

keyTouch 2.2 technical manual

Marvin Raaijmakers

20 August 2006

Contents

Preface	v
1 Introduction	1
2 Files	3
2.1 Keyboard files	3
2.1.1 file-info	4
2.1.2 keyboard-info	5
2.1.3 key-list and key element	6
2.2 The keyboard configuration file	7
2.3 The preferences file	8
2.4 The current keyboard file	8
3 The keytouch program	11
3.1 Data structures	11
3.2 Input files	11
3.2.1 Current keyboard file	12
3.2.2 Keyboard file	12
3.2.3 Keyboard configuration file	12
3.2.4 Preferences file	12
3.3 Output files	13
3.4 Plugins	13
3.5 The application tree	14
4 The keytouch-keyboard program	15
4.1 Data structures	15
4.2 Input files	15
4.2.1 The current keyboard file	15
4.2.2 Keyboard files	15
4.3 Output files	16
4.3.1 The current keyboard file	16
4.3.2 Keyboard files	16
4.4 The keytouch-keyboard script	16
4.5 Calling keytouch-init	16

5	The keytouchd program	17
5.1	Data structures	17
5.2	Input files	17
5.2.1	Current keyboard file	17
5.2.2	Keyboard file	17
5.2.3	Keyboard configuration file	18
5.2.4	Preferences file	19
5.3	Grabbing the keys	19
5.4	When a key is pressed	19
5.5	Plugins	19
5.6	How keytouchd is started	20
6	The keytouch-init program	21
6.1	Data structures	21
6.2	Input files	21
6.2.1	Current keyboard file	21
6.2.2	Keyboard file	21
6.3	Setting the keycodes	21
7	The keytouch-acpid program	23
7.1	Data structures	23
7.2	Input files	23
7.2.1	Current keyboard file	23
7.2.2	Keyboard file	23
7.3	How it works	23
7.4	How keytouch-acpid is started	24
8	Plugins	25
8.1	The plugin_struct structure	26
8.2	Real functions	27
8.3	KTPluginKeyFunction	28
8.4	Implementation	29

Preface

I (Marvin Raaijmakers) developed keyTouch to help people so that they can easily configure the extra function keys of their keyboard under GNU/Linux. I think that it is very important that my software is Open Source. This includes the idea that the source code of the program is available for everybody, so that it can be improved and that people can learn from it. However without good documentation a source code is useless. So I have done my best on the documentation in the source code of keyTouch. This technical manual does not describe the source code, but describes how keyTouch works. It will help you to better understand the code. Even when you do not want to read the source code or when you are not a programmer, this manual can still be very useful.

Chapter 1

Introduction

KeyTouch is a program which allows you to easily configure the extra function keys of your keyboard. This means that you can define, for every individual function key, what to do if it is pressed. KeyTouch version 2 is designed for Linux kernel 2.6 and is the first program of its kind that perfectly works together with kernel 2.6.

This technical manual describes how keyTouch 2.2 works. This is done by describing the different components of keyTouch. The keyTouch package contains 5 different programs: keytouch, keytouch-keyboard, keytouchd, keytouch-acpid and keytouch-init. For each of these programs a chapter will discuss its function, input, output and how it works. First however there will be a chapter that describes the configuration files that are used by the whole keyTouch package.

Note that when the word ‘keytouch’ is used, we mean the program that is part of the keyTouch package. When we use the word ‘keyTouch’ (with a capital T) we mean the complete keyTouch package.

Chapter 2

Files

This chapter describes the files that are read and created by keyTouch. These files are well-formed, invalid XML files. They are invalid XML files because they do not contain a document type declaration. For more information about XML, read the XML specification: <http://www.w3.org/TR/2004/REC-xml-20040204/>

2.1 Keyboard files

One of the most important files used by keyTouch are the keyboard files. These files contain information about a keyboard model so that the model can be supported by keyTouch.

Keyboard files are located in `{datadir}/keytouch/keyboards/`. The location of ‘`{datadir}`’ depends on how keyTouch is installed. When keyTouch is installed from source, then the keyboard files will be installed in `/usr/local/share/keytouch/keyboards/` by default. When keyTouch is installed by using an installation package for your GNU/Linux distribution, then the files are probably located in `/usr/share/keytouch/keyboards/`. Once a keyboard file is in this directory (`{datadir}/keytouch/keyboards/`) it should not change.

The file name is equal to “`{model}.{manufacturer}`”, where ‘`{model}`’ and ‘`{manufacturer}`’ are replaced by the name of the keyboards model and the manufacturer respectively.

A keyboard file contains:

- Keyboard model and manufacturer name.
- Information about the file: syntax-version, date of last change (optional), author (optional).
- Information for every extra function key.

The syntax-version is the version of the syntax that this document describes (which is 1.1).

For every extra function key, the file contains the following information:

- Keycode: The name of the code. The list of keycode names may be found in the header file `<linux/input.h>` (without `KEY_` prefix). Note that this keycode is the keycode which is used by the kernel and not by X.

