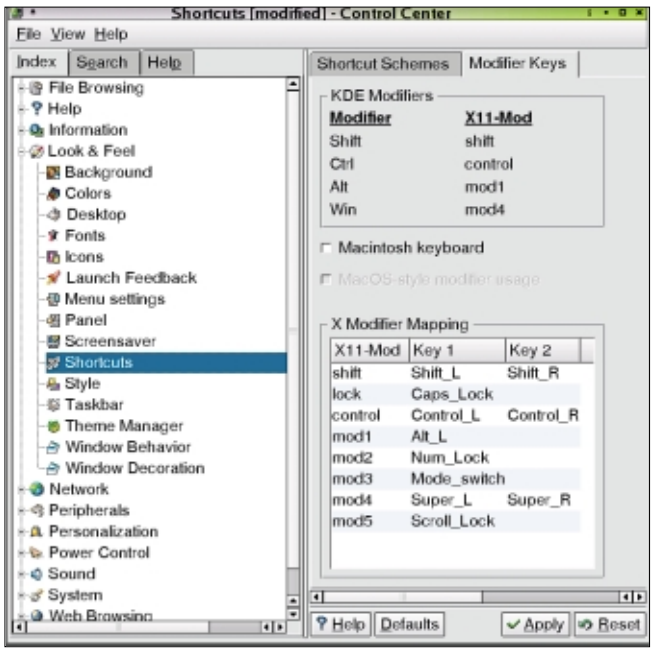

Figure 5: KDE 3 intergrates the xmodmap output into the standard KDE control panel

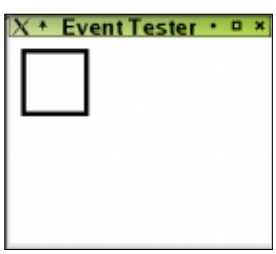

Figure 6: Move the Cursor into the Black Square and hit a Key

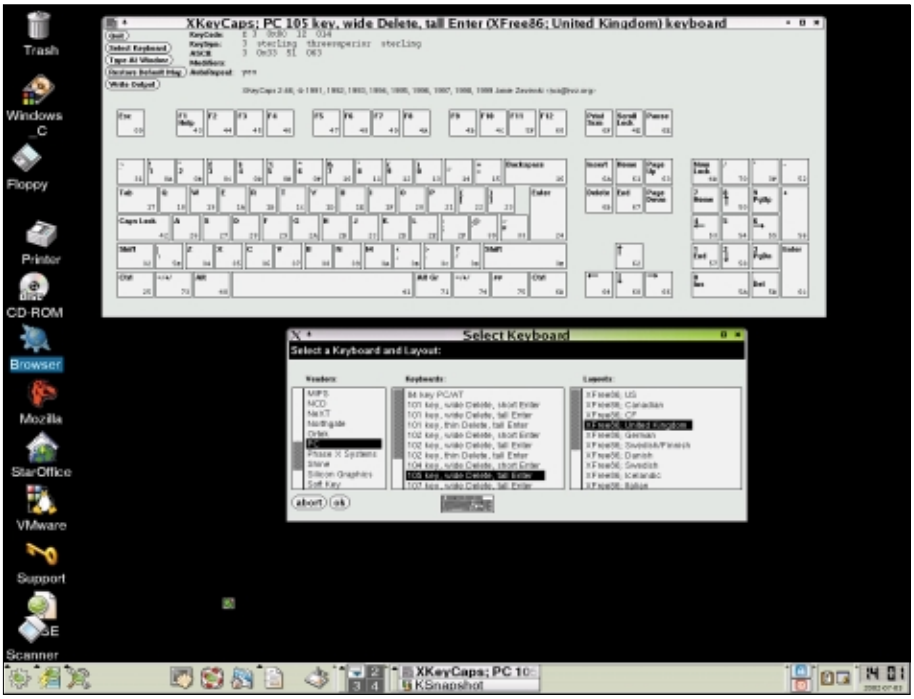

Figure 7: Using xkeycaps to create .Xmodmaps

to modify an existing scheme. In this case you keep the original name in the *Save Key Scheme* dialog box (Figure 4), instead of supplying a new name, as you would do to create a new scheme. Back in the Control Center, simply click the *Apply* button to apply the new or re-defined scheme to KDE 3.

No Caps Lock in X

But all of these modification options will not help you at all, if you use GNOME, XFce or a Standalone Window Manager, as these settings do not apply outside of KDE. Additionally, you cannot re-define the Caps Lock key using the methods we discussed previously.

