

# **Tavultesoft Keyboard Manager**

---

---

## **User's Guide and Reference**

**VERSION 3.2**

**Tavultesoft**

This documentation may be freely copied, but the copyright notice must not be altered or removed. No part of this documentation may be modified or edited. Tavultesoft holds no responsibility for any errors in this documentation or the use of its software.

© 1994 Marc Durdin / Tavultesoft. All rights reserved.

This documentation was created in Word for Windows 6.0.

Microsoft, Word for Windows, Access, and Excel are registered trademarks, and Windows is a trademark of Microsoft Corporation.

Ami Pro is a registered trademark of Lotus Corp.

Compaq is a registered trademark of Compaq Computer Corporation.

Any other trademarks referred to remain the property of their respective holders.

# C O N T E N T S

<b>Chapter 1</b>	<b>Introduction.....</b>	<b>5</b>
	Setting Up .....	5
	Before You Run Setup.....	5
	Running Setup .....	6
	Keyman Quick Start.....	6
	Starting Keyman .....	6
	Beginning Work with Keyman.....	7
	New Features in Tavultesoft Keyboard Manager 3.1 .....	8
<b>Chapter 2</b>	<b>Using Keyman.....</b>	<b>9</b>
	The Keyman Window .....	9
	Using the Options Dialog Box.....	10
	Loading and Unloading Keyboards.....	10
	Configuring the Keyboard Buttons .....	11
	Changing the Default Hotkeys .....	13
	Keyboard Hotkeys .....	14
	Conclusion .....	14
<b>Chapter 3</b>	<b>Advanced Keyman Functions.....</b>	<b>15</b>
	Starting Keyman When Windows Starts .....	15
	Adding Keyboards to the Keyman Startup List.....	15
	Using Macros to Switch Languages .....	16
<b>Chapter 4</b>	<b>Writing a Simple Keyboard Program.....</b>	<b>19</b>
	Overview .....	19
	Arranging the Layout on the Keyboard .....	19
	Writing the Keyboard Program.....	20
	Comments and Blank Lines.....	20
	The Header .....	21
	The Rules.....	23
<b>Chapter 5</b>	<b>Further Programming.....</b>	<b>31</b>
	Multiple Groups.....	31
	Constraints .....	32
	Groups Without the “using keys” Keyword .....	33
	Virtual Keys.....	33
	Other Features .....	35
<b>Chapter 6</b>	<b>TIKE.....</b>	<b>37</b>
	Overview on Using TIKE .....	37
	Creating, Loading, Saving, and Editing Files.....	38
	Selecting an Editing Font .....	38
	Searching and Replacing Text.....	38
	Testing Your Keyboard .....	38
	Character Map .....	39

Printing .....	39
Reference .....	39
<b>Appendix 1 The KEYMAN.INI File .....</b>	<b>41</b>
The [Options] Section.....	41
The [Advanced] Section .....	42
The [Extensions] Section.....	42
The [TIKE] Section .....	43
<b>Appendix 2 Some Keyboard Templates .....</b>	<b>49</b>

---

## CHAPTER 1

# Introduction

Welcome to the Tavultesoft Keyboard Manager. With the Tavultesoft Keyboard Manager (Keyman), it becomes practical to enter and edit documents that use languages and scripts other than English, for a wide variety of Windows application programs such as word processors, spreadsheets, databases, and desktop publishers.

Keyman has been developed with particular reference to the languages of South-East Asia and their scripts, but it can be readily adapted for many other languages. Keyman will allow you to mix many languages in one document, in your favorite word processor.

This manual will guide you through the basics of using Keyman and writing keyboards for it, and explain the more advanced options that Keyman gives you. Reference information is given in the *Reference Documentation*.

## Setting Up

You install Keyman on your computer using the program SETUP.EXE. The Setup program installs Keyman, the help file, sample keyboards, and adds icons to Program Manager.

---

**Important** You cannot simply copy files from the distribution disks to your hard disk and run Keyman. You must use the Setup program, which decompresses and installs the files in appropriate directories.

---

## Before You Run Setup

Before you install Keyman, you should make sure that your computer meets certain hardware and software requirements.

### Check the Hardware and System Requirements

To run Keyman, you must have certain hardware and software installed on your computer. The system requirements include:

- Any IBM®-compatible machine with a 80286 processor or higher.
- A hard disk.
- A 5 1/4" or 3 1/2" floppy drive.
- An EGA, VGA, Hercules or better display.
- One megabyte of memory.

- A mouse.
- Microsoft® MS-DOS® version 3.1 or later.
- Windows version 3.1 or later in standard or enhanced mode.

### Running Setup

When you run the Setup program, you'll set a path for Keyman and then select the Keyman files you want to install.

#### → To start Setup

1. Insert the Keyman Setup Disk in drive A: or B:
2. From the **File** menu of the Program Manager or File Manager, choose **Run**.
3. Type **a:setup** or **b:setup**, depending on which drive you are installing from.
4. Type the directory to install to, or press ENTER to use the default.
5. Select the keyboards from the list that you wish to be loaded automatically when Keyman starts. (Note: it will be easier to start using Keyman if you select at least one keyboard during installation.)
6. Follow any other instructions until Setup is finished.

### Keyman Quick Start

This documentation assumes you are familiar with basic Windows techniques. To brush up on these skills, review your Windows documentation.

---

**Note** If you're experienced with using SIL CC, you'll easily learn how to create keyboards with Tavultesoft Keyboard Manager. However, you should read the sections on how to use Keyman and the introductory programming section, because there are some things different in Keyman.

---

The rest of this chapter presents the basic skills and concepts that you need to start using Keyman with various applications. This is only a brief overview, and will help you in starting to use Keyman, so you should read later chapters for more information.

### Starting Keyman

Setup automatically creates a Tavultesoft Keyboard Manager group and one or more Keyman icons in Program Manager.

### → To start Keyman

- ☞ Double-click on the Keyman icon in Program Manager.
- ☞ Use CTRL+TAB to switch through the active groups until you get to the Keyman group, cursor through it to the Keyman item, and press ENTER.

Keyman will start, showing a brief introductory screen while loading. Normally, Keyman will be hidden from view after it has started. If the Keyman window appears, you can select the **Hide** button to hide Keyman from view. As Keyman works the same way as any ordinary Windows application, you can exit Keyman at any time, or switch to it, unless the window is hidden.

## Beginning Work with Keyman

Once Keyman is loaded, you will notice that there will be some more buttons in the title bar, next to the control menu. These buttons are called *keyboard buttons*.

---

**Note** The instructions and descriptions in this section apply only when you have just installed Keyman. If you change any settings in Keyman, some of the hotkeys or instructions may be incorrect.

---



**Figure 1.1** The Program Manager title bar after loading Keyman

Each one of these keyboard buttons is associated with a keyboard program that Keyman has loaded. Most keyboard programs define the keyboard layout for a foreign language.

### → To type in another language

1. Select the language you want to use in either of the following ways:
  - ☞ Click on the appropriate keyboard button in the title bar.
  - ☞ Press the hotkey for that keyboard, or ALT+= if it was the last keyboard used. (See **Keyboard Hotkeys** for more information on this).
2. Select a font that is appropriate for that language in your application.
3. Start typing in that language.

After you have selected the right font and keyboard button, Keyman will interpret the keys you type and convert them to the new language. If you are using a TrueType font for your foreign language, you can use bold, italic, and other font styles without any problems.

Most of the time, you will still want to be able to type in English.

➔ **To type in English**

☞ Turn off any active keyboard button by clicking on it.

☞ If any keyboard button is activated, press ALT+= to turn it off.

## **New Features in Tavultesoft Keyboard Manager 3.1**

The following table lists the features introduced since version 3.0 of Keyman and tells you where to look for more information on them:

<b>New Feature</b>	<b>For More Information</b>
Virtual key input	Chapter 5
Caps Lock support	Chapter 5
Hotkey to bring up a menu of loaded keyboards	Chapter 2
Setting hotkeys to any virtual key	Chapter 2,4,5



---

## CHAPTER 2

# Using Keyman

Learning how to use the Tavultesoft Keyboard Manager is very easy and should take less than fifteen minutes. First you will learn about the Keyman window, and then the more advanced features of the Options window.

This chapter provides instructions for the use of Keyman, not modifying or creating new keyboard programs. For an overview on creating keyboard programs, see chapter...

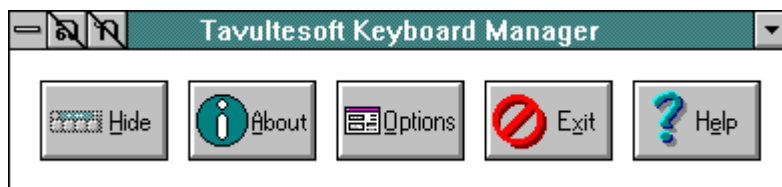
## The Keyman Window

Normally, when you start Keyman, its window is hidden from view. This is so that it will be out of the way; inexperienced users will not be confused by having another window floating around. However, most users will want to be able to use the Keyman window. There are several different ways to show the Keyman window.