- The name of the key.
- The default action to run when the key is pressed.

If the key is not of the type ‘acpi-hotkey’ it will have a scancode (= the code the kernel receives from the keyboard). If the key is an ‘acpi-hotkey’ the key will have an event description.

The keyboard file has one root element, of type **keyboard**, and contains the information described above:

```
<keyboard>
  ...
</keyboard>
```

The three dots are replaced by the following elements:

- **file-info**: Contains the file information.
- **keyboard-info**: Contains information about the keyboard.
- **key-list**: The list of extra function keys.

So it will look like:

```
<keyboard>
  <file-info>
    ...
  </file-info>
  <keyboard-info>
    ...
  </keyboard-info>
  <key-list>
    ...
  </key-list>
</keyboard>
```

The contents of these elements are discussed in the following subsections.

2.1.1 file-info

Contains the following elements:

- **syntax-version**: The syntax version.
- **last-change**: The date of the last change to the file (optional).
- **author**: The name of the author of the file (optional).

The date of the last change may be written in any form. The format is specified in the **format** attribute of the **last-change** element. This format is described in the text below, which is copied from the `strptime()` manual page:

The `strptime()` function processes the input string from left to right. Each of the three possible input elements (whitespace, literal, or format) are handled one after the other. If the input cannot be matched to the format string the function stops. The remainder of the format and input strings are not processed.

The supported input field descriptors are listed below. In case a text string (such as a weekday or month name) is to be matched, the comparison is case insensitive. In case a number is to be matched, leading zeros are permitted but not required.

`%%` The `%` character.

`%b` or `%B` or `%h`

The month name according to the current locale, in abbreviated form or the full name.

`%d` or `%e`

The day of month (1-31).

`%D` Equivalent to `%m/%d/%y`. (This is the American style date, very confusing to non-Americans, especially since `%d/%m/%y` is widely used in Europe. The ISO 8601 standard format is `%Y-%m-%d`.)

`%m` The month number (1-12).

`%x` The date, using the locale's date format.

`%y` The year within century (0-99). When a century is not otherwise specified, values in the range 69-99 refer to years in the twentieth century (1969-1999); values in the range 00-68 refer to years in the twenty-first century (2000-2068).

`%Y` The year, including century (for example, 1991).

So the **file-info** element can look like:

```
<file-info>
  <syntax-version>1.1</syntax-version>
  <last-change format="%d-%m-%y">4-9-2005</last-change>
  <author>Author Name</author>
</file-info>
```

2.1.2 keyboard-info

The **keyboard-info** element only contains one element: **keyboard-name**. This element contains the **manufacturer** and **model** elements. These two elements contain the names of the manufacturer and the model respectively. So the **keyboard-info** element can look like:

```
<keyboard-info>
```

```

    <keyboard-name>
      <manufacturer>The Keyboard Company</manufacturer>
      <model>Keyboard Number One</model>
    </keyboard-name>
  </keyboard-info>

```

2.1.3 key-list and key element

The **key-list** element contains for every extra function key an element called **key**. A **key** element can have an **key-type** attribute. If the value of this attribute is *acpi-hotkey* then the key is an ACPI hotkey. If the element has no **key-type** attribute, or the attributes value is not *acpi-hotkey*, then the key is a normal key which is part of the keyboard.

A **key** element contains the following elements:

- **scancode** (for normal keys): Contains the scancode that the kernel receives from the keyboard.
- **event-descr** (for ACPI hotkeys): Contains the ACPI event description of the key.
- **keycode**: Contains the name of the code. The list of keycode names may be found in the header file `<linux/input.h>` (without `KEY_` prefix). Note that this keycode is the keycode that is used by the kernel and not by X.
- **name**: Contains the name of the key.
- **default-action**: Contains the default action to run when the key is pressed.

The **default-action** element can have one attribute named **action-type**. This attribute can have two values: *program* or *plugin*. Explanations for the meanings of the values are obvious. If the attribute is *plugin*, then the **default-action** element will have a **plugin-name** (containing the name of the plugin) and a **plugin-function** (containing the name of the plugin function) element. If the attribute is *program*, then the **default-action** element will contain the command of the program to run. The **default-action** element will also contain the command when **action-type** is not set.

So the **key-list** element can look like:

```

<key-list>
  <key>
    <name>Play/Pause</name>
    <scancode>148</scancode>
    <keycode>PLAY</keycode>
    <default-action>xmessage "hello world!"</default-action>
  </key>
  <key>
    <name>WWW Home</name>
    <scancode>150</scancode>
    <keycode>WWW</keycode>
    <default-action action-type="plugin">

```

```

        <plugin-name>WWW Browser</plugin-name>
        <plugin-function>Home</plugin-function>
    </default-action>
</key>
<key key-type="acpi-hotkey">
    <name>E-Mail</name>
    <event-descr>hotkey ATKD 0000005d 00000002</event-descr>
    <keycode>EMAIL</keycode>
    <default-action action-type="plugin">
        <plugin-name>E-Mail</plugin-name>
        <plugin-function>E-Mail</plugin-function>
    </default-action>
</key>
</key-list>

```

2.2 The keyboard configuration file

The action that is performed when an extra function key is pressed is defined in a ‘keyboard configuration file’. These files are written by the graphical configuration program ‘keytouch’. Note that there can be multiple keyboard configuration files for one user. Which one is used depends on the chosen keyboard model.

