Version 2

MOUNTFOCUS INFORMATION SYSTEMS LTD.

Keyboard Designer

User's Guide

MOUNTFOCUS KEYBOARD DESIGNER

User's Guide

Section

1

MOUNTFOCUS INFORMATION SYSTEMS LTD.

Keyboard Designer User's Guide

Introduction

Chapter 1

Introduction

Thank you for choosing the MountFocus Keyboard Designer for your virtual keyboard needs.

This manual explains the use and operation of the Keyboard Designer as well as the Runtime Keyboard. The manual is divided into five sections; Introduction, Tutorials, Tool Window Reference; Object Reference and finally Runtime Reference. Due to the large amount of new features and changes to this version, we recommend that both new and old users read the first two sections in full. This will give you a good overview of the features available as well as a good understanding of how the Keyboard Designer and the Runtime Keyboard programs work.

What is The MountFocus Keyboard Designer?

The Keyboard Designer version 2.0 is a powerful and flexible program used to design onscreen virtual keyboards. The uses for a virtual keyboard are almost endless: Point of Sale applications, Info-kiosks, remote customer terminals, ATM machines, personal computers, computers for disabled users, click-able onscreen shortcuts for your favorite programs, etc.

The Keyboard Designer v. 2.0 has been redesigned to allow even more flexibility in designing your keyboards. The only limit to what your keyboard will look like or how it will function is your imagination. The user interface is easier to work with and many enhancements have been made to the functionality of the Runtime Keyboard.

Who Can Use the Keyboard Designer?

Keyboard Designer can be used by any level of computer user: from the inexperienced to very advanced. The interface has been completely redesigned to make using the Keyboard Designer, even easier than before.

The Keyboard Designer is perfectly suited to the needs of software programmers; disabled users and touch screen users and touch screen software programmers.

Programmers will appreciate the flexibility and adaptability of the Keyboards they design, since any keyboard is suitable for normal or touch screen environments, without re-writing any code. Developers who specialize in touch screen environments will notice much improved focus control.

Version 2.0 of the Keyboard Designer allows disabled users even greater control over the layout of their keyboard. Standard keyboards can be modified to meet the specific needs of every end user, and more flexibility in the Runtime Keyboard means it is even easier to use on screen.

Keyboards can be created to replace frequently used keystrokes, even for the average PC user. Closings such as name, web site address, etc (commonly known as sig. files) can be added to any letter or email with the click of a button.

We hope you enjoy using the Keyboard Designer and the Keyboards you make

Section

2

MOUNTFOCUS INFORMATION SYSTEMS LTD.

Keyboard Designer User's Guide

Tutorials

Chapter 2

Tutorial 1 - Overview

This tutorial will focus on familiarizing yourself with the MountFocus Keyboard Designer Interface. Estimated time for completion of this section: 17 minutes.

The following items will be discussed:

- Objects
- Property Editor
- Keystroke Editor
- Indicator control
- Object List
- Key Pool
- Selecting Multiple Objects

By the end of this tutorial, you will be able to place objects on the Keyboard Page, edit properties of objects, use the Keypool and have an overview of the Keystroke Editor.

To begin, start the MountFocus Keyboard Designer. Below is a view of what the Keyboard Designer looks like when the program is opened.

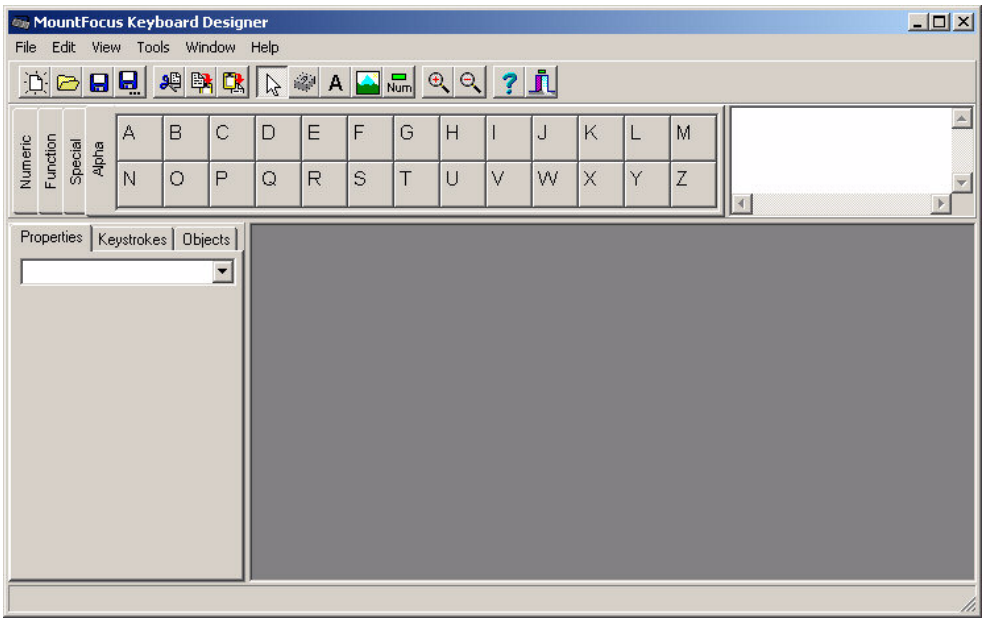

FIGURE 1 Keyboard Designer main screen upon opening.

Keyboard

Select **File ▸ New** from the command menu. This will open a new keyboard file with one Keyboard Page (called "Page1" by default). Each keyboard file can contain one or more pages.

The properties of each object can be set using the Property Editor. When a file is opened, the default object displayed in the Property Editor is the first page in the keyboard file. Click on the drop down list next to "Page1." You will see the words "Keyboard" and "Page1" on the list. If you click on the word "Keyboard," the Property Editor will change to reflect the properties associated with the Keyboard. You can select a Sound file to be played every time a Key is pressed, and change the size of the Runtime Keyboard.

> **Note**
>
> There is only one Keyboard object, but there can be many Pages, Keys, etc., and you can only edit one object at a time.

Keyboard Pages

Zooming will make editing your Keyboard Page simpler.

Click anywhere on the Keyboard Page to show the properties for "Page1" again. The default grid height and width is 5 x 5 pixels. You can select to hide or show the Grid from the View menu, or set "snap to grid" on or off. The grid is helpful in placing keys on the Keyboard Page. You can experiment with these settings to see

which you prefer. For the purpose of these tutorials, set snap to grid "On" and select **Show grid** from the view menu.

Right click anywhere on the Keyboard Page to see the popup menu. You can change the background color of the Keyboard Page, create a new region, create a new page or delete the current page.

Keys

Next, select **Tools ▸ Key Tool** from the menu, and move the mouse pointer to the Keyboard Page. The pointer will change to the Key Tool pointer. Each tool has a different style of pointer, which we will see later. Click anywhere on the Keyboard Page to place a Key. Every time you click on the Keyboard Page, a new Key will be placed, until you choose the select tool by right clicking on any blank area of the Keyboard Page, or by choosing **Tools ▸ Select Tool** from the menu.

Note

When the Key Tool, Label Tool, Image Tool or Indicator Tool is selected, right clicking on the Keyboard Page will automatically select the Select Tool. When the Select Tool is selected, right clicking on the Keyboard Page will open a popup menu.

Click on the Key you just placed on the Keyboard Page to view the properties for this object in the Property Editor. From here you can add a background image, change the position and size of the Key, etc. You can also re-size Keys by clicking and dragging on any of the handles. Keys can also be copied from the Keypool by clicking on the Key you would like to copy and then clicking on the Keyboard Page.

If the property "Close Keyboard" is set to True, the Runtime Keyboard will first execute any keystrokes and then shut down when that Key is pressed. If "Close Page" is set to True, any keystrokes will be executed, the current page will close and the page that was active before this one will open (the Runtime Keyboard automatically keeps track of which pages were opened and in which order).

Keys can have Label objects, Image objects and Indicator objects placed on them, but not other Keys. To place objects on Keys, the "Key edit mode" is used. Hold <*Shift*> while clicking on the Key and the appearance of the Key changes to indicate this mode. To exit this mode, click on any blank area of the Keyboard Page, or on another object.

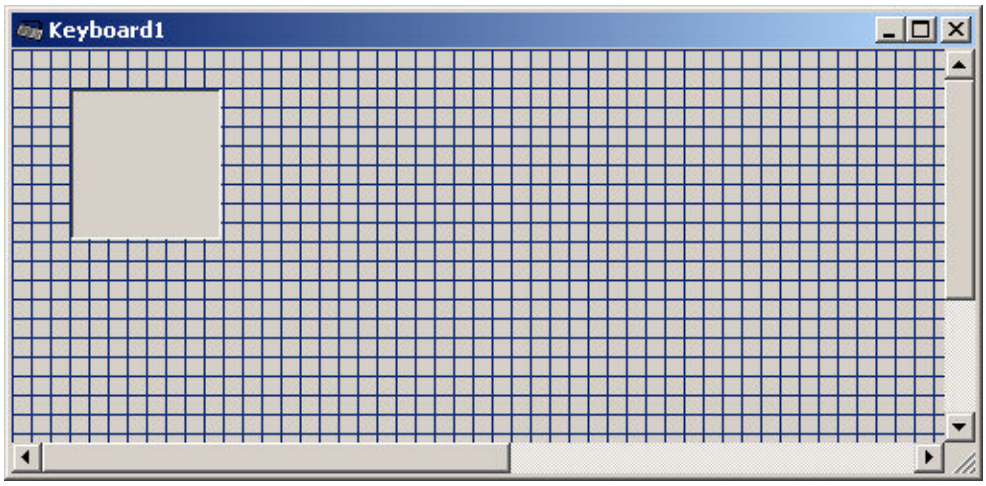

FIGURE 2 Key edit mode.