### ➔ To show the Keyman window

- ☞ Click the right mouse button on any keyboard button, or double click on the Keyman icon in Program Manager.
- ☞ Press CTRL+ALT+= to display a menu of possible items; select **Show Keyman**.

The Keyman window will appear; it will look like the picture shown below:



**Figure 2.1: The Keyman window**

As you can see, the Keyman window is very simple, and there is little to learn. The five buttons in the Keyman window, **Hide**, **About**, **Options**, **Exit** and **Help**, can be clicked on with the mouse or are available via the keyboard by pressing the underlined letter shown on the button.

The first button, **Hide**, is pretty obvious; it hides the Keyman window from view. The window may be shown again by following the instructions above.

The **About** button, is likewise, easy to understand. When you select this button, Keyman will display a picture and information about the version of Keyman, as well as copyright messages.

The Options button brings up the Options dialog box. The Options dialog box is described later on in this chapter.

The Exit button exits Keyman and unloads all keyboard programs from memory.

Help runs Windows Help with Keyman Help loaded.

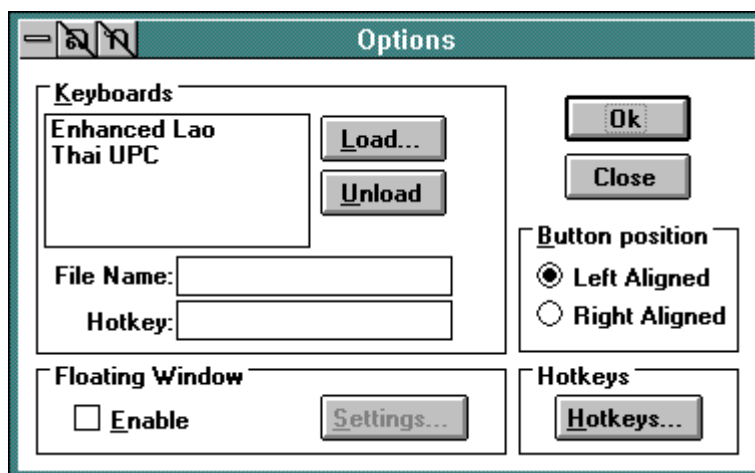
## Using the Options Dialog Box

The Options dialog box appears when you click the Options button in the Keyman window. It allows you to load and unload keyboards, configure the keyboard buttons, and set general hotkeys for Keyman.

---

**Note** You will notice, if you used Keyman 3.0, that the dialog box is smaller and has less in it than the previous version. This is because the debugging options have been removed, and the hotkey options moved to a separate dialog box.

---



**Figure 2.2** The Options dialog box

## Loading and Unloading Keyboards

You can load and unload keyboards using the Options dialog box. The Keyboards group has a list of the currently loaded keyboards, two boxes showing information about the keyboard you have selected in the list, and two buttons: Load and Unload.

### ➔ To get information on a loaded keyboard

1. Open the Options dialog box.
2. Select the keyboard you wish to get information about from the keyboard list.

The information will appear in the boxes below the list. You can scroll these boxes by clicking in them and moving right; however, you cannot edit them.

➔ **To load a keyboard**

1. Open the Options dialog box.
2. Click on the Load button.
3. Select the file name for the keyboard you wish to load, and click Ok.

The loaded keyboard will appear at the end of the list of keyboards, and its keyboard button will appear along with the others. (If Keyman detects a keyboard program error, it will be indicated, and the keyboard will not load.)

➔ **To unload a keyboard**

1. Open the Options dialog box.
2. Select the keyboard you wish to unload from the keyboard list.
3. Click the Unload button.

The keyboard will be unloaded, its name will be removed from the list, and its keyboard button will disappear.

## Configuring the Keyboard Buttons

The default placement and drawing options for keyboard buttons is for them to be drawn straight onto the active title bar. This works well in most situations, but in some programs, such as Ami Pro and Microsoft Word for Windows 6.0, the title bar is not a standard one, and when Keyman draws onto the title bar, it erases some things already there. Sometimes you might also use a program which uses the left side of the title bar for a quick menu, or a language switch (such as Microsoft Windows Thai Edition). You can move the keyboard buttons to the right hand side of the title bar in this case. Keyman can also use a "floating window" which can be placed either on the title bar, where it doesn't interfere with Ami Pro or Winword 6, or anywhere else on the screen.

➔ **To put the keyboard buttons on the right-hand side of the title bar**

1. Open the Options dialog box
2. Click on the Right Aligned radio button.

The keyboard buttons will move immediately to the right hand side of the title bar, and their order will change to right to left.

The Floating Window Settings dialog box lets you place the keyboard buttons anywhere on the screen

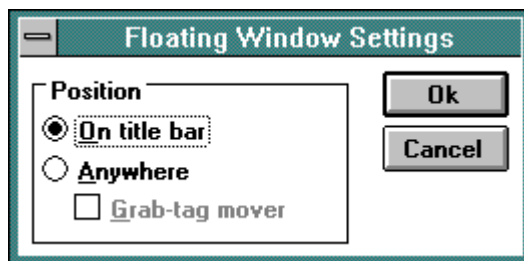


Figure 2.3: The Floating Window Settings dialog box

➔ **To put the keyboard buttons in the floating window on the title bar**

1. Open the Options dialog box.
2. Click on the Enabled checkbox in the Floating Window group.
3. Click on the Settings button in the same group.
4. Select On Title Bar in the dialog box shown below:
5. Click Ok to close the Floating Window Settings dialog box.

The keyboard buttons will be put in a floating window and will be on the title bar. Although this may not look any different, it works differently and may be more satisfactory with some applications, as mentioned above.

➔ **To put the keyboard buttons anywhere on the screen**

1. Open the Options dialog box.
2. Make sure the Enabled checkbox in the Floating Window group is checked.
3. Click on the Settings button in the same group.
4. Select Anywhere in the dialog box that appears
5. Check the Grab-tag mover checkbox.
6. Click Ok to close the Floating Window Settings dialog box.
7. Drag the floating window which will have appeared, via the tag on the edge of the window, to the location you want it on the screen.

The floating window will stay in that position until you drag it to another position, or you change the floating window settings. You can also lock the keyboard buttons in place, so that you cannot accidentally move them.

➔ **To lock the keyboard buttons in place**

1. Open the Options dialog box.
2. Click on the Settings button in the Floating Window group.

3. Make sure that the Grab-tag mover checkbox is unchecked.
4. Click Ok to accept your settings and close the dialog box.

The tag on the edge of the floating window will disappear and you will not be able to move the window.

## Changing the Default Hotkeys

Some programs may use the default hotkeys used by Keyman for other purposes; for example, Microsoft Excel uses ALT+= for a shortcut to entering a formula. If you commonly use these shortcuts, then you may wish to change the hotkeys set by default in Keyman. You may also wish to change them to a single key.

---

**Note** The reason the somewhat obscure ALT+= is used for the keyboard toggle hotkey is that when we were developing the first version of Keyman, we decided to make it similar to the hotkey to activate the control menu, which is ALT+SPACEBAR for application windows, and ALT+- for document windows. This hotkey also seemed to be unused. (We only started using Excel later!) Keyman 3.1 is the first version which allows you to change the default hotkeys.

---

There are three hotkeys you can change from the Set Hotkeys dialog box. They are (when installed): ALT+=, the keyboard toggle hotkey; CTRL+SHIFT+W, the hotkey to switch to English, and CTRL+ALT+=, the Keyman menu hotkey.

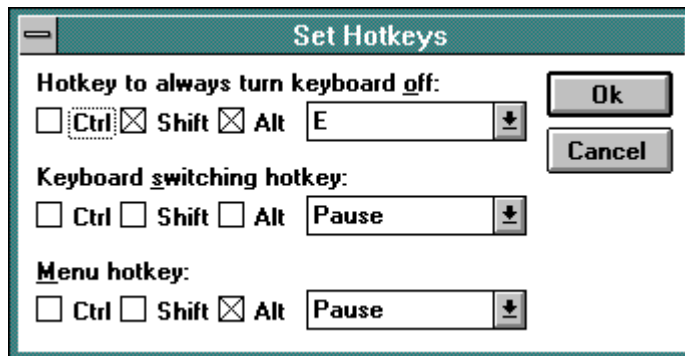


Figure 2.4: The Set Hotkeys dialog box

### → To change a hotkey

1. Open the Options dialog box.
2. Click on the Hotkeys button.
3. Select the Shift, Ctrl, and Alt states you want for the hotkey you are changing, and choose the hotkey you want to use from the dropdown list.
4. Click Ok to update the hotkey and close the dialog box.

A good key to use for toggle may be ` (backquote), or PAUSE. We have tried using CTRL+ALT+F1 for the English mode hotkey, CTRL+ALT+F2 for a second language (see the programming manual for information on how to change individual language hotkeys), and so on. Experiment!

### Keyboard Hotkeys

Each keyboard program has a hotkey that can be used to turn the keyboard on. This hotkey can be any combination of SHIFT, CTRL, ALT and any other key on the keyboard.

