X Keyboard Extension

Иван Паскаль pascal@tsu.ru

Настройка XKB.

При старте X-сервера, модуль XKB зачитывает все необходимые данные из текстовых файлов, которые образуют "базу данных" настроек XKB.
Строго говоря, большинство из этих файлов сам модуль XKB не читает. Он вызывает программу **xkbcomp**, которая переводит содержимое этих файлов в двоичный формат, понятный непосредственно модулю XKB. Но для настройки это не так уж важно, поскольку вызов **xkbcomp** происходит автоматически, незаметно для пользователя.
База данных, необходимых XKB, состоит из 5 компонентов

keycodes
: таблицы, которые просто задают символические имена для скан-кодов.
Например
<TLDE>= 49;
<AE01> = 10;

types
: описывает типы клавиш. Тип клавиши определяет - как должно меняться значение, выдаваемое клавишей в зависимости от модификаторов (Control, Shift и т.п.). Так, например, "буквенные" клавиши относятся к типу ALPHABETIC, что означает, что они имеют разное значение в зависимости от состояния Shift и CapsLock. А клавиша [Enter] имеет тип - ONE_LEVEL, что означает, что ее значение всегда одно и то же, независимо от состояния модификаторов.

compat (сокращенное от compability)
: описывает "поведение" модификаторов. В XKB имеется несколько внутренних переменных, которые, в конечном счете, и определяют - какой символ будет генерироваться при нажатии клавиши в конкретной ситуации. Так вот, в compat как-раз описывается - как должны меняться эти переменные при нажатии различных клавиш-модификаторов. В этом же разделе обычно описывается и поведение "лампочек"-индикаторов на клавиатуре.

symbols
: таблицы, в которых для каждого скан-кода (имени скан-кода, определенного в keycodes) перечисляются все значения (symbols), которые должна выдавать клавиша. Естественно, количество различных значений зависит от типа клавиши (котрые описываются в types), а какое именно значение будет выдано в конкретной ситуации, определяется состоянием модификаторов и их "поведением" (которое описывается в compat)

geometry
: описывает "геометрию" клавиатуры - то есть расположение клавиш на клавиатуре. Эти описания нужны не столько самому X-серверу, сколько прикладным программам, которые рисуют изображение клавиатуры.

Все эти компоненты разложены по одноименным директориям в директории **{XROOT}/lib/X11/xkb** (в дальнейшем, я буду обозначать ее **{XKBROOT}**).
Надо сказать, что в каждой такой директории имеется несколько файлов (иногда, довольно много) с разными настройками. Более того, каждый файл внутри себя может содержать несколько блоков (секций, разделов) типа

```
тип_компонента "имя_блока" {.....};
```

Поэтому, для того, чтобы выбрать конкретную настройку, ее обычно указывают в виде
имя_файла(имя_блока)
например,
us(pc104)
В тоже время, обычно один из блоков в файле (не обязательно самый первый) помечается флагом **default**, например

```
      xkb_symbols "pc101" {...};
default xkb_symbols "pc101euro" {...};
      xkb_symbols "pc102" {...};
      xkb_symbols "pc102euro" {...};
```

Это означает, что если указать только имя файла, то будет выбран именно этот блок.

Три способа задания полной конфигурации XKB.

Весь набор компонентов, необходимых для настройки XKB, описывается в файле конфигурации X-сервера в секции **Keyboard**.

Первый способ.

Первый способ задания конфигурации заключается в том, что вы можете указать непосредственно каждый из компонентов, например

```
XkbKeycodes                    "xfree86"
XkbTypes                       "default"
XkbCompat                      "default"
XkbSymbols                     "us(pc104)"
XkbGeometry                    "pc(pc104)"
```

Как легко догадаться, это означает, что

- описание **keycodes** берется из файла **"xfree86"** в директории **{XKBROOT}/keycodes**, причем из файла будет выбран тот блок, который помечен в нем флагом **default**;
- описание **types** берется из файла **"default"** в директории **{XKBROOT}/types**;
- описание **compat** берется из файла **"default"** в директории **{XKBROOT}/compat**;
- описание **symbols** берется из файла **"us"** в директории **{XKBROOT}/symbols**, блок **"pc104"**;
- описание **geometry** берется из файла **"pc"** в директории **{XKBROOT}/geometry**, блок **"pc104"**;

Надо заметить, что в любом блоке (в любых компонентах) может встретиться инструкция
include "имя_файла(имя_блока)" (естественно, имя_блока может отсутствовать)
что, как нетрудно догадаться, означает, что в текущий блок должно быть вставлено другое описание из указанного файла (указанного блока).
Поэтому полное описание может неявно включать в себя данные из многих других файлов, кроме тех, которые вы явно укажете в файле конфигурации X-сервера.

Второй способ.

Второй способ заключается в том, что вы можете указать одной инструкцией сразу полный набор настроек. Такие наборы называются **keymaps** и, также как и обычные компоненты конфигурации XKB, располагаются в отдельных файлах (которые, тоже содержат в себе несколько именованных блоков) в директории **{XKBROOT}/keymap**.
Обычно, каждом блоке **keymap** просто указывается - из каких файлов XKB должен извлечь соответствующие компоненты (хотя, в принципе, там может быть и полное описание всех компонентов), например

```
xkb_keymap "ru"                 {
   xkb_keycodes                 { include "xfree86"
                                };
   xkb_types                                                { include "default"
                                                            };
   xkb_compatibility            { include "default"
                                };
   xkb_symbols                                              { include "en_US(pc105)+ru"
                                };
   xkb_geometry                 { include "pc(pc102)"
                                };
};
```

Обратите внимание, что в одной инструкции **include** может быть указано несколько файлов (блоков) через знак "+". Понятно, что это означает, что должны быть вставлены последовательно все указанные файлы.
Таким образом, в файле конфигурации X-сервера можно вместо пяти компонентов указать сразу один их готовых наборов - **keymap**'ов, например

```
XkbKeymap                      "xfree86(ru)"
```

Кроме того, эти два способа можно комбинировать. Например, если вы выбрали одну из подходящих **keymap**, но вас не устраивает один из компонентов, например **geometry**, то в файле конфигурации можно указать

```
XkbKeymap                    "xfree86(ru)"
XkbGeometry                  "pc(pc104)"
```

При этом, в соответствии с первой инструкцией, все компоненты будут взяты из **keymap "xfree86(ru)"**, а вторая инструкция "перепишет" **geometry**, не затрагивая остальные компоненты (хотя я в этом не уверен :)

Третий способ.

И, наконец, третий способ. Он несколько отличается от предыдущих.
Набор настроек можно указывать не перечислением компонентов, а с помощью задания "правил" (Rules), "модели" (Model), "схемы" (Layout), "варианта" (Variant) и "опций" (Option). В этом наборе только Rules представляют собой некий файл, в котором находится таблица правил - "как выбрать все пять компонентов настроек XKB в зависимости от значений Model, Layout и т.д.".
Все остальные параметры представляют собой просто "ключевые слова" по которым в таблицах "правил" ищутся подходящие файлы настроек (**keycodes, types, compat, symbols** и **geometry**). Естественно, искать будет сам модуль XKB при старте.
Другими словами, Rules определяет некоторую функцию, аргументами которой являются Model, Layout, Variant и Options, а значение, которое возвращает эта функция представляет собой полный набор из компонентов настроек XKB - **keycodes, types, compat, symbols** и **geometry** (или полная **keymap**).
Если рассмотреть какой-нибудь файл rules (эти файлы тоже находятся в **{XKBROOT}** в поддиректории **rules**), то можно заметить, что в них есть строчки, начинающиеся со знака "!". Это "шаблоны", которые описывают - как интерпретировать следующие за ними строчки "правил".
Например, "шаблон"

```
! model = keycodes                    geometry
```

говорит о том, что в следующих строках описываются правила - как по имени "модели" XKB должен выбрать файлы **keycodes** и **geometry**. Например,

```
pc104 = xfree86 pc(pc104)
```

надо понимать так, что если значение аргумента Model - слово **"pc104"**, то **keycodes** надо взять из файла **{XKBROOT}/keycodes/xfree86**, а **geometry** - из файла **{XKBROOT}/geometry/pc**, блок с именем **"pc104"**.
А, например, "шаблон"

```
! model                              layout                              =
                                     symbols
```

говорит о том, что следующие за ним правила описывают - как по названию "модели" и "схемы" выбрать файл (и блок), с нужными **symbols**.
Рассматривая файл **rules** можно заметить, что в нем также есть строчки с wildcards - знаком "*". То есть, в качестве аргументов можно использовать не только те слова, которые встречаются в "правилах". Если программа не найдет указанные вами слова в левой части правила, то все равно подберет какие-нибудь названия для файлов - компонентов настройки.
Например, правило

```
! model                              layout                              =
                                     symbols

....
  *                                  *                                   =
                                     en_US(pc102)+%l%(v)
```

означает, что если указанные вами Model и Layout не были найдены в предыдущих правилах, то в качестве **symbols** надо взять блок **pc102** из файла **en_US**, и добавить к нему блок, название которого задано параметром Variant из файла, название которого задано аргументом Layout. (Таким образом, в некоторых случаях, некоторые из параметров могут являться и названиями файлов или блоков. Но в общем случае это все-таки просто "ключевые слова").
Можно также заметить, что...
Во-первых, не все параметры обязательно задавать для выбора всех компонентов. Обычно достаточно указать Model и Layout (ну и, естественно, Rules), а Variant и Options нужны только в некоторых случаях.
А во-вторых, что

- Model обычно определяет тип "железа" - клавиатуры;
- Layout - язык или, точнее, алфавит, который "навешивается" на кнопки клавиатуры;
- Variant - ну "вариант" он и есть вариант - различные варианты размещения знаков алфавита (заданных Layout'ом);

- Options - обычно меняет "поведение" или "расположение" модификаторов - Control и Group (переключатель групп - это переключатель "языка", например - русский/латинский).

Кстати, заметьте также, что хотя Options как правило состоят из двух слов, разделенных двоеточием, это не означает, что вы можете самостоятельно комбинировать левую и правую часть (двоеточие в данном случае просто делает их более понятными). Так, например, в rules есть правила для значений Options **"grp:toggle"** и **"ctrl:ctrl_aa"**. Но если вы попробуете "сконструировать" что-то типа **"ctrl:toggle"**, ничего хорошего из этого не получится, поскольку такого слова в правилах нет.
Итак, если вы используете третий способ указания конфигурации XKB, то в файле конфигурации X-сервера, надо задать параметры XkbRules, XkbModel, XkbLayout и, если вам нужно что-то не совсем стандартное - XkbVariant и XkbOptions.
Например,

XkbRules	"xfree86"
XkbModel	"pc104"
XkbLayout	"ru"
XkbVariant	""
XkbOptions	"ctrl:ctrl_ac"

означает, что XKB должен

- в соответствии с правилами, описанными в файле **{XKBROOT}/rules/xfree86**, выбрать настройки для
- клавиатуры типа **"pc104"** (104 кнопки),
- русского алфавита (английский алфавит будет включен "по умолчанию"),
- вариант - "стандартный" (то есть, этот параметр можно было не писать)
- и, наконец, дополнительные опции для вашей "раскладки клавиатуры" - **"ctrl:ctrl_aa"**.

Кстати, что означают различные опции, а также - какие "модели" и "схемы" определены в "правилах" (и что они означают) можно посмотреть в файле **xfree86.lst** (или другом файле ***.lst**, если вы выбрали "правила" не **xfree86**), который находится в той же директории, что и файл "правил", то есть - **{XKBROOT}/rules**.

Несколько практических рекомендаций.

Небольшое отступление - "о клавише - переключателе рус/лат".
Когда был написан первый вариант этих рекомендаций, сама раскладка "русской" клавиатуры (**symbols/ru**) включала в себя и "переключатель групп" - рус/лат, "подвешенный" на клавишу **CapsLock**. С одной стороны это было удобно - в простейшем случае достаточно было выбрать "русскую раскладку" и вы автоматически получали и клавишу для переключения "на русский". Но, с другой стороны, это было неудобно для тех, кто предпочитает в качестве переключателя рус/лат другую клавишу (или комбинацию клавиш). Конечно, выбрать другой переключатель не составляло труда, но при этом оставался и переключатель на CapsLock, что многим не нравилось. Для того, чтобы убрать его, надо было "залезть" в соответствующий файл и вручную подправлять соответствующую раскладку.
В конце концов (начиная с версии 3.3.4) сами разработчики XFree убрали этот "переключатель" из "русской раскладки". Но, в связи с этим появились и некторые проблемы - теперь клавишу-переключатель надо явно "заказывать" при конфигурировании **XKB**. Что я и попытся отразить в своих рекомендациях.
Итак...

Самый простой способ - использовать программу для автоматической настройки "иксов".

В XFree86 такая программа называется XF86Setup. Она использует третий метод задания конфигурации XKB. При этом "по умолчанию" используются "правила" (**XkbRules**) - xfree86. Вам нужно будет только выбрать "модель" (**XkbModel**), "схему" (**XkbLayout**) и "способ переключения групп" (переключатель "РУС/ЛАТ").

Кроме того, при желании вы можете изменить "положение клавиши Ctrl". Естественно, в конфигурации это будет выглядеть как соответствующие строчки **XkbOptions**.
Итак. Запустите программу XF86Setup, выберите раздел **Keyboard**. В этом разделе выберите из меню **Model** (тип клавиатуры) и **Layout** (язык). Не забудьте отметить в отдельных списках (в правой части) подходящий "переключатель групп" и, если хотите - "расположение Ctrl".
При выходе из программы она запишет соответствующие строчки в файл конфигурации XFree в секции **Keyboard**.
Но, если вы захотите вписать эти строчки самостоятельно, могу дать еще несколько советов.
Прежде всего, надо сказать, что "ключевыми словами" в этих настройках будут

- **xfree86** - название "архитектуры" X-Window;
- **pc101** (**pc104, pc105** и т.п.) - тип клавиатуры (количество кнопок);
- **ru** - название "раскладки клавиатуры" с русским алфавитом.

(Если у вас другая "архитектура"/клавиатура/алфавит, то подберите названия самостоятельно.)
Итак...

Начнем со второго способа - полная keymap.

В файлах конфигурации есть набор "полных **keymap'ов**" для архитектуры **xfree86**, отличающихся "языком". Все они лежат в файле **xfree86**, а название блока внутри файла отражает название "языка" (точнее - алфавита) - **xfree86(us), xfree86(fr), xfree86(ru)** и т.д. Полный список **keymap'ов** можно посмотреть в файле **{XKBROOT}/keymap.dir**.
Для "руссифицированной" клавиатуры вполне подойдет

```
XkbKeymap                    "xfree86(ru)"
```

К сожалению, после "выкидывания" CapsLock как переключателя рус/лат из русской раскладки (см. замечание в начале этого раздела) получилось так, что "полная **keymap**" для русского языка осталась вообще без какого-либо переключателя "по умолчанию". Но вы можете добавить его "вручную". Для этого придется найти в файле **{XKBROOT}/keymap/xfree86** блок **"ru"**. И дописать в строчку **xkb_symbols** ссылку на описание соответствующего переключателя групп. Для **CapsLock** это будет - **group(caps_toggle)**. То есть, строчка будет выглядеть как

```
xkb_symbols                  { include "en_US(pc105)+ru+group(caps_toggle)"};
```

Если вы выбираете третий способ - через "правила", "модель", "схему" и т.д.

Как я уже говорил,

- название "правил" (**rules**) соответствует "архитектуре" (**xfree86**);
- "модель" (**model**) соответствует типу клавиатуры (**pc101, pc102** и т.п.);
- "схема" (**layout**) отражает "язык" (**ru**).

Поэтому, подходящая конфигурация будет выглядеть примерно так -

```
XkbRules                     "xfree86"
XkbModel                     "pc104"
XkbLayout                    "ru"
```

С помощью **XkbOptions** можно подобрать "поведение" управляющих клавиш. Возможные значения **XkbOptions** и их смысл можно подсмотреть в файле **{XKBROOT}/rules/xfree86.lst**.
Не забудьте, что в последних версиях надо явно выбрать переключатель групп. Для **CapsLock** это будет -

```
XkbOptions                   "grp:caps_toggle"
```

И, наконец, первый способ - описание отдельных компонентов настройки (keycodes, compat, types, symbols, geometry).

Если вы не знаете с чего начать, подсмотрите соответствующий набор в **keymap**. Или попробуйте "вычислить" его "вручную" через **rules/model/layout**. (Подробнее об этом можно прочитать в "Примеры: Где будем экспериментировать?")
Могу посоветовать

- - для **keycodes** выбрать файл **xfree86**;
- - для **types** и **compat** подойдут файлы **default** ("по умолчанию") или **complete** ("полная");
- - **geometry**, скорее всего, **"pc"**, а количество кнопок задается названием блока в файле **pc - pc (pc101), pc(pc102), pc(pc104)**. (Полный список "геометрий" в файле **{XKBROOT}/geometry.dir**.)

А вот на **symbols** обратите особое внимание. Файл **symbols/ru** описывает только "буквенные" клавиши. Если вы укажете только его, то у вас не будут работать все остальные кнопки (включая Enter, Shift/Ctrl/Alt, F1-F12 и т.д.).
Поэтому **symbols** должен состоять по крайней мере из двух файлов - **en_US(pc101)** (в скобках - тип клавиатуры) и, собственно, **ru**.
Полный список symbols в файле **{XKBROOT}/symbols.dir**.
Сюда же надо добавить и описание подходящего "переключателя рус/лат"
Описание всех возможных переключателей групп находится в файле **{XKBROOT}/symbols/group**.
Итак. Для первого метода список может выглядеть так -

```
XkbKeycodes                                   "xfree86"
XkbTypes                                      "complete"
XkbCompat                                     "complete"
XkbSymbols                  "en_US(pc101)+ru+group(caps_toggle)"
XkbGeometry                                   "pc(pc101)"
```

Если вам хочется дополнительные изменения "поведения" управляющих клавиш (то, что в третьем методе задается **XkbOptions**), то подсмотрите подходящую "добавку" в **{XKBROOT}/rules/xfree86.lst** и "приплюсуйте" ее в строчку **XkbSymbols**. Например,

```
XkbSymbols          "en_US(pc101)+ru+group(shift_toggle)+ctrl(ctrl_ac)"
```

Еще один способ описания конфигурации XKB.

Кроме опций в общем файле конфмгурации X-сервера (**XF86Config**) XKB может иметь свой отдельный файл конфигурации. Правда он в XFree86 не применяется и (увы) не описан. В то же время, с помощью этого файла можно

- задать конфигурацию XKB для каждого дисплея по отдельности (если на машине запущено несколько X серверов на разных дисплеях)
- настроить намного больше внутренних параметров XKB, чем это позволяет **XF86Config**.

Собственный файл конфигурации XKB (назовем его так) должен находится в директории **{XRoot}/lib/X11/xkb** и называться **X**<цифра>**-config.keyboard**
где <цифра> - номер дисплея (обычно - **X0-config.keyboard**)

Формат этого файла.

Прежде всего надо заметить, что все инструкции в этом файле выглядят как операторы присваивания языка C.
параметр_XKB = выражение ;
Если оператор один в строке, то знак '**;**' в конце не обязателен. В любом месте строки могут находится комментарии, которые должны начинаться с '#' или '//'.
"Выражение" представляет собой

- числовое значение (например - величина задержки или номер группы);

- логическое значение - **on** или **off**;

- строка в кавычках - **""** (например - имя файла/блока содержащего описание компонентов конфигурации XKB)

- название модификатора, "управляющего флага" и т.п.

Если в выражении требуется указать несколько модификаторов (флагов) одновременно, они перечисляются через '+'.
Кроме того, в файле могут встретиться некоторые разновидности "присваивания". Если инструкция определяет, например, начальные значения для "набора управляющих флагов", в котором некоторые флаги и так уже установлены по умолчанию, то возможны следующие операции

- убрать из набора флаги, указанные в инструкции
  набор_флагов **-=** флаг1 + флаг2 + ...

- добавить в набор указанные флаги
  набор_флагов **+=** флаг1 + флаг2 + ...

- и наконец, полностью заменить набор флагов на тот, что вы укажете
  набор_флагов **=** флаг1 + флаг2 + ...

Итак. В этом файле можно задать следующие параметры

Компоненты конфигурации.

rules = "..."
model = "..."
layout = "..."
variant = "..."
options = "..."

keymap = "..."

keycodes = "..."
geometry = "..."
types = "..."
compat = "..."
symbols = "..." (или **symbolstouse = "..."**)
Описывают те же параметры, что и в главном конфигурационном файле X сервера (**XF86Config**). Напомню, что собственный файл конфигурации XKB может быть составлен для каждого дисплея по отдельности. Поэтому, если компоненты конфигурации уже описаны в **XF86Config**, то здесь имеет смысл указывать только те компоненты (или **rules/model/layout**/etc.), которые для данного дисплея отличаются от общих. Естественно, все эти параметры, указанные в "дополнительном файле конфигурации" имеют больший приоритет и переписывают соответствующие значения из файла конфигурации X сервера.

Начальное значения для набора модификаторов.

modifiers [= | **-=** | **+=** **]** модификатор1 + модификатор2 + ...
"Модификатор" - название одного из "реальных" модификаторов - **shift, lock, control** (или **ctrl**), **mod1, mod2, mod3, mod4, mod5**.
Как я уже говорил, здесь вместо присваивания могут использоваться операции '**-=**' - убрать модификатор(ы), '+=' - добавить модификатор(ы), '=' - заменить набор модификаторов на указанные в этой инструкции.
(Надо заметить, что по умолчанию при старте X сервера этот набор модификаторов пустой. Поэтому, имеют смысл только операции '+=' и '='. Причем разницы между ними нет.)

Начальное значение для набора "управляющих флагов".

controls [= | **-=** | **+=** **]** флаг1 + флаг2 + ...
Как и в предыдущей инструкции, операция может быть убрать/добавить/заменить (**'-='/'+='/'='**).

"Флаги" могут быть

repeat (или **repeatkeys**)
разрешить автоповтор клавиш
(по умолчанию он и так разрешен, поэтому имеет смысл только его убрать
controls -= repeat);
mousekeys
включить эмуляцию мыши;
mousekeysaccel
включить режим "ускорения" для эмуляции движения курсора мыши
overlay1
overlay2
включить соответствующие "перекрытия".
ignoregrouplock
игнорировать "текущую группу" в режиме GrabKey
audiblebell
включить (выключить) звуковой сигнал.
(Напомню, что XKB может вместо звукового сигнала посылать bell-event'ы для проигрывания звуков или мелодий "звуковой приставкой". Если такая "приставка" у вас есть, то обычную "пищалку" можно и выключить.
accessxkeys
slowkeys
bouncekeys
stickykeys
accessxtimeout
accessxfeedback
включение различных режимов AccessX (для людей с "ограниченными физическими возможностями")

Список модификаторов, которые игнорируются в режиме GrabKey

ignorelockmods [-= | **+=** | **=** **]** модификатор1 + модификатор2 + ...
(или **ignorelockmodifiers** ...)

Список "внутренних" модификаторов

internalmods [-= | **+=** | **=** **]** модификатор1 + модификатор2 + ...
(или **internalmodifiers** ...)
Это модификаторы, которые используются только внутри X сервера (для выбора "действия" клавиши, если таковое есть) и не сообщаются в клавиатурных event'ах, посылаемых приложениям.

"Метод выравнивания" групп.

groups = [**wrap** | **clamp** | число]
(или **outofrangegroups** ...)
Задает "метод выравнивания групп". Напомню, что метод может быть **wrap**, **clamp** или **redirect**. В последнем случае нужен дополнительный параметр - куда "редирект", если номер группы выходит за границы диапазона.
Так вот, метод **redirect** задается в виде
groups = номер_группы

Параметры "пищалки".

bell = число (или **bellvolume** = число)
bellpitch = число
bellduration = число
click = число (или **clickvolume** = число)
Все эти инструкции устанавливают параметры "пищалки" (**bell**) и "клика" клавиатуры (обычно это более короткий сигнал, чем bell).
Инструкции **bell** и **click** могут также иметь вид
bell = [**on** | **off**]

click = [**on** | **off**]
что (как нетрудно догадаться) просто означают включить/выключить эти сигналы. Если сигнал просто включен и не указано его "volume", то оно считается - 100.

Различные задержки (таймауты).

repeatdelay = число
: задержка (в миллисекундах) между нажатием клавиши и началом автоповтора;

repeatinterval = число
: интервал (в миллисекундах) между автоповторами;

slowkeysdelay = число
: в режиме **slowkeys** клавиша считается нажатой, если она удерживается некоторый промежуток времени, чтобы избежать срабатывания при случайном "задевании" клавиши. Этот параметр и определяет указанную задержку.

debouncedelay = число
: в режиме **bouncekeys** клавиша после нажатия/отпускания некоторое время не реагирует на повторные нажатия/отпускания, чтобы избежать ложных срабатываний при неаккуратном нажатии клавиши. Этот параметр и определяет указанную задержку.

accessxtimeout = число (или **axtimeout** = число)
: промежуток времени (в секундах) после которого режим AccessX автоматически отключается. Имеет смысл только если установлен соответствующий флаг (**controls += ... accessxtimeout ...**);

Параметры "ускорения" курсора мыши.

mousekeysdelay = число
: задержка (в миллисекундах) до начала автоповтора клавиши эмулирующей движение "мышиного" курсора (то же самое, что и repeatdelay для обычных клавиш);

mousekeysinterval = число
: интервал автоповтора (то же, что и repeatinterval);

mousekeysmaxspeed = число
: максимальная скорость движения курсора в пикселах за один event (точнее - если в соответствующем "действии" эмуляции движения курсора - **MovePtr** уже задан параметр "смещение за один event" больше единицы, то максимальная скорость будет "смещение"*maxspeed);

mousekeystimetomax = число
: через сколько повторов курсор достигает максимальной скорости

mousekeyscurve = число (в диапазоне -1000:1000)
: "степень кривизны" кривой ускорения (в "режиме ускорения" скорость курсора нарастает от начального "смещения" до максимального не линейно, а пропорционально **X^(1 + curve/1000)**. Если **mousekeyscurve** = 0, то зависимость линейная)

Включение/выключение отдельных событий дополнительной звуковой индикации в AccessX

accessxtimeoutctrlson [-= | += | =] событие1 + событие2 + ...
(или **axtctrlson** ...)
accessxtimeoutctrlsoff [-= | += | =] событие1 + событие2 + ...
(или **axtctrlsoff** ...)
где "событие" - **slowkeyspress, slowkeysaccept, feature, slowwarn, indicator, stickykeys, slowkeysrelease, slowkeysreject, bouncekeysreject, dumbbell.**
Подробнее об этих событиях можно прочитать в документации (**XKBlib**), поставляемой в "исходниках" **XFree86**.

Немного о "внутренностях" XKB.

Основные термины - коды и символы.

Основное назначение модуля XKB — преобразовывать скан-коды нажимаемых клавиш в коды символов. Обычно в документации XKB скан-коды называются **keycodes**, а коды символов - просто "символы" (**symbols**).

Естественно, "символы" - это не обязательно коды "печатных" символов (букв, цифр, знаков препинания и т. п.). Это могут быть также "управляющие" коды (Esc, Enter, Backspace и т. п.) или коды, которые никак не отображаются, но влияют на внутренне состояние самого XKB (переключение РУС/ЛАТ, Control, Shift, Alt и т.п.) и, следовательно, на то, какой символ будет выбран при нажатии обычных "буквенных" клавиш.

Две части XKB и "проблема совместимости".

Прежде всего следует сказать несколько слов о том, как происходит преобразование скан-кодов в символы при работе X Window System без модуля XKB.
Сам X-сервер переводом скан-кода в код символа не занимается. При нажатии или отпускании кнопки клавиатурный модуль X-сервера посылает сообщение о событии (**event**) прикладной программе.
В этом сообщении указывается только скан-код нажатой кнопки и "состояние клавиатуры" - набор битовых флагов, который отражает состояние клавиш-модификаторов (**Shift, Control, Alt, CapsLock** и т. п.).
Клиентская программа должна сама решить, какой символ соответствует скан-коду при таком сочетании битов-модификаторов.
Естественно, для этого программа может воспользоваться таблицей, хранящейся в X-сервере, в которой для каждого скан-кода перечислены возможные символы. Обычно програма при старте запрашивает эту таблицу у сервера. (Таблицу на сервере можно менять с помощью утилиты **xmodmap**).
Разумеется, при создании программ никто не пишет каждый раз процедуру для интерпретации скан-кодов. Для этих целей существуют специальные подпрограммы в библиотеке Xlib.
Конечно, Xlib используется не только для этого. Это библиотека низкоуровневых процедур для работы с X-сервером. Даже если программа пользуется высокоуровневыми библиотеками-"тулкитами" (Xaw, Motif, Qt, gtk и т. п.), библиотека Xlib служит "базовой" для этих тулкитов.
Итак, процедуры из Xlib, зная скан-код и состояние клавиатуры, в соответствии с таблицей символов, полученной от сервера при старте, выбирают подходящий символ.
Модуль XKB точно так же сообщает программе только скан-код и свое "состояние". В отличие от "старого" модуля (обычно говорят - "**core protocol**", будем его называть для краткости - "core-модуль"), XKB имеет более сложную таблицу символов, другой набор модификаторов и некоторые дополнительные параметры "состояния клавиатуры" (обо всех этих отличиях мы поговорим по ходу рассмотрения "внутренностей" XKB).
Для полноценной работы с XKB библиотека Xlib должна содержать процедуры преобразования скан-кодов, "знакомые с Xkb".
Естественно, "иксы", у которых X-сервер "укомплектован" модулем XKB, имеют и соответствующую библиотеку Xlib.
Таким образом, XKB фактически делится на две части - модуль, встроенный в X-сервер, и набор подпрограмм, входящих в библиотеку Xlib.
Однако же, никто не запрещает использовать программы, которые были собраны со старой библиотекой Xlib, "не подозревающей" о существовании XKB: при этом и возникает "проблема совместимости". Модуль XKB должен уметь общаться как со "своей" Xlib, так и со старой, работающей в соответствии с core protocol.