Labels

Label objects can be placed on Keys, Indicators, or on the Keyboard Page. Placement of Labels is very similar to placing Keys. Select **Tools ▸ Label Tool** from the menu, and the pointer will change to the Label tool. Click on the Keyboard Page or object and a label will be placed at that point. Click on the Label again, and you can view the properties for the Label in the Property Editor.

Right click on the Label object to see the pop-up menu. Select **Edit text** from the menu to open a small text editor. The text editor makes it easier to enter multiple lines of text.

Images

Image objects can be placed on Keys, Indicators, or on the Keyboard Page. Select **Tools ▸ Image Tool** from the menu, and the pointer will change to the Image Tool. Click on the Keyboard Page or object to place an Image object at that point. Click on the Image again to view the properties for the Image in the Property Editor.

Setting the AutoSize property to "True" will not allow the Image object to be resized; it will always be the size of the bitmap. Setting the Transparent property to "True" allows the background color of the Keyboard Page to show through the Image. Details on this can be found in the section on Understanding Images.

Indicators

Indicator objects can be placed on Keys or on the Keyboard Page. Select **Tools ▸ Indicator Tool** from the menu and click on the Keyboard Page. The Indicator has a separate caption for the "Off" and "On" states. The "Off" state is a frame drawn with black interior lines, and the "On" state is a frame drawn with

white interior lines. NOTE: Once you place another object on the "Off" or "On" caption, you will no longer see the interior lines.

You can switch between the two states by clicking on the Indicator to select it and then right clicking on the Indicator and choosing **Show off caption** or **Show on caption** from the pop-up menu.

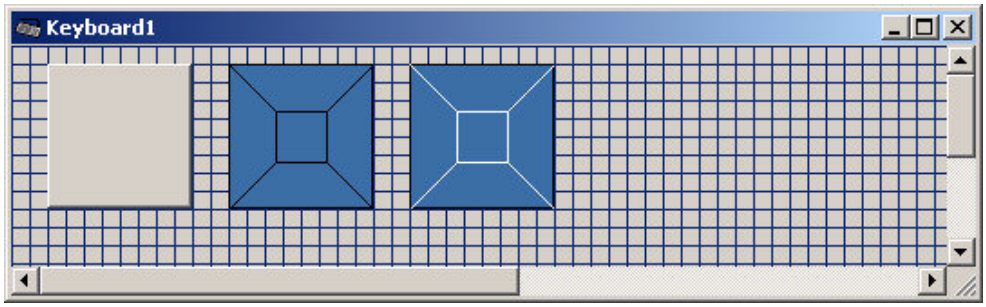

FIGURE 3 Indicators in off and on states.

To place objects on Indicators, the "Indicator edit mode" is used. Hold <*Shift*> while clicking on the Indicator to change to this mode. To exit this mode, click on any blank area of the Keyboard Page, or on another object.

Set the Transparent property of the Indicator to "False" and choose a color for the indicator to set it apart from the rest of the Keyboard Page during editing for better visual control. Once you have designed the Indicator, you can return these properties to their defaults.

The properties for the Indicator allow you to define what the Indicator will monitor (i.e., Caps Lock, Num Lock, Scroll Lock, or other user defined constants).

Regions

You can define which areas of the Keyboard Page should be visible by using Regions. If no Regions are defined, the whole Page is visible, but if one or more Regions exist, only the area defined by the region(s) will be visible in the Runtime Keyboard.

When **Send to back** is chosen, you can still move the Region around and resize, but it is easier to position, because it will not cover the other objects on the Keyboard Page. It is usually easier to define the Regions after the Keyboard Page has been designed.

Select **Tools ▸ New region** from the menu and a region will appear in the upper left corner of the Keyboard Page. Select the Region by clicking on it. You can now drag the Region anywhere on the Keyboard Page, and change the Height or Width. Right click on the Region to view the pop-up menu. You can select **Bring to front** or **Send to back** from this menu.

Note

Objects cannot be added on top of Regions while the Regions are visible. Regions can be set to visible or invisible by selecting **View ▸ Show regions** from the menu.

Keystroke Editor

The Keystroke Editor is activated when a Key is selected. Click Keystrokes tab and then on the Key you created earlier in this tutorial. You will see the word "Keystrokes" appear in the Keystroke Editor. Below the Keystrokes Editor is the Virtual Key List. To create keystrokes, select a virtual key name from the list and drag it to the Keystroke Editor. Keystrokes must be dropped on an existing element. The Keystroke Editor shows keystrokes in "tree view" style and shows the order in which key presses will be executed.

For this tutorial, select "A," and drag it to the Keystroke Editor. When you add a virtual key code to the list, the Key Preview area is activated. This area shows the exact key messages that will be sent.

Keystrokes can be deleted by highlighting the keystroke you wish to delete and pressing "Delete" on the keyboard. This will also delete any "child" keystrokes.

Indicator Control

The Indicator Control window is used to simulate indicators in the Designer so you can test their functionality. You can test the pre-defined Indicators for Caps Lock, Num Lock and Scroll Lock by setting the Interval spin edit to a non-zero value, and using the Keys on the physical keyboard. (This will be illustrated in tutorial 3).

For user defined Indicators, you can select the Indicator number with the Indicator # spin edit and change the state from "Off" to "On" by clicking on the State checkbox.

Object List

The Objects window allows you to see a list of objects that have been placed on your Keyboard Page. When an object is selected, the tool window has an alphabetical list of all objects associated with the selected object, including their position and size. If your Keyboard Page is selected, all objects on the Keyboard Page will be listed. If a Key is selected, all objects placed on the Key will be listed.

You can select multiple objects from the list, select single objects using the tool bar, and give the commands bring to front and send to back using the tool bar in the Object List tool window.

Keypool

The Keypool that comes with the MountFocus Keyboard Designer is a set of standard Keys, Labels and Indicators that can be copied to any Keyboard Page.

Objects can only be copied one at a time. Any Keyboard file can be saved as a Keypool, (and any Keyboard file can be opened in the Keypool tool window). Select **File ▸ Save As** to select file type Keypool (extension .kbp), and enter a name for your Keypool file. Keypool files can be edited by selecting **File ▸ Open** from the menu and selecting "Keypool files" in the "Files of type" drop down list. To change the Keypool that is displayed in the Keypool tool window; select **File ▸ Select Keypool** and choose a Keypool (or keyboard) from the list.

Selecting Multiple Objects

Multiple objects can be selected in two ways. You can click on any blank area of the Keyboard Page and drag the mouse to draw a frame. All objects that are completely within the frame drawn by your mouse will be selected. You can also hold <*Ctrl*> and click on the objects you want to select.

When using the Align command, first select the object you want to use as the standard, then use <*Ctrl*> click to choose the other objects you want to align.

After you have selected several objects, you can right click to bring up a pop-up menu that allows you to align the objects, bring them to front, send them to back or delete the selected objects.

Multiple objects can be moved around the Keyboard Page by clicking and dragging.

In tutorial 2, we will build a basic keyboard.

Chapter 3

Tutorial 2 – Basic Keyboard

This tutorial will focus on building a basic keyboard. Estimated time for completion of this section: 20 minutes.

The following tasks will be completed.

- Copying Keys from the Keypool
- Creating Keys
  - Adding Label objects
  - Changing Key Properties
  - Adding Keystrokes to Keys
  - Adding Image objects

Start the MountFocus Keyboard Designer and select **File ▸ New** from the menu.

Select **File ▸ Save** from the menu and type `Tutorial` <*Enter*> as the name for this keyboard.

For the basis of this keyboard, lets start by copying some Keys from the Alpha Page on the Keypool.

Click on the Alpha tab at the top of the screen and click on the "Q" to copy that Key. Place the cursor in the upper left corner of the grid and click again. You should now have a Key with the letter "Q" on it.

Click on the Key you just placed, to select it for editing and change the Left position to 0, and the Top position to 0. This will be our guide for placing the rest of the keys on the keyboard.

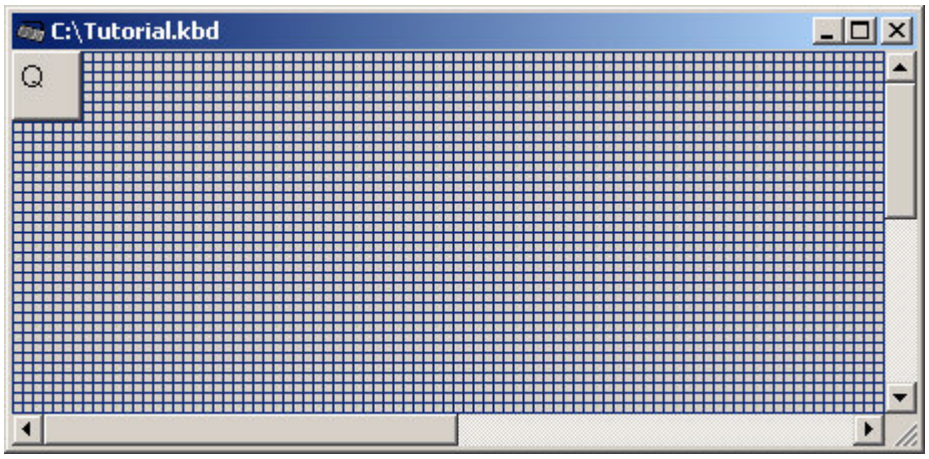

FIGURE 4 Placement of "Q" key.