#### → To find out what hotkey a keyboard has

1. Open the Options dialog box.
2. Select the keyboard you want to know the hotkey of from the list.

- Or -

1. Press the Keyman menu hotkey (CTRL+ALT+=, unless you have changed it).

If you used the first method, the file name of the keyboard and the hotkey it uses will be shown in a box below the keyboard list. You cannot change this hotkey from the Options dialog box, only view it.

Otherwise, the hotkey will be shown to the right of the name of the keyboard in the menu.

### Conclusion

You should now be able to select keyboards in different ways, load and unload keyboards, change general hotkeys, and arrange the keyboard buttons to your preference.

---

## CHAPTER 3

# Advanced Keyman Functions

When using the Tavultesoft Keyboard Manager, you may wish to make macros that switch keyboards, or a database that selects the correct language for a field, for example. All recent Microsoft applications let you do this very easily; most other applications with a good macro programming language will also let you do it. If you didn't setup Keyman to run when you started Windows, you will also learn how to do this in this chapter, along with adding keyboards to be loaded when Keyman starts.

## Starting Keyman When Windows Starts

When using Program Manager, it is very easy to set up Keyman to load when Windows does.

---

**Note** If you are using a program shell other than Program Manager, such as Norton Desktop for Windows, or Compaq's Tabworks, these instructions may not apply. See the documentation for your shell program for more information on how to run programs when Windows starts, for these applications.

---

### → To load Keyman when Windows starts

1. Find the group **Startup** in Program Manager, or create it if it doesn't exist (File•New, select Program Group, type **Startup**, and press ENTER).
2. Find the Keyman icon.
3. Hold down CTRL and drag the Keyman icon to the Startup group.

The Keyman icon will be copied to the Startup group, and the next time you start Windows, Keyman will start automatically.

---

**Note** You can stop Keyman (and all the other programs in your Startup group) from loading by holding down SHIFT when the Windows logo appears after typing WIN at the DOS prompt.

---

## Adding Keyboards to the Keyman Startup List

When you installed Keyman, you selected a list of keyboard programs to load when Keyman was started. You may wish to change this list later on; to add a keyboard, or remove one.

**Note** If you are using a program shell other than Program Manager, such as Norton Desktop for Windows, or Compaq's Tabworks, these instructions may not apply. See the documentation for your shell program for more information on how to modify the command line for an icon.

---

➔ **To add a keyboard to or remove a keyboard from the Keyman startup list**

1. Select the Keyman icon, but don't start Keyman. (i.e. click once, not twice).
2. Press ALT+ENTER, or choose **File•Properties**.
3. Use TAB to move to the Command Line text box, or click in the box.

The text in the box should look similar to this:

```
C:\KEYMAN\KEYMAN mykbd.kmn minetoo.kmn
```

and there -h may be in the command line too.

The start of the command line is the Keyman program; each keyboard listed after that is loaded when Keyman starts. You can add a keyboard simply by putting a space at the end and typing its name, and you can remove one using BACKSPACE.

You can also make Keyman hidden at startup, or make it stay visible.

➔ **To hide Keyman or make it visible at startup**

1. Look for a -h in the command line.
2. If you wish to make Keyman visible at startup, delete this -h, otherwise, if it doesn't exist, add it to the end of the command line.

The changes will be ignored until the next time you start Keyman.

## Using Macros to Switch Languages

Most major application programs include macro programming languages of differing degrees of complexity. Using these macro languages, it is usually possible to make a language switch inside a macro. You may want to use this for a database, or a word processor, to switch languages and font at the same time for example. This documentation will talk only about Microsoft applications, although most others should be fairly similar. Check your application's documentation for information on how to do macro programming.

➔ **To switch languages in Microsoft Word 2.0 or later and Microsoft Excel 5.0 or later**

1. Create a new macro, or edit an existing one.



2. At the point where you want the language switch, in your macro, add a blank line and the statement `SendKeys` followed by the key combination used for hotkey for the keyboard you want to switch to, in quotes.

Use the following table for the characters used for CTRL, SHIFT, and ALT:

<b>For the shift key:</b>	<b>Use the character:</b>
ALT	%
CTRL	^
SHIFT	+

For example, to send the hotkey CTRL+SHIFT+L, to turn on the Lao keyboard, use:

```
SendKeys "^+L"
```

In Microsoft Access, the process is much the same; however, you must set up an event for when you enter the field; see Access's documentation for more information.

Switching back to English is very simple; instead of using a keyboard hotkey, use the hotkey specified in the Hotkey dialog box for the 'hotkey to always turn keyboard off.' You are probably better off avoiding the keyboard switch hotkey (usually ALT+=), because you will not know the state of the keyboard at the time you send it.

For hotkeys that are not straight letters or numbers, see the documentation for your application on how to represent them.



## CHAPTER 4

# Writing a Simple Keyboard Program

There are two main steps to writing a keyboard program. The first step is to arrange the layout of the characters on the keyboard. Then, you enter the keyboard program according to your layout.

This chapter shows you the basic steps in writing a keyboard program; we will be using a simplified French keyboard as an example to follow through.

---

**Important** Even if you have some experience in writing CC tables, you should still read this chapter, as it shows you the basic steps and structure of a keyboard file, which is slightly different to CC tables. However, your experience should help you learn the format more quickly.

---

## Overview

You will be designing a simplified French keyboard for people who don't know the standard French keyboard layout; you will have to go through both steps mentioned above. This French keyboard (Quick French) doesn't follow the standard French layout; instead, it uses a basic English keyboard with some deadkeys to define vowel diacritics and other French characters needed.

You will need to know the character codes in the font that goes with this keyboard (for Quick French, Times New Roman and Arial work fine); the characters you will be using may be upper-ascii. If you do not have a font for the language you will be working with, you will need to obtain or create one; Keyman does not do anything about fonts.

## Arranging the Layout on the Keyboard

First of all, you have to know what codes are used for the characters you are mapping with your keyboard program. (You can use the Character Map application that is provided with Windows to help you find these codes.)

For the Quick French keyboard, you will need all the vowels with different diacritics, some French symbols, and c-cedilla (upper and lower case); the codes needed are listed below, along with some others that are used by other European languages:

Char	Code	Char	Code	Char	Code	Char	Code	Char	Code	Char	Code
À	192	È	200	Ì	204	Ò	210	Û	217		
à	224	è	232	ì	236	ò	242	ù	249		
Á	193	É	201	Í	205	Ó	211	Ú	218	Ý	221
á	225	é	233	í	237	ó	243	ú	250	ý	253

Char	Code	Char	Code	Char	Code	Char	Code	Char	Code	Char	Code
Â	194	Ê	202	Î	206	Ô	212	Û	219	Ç	199
â	226	ê	234	î	238	ô	244	û	251	ç	254
Ä	196	Ë	203	Ï	207	Ö	214	Ü	220	«	171
ä	228	ë	235	ï	239	ö	246	ü	252	»	187

Now that you know the character codes needed, you must decide how you want to key them. For the purposes of this example, the codes will be keyed according to the following table:

Code	Keys to create code
à, Â, ...	backquote (`), followed by the corresponding vowel key
á, Á, ...	quote ('), followed by the corresponding vowel key
â, Â, ...	caret (^, SHIFT+6), followed by the corresponding vowel key
ä, Ä, ...	double quote ("), followed by the corresponding vowel key
ç, Ç	quote ('), followed by small or capital C.
«,»	double less-than symbol (<<) or double more-than symbol (>>)

The basic design of the keyboard is done. There will be more to come, but first you are going to write the first version of the keyboard program.

## Writing the Keyboard Program

Before you start this section, start a text editor (such as Windows Notepad), and begin editing a new file. The Tavultesoft Integrated Keyboard Editor has been designed for this purpose and works closely with Keyman (see chapter 6). (DOS text editors are okay, too, but not quite as convenient.)

A keyboard program is divided into two sections: the header, and the rules. The header lets you define the name of the keyboard, hotkeys, and other general settings. The rules are divided into groups where you define how the keyboard responds to keystrokes. Every keyboard program must have both of these sections. A keyboard program normally has the extension .KMN and is an ANSI text file.

---

**Note** A keyboard program is in ANSI text format, not ASCII text format, because Windows uses the ANSI character set. If you are using a DOS text editor, you must remember that the characters you see on the screen aren't necessarily the same as the ones you'd see if you used a Windows text editor, such as Notepad.

---

## Comments and Blank Lines

A Keyboard Program can have a comment at any point. The comment is identified by a letter 'c', with a space on either side (unless the comment is at

the start of the line, in which case you only need a space on the right). The end of the line marks the end of the comment.

Blank lines are ignored by Keyman; you can have them anywhere in a keyboard program.

Examples:

```
c This is a comment
... c This is a comment, also
```

It is a good idea to use comments to document your keyboard program. They will help you remember why you did something in a particular way, and they will also help other people understand the program.

Add some comments to the start of the Quick French keyboard program, in your text editor, describing who wrote it, when it was written, and anything else needed, such as instructions on how to use the keyboard (from the tables above). You could even add a copyright notice, for a keyboard that uses complex algorithms, for example.