Естественно, "общение" не ограничивается только передачей сообщений о нажатии/отпускании клавиш. Процедуры Xlib могут обращаться к X-серверу с различными запросами (например, взять таблицу символов) и командами (например, поменять в этой таблице расположение символов).
На все эти запросы и команды модуль XKB должен реагировать так, чтобы даже "старая" Xlib могла работать правильно (насколько это возможно).

Таблица символов

Что же собой представляет таблица символов, связывающая символы со скан-кодами и состоянием модификаторов? Рассмотрим сначала "традиционную" таблицу символов, которая используется в core protocol.
Как и во многих других клавиатурных модулях, ее можно представить как обычную двумерную таблицу, где каждая строка соответствует скан-коду, а столбцы - модификаторам или комбинации модификаторов.
Прежде всего надо заметить, что в "сообщении о нажатии/отпускании клавиши" под биты-модификаторы отводится один байт (октет) и соответственно, существует восемь битов-модификаторов. Первые три называются - **Shift, Lock, Control**, остальные "безымянные" - **Mod1, Mod2, Mod3, Mod4, Mod5**.
Замечу, что хотя названия первых трех модификаторов явно намекают на то, каким клавишам они должны соответствовать, на самом деле они могут быть "привязаны" (то есть, менять свое значение при нажатии/отпускании клавиши) к любым другим кнопкам.
В данном случае нам важно, что восемь модификаторов могут составить 256 различных комбинаций. Поэтому теоретически таблица символов может содержать до 256 столбцов. В то же время, в core protocol'e строго определены только первые четыре колонки, в смысле - каким комбинациям модификаторов они соответствуют. Легко догадаться, что в этих комбинациях используются только два модификатора - **Shift** и

Mode_switch.
(Вы можете заметить, что среди модификаторов нет бита с именем **Mode_switch**. Совершенно верно. Его роль выполняет один из "безымянных", модификаторов - **Mod1-Mod5**. А **Mode_switch** - это название одного из служебных символов. Когда приложение запрашивает у сервера таблицу символов, оно также выясняет, какой из безымянных модификаторов связан с этим символом, и в дальнейшем по состоянию этого модификатора догадывается о том, нажал ли пользователь клавишу, соответствующую символу **Mode_switch**.)
Итак. Первые четыре колонки в таблице соответствуют "состояниям" -

"никаких модификаторов"	**Shift**	**Mode_switch**	**Mode_switch+Shift**

В терминах core protocol можно сказать, что состояние **Mode_switch** выбирает одну из двух групп (не надо путать их с XKB группами, о которых мы поговорим позднее), а состояние **Shift** - одну из двух колонок внутри группы.
Как вы скорее всего знаете, разные группы обычно используются для разных языков (алфавитов), а **Shift** выбирает строчные или прописные буквы в пределах одного алфавита.

	Group 1		**Group 2**	
		Shift	**Mode_switch**	**Mode_switch+Shift**
...				
keycode 38	a	A	Cyrillic_ef	Cyrillic_EF
keycode 39	s	S	Cyrillic_yeru	Cyrillic_YERU
keycode 40	d	D	Cyrillic_ve	Cyrillic_VE
...				

Заметьте также, что ни **Lock**, ни **Control** в выборе символа из таблицы не участвуют. Если эти модификаторы активны, то отдельные подпрограммки Xlib после выбора символа из таблицы делают соответствующие преобразования этих символов.
Вернемся к XKB. Как видите, недостаток "традиционной" таблицы - ее "негибкость". Хотя колонок может быть до 256, "стандартно" обрабатываются только первые четыре, причем их зависимость от состояния модификаторов жестко "зашита" в алгоритмах Xlib.
Поэтому одно из основных усовершенствований, которые были внесены при разработке XKB - большая гибкость в построении таблицы.

- Во-первых, в XKB колонки не связаны жестко с конкретными модификаторами. В одном из файлов конфигурации XKB описывается зависимость номера колонки от произвольного набора модификаторов. Естественно, эти зависимости можно изменять и добавлять простым редактированием соответствующего файла. (Подробнее о них смотри ниже: Вычисление "уровня" (shift level). Типы клавиш.)

- Во-вторых, в одной и той же "раскладке клавиатуры" разные клавиши могут соответствовать разному количеству символов в зависимости от модификаторов. Например,

  - клавиша **Enter** вообще не зависит от модификаторов, то есть в соответствующей строке имеет смысл только одна колонка;

  - клавиша с символами '1' и '!' содержит две колонки, выбор конкретной зависит только от **Shift**,

  - а клавиша с символами '-','_'и '=' может иметь три колонки, выбор которых зависит от двух модификаторов - **Shift** и **Mod1**

- При таком подходе понятие "группа" сильно меняет свой смысл. Мы уже не можем определять группу как "пачку" колонок, поскольку для разных клавиш размер этой "пачки" будет разным. Вообще-то, при той гибкости в связывании колонок с модификаторами, которую дает XKB, можно было бы вообще забыть о группах или просто говорить, что группа —это группа колонок, объединенных модификатором (тем же **Mode_switch**). Но...

Делить таблицу на группы удобнее. Во-первых, так легче провести границу между разными алфавитами (если, конечно, таковые используются в раскладке). Заметьте, что в некоторых случаях хотелось бы иметь не два разных алфавита, а больше. Причем раскладки для разных алфавитов могли бы составлять разные люди, независимо друг от друга
Во-вторых, чем больше модификаторов участвуют в выборе колонок, тем сложнее и запутаннее становятся соответствующие описания зависимостей.
Поэтому, авторы XKB использовали радикальный подход - в одной и той же раскладке может быть несколько (до четырех) отдельных таблиц. Вот эти таблицы и называются группами XKB (или просто **groups**).
Вообще-то, правильнее сказать не "несколько двумерных таблиц в одной раскладке", а - у каждого скан-кода (**keycode**) может быть до четырех "однострочных" таблиц - групп.
Потому, что ...

- Назначение некоторых клавиш (той же **Enter**) не меняется в разных алфавитах, поэтому для них не требуется деление таблицы на несколько отдельных групп.

- Даже в пределах одной группы (алфавита) разные клавиши могут иметь разное количество колонок. То есть, "ширина" таблицы меняется не только "от группы к группе" но и "от скан-кода к скан-коду". Поэтому удобнее описывать каждую отдельную группу для каждого отдельного скан-кода.

- И, наконец, с каждым скан-кодом связаны еще некоторые параметры (см. ниже), которые не зависят от номера группы. Поэтому, если бы мы описывали один и тот же скан-код в разных таблицах-группах, эта информация могла бы оказаться противоречивой.

Итак.

- Для каждого скан-кода (**keycode**) в XKB "раскладке" хранится несколько однострочных таблиц символов, которые называются "группами".

- Каждая такая таблица может делиться на несколько колонок - "уровней" (**shift level**)

- Какая из таблиц-строчек актуальна в данный момент, определяется текущим номером группы (или просто - **group**). Текущий номер группы хранится в X-сервере и сообщается приложению в событии о нажатии/отпускании клавиши вместе со скан-кодом и набором модификаторов.

- Выбор нужной колонки (**shift level**) определяется состоянием модификаторов.

- Разные **keycode** могут иметь разное количество групп.

- В разных группах даже одного и того же **keycode** может быть разное количество **shift level**.

keycode	**номер группы**	**количество level'ов**		
36	1	одна колонка	**Enter**	

38	1	две колонки	a	A		
	2	две колонки	Cyrillic_ef	Cyrillic_EF		
	3	две колонки	Greek_alpha	Greek_ALPHA		
21	1	две колонки	+	=		
	2	четыре колонки	+	=	\	;
...						

Наконец, надо сказать, что групп может быть от одной до четырех, а уровней - до 64.

Таблица действий.

Кроме "таблицы символов" к скан-коду может быть "привязана" аналогичная "таблица действий" (**actions**).
Эта таблица также делится на подтаблицы (группы) и колонки (уровни).
В отличие от таблицы символов, которая используется приложением (X-сервер ее только хранит, чтобы сообщить каждому вновь стартующему приложению), таблица действий используется самим сервером.
В ее ячейках располагаются вызовы внутренних процедур XKB, которые меняют его состояние - текущую группу, состояние модификаторов и внутренних флагов XKB.
Точнее - действия, выполняемые **actions**, не ограничиваются изменениями состояния XKB. Они также используются для

- эмуляции событий мыши (перемещения указателя и нажатия mouse buttons)
- генерации специальных events для приложений
- переключения экранов
- выключения X-сервера
- и т.п.

Немного подробнее о всех возможных **actions** написано в разделе Описание действий.
Если на клавише для группы и позиции в группе определено действие, то для этой же группы и позиции в группе для той же клавиши должен быть определен символ (обычно служебный).
Заметим, что в core-модуле понятия "действие" вообще нет.

Состояние XKB: номер группы.

Текуший номер группы хранится в "таблице состояния" XKB.
Точнее, внутри XKB имеются три ячейки

- **locked group** - некий аналог нажатия клавиши **CapsLock**, но для групп
- **latched group** - аналог действия клавиши **Shift**, но, опять же, для групп
- **base group** - базовое смещение номера группы

Обычно содержимое этих переменных меняется с помощью действий (**actions**), которые связываются с подходящими клавишами (их скан-кодами).

Наконец, существует понятие "фактической" (**effective**) или "действующей" группы. Именно ее значение используется для выбора подтаблицы-группы в таблице символов и таблице действий. Это значение не хранится, а каждый раз вычисляется в виде суммы трех вышеуказанных переменных.

Метод выравнивания номера группы.

Естественно, при таком суммировании может получиться число, большее чем количество заданных групп. Чтобы получить из него какой-нибудь допустимый номер группы, XKB может использовать три разных метода:

- **Wrap** ("заворачивание") - получившееся число делится на количество групп и берется остаток от деления. Этот метод используется по умолчанию.

- **Clamp** ("удержание в границах") - если получилось число большее чем номер последней группы, то берется этот самый "номер последней группы", если же получилось число меньше, чем номер самой первой группы, то берется первая группа.

- **Redirect** ("перенаправление" или, точнее - "замена") - если используется этот метод, то в "состоянии XKB" должен быть определен еще один дополнительный параметр - номер группы "куда перенаправить" (или - "чем заменить"). Если получившееся число выходит за пределы допустимых значений для групп, оно просто заменяется этой самой "чем заменить". Кстати, если и этот номер не уложится в допустимые границы, то XKB подставит первую группу.

Модификаторы.

Кроме переменных, содержащих номера групп, "состояние XKB" определяется "флажками" - модификаторами. Эти битовые флаги меняются при нажатии клавиш **Shift, CapsLock, Alt, Control** и т. п. и, естественно, влияют на выбор подходящего символа из таблиц.
Как я уже упоминал, "core protocol" (протокол "общения" между "традиционным" клавиатурным модулем и Xlib) тоже имеет "набор модификаторов" (который и сообщается прикладной программе при нажатии кнопок). Эти модификаторы (8 бит) называются **Shift, Lock, Control, Mod1-Mod5**. В зависимости от их состояния Xlib выбирает подходящий символ и может выполнять дополнительные действия — делать из обычных символов управляющие, менять прописные/строчные буквы и т. п.
Кроме того, программа может сама (помимо Xlib) по-своему интерпретировать эти модификаторы и менять свое поведение.
Поэтому, хотя XKB имеет больше модификаторов и его поведение (действия, выполняемые при определенной комбинации модификаторов) можно гибко перестраивать (как при старте, так и в процессе работы), модуль XKB вынужден эмулировать "классический" набор модификаторов, специально для старых клиентских программ (и старой Xlib).
Итак. Модификаторы core protocol'a в XKB называются "реальными" (**real**) модификаторами и их названия такие же, как в core protocol.
Кроме того, XKB имеет еще 16 своих внутренних модификаторов, которые называются "виртуальными" (**virtual**) модификаторами.
К сожалению, "проблема совместимости" заключается еще и в том, что в формате "сообщения о клавише" для модификаторов отведено только восемь бит. Поэтому XKB при всем желании не может сообщить приложению все свои шестнадцать виртуальных модификаторов и вынужден в любом случае отображать их в "реальные" (даже для приложений "знакомых с XKB"). Единственным утешением является то, что можно сколько угодно виртуальных модификаторов отобразить в один реальный и это отображение легко перенастраивается.
Итак. Эти модификаторы выполняют несколько функций

- по их состоянию "вычисляется" **shift level** для выбора нужного символа и/или "действия";

- по их состоянию "вычисляется" состояние индикаторов (см. ниже);

- и, кроме того, они могут влиять на выполнение некоторых других действий XKB (вычисление номера группы и т.п.).

Состояние XKB: модификаторы.

Так же как и номер группы, набор битов-модификаторов внутри XKB существует в трех экземплярах (в трех переменных)

- **locked modifiers**
- **latched modifiers**
- **base modifiers**

Содержимое этих переменных (так же как и "групповых") может меняться с помощью действий (**actions**). Как и для номеров групп, для модификаторов существует "фактический" набор модификаторов, который определяется как логическая сумма (побитная операция **ИЛИ**) трех указанных переменных. Соответственно, никаких дополнительных "выравниваний", как для значения номера группы, не требуется.

Вычисление "уровня" (shift level). Типы клавиш.

В отличие от номера группы, уровень не запоминается в состоянии XKB, а вычисляется из состояния модификаторов. При этом для различных клавиш зависимость уровня от модификаторов может различаться. Для того чтобы обеспечить такую гибкость, в XKB существут понятие "тип клавиши".
В каждом описании типа клавиши задается некая функция - какие модификаторы принимать к рассмотрению и какой уровень должен получится при определенном сочетании модификаторов.
Соответственно, у каждой клавиши (скан-кода) в каждой под-таблице (группе) указывается свой "идентификатор типа".
Естественно, при нажатии клавиши XKB (точнее, процедуры Xlib) определяет, к какому типу она относится и по описанию типа и текущему состоянию модификаторов вычисляет, из какого уровня следует выбрать символ для этого скан-кода.
Надо отметить, что хотя описания типов и принадлежность каждой клавиши к определенному типу можно гибко менять, в XKB по умолчанию задан основной набор типов и для всех клавиш расписаны идентификаторы типов. Поэтому обычно нет необходимости задавать эти параметры в файлах конфигурации XKB.

Какая еще информация хранится для каждого скан-кода.

Кроме таблицы символов (и, возможно, "действий") с каждым скан-кодом связаны еще несколько переменных

- метод "выравнивания" номера группы, специфичный для данной клавиши;
- "поведение" клавиши;
- набор "исключений";
- реальный модификатор (modmap) и виртуальный модификатор (vmodmap), связанные с этой клавишей.

Метод "выравнивания" номера группы.

Поскольку разные клавиши могут иметь разное количество групп, может возникнуть ситуация, когда вычисленный и "выровненный" эффективный номер группы все-таки не попадает в дапазон допустимых групп для данной клавиши.
В этом случае эффективный номер группы может дополнительно "выравниваться", специально для данной клавиши.
Возможные методы точно такие же, как и "глобальные". По умолчанию используется метод Wrap.

"Поведение" клавиши.

Эта переменная состоит из двух частей - набора флагов и дополнительного аргумента. Обычно аргумент не нужен. Но некоторые флаги подразумевают дполнительный числовой аргумент, а смысл этого аргумента зависит от флага.
Флаги определяют

- нужен ли "автоповтор" для этой клавиши (Строго говоря, флаг "автоповтор" хранится в другом месте, а не в переменной "поведение". Но в данном случае это не так уж важно.)

- обычная эта клавиша или "залипающая" (то есть, по первому нажатию клавша "залипает" и отпускается при повторном нажатии). При этом "залипание" клавиши может отрабатываться самим "железом" или эмулироваться модулем XKB.

- принадлежность этой клавиши к какой-нибудь "радио-группе". При этом дополнительный аргумент указывает номер такой "радио-группы".

- принадлежит ли эта клавиша к "перекрывающейся группе" (**overlay**). Таких групп может быть всего две. Какая группа в данный момент активна, определяется "управляющими флагами" **overlay1** и **overlay2**. Соответственно в "поведении клавиши" есть два флага которые и определяют принадлежность клавши к той или другой группе, а дополнительный аргумент указывает - какой скан-код должна эмулировать клавиша, если включен "режим перекрытия".

Набор "исключений".

Это битовая маска, которая указывает - какая информация, связанная с клавишей, "задана точно" (**explicit**) и не должна изменяться в некоторых случаях. Дело в том, что в core protocol определены команды, с помощью которых программы могут менять "раскладку клавиатуры" внутри клавиатурного модуля X-сервера. Естественно, эти комады меняют только "привязку" символов, поскольку другие "свойства" скан-кодов в core protocol'e не определены.
Для того, чтобы XKB мог при этом поменять и "привязку" других "свойств", в нем предусмотрен специальный механизм - "интерпретации" символов (об этом см.ниже).
Так вот. Набор "исключений" может защитить информацию, связанную с конкретным скан-кодом, именно от таких косвенных изменений.
Если прикладная программа будет пользоваться соответствующим запросами XKB модуля (а не core protocol), такой защиты не требуется.
Итак, с помощью этой маски можно запретить

- изменение типа клавиши (количество уровней) для каждой группы (можно установить запрет для каждой группы поотдельности);

- изменения, которые могут произойти при примениии "интерпретации", при этом можно запретить -

    - вообще все изменения, вызываемые "интерпретацией";

- изменение флагов "автоповтор" и "залипание";
- изменение набора виртуальных модификаторов.

Реальный и виртуальный модификаторы.

Обратите внимание, что с каждым скан-кодом связаны две разные переменные (**modmap** и **vmodmap**), одна для реальных модификаторов, другая - для виртуальных.
Набор реальных модификаторов служит для эмуляции "традиционного" набора модификаторов (в соответствии с core protocol), а набор виртуальных модификаторов может служить аргументами для "действий", связанных с этой клавишей.
Надо заметить, что изменения набора виртуальных модификаторов (**base, locked** и **latched**) делается с помощью соответствующих "действий". Естественно, одним из аргуметов этого "действия" должен быть модификатор (или несколько модификаторов), которые нужно установить/сбросить в соответствующих "переменных состояния". Так вот, при описании "действия", модификаторы можно указать явно, а можно просто сослаться на **vmodmap** (то есть - "установить витруальные модификаторы, привязанные к этой клавише").
А вот изменение модификатора в "эмулируемом наборе модификаторов" происходит автоматически при нажатии/отпускании клавиши. Естественно, устанавливаются/сбрасываются именно те модификаторы, которые указаны в наборе реальных модификаторов (**modma**p).
Кроме того, эти два набора служат для установления соответствия между реальными и виртуальными модификаторами. Надо заметить, что виртуальный модификатор не имеет никакого смысла, если не связан с каким-нибудь реальным модификатором.

Состояние XKB: Набор управляющих флагов (XKB Controls).

Кроме номера группы и набора модификаторов, в XKB существует аналогичный набор для "управляющих флагов". Правда, в отличии от номера группы и набора модификаторов, которые "размазаны" по трем переменным (**base, locked, latched**), набор управляющих флагов только один.
В "управляющие флаги" входят все те "флажки", которые не попали в "набор модификаторов". Они не сообщаются приложению, а используются XKB для переключения его внутренних режимов.
Эти флаги управляют

- включением режима перекрытия для части клавиатуры;
- включением режима эмуляции мыши.
- включением различных режимов подсистемы AccessX.
- и некоторыми другими сойствами XKB.

"Управляющие флаги" (как и номера групп и модификаторы) меняются соответствущими действиями, привязанными к подходящим клавишам.

Индикаторы.

Как известно, на клавиатуре кроме "кнопок" имеется несколько "лампочек"- индикаторов. Управление этими индикаторами тоже входит в обязанности XKB.
В XKB может существовать до 32 индикаторов.
Естественно, на клавиатуре отображаются только первые 3-4 из них. Эти индикаторы называются в XKB "физическими", а остальные - "виртуальными". Состояние виртуальных модификаторов может быть считано из XKB и отображаться на экране специальными программами (**xkbvleds**, **mxkbledpanel**).

Каждый индикатор может быть связан с компонентом "состояниеа XKB" (модификатором, номером группы, "управляющим флагом") и, соответственно, состояние индикатора будет автоматически отражать состояние своего "компонента состояния XKB".
Причем, в XKB существуют специальные запросы, которыми прикладная программа может управлять состянием какого-нибудь индикатора ("зажигать"/"гасить" его). Заметьте, речь идет о том, что программа может просто включать/выключать "лампочку", а не менять состояние клавиатуры, которое "отслеживается" этой "лампочкой".
Поэтому, с каждым индикатором связан набор флагов, который определяет

- можно ли прикладной программе управлять самим этим индикатором (или для его включения/выключения обязательно надо изменить "состояние клавиатуры";

- "привязан" ли этот индикатор к "состоянию клавиатуры" или он включается/выключается только прикладными программами;

- и, наконец, будет ли такое включение/выключение иметь "обратную связь". То есть, можно настроить индикаторы так, что включение/выключение "лампочки" прикладной программой будет вызывать автоматически соответствующие изменения "состояния клавиатуры" (модификаторов, номера группы, управляющих флагов).

Таблица "совместимости".

Как уже упоминалось, XKB вынужден решать "проблему совместимости" с программами, которые не подозревают о существовании XKB и обращаются с запросами к "core-модулю" X-сервера.
Естественно, модуль XKB может обработать эти запросы, но основная проблема, возникающая при этом, состоит в том, что в модуле XKB вводятся некоторые новые понятия (новые относительно core protocol'a), которые никак не отражены в "традиционных" запросах к клавиатурному модулю.

- Во-первых, в XKB больше модификаторов и их поведение может гибко перестраиваться. Поэтому, как уже упоминалось, для совместимости модулю XKB приходится поддерживать восемь "традиционных" модификаторов и осуществлять отображение (преобразование) своих модификаторов в "традиционные".

  Более того. XKB вынужден поддерживать "традиционный" формат "сообщения о нажатии/отпускании" клавиши, в котором передается "состояние" клавиатурного модуля, состоящее из набора реальных модификаторов. Даже если модуль XKB общается с XKB-совместимой Xlib, он вынужден, для передачи ей информации о состоянии своих виртуальных модификаторов, "превращать" их в реальные.
  (Поэтому, кстати, виртуальный модификатор не оказывает никакого эффекта, если он не "отображен" в какой-нибудь реальный).

- Во-вторых, в core protocol "номер группы" имеет другой смысл. Как уже говорилось - групп всего две, переключаются они символом **Mode_Switch** и в "состоянии клавиатуры" переключение на "дополнительную" группу отображается одним из модификаторов **Mod1-Mod5** - тем, который связан с символом **Mode_Switch**.

  Таким образом, в "традиционном" протоколе номер группы отображается одним из модификаторов - "основная группа/альтернативная группа", а в модуле XKB, в "состоянии клавиатуры" явно существует двухбитное поле - "номер группы" (которое, к тому же, не "перекрывается" с набором модификаторов).

Кроме того, в "традиционном" модуле клавиатуры каждая группа состоит ровно из двух "уровней". Хотя при этом допускается, что каждому скан-коду может быть сопоставлен одномерный массив символов длиной до 256. Первые четыре ячейки в нем соответствуют "двум группам, в каждой по два уровня". Остальные ячейки "клиентская" программа может выбирать сама, в зависимости от состояния модификаторов Mod1-Mod5.

Поэтому, для совместимости модуль XKB, при общении со "старыми" программами, вынужден ...

- Во-первых, "размазывать" свои подтаблицы-группы в одну строчку "традиционной" таблицы символов (честно говоря, я так и не понял общий алгоритм этого действия, кроме тривиального случая, когда скан-коду соответствует две группы, каждая из двух уровней).

- Во-вторых, отображать свой "номер группы" в состояние какого-нибудь "реального" модификатора.

- Наконец, в core-модуле отсутствуют "действия" - основной механизм, обеспечивающий "гибкость поведения" XKB. В связи с этим возникает проблема - "старая" программа может поменять "привязку" символов к скан-кодам, но, поскольку она не подозревает о том, что к скан-кодам могут быть "привязаны" также и "действия", может возникнуть ситуация, когда такая программа переместить, например, символ "переключатель РУС/ЛАТ" на другую клавишу, а соответствующее "действие" (которое в XKB реально осуществляет это переключение) останется на старом месте.

  Для решения этой проблемы в XKB хранится таблица "интерпретаций" (**interpretation**) управляющих символов. Эта таблица связывает коды символов и вызовы соответствующих "действий" (**actions**). Естественно, в этой таблице присутствуют только специальные "управляющие" символы (**Caps_Lock, Shift, Num_Lock, "переключатель РУС/ЛАТ"** и т.п.).

  Кроме самих символов и "действий" в таблице интерпретаций хранится также дополнительная информация - список "реальных модификаторов" и "критерий соответствия" ("любой из модификаторов", "все указанные модификаторы", "только указанные модификаторы" и т.п.).

  Каждый раз при изменении "привязки" символа (учтите, что это изменение может происходить при старте X-сервера, как часть "начальной настройки" сервера) модуль XKB проверяет - присутствует ли в этот символ в "таблице интерпретаций". И если символ там есть, то XKB, "присвоив" этот символ требуему скан-коду, добавляет этому скан-коду и соответствующее "действие". Дополнительная информация (модификаторы и "критерий соответствия") тоже может использоваться при "поиске" подходящего места для "действия".

  Кроме того, применение "интерпретации" может поменять некоторую другую информацию, связанную со скан-кодом - флаги, отвечающие за "автоповтор" и "залипание" клавиши и "список виртуальных модификаторов".

  Напомню, что у каждого скан-кода может быть указан набор "исключений", который защитит информацию, связанную со скан-кодом от применения "интерпретации".

  Надо отметить, что помещать все "действия" в "таблицу интерпретаций", а не присваивать их непосредственно скан-кодам - является "хорошим тоном". Поскольку в "таблице интерпретаций" "действие" связывается с символом, а не со скан-кодом, и, к тому же, "перенос" "действий" из "таблицы интерпретаций" в соответствуюшие таблицы "действий" скан-кодов XKB выполняет

каждый раз при загрузке конфигурации (в том числе и при старте X-сервера), модуль XKB сам найдет подходящее место для соответствующего "действия" и, таким образом, корректно "свяжет" символы и действия (в таблицах, назначенных скан-кодам).

Еще несколько определений.

Радио-группы. (Radio Groups)

Модуль XKB позволяет объединять несколько клавиш в "радио-группу". То есть, клавиши одной радиогруппы являются взаимозависимыми. При нажатии одно из клавиш группы все остальные из этой группы "отжимаются".
Естественно, нажатая клавиша остается "залипшей", пока не будет нажата другая клавиша из этой группы.
Принадлежность конкретной клавиши к некоторой радио-группе задается в поле "поведение клавиши". Там же можно определить дополнительное свойство - можно ли "отжать" все клавиши радио-группы. Обычное определение радио-группы подразумевает, что одна из клавиш группы должна быть нажата, а чтобы отжать ее, надо нажать другую клавишу в группе (естественно, теперь она будет "залипшей"). Если задано свойство "могут быть отжаты все", то "залипшую" клавишу группы можно "отжать" просто повторным нажатием (при этом никакая другая клавиша группы не окажется "залипшей").
В XKB можно объявить до 127 радио-групп (хотя это может зависить от конкретной реализации XKB).

Перекрытия (Overlay).

В модуле XKB можно для некоторых клавиш (группы клавиш) задать набор "альтернативных" скан-кодов. То есть, при нажатии такой клавиши клавиатура выдает некоторый скан-код, который и должен быть интерпретирован XKB (выбрать подходящий символ), но, если включен режим замены скан-кода на "альтернативный", то XKB прежде всего заменяет этот скан-код соответствующим "альтернативным", а потом уже новый скан-код пытается интерпретировать.
Такая группа называется "перекрытием" (**overlay**). Их может быть всего две. Принадлежность конкретной клавиши к конкретной группе-"перекрытию" и "альтернативный" скан-код задаются в поле "поведение клавиши".
Это свойство используется для клавиатур, у которых очень маленький набор клавиш (например в "лаптопах"). Например, на такой клавиатуре может полностью отсутствовать "дополнительная цифровая клавиатура" (**keypad**). Поскольку некоторые приложения могут потребовать, чтобы какие-то действия в них вызывались нажатием кнопок именно на **keypad**, модуль XKB может эмулировать эту дополнительную клавиатуру, используя, например, "цифровые" кнопки на основной клавиатуре.

AccessX.Дополнительные услуги для людей с ограниченными физическими возможностями.

Имеются ввиду люди у которых подвижность рук (пальцев) ограничена или они вынуждены пользоваться какими-либо маханическими приспособлениями для работы с клавиатурой.
Проблемы, которые могут возникнуть при этом:

- Невозможно нажимать одновременно несколько клавиш (например **Shift**+"буква"). Для решения этой проблемы в XKB предусмотрен режим **StickyKeys** ("прилипающие клавиши"). В этом режиме вместо одновременного нажатия нескольких клавиш можно нажимать их последовательно. Например, вместо **Shift+Control+C** можно последовательно нажать клавиши **Shift**, **Control** и **C**. Обратие внимание - не все клавиши должны "слипаться". Правило здесь простое - если клавиша является модификатором и без других кнопок ее нажатие не имеет смысла, то после нажатия такой клавиши XKB ждет нажатия других клавиш, чтобы "слепить" в одно событие. Если и последующие кнопки являются модификаторами (как **Control** после **Shift** в нашем примере), то XKB остается в состянии ожидания. И только после нажатия обычной (буквенной клавиши) генерируется **event** и XKB возвращается в нормальное состяние.
  Кроме того заметьте, что этот режим сам может вызвать проблему если клавиша модификатор нажата по ошибке. Любое следующее нажатие вызовет не то действие, которое хотелось бы. Поэтому в режиме **StickyKeys** предусмотрена специальная задержка (это настраиваемый параметр) в течении которой XKB ждет следующую клавишу. Если за это время другого нажатия не последовало, то клавиша отпускается.