Continue copying keys from the Keypool until you have a basic QWERTY keyboard like the one shown below.

Note

You will need to copy the SHIFT key from the Function Keypool and change the size to 35 x 35.

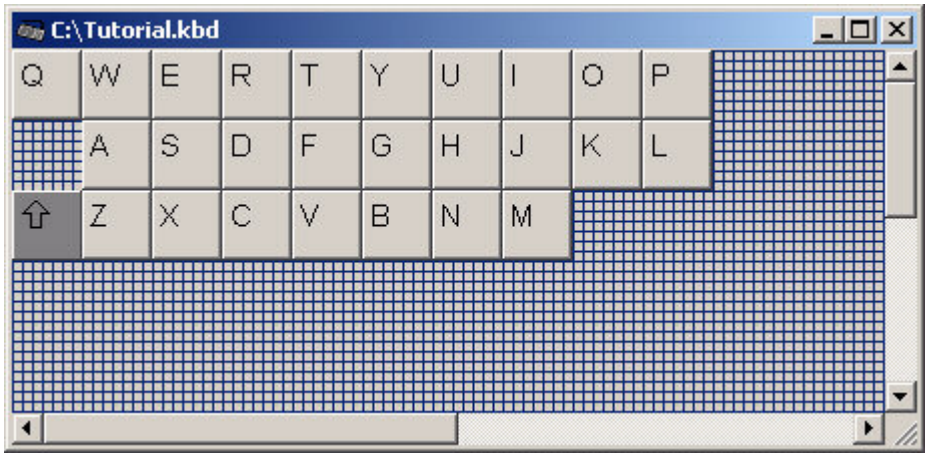

FIGURE 5 Basic QWERTY layout.

Now let's add a Space Bar to the keyboard.

Select **Tools ▸ Key Tool** from the menu and click in the area below the "Z" Key. All Keys are created with the default size set in the options window.

Click on the Key to select it. Make sure the Key is located at Left 50 and Top 105. If it's not, change the location in the Property Editor.

Change the Width of the Key to 215. Using the Up Arrow Key, move up to the Color property and choose "BtnShadow" from the drop down list. The color of

the Key should now be the same as the Shift Key that you copied from the Keypool.

Now, hold <*Shift*> and click on the new <*Space*> Key you just placed on the keyboard. Select **Tools ▸ Label tool** from the menu and click on the Key where you would like to place the upper left corner of the label. Click on the word "Label" that appears on the Key. Now you will be able to edit the label. Click on the "Text" field in the Property Editor and type `SPACE` <*Enter*>.

You can adjust the font, position of the label, height and width.

The text you typed now appears on the Key. Let's change the font to **Bold**, by clicking on the ellipsis button in the Font field and choosing **Bold** from the styles list.

Now let's make this Key send <*Space*> to the current application. Click on any blank space on the Keyboard Page to exit the edit mode and click on the Key again to select it.

If you don't want to scroll down the list of virtual keys, you can click on the key tool icon under the Virtual Key List and press the Space bar on your keyboard. This will reveal the virtual key code for Space and you can drag the code using the **right mouse button** dragging it to the Keystroke Editor as before. The code will be converted to the name of the key when it is dropped.

Click on the Keystrokes tab and scroll down the list of available keys in the Virtual Key List. Choose *Space* and drag it to the Keystroke Editor until the word Keystrokes is highlighted and release the mouse. The Key will now send <*Space*> to the target application once your keyboard is running. Notice that the Key Preview area now shows the key down and key up message for *Space*.

You have now defined a Key with the text "SPACE" that will send the keystroke <*Space*> to any application that has focus.

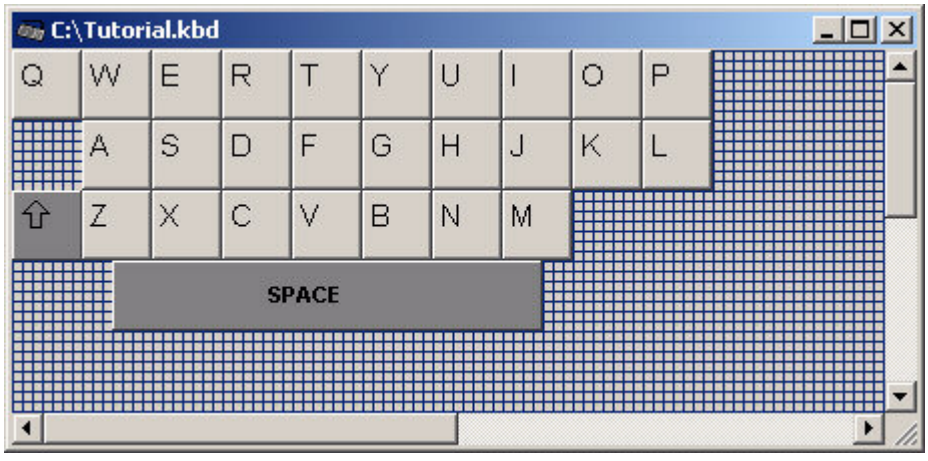

FIGURE 6 Basic QWERTY layout with Spacebar.

Click on any blank area of the Keyboard Page to exit the Key edit mode.

The next Key we will add is a "Caps Lock" Key. Create a Key to the left of the "A" Key and place a label on the Key. Since the text "Caps Lock" will not fit on one line on the Key, we will make the caption two lines. The vertical bar character "|" is used to signify <*Return*> for multi-line captions. Type: `Caps|Lock` in the Text field of the Property Editor.

You can adjust the placement of the Label until you are happy with the results.

Next, click on the Keyboard Page and again on the Key to select it. Click on the Keystrokes tab and choose *Capital* from the virtual key list and drag it to the Keystroke editor. This Key will now control Caps Lock "On" or "Off."

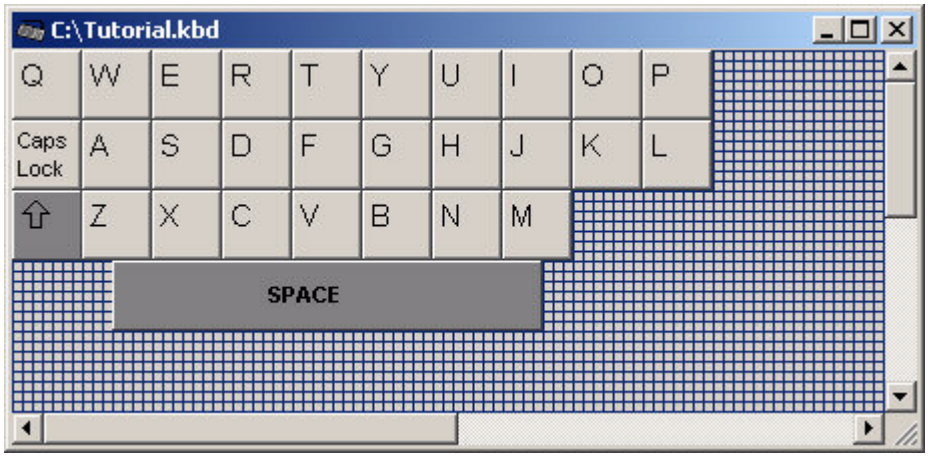

FIGURE 7 Basic QWERTY layout with Caps Lock key.

Now let's give a name to the page we are working on. Select Page1 from the drop down list at the top of the property editor and type `Alpha` in the Page Name field as the name for this page. Select **File ▸ Save** to save the changes you have made.

Let's add a new Key to the right of the "M" Key. For this Key, we will follow the same procedure as adding the Space bar, but we won't change the look of the Key. Hold <*Shift*> and click on the Key to add a Label to the Key. This Key will send the keystroke for the "left angle quote."

Note

The left angle quote is a special character, since it does not appear on the standard keyboard. Special characters can be added by holding <*Alt*> and using the Numpad keys.

Once you have added the Label, change the text field to "«" by typing <*Alt*> + `0171` (using the Numpad keys). Set the Left position to of the Label to 4, and the top position to 2. Also, change the font to Arial, 12 to make it appear larger on the Key to match the other keys.

Now it's time to add the keystrokes. Click on the Keyboard Page and then again on the Key to edit. Choose *Menu* from the virtual key list and drag it to the Keystroke editor (*Menu* is the virtual key name for *Alt*). Next choose NUMPAD0 from the virtual key list and drag it to the word *Menu* and release. You should see the NUMPAD0 press and release messages appear ***between*** the *Menu* press and release messages.

Now drag NUMPAD1, NUMPAD7, and NUMPAD1 to the word *Menu* in the Keystroke editor. Check the screen shot right to see what your windows should look like.