The solution to this problem is to change the foundation, i.e. the X Window system. The basic configuration file *XF86Config* (normally to be found in the */etc* or */etc/X11* directories and sometimes containing a 4 suffix in XFree 4) is the place to look for the basic settings, such as the keyboard type and the mappings. The syntax for the entries in these files differs between versions 3 and 4 of XFree86 (Listing 4).

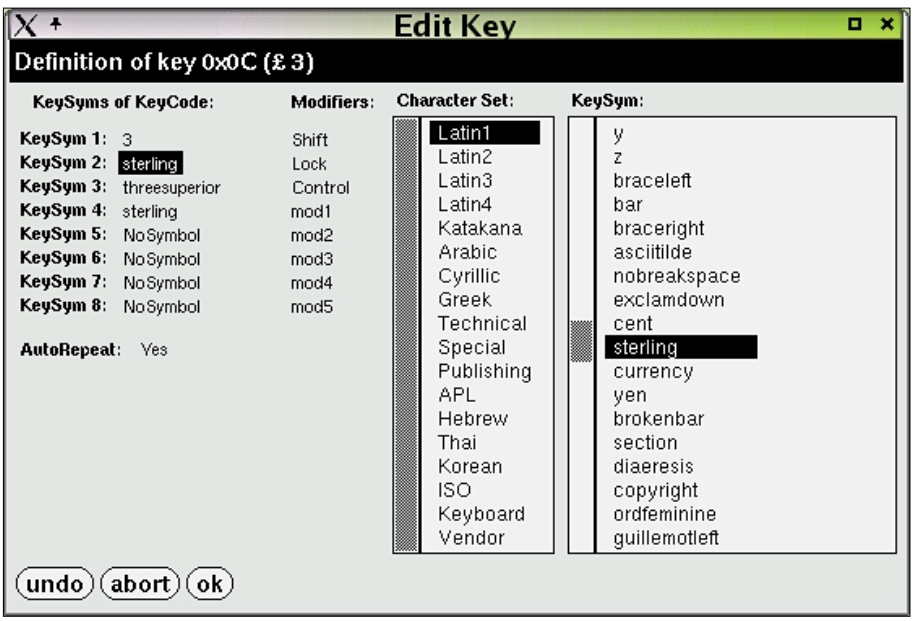

Figure 8: You should restrict your keyboard mappings to displayable characters only

But like so many other global system settings, the keyboard mappings set for all X users are by no means a holy cow. You can use:

```
trish@linux:~ > apropos key | ➥
grep -w X
```

```
xmodmap (1x)   -  utility for ➥
modifying keymaps and pointer ➥
button mappings in X
setxkbmap (1x) - set the ➥
keyboard using the X Keyboard ➥
Extension
CentnerCentner (1x)  - ➥
graphically display and edit ➥
the X keyboard mapping
```

to discover an extremely interesting entry: *xmodmap* is used in the defining of individual keyboard layouts. (Refer to Box 1 for details on *xkeycaps*). Typing *xmodmap* in the command line shows you the keys used as modifiers. You can see the output from xmodmap in the *Shortcuts Modifier Keys* tab under the *X-Modifier Mapping* in the KDE 3 Control Center (Figure 5). To (*shift*) between the two basic settings (i.e. between capitals and non-capitals in the case of letters), you can use the left (*Shift_L*) and right (*Shift_R*) Shift keys. The Caps lock key locks the keyboard in the Shift position (*lock*) and so on.

To counteract this effect we must now remove *Caps_Lock* from the list of *lock* modifiers. After referring to the *xmodmap* man page, we try the following syntax:

```
xmodmap -e "remove lock = ➥
Caps_Lock"
```

And as you will see, *Caps_Lock* is now missing in the output from *xmodmap*:

```
xmodmap:  up to 3 keys per ➥
modifier, (keycodes in ➥
parentheses):
shift       Shift_L (0x32),  ➥
Shift_R (0x3e)
lock
[...]
```

GLOSSARY

grep -w: *This command searches the data stream piped (|) to it, or a file supplied as an argument, for the search key (In our example X), provided it occurs as a single word.*

If we now add *Caps_Lock* to the *shift* list, using *xmodmap -e "add shift = Caps_Lock"*, a quick test shows that we have now tamed the beast, which acts just like a Shift key in any application running on the current X Server. A call to *xmodmap* thus shows:

```
shift  Shift_L (0x32), Shift_R (0x3e),  Caps_Lock (0x42)
```

A New Home for Braces and Brackets

Redefining the Windows keys means the setting our sights a little lower than we have previously seen in KDE. Opening menus, changing virtual desktops and the like belong to the Window Managers realm. We can tell the X Server to output braces with the Windows keys on our keyboards in combination with the AltGr key, whereas with Shift-Windows we can produce left and right brackets.