- "Дребезг" клавиши при нажатии. Для решения этой проблемы в XKB предусмотрен режим **BounceKeys** ("прыгающие клавиши"). В этом режиме после первого нажатия на клавишу XKB на некоторое время становится "нечувствительным", то есть не обращает внимание на изменение состояния клавиши и таким образом игнорирует лишние нажатия. Естественно интервал времени "нечувствительности" можно менять.

- Ошибочное "задевание" клавиш. Имеется ввиду, что человек при перемещении руки от одной клавиши к другой может случайно задеть несколько клавшь, которые нажимать не собирался. Для решения этой проблемы предусмотрен режим **SlowKeys** ("медленные клавиши"). Смысл его в том, что клавиша считается реально нажатой если она остается в таком состоянии некоторое время (опять же это время можно настраивать). Другими словами, если клавишу нажать и быстро отпустить, то XKB игнорирут такое скоротечное нажатие.

- Невозможность управлять "мышью". Для решения этой проблемы в XKB предусморена возможность эмулировать события мыши с помощью клавиатуры. Подробнее об этом режиме мы поговорим ниже.

Все эти режимы осуществляются частью XKB модуля, которая называется AccessX. Все режимы можно включать и выключать по отдельности изменяя состяние одноименных "управляющих флагов XKB" (XKB Controls). Кроме того всю подсистему AcceessX можно выключить одним из флагов (**AccessXKeys**). (Естественно, чтобы заработал хотя бы один из режимов нужно включить и AccessX, и нужный режим). Обратите внимание, что само включение режимов AccessX может быть проблемой. (Представьте себе, что человеку нужен режим **StickyKeys**, а для его активизации нужно нажать сложную комбинация из нескольких клавиш). Поэтому для включения некоторых режимов используются специальные действия:

- Если нажать клавишу **Shift** и удерживать ее нажатой в течении 8 секунд, то включается режим **SlowKeys**.

- Если нажать тот же **Shift** пять раз подряд, то включется режим **StickyKeys**.

Правда для этого должен быть включен сам AccessX. Его можно включить

- либо указав соответствующий ключ при запуске X-сервера,

- либо установив нужный флаг в "файле конфигурации XKB",

- или включить тот же флаг с помощью клавиши с нужным **action** (по умолчанию такой клавиши нет, но ее можно предусмотреть в раскладке клавиатуры).

С другой стороны, если один из режимов AccessX включен, а компьютер используется разными пользователями, то человку не имеющему ограничений он может наоборот мешать. Поэтому в XKB предусмотрено автоматическое отключение режимов AccessX, если в течении некоторого времени клавиатура никак не использовалась. Кроме того, режим **StickyKeys** (эмуляция одновременного нажатия через последовательное) автоматически выключиться, если вы нажмете несколько клавиш действительно одновременно.
Саму возможность автоматического выключения AccessX по таймауту можно (как и другие режимы) включить/выключить соответствующим флагом (**AccessXTimeout**) в XKB Controls.
И наконец - в AccessX предусмотрен дополнительный режим звуковой индикации всех происходящих событий - начало и окончание всевозможных таймаутов, а также включение и выключение LED'ов. Для того, чтобы лучше ориентироваться в этх событиях (особенно если включено несколько разных режимов AccessX) XKB старается озвучивать их звуками разной длительности и тона (насколько позволяет "железо").
Этот режим, как и другие может быть включен/выключен соответсвующим флагом (**AccessXFeedback**) в XKB Controls. Кроме того, можно по отдельности включить/выключить озвучивание отдельных событий.

Эмуляция "мыши"

XKB может эмулировать события мыши с помощью клавиатуры. То есть, его можно настроить так, что при нажатии определенных клавиш X-сервер будет посылать приложению сообщения (**events**) не о нажатии/отжатии кнопок, а о перемещении мыши и нажатии кнопок на мышке.
Делается это с помощью соответсвующих действий (**actions**) - перемещение мыши, нажатие кнопки мыши, выбор кнопки "по умолчанию".
Рассмотрим их немного подробнее.

- В **action** "перемещение" можно задавать сразу обе координаты (или их приращения). Координаты могут быть как абсолютные (это означает, что при выполнении такого действия указатель мыши должен встать в заданную точку экрана), так и относительные - собственно перемещение на несколько точек в нужном направлении. Обычно для удобства делают несколько разных **action** для одного направления, но с разными приращениями - на одну точку, на десять точек и т.д. Эти **action** можно привязать к одной кнопке, но в разных уровнях. Получится, что при обычном нажатии кнопки указатель смещается, на одну точку, а нажатие той же кнопки с Shift'ом перемещение будет на несколько точек при одном нажатии.

- В **action** "нажатии кнопки мыши" можно кроме самой кнопки указать и число повторов. То есть, одиночное нажатие клавиши может эмулировать "двойной клик" (или тройной, четверной и т.п.).

- Кроме того, существует разновидность этих действий, которая не просто "нажимает кнопку мыши" по нажатии клавиши, а оставляет ее нажатой после отпускания клавши (как все **Lock** клавши). То есть, вы можете с помощью такого действия "прижать кнопку мыши", переместить указатель и повторным нажатием клавиши "отпустить кнопку мыши". Естественно, это выглядит так, как будто вы двигали реальную мышь с нажатой кнопкой.

- И наконец, обратите внимание, что кнопок мыши может быть до пяти, а с учетом "многократных кликов" и Lock'ирующих нажатий может получится, что под "мышинные кнопки" придется занять довольно много клавиш. Для того, чтобы сократить это количество в XKB предусмотрено, что одну из кнопок можно объявить "кнопкой по умолчанию" (**default button**). Для такого назначения и его изменения предусмотрены специальные действия, в которых можно опять же указать, как конкретную кнопку (по номеру), так и относительное изменение номера "кнопки по умолчания". Комбинируя эти действия, можно одной клавишей перебирать циклически кнопки (делая их по очереди "умолчательными"), а в действиях "нажимающих" кнопки указать, что они относятся к той кнопке, которая сейчас выбрана как **default button**. (Проблема только в том, как не запутаться и не забыть - какая кнопка сейчас выбрана :-)

Кроме того, существует два режима (**mode**) перемещения мыши - простой и "с ускорением" (**accelerated**). При одиночном нажатии клавиши разница между ними незаметна. А вот если вы держите клавишу нажатой и работает автоповтор, тот в обычном режиме на каждый автоповтор указатель смещается на одно и то же количество точек (заданое в аргументах действия). А в **accelerated mode** указатель начинает ускоряться, то есть с каждым новым автоповтором размер одиночного шага растет.
Весь процесс ускорения описывается несколькими дополнительными параметрами, которые хранятся во внутренних переменных XKB (являются частью "состояния" XKB). (Надо заметить, что режим ускорения имеет свои параметры автоповтора - задержку между реальным нажатием клавиши и первым повтором и интервал между повторорами.) Итак, эти параметры (числовые) описывают

- **delay** - интервал (в миллисекундах) от нажатия клавиши (эмулирующей движение мыши) до первого автоповтора.

- **interval** - интервал (в миллисекундах) между повторами

- **maxspeed** - максимальная скорость (в точках на один повтор)

- **timetomax** - количество повторов, за которое достигается максимальная скорость движения. Как видите, здесь нет в явном виде величины ускорения. XKB сам вычисляет его из начальной величины смещения (задается в конкретном действии), **maxspeed** и **timetomax**.

- **curve** - "степень кривизны" ускорения. Дело в том, что скорость может расти не линейно. В общем виде приращение скорости пропорционально функции **X^(1 + curve/1000)**. То есть, при curve=0 скорость растет линейно, а при других значениях (от -1000 до +1000) с некоторой "кривизной".

По умолчанию используется именно режим с ускорением.
Включение и выключение эмуляции мыши и выбор режима движения выбирается двумя управляющими флагами XKB (XKB Controls) - **MouseKeys** и **MouseKeysAccel**.
Надо заметить, что по умолчанию все нужные действия описаны и привязаны к кнопкам "дополнительной цифровой клавиатуры" (**NUMPAD**). Для включения и выключения режима эмуляции мыши используется комбинация **Shift+NumLock**.

Расширенные возможности "пищалки" (Bell).

Вообще-то, эта часть XKB имеет малое отношение к клавиатуре.
Более того, для пользователей компьютеров, в которых клавиатура, дисплей и "спикер" - независимые устройства, такое совмещение может показаться очень странным. Но поскольку, в некоторых архитектурах динамик "пищалки" находится в клавиатуре и издает звук (**key_click**) при каждом нажатии клавиши, в "иксах" считается, что управление "пищалкой" - дело клавиатурного модуля.
Надо отметить, что управление "пищалкой" заложено еще в традиционном (core) модуле клавиатуры. С помощью стандартных запросов к X-серверу, приложение может регулировать параметры **key_click**'а (высоту тона, длительность и громкость), а также вызвать этот **click** при необходимости.
В XKB разработчики пошли еще дальше и предусмотрели возможность вместо простого "писка" проигрывать "музыкальные фрагменты". Естественно, обеспечить такое "музыкальное сопровождение" для клавиатурного модуля слишком сложная задача. Во-первых нужна "база данных" различных звуков (хотя бы просто в виде набора аудио-файлов), а во-вторых музыка может воспроизводится различными "железками" (звуковыми картами), каждая из которых требует "особого подхода" (драйвера).
Поэтому, по замыслу разработчиков, выбором конкретного звука и его воспроизведением должна заниматься отдельная программа (назовем ее - "музыкальная приставка"). А модуль XKB вместо "писка" просто генерирует специальное сообщение (**event**), которое так же как и обычные **event**'ы X-сервера могут быть доставлены любой программе. "Музыкальная приставка" только должна при старте сообщить X-серверу, что ее интересует определенная разновидность **event**'ов (в данном случае - **bell-event**'ы от модуля XKB).
Естественно, если такая "музыкальная приставка" существует, ее возможности не ограничиваются воспроизведением "писков" разной частоты и длительности. Она может проигрывать множество разных музыкальных фрагментов из своей "базы данных".
Поэтому, в сообщениях XKB указывается не "параметры писка", а просто некое "имя звука" (**bell name**). Естественно, "музыкальная приставка" должна иметь какой-то конфигурационный файл, в котором для каждого "имени звука" назначен свой музыкальный фрагмент.
Соответственно, теперь любое приложение с помощью соответствующих запросов к X-серверу (теперь уже не в core-формате, а в формате XKB), может заказать не просто "писк" с определенным тоном и длительностью, а любой звук, указав его имя (**bell name**).
При этом XKB собственно является просто "ретранслятором". Он даже не анализирует правильность "имени звука", а просто, получив запрос от любого приложения, передает его в виде своего **event**'а "музыкальной приставке".
Естественно, в такой схеме XKB кажется даже лишним. Приложение могло бы и само общаться с "музыкальной приставкой". Но не забудьте, что сама идеология "иксов" допускает, что приложение, X-сервер и "музыкальная приставка" могут быть запущены на различных машинах. Поэтому, нормальное приложение должно было бы, кроме соединения с X-сервером, найти еще и "музыкальную приставку" и установить с ней контакт.
В схеме с XKB, любое приложение отправляет свои запросы тому же самому X-серверу (с которым ему и так приходится общаться), а уж администратор сервера определяет - кто и как будет обслуживать эти запросы. Кроме того, при этом вводится некоторый стандарт на общение между приложением и "звуковой приставкой" (разве что, кроме самих "имен звуков").
Надо заметить также, что XKB не только ретранслирует запросы, но и сам может "заказать звук" при некоторых изменениях своего внутреннего состояния (состояния индикаторов, флагов и т.п.).

Общие замечания о языке файлов конфигурации XKB.

Для описания конфигурации **XKB** используется язык с синтаксисом похожим на синтаксис языка **C**. Поэтому, формат числовых и строковых констант, встречающихся в этих файлах, обычно соответствут формату констант языка C.
Так строковые константы представляют собой набор символов ограниченный "двойными кавычками" (например - "Num Lock"). Причем, внутри строк могут встречаться "спецсимволы", которые обозначаются так же как в языке C (\r, \n, \t, \v, \b, \f или через восьмеричный код - \0**).
Числовые константы могут записываться как в десятичном формате (например - 123), так и в шестнадцатеричном (0x123, 0xff) и восьмеричном (033).
Главное отличие от языка C в том, что во всех словах маленькие и большие буквы НЕ различаются (case insensitive). То есть, например - **SETMODS, SetMods, setMods** и **setmods** означают одно и то же.

Объявления в файлах конфигурации, общие для всех типов файлов.

Каждый файл или блок конфигурации состоит из группы объявлений (деклараций, инструкций, определений ?).
Какие именно объявления допустимы в конкретном файле (и как они выглядят) зависит от "Типа Файла". Однако, есть некоторые слова, которые могут встречаться в файлах (блоках) любых типов.

Инструкция include.

Понятно - что она означает - включить в этот блок описания из другого файла (блока). Причем, аргументом этой инструкции может быть не просто имя файла и блока, а, вообще говоря, просто строка, например
```
include "en_US(pc104)+ru"
```
Естественно, все "слова", соединенные плюсами, должны представлять собой имена существующих файлов и блоков в них, причем того же типа, что и блок, в котором встретилась эта инструкция.

Способ добавления.

Перед каждым отдельным объявлением может стоять дополнительное слово, которое определяет "способ добавления" или "замещения" (**merge mode**).
Оно определяет - как должна поступить программа, считывающая конфигурацию из файла, если такое объявление уже встречалось и новое объявление "конфликтует" с предыдущим. Например, это может быть определение кодов символов (блок типа **xkb_symbols**) для скан-кода, который уже был определен ранее или описание "поведения" модификатора (тип - **xkb_compat**) для модификатора, который уже описан. Итак, способ добавления может быть

- **augment** - в случае конфликта оставить старое определение, новое игнорировать
- **override** - заменить строе определение на новое
- **replace** - обычно то же самое, что и **override**, но в файле **xkb_symbols** означает немного другое. Поскольку, каждому скан-коду соответствует целый массив возможных значений (какое именно выбирается - зависит от модификаторов), различные определения одного и того же скан-кода могут описывать значения только для части ячеек этого массива. Так вот, если новое описание определяет только некоторые ячейки и "способ добавления" - **override**, то переписаны будут только те ячейки, которые присутствуют в новом определении, остальные останутся без изменений. А вот если "способ добавления" - **replace**, то все старое описание уничтожается (включая те ячейки, которые отсутствуют в новом описании) и заменяется на новое, даже если оно не полное.
- **alternate** - допустимо только для файлов типа **xkb_keycodes**, означает, что, если новое имя конфликтует с уже существующим, то просто рассматривать его как "alias скан-кода", то есть - просто другое имя для того же числового значения скан-кода.

Еще раз об include.

Кстати, возвращаясь к инструкции **include**. Если объявления из "включаемого" файла "перекрываются" с уже имеющимися, то "по умолчанию" считается, что они добавляются в режиме **override** (если внутри файла они не помечены другими "способами добавления"). То же самое "умолчание" действует, если в строке-аргументе **include** есть дополнительные файлы, через знак '+'.
А вот если вместо плюса стоит знак '|', то это означает, что следующий файл должен добавляться в режиме **augment** (опять же, некоторые объявления внутри него могут иметь свои "способы добавления").
Кроме того, "способ добавления" (кроме **altrenate**) может использоваться вместо инструкции include. То есть, вместо
```
include "group(toggle)"
```
можно использовать, например, инструкцию
```
replace "group(toggle)"
```
Нетрудно догадаться, что это означает, что все инструкции из файла (блока) **"group(toggle)"** должны быть вставлены в текущий файл (как по инструкции include), но при этом подразумевается, что у всех инструкций "способ добавления" - **replace**.

Общее строение файлов конфигурации XKB.

Возможны три типа файлов конфигурации

"Простая" конфигурация

Если в файле находится "простая" конфигурация, то в начале файла должен быть заголовок
```
 [ Флаги ] ТипФайла [ Имя ]
```
после которого сразу следуют объявления (или инструкции). Например,
```
xkb_keycodes

  <TLDE> = 49;
  <AE01> = 10;
.......
```

Последовательность из "простых" блоков.

Однако, чаще используется другой формат файла - последовательность "простых" блоков. В таком файле объявления группируются в блоки, которые ограничиваются фигурными скобками - '**{...}**' (после каждого блока должна быть "точка с запятой" - '**;**').
Перед каждым блоком должен быть заголовок, такой же как и в файле с "простой" конфигурацией
```
 [ Флаги ] ТипФайла   [ Имя1 ] '{' [ Объявления ] '};'
 [ Флаги ] ТипФайла   [ Имя2 ] '{' [ Объявления ] '};'
...
```
Например,
```
xkb_symbols "basic" {....};
xkb_symbols "us" {....};
....
```

Типы файлов.

И в том и в другом формате используются одинаковые **"Типы Файлов"**. Это может быть одно из пяти слов

- **xkb_keycodes** - файл (или блок), в котором числовым значения скан-кодов даются символические имена
- **xkb_types** - файл, в котором описываются возможные типы клавиш (тип определяет - сколько различных значений может иметь одна и та же клавиша в зависимости от состояния модификаторов)
- **xkb_compat** - файл, в котором описывается "поведение" модификаторов
- **xkb_symbols** - основной файл, в котором каждому скан-коду (заданному символическим именем) назначаются все возможные значения, которые может выдавать конкретная клавиша (другими словами - "раскладка клавиатуры").

- **xkb_geometry** - описание расположения кнопок и индикаторов на клавиатуре

Надо отметить, что, если файл состоит из нескольких блоков, то все блоки должны быть одного типа. Отличаются они только именем.
Имя представляет собой произвольное слово в двойных кавычках.
Как можно заметить, в формате заголовка только **"ТипФайла"** должен присутствовать обязательно, а **"Имя"** может отсутствовать. Естественно, если файл предствляет собой "простую" конфигурацию или содержит только один блок, то именовать их необязательно. В этом случае, чтобы сослаться в настройках X-сервера на такую конфигурацию достаточно указать имя файла.
Но, если в файле находится несколько блоков, разумеется они должны иметь имена, причем разные. При этом, чтобы сослаться на конкретную конфигурацию, ее указывают в виде
имя_файла(имя_блока)
например,
us(pc104)

Флаги.

В каждом заголовке может быть несколько флагов

- **default** - этот флаг имеет смысл, если файл состоит из нескольких блоков. Он помечает один из блоков (только один !) как блок "по умолчанию". То есть, если где-то будет указано, что конфигурацию надо брать из такого файла, но при этом не сказано - из какого блока, то будет взят именно тот из блоков, который помечен флагом **"default"**.

- **partial** - означает, что в этом блоке дано не полное описание соответствующего типа, а только его часть. Например, это может быть блок типа **xkb_symbols**, в котором описаны только "функциональные" клавиши, или блок типа **xkb_geometry**, в котором описано только расположение индикаторов.

- **hidden** - означает, что определения из этого блока в "нормальном" состоянии клавиатуры (когда не активен ни один из модификаторов) не видны, и проявляются только при нажатии какого-нибудь модификатора. Например, блок типа **xkb_symbols** в котором описаны коды которые выдает "дополнительная цифровая клавиатура" (keypad) при нажатом **Num_Lock**.

Следующие флаги имеют смысл только для файлов (и блоков) типа **xkb_symbols** и просто отмечают - какие группы клавиш описаны в этом блоке

- **alphanumeric_keys** - "алфавитно-цифровые"

- **modifier_keys** - модификаторы (Control, Alt, Meta и т.п.)

- **keypad_keys** - клавиши "дополнительной цифровой" клавиатуры

- **function_keys** - "функциональные" клавиши

- **alternate_group** - "альтернативная" группа, то есть символы какого-нибудь национального алфавита.

Надо заметить, что для самого X-сервера (точнее, для программы **xkbcomp**) имеет значение только флаг **"default"**, поскольку от него может зависеть выбор нужного блока. Остальные флаги нужны скорее юзеру, чтобы лучше ориентироваться в куче различных блоков и файлов.

Кстати, сводный список всех блоков конфигураций с их флагами можно найти в файлах ***.dir** в директории **{XROOT}/lib/X11/xkb**. Названия файлов аналогичны названиям типов конфигурации - **keycodes, types, symbols** и т.д. Флаги там обозначаются одной буквой - первой буквой из названия соответствующего флага.

Последовательность из "составных" блоков.

Наконец, рассмотрим третий тип конфигурационного файла - последовательность из "составных" блоков. Каждый такой "Составной Блок" оформляется как "простой" блок

```
[ Флаги ] СложныйТип [ Имя ] '{' Блок { Блок } '}' ';'
```

но внутри содержит не просто объявления, а блоки "простых" типов, например

```
xkb_keymap "complete" {
        xkb_keycodes  {...};
        xkb_types     {...};
        xkb_compat    {...};
        xkb_symbols   {...};
        xkb_geometry  {...};
};
```

Так же, как и файл с "простыми" блоками, файл с "составными" блоками может содержать несколько таких "составных" блоков, отличающихся именами (один из блоков может быть помечен флагом **"default"**). Существует три типа "составных" блоков

- **xkb_semantics** - такой блок должен содержать блок **xkb_compat** и может также иметь в себе блок **xkb_types**;

- **xkb_layout** - должен содержать блоки **xkb_keycodes**, **xkb_types**, **xkb_symbols** и может, также, иметь в себе блок **xkb_geometry**;

- **xkb_keymap** - наиболее полный блок, должен включать в себя все, что должны содержать предыдущие два типа (то есть - **xkb_keycodes, xkb_types, xkb_compat xkb_symbols**) и может включать дополнительно те компоненты, которые могут иметь в себе два предыдущих типа.

Файл типа xkb_keycodes.

Эти файлы (блоки) имею очень простую грамматику. В них просто для всех скан-кодов задаются символические имена, которые потом используются в файлах типа **xkb_symbols** для назначения каждой "физической" клавише всех возможных значений.
Надо заметить, что те скан-коды, которые в них используются, имеют весьма отдаленное отношение к "физическим" скан-кодам, которые считываются из "железного" контроллера клавиатуры. Во-первых, X-сервер сам не считывает скан-коды непосредственно из регистров контроллера, а берет их от соответствующих драйверов (например, сервер Xfree86 запущенный на машине с FreeBSD, берет все коды от драйвера "системной консоли" - syscons).
А, кроме того, сам X-сервер переводит эти коды в свои "унифицированные" скан-коды, которые не зависят от архитектуры ("железной" и "софтверной") системы.
Поэтому, при выборе подходящей таблицы **xkb_keycodes** надо ориентироваться на тип X-сервера, который вы используете. Так, для сервера из "семейства" XFree86, самое правильное - брать определения **keycodes**, которые соответствуют набору скан-кодов, выдаваемых XFree86.
В файле **xkb_keycodes** могут встречаться четыре типа объявлений.

- Объявление переменной.

- Объявление Имени Клавиши.

- Объявление Алиаса Клавиши.

- Объявление Имени Индикатора.

Объявление Переменной.

В файле **xkb_keycodes** могут быть определены две переменные - **minimum** и **maximum**.
Они задают минимальное и максимальное значение для скан-кодов. Естественно, что все скан-коды, которые будут использоваться для задания символических имен должны "вписатся" в диапазон между **minimum** и **maximum**. Вообще говоря, "объявление переменной" имеет вид
имя_переменной '=' Выражение ';'
Поэтому, слева от знака '**=**' должно быть либо слово '**minimum**', либо - '**maximum**'. А справа может быть любое арифметическое выражение (только вот - зачем ?), которое может быть вычислено на этапе "разборки" файла и дать в результате константу типа INTEGER.

Объявление Имени Клавиши.

Основное объявление, которое используется в файлах этого типа.
Имеет вид
KEYNAME '=' Выражение ';'
KEYNAME, вообще говоря, просто некоторая строка символов (string), допустимая в языке C. Отличается только тем, что она ограничивается "угловыми скобками" - знаками '<' и '>' и должна быть не длиннее 4 символов.
Например,
<AE01> = ... ;
"Выражение" в правой части объявления может быть любым арифметическим выражением, допустимым в языке C. То есть, содержать операции '+ - / *' и "круглые" скобки, при этом операндами могут быть числовые константы десятичного, шестнадцатеричного и восьмеричного типов (в терминах языка C).
Естественно, это выражение должно быть "вычисляемым" на этапе "разборки" файла. (? наверное, в этом выражение могут использоваться ранее определенные переменные - **maximum** и **minimum**).

Объявление Алиаса Клавиши.

Эти объявления имеют вид
'alias' KEYNAME '=' KEYNAME ';'
Они служат просто для того, чтобы одному скан-коду дать несколько разных имен.

Объявление Имени Индикатора.

Вообще-то, эти объявления не имеют никакого отношения к скан-кодам, но, поскольку они тоже имеют непосредственное отношение к "железу", то самое подходящее место для них в файлах этого типа.
Заметим, что в в модуле XKB, существует 32 индикатора. Часть из них отображается на светодиодах (LED) клавиатуры (обычно - первые три) и являются "физическими" индикаторами. Остальные никак не отображаются на "физической" клавиатуре, но могут изображаться специальными программами (**xkbvleds, mxkbledpanel**). Поэтому, они называются "виртуальными индикаторами".
Поведение этих индикаторов ("физических" и "виртуальных"), то есть - в каком случае их включать/выключать, описываются в других файлах - **xkb_compat**.
В файлах типа **xkb_keycodes** им только даются символические имена, которые потом используются в файлах типа **xkb_compat** и **xkb_geometry** для указания индикатора.
Объявление имени индикатора имеет вид
'indicator' INTEGER '=' STRING ';'
или
'virtual indicator' INTEGER '=' STRING ';'
Здесь, **INTEGER** - числовая константы типа INTEGER. Она просто определяет номер индикатора. Обычно, первые три индикатора являются "физическими" (светодиоды **NumLock, CapsLock** и **ScrollLock** на клавиатуре), остальные - с 4 по 32, являются "виртуальными" (то есть, никак не отображаются на "физической" клавиатуре).
STRING - произвольная строка символов в "двойных кавычках".
Замечу также, что это имя не только служит для указания конкретного индикатора в других файлах конфигурации XKB, но и запоминается во внутренних структурах модуля XKB и может быть потом считано оттуда соответствующими программами (которые изображают "панель индикаторов" или рисуют "геометрию" клавиатуры).

Пример файла типа xkb_keycodes

```
default xkb_keycodes "xfree86" {

    minimum= 8;
    maximum= 134;

    <TLDE> =  49;
    <AE01> =  10;
    <AE02> =  11;
.......

    indicator 1 = "Caps Lock";
    indicator 2 = "Num Lock";
    indicator 3 = "Scroll Lock";

    virtual indicator 4 = "Shift Lock";
    virtual indicator 5 = "Altrnate Group";

    alias <ALGR> = <RALT>;
};
```

Файл типа xkb_types.

В этом файле описывается - каким образом вычисляется "уровень" (**shift level**) в таблице символов (**symbols**) для каждой клавиши.
Напомню, что с каждой клавишей (скан-кодом) в XKB связан двумерный массив символов. "Координатами" в этом массиве являются "номер группы" (**group**) и "уровень" (**shift level**). То есть, когда нажата какая-то кнопка и XKB должен выбрать подходящий символ для данной клавиши, этот символ выбирается из массива в соответствии с текущими "номером группы" и "уровнем".
Обычно, разные "номера групп" используется для разных национальных языков (точнее - алфавитов), а разные "уровни" используются для больших/маленьких букв (в общем-то, все знают - что меняется при нажатии кнопки **Shift**). Хотя заметим, что XKB позволяет иметь до четырех групп и до 64 (!) "уровней" (в то же время, если не ошибаюсь, **xkbcomp** допускает номер "уровня" только до 8).
Так вот. Если изменение "номера группы" описывается в файле **xkb_compat**, то зависимость "уровня" от нажатия клавиш-модификаторов (**Shift, Control, Alt** и т.п.) описывается как-раз в файлах **xkb_types**.
Точнее, в этих файлах описываются "типы" клавиш. Каждый тип имеет название (в общем-то, произвольное), а в "описании типа" определяется - как вычислять "уровень" для клавиш этого типа.
Соответственно, в файлах **xkb_symbols**, где скан-кодам "приписываются" необходимые массивы возможных символов для каждой клавиши, указывается и тип этой кнопки. Правда, принадлежность каждой клавиши к конкретному типу, уже определена "по умолчанию" в модуле XKB (и соответственно, определены четыре типа "по умолчанию"). Поэтому, не ищите в файлах **xkb_symbols** явного указания типа, они используются только в том случае, когда хочется изменить "поведение" клавиши.
Итак. В файле **xkb_types** могут встретиться объявления

- Объявление виртуальных модификаторов.
- Объявление типа.

Объявление виртуальных модификаторов.