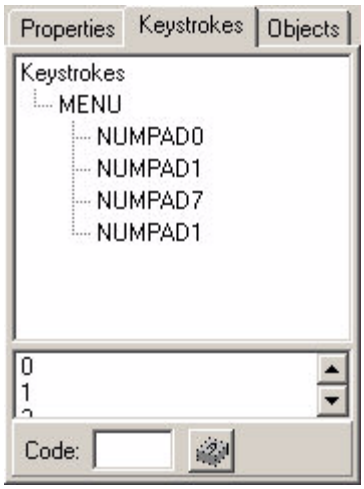

FIGURE 8 Keystroke editor for left angled quote.

Add another Key to the right of the one you just made and we'll make this Key send the keystrokes for the "right angle quote." Follow the same procedure as above for placing the label, positioning and font, but this time the Key codes are <*Alt*> + 0187.

Let's make an <*Enter*> Key next to the "L" and "right angle quote" Key. Create a new Key with a Left position of 350 and a Top position of 35. Change the Height to 70 and the Width to 40.

Hold <*Shift*> and click on the new Key and select **Tools ▸ Image Tool**. Click on the Key to place the Image and then click on the Image frame that appears. Click on the Image field in the Property Editor, and again on the ellipsis button that appears. Select the file "enter.bmp" from the file list, and click OK. Change the Top position of the image to 5, and the Left to 0. Change the Transparent field to False and see what happens, and change it back to True. Set the Auto Size field to True.

A word about backgrounds: light colors with dark captions tend to look best. If you want, you can really get creative with this, but remember that your end user should be able to look at a Key and immediately understand it's function – not admire its great look.

Also, you can choose a background color or a background image for your Key. If you choose to use a background image, the image will be stretched (shrunk or enlarged) to fit your Key. The background color of your Image will be invisible if "Transparent" is set to true. When "Transparent" is set to false, the color present in the lower left corner of your Image will be treated as the transparent color of the background bitmap, allowing the background color to show through.

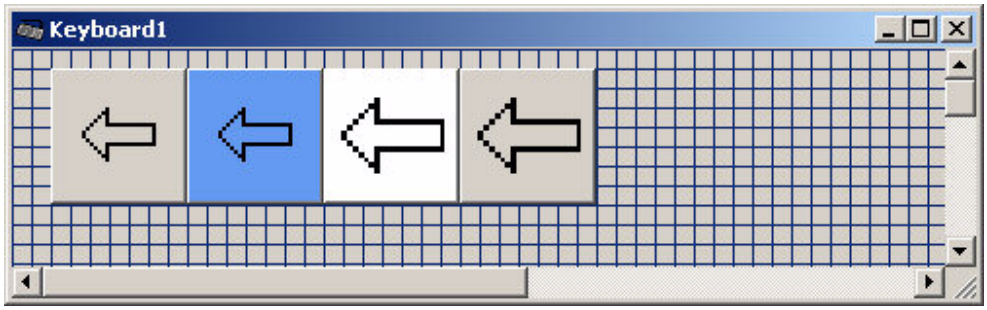

FIGURE 9 Images used as captions and as background images.

The screen shot above shows the difference between using a bitmap file as a caption (Image object) as in the first two keys on the left, and using a bitmap file as a background image (the third and fourth keys). The first Key shows how the default color of the Key shows through the Image object placed on the Key with Transparent set to True. The second Key shows the Key background color of the key changed to blue with Transparent set to "True". The third Key shows the backspace bitmap used as a background image with the "Transparent" option set to "False". The fourth Key shows the same background bitmap with Transparent set to "True".

Note

If you do not want your image to be stretched, or want to control the size, you should place the image on the Key, not use it as a background.

Further explanation of Transparency can be found in the section Understanding Images. You can experiment with background colors and images to see the results. Note that all bitmaps you use for caption or background will be included in the keyboard file. You don't need to deploy them to your users.

Now select **Tools ▸ Label tool** from the menu and place a Label on the last Key you created, with the text "Enter." Change the position of the Label to Left 6 and Top 41. While you still have the handles of the "Enter" label showing, hold <*Ctrl*> and click on the Image to choose both captions. Now Right click and choose **Align ▸ Right edges**. Click anywhere on the Keyboard Page and again on the Key. Select *Return* from the virtual key list and drag it to the Keystroke editor.

This is what your keyboard should now look like.

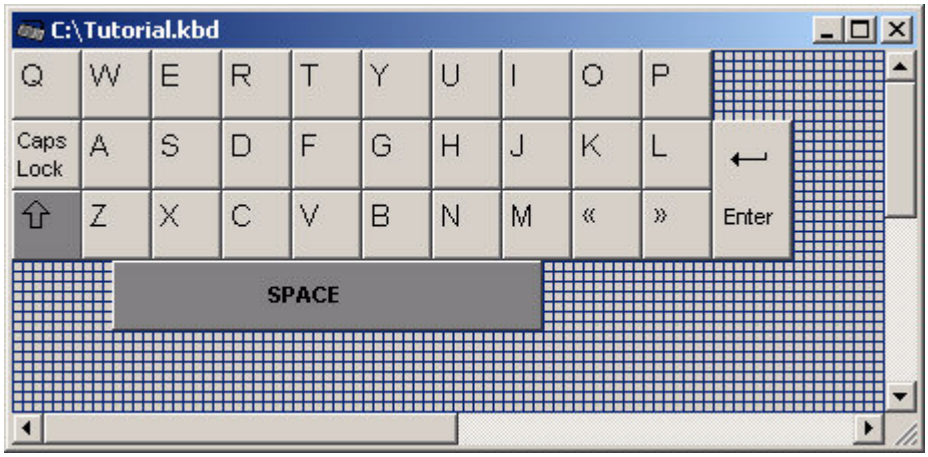

FIGURE 10 QWERTY basic keyboard with Enter key.

Select **File ▸ Save** to save your Keyboard.

In the next tutorial we will build a function key / numeric keyboard.

Chapter 4

Tutorial 3 – Numeric Keyboard

This tutorial will focus on building a numeric / function key keyboard.
Estimated time for completion of this section: 15 minutes.

The following tasks will be completed:

- Copying Keys from the Keypool
- Creating Keys
  - Adding Label objects
  - Adding Keystrokes to Keys
  - Adding Indicators

Start the MountFocus Keyboard Designer and select File ▸ Open from the menu. Select the file Tutorial.kbd and click on Open. Right click on the Keyboard Page and select New page from the popup menu.

For the basis of this keyboard, lets start by copying some keys from the Function Keypool.

Click on the Function tab in the Keypool and click on the "F1" to copy that key. Place the cursor in the upper left corner of the grid and click again. You should now have a key with "F1" on it.

Click on the Key you just placed, to select it for editing and change the Left position to 0, and the Top position to 0 if necessary. This will be our guide for placing the rest of the Keys on the keyboard.

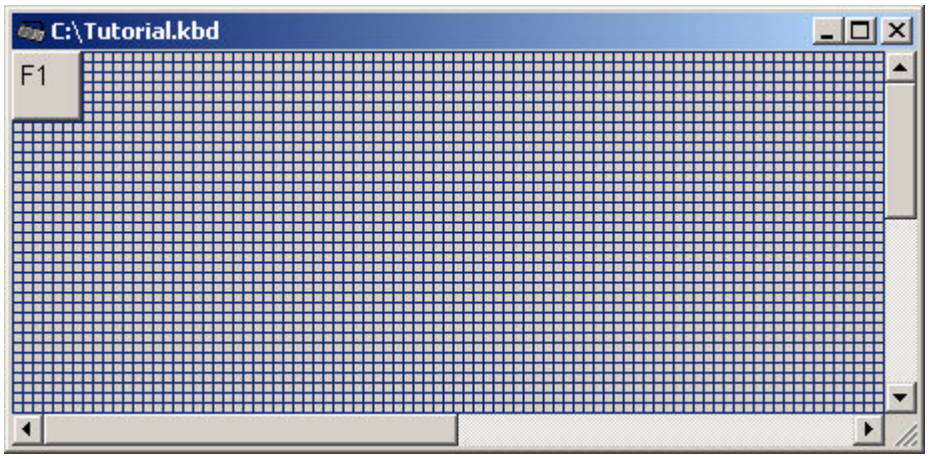

FIGURE 11 Placement of F1 key.

Continue copying keys from the Function and Numeric Keypools until you have a basic Function / Numeric keyboard like the one shown below.

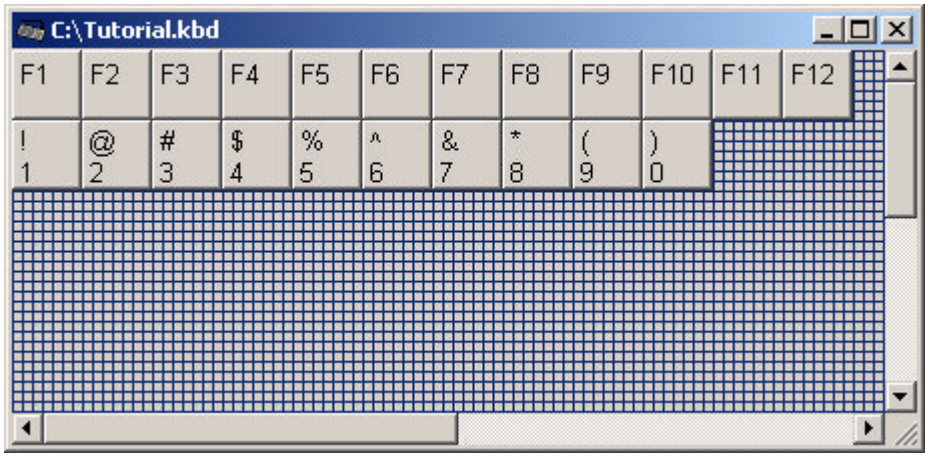

FIGURE 12 Basic Numeric / Function Key Keyboard

Now let's add a new Key to the Keyboard Page.