Example:

```
c
c   Simplified French Keyboard Program for Keyman 3.2
c
c   This keyboard program uses a simplified set of keys
c   for typing French, especially for those who don't know the
c   standard French keyboard.
c
c   NOTE: This keyboard was created from the Keyman keyboard
c   programming tutorial.
c
c   Written by Anybody, 15 December 1994
c
```

## The Header

The header is easy to create; it consists of statements that help Keyman identify the keyboard and set default options for it. Each statement in the header must be on a separate line and is usually written with capital letters, although that is not required. Keeping the statements in the header in upper case helps you identify them easily, and keeps them consistent with keyboard programs other people might write.

Five statements are required in the header. Some other optional statements are described in later chapters. The required statements are: NAME, BITMAPS, HOTKEY, VERSION, and begin. The begin statement is usually written in lower case.

### The NAME Statement

The NAME statement tells Keyman the long descriptive name of the keyboard, which can be as long as eighty characters, unlike a file name. This name is

used in the Keyman menu, and in the Options dialog box. The name must be enclosed in double quotes ("). Any character except the double quote is legal within the name.

For our Quick French keyboard, we will use the name "Quick French." Add the NAME statement to the keyboard program, as follows:

```
NAME "Quick French"
```

You can give your keyboard program a different name, if you wish.

### The BITMAPS Statement

The BITMAPS statement tells Keyman which bitmap files are used for the keyboard button for that keyboard program, both on and off. The bitmaps use the standard Windows .BMP format; you can create them using Paintbrush. You can make them any size, but if you wish to have them on the title bar, then the height should be 18 pixels, for VGA. (See the *Reference Manual* for the heights for other standard screens.) The bitmaps can be monochrome or color, but the .BMP file must be less than 64 kilobytes. The BITMAPS statement accepts the names of the bitmap files, in upper or lower case, with or without the .BMP extension included, with the on bitmap filename first. The file names are **not** enclosed in quotes.

You can create your own bitmaps for the Quick French keyboard, or use two supplied bitmaps; FRKEY1.BMP and FRKEY0.BMP (on and off respectively). Add the BITMAPS statement, as follows:

```
BITMAPS FrKey1 FrKey0
```

If you make your own bitmaps, replace these names with your new filenames.

### The HOTKEY Statement

The HOTKEY statement tells Keyman the hotkey that will turn the keyboard on. You can represent the hotkey in two different ways; for now we will stick with the simpler one. The hotkey can have a combination of CTRL, SHIFT, ALT and a letter or number key. You represent the shift codes according to the following table:

<u>Shift key</u>	<u>Character to represent key</u>
CTRL	^ (caret)
SHIFT	+ (plus sign)
ALT	% (percent sign)

For example, to represent CTRL+SHIFT+Z, you would have "^+Z". The hotkey must be in double quotes.

For the Quick French keyboard, a good hotkey to use would be CTRL+ALT+F. (CTRL+SHIFT+F would also be okay, however, Winword 6 already uses that.) Add the HOTKEY statement to the program, as follows:

```
HOTKEY "^%F"
```

## The VERSION Statement

The VERSION statement is the simplest statement; for a keyboard intended for version 3.2 of Keyman, simply add 3.2 to the end of the line.

The Quick French keyboard is intended to be used in Keyman 3.2. Therefore, add the following line to your program:

```
VERSION 3.2
```

## The begin Statement

The begin statement tells Keyman which group to start processing with when it receives a keystroke. Later on you will learn how to use multiple groups to process keystrokes, but at present all you need to know is to include this line in the header.

For the Quick French keyboard, add the following line to tell Keyman to start in the Main group:

```
begin > use(Main)
```

## Conclusion

Those five statements are the only ones required in the header. You can add comments to the ends of the statements to help other people understand them.

Your Quick French keyboard should so far look like this:

```
c
c   Simplified French Keyboard Program for Keyman 3.2
c
c   This keyboard program uses a simplified set of keys
c   for typing French, especially for those who don't know the
c   standard French keyboard.
c
c   NOTE: This keyboard was created from the Keyman keyboard
c   programming tutorial.
c
c   Written by Anybody, 15 December 1994
c

NAME "Quick French"
BITMAPS FrKey1 FrKey0
HOTKEY "^%F"
VERSION 3.2

begin > use(Main)
```

## The Rules

Before we start on the rules, we will define some terms:

<b>Term</b>	<b>Definition</b>
rule	A rule tells Keyman what output is associated with a keystroke under

<b>Term</b>	<b>Definition</b>
context	certain conditions; it is divided into three parts: the context, key, and output The context specifies the conditions under which the rule is acted upon. It is compared with the most recent characters output.
key	The key is the code for a single keystroke that the rule acts on.
output	The output is the part of the rule that defines what characters are to be put on the screen when the rule's conditions are met.

### The Groups

The first thing to do is to define the group. There are two types of groups: one that processes the key pressed and the context and another that does further processing using only the context. For most purposes, the first type of group will do all you need. The group of rules ends at the next `group` statement or at the end of the file if there are no more `group` statements. We said in the section about the header that the group to be processed first would be identified by the `begin` statement.

The `begin` statement defined the first group as `Main`. In your program, add a new line:

```
group(Main) using keys
```

`using keys` tells Keyman that the group processes keystrokes as well as context. If you leave this out, the keystrokes will be ignored.

### Simple Rules

The simplest rule you can have tells Keyman to convert one key into another. A rule of this sort is represented in Keyman in the following way:

```
+ key > output
```

where *key* is the key to be translated, and *output* is the character to be output. The plus sign (+), is optional, but shows you that the next character in the string is the keystroke. More complex rules can have characters before the plus sign (the context). The right angle-bracket (>) tells Keyman which part of the rule is output and which part is the key and context. Single characters can be represented in several different ways; the possible methods are listed below:

<b>Representation</b>	<b>Example of the character "x"</b>
Inside single quotes	'x'
Inside double quotes	"x"
As a decimal number (base-10)	d120



As a hexadecimal number (base-16)      x78  
 As an octal number (base-8)              170

These different ways of representing a single character follow the SIL-CC conventions. The first three ways are the most often used, and octal representation is rarely used. Multiple characters (a *string*) can be represented in quotes simply by having more than one character in the string. You can have any combination of these representations in a rule, with spaces between them.

For example, to convert the key “a” to the character “z”, you would include the following line in your keyboard program:

```
+ 'a' > 'z'
```

Or, to convert “?” to “Hello World!”, you would have this line:

```
+ "?" > "Hello World!"
```

You can use either single or double quotes.

A use of the decimal representation is, for example, in the British English keyboard, where the hash sign (#) is converted into a pounds sign (£, decimal code 163):

```
+ '#' > d163
```

### Using the Context in More Complex Rules

Often you will want to know the previous characters that have been typed and translate the keystrokes accordingly. *Keyman remembers the characters that came out on the screen, and not the actual keys typed.* It is important to remember this, because some programs such as SIL’s Keyswap (for DOS) work with sequences of keys rather than characters. The characters that came out on the screen are called the *context*. The context is represented in a rule to the left of the keystroke, before the plus sign. For example,

```
"^" + "e" > "ê"
```

In this example, if you type a “^” (caret) followed by the letter “e”, it will come out with the European letter “ê”. The caret is the context, the letter “e” is the key, and the letter “ê” is the output.

You can add some of the rules to the Quick French keyboard program now. Add the rules for all the “a”-related characters; you will quickly see how many rules it would require for a complex keyboard. Another example rule for the Quick French keyboard program is:

```
'`' + 'a' > 'à'
```

## The Any, Index, and Store Statements

Keyman lets you translate a group of characters in one rule. It does this with the `any`, `index`, and `store` statements. A `store` statement creates a set of characters that can be operated on together under a name. The `any` statement lets you match a character in a store and the `index` statement lets you output a selected character from it.

A `store` statement comes between the `begin` statement and the first group. It must all be on one line. (No `endstore` statement is required as in SIL-CC; in fact, it is not supported.) The statement has the following syntax:

```
store(name) string
```

*name* is the name to give to the store. A store name can be up to 16 letters long, but it is usually best to keep it short. The name can be any combination of letters and numbers; spaces and punctuation characters are illegal. The second part of the statement, *string*, is the string to put in the store; it can use any combination of the character representations talked about in the previous sections. An example:

```
store(lwrvowel) 'aeiou'
```

In this example, Keyman will create a store called “lwrvowel”, and make the contents of the store equal to “aeiou”.

To use a store, you must have an `any` statement on the left hand side of a rule, and, optionally, a corresponding `index` statement on the right hand side.

An `any` statement allows you to designate a set of characters instead of a single character for the key or, as part of the context. The syntax of the statement is:

```
any(storename)
```

For example, you could have the following:

```
store(stops) '!?.'  
.  
.  
.  
+ any(stops) > 'GO!'
```

This example would convert any of the characters “!”, “.”, and “?” to a “GO!”. (This example is actually not very useful.)

But, to make the `any` statement useful, you really need to have a statement that lets you know the matched character. The `index` statement lets you do that.