Keyboard configuration files are located in `~/.keytouch2/`. The file name is equal to “`{model}.{manufacturer}`”, where ‘`{model}`’ and ‘`{manufacturer}`’ are replaced by the keyboard model of the file and the keyboard manufacturer of the file respectively.

A keyboard configuration file has one root element called **keyboard**, which contains the following elements:

- **keyboard-name**: Contains the name of the keyboard.
- **key-list**: Contains for every key the settings.

The **keyboard-name** element contains two elements: **manufacturer** and **model**. These two elements contain the names of the manufacturer and the model respectively. So the **keyboard-name** element can look like:

```

<keyboard-name>
    <manufacturer>The Keyboard Company</manufacturer>
    <author>Keyboard Number One</author>
</keyboard-name>

```

The **key-list** element has for every key an element called **key**. The **key** element has two elements: **name** and **action**. The **name** element contains the name of the key and is equal to the name used in the corresponding keyboard file. The **action** element has an attribute called **isdefault** which can have the values *true* or *false*. The value of this attribute indicates if the action is equal to the default action for that key (which is defined in the keyboard file). The **action** element also has another attribute called **action-type**. This attribute can have two values: *program* or *plugin*. Explanations for the meanings of the values are obvious. If the attribute is *plugin*, the **default-action** element will have

a **plugin-name** (containing the name of the plugin) and a **plugin-function** (containing the name of the plugin function) element. If the attribute is *program* the **default-action** element will contain the command of the program to run. The **default-action** element will also contain a command when **action-type** is not set. So a **key** element can look like:

```
<key>
  <name>WWW Home</name>
  <action isdefault="true" action-type="plugin">
    <plugin-name>WWW Browser</plugin-name>
    <plugin-function>Home</plugin-function>
  </action>
</key>
```

2.3 The preferences file

The preferences file contains the users preferences and is written by the program 'keytouch'. The preferences can be used by the special action plugins of keyTouch. The file name of the preferences file is `~/.keytouch2/preferences.xml`.

The file has a root element named **preferences**. This element contains the following sub-elements:

- **documents-dir**: Contains the directory where the user stores his/her documents.
- **home-page**: Contains the URL of the users home page.
- **search-page**: Contains the URL of users favorite search engine.
- **browser**: Contains the command of the preferred web browser.
- **email-program**: Contains the command of the preferred e-mail client.
- **chat-program**: Contains the command of the preferred chat client.

So a preferences file can look like:

```
<preferences>
  <documents-dir>~/Documents</documents-dir>
  <home-page>http://keytouch.sf.net</home-page>
  <search-page>http://www.google.com</search-page>
  <browser>firefox</browser>
  <email-program>evolution</email-program>
  <chat-program>gaim</chat-program>
</preferences>
```

2.4 The current keyboard file

The current keyboard file contains the name of the keyboard that was chosen by the user, and is written by the 'keytouch-keyboard' program. The file name of the current keyboard file is `{sysconfdir}/keytouch/current_keyboard.xml`. The location of '{sysconfdir}' depends on how keyTouch is installed. When keyTouch

is installed from source, then the file can be found in `/usr/local/etc/keytouch/` by default. When keyTouch is installed by using an installation package for your GNU/Linux distribution, then the file will probably be located in `/etc/keytouch/`.

The root element is called **current-keyboard** and has two elements: **manufacturer** and **model**. These two sub-elements contain the name of the manufacturer and the name of the model respectively. Example:

```
<current-keyboard>
  <manufacturer>The Keyboard Company</manufacturer>
  <model>Keyboard Number One</model>
</current-keyboard>
```


Chapter 3

The keytouch program

The keytouch program has a graphical user interface for configuring the users configuration. The function of keytouch is to let the user configure the actions of the extra function keys.

3.1 Data structures

There are 3 important data structures that are internally used by keytouch:

- A *key settings list* contains an entry for every extra function key of the current keyboard. Each entry contains the keys name, the action chosen by the user and the keys default action. The user interface allows the user to change the action member of an entry in this list. In other chapters a *key settings list* may also be used, but will not have the same contents as the list meant in this chapter.
- A *preferences structure* that stores the user preferences that will be written to the preferences file (see section 2.3). The user interface allows the user to change the contents of each member in this structure.
- A *plugin list* that has an entry for each plugin. Such entry contains the plugin's name, file name, version number, author, license, description and a list of function names. The user interface allows the user to add entries to and remove them from this list.

3.2 Input files

This section describes the input files of keytouch.

Two of these input files are also output files, namely the keyboard file and preferences file. It is very important that when keytouch fails to read the contents of one of these files, the program must continue and provide default values. We can for example not expect that the user will open one of these files in a text editor and correct its contents. We may only suppose that the user creates these files by using keytouch and the user should not have to know that these files exist.