Place a new Key in the area to the left of the "F12" key. Make sure the key is located at Left 425 and Top 0. If it's not, change the location in the Property Editor. Change the Width to 55.

Now, place a Label on the Key. Click on the "Text" field in the property editor and type `Minimize` <*Enter*>. Adjust the font, position, height and width of the label if you wish.

We will add a function to this Key in the next tutorial.

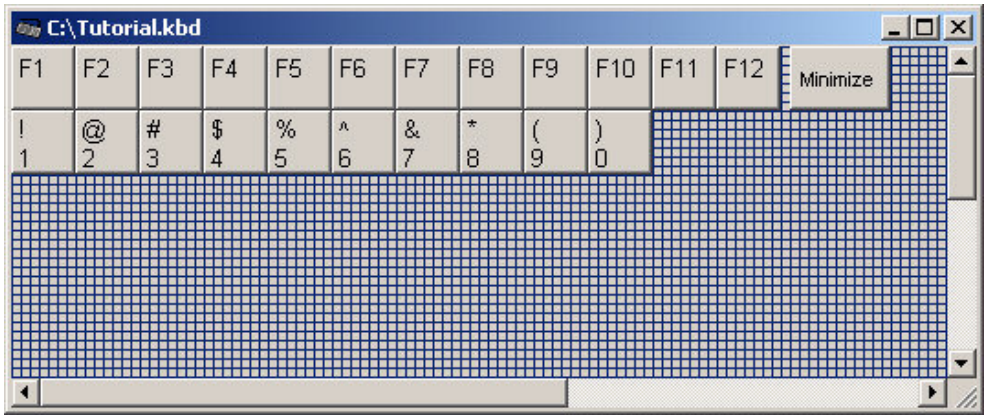

FIGURE 13 Numeric / Function Key Keyboard with Minimize key.

Click on any blank area of the Keyboard Page to exit the Key edit mode.

The next Key we will add is a Multi-line Key that will type a sentence when pressed. Create a Key below the "Minimize" Key, Height: 45, Width: 55; and place a label on the Key. Right click on the label, select **Edit text** from the pop-up menu and Type: `This is my<Enter>practice<Enter>keyboard.` in the text editor and select OK.

> **Note**
>
> Entering a hard return after the last line will insert a blank line in the caption.

You can adjust the placement of the Label until you are happy with the results.

Only the letter "T" should be a child of the *Shift* key. All other letters and commands should be dragged to the word "Keystrokes" in the keystrokes window, or all the letters will be capitalized.

Next, click on the Keyboard Page and again on the key to select it. Choose *Shift* from the virtual key list and drag it to the Keystroke editor. To make the Capital "T," Drag the letter "T" to the word "*Shift*" in the keystrokes window.

This is what the key message window should now look like.

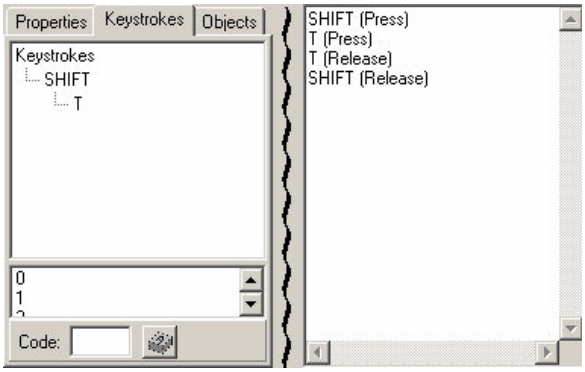

FIGURE 14 Keystrokes and Key messages for Shift + T.

Drag the following letters and commands from the virtual key list to the keystrokes window, in this order: "H, I, S, *Space*, I, S, *Space*, M, Y, P, R, A, C, T, I, C, E, K, E, Y, B, O, A, R, D." To find the virtual key code for "Full Stop," click on the key tool below the virtual key list as before and press the key on your physical keyboard to reveal the code. Drag the code, using the right mouse button, to the word "Keystrokes," and your Key is now complete.

This Key will now send the message to type "This is my practice keyboard." to your target application.

Here's what your keyboard should look like now.

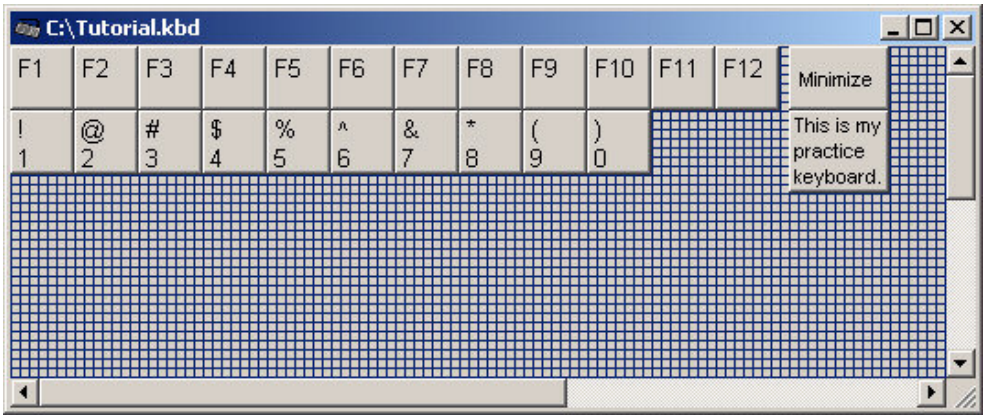

FIGURE 15 Numeric / Function Key Keyboard with multi-line caption key.

Now let's give a name to the page we are working on. Select Page2 from the drop down list at the top of the property editor and type `Numeric` in the Page Name field as the name for this page. Select **File ▸ Save** to save the changes you have made.

Choose the Alpha page from the Property editor drop down list and let's add a new Key to the right of the "P" Key. For this Key, we will add a Key that is identical to the Minimize Key on the numeric page. The position of the key and the label should be the same as before. We will add the functionality in the next tutorial. Check the screen shot below to see what your keyboard should look like.

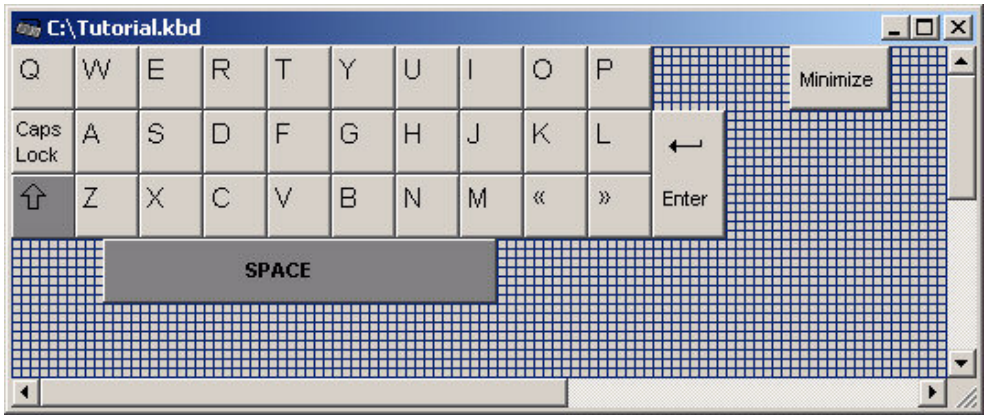

FIGURE 16 QWERTY keyboard with minimize key.

Let's add another page to the keyboard and add a Key in the same position as the Minimize keys on the Alpha and Numeric pages. This time, the Label should read "Maximize".

This is what your page should look like.

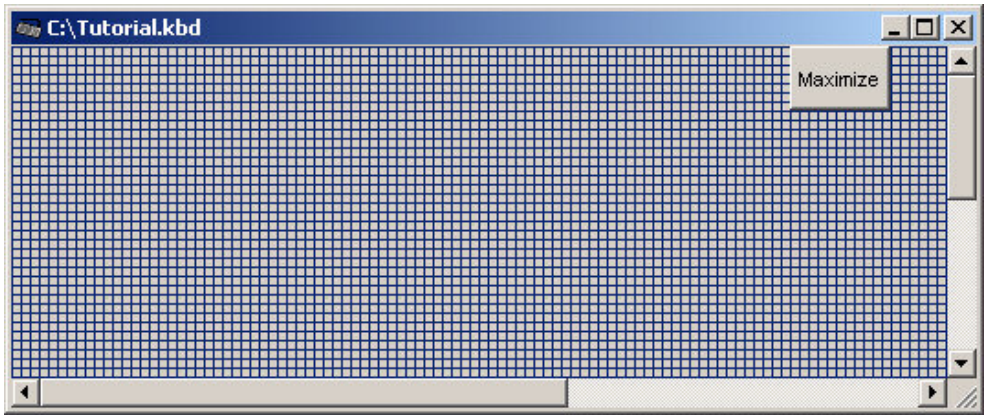

FIGURE 17 Maximize key.

Name this page "Max", and save your Keyboard.