The `index` statement lets you output a character in a store that is at the same position as the matched character from the equivalent `any` statement’s store. The `index` statement has the following syntax:

```
index(storename,offset)
```

The *storename* is obvious; however, the *offset* part needs some explaining. As Keyman allows you to have more than one *any* statement in a single rule, the *index* statements in that rule need to know which *any* statement they are to take their matched character information from. The *offset* parameter tells Keyman the position of the character of the *any* statement that is to be used, with the first character of the context having the offset 1. For example,

```
+ any(lwrvowel) > index(uprvowel,1)
```

This rule would convert all lower case vowels to upper case. Or,

```
any(stops) + any(lwrvowel) > index(stops,1) index(uprvowel,2)
```

This one capitalizes any lower-case vowel following a full-stop, question, or exclamation mark.

### The context Statement

If the context of the rule is not modified in the output, then you can replace the *index* statements on the RHS of the rule with a *context* statement. For example, the previous rule becomes:

```
any(stops) + any(lwrvowel) > context index(uprvowel,2)
```

This is faster and, for more complex rules, easier to read. Use the *context* statement wherever possible in preference to using *index* statements.

The Quick French example keyboard can make use of this quite easily; an example will be shown for “^” and a vowel:

```
store(vowel) 'aeiouAEIOU'
store(caret) 'âêîôûÂÊÎÔÛ'
.
.
.
'^' + any(vowel) > index(caret,2)
```

You should be able to add all the rest of the rules fairly easily. At present, leave out the “<”, “>”, and c-cedilla rules. For the “ý” and “Ý”, just add a single rule (don’t use *any* and *index*). You can now delete the single rules applying to “a”.

So far, your Quick French keyboard should look like this:

```
c
c   Simplified French Keyboard Program for Keyman 3.2
c
c   This keyboard program uses a simplified set of keys
c   for typing French, especially for those who don't know the
c   standard French keyboard.
c
c   NOTE: This keyboard was created from the Keyman keyboard
c   programming tutorial.
```

```
c
c Written by Nobody, 15 December 1994
c

NAME "Quick French"
BITMAPS FrKey1 FrKey0
HOTKEY "%F"
VERSION 3.2

begin > use(Main)

store(vowel) 'aeiouAEIOU'
store(caret) 'âêïôûĂĔİŎŪ'
store(acute) 'áéíóúÁÉÍÓÚ'
store(grave) 'àèìòùÀÈÌÒÙ'
store(colon) 'äëïöüĂĖİŎŪ'

group(main) using keys

'"' + 'y' > 'ý'
'"' + 'Y' > 'Ý'
'^' + any(vowel) > index(caret,2)
'"' + any(vowel) > index(acute,2)
'`' + any(vowel) > index(grave,2)
'!' + any(vowel) > index(colon,2)

c End of file
```

### Testing the Keyboard

Before you go any further, you should test your keyboard. Save your file, and open the Keyman Options dialog box. Select Load, and choose your keyboard. When you click Ok, the keyboard will load. If any errors occur, refer to the section Fixing Load-Time Errors.

Load a text-editor or word-processor, such as Notepad, or WinWord. Select the Quick French keyboard and try it out. Type sequences like `^a^e'a'e`. Once you are sure that that is all right, then try typing something like this: **“A problem in the keyboard.”** You can see the problem: when you type something in quotes, if the letter after the quote character is a vowel, it will be converted.

### Fixing the Problems

Open up your keyboard program again. The problem exists with two lines; both of the lines regarding quotes will need to be changed. But first you have to decide how you are going to represent the quote character when it is to be used as a quote character. Probably the easiest way is just to type it twice.

The line you need to add is this; this will fix it for double quotes; add another line to fix it for single quotes.

```
'""' + any(vowel) > '"' index(vowel, 3)
```

Another thing that would be nice is to make the diacritics as *deadkeys*. A deadkey is a key that does not come out on the screen when it is pressed, but is still remembered in the context. Many European keyboards use deadkeys.

We will show you the line needed; you will need to remove the old rule to do with carets and explain it for the caret (^) character:

```
+ '^' > deadkey(1)
deadkey(1) + any(vowel) > index(caret,2)
```

The deadkey, or dk statement accepts a number identifying it; it will not appear on the screen, but it does stay in the context. You can have up to 254 different deadkeys, starting from 1.

You will want to add the deadkey rules for all the other characters; don't forget to use a different deadkey identifying number for each one. You will also need to modify the quote modification statements talked about above, to work with the deadkey better; it becomes simpler, as shown below:

```
dk(2) + "'" > "'"
```

You should add this sort of statement for all the diacritics, in case you wish to use the original character.

There are some other characters we haven't got support for yet: «, », ç, and Ç. We decided to represent the “«” and “»” characters with double less-than and double more-than symbols. You should be able to add rules for these, as well as for the “ç” and “Ç” symbols.

But what if someone wanted to type “<<<<<<< \*\*\* >>>>>>>”, for instance, as a divider to a section of a book. They wouldn't want it to come out as “<<<<< \*\*\* >>>>>”. So we will make use of a deadkey to have it come out correctly, as shown below:

```
"«" + "<" > "<<" dk(5)
"»" + ">" > ">>" dk(5)
dk(5) + "<" > "<" dk(5)
dk(5) + ">" > ">" dk(5)
```

You should be able to see what this does. Test your keyboard again; there should not be any more problems. You have completed the Keyman keyboard tutorial.

You can extend this keyboard to add support for every European character—then you would call it EUROPE.KMN. The following chapter will explain some of the more advanced features of the Keyman keyboard language; you could use them to extend this keyboard.



---

## CHAPTER 5

# Further Programming

This chapter will build on what you have already learnt from chapter 4; it will be assumed that you understand the basic Keyman keyboard program format.

The following subjects will be discussed in this chapter:

- Multiple groups
- Constraints
- Groups without the “using keys” keywords
- Virtual keys
- Other features

## Multiple Groups

In chapter 4, you learnt how to create a file with one group. Multiple groups can be useful for doing further processing such as changing characters in certain contexts, or, as is done for some South-East Asian languages, syllable splitting.

### The use Statement

A group can be added with the `group` statement; the previous group ends at the line before the statement. However, to use the group, you must have a way of jumping from one group to another. The `use` statement lets you do this. This statement is legal on the right hand side of a rule; you can put it anywhere in the string, with one limitation: no `index` or `context` statements are allowed after the `use` statement; using them will cause run-time errors.

Any output in a group will affect groups called by it, as well as groups after it. The current context will be modified to add the changes made by a group, before returning to the previous group, or jumping to another one.

The `use` statement has the following syntax:

```
use (groupname)
```

Where *groupname* is the name of the group to jump to. After the new group has finished processing, control will return to the statement after the current one, in the same rule. You can nest quite a few groups; the exact number is not known.

### The return Statement

The `return` statement stops all processing of rules and returns control to the typist. No statements are executed after the `return` statement, even if it jumps back through multiple groups.

The `return` statement has no parameters; it must be on the right hand side of the rule.

### The match and nomatch Rules

Two special rules can be included in a group: the `nomatch` rule and the `match` rule. One of these two rules will be executed every time the group is entered, unless the rule matched contains a `return` statement.

The rules are represented in the following way:

```
nomatch > right-hand-side  
match > right-hand-side
```

Where the *right-hand-side* can include any of the statements legal to the right hand side of a rule, except for `context` and `index`. Both of these rules can jump to other groups with the `use` statement, output characters, or stop processing with the `return` statement.

If no rule is matched, the `nomatch` rule will be executed; if a rule is matched, the `match` rule will be executed, *after* the matched rule has been executed.

### Summing Up

Several of the example keyboards included with this program illustrate the usage of these statements and rules. To sum up:

```
LHS > RHS or use(group) or return  
match > RHS or use(group) or return  
nomatch > RHS or use(group) or return
```

## Constraints

Constraints are ordinary rules that restrict certain combinations from being typed. These rules can occur anywhere, even in a `nomatch` or `match` rule.

A constraint rule can just be something like the following line:

```
any(vowel) + any(vowel) > context
```

This would restrict two vowels from being typed in a row; the second key would just be ignored.

However, you might wish to let the typist know that they typed an illegal combination. This can be done with the `beep` statement. The `beep` statement simply makes a beep at the PC speaker.



---

**Note** If you have a sound driver installed, such as the PC Speaker sound driver, or a driver for a sound card, the `beep` statement plays the sound identified by the Asterisk entry in the Sounds option in Control Panel.

---

The `beep` statement is legal only on the right hand side of a rule; it just tells Keyman to make a beep, nothing else. A `beep` statement can be used both in `match` and `nomatch` rules. Examples of the `beep` command:

```
any(vowel) + any(vowel) > context beep
+ any(illegal) > beep
```

When you are restricting a set of keys, without context, from being typed, but you don't want a beep, another statement is required. The `nul` statement tells Keyman that nothing is on the right hand side of the rule. The rule above would become:

```
+ any(illegal) > nul
```

This would simply ignore any illegal keys. In some situations with multiple groups, this can be more useful than it appears. The `nul` statement is not necessary for the `nomatch` and `match` rules; just don't add them if you don't want them to do anything.

Typically, you would put constraints in the first group of a keyboard program, and every rule matched would simply be a rule testing for illegal context and key combinations. The `nomatch` rule would then be:

```
nomatch > use(maingroup)
```

Obviously, you could name the next group anything you like. The `match` rule would probably be left out.