The other two input files are the output of keytouch-keyboard and because of that keytouch-keyboard is responsible for their contents. If keytouch fails to read one of these files, it will run keytouch-keyboard so that that program corrects the contents.

3.2.1 Current keyboard file

The manufacturer and model name of the keyboard, that was selected by the user, will be read from the current keyboard file (see section 2.4). If this file does not exist or is invalid, keytouch will start keytouch-keyboard so that the user can select the keyboard model. Keytouch-keyboard will write the current keyboard file and keytouch will be started after keytouch-keyboard closed.

3.2.2 Keyboard file

Keytouch will read the keyboard file, that belongs to the model and manufacturer read from the current keyboard file. It reads the **name** and **default-action** element from every **key** element in the keyboard file and puts them in the *key settings list*.

The *key settings list* is being initialized when the keyboard file is read. As described above the names and default actions of the keys will be read from the keyboard file. However entries in the *key settings list* also have a member that contains the action chosen by the user, so that will also need to be initialized. This member will be set to the default action, which is an advantage when the user fails to read the users action from the keyboard configuration file. Note that the keyboard configuration file is read after the keyboard file.

If keytouch fails to read the keyboard file, keytouch starts keytouch-keyboard so that the user can chose the keyboard model. If the keyboard file is invalid, then it will not appear in the list of models in keytouch-keyboard.

3.2.3 Keyboard configuration file

After reading the current keyboard file and the keyboard file, the keyboard configuration file for the current keyboard will be read. Keytouch reads every **key** element from the file. It reads the **name** element from the **key** element and searches in the *key settings list* for the entry that has the same name. If such entry is found, the **action** element will be read and its contents will be applied to the users action in the entry. If no such entry is found, keytouch will ignore the **key** element and read the next **key** element.

If keytouch failed to read the keyboard file (reasons for this can be: the file is invalid or does not exist), it will not be a problem for the contents of the *key settings list*. This is because the list is already initialized and the actions are set to their default values (see subsection 3.2.2).

3.2.4 Preferences file

The contents of the preferences file will be read and the values of the elements will be copied to the *preferences structure*. If keytouch fails to read the preferences file, then the members of the *preferences structure* will be initialized by their default value:

- Documents directory: “~/Documents”
- Home page: “http://keytouch.sf.net”
- Search page: “http://www.google.com”
- Browser: “”
- E-mail program: “”
- Chat program: “gaim”

Note that when no browser is set (as you can see above, this is the default) keytouch and the “WWW Browser” plugin will automatically find out which browser it should use.

3.3 Output files

The most important output file is the keyboard configuration file. Keytouch will use the contents of the *key settings list* to generate this file for the current keyboard. The action chosen by the user will be used as the contents of the **action** element in the file. The keys name will be used as the contents for the **name** member in the file.

The second output file is the preferences file which will contain the contents of the *preferences structure*.

3.4 Plugins

KeyTouch has a plugin mechanism described in chapter 8. These plugins have functions that can be used as actions for the extra function keys. The plugins are spread over two directories: in a system-wide-readable directory (we will call this the “system plugin directory”) and in `~/.keytouch2/plugins/` (we will call this the “user plugin directory”). We call the plugins in the system plugin directory “system plugins” and in the user plugin directory “user plugins”. Note that the plugins that are included with the keyTouch package are system plugins.

When keytouch starts it will initialize the *plugin list* by loading information about all available plugins into it. It will first load the information about the system plugins and then from the user plugins. A user plugin will not be loaded if it has the same name (note: the plugins name is not its file name) as one of the system plugins. Keytouch will read the following information about the plugin:

- Name;
- File name;
- Version;
- A description that gives the user an idea of what the plugin can do;
- The plugins author;
- A list of function names.

A small implementation description might be useful here. KeyTouch plugins are dynamically loadable libraries, that contain a structure describing the information listed above. Keytouch loads them and copies the contents of this structure to the programs memory. Because there is no reason for keeping the plugin loaded, keytouch will unload it.

The user can add plugins to the *plugin list*. The user will have to give up the file name of the plugin to do this. Keytouch will then load the plugin and read the plugin information. If this is successfully done and the name of the plugin does not already appear in the *plugin list*, the plugin will get an entry in the *plugin list* and the plugin will be copied to the user plugin directory. Note that the file name that the plugin will get in the user plugin directory, will be equal to the name that is defined in the plugins structure.

The user can also remove plugins from the *plugin list*. This can only be done for user plugins. When the entry of a plugin is removed from the list, the corresponding file in the user plugin directory will also be removed.

3.5 The application tree

To make it easier for the user to choose an application, keytouch provides an application tree where the user can select the application. This menu tree is build using a menu file that describes the structure of the menu tree. The specification of this menu file can be found at <http://standards.freedesktop.org/menu-spec/latest/>. The file name of menu file is 'applications.menu' and is located in {datadir}/keytouch/. The location of '{datadir}' depends on how keyTouch is installed. If keytouch fails to read the menu file, the application tree will be empty.