Now for the final step in this tutorial: we will add a Label and an Indicator directly on the Keyboard Page. Place a Label on the "Alpha" Keyboard Page between the "P" key and the "Minimize" key. Position: Left: 375, Top: 0. Change the text to read "Caps".

Place an Indicator just below the Label, change the Transparent value to "False" and choose a bright background color. Shift click on the Indicator and place an Image on the Indicator. This will be the "Off" state of the Indicator. Click on the Image again to select it for editing and click on the ellipsis button that appears when you click on the Image field in the Property Editor. Select "off.bmp" from the file list.

Change the position of the Image to Left: 0, Top: 0, and click on the Keyboard Page. Click on the Indicator and then right click to bring up the popup menu.

Select **Show on caption** from the menu. You should now see the Indicator with the interior drawn in white. Place another Image on the Indicator in the same way as above, in the same position, but this time use "on.bmp" as the Image file.

Change the size of the Indicator to Height: 5, Width: 10 and the position to Left: 380, Top: 15. Also change the value of the Indicator field to "-1", because we want to monitor the Caps Lock function. Change the Transparent option back to "True."

This is what your Keyboard Page should look like.

Turn off the grid to view captions placed on the keyboard more easily.

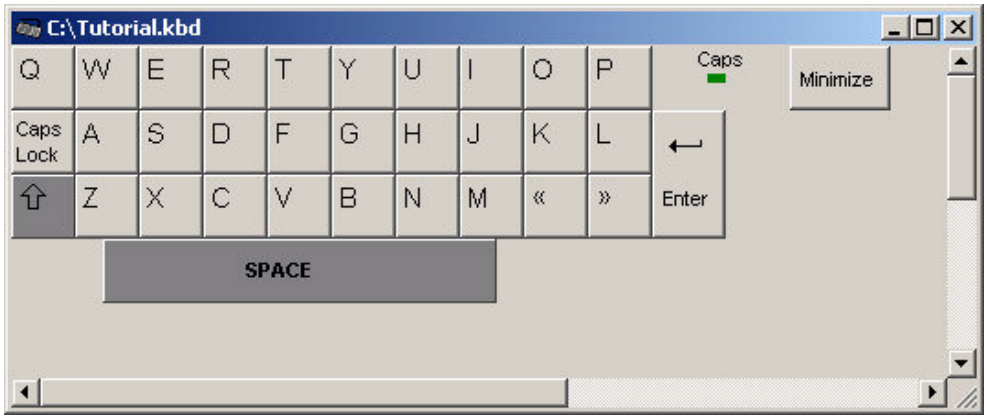

FIGURE 18 QWERTY keyboard with indicator

You can test your Indicator opening the Indicator Controller tool window and setting the Interval spin box to "50" and pressing the Caps Lock key on and off on your physical keyboard. Your Indicator should change color.

Save your keyboard.

In the next tutorial we will add functionality to the Minimize and Maximize keys and test all pages of the keyboard together.

Chapter 5

Tutorial 4 – Changing Pages / Regions

This tutorial will focus on changing pages and placing regions on a keyboard page Estimated time for completion of this section: 10 minutes.

The following tasks will be completed:

- Using Keys to change pages of a keyboard
- Adding Regions to a page
- Changing the size of the keyboard

Start the MountFocus Keyboard Designer and select File ▸ Open from the menu. Select the file Tutorial.kbd and click on Open.

Let's start by adding functionality to the Minimize and Maximize Keys you defined in the last tutorial. On the Alpha page, click on the Minimize key to select it. Click on the SelectPage field in the Property editor and select the Max page from the drop down list. When this key is selected, it will close the current page and open the Max page.

Select the Numeric page from the drop down list of pages and follow the same procedure for the Minimize Key on that page.

Now select the Max page from the drop down list and click on the Maximize Key to select it. This time, change the ClosePage value in the Property Editor to True. This Key will close the current page and open the last page that was open.

Now we can make the Maximize page appear to change size by adding a region to that page. Right click on any blank area of the Keyboard Page and select **New region** from the popup menu. A new Region will appear in the upper left corner of the Keyboard Page. Click on the Region to select it, right click and select **Send to back** from the popup menu. Move the Region to the area under the Maximize Key. The position should be Left: 420, Top: 0, Height: 40, Width: 60.

If you make the Region the same size as the key, you will not be able to move your keyboard around on the screen using the mouse. There must be some visible area of the keyboard not covered by an object in order for you to click on. Always try to make the Region just slightly bigger than the objects placed on it.

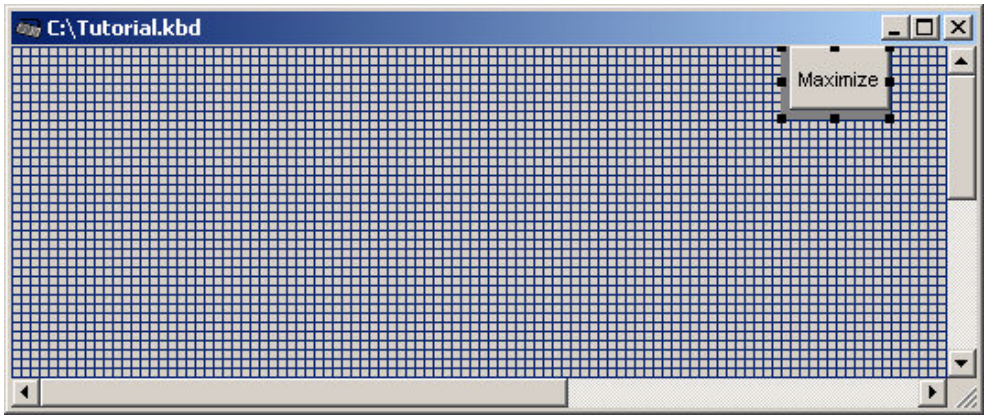

FIGURE 19 Placing a Region on the Keyboard Page.

The last thing we need to do is change the size of the Runtime Keyboard. Select the Alpha page from the drop down list, since this is the Keyboard Page with the largest area covered. Now select Keyboard from the drop down list and change the Height to 140 and the Width to 480.

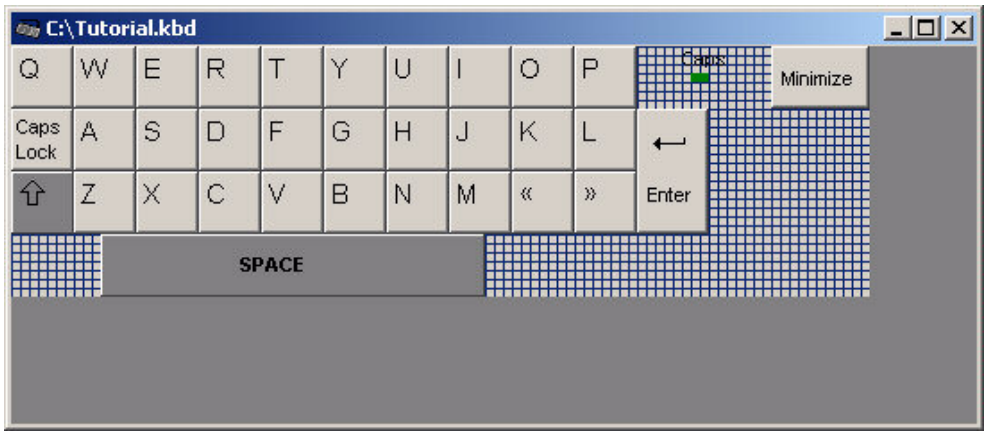

FIGURE 20 Resizing the Runtime Keyboard.

Notepad is a good application to use for testing.

Save your keyboard and let's test it! Click the "Start" button in Windows and select "Run". Find the "KBD.EXE" program (MountFocus Runtime Keyboard) that was installed together with the Keyboard Designer and enter the full path and filename for your keyboard file as a parameter (i.e. "C:\Program Files\MountFocus\Keyboard\Kbd.exe" Tutorial.kbd). This will start your keyboard. Activate any other application and click on any of the keys you defined.

Congratulations! You have completed the tutorials included in the MountFocus Keyboard Designer.

Section

3

Tool Window Reference

Chapter 6

The Property Editor

The property editor is where you set the values for an object's size, position and appearance. This chapter describes the property editor's functions in detail. For details about the different objects and their properties, refer to section 4.

When you select a Key, Label, Page or other object the Property Editor will display that particular object's properties. Each property is then available for you to manipulate. The method of manipulation will depend on the type of property you want to change. For example, the size and position properties can be changed in different ways. You can type a number in the Property Editor to change an object's size or position, or you can click on the object and move it around or size it using the mouse. The Property Editor will then display the objects new size and position as you move or size the object

When entering values in the Property Editor you can use the <*Arrow Up*> and <*Arrow Down*> keys to select the previous or next property in the list. Any changes you may have made will be stored.

Numeric properties

FIGURE 21 Numeric property ready for editing in the Property Editor.

When entering a numeric property, such as left position, width etc. you are only allowed to enter digits and the minus sign (-). You can press <*Enter*> to store or <*Esc*> to cancel the new value.

If the value you enter is invalid, the Property Editor will set a legal value for the property and display it in the Property Editor. For example if you have Snap to grid on and try to position an object between to grid cells, the Property Editor will automatically set the position to a value that fits in the grid.

Text properties

FIGURE 22 Text property ready for editing in the Property Editor

The Text property for the Label object also allows multiline text. See "Understanding Labels" for Details.