Это объявление просто перечисляет - какие виртуальные модификаторы далее могут использоватся в объявлениях типов.
Напомню, что в самом X-сервере (не в модуле XKB) определенны восемь модификаторов - **Shift, Lock, Control, Mod1-Mod5**, которые в терминах XKB называются "реальными" (**real**). В дополнение к ним XKB может иметь еще своих 16 модификаторов, которые, соответственно в нем именуются "виртуальными" (**virtual**). Обычно, в XKB вводятся виртуальные модификаторы **NumLock, ScrollLock, Alt, AltGr** и т.п.
Назначение модификаторов на конкретные клавиши делается в файле **xkb_symbols**.
Так вот. В описаниях типа могут фигурировать как реальные, так и виртуальные модификаторы. Поскольку, "реальные" определены "по умолчанию", никаких специальных объявлений для них не требуется. А вот названия "виртуальных" желательно объявить, прежде чем описывать типы.
Объявление виртуальных модификаторов имеет очень простой вид
'virtual_modifiers' список_модификаторов ';'

где **"список_модификаторов"** - просто перечисление используемых модификаторов через запятую.
Например, строчка
virtual_modifiers NumLock, Alt;
говорит о том, что в описаниях типов могут встречаться кроме реальных модификаторов (**Shift, Lock** и т.д.), также модификаторы **NumLock** и **Alt**.

Объявление типа.

Эти объявления выглядят как
'type' ИмяТипа '{' Инструкции '};'
"ИмяТипа" - это призвольная константа типа STRING (то есть строка символов в "двойных кавычках").
Это имя потом может использоваться в **xkb_symbols** для явного указания типа клавиши.
А **"Инструкции"** это несколько объявлений (они выглядят как "присваивание" переменной некоторого значения), кадое заканчивается "точкой с запятой".
В описании типа могут всречаться инструкции

- modifiers = ...;
- map[...] = ...;
- level_name[...] = ...;
- preserve[...] = ...;

modifiers

Просто перечисляет, какие модификаторы (реальные и виртуальные) влияют на выбор "уровня" в данном типе. Если модификаторов несколько, они перечисляются через знак '+'.
Например,
modifiers = NumLock;
или
modifiers = Shift+Lock;

map[...]

Как-раз описывает - какой "уровень" выбирается в зависимости от состояния (активности) модификатора. В квадратных скобках указывается модификатор или их комбинация (через знак '+'), а справа от "знака присваивания" - соответствующий "уровень" (**Level1, Level2** ...). Кроме того, в качестве модификатора (внутри скобок) может встречаться специальное слово **"None"**, что, как не трудно догадаться, означает отсутствие (точнее - неактивное состояние) модификаторов.
Например,
map[None] = Level1;
если модификатор (модификаторы) не активен, то используется "уровень" 1,
map[Shift] = Level2;
если активен модификатор **Shift**, то выбирается "уровень" 2,
map[Control+Alt] = Level3;
если активны сразу два модификатора - **Control** и **Alt**, то выбрать "уровень" 3.
Обратите внимание, что в последнем примере каждый из модификаторов **Control** и **Alt**, по-отдельности могут никак не влиять на изменения "уровня", (а только нажатые вместе). В этом случае в описании типа не будет строчек с **map[Control]** и **map[Alt]**.
А вот **map[None]**, как правило, присутствует в каждом типе.

level_name[...]

Эта инструкция присваивает призвольное символьное имя для каждого "уровня", допустимого для данного типа. Соответственно, в квадратных скобка указывается "уровень" (**Level1, Level2** ...), а справа от

"присваивания" константа типа STRING, которая и является названием этого уровня.
Например,
```
level_name[Level1] = "Base";
level_name[Level2] = "Shifted";
```
(вместо слова level_name можно использовать levelname).
Надо заметить, что для функционирования XKB это названия (и, соответственно, вся эта инструкция) не имеет никакого значения. Они могут использоваться прикладными программами, которые показывают состояние клавиатуры.

preserve[...]

Здесь требуются некоторые пояснения.
Напомню, что X-сервер передает прикладной программе сообщение о событии (нажатии/отпускании клавиши) в котором указывается скан-код клавиши и слово - "состояние" состоящее из набора модификаторов.
Для перевода этого сообщения в символ используются соответствующие подпрограммы Xlib.
Естественно, эти программы в используют в качестве аргументов и скан-код и "состояние". Причем, отдельные подпрограммы (их там несколько) могут для принятия решения использовать не все модификаторы из "состояния".
Для того, чтобы избежать нежелательных эффектов, когда несколько таких подпрограмм обрабатывают сообщение последовательно, каждая подпрограмма обычно "вычищает" "использованные" модификаторы из слова-"состояния".
Но, в то же время, бывают ситуации, когда это нежелательно и какой-нибудь модификатор должен "приниматься во внимание" несколькими подпрограммами.
(Что это за ситуации - я сам не знаю.)
Для таких случаев и используется инструкция preserve - "сохранить" (имеется ввиду - сохранять модификатор в "состоянии").
В этой инструкции в квадратных скобках указывется модификатор (или комбинация модификаторов), такой же, как в одной из инструкции **map[...]**, а справа от "присваивания" - модификатор (или набор модификаторов), который нужно сохранять.
Обратите внимание, что в скобках обязательно должна быть комбинация (или модификатор), точно такая же как и в одной из инструкций **map[...]**.
Дело в том, что инструкция **preserve[...]** не является самостоятельной инструкцией, а представляет собой "продолжение" соответствующей инструкции **map[...]**. Поэтому, по "комбинации в скобках" XKB "сшивает" эти две инструкции.
А вот в правой части может быть только часть этих модификаторов (или даже один). То есть, в обработке будут учитываться все модификаторы из левой части, а сохраняться только те, которые указаны в правой.
Надо сказать, что в правой части инструкции **preserve** может, также, стоять и **"None"**, что означает, что "ничего сохранять не нужно".
Но, поскольку "по умолчанию" и так ничего не сохраняется, то такие инструкции особого смысла не имеют и их можно не писать.

Предопределенные типы.

В модуле XKB "по умолчанию" определены четыре типа и, соответственно, каждая клавиша "по умолчанию" приписана к одному из этих четырех типов.

- **"ONE_LEVEL"** - клавиши, которые имею только одно значение, независимо от состояния модификаторов (**Enter, Escape, Space** и т.п.)

- **"TWO_LEVEL"** - клавиши с двумя уровнями (но не "буквенные"), второй уровень выбирается модификатором **Shift** (но не зависит от **Lock**). Это, в основном, клавиши на основной клавиатуре с цифрами и "специальными" символами (1/!, 2/@, 3/# и т.д.)

- **"ALPHABETIC"** - "буквенные" клавиши. Они имеют два уровня (прописные/строчные), но в отличии от клавиш типа **"TWO_LEVEL"** зависят не только от **Shift**, но и от **Lock**.

- **"KEYPAD"** - клавиши на "дополнительной цифровой клавиатуре" (**keypad**). Тоже имеют два уровня, зависят от состояния **NumLock** и **Shift**.

Надо заметить, что, если клавиша отнесена к соответствующему типу, в ее описании (в **xkb_symbols**) массив значений должен иметь необходимое количество "уровней".
Как следствие этого, хотя вы можете переопределить любой из "предопределенных" типов (поменяв в нем модификаторы или названия уровней), но при этом нельзя менять в них количество уровней.
Если вам хочется иметь для каких-то клавиш большее количество уровней, придется сочинить для них новый тип (напомню, что назвать его можно как угодно).
Примеры описаний типов, можно посмотреть в соответствущих файлах в директории **{XKBROOT}/types/**, поэтому я здесь их приводить не буду.
А пример составления нового типа и его использования можно посмотреть в разделе "Примеры":"Новый тип для клавиши Enter".

Файл, типа xkb_compat.

В этом файле описывается поведение клавиш модификаторов - какие изменения происходят в состоянии клавиатуры (изменения битов-модификаторов и "номера группы") при нажатии этих клавиш.
Напомню, что внутри XKB существует структура (таблица) - **Xkb Compability Map**, которая состоит из двух частей

- набора (массива) "интерпретаций" (**interpret**)

- четырех переменных, которые определяют - какие "реальные" модификаторы (модификаторы "традиционного" клаватурного модуля X-сервера) будут отображать изменение "номера группы".

Если прикладные программы обращаются к X-серверу не специальными XKB-запросами, а запросами к "традиционному" клавиатурному модулю X-сервера, которые должны поменять "привязку" кодов символов (или реальных модификаторов) к скан-кодам, то модуль XKB, выполнив требуемый перенос, пытается также пренести и "действия", "привязанные" к скан-кодам (и некоторые другие параметры клавиши).
Для выполнения такого "переноса" и используется **Xkb Compability Map**.
В файле типа xkb_compat могут встречаться объявления

- Объявление виртуальных модификаторов.

- Описание "интерпретации".

- Объявление "отображения номера группы в модификатор".

- Описание поведения индикатора.

- Объявление "умолчания".

Объявление виртуальных модификаторов.

Как и в файле типа **xkb_types**, прежде всего должны быть объявлены виртуальные модификаторы, которые могут встречаться в дальнейших описаниях. Реальные модификаторы (если они также используются) описывать не надо, поскольку, они имеют стандартные названия. А вот виртуальные модификаторы могут иметь произвольные названия, поэтому, для правильной интерпретации остальных записей необходимо сообщить программе, которая будет "разбирать" этот файл, что соответствующие слова являются названиями виртуальных модификаторов.
Объявление виртуальных модификаторов имеет вид

'virtual_modifiers' список модификаторов ';'
Например,
virtual_modifiers NumLock, AltGr ;

Описание "интерпретации".

Каждая "интерпретация" (**interpretation**) устанавливает соответствие между кодом какго-нибудь "управляющего" символа (**symbol**) и "действием" (**action**), которое должен будет выполнить модуль XKB при нажатии клавиши к которой "привязан" этот символ.
Полностью внутренняя структура, описывающая интерпретацию, состоит из

- кода символа
- "действия"
- набора реальных модификаторов
- "критерия соответствия" модификаторам
- флагов ("автоповтор" и "залипание")
- виртуального модификатора для клавиши

Естественно, не все эти поля обязательно должны быть заполнены.
Итак, поле "код символа", естественно, определяет символ, к которому "привязывается" "действие", а поле "действие" - само это "действие".
Также, в "интерпретации" могут быть заданы поля "реальные модификаторы" и "критерий соответствия".
Для чего нужны эти поля?
Напомню, что к любому скан-коду может быть "привязан" один или несколько реальных модификаторов.
При поиске подходящего места для "действия", XKB может использовать не только код символа, но и расположение этих реальных модификаторов.
Если эти поля НЕ заданы, то XKB, при изменнии "привязки" символа к скан-коду, просто перенесет туда же соответствующее "действие".
А вот если эти поля заданы, то прежде чем выполнить перенос, XKB сравнивает набор реальных модификаторов, привязанных к скан-коду и набор реальных модификаторов, указанный в "интерпретации".
"Критерий соответствия" определяет - как сравнивать эти два набора (см. ниже).
Если условие выполняется, "действие" переносится.
Кстати, если используются эти два поля (набор модификаторов и "критерий соответствия"), то кода символа может в интерпретации и не быть.
При этом, поиск подходящего места делается XKB только на сравнении наборов модификаторов. Например - найти тот скан-код, к которому "привязан" реальный модификатор **Lock**, и перенести туда "действие", независимо от того, какой код символа соответствует этому скан-коду.
Итак. Поле "реальные модификаторы" представляет собой просто один или несколько битов модификаторов.
А "критерий соответствия" представляет собой одно из условий

- **AnyOfOrNone** (**любой из... или ни одного**) - практически означает, что поле "модификаторов" не имеет никагого смысла, условие выполняется, если совпадает любой из модификаторов или никакой;
- **NoneOf** (**ни один из...**) - у скан-кода не должно быть ни одного из указанных модификаторов;

- **AnyOf** (**любой из...**) - у скан-кода должен быть хотя бы один из указанных модификаторов;

- **AllOf** (**все из...**) - должны совпасть все указанные модификаторы;

- **Exactly** (**точно**) - то же, что и **AllOf**, но при этом у скан-кода не должно быть ни одного модификатора, не попавшего в список.

Кроме того, вместе с любым из перечисленных "критериев" может быть указан "критерий"

- **LevelOneOnly** (**только первый уровень**) - условие выполняется, если символ привязывается к первому уровню первой группы в соответствующем скан-коде. Выполнение этого условия также требуется, для переноса "флагов" и "виртуального модификатора".

Естественно, "по умолчанию" поле модификаторов пустое, а "критерий" - "**любой из.. или ни одного**".
Поле флагов и виртуальный модификатор (он должен быть только один), могут быть тоже "перенесены" в описание скан-кода, если символ "переносится" в первый уровень первой группы таблицы, привязанной к скан-коду.
Флаги добавляются в поле "поведение клавиши", а модификатор в поле "виртуальные модификаторы" (эти поля есть у каждого описания скан-кода).
"По умолчанию" поле "флаги" содержит флаг "автоповтор", а поле "виртуальный модификатор" - пустое.
Итак. Описание "интерпретации" имеет вид
'interpret' символ '{' описание '};'
или
'interpret' символ '+' модификатор '{' описание '};'
или
'interpret' символ '+' критерий '(' модификаторы ') {' описание '};'
Например
```
interpret  Nun_Lock {...};
interpret  ISO_Level2 + Shift {...};
interpret  ISO_Lock + AnyOf(Lock+shift) {...};
```
Если в заголовке указан только код символа, критерий - **AnyOfOrNone**, поле модификаторов - пустое.
Если указан, код символа и название модификатора (не указан "критерий"), то "критерий" - **Exactly**.
Если указан "критерий", то в скобках вместо списка модификаторов может стоять слово **all**. Понятно, что это означает - все модификаторы.
Кроме того, вместо "критерия" и списка модификаторов может использоваться слово **Any**. Это означает - **AnyOf(all)**.
Наконец, как уже говорилось, если есть набор модификаторов и "критерий", то кода символа может и не быть ("привязка" осуществляется путем сравнения наборов модификаторов). В этом случае, вместо кода символа также ставится слово - **Any**.
Например,
```
interpret Any + Any {...};
```
означает, что эта "интерпретация" применяется ко всем клавишам, у которых есть реальные модификаторы.
Внутри описания "интерпретации" могут быть строчки типа оператора присваивания

- useModMapMods = ...; или useModMap = ...;

- repeat = ...;

- locking = ...;

- virtualModifier = ...; или virtualMod = ...;

- action = ...;

useModMapMods

Служит для указания "критерия" **LevelOneOnly**. Если справа стоит слово "**level1**" или "**levelone**", то "критерий" проверяется. Если слова "**anylevel**" или "**any**" - игнорируется. Кстати, "по умолчанию" он игнорируется, так что строчки вида
```
useModMapMods = anylevel;
```
особого смысла не имеют.

repeat и locking

Устанавливают значения для флагов "автоповтор" и "залипание". Справа от присваивания должно быть логическое значение - **True** или **False**.
Например,
```
repeat = True;
locking = False;
```

virtualModifier

Указывает виртуальный модификатор. Напомню, что этот модификатор тоже може быть добавлен к описанию скан-кода, если символ переносится в "первый уровень первой группы" таблицы значений для скан-кода.
Справа от присваивания просто указывается название виртуального модификатора.
Например,
```
virtualModifier = AltGr;
```

action

Описывает "действие". Подробнее об этом читайте "Описание действий".
Здесь замечу, что "действие" также может быть "пустым". Если "интерпретация" нужна для того, чтобы перенести не "действие", а только "флаги" или "виртуальный модификатор", то ее описание может выглядеть как
```
interpret ... {
  repeat = False;
  locking = True;
  action = NoAction();
};
```

Объявление "отображения номера группы в модификатор".

Напомню, что в "состоянии XKB" (которое может быть "считано" прикладной программой) есть специальное двухбитное поле, в котором указан текущий номер группы. В "традиционном" "состоянии клавиатуры" такого поля нет, а смена группы индицируется одним из модификаторов.
Поэтому, для программ, понимающих только "традиционное состояние клавиатуры", XKB преобразует номер группы в активное состояние какого-нибудь модификатора.
Для каждого из четырех номеров групп может быть объявлен отдельный модификатор (хотя обычно, используется один для всех групп, отличных от первой).
Это объявление имеет очень простой вид.
```
'group' номер группы '=' модификатор ';'
```
Например,
```
group 2 = AltGr;
```

Описание поведения индикатора.

В файлах типа **xkb_compat** также описывается "поведение индикаторов", хотя к "таблице совместимости" (**compability**) они отношения не имеют.
Напомню, что в XKB можно определить до 32 индикаторов. Первые 3-4 (в зависимости от типа клавиатуры) отображаются реальными "лампочками" на клавиатуре, а остальные считаются "виртуальными" и могут отображаться специальными программами.
В файле типа **xkb_keycodes** индикаторам даются символические имена (связывается номера индикаторов и "имя индикатора").
А в файле **xkb_compat** описывается - как эти индикаторы ведут себя в зависимости от "состояния клавиатуры". Напомню, что...
Во-первых, индикаторы могут отображать состояние

- модификаторов,
- номера группы,
- управляющих флагов.

Причем, поскольку первые два из перечисленных "компонентов состояния XKB" "размазаны" по трем переменным (**base, locked, latched**), поведение индикаторы можно связать с любой из указанных переменных или с их "эффективным" (суммарным) значением.
Надо отметить, что один и тот же индикатор может одновременно "отслеживать" изменения и "своего" модификатора, и какого-нибудь номера группы, и управляющего флага (хотя, зачем это нужно?).
Во-вторых, индикатор может не только "отслеживать" состояние XKB, но и включаться/выключаться прикладными программами. При этом, в описании индикатора можно разрешить/запретить такое включение/выключение или установить "обратную связь" (то есть, при включении/выключении индикатора будут происходить соответствующие изменения в состоянии XKB).
Итак. Описание поведения индикатора имеет вид
'indicator' имя_индикатора '{' описание '};'
Здесь "**имя_индикатора**" - это то символическое имя (строка символов в двойных кавычках), которое было дано ему в файле **xkb_keycodes**.
А "**описание**" обычно имеет вид оператора присваивания (исключение - логические переменные-флаги, которые могут принимать значения только **True/False**).
В этих "описаниях" могут встретиться строчки типа

- modifiers = ...; или mods = ...;
- groups = ...;
- controls = ...; или ctrls = ...;
- whichModState = ...; или whichModifierState = ...;
- whichGroupState = ...;
- allowExplicit = ...;
- drivesKeyboard = ...; (имеет кучу "синонимов", см. ниже)
- index = ...;

modifiers, groups и controls

Определяют - какие компоненты "состояния" должен отслеживать индикатор.
Естественно, справа от знака присваивания должен быть ...

- для **modifiers** - название модификатора или нескольких модификаторов через знак "+";

- для **groups** - номера групп;

- для **controls** - название "управляющего флага" (флагов).

Надо отметить, что номер группы можно задавать

- просто числовым значением

- в виде - **group1, group2** и т.п.;

- можно использовать слова "**none**" (0) и "**all**" (0xFF);

- и, наконец, с помощью простых арифметических выражений, например, **All-1** ("все, кроме первой")

whichModState и whichGroupState

Поскольку набор модификаторов и номер группы "размазаны" по трем переменным (**base, locked, latched**), эти инструкции уточняют - в каких переменных надо отслеживать модификаторы и номер группы, соответственно.
Справа от присваивания может быть слово

- **base** - отслеживать изменение в переменных **base** (**base Group** или **base Modifiers**, соответственно)

- **locked** - то же самое, в переменных **locked**

- **latched** - то же самое, в переменных **latched**

- **effective** - отслеживать изменения "эффективных" номера групп или модификаторов (то есть в суммарных значениях всех трех переменных)

- **any** - отслеживать изменения во всех трех переменных (то есть, индикатор будет включаться/выключаться, если указанное значение модификатора или группы будет меняться в любой из трех переменных); надо заметить, что для модификаторов (**whichModState**) это значение эквивалентно **effective**, поскольку "включение" модификатора в любой из трех переменных, неизбежно "включит" его и в "эффективном" наборе модификаторов;

- **none** - "ни в каком"; особого смысла не имеет, поскольку при этом просто не отслеживаются модификаторы и/или номер группы.

По умолчанию (то есть, если **which...State** явно не указаны) подразумевается **effective**.

allowExplicit

Логический флаг, который разрешает (запрещает) прикладным программам включать/выключать индикатор. Обратите внимание, конечно, для включения/выключения индикатора прикладная программа посылает специальные запросы к XKB. Но XKB по этим командам только меняет (если это разрешено) только

состояние индикатора, не затрагивая свое "состояние". Естественно, при этом состояние индикатор может не соответствовать состоянию XKB.
Поскольку **allowExplicit** является логической переменной, справа от присваивания должно быть только **True** или **False**.
Можно также использовать другую форму этой инструкции. Так, просто указание
allowExplicit;
эквивалентно
allowExplicit = True;
а строчка
!allowExplicit;
эквивалентна
allowExplicit = False;
По умолчанию этот "флажок" - **True**. То есть, прикладным программам разрешено менять состяние индикатора, помимо "состояния клавиатуры".

drivesKeyboard

Имеет много синонимов - **drivesKbd, ledDrivesKbd, ledDrivesKkeyboard, indicatorDrivesKbd, indicatorDrivesKeyboard**.
Это тоже логический флаг, который заставляет XKB устанавливать "обратную связь" между индикатором и "состянием клавиатуры". То есть, если это флаг "взведен" (и разрешено **allowExplicit**), то, при изменении прикладной программой состяния индикатора, XKB должен изменить и связанные с ним компоненты "состояния клавиатуры".
Обратите внимание, что

- меняться должны те компонеты, которые заданы инструкциями **modifiers**, **group** и **controls** (обычно задан только один из компонентов);

- инструкции **whichModState** и **whichGroupState** указывают - в каких из трех переменных (**base, locked, latched**) следует поменять модификатор или группу.

При этом, если "**which...state**" - **none, base** или **any**, никакого эффекта не будет. А **effective** эквивалентно **locked**. Напомню, что по умолчанию подразумевается значение **effective**, следовательно - если инструкций вида "**which...state**" в описании нет, то изменения будут делаться в **locked Group** или **locked Modifiers**, соответственно.
Как и в случае с флагом **allowExplicit**, объявление **drivesKeyboard** должно иметь вид
drivesKeyboard = True; (эквивалент - drivesKeyboard;)
или
drivesKeyboard = False; (эквивалент - !drivesKeyboard;)

index

Позволяет указать номер индикатора (физического или виртуально). Вообще-то, номер индикатора связывается с "именем индикатора" в файле типа **xkb_keycodes**. Но можно указать его явно здесь.

Объявление "умолчания".

Это объявление является воспомогательным и позволяет определить какое-нибудь поле (инструкцию) для всех записей типа **interpret** или **indicator**. Естественно, обычно эти объявления помещаются в начале файла (или блока в файле).
Они меют вид оператора присваивания, где в левой части указывается конструкция типа "поля структуры" в языке C.
Например,
indicator.allowExplicit = False;
что означает - во всех дальнейших описаниях индикаторов (**indicator**) подразумевается "**allowExplicit = False;**", если конечно, эта инструкция не указана явно.
Первым словом в левой части (то, что до точки) должно быть

- **interpret** - "умолчания" для описания "интерпретаций";

- **indicator** - "умолчания" для описания индикаторов;

- **название "действия"**, встречающегося в описаниях **interpret** - задает "умолчание" для соответствующих полей "действия", которое может встретиться в дальнейших описаниях "интерпретаций".

Файл типа xkb_symbols.

В этих файлах собственно и описывается "раскладка клавиатуры". То есть, для каждой физической клавиши (скан-кода) задается набор всех возможных символов, которые будут выдаваться в зависимости от текущего "состояния клавиатуры" (номера группы и состояния модификаторов).
Напомню, что с каждой клавишей связана двумерная таблица символов (**symbols**). Эта таблица делится на группы (**group**), выбор конкретной группы зависит от текущего номера группы в "состоянии клавиатуры". Каждая группа, в свою очередь делится на уровни (**shift level**), выбор уровня зависит от типа клавиши (**type**) и состояния модификаторов.
Надо заметить, что разные клавиши могут иметь разное количество групп, и разные группы одной клавиши могут иметь разное количество уровней.
Также, с некоторыми клавишами может быть связана аналогичная двумерная таблица "действий" (**actions**). Хотя обычно, действия "привязывают" не к скан-кодам в файлах **xkb_symbols**, а к соответствующим символам в файлах типа **xkb_compat**.
Прежде чем рассматривать грамматику файла **xkb_symbols**, рассмотрим - какие еще данные могут быть связанны со скан-кодами, кроме таблиц символов и "действий". Как правило, для всех этих данных есть значения по умолчанию, поэтому, обычно они в файлах **xkb_symbols** явно не указываются. Но, если есть необходимость, их можно также явно задать в файлах этого типа.
Итак. С каждым скан-кодом связаны

- **тип клавиши** - типы описываются в файлах **xkb_types** и определяют зависимость уровня от состояния модификаторов. Каждая группа одно и той же клавиши может иметь разный тип.

- **"метод выравнивания" номера группы** - напомню, что некоторые клавши могут иметь меньшее количество групп, чем все остальные. Поэтому, при нажатии такой клавиши может оказаться, что номер группы в "состоянии клавиатуры" выходит за границы, допустимые для данной клавши. В этом случае он "выравнивается" до приемлимого значения. "Методы выравнивания" для отдельных клавиш такие же, как и глобальные (см. "Внутренности...":Метод Выравнивания)

- **автоповтор (autorepeat)** - логический "флаг", который определяет - нужен ли автоповтор для данной клавиши.

- **"поведение" клавиши (behavior)** - набор флагов и дополнительный аргумент, которые определяют...

    - **"залипание" (locking)** - если клавиша "залипающая", то при первом нажатии/отжатии выдается только сообщение о нажатии клавиши, а при повторном нажатии/отжатии - только сообщение об отжатии клавиши.

    - **"радио-группа"** - клавиша принадлежит к радио-группе клавиши, дополнительный аргумент определяет номер этой радио-группы. Напомню, что клавши одной радио-группы являются взаимозависимыми. То есть, при нажатии одной из клавиш группы, она "залипает", а остальные клавиши этой группы "отжимаются".

    - **допускается "отжатие всех" (allow none)** - имеет смысл для клавиш радио-группы. если этот флаг установлен, то повторное нажатие на клавшу - члена радио-группы, она "отжимается". При этом все члены группы могут находиться в отжатом состоянии. Если же этот флаг не стоит, то для отжатия клавиши надо нажать любую другую из той же группы. При этом в группе одна из клавиш остается нажатой.

    - **перекрытие 1** - указывает, что клавиша относится к группе "перекрывающихся" клавиш (**overlay**). Если в состоянии клавиатуры установлен "управляющий флаг" **Overlay1**, то эта клавиша должна "отослать" XKB к другому скан-коду, который задан дополнительным аргументом.

    - **перекрытие 2** - то же самое, что и предыдущий, только эти клавиши зависят от "управляющего флага" **Overlay2**.

    - **permanent** - может комбинироваться с другими флагами и означает, что соответствующая функция выполняется "железом" клавиатуры и нет необходимости эмулировать её в XKB программно.

- **виртуальный модификатор** (или несколько модификаторов) - этот модификатор может использоваться в качестве аргумента для "действия", если с клавишей связаны какие-нибудь "действия". Надо заметить, что, как правило, виртуальные модификатооры "назначаются" не в файлах **xkb_symbols**, а, как и "действия", в файлах **xkb_compat**.

- **"набор исключений"** - запрещает выполнении "интепретаций" - изменения привязки "действий" при изменении привязки символов к скан-кодам. Можно запретить выполнение всех действий "интерпретации" для данной клавиши или только отдельных ее шагов - перенос виртуального модификатора, перенос "автоповтора", перенос "залипания".

- и, наконец, в отдельной таблице может быть задана "привязка" реальных модификаторов к скан-кодам. Если с клавишей связан реальный модификатор, то, при нажатии клавиши, автоматически меняется состояние соответствующего модификатора в наборе "традиционных модификаторов", который эмулируется XKB для старых клиентских программ, "не знающих об XKB". Кроме того, "привязка" реальных модификаторов может использоваться при выполнении "интерпретаций" (**interpretation**).

Объявления в файле xkb_symbols.

В файлах этого типа могут встретиться

- Объявление виртуальных модификаторов.

- Объявление имени группы.

- Описание клавиши.

- Объявление "привязки" реальных модификаторов.

- Объявление "умолчаний".

Объявление виртуальных модификаторов.

Просто перечисляет названия виртуальных модификаторов, которые могут встретиться в описаниях "действий" и в качестве модификатора "привязанного" к клавише.
Имеет вид
'virtual_vodifiers' список_модификаторов ';'
Надо заметить, что обычно ни действия, ни связанные с клавишей виртуальные модификаторы, не "привязываются" непосредственно к скан-кодам. Как правило, они описываются в файле **xkb_compat**, как часть "интерпретаций". Поэтому, это обявление, обычно, в файлах **xkb_symbols** не встречается.

Объявление имени группы.

Задает символическое имя для группы. Это имя может потом использоваться прикладными программами, которые рисуют изображение клавиатуры или показывают "состояние клавиатуры". Для самого XKB эти имена значаения не имеют.
Это объявление имеет вид
'name[' название_группы ']=' имя_группы ';'
Например,
name[Group1] = "English" ;
name[Group2] = "Russian" ;

Описание клавиши.