## Groups Without the “using keys” Keyword

The `using keys` keyword, introduced in chapter 4, was used in a `group` statement to tell Keyman that the group would be needing information on the key pressed. In some situations, you might want ignore the keystrokes sent, such as for syllable splitting, or for changing the order of stacked diacritics, which only depends on the context.

## Virtual Keys

With what you have learnt so far, any letter, number, or punctuation mark can be identified as a key in a rule. However, you cannot test the `Ctrl` and `Alt` states of these keys; with some keyboards, it is necessary to do so.

Virtual keys allow you to do that. A virtual key keyword can identify almost any key on the keyboard; a few specialized ones are either reserved or unable to be used.

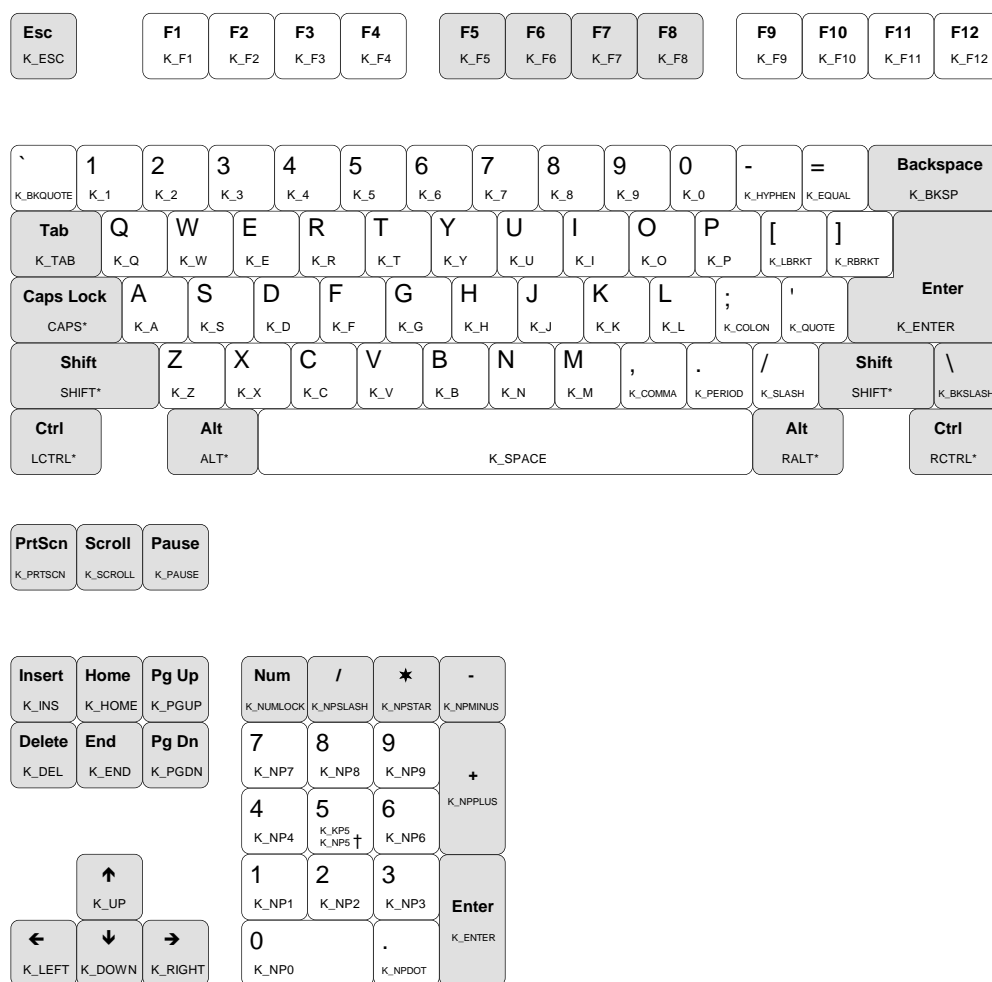
Virtual keys are allowed only in the key section of a rule, not in the context or the output. A virtual key is identified by an opening bracket character (‘[’). It ends at a closing bracket character (‘]’). Inside the brackets, you can have a combination of shift-key codes and the actual virtual key, which is identified by a “K\_” at the start of the keyword.

The keyboard shown further on gives the virtual keys for all keys on the standard US 101 key keyboard. (**Note:** The arrangement may not be identical to your keyboard.)

---

**Important** You must not use virtual keys in the output. Keyman will recognize them, but output will be garbled. This version of Keyman does **not** support virtual keys in the output.

---



**Figure 5.1: The Virtual keys keyboard layout**

\* Keys marked by a star are special keys that will be discussed in more detail further on.

† This key can be either K\_KP5 when NUM LOCK is off, or K\_NP5 when NUM LOCK is on; this applies to all the keys on the number pad - when NUM LOCK is off, the movement keys will be used.

For example, to test for the SCROLL LOCK key, you would have the following line:

```
+ [K_SCROLL] > output
```

If you want to test for a key that is used with SHIFT, CTRL, ALT, or CAPS LOCK, then you would proceed it with one of the following keywords:

<b>Shift key to test</b>	<b>Keyword</b>
SHIFT	SHIFT
Either CTRL	CTRL
Left CTRL	LCTRL
Right CTRL	RCTRL
Either ALT	ALT
Left ALT	LALT
Right ALT	RALT
CAPS LOCK on	CAPS
CAPS LOCK off	NCAPS

So, if you wanted to test for Right ALT + the letter “e”, you would have the following line:

```
+ [RALT K_E] > output
```

This version of Keyman does not let you use stores for virtual keys.

## Other Features

### Other Header Statements

There are three optional header statements that Keyman recognizes, all working with CAPS LOCK.

The first statement, `CAPS ALWAYS OFF`, makes sure that CAPS LOCK cannot be turned on while the keyboard is active, and it turns CAPS LOCK off when the keyboard is switched on. Put this statement on a single line in the header, as follows:

```
CAPS ALWAYS OFF
```

The other two statements, `CAPS ON ONLY`, and `SHIFT FREES CAPS` are usually used together. `CAPS ON ONLY` makes the CAPS LOCK key like a typewriter CAPS LOCK, where pressing it turns it on only.

`SHIFT FREES CAPS` tells Keyman to recognize SHIFT and turns capitals off. Using these two together makes Keyman work like many European keyboards. These two statements each take a single line in the header, as shown below:

```
SHIFT FREES CAPS
CAPS ON ONLY
```

### nul In the Context

The `nul` statement is used at the start of the context to tell Keyman only to match that rule if there are only as many characters output on the screen as in the context. This statement is not very likely to be used; there is a possibility you may use it for testing after a keyboard has been turned on, or to change a character into the best possible output without knowing what is before it. For example,

```
nul + 'a' > 'A'           c Not very useful!
```

### The outs Statement

The `outs` statement places the content of a store into the string at its position. You would probably only use the `outs` statement for creating large stores. Usage:

```
outs(storename)
```

### Long Rules

When you are making your keyboard, you may find that some lines are very long and are hard to read if made shorter. Keyman has a way of getting around this: by putting a backslash (\) on the *very end* of the line, Keyman is told that the line should be joined with the next one. You can do this for multiple lines if necessary, up to 1K (about 1000 characters) long. The backslash must come *after* comments if you have them. For example,

```
any(LowerCaseVowel) + any(UpperCaseVowel) > \  
    index(UpperCaseVowel,1)           c From previous line \  
    index(LowerCaseVowel,2)           c From previous line
```

### Keyman and Control Panel's International Settings

Keyman only interferes with one part of the International settings in Control Panel, the keyboard layout. Keyman requires that you use the US keyboard layout for it to work properly in most cases; you could create a keyboard for other languages you wanted to use instead of using the default Windows one.

However, if you must use a different keyboard layout, Keyman 3.2 lets you get around it when you use virtual keys. Virtual keys work properly with other keyboard layouts, except that you may not be able to use right ALT or CAPS LOCK with some keyboards.

## CHAPTER 6

# TIKE

TIKE is the keyboard program editor supplied with Keyman. It stands for Tavultesoft Integrated Keyboard Editor. With TIKE, design and testing of your keyboards is simplified because TIKE works closely with Keyman to load your keyboards, report error messages, and test them. You can edit multiple files and switch between them instantly, using the standard Windows Multiple Document Interface.

This chapter will discuss the various parts of TIKE, and explain how to use them.

## Overview on Using TIKE

When you start TIKE, a startup dialog will briefly appear, as the TIKE window is opened. After the startup dialog closes, you will see a screen that looks like figure 6.1.



**Figure 6.1: The TIKE screen (blank space has been removed)**

You will notice that there is a status bar (at the bottom of the screen), a menu bar, and a toolbar. If you have use Microsoft Word for Windows, these parts of the screen will look familiar.

### The Toolbar

The toolbar is accessible only with the mouse, but each button on the toolbar is also available in the menu, although the menu will only show the options applicable to the current situation. When you click and hold on a toolbar button, the status bar will tell you what the button does.

### The Status Bar

The status bar gives you information on the font, current position, menu items, and toolbar buttons. When you are editing a file, the status bar will update frequently with the current position and font.