Chapter 4

The keytouch-keyboard program

The keytouch-keyboard program provides a graphical user interface for choosing the keyboard model. The user can also import keyboard files so that he/she can choose the imported keyboard model.

Keytouch-keyboard needs to be executed by the root user, because the chosen model is a setting that will be used by all users on the system and keytouch-keyboard calls keytouch-init (keytouch-init needs to have root permissions).

4.1 Data structures

Keytouch-keyboard has one list containing the model and manufacturer names of all supported keyboard models. This list is called the *keyboard list*.

4.2 Input files

4.2.1 The current keyboard file

Keytouch-keyboard reads the model and manufacturers name of the currently chosen keyboard from the current keyboard file. If it fails to read this file, then the model and manufacturer will be empty.

4.2.2 Keyboard files

Keytouch-keyboard reads every file that is located in the directory for the keyboard files. It checks the syntax of every file and if the file is correct, then the name of the keyboard will be added to the *keyboard list*.

The user can also import a keyboard file. In this case the user will have to specify the location of the keyboard file and keytouch-keyboard will do a syntax check on that file. The file will be copied (see the “Output files” section) to the keyboard files directory and the keyboards name will be add to the *keyboard list* if its syntax is correct.

4.3 Output files

4.3.1 The current keyboard file

If the user has chosen the correct keyboard model and confirms this choice, the model and manufacturer name of the keyboard will be written to the current keyboard file.

4.3.2 Keyboard files

If the user wants to import a keyboard file, then this file will be copied to the keyboard files directory. The file will have the following name: '{model}.{manufacturer}'. Here '{model}' is replaced by the keyboards model name and '{manufacturer}' by the keyboards manufacturer name.

4.4 The keytouch-keyboard script

The binary for keytouch-keyboard is named 'keytouch-keyboard-bin' and is called by a script 'keytouch-keyboard'. The reason that a script is used for calling the binary is that the binary needs root privileges. The script will run a program that has a graphical user interface for running keytouch-keyboard-bin as root. If the program 'gksu' is found on the system, then this program will be used. If 'gksu' is not found, then the 'kdesu' program will be used.

The keytouch program will be started after the execution of keytouch-keyboard-bin, if the keytouch-keyboard script was called with the '-restart-keytouch' parameter. This parameter is used by the keytouch program when keytouch calls keytouch-keyboard.

4.5 Calling keytouch-init

After keytouch-keyboard has written the current keyboard file, it will call the keytouch-init program to get the extra function keys of the current keyboard working (see chapter 6).

Chapter 5

The keytouchd program

The keytouchd program runs at the background and waits for an extra function key to be pressed. When an extra function key is pressed, the action chosen by the user will be performed.

5.1 Data structures

There are 3 important data structures that are used by keytouchd:

- A *key settings list* contains an entry for every extra function key of the current keyboard. Each entry contains the key's name, the key's X keycode and the action chosen by the user. It also contains a boolean that indicates whether auto repeat mode of the key should be enabled or disabled. In other chapters a *key settings list* may also be used, but will not have the same contents as the list meant in this chapter.
- A *preferences structure* contains the user preferences that were read from the preferences file (see section 2.3).
- A *plugin list* that has an entry for each plugin. Such entry contains the plugin's name, file name, version number, author, description and a list of plugin functions.

5.2 Input files

5.2.1 Current keyboard file

The manufacturer and model name of the keyboard that was selected by the user will be read from the current keyboard file (see section 2.4). If this file does not exist or is invalid, keytouchd will exit.

5.2.2 Keyboard file

Keytouchd will read the keyboard file, that belongs to the model and manufacturer read from the current keyboard file. It reads the **name** and **keycode**

element from every **key** element in the keyboard file and puts them in the *key settings list*.

The *key settings list* will be initialized when the keyboard file is read. As described above the names and keycodes of the keys will be read from the keyboard file. However entries in the *key settings list* also have a member that contains the action chosen by the user and the boolean that indicates auto repeat mode, so these will also need to be initialized.

The action member will be initialized as an empty command, so that nothing will happen when the extra function key is pressed. Note that if the key has a **key** element in the keyboard configuration file then the action specified there will be used as the action. This means that if, for some reason, there is no such **key** element in the keyboard configuration file then nothing will happen when the key is pressed. You might wonder why the default action is not chosen in such situation. Well, such situation occurs for example when the user never use the keytouch program before an no keyboard configuration file exists. So it is very likely that the user does not want anything to happen when he/she presses an extra function key and maybe the user is not even aware that keytouchd is running.

The value of the boolean indicating the auto repeat mode of the key depends on the keys keycode. If the keycode “VOLUMEUP” or “VOLUMEDOWN”, then the boolean will be true and otherwise it will be false. When auto repeat is on for a key this means that when the key is hold down, the X server will send key press events until the key is released. When auto repeat is off then the X server only sends one key press event when the key is pressed. Like you may conclude from the above, auto repeat will only be turned on for the volume up/down keys (that is when they have the correct keycodes), because only for those keys auto repeat is a desired mode.