Это основное обявление в файлах этого типа. Именно оно описывает таблицу символов (и, если надо, "действий") связанных со скан-кодом.
Имеет вид
'key' имя_скан-кода '{' описания '};'
Напомню, что "имя_скан-кода" описывается в файлах типа **xkb_keycodes** и представляет собой произвольную строчку символов (но длиной не более четырех), ограниченную "угловыми скобками".
Например,
key <LCTL> {...};
"Описания" внутри фигурных скобок разделяются запятой. Обратите внимание, что именно "запятой", а не "точкой с запятой", как это делается в файлах других типов.
Итак, внутри скобок могут быть строчки типа

- type = ..., или type = ...,
- locks = ..., (синоним - **locking**)
- repeat = ..., (синонимы - **repeats, repeating**)
- groupswrap, или warpgroups,
- groupsclanp, или clampgroups,
- groupsredirect = ..., или redirectgroups = ...,
- radiogroup = ...,

- allownone = ...,
- overlay1 = ..., или overlay2 = ...,
- permanent...
- vmods = ..., (синонимы - **virtualmods, virtualmodifiers**)
- symbols[...] = ...,
- actions[...] = ...,
- просто [...]

type

Определяет тип клавиши. Справа от присваивания должно стоять название одного из типов, определенных в файле **xkb_types**.
Обратите внимание, поскольку разные группы могут иметь разные типы (напомню, что тип определяет количество уровней в группе), то это описание в общем случае должно иметь вид
type[номер_группы] = название_типа,
например,
type[Group1] = "ONE_LEVEL",
type[Group2] = "ALPHABETIC",
Но, если все группы имеют одинаковое количество уровней и относятся к одному типу, то указание группы (вместе с квадратными скобками) можно пропустить. Например,
type = "ALPHABETIC",
Надо заметить, что для всех клавиш, внутри XKB имется значения типа "по умолчанию". Поэтому, как правило, тип клавиши (группы) не указывется.

locks

Логический флаг, который указывает на то, что клавиша должна быть "залипающей".
Справа от присваивания могут стоять слова - **true, yes, on** ("поднять" флаг) или - **false, no, off** ("сбросить" флаг). Кроме того, там же может встретиться слово **permanent**. В этом случае подразумевается, что клавиша "залипающая", но ее "залипание" делается "железом" клавиатуры (в общем-то, это означает, что самому XKB об этой клавише заботиться не надо).
По умолчанию все клавише не "залипающие".

repeat

Логический флаг, который определяет - нужен ли "автоповтор" для данной клавиши. Так же, как и для флага **locks**, справа от присваивания могут быть слова - **true, yes, on** (нужен автоповтор) или - **false, no, off** (автоповтор не нужен).
Кроме того, там же может стоять слово **default**. Дело в том, что автоповтор, обычно, выполняет само "железо" клавиатуры. Поэтому, XKB не надо заботиться о повторении нажатия клавши. Чаще всего, ему приходится наоборот - "подавлять" автоповтор для некоторых клавиш. Так вот, значение "**default**" означает, что автоповтор надо "оставить на совести" "железа" и не пытаться что-либо менять.
По умолчанию большинство клавиш отрабатывают автоповтор и значения **repeat** для них - **default**. А для клавиш-модификаторов - **Control, Shift, Alt, CapsLock, NumLock** и т.п., автоповтор подавляется, и **repeat** для них - **false**.

groupswrap, groupsclanp, groupsredirect

Определяют "метод выравнивания" номера группы (подробности см. в "Внутренности":Метод выравнивания). Естественно, имеет смысл в описании клавиши указывать только один из них.
Объявления **groupswrap** и **groupsclamp** являются просто логическими переменными. Поэтому они задаются либо в виде присваивания, где в правой части могут быть только слова **True** или **False**, либо в виде
groupswrap, - подразумевается "**= True**"
или
!groupswrap, - подразумевается "**= False**"
А вот метод **groupsredirect** подразумевает дополнительный аргумент - "куда redirect". Поэтому всегда имеет вид присваивания, где в правой части должен стоять номер группы. Например,
groupsredirect = 1,
По умолчанию для всех клавиш метод выравнивания - **Wrap**.

radiogroup и allownone

Означает, что данная клавиша является членом радио-группы. Справа от знака присваивания должен быть номер группы.
Номер группы может быть произвольный, в пределах 2-128 (обратите внимание, что нельзя сделать радио-группу номер 1, хотя это скорее всего "баг" в реализации XKB).
Allownone устанавливает соответствующий флаг для этой радио-группы и является просто логической переменной.
По умолчанию никаких радио-групп нет.

overlay1 или overlay2

Означает, что данная клавиша принадлежит к одной из двух групп "перекрытий". Напомню, что когда режим "перекрытия" активен (установлен соответствующий "управляющий флаг" в состоянии клавиатуры), такая клавиша эмулирут нажатие клавиши с другим скан-кодом. Поэтому, справа от знака присваивания должно быть "название скан-кода" клавиши, нажатие которой эмулируется. Это название имеет такой же вид, как и в заголовке описания клавиши и должно быть определено в файле типа **xkb_keycodes**.
Например,
overlay1 = <XY01>,
По умолчанию никаих "групп перекрытий" нет.

permanent...

Это не отдельное слово, а префикс, который может соединяться со словами **radiogroup, overlay1, overlay2.**
Например,
permanentradiogroup = ...,
permanentoverlay1 = ...,
permanentoverlay2 = ...,
Означает, что данная функция и так отрабатывается "железом" и XKB не надо об этом заботиться.

vmods

Задает список виртуальных модификаторов, связанных с этой клавишей. Справа от знака присваивания должно быть название модификатора (или нескольких модификаторов через знак '+').
Напомню, что обычно виртуальные модификаторы "привязываются" не здесь, а в файлах **xkb_compat**.

symbols

Основная часть описания. Задает набор символов для клавиши. Одно такое объявление задает набор символов для одной группы. Поэтому, в левой части, в квадратных скобках указывается название группы, а в правой части, опять же в квадратных скобках - список символов для всех уровней этой группы (через запятую).
Например,
symbols[Group1] = [semicolon, colon],
symbols[Group1] = [Cyrillic_zhe, Cyrillic_ZHE],
В качестве "символов" могут быть числовые значения кодов (десятичные, восьмеричные, шестнадцатеричные) или специальные "названия символов".
Названия символов можно найти в файле **X11R6/include/X11/keysymdefs.h**. Только там они еще имеют префикс "**XK_**". То есть, если в это файле есть, например, определения

```
#define XK_Escape 0xFF1B
#define XK_Delete 0xFFFF
....
```
это означает, что в файле типа **xkb_symbols** можно использовать слова **Escape** и **Delete** в качестве "названий символов".
Надо заметить, что если в качестве символа указаны числа 0 - 9, то они интерпретируются как коды символов '**0**' - '**9**', а не как числовой код символа.
Если для какого-то уровня в группе символ не нужен (не определен), можно использовать специальное "название символа" - **NoSymbol**.

actions

Аналогично предыдущему объявлению, только задает не список символов, а список "действий" для данной клавиши. Имеет такой же вид как объявление **symbols**, только вместо "имен символов" должны быть описания "действий". Немного подробнее об этих описаниях смотри "Описание действий".
Здесь замечу только, что если какой-то уровень не имеет соответствующего "действия", можно использовать специальное название - **NoAction()**.

просто [...]

Чаще всего, описание клавиши состоит из списков символов, заключенных в квадратные скобки без всякого указания типа - "**symbols[...] =**". Поскольку обычно для клавиши задается только набор символов, можно использовать сокращенную форму описания.
Например, описание
```
key <AE03> { [   3,      numbersign ], [  apostrophe,  3 ] };
```
полностью эквивалентно описанию
```
key <AE03> {
  symbols[Group1]= [             3,      numbersign ],
  symbols[Group2]= [      apostrophe,             3 ]
};
```
То есть, первая пара квадратных скобок (с неким содержимым внутри) интерпретируется как описание **symbols** для первой группы, втора пара скобок - как описание **symbols** для второй группы и т.д.

"Набор исключений".

Напомню, что с каждой клавишей может быть связан набор исключений, который запрещает изменять "привязку" "действий", флагов "залипания" и автоповтора и набора виртуальных модификаторов при выполнении "интерпретаций".
Заметьте, что в описании клавиши нет явных инструкций для задания "набора исключений". Но этот набор все-таки создается в некоторых случаях

- если в описании клавши явно указан набор "действий" (инструкция **actions**), то устанавливается запрет "выполнения интерпретации" для этой клавиши;
- если задан явно автоповтор (инструкция **repeat**) - запрещается "изменение автоповтора";
- если задан явно флаг "залипания" или радио-группа (инструкции **locks** и **radiogroup**) - запрещается "изменение залипания";
- и, наконец, если указан явно список виртуальных модификаторов (инструкция **vmod**), то устанавливается "запрет изменения" набора модификаторов.

Объявление "привязки" реальных модификаторов.

Это объявление заполняет внутреннюю табицу XKB - **modmap**, которая "привязывает" реальные модификаторы к клавишам (скан-кодам). Напомню, что эти модификаторы будут автоматически устанавливаться/сбрасываться, при нажатии/отпускании клавиши, в "эмулируемом наборе модификаторов".

Объявление имеет вид
'modifier_map' имя_модификатора '{' список_клавиш '};'
Вместо слова "**modifier_map**" могут использоваться синонимы - **modmap** или **mod_map**.
"**Имя_модификатора**" должно быть названием одного из реальных модификаторов - **Shift, Lock, Control, Mod1, Mod2, Mod3, Mod4, Mod5**.
А вот "**список_клавиш**" может состоять из названий скан-кодов (через запятую), например,
modifier_map Control { <LCTL>, <RCTL> };
или из названий символов, например,
modifier_map Mod1 { Alt_L, Alt_R };
Во втором случае, XKB (точнее - **xkbcomp**) должен найти скан-коды, к которым "привязаны" эти символы и занести в **modmap** эти скан-коды.
Обратите внимание, один и тот же модификатор может быть "привязан" ко многим скан-кодам, но не наоборот - разные модификаторы к одному скан-коду. Это означает, что название скан-кода может появиться в определениях **modmap** только один раз. Это же ограниичение действует, если клавиши предствлены не скан-кодами, а символами.
Однако, как ни странно, **xkbcomp** не проверяет ситуацию, когда одна и та же клавиша представлена один раз скан-кодом, а другой - символом, или разными символами "привязанными" к одному скан-коду. В этом случае может получиться ситуация, когда к одному скан-коду "привязаны" несколько реальных модификаторов.

Объявление "умолчаний".

Это объявление задает значение "по умолчанию" для некоторых аттрибутов клавиш и выглядит как присвоение значения "полю структуры" в языке C.
Например,
key.repeat = no;
При этом, в левой части присваивания первое слово (до точки) должно быть слово "**key**", а второе - любое из допустимых в описании клавиши (**type, locks, radiogroup** и т.п.).
Естественно, это "умолчание" будет действовать, пока в тексте не встретится другое объявление для того же аттрибута.
Кроме того, объявление "умолчаний" может спользоваться для "умолчаний" в описании "действий" (подробнее см. "Описание действий"). В этом случае первое слово будет названием "действия", например,
SetMods.clearLocks = True;
И, наконец, к объявлениям "умолчания" можно отнести инструкцию, которая устанавливает флаг "**допускается отжатие всех**" (**allownone**) для радио-групп.
Напомню, что этот флаг можно указать непосредственно в описании клавиши, относящейся к радио-группе. Но, поскольку радио-группа "размазана" по нескольким клавишам, а флаг **allownone** является аттрибутом радио-группы, а не конкретной клавиши, можно указать флаг для нее отдельной инструкцией (не внутри описании какой-либо клавиши). Например,
allownone = 10;
означает, что для радио-группы 10 устанавливается соответствующий флаг.

Описание "действий".

- "Действия" для -
    - Изменения номера группы.
    - Изменения виртуальных модификаторов.
    - Изменения управляющих флагов.
- Как описываются "действия" в файлах настройки.
- Объявление "умолчания".

Описание "действий" (**actions**) используются в файлах типа **xkb_symbols**, где они "привязываются" к скан-кодам клавиш, и в файлах типа **xkb_compat**, где они "привязываются" к управляющим символам. (Напомню, что в **xkb_compat**, описываются "интерпретации" - таблички, которые помогают менять привязку "действия" к скан-кодам, когда прикладные программы меняют привязку соответствующих "управляющих символов").
В XKB определен большой набор возможных "действий" (функций). Поэтому, описывать все действия здесь я не буду. Упомяну только, что с помощью действий можно

- менять номера групп, наборы модификаторов и управляющие флаги
- менять положение и состояние кнопок "указателя" ("мышки")
- посылать сообщения программам
- перключать виртуальные экраны X-сервера
- и т.п.

Естественно, от типа "действия" зависит набор (и количество) аргументов и возможные значения этих аргументов.
Надо также заметить, что XKB отслеживает как нажатие, так и отпускание клавиши. При этом "действие" может срабатывать только при нажатии или только при отпускании клавиши. А может, также, выполнять различные действия при нажатии и при отпускании (обычно - противоположные, см. ниже).
Рассмотрим подробнее несколько "действий". Те, которые меняют модификаторы, номера групп и "управляющие флаги" в "состоянии XKB".

- Изменение номера группы.
- Изменение виртуальных модификаторов.
- Изменение управляющих флагов.

Изменение номера группы.

Напомню, что номер группы "размазан" по трем переменным (**base, locked** и **latched**). Поэтому для его измения используются три разных действия

- **SetGroup** - меняет переменную **base group**
- **LockGroup** - меняет переменную **locked group**
- **LatchGroup** - меняет переменную **latched group**

Все эти действия имеют два аргумента

- номер группы (естественно)

- набор флагов

Флаги для этих "действий"

- **groupAbsolute** - определяет, что аргумент "номер группы" задает абсолютное значение, которое нужно занести в соответствующую переменную. В противном случае в поле "номер группы" задана "добавка", которую нужно добавить или вычесть из соответствующей переменной.

Заметьте, что этот флаг в описании "действий" явно не указывается. Если необходимо задать относительное значение для "номера группы" оно пишется со знаком (+ или -), абсолютное - без знака. Таким образом, "номер группы" = 2, означает, что в соответствующую переменную надо записать двойку, а "номер группы" = +2 означает, что к переменной надо добавить двойку.

- **clearLock** - применяется в "действиях" **SetGroup** и **LatchGroup** и означает, что попутно с этим действием надо "почистить" переменную **lock group** (точнее установить в ней **group = 1**).

- **latchToLock** - применяется только в **LatchGroup** и означает, что при нажатии кнопки значение "номер группы" помещается в **latched group** (точнее, это всегда делается при выполнении **LatchGroup**), а вот при отпускании кнопки, перенсти значение из **latched group** в **locked group**.

Есть еще одна тонкость в поведении этих трех "действий". Как уже говорилось, "действие" может по разному вести снебя при нажатии и при отпускании кнопки.
Так вот. **SetGroup** и **LatchGroup** при нажатии заносят в соответствующие переменные номер группы (или добавляют/отнимают "добавку"), а при отпускании - делают обратное действие, возвращают "все как было". И только **LockGroup** при отпускании "запоминает" значение в **locked group**.
Таким образом, изменения в **base group** и **latched group** "держатся" только пока соответствующая клавиша нажата.

Изменение виртуальных модификаторов.

Также, как и для номеров групп, для изменения виртуальных модификаторов в "состоянии XKB" предназначены три разных "действия"

- **SetMods** - меняет переменную **base modifiers**

- **LatchMods** - меняет переменную **latched modifiers**

- **LockMods** - меняет переменную **locked modifiers**

Аргументы - список модификаторов и набор флагов.
Значение флагов

- **useModMapMods** - означает, что аргумент "список модификаторов" игнорируется, а сами модификаторы берутся из переменных "привязанных" к клавише (скан-коду). Напомню, что с каждой клавишей могут быть связаны набор реальных модификаторов (**modmap**) и набор виртуальных модификаторов (**vmodmap**), именно эти два набора и используются в качестве аргумента.

- **clearLocks** - имеет смысл только для **SetMods** и **LatchMods**, означает, что попутно надо очистить **locked modifiers** (имеется ввиду - "сбросить" указанные модификаторы).

- **latchToLock** - имеет смысл только для **LatchMods**, означает, что при отжатии кнопки значение из **latched modifiers** переписывается в **locked modifiers**.

- **LockNoLock** - имеет смысл только для **LockMods**, означает, что указанные модификаторы не записываются по нажатию в **locked modifiers** (какой в этом смысл - см.ниже)

- **LockNoUnlock** - также имеет смысл только для **LockMods** и означает, что при отжатии кнопки модификаторы не "сбрасываются".

Рассмотрим немного подробнее поведение этих "действий" при нажатии и при отжатии кнопок.

- **SetMods**
    - при нажатии: заносит модификаторы в base **modifiers**
    - при отжатии: "сбрасывает" модификаторы в **base modifiers**; если стоит флаг **clearLocks**, то эти модификаторы "сбрасываются" также и в **locked modifiers**.

- **LatchMods**
    - при нажатии: заносит модификаторы в **latched modifiers**
    - при отжатии: "сбрасывает" модификаторы в **latched modifiers**; если стоит флаг **clearLocks**, то эти модификаторы "сбрасываются" также и в **locked modifiers**, если стоит флаг **latchToLock**, то эти модификаторы переносятся в **locked modifiers** (получается аналог **LockMods**)

- **LockMods**
    - при нажатии: заносит модификаторы в **base modifiers**, если при этом в **locked modifiers** указанный модификатор не активен, то он "взводится" в **locked modifiers**, в противном случае - наоборот, "сбрасывается". если стоит флаг **LockNoLock**, модификаторы устанавливаются только в **base** (получается аналог **SetMods**).
    - при отжатии: "сбрасывает" модификаторы в **base modifiers**, но оставляет его в **locked modifiers** (если, конечно, он там "взвелся", а не "сбросился") То есть, при первом нажатии/отжатии модификаторы запоминаются в (и после отжатия клавиши), а при повторном - сбрасываются (по нажатию).

Изменение управляющих флагов.

Поскольку набор управляющих флагов в XKB только один (а не три, как для модификаторов и номера группы), для изменения это набора существует только два действия

- **SetControls** - действует как **Set(Group|Mods)**, то есть - при нажатии кнопки устанавливает флаги, при отжатии - сбрасывает

- **LockControls** - действует как **Lock(Group|Mods)**, то есть - не сбрасывает установленные флаги при отжатии кнопки.

Аргумент только один - список "управляющих флагов".
Строго говоря, флаги у **LockControls** тоже есть, но при описании этой функции в файлах настройки они игнорируются (их можно установить, только если действие модифицируется прикладной программой с помощью специальных запросов к XKB).

Как описываются "действия" в файлах настройки.

Описание "действия" имеет такой же вид, как вызов функции в языке С. то есть
название функции '(' список аргументов через запятую ')'
Небольшое отличие только внутри списка аргументов. Флаги, обычно, просто указываются по имени. А вот другие аргументы (номер группы, список модификаторов и т.п.) указываются как
название аргумента '=' аргумент
Для описанных выше "действий" аргументы называются

- для xxx**Group - group**
- для xxx**Mods - modifiers**
- для xxx**Controls - controls**

Например
LockGroup(group=2)
заметьте, подразумевается флаг **groupAbsolute**
SetGroup(group=+1,clearLock)
здесь указывается относительная "добавка" и флаг **clearLock**
SetMods(modifiers=NumLock, clearLock)
здесь явно задан модификатор, и флаг **clearLock**
LockMods(modifiers=useModMapMods)
обратите внимание - флаг **useModMapMods** указывается вместо списка модификаторов, а не как отдельный флаг.
Кстати, флаги **lockNoLock** и **lockNoUnlock** в описаниях игнорируются (их можно выставить только с помощью отдельных программ).
LockControls(controls=Overlay1)
устанавливается флаг **Overlay1** (включения "режима перекрытия" для "перекрытия" номер 1).

Объявление "умолчания".

В тех файлах, где могут появиться описания "действий" - **xkb_compat** и **xkb_symbols**, могут также использоваться объявления "умолчания".
Они выглядят как оператор присваивания полю структуры в языке С. То есть, в левой части присваивания стиот конструкция состоящая из двух слов, разделенных точкой.
Эти объявления могут использоваться для задания значений флагов "по умолчанию" для "действий" встречающихся в файле. В этом случае первое слово - название "действия", а второе - название флага.
Естественно, справа от знака присваивания может быть только логическое значение - **True/False**.
Например
setMods.clearLock = True;
означает - во всех дальнейших описаниях **SetMods**, добавляется флаг **clearlock**.
latchMods.clearLock = True;
latchMods.latchToLock = True;
во всех дальнейших описаниях **LatchMods**, добавляются флаги **clearLock** и **latchToLock**.

Примеры изменения конфигурации XKB.

Прежде всего, хочу заметить, что все решения, рассмотренные в примерах, не претендуют на "правильность".
Более того, многие из них я сам считаю или излишне "корявыми" (громоздкими), или "идеологически неправильными".
Но, с другой стороны, я не предлагаю готовые решения для всех проблем, а только хочу показать - чего можно добиться простым изменением текстовых "конфигов" XKB.

Во-первых, давайте решим вопрос -

Где будем экспериментировать?

Конечно, все изменения конфигурации можно сделать непосредственно в соответствующих файлах настройки XKB. Но это очень неудобно по нескольким причинам

- всегда хочется иметь возможность "откатиться" к стандартной конфигурации, поэтому придется сохранять отдельно оригинальные файлы;
- если в следующих версиях "иксов" эти файлы изменятся, то вам придется либо переносить свои изменения вручную, либо продолжать пользоваться старыми файлами;
- ну и, наконец, если мы вносим изменения только в одно-два определения, зачем нам искать их каждый раз в "многоэтажном" тексте.

При этом **xkbcomp** позволяет легко "нанизывать" несколько файлов при описании одного компонента настройки XKB. Например, файл (блок) описания компонента **xkb_types** может выглядеть как

```
xkb_types {
  include "basic+pc+мои_типы+еще_один_полезный_тип";
};
```

Что означает -

- взять описание из файла **basic**;
- добавить к нему описание из файла **pc**;
- добавить описание из файла **мои_типы**;
- и, наконец, добавить еще описание из файла **еще_один_полезный_тип**.

При этом, если в добавляемом файле встретится определение для какого-то элемента, уже описанного в предыдущих файлах, новое определение, обычно, замещает старое, не затрагивая все остальные определения.
Итак. Давайте все изменения/исправления/дополнения помещать в отдельных файлах и просто "приплюсовывать" эти файла к уже имеющимся.
Хотя, в некоторых случаях (особенно это касается добавлений в **xkb_symbols**), боле эффективным может оказаться не "приплюсовывание", а объявление добавляемого файла отдельной инструкцией - replace. Напомню, что "плюсик" в инструкции **include** означает, что инструкции из файла будут добавляться в режиме override (см "Способ добавления".). А при переопределении клавиш часто требуется способ replace. Поэтому, при добавлении в **xkb_symbols**, вместо одного длинного **include** лучше использовать конструкцию типа

```
xkb_symbols {           include "en_US(pc104)"
                        replace "my.symbols"
                        replace "one_another_symbol" };
```

Теперь осталось решить вопрос - куда "приплюсовывать"?
Во-первых, напомню, что программа **xkbcomp** может "на ходу" поменять настройки XKB прямо в работающем X-сервере. Для этого вторым ее аргументом ("куда") нужно указать "X дисплей". Если вы работаете на той же машине, где и запущен X-сервер, то это выглядит как

```
xkbcomp ... :0.0
```

(можно еще проще - ":0")
А вот первым аргументом должен быть файл с описанием одного или нескольких компонентов настройки. Для того, чтобы одной командой загружать все необходимые компоненты настройки, давайте сначала составим файл с полным описанием всех компонентов, соответствующий вашей текущей конфигурации. Это совсем не сложно, но зависит от того - какой способ задания полной конфигурации используется у вас в XF86Config (см. "Настройка XKB")

(Надеюсь, что у вас используется один из способов "в чистом виде", а не "каша" из всех возможных инструкций).

"Первый способ"

Если у вас используется первый способ - перечислением необходимых компонентов (**keycodes, types, compat, symbols, geometry**).
Просто скопируем из файла XF86Config все инструкции типа **Xkb****** из секции "**Keyboard**" в наш файл. И слегка подправим.
Например, у вас там написано

```
  XkbKeycodes             "xfree86"
  XkbTypes    "default"
  XkbCompat "default"
  XkbSymbols              "us(pc104)+ru"
  XkbGeometry             "pc(pc104)"
```

Надо -

- во всех строчках заменить префикс **Xkb** на **Xkb_** ("case" букв менять не обязательно),
- вставить в каждую строчку инструкцию **include**,
- получившиеся инструкции i**nclude "..."** взять в фигурные скобки и закончить знаком "**;**",
- и, наконец, добавить "обрамление" - **xkb_keymap { .... };** .

Должно получится

```
xkb_keymap {
  Xkb_Keycodes            { include "xfree86" };
  Xkb_Types  { include "default" };
  Xkb_Compat              { include "default" };
  Xkb_Symbols             { include "us(pc104)+ru" };
  Xkb_Geometry            { include "pc(pc104)" };
};
```

Это и есть полное описание настройки XKB. Которое можно загружать в X-сервер, программой xkbcomp.
Все наши добавки мы можем "приплюсовывать" в соответствующие строчки этого описания.

"Второй способ"

Если у вас используется второй способ - указание полной **keymap**. В этом случае надо просто найти конкретную **keymap** и скопировать в наш файл.
Например, у вас в XGF86Config есть только строчка

```
XkbKeymap  "xfree86(ru)"
```

Она указывает на то, что полное описание лежит в файле **{XKBROOT}/keymap/xfree86**, в блоке "**ru**".
Находим этот файл. Находим в нем блок

```
xkb_keymap "ru" {
 ....
};
```

И "выкусываем" его оттуда. (Поскольку в нашем файле только один блок, название блока можно убрать).
Больше никаких исправлений не требуется.

"Третий способ".

Если у вас используется третий способ - через задание **"правил", "модели", "схемы"**.
В этом случае все немного сложнее, поскольку непосредственно **xkbcomp** не понимает этот способ.
Однако, в этом случае можно "вручную" выполнить преобразование правил/модели/схемы в компоненты настройки (**keycodes,symbols** и т.п.).
Например, у вас в файле конфигурации написано

```
XkbRules      "xfree86"
XkbModel      "pc104"
XkbLayout     "ru"
XkbOptions    "grp:shift_toggle"
```

Сначала надо найти файл **"правил"** (**rules**). Это будет файл **{XKBROOT}/rules/xfree86**.
В первой секции, которая после "шаблона"
```
! model = keycodes          geometry
```
по вашей модели - **"pc104"** находим название файлов (блоков) для **xkb_keycodes** и **xkb_geometry**. Скорее всего это будет
```
xkb_keycodes - "xfree86"
xkb_geometry - "pc(104)"
```
Теперь, во второй секции, после "шаблона"
```
! model       layout        =             symbols
```
найдем по **"модели" - "pc104"** и **"схеме" - "ru"** подходящий файл для **xkb_symbols**.
Скорее всего, схема **"ru"** там не упомянута. Но зато есть правило
```
pc104         *             = en_US(pc104)+%l%(v)
```
где **%l** надо "заместить" названием **"схемы"** (**layout**), а **%(v)** - названием **"варианта"**.
Поскольку "вариант" у вас не задан, то это правило "развернется" в
```
xkb_symbols - "en_US(pc104)+ru"
```
Следующая секция, после "шаблона"
```
! model       layout        =             compat        types
```
вообще очень простая
```
*             *             =             complete      complete
```
То есть, независимо от конкретных значений **model** и **layout**, и **xkb_compat**, и **xkb_types** надо брать из файлов **"complete"**.
Таким образом, для нашего файла полной конфигурации значения
```
xkb_types  - "complete"
xkb_compat - "complete"
```
И, наконец, последняя секция, после "шаблона"
```
! option      =             symbols
```
указывает, что для нашей **"опции" - grp:shift_toggle**, к уже выбранному файлу для **xkb_symbols** надо "приплюсовать" еще и блок "**group(shift_toggle)**"
Теперь не забудьте добавить слова **include**, скобки в нужном месте и "обрамление" **xkb_keymap { ... }**;
Должно получится
```
xkb_keymap {
  xkb_keycodes          { include "xfree86" };
  xkb_types     { include "complete" };
  xkb_compat { include "complete" };
  xkb_symbols           { include "us(pc104)+ru+group(shift_toggle)" };
  xkb_geometry          { include "pc(pc104)" };
};
```
Это и есть наша рабочая "полная конфигурация", к которой можно писать "добавки" - исправления/дополнения.
Наконец, надо заметить, что делать все это (и полное описание и фалы-добавки) вы можете в отдельной директории, поскольку **xkbcomp** при "разборке" **include** сначала ищет файла в текущей директории, а только потом в "стандартном" месте - **{XROOT}/lib/X11/xkb**. Естественно, подразумевается, что мы при экспериментах запускаем **xkbcomp**, находясь в этой директории.
А вот потом, если вы решите, что "это хорошо", можно будет разложить файлы с исправлениями в соответствующие поддиректории (**keycodes, types, symbols** и т.д) "домашней директории" XKB - **{XROOT}/lib/X11/xkb**. И подправить файл конфигурации X-сервера так, чтобы он при старте загрузил вашу конфигурацию.

Итак. Примеры изменения конфигурации XKB.

Новый тип для клавиши Enter.