## The Menu

The menu bar will have different items and sub-menus at different times, to hide items that are inappropriate for the situation.

## Creating, Loading, Saving, and Editing Files

You can start a new file either by clicking on the new file button on the toolbar (a white sheet of paper), or by selecting **File•New** from the menus. A new window will open, and you can type and edit in the same way as Notepad.

To open an existing file, either click on the opened folder button or select **File•Open...** from the menu. An Open dialog box will appear which works the same way as the Open dialog box from Notepad (but looks a whole lot nicer!)

To save the file you are editing, click the disk button, or select **File•Save** or **File•Save As....** These options also work in the same way as Notepad.

## Selecting an Editing Font

TIKE lets you select an separate editing font for each file you are working on. Click on the “ABC” toolbar button, or select **Options•Font....** The font dialog box that appears is another standard one. Select the font, size, and style you wish to use, and click Ok.

## Searching and Replacing Text

The **Search** menu allows you to search for text, and replace it if you wish with new text. The editing font you have selected will be used in the dialog boxes; other than that, the dialog boxes are standard.

## Testing Your Keyboard

Before you can test your keyboard, you must save it. When you click the testing button (a keyboard with a “T”) on the toolbar, or select **Options•Test**, TIKE asks Keyman to load the keyboard into memory. If any errors occur, TIKE moves to the appropriate line in the file and displays the error message in the status bar. If the keyboard loads, a new window will open: the testing window. The testing window simply lets you test the keyboard you loaded by typing using that keyboard. TIKE will automatically switch keyboards when you switch windows. You can select any font in the testing window the same as in the editing windows.

When you have finished testing your keyboard, click the unload button (a keyboard with an arrow), or select **Options•Unload Test Keyboard**.

## Character Map













You will often want to use Character Map to find character codes in the fonts you are working with; the last button on the toolbar loads Character Map, as does the Options•Run Character Map menu item.

## Printing

TIKE lets you print your keyboards using the currently selected font; simply click on the printer button, or select File•Print.... You can set the margins, header and footer using the Page Setup dialog box. By default, the paper will have one inch margins and the name of the keyboard will printed at the top of each page. The header and footer will print with the selected font in bold and italic.

## Reference

A summary of the menu options, associated toolbar buttons, and what they do is provided below:

	File•New	Create a new file
	File•Open...	Open an existing file
	File•Save	Save the current file
	File•Save As...	Save the current file to a new name
	File•Print...	Print the current file
	File•Page Setup...	Set margins, header and footer for printing
	File•Exit	Exit TIKE
	Edit•Undo	Undo the previous action
	Edit•Cut	Move selected text to clipboard
	Edit•Copy	Copy selected text to clipboard
	Edit•Paste	Insert clipboard contents at the insertion point
	Search•Find...	Find specified text
	Search•Replace...	Find specified text and replace with new text
	Options•Font...	Change the editing font for the current file
	Options•Run Character Map	Load the Character Map accessory
	Options•Test	Test the current keyboard file
	Options•Unload Test Keyboard	Unload the current keyboard being tested
	Window•Tile	Tile the open windows
	Window•Cascade	Arrange the windows in cascading format
	Help•About	Provide information on the TIKE version





---

## A P P E N D I X 1

# The KEYMAN.INI File

Keyman stores all its options in KEYMAN.INI, in the Windows directory. The KEYMAN.INI file is a standard Windows INI file, divided into sections with options in each section.

There can be four sections in KEYMAN.INI: [Options], [Advanced], [Extensions], and [TIKE]. The [Options] section contains options about the position of the keyboard buttons, hotkeys, and similar settings. The [Advanced] section contains settings about rules and the sizes of buffers used with the rules. The [Extensions] section has the names of the Extension handlers (documented fully in Technical Documentation) installed in Keyman. [TIKE] contains settings about TIKE.

### The [Options] Section

The [Options] section has the following settings:

<b>Setting</b>	<b>Default</b>	<b>Description</b>
DisableHotKey	Ctrl+Shift+K_W	Turns all keyboards off
SwitchHotKey	Alt+K_EQUAL	Toggles the keyboard state
MenuHotKey	Ctrl+Alt+K_EQUAL	Brings up the Keyman menu
IconWindow	Off	Whether the Keyman floating button window is on or off
IconPosition	Left	The position of the Keyman buttons on the title bar
IconWindowPos	TitleBar	The position of the Keyman floating button window, if on
IconWindowGrab	Off	True/On if the Grab-Tag Mover option is set
IconWindowLocation		The position of the icon window on the screen, in pixels
FloatingCompatMode	Off	Use Windows-NT compatibility mode

## The [Advanced] Section

The [Advanced] section controls the size of Keyman's internal rule buffers. It has the following settings:

Setting	Default	Description
MsgStackSize	80	The maximum number of messages that Keyman can output in a single rule; keep in mind that one backspace takes two messages, and several messages are required for internal use.
KeyContextSize	16	The maximum length of the context in a rule, in bytes.
KeyOutputSize	16	The maximum length of the output in a rule, in bytes.
MaxKeyboards	8	The maximum number of keyboards that can be in memory at once.
ContextStackSize	64	The length of the output that Keyman will remember; when it gets longer than this, Keyman shifts the first characters off the left. You should never need to change this.
MaxKeyCombos	1024	The maximum number of rules in a group. The default maximum can be increased if you reduce KeyContextSize or KeyOutputSize; the maximum number of rules in a group can be calculated with this formula:

$$\text{MaxKeyCombos} = \frac{65535}{(\text{KeyContextSize} + \text{KeyOutputSize} + 5)}$$

If you change the KeyOutputSize and KeyContextSize and thereby exceed this maximum, Keyman will display a warning and tell you the appropriate value for MaxKeyCombos.

Each rule with an `any` statement in the key will be expanded to multiple rules when loaded - one rule for each character in the store.

## The [Extensions] Section

The [Extensions] section consists of lines that each add a Keyman File Extension Handler to Keyman. A Keyman File Extension Handler is a library that loads a file into Keyman format in memory. The default Extension Handler supplied with Keyman loads "KMN" files. Other Extension Handlers would load other formats of files, for example, a compiled "KMN" file.

Each line looks like this:

```
ext=filename.dll,description
```

Where *ext* is the three character extension of the file names that the Extension Handler will process; *filename.dll* is the name of the library that contains the Extension Handler, and *description* is a description of the files it handles that appears in the Load dialog box. The default Extension Handler is added to this list internally; it does not need to be in the INI file; however, as an example, a line for it would look like this:

```
KMN=KMDFILE.DLL,Keyman Files (*.kmn)
```

## The [TIKE] Section

The [TIKE] section contains settings about the position of the TIKE window. You do not need to change any of the settings, which are shown below:

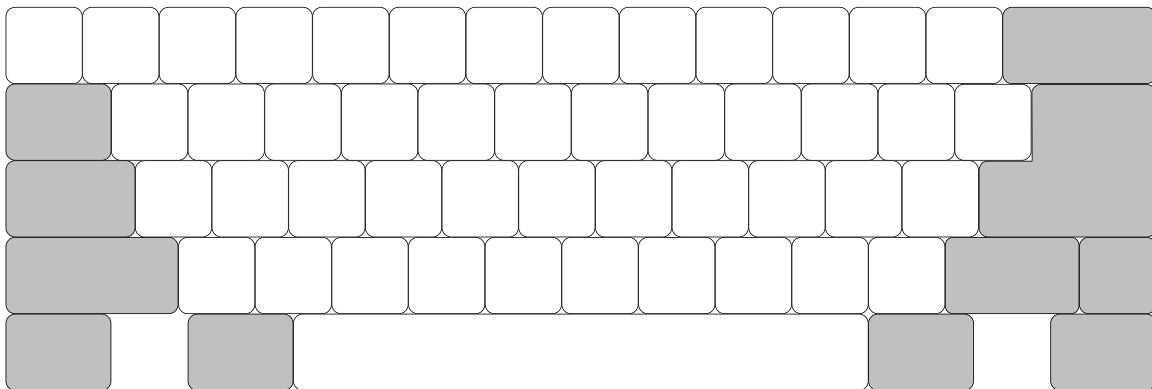
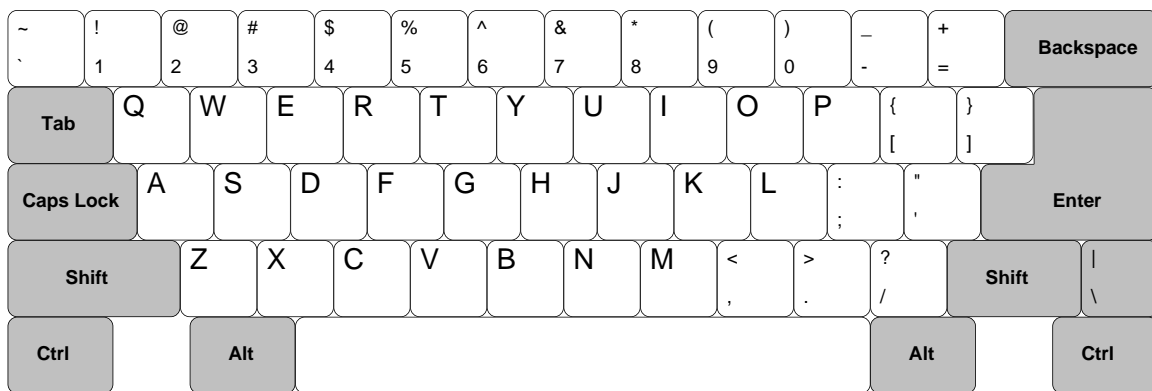
<b>Setting</b>	<b>Description</b>
WindowPos	The position of the restored TIKE window on the screen
PointPos	The positions of the icon, and the top left corner of the maximized window
ShowCmd	The mode to show the window in.