Text properties work in the same way as numeric properties, except of course, that they allow you to enter any character, not only digits.

Color properties

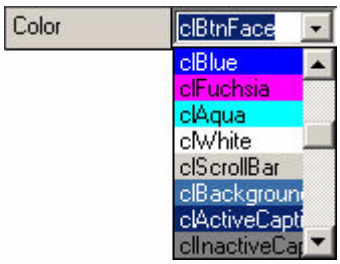

FIGURE 23 Color property selection in the Property Editor

In addition to the list of colors that appear in the Property Editor, you can also set the RGB color value from the object's popup (right click) menu.

Most of the objects in the MountFocus Keyboard Designer can have its own color. The Color editor in the Property Editor contains a list of standard colors you can choose from. In addition to fixed colors like Black, Red, Blue etc. you will see a list of user defined "Windows colors". These colors will vary in appearance depending on the selections made in the Display properties dialog box in Windows. By selecting colors from this list you can ensure that your keyboard will look good on any Windows computer, because the colors will change to fit the user's own selections.

You can also set an RGB value to specify an exact color for your object. To do this, select the object, right click on it, and select **Set background color** from the popup menu. This will bring up the standard Windows color dialog.

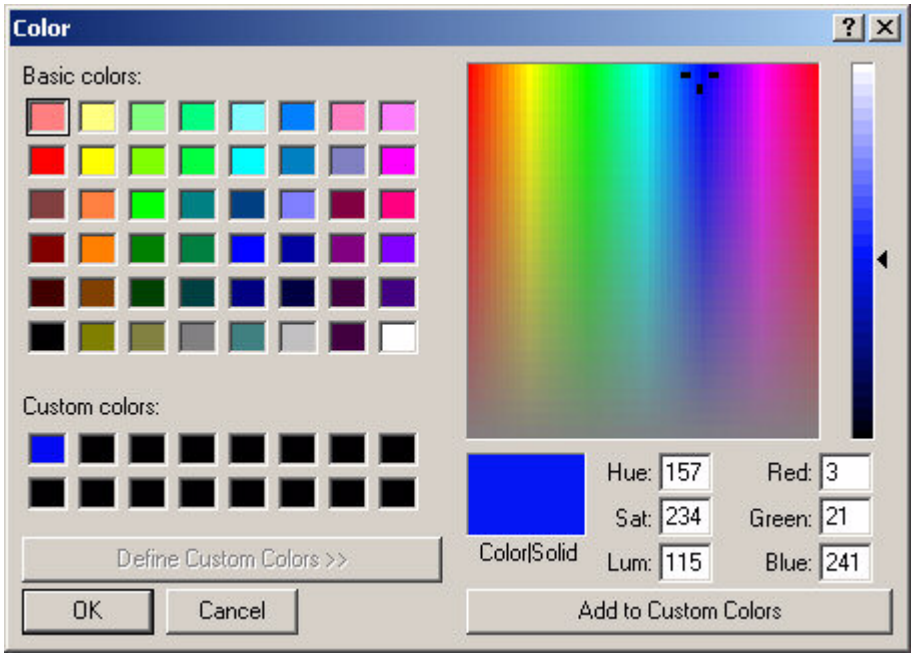

FIGURE 24 The Color dialog lets you choose any RGB color available. If you want to use the same color for many objects, click on the Add to Custom Colors button. You can then select you custom color for all the other objects.

In addition to a predefined set of 48 colors, you can specify any color by dragging the black cross hair to the color you want. You can also set the Red, Green and Blue values numerically (each ranging from 0 to 255).

Image properties

FIGURE 25 Image property in the Property Editor. Clicking on the ellipsis button (…) opens the Open dialog box that allows you to load a bitmap into the Image property.

The Property Editor displays the text "No image" or "Image" depending on whether or not a bitmap has been loaded for the property. To load a new bitmap, click on the ellipsis (…) button to open the Open image dialog box and then select the bitmap file you want to use. Only standard Windows .bmp files can be loaded.

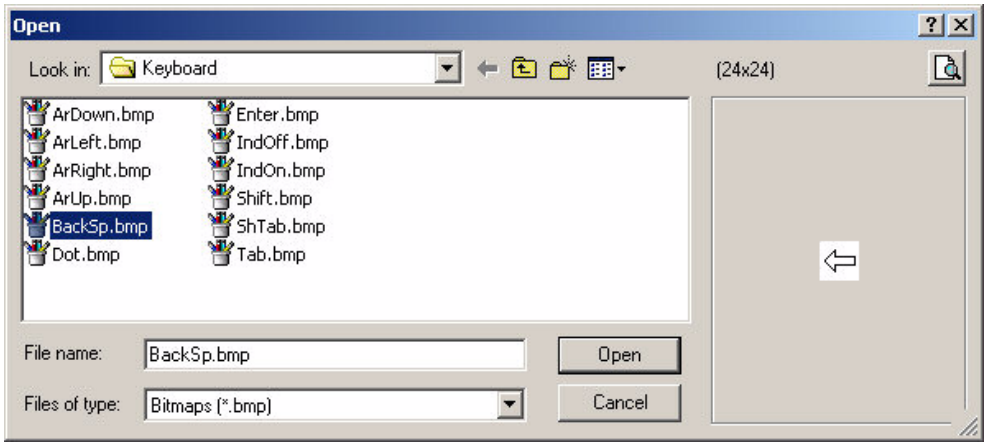

FIGURE 26 The Open picture dialog also shows you a preview of the picture you have selected.

To clear a bitmap, select the property in the Property Editor and press <*Delete*>.

Selection properties

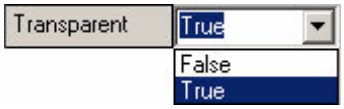

FIGURE 27 A selection property in the Property Editor, like the Transparent property, shows available options in a drop down list.

Properties that can be either on or off will appear as selection properties where "True" means "On" and "False" means "Off".

Many properties have a limited set of possible options that are presented in a drop down list. To change the value of a selection property, select the value you want from the drop down list, or double click the property to select the next value in the list.

Font properties

FIGURE 28 Font property in the Property Editor. Click on the ellipsis button (…) to open the Font dialog where you can view and edit all font settings.

The Property Editor will display the selected font's name and size to represent a font property. However, you can change all available font properties by clicking on the ellipsis (…) button to open the Font dialog box.

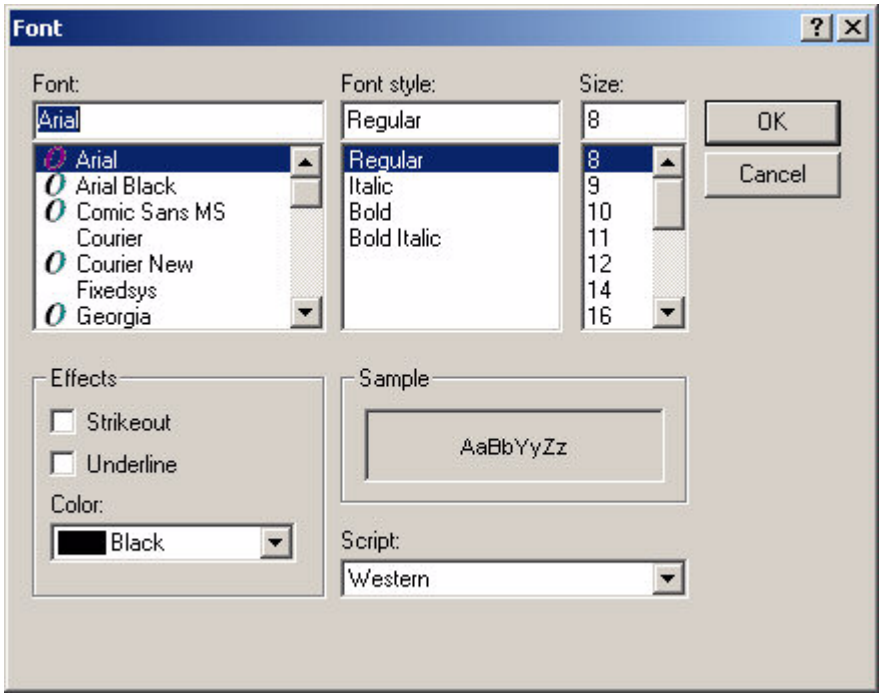

FIGURE 29 The Font dialog box allows you to select which font to use, as well as the font's style, size, color and script.

Note

Remember that fonts are installable components in Windows and your end user may not have the same fonts installed as you have on your computer. For this reason, it is usually a good idea to stick to "standard" fonts that ship with Windows.

Sound properties

FIGURE 30 Sound property in the Property Editor. Click on the ellipsis button (…) to open the Open dialog box that allows you to load a sound file into the sound property.

A sound file (.wav) can be loaded into your keyboard to be played every time the user presses a key. The Property Editor will display "No sound" or "Sound" depending on whether or not a sound file has been loaded. Again, you can click on the ellipsis (…) button to open a sound file, or press <*Delete*> to clear the sound property.