Рассмотрим пример - для чего может понадобится новый тип. И что нужно сделать.
Итак. Проблема...
"По умолчанию" клавиша **Enter** относится к типу **ONE_LEVEL**, то есть выдает всегда один и тот же код (**CR**) при любых модификаторах.
Я же привык, что в "консольном режиме" эта же клавиша, нажатая вместе с **Control** меняет свое значение на **LF** (во всяком случае, так во FreeBSD).
Для того, чтобы и в XKB она вела себя так же, можно описать новый тип (сделаем для этого файл **my.types** и не забудем "приплюсовать" его в строчку, указывающую на файлы для **xkb_types**).

```
type "CTRL_DEPEND" {
  modifiers = Control;
  map[None] = Level1;
  map[Control] = Level2;
  level_name[level1] = "Base";
  level_name[level2] = "Control";
};
```

(В общем-то, имена уровней - **level_name** для работы XKB не нужны, но X-сервер очень не любит "неполные" определения.)
Соответственно, в описании **xkb_symbols** надо изменить определение для клавиши **Enter** (это скан-код **<RTRN>**).
Сделаем файлик **my.symbols** (не забудьте "приплюсовать" его куда надо) и поместим туда наше новое определение.
"Классическое" определение для **<RTRN>** (его можно найти найти, например, в файле **symbols/us**) выглядит как

```
key             <RTRN>      { [Return] };
```

Нам надо указать, что эта клавиша имеет тип **"CTRL_DEPEND"**, и добавить значение для появившегося второго "уровня" - **Linefeed** .

```
key             <RTRN>      { type="CTRL_DEPEND", [ Return, Linefeed ] };
```

Осталось перегрузить конфигурацию командой **xkbcomp**. И убедиться, что все работает.
Надо заметить, что уже запущенные приложения (например - **xterm**) этих изменений не почувствуют, поскольку все описания типов используются библиотекой Xlib и загружаются при старте приложения.
Поэтому запущенные приложения не заметят, что появился новый тип.
Но все, что запущено после перезагрузки конфигурации, должно почувствовать изменения.
А теперь настало время сказать, что для данной задачи, новый тип не требуется.
(Я же предупреждал, что мои примеры не являются "хорошим" или "правильным" решением :-).
Обычно в загружаемых "конфигах" уже есть подходящий тип - **"PC_BREAK"**. Он используется только для клавиши **Break**, но ничто не мешает "приписать" его и клавише **Enter**.
Поэтому достаточно составить только новое определение для скан-кода **<RTRN>**.

```
key             <RTRN>      { type="PC_BREAK", [ Return, Linefeed ] };
```

Добавляем новую "старую" раскладку клавиатуры.

Рассмотрим - как добавить еще одну группу с другой раскладкой клавиатуры.
Зачем это может понадобится?
Ну, например, проблема (описанная в "Почему руссификация не работает?") - у вас есть программы, в "бинарниках", статически слинкованные, которые напрочь отказываются понимать коды типа **Cyrillic_***.
Можно специально для них изготовить раскладку, в которой будут не двубайтные коды русских букв, а однобайтные коды **KOI8-R**.
Возможно, вам захочется добавить раскладку в кодировке **cp1251** или еще какой-нибудь, которая отличается от стандартной расположением русских букв.
(Надо заметить, что этот путь (добавление новой кодировки русских букв с помощью дополнительной группы) - в общем-то, плохое решение. Тем более, если вы не используете новую locale. Правильно было бы - добавить соответствующую таблицу перекодировки в Xlib и изготовить подходящую "иксовую" locale.
Но, как я уже сказал, я не предлагаю правильные решения :-), а только привожу примеры - как это можно сделать.)
Итак, давайте в этом примере сосредоточимся на задаче - добавление "однобайтной **koi8-r** кодировки" для "старых" или "тупых" клиентских программ.
Прежде всего, надо заметить, что у вас должны быть задействованы уже две группы. Первая - "латиница", вторая - "кириллица" с "правильными" кодами для русских букв (**Cyrillic_***).
Новую группу надо добавлять не "в конец" (как третью), а "в середину" - так чтобы она была второй, а "правильная" кириллица - третьей.
Почему? Потому, что "традиционные" программы (со старой Xlib) понимают только первую и вторую группу. Причем, вторую выбирают тогда, когда установлен модификатор, соответствующий символу **Mode_switch**. Ну, об установке модификатора позаботится "таблица" совместимости XKB. Если она у вас стандартная, то соответствующий модификатор будет выставляться для всех групп, кроме первой (то есть, в нашем случае и для "старой" кодировки и для "новой").

А вот искать символы "старые" программы будут всегда только во второй группе (об остальных они даже не подозревают). А "новые" программы, совместимые с XKB и так найдут свою раскладку, будь она во второй, третей, или даже в четвертой группе.
Еще одно замечание. Естественно, добавлять новые символы мы будем в **xkb_symbols**. При этом будет логично не писать ее "с нуля", а взять за основу уже существующий файл **symbols/ru** и дополнить его. Если мы наш файл "приплюсуем" к соответствующему описанию **xkb_symbols**, то у нас получится два фйла описания одних и тех же клавиш, при этом второй полностью переписывает первый.
Поэтому, логично, если мы из описания вообще выкинем "стандартный" файл "**ru**", а оставим только свой.
То есть соответствующая строчка в нашем "полном описании конфигурации" будет выглядеть не как
```
xkb_symbols { include "en_US(105)+ru+new-ru" };
```
а немного короче
```
xkb_symbols { include "en_US(105)+new-ru" };
```
Итак. Берем в свою директорию файл **symbols/ru** и начинаем его "корежить".
Надо заметить, что, скорее всего в нем вы обнаружите три блока -
```
xkb_symbols "toggle" {...};
xkb_symbols "shift_toggle" {...};
xkb_symbols "basic" {...};
```
Причем, реально расположение русских букв описывает только третий, а первые два просто добавляют два разных способа преключения "рус/лат".
Обычно, если у вас в полной конфигурации указан просто файл **ru**, загружается первый блок. И переключателем "рус/лат" становится клавиша **CapsLock**.
Во-первых, для нашей задачи это очень плохо (то, как описаны символы для этой конопки). Но об этом поговорим немного позже.
А сейчас я предлагаю просто выкинуть ("вычистить") два первых блока и оставить только блок "**basic**". А переключатель допишем потом прямо в блок "**basic**", или "приплюсуем" подходящий блок из файла **symbols/group** (в нем описано аж шесть разных способов переключения).
Итак. Выкинули два первых блока и начали исправлять/дополнять блок "**basic**".
Нам нужно для каждой кнопки, которая в описаниях содержит символы **Cyrillic**, дописать в середину (второй группой) еще одну группу с однобайтными символами в кодировке **koi8**. Напомню, что символы можно задавать не только символическими именами (типа **Cyrillic_***), а просто цифровым кодом. Например, клавишу
```
key <AB01> {        [                   z,                  Z       ],
                    [   Cyrillic_ya,    Cyrillic_YA         ]       };
```

мы должны описать как
```
key <AB01> {        [                   z,                  Z       ],
                    [       0xd1,       0xf1    ],
                    [   Cyrillic_ya,    Cyrillic_YA         ]       };
```
Естественно, первый код соответсвует маленькой букве, а второй - большой.
Как подобрать коды? Ну, во-первых, по названию букв можно догадаться - какую русскую букву они имеют ввиду и, если у вас есть под рукой табличка - какой русской букве, какой код **koi8-r** соответствует, просто переписать оттуда.
А во-вторых, могу подсказать, что младший байт кода **Cyrillic** на самом деле соответствует коду этой буквы в **koi8**, а в старшем байте всегда шестерка.
Поэтому, можно взять файл, в котором описываются числовые значения для кодов типа **Cyrillic_*** - это файл **/usr/X11R6/include/X11/keysymdef.h**. И списать соответствующие коды оттуда, отбрасывая первую шестерку.
Особо ленивые могут взять готовый файл здесь.
Итак, мы составили новый файл описания клавиатуры, в котором теперь три группы. Надо не забыть о переключателе меджу группами.
Во-первых, надо заметить, что все варианты переключателей используют для своих целей специальный символ - **ISO_Next_Group**, а его семантика, описанная в **xkb_compat** такова, что он просто перебирает все возможные группы. То есть, при нажатии клавиши (или комбинации клавиш) с таким символом просто текущее значение группы увеличивается на единицу, а когда счетчик доходит до последней группы, он просто возвращается на первую (см. "Внутренности":"Методы выравнивания номера группы").
Таким образом тем же самым перключателем "рус/лат" мы можем последовательно перебирать все три группы.
Вы можете выбрать ваш любимый способ переключения из файла **symbols/group** и "приплюсовать" его к описанию **xkb_symbols**, например,
```
xkb_symbols { include "en_US(105)+new_ru+group(shift_toggle)" };
```
Только одно замечание о переключении клавишей **CapsLock**.
Дело в том, что традиционно на эту же клавишу "подвешивают" и символ **Caps_Lock**, чтобы она могла выполнять и свою основную функцию (нажатая с Shift'ом).
Проблема в том, что к этому символу присоединен реальный модификатор **Lock**. При этом, в конечном счете, XKB привязывает реальные модификаторы не к символу, а к скан-коду клавиши. Поэтому при нажатии этой клавиши в "состоянии модификаторов" появится не только модификатор, который указывает

на то, что выбрана альтернативная группа, но и реальный модификатор **Lock** (хотя вы нажимаете клавишу как **ISO_Next_Group**, а не как **Caps_Lock**).
В результате, клиентская программа увидит, что вы не только выбрали альтернативную группу, но "намертво" прижали **Shift** (хотя **Shift** должен отменять действие **Lock**, но... почему-то не работает).
Естественно, при этом жми, не жми **Shift** - у вас всегда будут получаться только маленькие (или только большие буквы).
Для того, чтобы этого не происходило, надо бы "отцепить" реальный модификатор **Lock** от этой клавиши. К сожалению, "привязка" модификатора к символу **Caps_Lock** "зарыта" глубоко в файлах, которые "инклюдятся" в **en_US**. А отменить это присвоение в дополнительных файлах уже нельзя.
Поэтому, чтобы не "перелопачивать" все файлы, которые неявно включаются в нашу полную конфигурацию, лучше просто убрать символ **Caps_Lock** из описания клавиши **<CAPS>**.
Если вам жалко расставаться с этой функцией - "подвесьте" ее на какую-нибудь другую клавишу. А если вы используете для "рус/лат" другой способ - то и описанной проблемы у вас не будет.
Итак. Если ваш любимый способ переключения - клавиша **CapsLock**, то последнее, что нам надо сделать - не "приплюсовывать" этот способ из файла **symbols/group** (там эта клавиша с кодом **Caps_Lock**), а просто вписать в нашу новую раскладку определение для клавиши **<CAPS>** -

```
key <CAPS> { [ISO_Next_Group] };
```

Теперь можно прегрузить конфигурацию программой **xkbcomp** и посмотреть результат.
Кстати, забавно, что "старые" программы теперь работают когда у вас включена и вторая группа и третья. Потому, что они в обоих случаях видят в "состоянии модификаторов" модификатор, который указывает, что включена альтернативная группа, а коды символов всегда берут из второй группы. То есть, для них не заметна разница между двумя состояниями XKB (включена вторая или третья группы).
Интересно, что некоторые "новые" программы, например - xterm, тоже правильно работают с обоими группами. Потому, что... фиг его знает - почему :-)
Единственное неудобство - сложное переключение групп (особенно, если вы не пользуетесь никаким индикаторами перключения групп). Очень непривычно, когда переключатель "рус/лат" вдруг обретает не два, а три состояния.
А вот о том, как сделать переключение между тремя (и больше) группами более приятным, мы рассмотрим в следующем примере -

"Вариации на тему" - переключатели "рус/лат" (и еще раз - "рус").

"Вариации на тему" переключатели "рус/лат" (и еще раз - "рус").

Предположим, что вы выполнили предыдущий пример или еще откуда-нибудь взяли (сделали) раскладку клавиатуры с тремя группами.
И вам не нравится способ переключения между группами - три положения у одного переключателя "лат/рус/рус".

Что можно сделать?

Ну, во-первых, давайте вообще забудем пока про символ **ISO_Next_Group**, к которому "прицеплено" такое неудобное "действие". (Вообще-то, семантику **ISO_Next_Group** можно и переделать, но пока отложим этот вопрос).
Для простоты рассмотрения будем считать, что переключение у вас делается одной клавишей (обычно - это **CapsLock**),а не комбинацией клавиш (типа - **Shift+Shift** или **Alt+Shift**).

Напомню, что непосредственно в описании клавиши можно указать "действия", причем, разные для разных групп.
Кстати. Напомню, что, во-первых, большинство изменений мы будем делать в файле типа **xkb_symbols**. А, во-вторых, поскольку эти изменения не просто дополняют существующие определения клавиш, а изменяют их радикально, то лучше всего добавлять их в "общую конфигурацию" не "приплюсовыванием", а отдельной инструкцией **replace**, например -

```
xkb_symbols {               include "en_US(pc104)"
                            replace "new.symbols" };
```

(Вообще-то, можно "способ добавления" **replace** указывать в файле перед каждой инструкцией. Но это, почему-то, не всегда срабатывает.)

Первый способ - простой и неудобный.

Самый тривиальный способ - сделать две клавиши-переключатели, каждая с двумя состояниями. Одна клавиша переключает **"лат./старый рус."**, другая - **"лат/новый рус."**. Или в терминах "номер группы", первая клавиша переключает **"group1/group2"**, другая - **"group1/group3"**.
Выберем два скан-кода и "подвесим" на них "действия" -

```
key <...> { actions[Group1]= [ LockGroup(group=2) ],
                    actions[Group2]= [ LockGroup(group=1) ] };

key <...> { actions[Group1]= [ LockGroup(group=3) ],
                    actions[Group3]= [ LockGroup(group=1) ] };
```

Конечно, это - неполное описание. Надо добавить к первой клавише ее "поведение", когда включена группа 3, на тот случай, если мы нажмем ее в ситуации, когда с помощью второго переключателя уже выбрана третья группа. И, соответственно, "поведение" второй клавиши в состоянии "группа 2".
Кроме того, в описании должна быть кроме таблицы "действий" еще и таблица символов. В нашем случае можно использовать специальный "псевдосимвол" - **NoSymbol**.
Тогда полное описание будет выглядеть как

```
key <...> { [NoSymbol],[NoSymbol],[NoSymbol],
                    actions[Group1]= [ LockGroup(group=2) ],
                    actions[Group2]= [ LockGroup(group=1) ],
                    actions[Group3]= [ LockGroup(group=1) ] };

key <...> { [NoSymbol],[NoSymbol],[NoSymbol],
                    actions[Group1]= [ LockGroup(group=3) ],
                    actions[Group2]= [ LockGroup(group=1) ],
                    actions[Group3]= [ LockGroup(group=1) ] };
```

Но, на мой взгляд, такой способ переключения еще более неудобный. (Я же не обещал "хороших" решений :-).
Поэтому, я его даже не пробовал и вам не предлагаю.

Второй способ (через модификатор).

Более удобным мне представляется способ, когда наш привычный перключатель (пусть это будет **CapsLock**) продолжает переключать "лат/рус". А вот - какой именно "рус" (вторую или третью группу) он выберет - будет определяться другой клавишей.
Естественно, при этом каждый из переключателей имеет два состояния. Основной переключатель - "лат/рус", дополнительный - "группа2/группа3".
Другими словами, "поведение" основного переключателя должно меняться в зависимости от того, нажимали ли мы дополнительный переключатель (и сколько раз - четное или нечетное количество).
Собственно эти два "поведения" я уже написал в предыдущем примере. Только там были две разные клавиши, а нам надо сместить это на одной. Кстати, можно заметить, что эти два описания отличаются только в одной строчке - "переход из состояния Group1", остальные части описания совпадают.
Итак. Подзадача - как сделать так, чтобы "поведение" клавиши менялось в зависмости от состяния какой-нибудь другой.
Первое, что приходит в голову - с помощью какого-нибудь модификатора. Напомню, что каждая группа может делиться на уровни (**shift level**), а выбор конкретного уровня зависит от состояния модификаторов. Причем, эту зависимость мы можем как угодно менять, сочиняя новые "типы" клавиш.
Все, что нам надо - взять какой-нибудь модификатор (виртуальный), сочинить "тип", в котором переход на второй уровень зависит только от данного модификатора (чтобы наш основной перключатель реагировал только на этот модификатор) и, наконец, разделить у нашего переключателя (основного) первую группу на два уровня и "присвоить" им разные действия.
Естественно, "установку/сброс" модификатора "повесить" (с помощью соответствующих "действий") на второй переключатель.
Итак. Описание основного переключателя теперь будет иметь вид

```
key <...> { type[Group1]=".....",
                [NoSymbol, NoSymbol], [NoSymbol], [NoSymbol],
                actions[Group1]= [ LockGroup(group=2), LockGroup(group=3)],
                actions[Group2]= [ LockGroup(group=1) ],
                actions[Group3]= [ LockGroup(group=1) ] };
```

("тип" придумаем немного позже).

Теперь надо "сочинить" модификатор и новый тип. И здесь нас ждет "засада". Дело в том, что **xkbcomp** (по крайней мере в нынешнем состоянии) не позволяет объявить новый виртуальный модификатор. Можно использовать только те, которые в нем уже предопределены.

К счастью, среди его модификаторов уже есть подходящий - **LevelThree**. А подходящий он потому, что

- во-первых, он практически не используется (по крайней мере в "русских" конфигурациях),
- а во-вторых, есть уже подходящий тип, с использованием этого модификатора, и нам не придется его описывать.

Таким образом, берем модификатор **LevelThree** и тип **"THREE_LEVEL"** -

```
type "THREE_LEVEL" {
                modifiers = Shift+LevelThree;
                map[None] = Level1;
                map[Shift] = Level2;
                map[LevelThree] = Level3;
                map[Shift+LevelThree] = Level3;
                level_name[Level1] = "Base";
                level_name[Level2] = "Shift";
                level_name[Level3] = "Level3";
};
```

Напомню, что он уже должен быть в нашей "полной конфигурации" по умолчанию. Только обратите внимание, что он определяет не два, а три уровня. Причем, "наш" модификатор "посылает" на третий уровень. Но ничего страшного в этом нет. Мы просто сдвинем второе "действие" на третий уровень, а второй можно заполнить "заглушкой" - **NoAction()** или сделать его таким же как и первый уровнь (обратите внимание - второй уровень, это - когда прижат **Shift**).

Таким образом, наш основной переключатель (давайте уже определимся, что это будет **<CAPS>**), приобретает вид

```
key <CAPS> { type[Group1]="THREE_LEVEL",
                [NoSymbol, NoSymbol, NoSymbol], [NoSymbol], [NoSymbol],
                actions[Group1]= [ LockGroup(group=2), NoAction(), LockGroup(group=3)],
                actions[Group2]= [ LockGroup(group=1)],
                actions[Group3]= [ LockGroup(group=1)] };
```

Теперь можно заняться второй клавишей, которая будет "поднимать/опускать" модификатор **LevelThree**. Конечно, это может быть не одиночная клавиша, а какая-нибудь комбинация клавиш. Но для простоты, сначала возьмем отдельную клавишу. Я на своей клавиатуре ("Микрософтовской") использовал для этого клавишу "**Menu**" (скан-код **<MENU>**). Но, если у вас такой клавиши нет, то можно задействовать одну из парных клавиш (**Shift, Control, Alt**).

Итак. Сочиняем "поведение" клавиши **<MENU>**.

Прежде всего - я думаю, что вам не хочется держать ее все время нажатой, пока это нужно. То есть, нам надо, чтобы по первому нажатию она "подняла" модификатор, а по второму - опустила.

Как раз это делает "действие" **LockMods** (в переменной **locked modifiers**). Напомню, что **SetMods** и **LatchMods** "держат" модификаторы только пока нажата клавиша.

Итак, используем "действие" **LockMods** -

```
key <MENU>              { [ NoSymbol ],
                          actions[Group1]=[ LockMods(modifiers=LevelThree) ]};
```

(Описывать все три группы не обязательно. Если номер группы больше, чем допустимо для клавиши, XKB будет его "выравнивать", пока не попадет в допустимый диапазон.)

Теперь остается одна тонкость. Дело в том, что сам по себе виртуальный модификатор работать не будет, если он не связан с каким-нибудь реальным модификатором. Напомню, что при общении модуля XKB с процедурами Xlib, передается только "эмулируемый набор модификаторов" (то есть - реальные модификаторы). И, хотя выполнение "действий" происходит внутри модуля XKB, а не в Xlib, чтобы не возникало "разногласий" между XKB и Xlib по поводу "состояния", необходимо "связать" виртуальный модификатор с каким-нибудь реальным.

Поэтому, последним шагом будет - связать с клавишей какой-нибудь реальный модификатор (с помошью объявления **modifier_map**). И вот здесь нас ждет вторая "засада". В полной конфигурации все модификаторы (даже "безымянные" - **Mod1-Mod5**) уже расписаны. То есть, каждый уже соединен с какой-нибудь клавишей и, соответственно, каким-нибудь символом.

Самое неприятное то, что клиентские приложения могут реагировать на эти модификаторы и менять свое поведение. Поэтому, если мы свяжем наш **LevelTree**, например, с модификатором **Mod5**, который

соответствует символу **Scroll_Lock**, то, при "поднятии" модификатора **LevelThree**, приложения будут считать, что нажата кнопка **ScrollLock**. А вот как они будут вести себя при этом - кто как.
Здесь надо немного пояснить - как реагируют программы на "безымянные" модификаторы. Поскольку этими модификаторам явно никакая функция не соответствует, программы ориентируются на символы, которые с ними связаны.
То есть, если программа обнаружит, что в "состоянии" модификаторов взведен модификатор **Mod1**, она попытается найти соответствующий ему символ. Обычно это - **Alt_L** и **Alt_R**, то есть любая из клавиш **Alt**. Соответственно, программа считает, что нажата клавиша **Alt** и ведет себя в соответствии со своим пониманием этой клавиши.
Поэтому, чтобы избежать всяких "побочных" эффектов, нам надо не только связать модификатор с нашей клавишей **<MENU>**, но и "отцепить" этот реальный модификатор от других. Как я уже говорил в примере "добавление новой группы", "отцепить" модификатор, который был объявлен где-то "в глубинах" **includ'ов**, очень трудно. но, к счастью, они там привязываются не к скан-кодам, а к символам. Поэтому, если мы в своей конфигурации просто уберем эти символы из описаний скан-кодов, то связка "модификатор-символ" просто "повиснет в воздухе" и таким образом нам удастся "отцепить" модификатор.
Итак. Давайте возьмем реальный модификатор **Mod5**. Он используется только для символа **Scroll_Lock**, а этот символ, в свою очередь, "подвешен" только на клавишу **ScrollLock** (скан-код **<SCLK>**). Да и клавиша эта не самая "часто используемая".
Теперь нам надо добавить в описание клавиши **<MENU>** виртуальный модификатор **LevelThree**. Добавить инструкцию **modifier_map**, который свяжет **Mod5** с нашей клавишей. И, наконец, переопределить клавишу **<SCLK>**, чтобы в ней не было упоминания и символе **Scroll_Lock**.

```
key <MENU>              { virtualMods = LevelThree,
                          [ NoSymbol],
                          actions[Group1]=[ LockMods(modifiers=LevelThree) ]};

modifier_map Mod5 { <MENU> };

key SCLK { [NoSymbol] };
```

Вот теперь можно пробовать - что получилось.
Конечно, это не самое лучшее решение.

- Во-первых, из-за того, что мы заняли реальный модификатор **Mod5** (кстати, и отмена **ScrollLock** не во всех случаях спасает).

- Во-вторых, наш дополнительный переключатель сам группы не меняет, а только "машет флагом". Поэтому, если переключение "из лат. в рус" работает всегда, то для переключения "из одного рус. в другой рус." надо не только нажать дополнительный преключатель, но и "щелкнуть" основным.

- Ну и, наконец, если мы захотим задействовать все четыре группы, то решение будет еще сложнее (понадобится как минимум два виртуальных модификатора, с которыми и так "напряженка").

Поэтому рассмотрим еще один способ -

Третий способ (через дополнительную переменную номера группы).

Вспомним еще раз - в чем проблема. В том, что основной переключатель должен переходит из одного состояния (текущая группа - Group1), в два разных, в зависимости от состояния какой-то другой клавиши. Но ведь можно "запомнить" состояние дополнительного переключателя нет только с помощью модификатора.
Напомню, что номер группы может храниться в трех внутренних переменных XKB - **locked, latched** и **base group**, значение которых можно менять независимо. Причем для выбора символа у клавиши используется суммарное значение этих переменных - **effective group**.
Напомню также, что благодаря "методу выравнивания номера группы" (см. "Внутренности":Метод выравнивания..."), если суммарное значение окажется больше, чем количество существующих групп на единицу, то опять получится первая группа и т.д.
Таким образом, можно заставить дополнительный переключатель "запоминать" номер "альтернативной" группы в дополнительной "групповой" переменной, например - **base group**. А основной переключатель пусть манипулирует значением в другой переменной, например - **locked group**.
Итак. Пусть дополнительный переключатель запоминает - какая из "русских" групп нам требуется (2 или 3), например, в переменной **base group**.

Тогда основной переключатель, чтобы перейти из "лат" в из "рус" раскладок должен просто обнулить **locked group**. А для того, чтобы вернуться обратно в "лат", надо в **locked group** записать "добавку", достаточно большую, чтобы "метод выравнивания" "завернул" значение обратно к "лат" раскладке.
Таким образом мы нашу проблему (переход из Group1 в два разных состояния) решаем. Правда теперь у на появляется проблема - как вернуться обратно из разных состояний в одно и ту же группу. Но она то, решается очень просто. Поскольку клавиатура при этом находится в разных состояниях (Group2 и Group3), то мы без проблем "подвешиваем" на клавишу два разных "действия", каждое со своим значением "добавки".
Прежде чем переходить написанию конфигурации клавишь, давайте проясним вопрос: что значит "обнулить" переменную с номером группы, и - какие "добавки" нам понадобятся.
Надо сказать об одной тонкости - хотя в "конфигах" группы нумеруются - 1,2,3..., внутри XKB это означает - 0,1,2...
То есть, group1 - на самом деле 0, group2 - 1 и т.д.
Итак. Давайте сначала рассмотрим задачу во внутренних значениях XKB (0,1,2).
Тогда у нас

- первая группа ("лат") - 0
- вторая группа ("старый рус.") - 1
- третья группа ("новый рус.") - 2
- максимальный номер группы - 2

При этом

- если к 1 добавить 2 - получится снова 0 (3 на единицу больше, чем "максимальный номер группы")
- аналогично, если к 2 добавить 1 - снова вернемся к группе 0.

Тогда алгоритм работы переключателей мог бы быть таким -

- в дополнительной переменной (**base group**) запоминается номер одной из русских раскладок - то есть, 1 или 2;
- для того, чтобы перейти из состояния, когда текущая группа "лат", в выбранную "рус", надо просто в **locked group** записать 0;
- а для того, чтобы вернуться в "лат" из состояния "рус." надо в переменную **base group** записать -
    - 2, если текущая группа 1;
    - 1, если текущая группа 2.

Однако, не все так просто. Если мы попытаемся воплотить это в конфигурацию клавиши, то нарвемся на очередную "засаду".
Дело в том, что значения номера группы "запоминаются" только в **locked group**. В других переменных (**base** и **latched**) они "держаться" только пока соответствующая клавиша нажата и "испаряются" оттуда, если клавишу отпустить.

Правда, у нас есть одна "лазейка". Мы можем изменить поведение клавиши (той, что будет менять **base group**) и сделать ее "залипающей". Тогда XKB при первом нажатии/отжатии клавиши выполнит только ту задачу, которая выполняется при нажатии клавиши, а при повторном нажатии/отжатии наоборот - только то, что выполняется при отжатии клавиши.
Только обратите внимание, что когда мы ее все-таки "отожмем" там получится 0. То есть, мы не сможем держать там "2 или 3", а только "2 или 0", "1 или 0" и т.п.
Ну и ладно. Придется в ней держать не "номер алтернативной раскладки", а только "добавку" к номеру. То есть - не "2 или 3", а "0 или 1".
Тогда нам придется слегка подправить наш алгоритм -

- в дополнительной переменной (**base group**) запоминается "смещение" от первой "русской" раскладки - 0 или 1;
- для того, чтобы перейти из состояния "лат", в нужную "рус", надо в **locked group** записать 1;
- для того, чтобы вернуться в "лат" из состояния "рус." надо в переменную **base group** записать -
    - если текущая группа 1 (в base 0) - 0;
    - если текущая группа 2 (в base 1) - 2.

Теперь, переходя к обозначениям из "конфигов" (**group1, group2** и т.д.), получим -

- дополнительный переключатель:
    - ну, 0 или **Group1** там получится автоматически при "отжатии" клавиши
    - "запоминать" в **base group** надо **Group2**,
- основной переключатель:
    - "действие" для **Group1** - записать в **locked group** значение **Group2**
    - для **Group2** - записать **Group1**
    - для **Group3** - записать **Group3**

Сочиняем описание клавиш (пусть это будут те же **<CAPS>** и **<MENU>**)

```
key <CAPS> { [NoSymbol],[NoSymbol],[NoSymbol],
                actions[Group1]= [ LockGroup(group=2) ],
                actions[Group2]= [ LockGroup(group=1) ],
                actions[Group3]= [ LockGroup(group=3) ] };

key <MENU>              { [ NoSymbol ], locks= yes
                        actions[Group1]=[ SetGroup(group=2) ]};
```

(обратите внимание - мы для дополнительного переключателя опять описываем только одну группу. Все равно он может "перещелкивать" только два состояния, и делать его "чувствительным" к текущей группе нет смысла.)
Можно пробовать.

Правда, надо признаться, что этот метод тоже не лишен недостатков. Если вы начнете менять альтернативную раскладку, когда основной переключатель стоит в положении **"какой-нибудь рус."** - все будет нормально.
Но вот если "пощелкать" им, когда основной переключатель стоит в состоянии **"лат"**, то результаты будут несколько "странные" (вы можете сами их "просчитать"), поскольку "состояние лат." на самом деле - "хитро" подобранная сумма двух переменных (**base** и **locked**).
С другой стороны, с этим можно примириться. Скорее всего в реальной работе вы будете "подгонять" русскую раскладку, когда основной переключатель стоит в сотоянии **"рус"**, а не готовить ее заранее (когда основной переключатель в состоянии **"лат"**).
Замечу также, что это не единственный вариант переключения "с помощью двух переменных group". Можно, например, сделать так, чтобы дополнительный переключатель работал с **locked group**, а основной - с **base group** (если посмотреть исходный алгоритм, то там у основного переключателя есть состояние когда он "держит" в "своей" переменной 0). Но этот вариант не лучше в смысле "побочных эффектов", а в описании, пожалуй, сложнее.
Поэтому, пока не этом и остановимся.
Тем более, что у нас остался еще -

"Заключительный аккорд" - "отцепляемся" от скан-кодов.