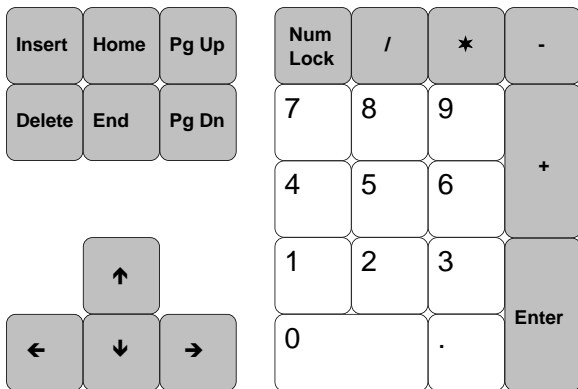
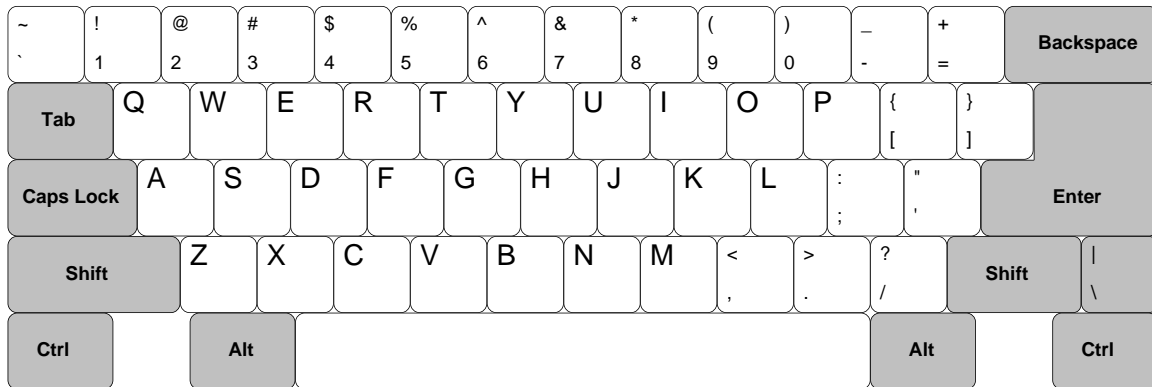
## APPENDIX 2

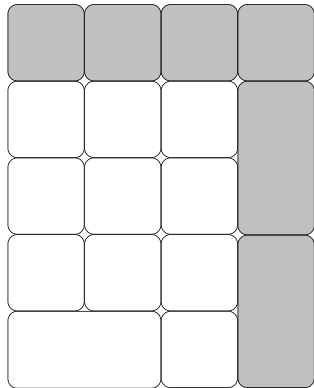
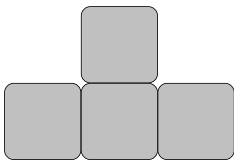
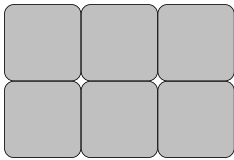
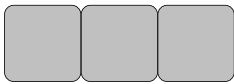
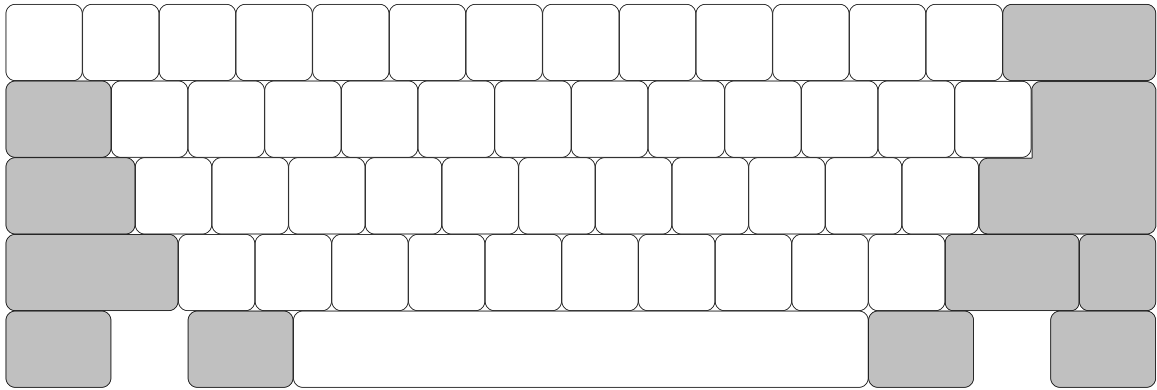
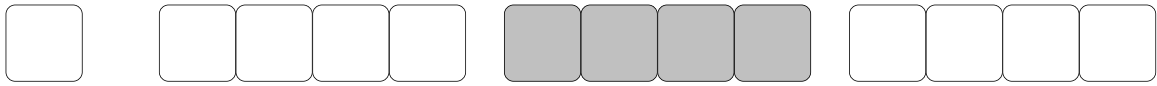
# Some Keyboard Templates

Some blank keyboard templates are provided on the following pages which you can photocopy, fill in, and provide with your keyboards. These templates are also included with Keyman in the file KBDS.DOC, in Word for Windows 2.0 format. You can copy this document and add your own characters to the document.



## 46 Tavultesoft Keyboard Manager









# I N D E X

## —[—

[Advanced], 42  
 [Extensions], 42  
 [Options], 41  
 [TIKE], 43

## —A—

About button, 9  
 ANSI text format, 20  
 Any statement, 26  
 ASCII text format, 20

## —B—

beep statement, 32  
 begin Statement, 23  
 BITMAPS Statement, 22  
 Blank Lines, 21

## —C—

CAPS ALWAYS OFF, 35  
 caps lock, 35  
 CAPS ON ONLY, 35  
 CC tables, 19  
 Character Map, 19, 39  
 Comments, 21  
 Constraints, 32  
 context, 25  
   defined, 24  
 context statement, 27  
 Control Panel, 36

## —D—

descriptive name, 22

## —E—

endstore, 26  
 English  
   typing in, 8  
 Exit button, 10

## —F—

Floating window, 12

## —G—

Groups, 24  
   multiple, 31

## —H—

Hardware requirements, 5  
 Header, the, 21  
 Help button, 10  
 Hide button, 9  
 hotkey  
   keyboard on, 22  
 HOTKEY Statement, 22  
 Hotkeys  
   Changing, 13

## —I—

Index statement, 26  
 International dialog in Control Panel, 36

## —K—

key  
   defined, 24  
 keyboard button, 22  
 keyboard buttons, 7, 11  
   anywhere on the screen, 12  
   in floating window, 12  
   locking in place, 13  
   On title bar, 11  
 Keyboard Hotkeys, 14  
 keyboard program, 19  
 Keyboards  
   Loading, 10  
   testing, 28, 38  
   Unloading, 10  
 Keyman Menu, 9  
 Keyman startup list, 15  
 Keyman Window, 9  
 KEYMAN.INI, 41

## —L—

Languages  
   Typing in other, 7  
 Load button, 10  
 Loading Keyboards, 10  
 Long Rules, 36

## —M—

Macros, 16  
 match Rule, 32  
 multi-line rules, 36  
 Multiple Groups, 31

## —N—

NAME Statement, 22  
 nomatch Rule, 32

nul statement, 33  
in the context, 36

### —O—

Options button, 10  
Options Dialog Box, 10  
output  
defined, 24  
outs Statement, 36

### —P—

Paintbrush, 22

### —R—

**Representation of characters**, 25  
Requirements, hardware and system, 5  
return Statement, 32  
Right Aligned radio button, 11  
rule  
defined, 24

### —S—

Set Hotkeys dialog box, 13  
Setup program, 5  
SETUP.EXE, 5  
SHIFT FREES CAPS, 35  
Starting Keyman, 6  
Starting Keyman Automatically, 15  
Store statement, 26

### —T—

Testing the Keyboard, 28  
text editor, 20  
TIKE, 37  
Creating, Loading, Saving, and Editing Files, 38  
Editing Font, 38  
Menu, 38  
Printing, 39  
Searching and Replacing Text, 38  
Status Bar, 37  
Toolbar, 37  
Title bar  
Location of keyboard buttons, 11  
Typing  
in English, 8  
In other languages, 7

### —U—

Unload button, 10  
Unloading Keyboards, 10  
use Statement, 31  
using keys, 24, 33

### —V—

version 3.2, 23  
VERSION Statement, 23  
Virtual Keys, 33