The keycodes stored in the *key settings list* are not the same as in the **keycode** elements of the keyboard file. The keycodes in the keyboard files are aliases for kernel keycodes and the keycodes in the *key settings list* are X keycodes. So first the aliases will be converted to kernel keycodes and then from kernel keycodes to X keycodes. The translation from kernel- to X keycodes is done using a fixed translation table. This translation table is made for the X.org server using the ‘kbd’ keyboard driver on an x86 platform.

Keytouchd will exit when fails to read the keyboard file.

5.2.3 Keyboard configuration file

After reading the current keyboard file and the keyboard file, the keyboard configuration file for the current keyboard will be read. Keytouch reads every **key** element from the file. It reads the **name** element from the **key** element and searches in the *key settings list* for the entry that has the same name. If such entry is found, the **action** element will be read and its contents will be applied to the users action in the entry. If no such entry is found, keytouch will ignore the **key** element and read the next **key** element.

If keytouchd fails to read the keyboard file (reasons for this can be: the file is invalid or does not exist), then it will exit.

5.2.4 Preferences file

The contents of the preferences file will be read and the values of the elements will be copied to the *preferences structure*. Keytouchd will exit when it fails to read the preferences file.

5.3 Grabbing the keys

When the *key settings list* is filled keytouchd is ready to grab the extra function keys. This is done by calling the XGrabKey function of Xlib. As a result the keytouchd program will receive an X11 event when the extra function key is pressed. If another X client has already grabbed the key, then XGrabKey will fail, but keytouchd will continue. After calling XGrabKey, the keys auto repeat mode will be set.

5.4 When a key is pressed

After a grabbed extra function key is pressed, keytouchd will receive an X11 key press event. As a result of the XGrabKey function call, the whole keyboard is actively grabbed after receiving the key press event. Because keytouchd does not want to grab the whole keyboard, it will call the XUngrabKeyboard function before doing anything else.

After calling XUngrabKeyboard, keytouchd will search for the pressed key's entry in the *key settings list* and looks what action to perform. If the action is a program (and its command is not empty) then it will execute its command. If the action is a plugin function and this function is found in the *plugin list* then this plugin will be executed.

There are two types of plugin functions: “key functions” and normal functions. Keytouchd will emulate a shortcut key press that depends on the name of the currently active window, when the function is a key function. Normal functions are the wellknown program functions and will be called by keytouchd. The *preferences structure* will be used as a parameter of a normal function. For more information about plugins see chapter 8.

5.5 Plugins

KeyTouch has a plugin mechanism described in chapter 8. These plugins have functions that can be used as actions for the extra function keys. The plugins are spread over two directories: in a system-wide-readable directory (we will call this the “system plugin directory”) and in `~/.keytouch2/plugins/` (we will call this the “user plugin directory”). We call the plugins in the system plugin directory “system plugins” and in the user plugin directory “user plugins”. Note that the plugins that are included with the keyTouch package are system plugins.

When keytouchd starts it will initialize the *plugin list* by loading information about all available plugins into it. It will first load the information about the system plugins and then from the user plugins. A user plugin will not be loaded if it has the same name (note: the plugins name is not its file name) as one of the system plugins.

Unlike `keytouch`, `keytouchd` does not unload the plugin after reading the information, because it needs to use the functions when an extra function key is pressed. How `keytouchd` calls these functions is described in section 5.4.

5.6 How `keytouchd` is started

`Keytouchd` will be started when the user starts a new X session. The program is started by the script `/etc/X11/Xsession.d/40keytouchd`. If the directory `/etc/X11/Xsession.d/` does not exist then the script `/etc/X11/Xsession` will start `keytouchd`.

`Keytouchd` is killed when the user saves the configuration in `keytouch`. After the program is killed, it will be started again by `keytouch`.

Chapter 6

The keytouch-init program

The task of the keytouch-init program is to assign kernel keycodes to the scan-codes of the extra function keys to get these keys working.

6.1 Data structures

Keytouch-init has a list called the *key settings list*, that contains for each extra function key the its scancode and kernel keycode.

6.2 Input files

6.2.1 Current keyboard file

The manufacturer and model name of the keyboard that was selected by the user, will be read from the current keyboard file (see section 2.4). If this file does not exist or is invalid, keytouch-init will exit.

6.2.2 Keyboard file

Keytouch-init will read the keyboard file, that belongs to the model and manufacturer read from the current keyboard file. It reads the **scancode** and **keycode** element from every **key** element, whose **key-type** attribute is not *acpi-hotkey*, in the keyboard file and puts them in the *key settings list*.

Keytouch-init will exit when it fails to read the keyboard file.

6.3 Setting the keycodes

For each key in the *key settings list* keytouch-init will set the keycode to its scancode. The code that sets the keycode is that same as the code used in the 'setkeycodes' command. Keys with scancode 0 will be ignored. This is usefull for keyboard files for USB keyboards, because for those keyboards it is not possible to set keycodes and the keyboards do not send scancodes.