Как я уже говорил - не очень хорошо, когда "действия" назначаются прямо в описании клавиши (в файлах **xkb_symbols**).
Во всяком случае, при этом трудно распространить наше решение на случай, когда роль основного или дополнительного переключателей (или обоих) выполняет не одиночная клавиша, а комбинация клавиш.
Обычно "хорошим тоном" является - привязать "действия" к каким-нибудь сиволам (через "интерпретации"), а в описании клавиш использовать только символы.
Давайте проделаем это для любого из примеров, например, последнего.
Единственная проблема - надо подобрать подходящии символы. Дело в том, что брать коды обычных символов (даже "экзотических" - типа "умлаутов") не очень хорошо. Потому, что при нажатии клавиши XKB будет не только выполнять соответствующие "действия", но и выдавать в прикладную программу эти символы. А программа, соответственно, будет пытаться их напечатать.
К счастью, в наборе символов есть группа кодов, которые должны использоваться только для внутренних нужд XKB и игнорироваться прикладными программами. Вы можете найти их в файле **{XROOT}/include/X11/keysymdef.h** их названия начинаются на **XK_ISO_**. Например,

```
XK_ISO_First_Group
XK_ISO_First_Group_Lock
XK_ISO_Last_Group
XK_ISO_Prev_Group_Lock
```

и т.п.
Конечно, там нет символов из названия которых следует, что по этому символу **"записать в locked group число 3"**. Но мы можем изменить семантику любого из символов (тем более, что и не для каждого из этих символов задана семантика в "конфигах" XKB).
Давайте сначала выясним - сколько нам надо символов. Для этого посчитаем - сколько у нас различных действий в описании обоих переключателей.

```
LockGroup(group=1)
LockGroup(group=2)
LockGroup(group=3)
SetGroup(group=2)
```

Итого - 4 штуки.
Давайте подберем им символы. Конечно, можно взять любые из тех о которых говорилось выше. Но, чтобы лучше в них ориентироваться, подберем такие, чтобы их название хоть немного походили на соответствующие действия.
Обратите внимание, чтов названиях присутствуют - **First_Group, Last_Group, Next_Group** и **Prev_Group**.
Давайте считать, что для нашего случая

- **First_Group** - group=**1**
- **Next_Group** - group=**2**
- **Last_Group** - group=**3**

Тогда логично выбрать

- для **LockGroup(group=1) - ISO_First_Group_Lock**
- для **LockGroup(group=2) - ISO_Next_Group_Lock**
- для **LockGroup(group=3) - ISO_Last_Group_Lock**
- для **SetGroup(group=2) - ISO_Group_Lock**

Составим соответствующие "интерпретации". Напомню, что это нужно делать в файле типа **xkb_compat** и, соответственно, "приплюсовать" это файл к строчке в "полной" конфигурации, которая описывает именно **xkb_compat** (а не **xkb_symbols**).
Итак, наша "добавка" к **xkb_compat** (не забудьте ее "приплюсовать" куда надо) -

```
xkb_compat {
  interpret ISO_First_Group_Lock { action=LockGroup(group=1); };
  interpret ISO_Next_Group_Lock  { action=LockGroup(group=2); };
  interpret ISO_Last_Group_Lock  { action=LockGroup(group=3); };
  interpret ISO_Group_Lock       { action=SetGroup(group=2); locking = True;};

  group 2 = AltGr;
  group 3 = AltGr;
};
```

(Последнии две инструкции, "по идее" не нужны, поскольку уже должны быть в "общей конфигурации". Но почему-то они иногда "теряются".)
А описание символов (в **xkb_symbols**) теперь будут выглядеть как

```
key <CAPS> {                [ ISO_Next_Group_Lock ],
                            [ ISO_First_Group_Lock ],
                            [ ISO_Last_Group_Lock ] };

key <MENU>                  { [ ISO_Group_Lock ] };
```

Можно пробовать.
На этом наши эксперименты на тему переключателей не заканчиваются.
Дальше мы рассмотрим еще более сложный случай (и другие механизмы) -

Еще несколько "переключателей".

Еще несколько "переключателей".

Предположим, что у вас раскладка с четырьмя группами (больше вам не позволит XKB).
Причем, опять же, первая группа - "латиница". А остальные три - варианты "русской" раскладки (например - **koi8, alt, cp1251**).
И опять же, нам хочется, чтобы основной переключтель имел только два состояниия - **"лат"/"какой-нибудь рус"**.
Попробуем придумать какой-нибудь переключатель для выбора одной из трех "альтернативных" раскладок.
Понятно, что одним переключателем с двумя состояниями мы тут не обойдемся (хотя можно сделать один с тремя состояниями, но я его рассматривать не буду).
Давайте поступим радикально - выберем три кнопки, и пусть каждая включает свою группу.

Как это можно сделать?

- Вариант первый - "выделенные скан-коды".
- Вариант второй - "перекрытия".

Прежде всего, хочу заметить, что я не могу предложить вам раскладку с четырьмя группами (откуда я знаю - что вы в них хотите "вложить").

Но если вам надо только поэкспериментировать, можно взять одну из кнопок (не обязательно "буквенную") и "навесить" на нее четыре группы.
Я сделал такую кнопку из клавиши "1".

```
key <AE01> {        [   1, exclam ],
                    [   2, exclam ],
                    [   3, exclam ],
                    [   4, exclam ]  };
```

(значение клавиши отражает текущий номер группы :-).
Итак.

Вариант первый - "выделенные скан-коды".

Возьмем за основу метод, рассмотренный в предыдущих примерах - две переменные для номера группы.
Пусть у нас дополнительная переменная (например - **base group**) хранит номер альтернативной группы.
Забудем пока про "гнусное поведение" base group (что она хранит значение только пока нажата соответствующая клавиша).
Тогда пусть основной переключатель манипулирует значением в **locked group**.
Он должен

- для выбора **"одной из рус"** просто "обнулить" **locked group**
- для "возвращения" в **"лат"** записать в **locked group** подходящую "добавку".

Для вычисления "добавки" на время перейдем к "внутренним представлениям номера группы" - 0,1,2...
Тогда

- группа "лат" - 0
- первая "рус" - 1
- вторая "рус" - 2
- третья "рус" - 3
- максимальное значение номера группы - 3

"Добавки"

- если текущая группа - 0, "добавка" - 0
- если текущая группа - 1, "добавка" - 3
- если текущая группа - 2, "добавка" - 2
- если текущая группа - 3, "добавка" - 1

Возвращаясь к обозначениям групп из "конфигов", описание клавиши - основного переключателя получится

```
key <CAPS> {        [NoSymbol], [NoSymbol], [NoSymbol], [NoSymbol],
                    actions[Group1]=[ LockGroup(group=1) ],
                    actions[Group2]=[ LockGroup(group=4) ],
                    actions[Group3]=[ LockGroup(group=3) ],
```

```
                actions[Group4]=[ LockGroup(group=2) ] };
```

Все хорошо. Но что нам делать с дополнительным переключателем (переключателями)?
Давайте сформулируем - как должны работать наши переключатели.
Итак, у нас три кнопки, каждая включает свою **"рус"**. Поскольку они должны сохранять соответствующие значения в **base group**, мы должны повесить на них "действия" **SetGroup** (с подходящим аргументом).
Естественно, поскольку **base group** "держит" значение только пока нажата соответствующая клавиша, наши кнопки должны быть "залипающими". Но при этом...
Если мы нажали кнопку, ответственную за первую "рус", она должна записать свое значение в **base group** и "залипнуть". Если мы после этого нажмем кнопку, отвечающую за вторую "рус", она должна

- "отлепить" "залипшую" кнопку
- выполнить свое "действие"
- и, в свою очередь, "залипнуть".

Аналогично должна работать и третья кнопка (из дополнительных переключателей).
Но то, что я только что описал, точь в точь описывает поведение кнопок, принадлежащих одной "радио-группе".
Ну и отлично! Объединим наши три дополнительных переключателя в радио-группу, подвесим на них **SetGroup** с соответствующими аргументами, и... все.
Однако, "засада" в этом решении (как же без "засады"?) в том, что "залипает" именно клавиша (скан-код), а не отдельное ее "действие" (или символ). То есть, нам понадобятся три кнопки, которые будут использоваться только для этих целей и ни для чего больше, поскольку, даже если мы и "подвесим" на них еще какие-нибудь символы (ну, например, на другой **shift level**), то воспользоваться ими все равно не сможем. Так как -

- кнопка будет "залипать" в любом случае, независимо от того - какая группа и уровень выбраны;
- если уж она "залипла", то использовать ее, пока не "отлипнет", невозможно.

Итак. Нам придется выделить на клавиатуре три кнопки, которые будут только переключателями "альтернативных" "русских" групп и ничем больше.
(Не огорчайтесь, потом мы это ограничение снимем :-)
Я предлагаю взять кнопки **F10, F11, F12**. Они стоят рядом и используются сравнительно редко.
Можно составлять их описание (это скан-коды **<FK10>,<FK11>,<FK12>**)

```
key <FK10> {        radiogroup=2, [NoSymbol],
                    actions[Group1]=[ SetGroup(group=2) ] };
key <FK11> {        radiogroup=2, [NoSymbol],
                    actions[Group1]=[ SetGroup(group=3) ] };
key <FK12> {        radiogroup=2, [NoSymbol],
                    actions[Group1]=[ SetGroup(group=4) ] };
```

Обратите внимание, что номер радио-группы должен быть больше единицы. Вообще-то, это "баг" XKB, тем более неприятный, что **xkbcomp** не будет "возражать" если вы объявите радио-группу номер 1, а вот X-сервер потом "сдуреет".
Теперь можно пробовать.
Надо заметить, что есть еще одно маленькое неудобство. В самом начале, пока еще не нажата ни одна из кнопок радио-группы, получается, что ни одна из альтернативных "рус" раскладок не выбрана и основной переключатель не будет ничего переключать.
Но после первого же нажатия одного из "дополнительных переключателей" все будет работать как задумано.
(Если не считать того, что опять же - менять номер альтернативной группы надо только тогда, когда основной переключатель находится в положении "рус". Иначе результаты получаются не очень приятные).

Вариант второй - "перекрытия".

Конечно, предыдущий вариант имеет очень серьезный недостаток - три клавиши "изымаются из общего пользования".
Давайте, все-таки попытаемся вернуть их обратно.

Эта проблема легко решилась бы, если бы могли свободно добавлять на свою клавиатуру новые кнопки (не символы, а именно - физические кнопки).
Хотя это и звучит как пустые фантазии, но не лишено смысла. Конечно, новую физическую кнопку мы на клавиатуре не сделаем, но вот скан-кодов к существующим кнопкам добавить можем.
Напомню, что если кнопка принадлежит к "группе перекрытия" (**overlay**), она может эмулировать нажатие другой кнопки (с другим скан-кодом), естественно, эта другая кнопка не обязана физически присутствовать на клавиатуре и може быть вполне "виртуальной". К тому же, та кнопка, которая будет эмулировать "виртуальную" не обязана заниматься этим все время. Для того, чтобы он перешла в такой режим, нужно "поднять" "управляющий флаг" **Overlay1** (или **Overlay2**), а все остальное время она может ваполнять свои "основные обязанности".
В свою очередь, "управляющий флаг" можно "выставить" с помощью специального "действия" (такого же как **SetGroup** или **SetMods**), причем можно сделать так, чтобы этот флаг "держался" только пока нажата соответствующая кнопка.
Итак. Давайте сначала объявим скан-коды "виртуальных" кнопок. Заглянем в файл типа **xkb_keycodes** (скорее всего у вас это **keycodes/xfree86**).
Можно заметить, что под реальные кнопки задействованы скан-коды от 9 до 120 (ну, на самом деле - даже меньше, в зависимости от типа клавиатуры).
То есть, мы для своих нужд можем спокойно взять скан-коды, например - 121, 122, 123.
Сочиним файл-"добавку" к описанию **keycodes** (не забудьте его "приплюсовать" к полной конфигурации в строчку, описывающую **xkb_keycodes**).

```
xkb_keycodes {
  <RUS1> = 121;
  <RUS2> = 122;
  <RUS3> = 123;
};
```

(напомню, что "название скан-кода" может быть произвольное, но не длинее 4 символов).
Теперь мы можем наши "действия" убрать с реальных кнопок **<FK10> - <FK12>** и "повесить" на наши виртуальные (просто поменяем названия скан-кодов) -

```
key <RUS1> {        radiogroup=2, [NoSymbol],
                    actions[Group1]=[ SetGroup(group=2) ] };
key <RUS2> {        radiogroup=2, [NoSymbol],
                    actions[Group1]=[ SetGroup(group=3) ] };
key <RUS3> {        radiogroup=2, [NoSymbol],
                    actions[Group1]=[ SetGroup(group=4) ] };
```

Дальше нам надо решить - на какие реальные кнопки мы возложим задачу эмулировать наши виртуальные клавиши.
Поскольку это никак не "ущемляет" реальную кнопку (ее основные обязанности), выбирать мы можем так - как нам будет удобнее. Можно для этого выбрать "цифровые" кнопки - 1,2,3. А можно - какие-нибудь "буквенные", например - **K (koi8), A (alt), W (Windows)**.
Я остановлюсь на первом варианте.
Итак. Описание реальных кнопок (можно не писать их заново, а просто добавить нужную строчку в те описания, которые уже должны быть в нашей раскладке) -

```
key <AE01> {        overlay1 = <RUS1>,
                    [           1,  exclam ],
                    [  exclam,      1 ] };

key <AE02> {        overlay1 = <RUS2>,
                    [           2,          at ],
                    [ quotedbl,     2 ] };

key <AE03> {        overlay1 = <RUS3>,
                    [   3,  numbersign ],
                    [ apostrophe,   3 ] };
```

Осталось только выбрать кнопку, которая будет "махать флагом" и превращать наши реальные кнопки в виртуальные. В общем-то, это может быть не одиночная кнопка, а комбинация кнопок, но не забудьте, что вместе с этой "комбинацией" нам надо будет нажимать и одну из "виртуальных" кнопок (пальцев хватит?).
Я остановлюсь на той же **<MENU>**.
Нам нужно "подвесить" на нее "действие" **SetControls(controls=overlay1)**. Напомню, что это "действие" "держит" флаг только пока кнопка нажата. То есть, кнопки 1-2-3 нажатые вместе с **Menu** (**Menu+1, Menu+2, Menu+3**) будут действовать как дополнительные переключатели. А если кнопку Menu не трогать, то эти клавиши работают как обычно.

```
key <MENU> {        [NoSymbol],
                    actions[Group1]=[ SetControls(controls=overlay1) ]};
```

Все. Можно попробовать.
(Если это не будет работать, попробуйте заменить **SetControls** на **LockControls**. Естественно, в этом случае клавишу **<MENU>** не надо удерживать в нажатом состоянии. По первому нажатию она будет "включать" управляющий флаг **overlay1** и кнопки 1-2-3 превратятся в переключатели групп, а по повторному нажатию флаг **overlay1** будет "сбрасываться" и кнопки 1-2-3 вернутся к своей "основной работе").

На этом я заканчиваю свои примеры, хотя можно было бы выдумать еще пару-тройку переключателей. Но, основные механизмы (и "подводные камни") я описал. Все остальное зависит от вашей фантазии.

Почему руссификация через XKB не работает?

К сожалению, настроить XKB с "русской" раскладкой клавиатуры - это еще "полдела".
Очень часто при этом русские буквы вообще "не вводятся" или вводятся, но "не те", или вводятся, но "не везде" (не во всех программах).

Кто виноват?

Что такое "русские буквы"?

Давайте не поленимся и рассмотрим проблему с самого начала.
Под русским алфавитом в Юниксах обычно понимают кодировку **KOI8-R** (хотя иногда используется **iso-8859-5** или "вариации на тему" **KOI8-R - KOI8-U**).
В этой кодировке русские буквы занимают коды 0xc0-0xff (в кодах 0x40-0x7f расположены латинские буквы).
Но в этом же диапазоне могут размещаться

- буквы национальных алфавитов стран Западной Европы (буквы с "шляпкой", "тильдой", "умлаутом" и т.п.)
- греческий алфавит
- арабский алфавит
- и т.п.

То есть, на одно и то же место в "кодовой таблице" претендуют куча разных алфавитов.
Попытки совместить весь этот "зоопарк" в X-Window, привели к тому, что под код символа в раскладке клавиатуры были выделены два байта. В старшем байте хранится некий номер определяющий charset (набор символов), а в младшем - собственно код символа. (В общем, идея та же что и у UNICODE, хотя это и не "уникод").
Кстати, "кириллическому" алфавиту (не обязательно "русскому") достались коды, у которых в старшем байте число 6 (признак "кириллицы"), а младший байт содержит код, который совпадает, как правило, с кодом буквы в koi8-r (хотя это и не так уж важно, как мы увидим в дальнейшем).
Называются эти символы - Cyrillic_a, Cyrillic_be, Cyrillic_tse и т.д. Обычно про них говорят - Cyrillic коды.
Итак. При нажатии клавиши, когда активна "русская" раскладка, в программу должны попадать двубайтные Cyrillic коды.
Но проблема в том, что большинство программ внутри хранит строки (и выдает их на экран) как цепочку байтов. То есть, двубайтные коды на входе в программу должны быть преобразованы в однобайтные. При этом, естественно, часть информации (принадлежность символа к конкретному национальному алфавиту) теряется.

Процедуры Xlib для преобразования кодов.

Для такого преобразования в **Xlib** существует две процедуры - **XLookupString** и **XmbLookupString** (**mb** означает - **"multi byte"**). В качестве аргумента этим подпрограммам передается "сообщение о нажатии" (или отпускании) клавиши (Key Press Event), а "на выходе" должен получаться однобайтный код символа.
(Строго говоря, есть еще одна аналогичная процедура - **XwcLookupString** (**wc** означает - **"wide char"**). Но она используется крайне редко.)
Надо отметить, что в большинстве современных программ выбор конкретной процедуры происходит следующим образом.

Процедура **XLookupString** более "древняя" и более "глупая". Процедура **XmbLookupString** более сложная и более гибкая, но в своей работе пользуется "input context'ом" текущего "окна". Не вдаваясь в подробности, можно сказать, что "input context" - это некий объект, описывающий особенности ввода (input) в "свое окно", и содержащий список "методов" для различных преобразований входных цепочек символов.
Поэтому, большинство "иксовых" программ пытаются создать для своих "окон" input context. И, если им это удается, используют **XmbLookupString**, а если по каким-то причинам input context не может быть создан, то используют **XLookupString**.
Так вот. Для того, чтобы избежать путаницы в алфавитах, обе эти процедуры ориентируются на "текущую locale" (которая, в частности, определяет "национальный" алфавит с которым в данный момент работает система).
Если вы не знаете - что такое locale, могу порекомендовать сайт - "Locale AS IS", где это об этом рассказано достаточно подробно (и по русски!).
Здесь замечу только, что ...

- программы обычно узнают о текущей locale из "переменных окружения" (environment) **LANG**, или **LC_ALL, LC_CTYPE** и т.п.;

- для каждой locale в системе существует набор файлов, в которых содержится вся информация "зависящая от языка" (и, соответственно - алфавита);

- в системе X-Window (а ее можно рассматривать как отдельную ОС), основными файлами, описывающим "категории locale" является файлы **XLC_LOCALE**, разложенные по отдельным поддиректориям в директории **X11R6/lib/X11/locale**. Каждая поддиректория соответствует одной конкретной locale.

- процедуры **Xlib** в своей работе используют некоторые значения (какие именно - рассмотрим чуть позже), описанные в файле **XLC_LOCALE**.

Кроме того надо заметить, что не бывает "никакой locale". Если locale не задана "переменными окружения" или задана, но такая, что система затрудняется найти нужные файлы, то будет использоваться locale - "**C**".
Так вот. При старте программа должна установить нужную locale (ниже мы поговорим об этом немного подробнее). Библиотека **Xlib** найдет соответствующий файл, описывающий эту locale. Процедуры ***LookupString** по этому описанию будут принимать решение - как преобразовать двубайтные коды (в нашем случае - Cyrillic коды) в однобайтные символы.
Если заглянуть немного глубже в работу этих процедур, то можно заметить, что

- обе процедуры ориентируются на параметр locale (точнее, он назывется - класс) - **encoding_name**;

- значение этого класса (для работы с Cyrillic) должно быть **KOI8-R** или **ISO8859-5**;

- в зависимости от значения этого класса...

    - если он **KOI8-R** - Cyrillic превращаются в однобайтные коды **koi8-r**

    - если **ISO8859-5** - Cyrillic превращаются в коды **iso8859-**5

    - если ни то ни другое - Cyrillic вообще ни во что не преобразуется. То есть "на выход" выдается строчка нулевой длины. Это и выглядит как "русские буквы не вводятся".

(Надо отметить, что эти названия "зашиты" в библиотке **Xlib** вместе с таблицами перекодировки. Поэтому никакими внешними файлами изменить их, или добавить новые, нельзя.)
Кстати, если locale "никакая" (то есть - **C**), или **encoding_name** не определен, то нормально вводятся коды "национальных алфавитов Западной Европы", которые занимают то же место в кодовой таблице, что и **koi8-r**. А если нормально преобразуются коды Cyrillic, то наоборот - "подавляется" ввод "западноевропейских" символов.
Кроме того, есть отличия в работе двух ***LookupString**.

- **XLookupString** прежде всего пытается взять **encoding_name** из "переменной окружения" **_XKB_CHARSET**. Если такая переменная есть, то используется ее значение и locale устанавливать не нужно.

- **XmbLookupString**, напротив, не только обязательно требует правильной locale, но и еще использует в своей работе значение двух классов - **encoding_name** и **ct_encoding** (**ct - "compaund text"**). Поэтому, для ее нормальной работы, значения этих двух классов должно совпадать (по крайней мере - для "кириллицы").

Что значит - "установить locale"?

Конечно, об этом лучше прочитать в man'ах или уже упомянутой "Locale AS IS".
Но я постараюсь вкратце описать основные моменты этого действа.
Во-первых, надо заметить, что существует "системная" locale (или "libc'ишная"), которая влияет на работу процедур **libc**, а не "иксов".
В X-Window существует как бы "продолжение" этой locale - дополнительные файлы, в которых описываются параметры влияющие на работу процедур из **Xlib**.
Для того, чтобы программа "настроила" **libc** под нужную locale, она вызывает в начале процедуру **libc - setlocale()**.
Вызов это процедуры имеет три формы (возможно, так о них не говорят, но мне легче будет ссылаться)

1. **setlocale(..., "ru_RU.KOI8-R")** - в вызове явно указывается - какую locale требуется установить;

2. **setlocale(..., "")** - в этом случае процедура пытается взять название соответствующей "категории locale" (вот что это такое, я объяснять не буду), заданной первым аргументом, из "одноименной" переменной окружения; если такой пременной нет, то из переменной окружения **LANG**, если и такой нет, то locale будет "никакая", то есть - **"C"**.

3. **setlocale(..., NULL)** - а вот это, скорее, не "**set**", а "**get**", поскольку она ничего не устанавливает, а наоборот - возвращает название locale, которое было установленно с помощью одной из первых двух форм вызова **setlocale()**.

Итак. Для того, чтобы "установить locale" ("libc'ишную"), программа вызывает **setlocale()** в первой или второй форме. Обычно используется вторая форма, поскольку это позволяет пользователю гибко менять текущую locale с помощью переменных окружения.
Если программа "иксовая", то есть использует **Xlib**, то при первых же вызовах процедур **Xlib**, зависящих от locale, происходит настройка "иксовой" locale.
Надо заметить, что это делается автоматически. То есть никаких дополнительных вызовов не требуется.
Соответствующие процедуры **Xlib** узнают название текущей "libc'ишной" locale с помощью "третей формы вызова" **setlocale()** (это важно!) и по этому названию пытаются найти соответствующий файл (**XLC_LOCALE**), содержащий "иксовые" компоненты locale.
Для этого они ищут подходящую поддиректорию в файле **X11R6/lib/X11/locale/locale.dir**. Если там название не находится, то сначала пытаются "подменить" его с помощью файла **X11R6/lib/X11/locale/locale.alias**, а потом, опять же, найти с помощью **locale.dir**. (Содержимое этих файлов достаточно понятно без дополнительных пояснений.)
Если и после этого "ничего подходящего" не находится, то используется "иксовая" locale **"C"** (хотя "libc'ишная" может быть другая).

Таким образом, для "настройки locale", как "системной", так и "иксовой", необходимо в начале программы вызвать "libc'ишную" процедуру **setlocale()**.
И вот здесь есть одна тонкость...

"Иксовая" setlocale().

Дело в том, что процедуры **Xlib** для того, чтобы узнать название "текущей locale" используют "третью форму" вызова "libc'ишной" **setlocale()**.
Но, на тот случай, если в "системной" **libc** нет такой процедуры, в **Xlib** существует "заглушка" - (вообще-то, она называется **_Xsetlocale()**), которая может вызываться вместо системной **setlocale()**.
Для того, чтобы работала "заглушка", библиотека **Xlib** должна быть собрана с "опцией"
```
#define	X_LOCALE
```
(при этом вызовы **setlocale()** автоматически заменяются на вызовы **_Xsetlocale()**)
Надо заметить, что "иксовая" **setlocale()**, хотя и вызывается точно так же, как и "системная" (те же три формы вызова), имеет некоторые отличия во "второй форме" (наиболее популярном способе установки "текущей locale").

- Категории (первый аргумент) могут быть только **LC_ALL** или **LC_CTYPE**.

- При этом название locale она пытается взять из переменных окружения **LC_CTYPE** и, если не получилось - **LANG**.

- На переменную окружения **LC_ALL** она внимания не обращает (даже если первый аргумент - "категория" **LC_ALL**).

Ну и, естественно, эта "заглушка" устанавливает только "иксовые" параметры locale, а не "libc'ишные".
Так вот. Проблемы могут возникнуть, если у вас в системе **Xlib** почему-то собрана с "опцией" **X_LOCALE**, хотя в **libc** соответствующая процедура имеется.
Тогда вызов **setlocale()** из **libc** запомнит название locale в своих внутренних переменных, а процедуры **Xlib** будут спрашивать "текущую locale" у своей "заглушки", которая, естественно, ее не знает.
Если уж у вас **Xlib** собрана с **X_LOCALE**, то и программы должны вызывать не "системную" **setlocale()**, а "иксовую".
Для этого перед вызовом **setlocale()** должно стоять
```
#define	X_LOCALE
#include <X11/Xlocale.h>
```
Если **Xlib** "нормальная" (то есть ориентруется на "системную" **setlocale**), то этих строчек не нужно. (Хотя, конечно, понадобится **#include <locale.h>**)

Программы "правильные" и "неправильные".

Итак. Можно сказать, что программы могут быть "правильными" и "неправильными" в смысле "установки locale".
"Правильные" программы вызывают в начале **setlocale()** и в них "русские буквы вводятся" (если, конечно, у вас переменные окружения указывают на существующий файл **XLC_LOCALE**).
"Неправильные" программы "забывают" установить текущую locale (соответственно, в них используется locale **"C"**) и в них "русские буквы НЕ вводятся" (хотя замечательно вводяться "западноевропейские" буквы).

Почему работают другие способы руссификации?

Кроме "руссификации через XKB" часто используются "старые" методы - "загрузка **xmodmap**" и "программы - переключатели клавиатуры" (**xruskb, xes** и т.п.).
В общем-то, принципиальная разница между ними в том, что ...

- **Xmodmap** "честно" размещает русские буквы во второй группе таблицы символов (при этом используются функции core protocol'a, а не XKB) и определяет клавишу - переключатель между группами.

- А программы-переключатели "сидят резидентом" в памяти и на каждое переключение раскладки, просто "нагло" переписывают первую группу таблицы символов.

А общее у них то, что они в качестве русских букв используют не коды Cyrillic, а их однобайтные аналоги. Которые, вообще-то, в X-Window отведены под символы национальных алфавитов Западной Европы ("умлауты", "тильды" и т.п.)
То есть, "неправильные" программы (работающие с locale **"C"**) воспринимают их как "родные".
Проблемы как раз возникают с "правильными" программами, которые выставляют "кириллическую" locale. Но с ними, обычно борятся "шаманскими" методами - либо устанавливают "переменные окружения" так, чтобы они НЕ указывали на "кириллическую" locale, либо убирают из **XLC_LOCALE** строчку, описывающую **encoding_name** (именно по ней ***LookupString** "догадываются", что допустимыми являются только коды Cyrillic).

Почему иногда вводятся "не те буквы"?

Надеюсь, вы поняли - почему бывает, что "русские буквы не вводятся" или "вводятся, но не везде" (не вводятся, если программа - "неправильная").
Почему же бывает, что "вводятся, но не те"?
Ну, во-первых, возможно у вас программа использует "не те шрифты" (**fonts**).
В обычных "фонтах" на месте "русских" букв расположены "западноевропейские" (которые имеют те же коды, то и русские в **koi8-r**).
Во-вторых, может быть, что у вас неправильно настроена locale (переменными окружения). То есть, она указывает на "русский" **XLC_LOCALE**, но не для кодировки **KOI8-R**, а для **ISO8859-5**. Такая ситуация может быть, если вы используете "сокращенное" название locale (**ru, ru_RU, ru_SU, russian**), которое в файле **locale.alias** указывает на locale **"ru_RU.ISO8859-5"**.
И, наконец, как это ни странно звучит, возможно, что ошибка в **Xlib**. Дело в том, что в **XFree86 3.3.3** как-раз в **Xlib** содержится ошибка. Из-за которой Cyrillic коды перекодируются в **iso8859-5**, если **encoding_name - KOI8-R** (если **encoding_name ISO8859-5**, то вообще ничего не получится, поскольку они, в свою очередь, перепутаны там с Arabic).