To do this, we first need the keycodes for the Windows keys. This is done using the simple *xev* program (Figure 6) referred to by the *xmodmap* man page, provided you are not confused by the fact that the program runs on an X Terminal not only shows keypresses but mouse events. Listing 5 shows sample output for pressing (*KeyPress event*) and releasing (*KeyRelease event*) the left Windows key (*keycode 115*) and the equivalent key on the right (*keycode 116*), if the mouse focus does not move outside of the black square in the *xev* window.

Now we only need the symbols for the braces and parentheses. Since these are already assigned to [AltGr-7] ({), [AltGr-8] ([), [AltGr-9] (]) and [AltGr-0] (}), it should be no problem to query *xmodmap* for them. *xmodmap -pke | less* (*-pke* stands for "*p*rint *k*eymap *e*xpression") should do the trick:

```
keycode  16 = 7 ampersand
braceleft seveneighths
keycode  17 = 8 asterisk
bracketleft trademark
keycode  18 = 9 parenleft
bracketright plusminus
keycode  19 = 0 parenright
braceright degree
```

That looks a lot like the map file for the console and shows you how to assign an X task to a keycode. The third entry that following the equals sign appears to be the name of the brace, bracket or parentheses ("brace" – curved brackets, "bracket" – square brackets). This being the case, the following syntax:

```
xmodmap -e "keycode 115 = braceleft bracketleft"
xmodmap -e "keycode 116 = braceright bracketright"
```

should help us achieve our goal.

Saving the Changes

Of course you could place all of these *xmodmap* commands in your personal X startup file *~/.xinitrc* and/or *~/.xsession*. But the syntax shown on the *xmodmap* man page…

```
xmodmap [-options ...] [filename]
```

… suggests that you might like to place your keyboard assignments in a single file (Listing 6). The man page suggests *~/.xmodmaprc* and distributions such as SuSE take care of parsing *.Xmodmap* in your home directory while starting X via *xmodmap*. If you look at the man page, you may note that comments are not indicated by a hash sign #, but by an !.

You can now use an *xmodmap ~/.Xmodmap* entry in *~/xinitrc* (if this is not the default setting for your Linux distribution) to take care of loading the appropriate key maps on starting X with *startx*. If you use a GUI login, you will need to place this command in the *~/.xsession*.

As a KDE user you will notice that the Windows keys are no longer available for KDE. They have been re-defined to output braces and so it is not a good idea to use them as hotkeys. ■

INFO

[1] xkeycaps: http://www.jwz.org/xkeycaps/

Listing 5: *xev* Output of Windows Keys

```
KeyPress event, serial 28, synthetic NO, window 0xe00001,
    root 0x2c, subw 0xe00002, time 1060529679, (30,37), root:(34,57),
    state 0x0, keycode 115 (keysym 0xffe7, Meta_L), same_screen YES,
    XLookupString gives 0 characters:  ""

KeyRelease event, serial 28, synthetic NO, window 0xe00001,
    root 0x2c, subw 0xe00002, time 1060529913, (30,37), root:(34,57),
    state 0x40, keycode 115 (keysym 0xffe7, Meta_L), same_screen YES,
    XLookupString gives 0 characters:  ""

KeyPress event, serial 28, synthetic NO, window 0xe00001,
    root 0x2c, subw 0xe00002, time 1060531499, (30,37), root:(34,57),
    state 0x0, keycode 116 (keysym 0xff20, Multi_key), same_screen YES,
    XLookupString gives 0 characters:  ""

KeyRelease event, serial 28, synthetic NO, window 0xe00001,
    root 0x2c, subw 0xe00002, time 1060531733, (30,37), root:(34,57),
    state 0x0, keycode 116 (keysym 0xff20, Multi_key), same_screen YES,
    XLookupString gives 0 characters:  ""
```

Listing 6: Personal *~/.Xmodmap*

```
! Do not lock on pressing Caps_Lock
remove lock = Caps_Lock
! ... use Caps_Lock as an additional Shift key instead
add shift = Caps_Lock
! left Windows key = {, plus [Shift] = [
keycode 115 = braceleft bracketleft
! right Windows key = }, plus [Shift] = ]
keycode 116 = braceright bracketright
```