Chapter 7

The keytouch-acpid program

The keytouch-acpid program provides support for ACPI hotkeys. It listens for ACPI events and when it receives an event that was described in the keyboard file, then it will simulate a key press event that will be received by keytouchd.

7.1 Data structures

Keytouch-acpid has a list called the *key list* that contains for each extra function key, that is an ACPI hotkey, its event description and X keycode.

7.2 Input files

7.2.1 Current keyboard file

The manufacturer and model name of the keyboard that was selected by the user will be read from the current keyboard file (see section 2.4). If this file does not exist or is invalid, keytouch-acpid will exit.

7.2.2 Keyboard file

Keytouch-acpid will read the keyboard file, that belongs to the model and manufacturer read from the current keyboard file. It reads the **event-descr** and **keycode** element from every **key** element, whose **key-type** attribute is *acpi-hotkey*, in the keyboard file and puts them in the *key list*.

The keycodes read from the keyboard file are converted to X keycodes in exactly the same way as described in subsection 5.2.2.

Keytouch-acpid will exit when it fails to read the keyboard file.

7.3 How it works

Keytouch-acpid opens the ACPI socket file `/var/run/acpid.socket` and waits to read an event from the socket. When a event is read, keytouch-acpid will first check if the current keyboard file changed and if it changed then the *key list* will be cleared and the keyboard file of the new current keyboard will be read. After that the program will search for the received event description in the *key*

list. If it finds an entry then a key press event, with the keycode specified in that entry, will be simulated using the XTest extension.

7.4 How keytouch-acpid is started

Keytouch-acpid will be started when the user starts a new X session. The program is started by the script `/etc/X11/Xsession.d/39keytouch-acpid`. If the directory `/etc/X11/Xsession.d/` does not exist then the script `/etc/X11/Xsession` will start keytouch-acpid.

Chapter 8

Plugins

KeyTouch has a plugin mechanism, where the plugins have functions that can be used as actions for the extra function keys. These plugins are dynamically loadable libraries. This chapter describes how keyTouch plugins are structured and how they can be made.

Every plugin needs to include the plugin.h header file that contains the following code:

```
#define TRUE    1
#define FALSE  0

typedef char Boolean;

typedef struct {
    char    *documents_dir,
            *home_page,
            *search_page,
            *browser,
            *email_program,
            *chat_program;
} KTPreferences;

typedef struct {
    struct {
        char    *name;        /* This string must appear in the name of the window */
        Boolean is_end;      /* If TRUE name appears at the end of the
                             * window's name, otherwise at the beginning */
    } window_name;
    unsigned int state;     /* The state of the key-press event to send */
    char        *keysym;    /* The keysym of the key to send */
} KTPPluginKey;

typedef struct {
    int          num_programs;
    KTPPluginKey *key;
```

```

} KPluginKeyFunction;
/* The structure above contains information about shortcuts.
 * Each element of this array contains shortcut information
 * for one specific application. All shortcuts perform the
 * same action.
 * The window name of the last element may be
 * NULL. In this case the shortcut will be used for every
 * application that does not appear in the array. */

typedef enum {
    KPluginFunctionType_Key,
    KPluginFunctionType_Function
} KPluginFunctionType;

typedef struct {
    char *name;
    KPluginFunctionType type;
    union {
        KPluginKeyFunction key_function;
        void (*function) (KTPreferences *preferences);
    } function;
} KPluginFunction;

typedef struct {
    struct {
        char *name;
        char *author;
        char *license;
        char *version;
        char *description;
    } info;
    char *file_name;
    int num_functions;
    KPluginFunction function[];
} KeytouchPlugin;

```

8.1 The `plugin_struct` structure

A keyTouch plugin must have a global KeyTouchPlugin named *plugin_struct* whose members have initial values. The *info* member needs to be initialized with the plugin's name, author, license, version number (as a string) and a description. The *file_name* member should contain the file name (without a path) that the plugin will get when it is installed. The *num_functions* member contains the number of plugin functions that the plugin has. This number is equal to the number of elements of the *function* member which is an array of type KPluginFunction describing each function.

The KPluginFunction elements need have initial values as followed. The *name* member will be the name of the function that will be visible to the user.

The *type* member can have one of the two possible values: `KTPluginFunctionType_Key` or `KTPluginFunctionType_Function`. The value of the *function* member depends on the value of this *type* member. If *type* is `KTPluginFunctionType_Key`, then the *key_function* member of the union will be used. If *type* is `KTPluginFunctionType_Function`, then the *function* member of the union will be used.

For a function of type `KTPluginFunctionType_Function`, the *function* member of the *function* union should contain the address of the function (so this is a real program function) that should be called.

For the contents of the *key_function* member of the *function* union see section 8.3.

8.2 Real functions

Let us call the functions, pointed to by the *function* union in a `KTPluginFunction`, real functions. Such functions should take one argument that is a pointer to a `KTPreferences` structure. When the function is called, this structure will contain the values that were read from the preferences file. The function should not change the contents of the structure.