Что делать?

Так что же делать?
Давайте сначала найдем хотя бы одну программу, которая работает с "XKB руссификацией" правильно. Что делать с "неправильными" программами решим потом.
Лучше всего взять программу **xterm**, если, конечно, она у вас из того же "комплекта", что и X-сервер. То есть собрана с "текущей **Xlib**" и с теми же "опциями", что и **Xlib**.

Настройка системы.

Если у вас клавиатура уже руссифицирована каким-нибудь "старым" способом (с помощью **xmodmap** или программ-руссификаторов **xruskb, xes** и т.п.), то уберите их. Они будут только мешать. Во всяком случае, если будет "очень надо", вернете их потом.
Короче, позаботьтесь о том, чтобы у вас не было русского **.Xmodmap** в домашней директории (и/или в **{XROOT}/lib/X11/xinit/**) и никакие **xruskb** не стартовали бы автоматически при запуске "иксов".
Итак. Прежде всего убедитесь, что у вас правильно установлены "русские фонты". Для этого нужно взять текст, "набитый" в **koi8** (например, в какой-нибудь "консольной" программе). Если он нормально читается в **xterm**, то шрифты установлены "правильные". (Как установить "кириллические" шрифты я здесь описывать не буду. В Интернете достаточно инструкций на эту тему.)
Дальше. Проверьте, что у вас правильно установлена locale. Обычно она устанавливается переменной окружения **LANG**. Убедитесь, что значение этой переменной указывает на существующие файлы для "libc'ишной" locale (в **/usr/share/locale**) и "иксовой" locale (через **locale.dir** и, если нужно, **locale.alias**).
Не полагайтесь на результаты выдачи команды **locale**. В некоторых случаях она может показывать не совсем то, что и процедура **setlocale()** вызываемая внутри прикладных программ.
Если будут проблемы, можете воспользоваться моей тестовой программкой testXlc.
Она определяет

- "системную" locale,

- locale для **Xlib**,
- полный путь до файла **XLC_LOCALE**,
- значения некоторых классов "иксовой" locale,
- в том числе - **"encodingName"** (которое влияет на работу ***LookupString**).

Для "сборки" этой программы надо последовательно набрать команды
```
xmkmf
make
```
Надо также заметить, что эта программа использует внутренние процедуры **Xlib** (не предназначенные для вызова в прикладных программах). Поэтому, нет гарантии, что она будет "собираться" и правильно работать во всех версиях XFree. Во всяком случае, она должна давать требуемый результат в версиях 3.3.2, 3.3.3, 3.3.3.1.
Далее. Можно проверить, что XKB действительно переключается на "русскую" раскладку и выдает коды Cyrillic.
Для этого можно воспользоваться программой **xev** (если ее нет в вашей системе, то можно найти ее в **XFree86-contrib**).
Вообще-то, это "универсальный тестер" "событий" (X events). Ее надо запустить из под **xterm**. При нажатии кнопок она должна писать что-то вроде этого
```
KeyPress event, serial 21, synthetic NO, window 0x5800001,
    root 0x25, subw 0x0, time 3744190622, (533,270), root:(610,437),
    state 0x2000, keycode 38 (keysym 0x6c6, Cyrillic_ef), same_screen YES,
    XLookupString gives 0 characters:  ""

KeyRelease event, serial 21, synthetic NO, window 0x5800001,
    root 0x25, subw 0x0, time 3744190755, (533,270), root:(610,437),
    state 0x2000, keycode 38 (keysym 0x6c6, Cyrillic_ef), same_screen YES,
    XLookupString gives 0 characters:  ""
```
Обратите внимание, что если у вас клавиатура перключена в "русский режим", то в третей строчке сообщения должно быть название Cyrillic кода клавиши
Кстати, **xev** - "неправильная" программа. Если сделать ее "правильной", добавив в начале вызов (и "пересобрав" ее)
```
setlocale(LC_CTYPE, "");
```
то она будет показывать и результат трансляции Cyrillic кодов в **koi8-r** (ну, или **iso8859-5**), например -
```
KeyPress event, serial 21, synthetic NO, window 0x5800001,
    root 0x25, subw 0x0, time 3744190622, (533,270), root:(610,437),
    state 0x2000, keycode 38 (keysym 0x6c6, Cyrillic_ef), same_screen YES,
    XLookupString gives 1 characters:  "ф"

KeyRelease event, serial 21, synthetic NO, window 0x5800001,
    root 0x25, subw 0x0, time 3744190755, (533,270), root:(610,437),
    state 0x2000, keycode 38 (keysym 0x6c6, Cyrillic_ef), same_screen YES,
    XLookupString gives 1 characters:  "ф"
```
Ну и, наконец, надо убедится, что у вас не XFree86 3.3.3 (хотя с этого надо было начинать :-).
(Надо бы написать простенький "тестер", который проверяет правильность работы ***LookupString** при заведомо правильной **encoding_name**. Но пока сойдет "подправленная" **xev**. Она, кстати, использует только **XLookupString.**)
Итак, если

- у вас стоят "кириллические" шрифты;
- программа **testXlc** показала, что название "системной" и "иксовой" locale совпадают, и значение **"encodingName" - KOI8-R**;
- **xev** показывает Cyrillic коды;

- "поправленный" **xev** транслирует их в нормальные **koi8-r** коды;

то система у вас настроена. И "правильные" программы должны работать нормально.
А вот что делать с "неправильными" программами - разговор отдельный...

Что делать с "неправильными" программами?

(Если мои советы вам не помогли (или есть какие-то "непонятки"), можете обращаться ко мне "мейлом".
Но я не хотел бы видеть вопросы типа - "я все сделал, как написано, но все равно не работает". Будет лучше, если вы подробно напишете о проделанных тестах и их результатах.) Иван Паскаль pascal@tsu.ru

---

Что делать с "неправильными" программами?

Вообще-то, первый ответ - узнайте - не решены ли эти проблемы в последней версии данной программы.
Возможно "бороться" уже не надо. И только если проблемы остались, то ...
Прежде чем пытаться давать какие-то рецепты, я хотел бы поделить все "неправильные" программы на несколько категорий.

- Программы "в исходниках".
- Мультиязыковые программы.
- Программы "в бинарниках".

Программы в "исходниках".

Если программы доступны в "исходниках", то их можно исправить. Обычно для этого достаточно вставить вызов **setlocale()** где нибудь в самом начале программы (и уговорить авторов или maintainer'ов - включить этот вызов в официальный дистрибутив).
Не забудьте при этом о "иксовой заглушке" **setlocale()**.
Если у вас **Xlib** собрана с **X_LOCALE**, то ...

- Самое правильное - "выкинуть" ваши "иксы" и поставить "нормальные".
- Если же этого сделать нельзя, то и программы "лечить" с помощью "иксовой" **setlocale()** (что для этого нужно делать - я уже написал).

Надо также отметить, что не всегда для "лечения" требуется именно **setlocale(...)**. В некоторых библиотеках-"тулкитах" (toolkits) существуют свои функции, которые выполняют те же действия. В конечном счете они вызывают ту же **setlocale()**, но могут выполнять и какие-то дополнительные настройки, необходимые для работы остальных подпрограмм своего "тулкита".
Естественно, программы использующие эти "тулкиты" лучше исправлять с помощью "родной" подпрограммы настройки locale.
Например. Своя собственная процедура для установки locale в приложении имеется в библиотеке **Xt**, которая работает "поверх" **Xlib** и, в свою очередь, является основой для популярных "тулкитов" **Xaw*** и **Motif** (**lesstif**).
Поэтому, как сказано в документации Андрея Чернова (о "коификации" всего и вся):
Если программа использует один из перечисленных "тулкитов" (**Xt, Xaw*, Motif/lesstif**), то для ее исправления лучше вставить вызов процедуры **XtSetLanguageProc**, например в таком виде
```
XtSetLanguageProc(NULL, NULL, NULL);
```
Также своя процедура имеется и в библиотеке **GTK - gtk_set_locale()** (без аргументов). Хотя она делает практически то же, что и обычная **setlocale(...)**, но по крайней мере еще и выдает разнообразную диагностику, "если что не так".

Еще одна проблема, на которую надо обратить внимание при "исправлении" программ - шрифты (fonts). В некоторых программах могут оказаться "вшитыми" названия шрифтов, которыми эта программа должна отображать вводимые символы.
К сожалению, никаких общих рекомендаций для этого случая я дать не могу. Замечу только, что ...

- если русские буквы "вводятся, но не так" (то есть, вместо русских букв печатаются какие-то "западноевропейские")

- и ни в каких настройках (конфигурационных файлах) исправить "фонт" не удается

- то имеет смысл поискать в "исходниках" - не прописаны ли там полные названия "фонтов".

И наконец, особые проблемы могут возникнуть с "мультиязыковыми" программами ...

Мультиязыковые программы.

Это программы, которые пытаются обеспечить ввод на нескольких языках и, соответственно, ввод символов из нескольких "несовместимых" алфавитов (западноевропейского, русского, греческого и т.п.). Обычно, это - разнообразные редакторы и "текстпроцессоры" (Lyx, emacs и т.п.)
Как я уже говорил, процедуры Xlib, ответственные за перевод кодов клавиш в однобайтные символы (**XLookupString, XmbLookupString**), "на выходе" дают только те символы, которые являются "допустимыми" в текущей locale. То есть, если текущая locale - русская, то будут преобразовываться в байтовые коды только русские буквы, а, например, греческие или "западноевропейские" - будут подавляться. И наоборот, при греческой locale, эти процедуры будут подавлять ввод руссих букв.
Поэтому, многоязыковые программы просто игнорируют однобайтные коды, которые им возвращают процедуры **X*LookupString**, а используют двубайтные коды клавиш (symbols в терминах XKB) и интерпретируют непосредственно их в меру своего понимания.
К сожалению, далеко не все из этих программ правильно понимаю - что делать с кодами типа Cyrillic_*.
Естественно, "лечить" эти программы с помощью **setlocale()** бесполезно. Во-первых, большинство из них и так уже используют этот вызов. А во-вторых, установка locale повлияет только на то - какие символы будут подавляться при преобразовании, а какие - нет. Как я уже сказал, многоязыковые программы это как-раз и не интересует.
К сожалению, общих рецептов и в этом случае нет. Но, с другой стороны, некоторые из таких и программ и не надо "патчить". Им можно объяснить, что "Cyrillic коды" - это обычные "буквенные" коды, с помощью их же файлов конфигурации.
Некоторые решения можно найти ...

- "Лечение" xemacs можно найти на сайте Алексея Выскубова - emacs.zip .

- То же самое для Lyx (и некоторых других программ) на сайте "Пингвин при галстуке" .

Кстати, исправления для популярного "тулкита" **Tcl/Tk** можно найти на сайте Виктора Вагнера - http://www.ice.ru/~vitus/tcl/locale-tcl.html.

Программы в "бинарниках".

Что делать, если программы имеются только в "бинарниках"?
К сожалению - это самый трудный случай. Конечно, речь идет не о тех программах, исходные коды которых доступны, но почему-то отсутствуют у вас. Хуже всего то, что программы, распространяемые только в "бинарном" виде, зачастую еще и "статически слинкованы". То есть, они используют свою библиотеку Xlib "встроенную" в само приложение, а не ту, что в вашей системе. Поэтому, изменить "поведение" такой программы внешними настройками (о некоторых из них я расскажу позже), часто бывает просто невозможно.
В этом случае, самое правильное - попробовать уговорить авторов на соответствующую правку.
Но в некоторых случаях может помочь один из "хакерских" способов ...

Методы "грубого хака".

Первый метод - самый плохой.

Можно "откатиться" к методам, которые использует **xmodmap**.
Для этого в раскладке клавиатуры Cyrillic коды надо заменить на однобайтные коды **koi8-r** (раскладку можно взять здесь). При этом "неправильные" программы начнут их воспринимать, принимая за "западноевропейские" символы.
А чтобы "правильные" программы их не "давили", нужно "испортить" **XLC_LOCALE** для **koi8**. Для этого достаточно "закомментарить" (или вообще выкинуть) строчку с **encoding_name** в файле **koi8-r/XLC_LOCALE**.
Этот метод будет работать почти со всеми приложениями. Даже со "статически слинкованными бинарниками". Хотя могут возникнуть проблемы с некоторыми мультиязыковыми программами. Но он же, как я уже сказал - самый плохой. По сути, это "возврат к старому". Ради нескольких старых "глупых" программ, придется обманывать все новые "правильные" программы.

Второй метод - намного лучше.

Идея этого метода принадлежит Александру Канавину (хотя реализовал он его немного по-другому).
Поскольку "неправильные" программы, забывая установить нужную locale, используют locale **"C"**, можно им вместо **C** "подсунуть" русскую locale, например - **ru_RU.KOI8-R**.
Для этого достаточно в файле "алиасов локалей" (**X11R6/lib/X11/locale/locale.alias**) добавить строчку
C ru_RU.KOI8-R
Таким образом, "правильные" программы будут пользоваться правильной locale - **KOI8-R**, а "неправильные" вместо **C** получат ту же **KOI8-R**.
К сожалению, и этот способ неидеальный.

- Во-первых, это "локальный хак" (для территорий с преобладанием "русскопишущего" населения :-). Поэтому, он не имеют никаких шансов войти в официальные дистрибутивы XFree. И следовательно, эти действия придется повторять каждый раз при установке новой версии "иксов".

- Во-вторых, он может и "не сработать" для программ "статически слинкованных".

- Ну и наконец, тем самым вы "испортите" locale "C", и если какой-то из программ требуется именно она, то ... будет плохо. (Я слабо представляю - кому нужна именно такая "локаль", но - кто его знает.)

Третий способ - самый правильный.

Еще один способ описан в "Примеры: Новая 'старая' раскладка".
В нем вводятся две русские раскладки - одна для "неправильных" программ, другая - для "правильных".
Эту раскладку можно положить в **X11R6/lib/X11/xkb/symbols**, переименовав ее в **ru** (или подправить название русской раскладки в других файлах конфигурации **XKB**).
Тогда клавиша "переключатель групп" будет циклически перебирать три группы, что конечно же не очень-то удобно в работе. Чтобы "облегчить жизнь", можно воспользоваться моим индикатором-переключателем клавиатуры xxkb . (Для работы с тремя раскладками надо немного изменить настройки **xxkb**. Об этом подробно написано в соответствующей инструкции .)
Как я уже заметил, этот способ - самый правильный. "Правильные" программы будут работать с "правильной" раскладкой (если вы настроите **xxkb**, то эта раскладка будет включаться "по умолчанию"), а для "неправильных" программ вам придется при старте такой программы выбрать "неправильную" раскладку. В этом есть, конечно, некоторые неудобства, но **xxkb** сведет их к минимуму.

Программы, имеющие отношение к XKB.

- xkbcomp

- setxkbmap
- xmodmap
- xkbwatch
- xkbvleds и mxkbledpanel
- xkbevd и xkbbell
- xkbprint
- Программа для редактирования раскладки - xkeycaps.
- Индикаторы-переключатели.

xkbcomp

Это самая важная программа. Во всяком случае, она отрабатывет каждый раз при старте "иксов" для того, чтобы перевести файлы конфигурации XKB в бинарный формат, понятный самому X-серверу. (Как понятно из названия, это - "XKB compiler").
Но более полезно то, что ее можно вызвать и "вручную". При этом, она может выполнять и обратное преобразование - из бинарной формы в "человекочитаемый" конфигурационный файл. И что самое полезное - она может обмениваться данными (записывать и читать) непосредственно с работающим X-сервером.
То есть, ее можно использовать для того, чтобы в ходе работы загрузить новую конфигурацию XKB в X-сервер, или наоборот - прочитать текущую конфигурацию и перевести ее в "читабельный" формат.
Последнее тоже важно. Поскольку полная конфигурация складывается из содержимого многих файлов/блоков (не забудьте, что в любом файле может быть '**include**'), понять - что же должно получится в результате такого "сложения" - не так-то просто. С помощью **xkbcomp** можно "вытащить" из X-сервера "суммарную" конфигурацию (в виде полной **xkb_keymap**).
Полную информацию о **xkbcomp** можно прочитать в соответствующем **man**'е.
Поэтому, приведу лишь примеры - как можно "обменяться информацией" с X-сервером. Вообще-то, если в качестве источника или приемника данных надо указать X-сервер, это можно сделать (как сказано в **man X**) в форме
```
hostname:dispalay_number.screen_number
```
В простейшем случае, когда X-сервер запущен на вашей же машине, причем - только один и с одним "скрином", эта последовательность выглядит как "**:0**".
Таким образом, для чтения текущей конфигурации XKB можно использовать команду
```
xkbcomp :0 outfile
```
(можно даже не указывать **outfile**, в этом случае программа сама сочинит имя для выходного файла, типа - **server-0.xkb**).
Обратите также внимание на ключи **-a** (all) и **-dflts** (defaults). С ними выходная информация будет более полной.
Соответственно, для загрузки новой конфигурации можно использовать команду
```
xkbcomp xkbfile :0
```
Подразумевается, что **xkbfile** в этом случае - файл типа **xkb_keymap**. Хотя он может и не содержать непосредственно в себе все необходимые описания (как это получается в выходном файле при чтении из X-сервера), а только указывать с помощью **include** файлы-компоненты полной конфигурации.
Кстати, эти файлы **xkbcomp** ищет сначала в текущей директории и, если не находит, то в "базе данных" конфигураций (**X11R6/lib/X11/xkb/**).
Примеры составления своих файлов конфигурации для "подсовывания" их **xkbcomp** можно посмотреть в "Примеры изменения конфигурации XKB: Где будем экспериментировать."

setxkbmap

Еще одна утилита, которая позволяет "на ходу" поменять конфигурацию XKB.
Пожалуй, главное ее преемущество перед **xkbcomp** в том, что она понимает задание конфигурации в виде **rules-model-layout...** (**xkbcomp** этих слов не понимает).
С другой строны, **setxkbmap** может брать данные только из файлов "базы данных" настроек (причем, только на той же машине, где запущен X-сервер). То есть, если вы составите свой файл конфигурации, то прежде чем воспользоваться **setxkbmap**, необходимо поместить этот файл в нужное место (в соответствующую поддиректорию) "базы данных". (Кстати, **setxkbmap** еще и не понимает входные данные в виде **xkb_keymap**, хотя это просто "глюк" программы.)
Кстати, если будете с ее помощью менять русскую раскладку, не забудьте, что файл **ru** и все его "вариации" обычно содержать только "частичное" описание клавиатуры.
Если вы выполните команду типа
```
setxkbmap -symbols ru
```
то результат будет очень неприятным. У вас "отсохнут" все "небуквенные" клавиши, включая Esc, BackSpace, Enter (!) и т.д.
Поэтому, правильная команда должна выглядеть как
```
setxkbmap -symbols "en_US+ru"
```
Самое полезное применение этой программы - "сброс" XKB в исходное состояние после неудачных экспериментов со своими файлами конфигурации. :-)
Если запустить ее без параметров, то она прочитает ту конфигурацию, которая описана в **XF86Config** и загрузит ее.
Более подробно об этой утилите можно прочитать в соответствующем man'e, а краткую сводку ключей можно получить по команде
```
setxkbmap -?
```

xmodmap

Как ни странно, но эта программа тоже может использоваться для работы с XKB. Просто, она работает с клавиатурным модулем с помощью "старого" протокола (core protocol). Но поскольку XKB "из соображений совместимости" этот протокол понимает, то вполне может воспринять таблицы, загружаемые с помощью **xmodmap**.
Естественно, используя **xmodmap**, вы лишаетесь всех дополнительных возможностей XKB. Но для простейших исправлений раскладки клавиатуры ее вполне можно использовать.
Единственное, что я бы не советовал - использовать широкораспростаненные файлы для руссификации клавиатуры в формате **xmodmap** (.Xmodmap). Почему этого делать не стоит, написано в "Почему руссификация через XKB не работает".
Подробности об этой программа читайте в соответствующем **man**'e. (Но если вы не знаете - что такое **xmodmap**, то ... лучше и не знать :-).

xkbwatch

Эта утилитка показывает текущее состояние "виртуальных" и "реальных" модификаторов. К сожалению, она имеет очень "скромную внешность" (никаких подписей - только какие-то безымянные "лампочки") и, к тому же, остутствует какое-либо описание к ней. (Похоже, авторы писали ее "только для себя", но зачем-то засунули в дистрибутив. :-)
На самом деле, если знать - что именно она показывает, **xkbwatch** может оказаться весьма полезной при отладке своих файлов конфигурации.
Итак, восполняя пробелы в описании, привожу табличку, поясняющую значение индикаторов **xkbwatch**

base	O	O	O	O	O	O	O	O
latched	O	O	O	O	O	O	O	O
locked	O	O	O	O	O	O	O	O
effective	O	O	O	O	O	O	O	O
реальные	O	O	O	O	O	O	O	O
	Mod5	Mod4	Mod3	Mod2	Mod1	Control	Lock	Shift

xkbvleds и mxkbledpanel

Утилита **xkbvleds** предназначаена для отображения состояния виртуальных индикаторов XKB (в том числе и тех, которые имеют "реальные" аналоги в виде светодиодов на клавитатуре).

К сожалению, она выполнена в том же аскетичном стиле (никаких подписей), что и **xkbwatch**. И точно так же не имеет никакого описания.
Поэтому я не буду ничего о ней рассказывать (хотя она даже имеет какие-то ключи), а посоветую забыть о ней и, если хочется чего-то подобного, воспользоваться другой аналогичной утилиткой - **mxkbledpanel**.
Эта утилита имеет более "человеческое лицо". Все индикаторы на ней подписаны (естественно - те, которые реально используются; напомню, что в XKB может быть до 32 виртуальных индикаторов, но это не значит, что все они задействованы в конкретной конфигурации).
Кроме того, эта программа позволяет осуществлять и "обратную связь", если это предусмотренно в конфигурации (подробнее - смотрите в "Немного о внутренностях XKB":"Индикаторы" и "Описание поведения индикатора":drivesKeyboard). То есть, "щелкая мышью" по индикатору можно включать/выключать его и таким образом менять соответствующее состояние XKB. (Кстати, благодаря этому, **mxkbledpanel** может служит простейшим индикатором-переключателем "РУС/ЛАТ").
Недостатки этой программы в том, что она во-первых требует для сборки Motif (хотя должно хватить и lesstif'а), а во-вторых - не входит в дистрибутивы XFree. Найти ее можно, например, на - ftp://ftp.x.org/pub/unsupported/Motif/mxkbledpanel/

xkbevd и xkbbell

Судя по названию, **xkbevd** - "XKB event daemon". Как сказано в **man xkbevd**, "эта команда очень 'сырая'... мы предлагаем ее скорее как 'черновой прототип' для разработчиков, а не как инструмент для 'конечных юзеров'".
В основном она принимает **bell event**'ы и может служить "музыкальной приставкой" (подробнее смотрите в "Внутренности XKB: Расширенные возможности 'пищалки'").
Правда сам **xkbevd** звуки не воспроизводит. В его конфигурационном файле можно указать имя программы для воспроизведения звуков и соответствия между "именем звука" и аудио-файлом, который надо передать в качестве аргумента этой программе.
А вот программа **xkbbell** служит "тестером" для "расширенных возможностей 'пищалки'". Она просто посылает запрос модулю **XKB** для воспроизведения звука. Причем, с ее помощью можно заказать как обычный "писк" (с помощью ключа **-force**), так и "музыкальный фрагмент", указав в качестве аргумента "имя звука".
С помощью дополнительных ключей можно симитировать запрос от конкретного приложения (окна).
Как я уже сказал, **xkbbell** служит скорее "тестером", чем полезной утилитой. Хотя ее можно использовать и в своих "скриптах", как команду для проигрывания звуков (естественно, должна быть запущена хоть какая-нибудь XKB-совместимая "музыкальная приставка").
Подробности о **xkbevd** можно почитать в соответствующем **man**'е. А для **xkbbell** никакого описания нет. Правда, можно посмотреть хотя бы список ключей, запустив ее с ключем **-help**.

xkbprint

Эта утилита генерирует картинку (в формате postscript'а) изображающую клавиатуру. При этом она использует как описание геометрии (**xkb_geometry**), так и "назначение" клавиш. То есть, кроме отрисовывания самой геометрии клавиатуры, она еще и подписывает - какой символ генерируется каждой клавишей.
Особой пользы от нее я не вижу. Разве что - проверить правильность описания геометрии (собственно, кроме этой утилитки ее никто и не использует), да еще увидеть на картинке расположение сразу всех символов.
Подробности - в **man xkbprint**.

Программа для редактирования раскладки - xkeycaps.

Вообще-то, нормальной программы для создания/редактирования раскладок клавиатур в формате XKB - не существует.
Но при желании, для этой цели можно использовать "редактор xmodmap'ов" - xkeycaps. Обратите внимание, что она позволяет на каждую клавишу "подвешивать" до 8-и символов. Что вполне соответствует "четырем группам по два уровня" в терминах XKB.
И хотя она и сохраняет результат вашей работы в формате **xmodmap**, преобразовать его в формат файла типа **xkb_symbols** достаточно просто. Можете воспользоваться моим "скриптом" modmap2xkb, который делает это автоматически (конечно, программка очень "сырая". Если у кого-то есть желание ее улучшить - welcome).
Не могу не заметить, что **xkeycaps** в нынешнем виде - довольно плохой редактор. И дело даже не в том, что она не работает с файлами XKB и, соответственно, не может использоваться для редактирования **xkb_types,**

xkb_compat и т.п. и, кроме того, не использует "геометрию" из **xkb_geometry** (все "геометрии" "зашиты" в самой программе).
Основные ее недостатки в том, что

- нет групповых операций - перемещения символов сразу всех (или выбраных) клавиш в другую группу
- вообще нет "вырезания-вставки" даже для одной клавиши
- нет возможности "подгружать" раскладки из файлов (даже в формате xmodmap). Выбирать можно только из тех раскладок, которые "вшиты" в саму программу.
- и наконец - при сохранении результата она сама сочиняет имя файла (если вы хотите наделать несколько раскладок, вам после каждого сохранения придется вручную переименовывать файл)

Короче, **xkeycaps** представляет собой скорее забавный "эмулятор клавиатуры", чем полноценный "редактор раскладок".

Индикаторы-переключатели.

Несмотря на то, что XKB сам может поддерживать несколько (до четырех) различных раскладок и легко переключать их "на ходу", ему очень не хватает подходящей программы для удобной манипуляции раскладками и отображения текущего состояния ("лампочек" на клавиатуре явно недостаточно для индикации состояния, если раскладок больше двух).
Кроме того, большинство пользователей уже привыкло к "индикаторам-переключателям" (типа **xrus**), которые позволяют переключать раскладки не только с клавиатуры, но и "щелчком мыши" и, к тому же, отображать "текущий язык" иконкой на дисплее.
Что же из таких программ можно посоветовать для работы "в паре с XKB"?
Надо заметить, что популярные "переключалки" типа **xruskb, xes, cyrx, kikbd** (кого я еще забыл упомянуть?) для этого плохо подходят.
Во-первых, они используют раскладки "в старом стиле", то есть не с кодами Cyrillic, а "западноевропейскими" буквами. А это вызывает проблемы у "правильных" программ при корректно установленной locale (подробнее об этом в "Почему руссификация через XKB не работает?").
А во-вторых, все они не используют возможности XKB для переключения раскладок, и "на каждое телодвижение" сами перезагружают раскладки в X-сервер.
"Настоящими" XKB-переключателями являются -

- **kkb** - http://www.logic.ru/peter
- **FvwmKb** - http://linuxfan.com/~sparrow/software/
- **fookb** - http://linux.piter-press.ru/fookb/
- **xxkb** - http://www.tsu.ru/~pascal/other/xxkb/

Не желая никого обидеть :-), все-таки выскажу некоторые замечания относительно этих программ (возможно - несколько запоздалые).

kkb

"Честная" Xkb-переключалка. Хотя в последних версях автор несколько отошел от "идеи XKB" и добавил "подгрузку" раскладок в процессе работы. Но надо заметить, что сделано это средствами XKB, то есть наиболее оптимальным способом.
Главный недостаток - для сборки требуется "тулкит" **Qt**.

FvwmKb

Главное достоинство этой программы - она запоминает состояние клавиатуры ("язык") для каждого открытого приложения и автоматически переключает его при изменении фокуса окна. Кроме этого, состояние клавиатуры отображается на "обрамлении" окон - иконкой, цветом рамки и т.п.
Главный недостаток - она "заточена" для работы с конкретным window manager'ом - Fvwm-2 (хотя со временем, возможно, будут версии и для других wm).

fookb

Очень простая и "легкая" переключалка. Основное достоинство - "легкость". Для сборки достаточно стандартных библиотек и, кроме того, из всех собратьев она - самая компактная.
Недостаток - опять же - простота. :-) Все остальные программы имеют какие-либо "фичи", недоступные **fookb**.

xxkb

Себя сильно хвалить не буду :-).
Замечу только, что основное достоинство как и у FvwmKb - индивидуальное состояние для каждого окна. Но в отличии от FvwmKb - работает с любым wm.
Недостаток - довольно "аскетичный" вид. Выбор раскладки осуществляется просто перебором "иконок", а не с помощью "меню".