Note that the execution of the function should not take too long. When `keytouchd` calls the function, `keytouchd` will continue after the function returns. So until the function returns, `keytouchd` cannot process key press events. When the execution of the function will take too much time, then the function should create a child process that does the execution. For example:

```
void
my_function (KTPreferences *preferences)
{
    if (fork() == 0)
    {
        /* Put the code that needs a lot of time here */
        exit (0);
    }
}
```

If you are not familiar with the `fork` function you should read its manual page. Note that the `exit` call in the code is very important. If this is not done, there will be two `keytouchd` processes, and that is definitely not what we want.

Creating a child process (using `fork`) is very useful when you want the function to execute a program:

```
void
my_function (KTPreferences *preferences)
{
    if (fork() == 0)
    {
        execlp ("sh", "sh", "-c", preferences->browser, NULL);
    }
}
```

This code executes the browser that was set by the user. You will notice that the `exit` call is left out here. This is because the child process created by `fork` will be replaced the command that was set as parameter of the `execlp` function. So when that command exits, the child process will also exit (in fact the command IS the child process).

8.3 KPluginKeyFunction

When a plugin function has the type `KPluginFunctionType_Key` then it is not a normal program function that will be called. A `KPluginKeyFunction` structure will contain a list of key shortcuts. One, depending on the name of the window that has input focus, of the key shortcuts will be simulated. A `KPluginKeyFunction` structure has a member named `num_programs` that contains the number of elements of the array pointed to by the other member `key`. This array is an array of type `KPluginKey`. A `KPluginKey` has 3 members: `window_name`, `state` and `keysym`.

`window_name` describes the title of the windows on which the key shortcut should be applied. Its first member, `name`, contains a string that must appear at the beginning or the end of the window's name. Whether it should appear at the end or the beginning is indicated by its second member, `is_end` (which is `TRUE` when it should appear at the end). If the `name`, of the last `KPluginKey` in the array pointed to by the `key` of the `KPluginKeyFunction`, points to `NULL`, then that key shortcut will be used if the name of the window that has input focus, does not match any of the other names in the array.

`state` describes the state of the modifiers keys during the key simulation, which is the bitwise inclusive OR of one or more of the button or modifier key masks: `Button1Mask`, `Button2Mask`, `Button3Mask`, `Button4Mask`, `Button5Mask`, `ShiftMask`, `LockMask`, `ControlMask`, `Mod1Mask`, `Mod2Mask`, `Mod3Mask`, `Mod4Mask`, and `Mod5Mask`. These masks are defined in the `<X11/Xlib.h>` header file.

`keysym` contains a string that is the name of the keysym of the key to simulate.

Here an example of an `KPluginKey` array:

```
KPluginKey key_zoom_in[] = {
    {"Firefox", TRUE},    ControlMask,    "equal"},
    {"Konqueror", TRUE}, ControlMask,    "equal"},
    {"KPDF", TRUE},      ShiftMask | ControlMask, "plus"},
    {"KView", TRUE},     ShiftMask | ControlMask, "plus"},
    {"digiKam", TRUE},   ShiftMask | ControlMask, "plus"},
    {NULL, TRUE},        ShiftMask,      "plus"}
};
```

The example above shows key shortcuts for zooming in. The most common shortcut for this is `Shift+Plus` key combination (see last element in the array), so that shortcut will be used for all other applications that are not listed in the array. For windows whose name ends with "Firefox" or "Konqueror" the key combination `Ctrl+Equal` will be used. For windows whose name ends with "KPDF", "KView" or "digiKam" the key combination `Ctrl+Shift+Plus` will be used.

Note the simulated key events will be sent to the window that has input focus. This means that the events will not be received by the window manager and as a result window manager shortcuts (Alt+F4 for example) will not work.

8.4 Implementation

The structures described above can be programmed with initial values using the C99 standard. So the declaration of the *plugin_struct* variable may look like:

```
KeytouchPlugin plugin_struct = {
    {"Plugins name", "Author", "License", "x.x", "Description"},
    "filename.so",
    4, /* <= number of functions */
    {"Function 1", KTPluginFunctionType_Function, {.function = func1}},
    {"Function 2", KTPluginFunctionType_Function, {.function = func2}},
    {"Function 3", KTPluginFunctionType_Key,      {.key_function = {1, func3}}},
    {"Function 4", KTPluginFunctionType_Key,      {.key_function = {3, func4}}}
}
};
```

Prototypes of the functions *func1* and *func2* functions and the declarations of the *func3* and *func4* array, of course need to appear before the declaration of *plugin_struct*:

```
void func1 (KTPreferences *preferences);
void func2 (KTPreferences *preferences);

KTPluginKey func3[] = {
    {NULL, TRUE}, ControlMask | ShiftMask, "a"}
};

KTPluginKey func4[] = {
    {"Window name 1", TRUE}, ControlMask, "a"},
    {"Window name 2", FALSE}, 0, "F1"},
    {NULL, TRUE}, ControlMask, "c"}
};
```