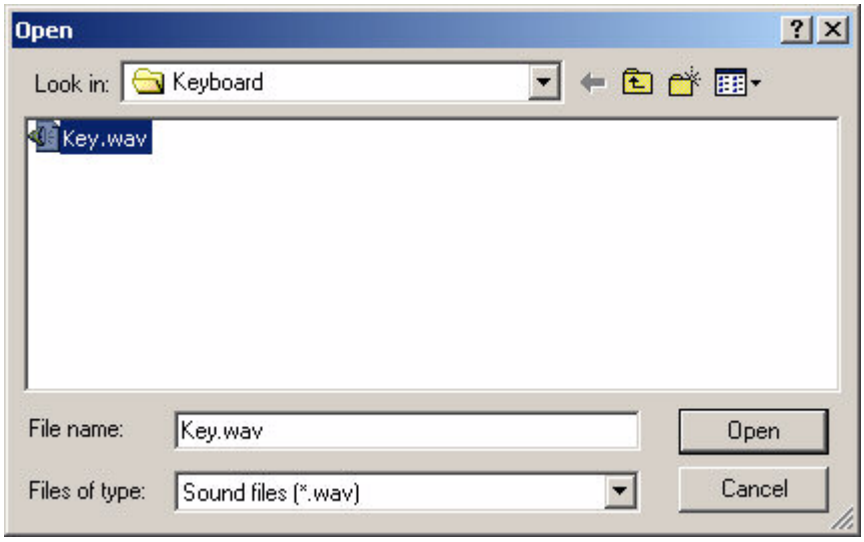

FIGURE 31 The Open sound dialog box allows you to choose a sound file to be loaded.

Only .wav files can be loaded into a sound property. If you have a sound in a different format, you must convert it before loading it into the Keyboard Designer. The Runtime Keyboard play sounds asynchronously, meaning that the operation of the Runtime Keyboard is not delayed by playing a sound. Still, it is a good idea to keep you sounds short so you don't drive your end users mad.

Chapter 7

The Keystrokes Editor

The Keystrokes Editor is where you define the keystrokes that a key will send to the target application. The Keystrokes editor works closely with the Virtual Key List and the Keystroke Preview tool windows.

Defining keystrokes in the MountFocus Keyboard Designer is mostly a drag and drop operation. By dragging virtual key codes from the Virtual Key List, you can define just about any keystroke combination imaginable. The Keystroke Editor shows the keystrokes as a tree view representing the order of the individual keyboard messages that will be produced by the Runtime Keyboard.

Basically, the Runtime Keyboard produces two different Windows messages that are sent to the target application. First a Key Down message to indicate that a key has been pressed, and then a Key Up message to indicate it has been released.

The order of these messages is vital. Imagine if you want to make a key to send <*Ctrl*>+<*C*> (standard Windows "Copy" shortcut) to the target application. If you send the *Control* virtual key down and up and then the *C* virtual key down and up, you will not get the desired result because the <*Ctrl*> key was released before the <*C*> key was pressed. To produce the correct result, you will have to define a key that presses and holds <*Ctrl*> while <*C*> is pressed and released.

FIGURE 32 You can define keystrokes by dragging a virtual key from the Virtual Key List to the Keystrokes Editor

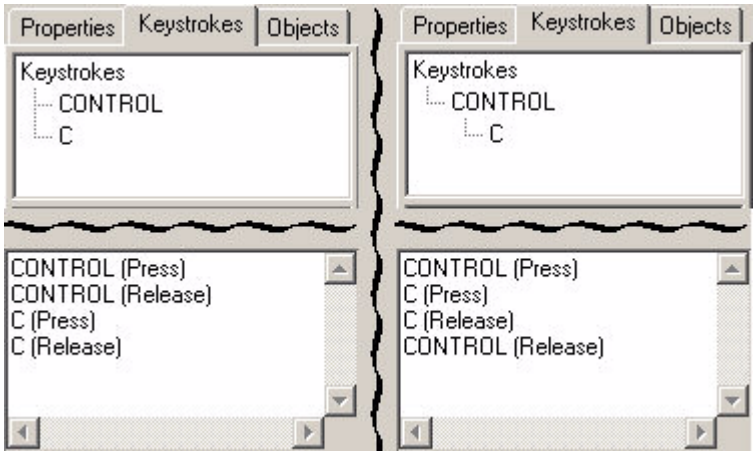

FIGURE 33 The Keystrokes Editor and Keystrokes Preview tool windows showing the keystrokes <*Ctrl*> then <*C*> (left) and <*Ctrl*> + <*C*> (right).

The Keystroke Preview shows, in detail, the messages that will be produced by the Runtime Keyboard.

Hence, the tree view. When you place the *C* virtual key as a child to the *Control* virtual key, the <*Ctrl*> key up message is not sent until the <*C*> message down and up are sent. In other words, the tree view produces keyboard messages by reading the tree top to bottom in this fashion:

1. Send Key Down message
1. Handle all children
2. Send Key Up message

Every virtual key is handled in the same way, so you can build this tree as deep as you like.

Dropping keystrokes

When you drop a keystroke from the Virtual Key List in the Keystrokes Editor, you must drop the new keystroke on an existing element in the tree view. The new keystroke will then become the last child of the element you dropped it on.

This means that when defining keystrokes you should drop them, as you would press them on a physical keyboard. I.e. to define <*Ctrl*>+<*C*>, first drop the *Control* virtual key on the "Keystrokes" element and then drop the *C* virtual key on the *Control* element.

Deleting keystrokes

To delete a keystroke, select it in the Keystrokes Editor and press <*Delete*> on your keyboard.

Note

When you delete an element in the Keystrokes Editor, all child elements are also deleted.

Complex keystrokes

Any keystroke that is available on your physical keyboard (except <*Tab*>) can be detected from the Virtual Key List tool window.

Some keystrokes are not defined in the Virtual Key List. These keystrokes include special symbols that some applications allow you to insert from a character map. The Character Map applet that comes with Windows allows you to choose such characters and insert them into your documents, but using the MountFocus Keyboard Designer you can now create keys that produce these special characters.

Let's say you want to define a copyright (©) key. If you open the Character Map applet and click on the © symbol, you will see the message "Keystroke: Alt+0169" on the status bar. This means you can hold the <*Alt*> key and press <*0*><*1*><*6*><*9*> on the **numeric keypad** to produce this character.

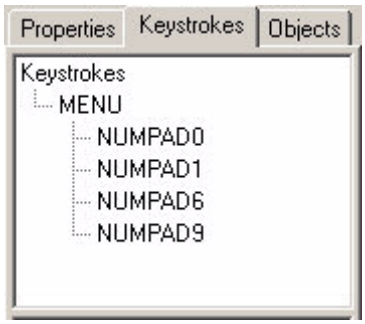

FIGURE 34 The Keystrokes Editor showing the keystrokes to produce a copyright (©) character.

Menu is the virtual key name for <*Alt*>

To make a key do this from the MountFocus Keyboard Designer, first drop the *Menu* virtual key on the Keystrokes Editor and then drop the *Numpad0*, *Numpad1*, *Numpad6* and finally *Numpad9* virtual keys on the *Menu* element – in that order. Voila, you now have a copyright key.

Extended keys

Because the original IBM PC, back in the dark ages, had a smaller keyboard (without the arrow key section in the middle), some keys are defined as extended. The MountFocus Keyboard Designer will automatically set the extended flag for known extended keys, but you can also do this manually, if you prefer.

To toggle the extended flag for a keystroke, double click the keystroke in the Keystrokes Editor. The text "(Ext)" will appear if the extended flag is on, and disappear if it is off.

Which keys are "extended keys"? The right hand side <*Ctrl*> and <*Alt*>, the <*Insert*>, <*Delete*>, <*Home*>, <*End*>, <*Page Up*>, <*Page Down*> and arrow keys in the keyboard section to the left of the numeric key pad as well as the <*Num Lock*>, <*Break*>, <*Print Screen*>, <*Divide*> (on the numeric key pad) and <*Enter*> (on the numeric key pad) keys.

The most important key to notice is the <*Enter*> (on the numeric keypad) key. This key has the same virtual key code as the "normal" <*Enter*> (*Return*) but is an extended key. When you drag the virtual key *Return* to the Keystrokes Editor, the extended flag will not be set by the Keyboard Designer, because the "normal" <*Enter*> key is not extended. Therefore, to produce a correct numeric keypad <*Enter*> you have to set this flag yourself. If you don't, no one will probably ever notice ☺.

Chapter 8

The Virtual Key List

The Virtual Key List displays a list of all virtual key codes defined by Windows. You can drag these codes to the Keystrokes Editor to build a sequence of keystrokes to be sent to the target application. You can also use the Virtual Key List tool window to find key codes for special or foreign characters not specifically defined by Windows.

The Virtual Key List can be used in two different ways to find the keystroke you are looking for. Most of the keys you will need are already defined in the Virtual Key List and have logical names. However, the MountFocus Keyboard Designer uses the standard internal Windows names for the virtual keys, and some of the names are not so logical. The first one you will probably look for (and have trouble finding) is the <*Alt*> key that is called *Menu* in virtual key lingo.

The quick and easy way to define keystrokes is to click once in the Virtual Key List and then press the first letter of the name for the virtual key you are looking for. When you find it, drag it over to the Keystrokes Editor and then press the first letter of the next key you want and so on.

FIGURE 35 Dragging a virtual key from the Virtual Key List to the Keystrokes Editor.

Finding odd-named virtual keys

You can not use this method to find <*Tab*>, as the <*Tab*> key is used to move focus in the Window. The virtual key name for <*Tab*> is (drumroll) *Tab*

As mentioned before, some virtual keys have strange names and others, especially symbols and foreign characters, don't have a virtual key name at all. You can find the virtual key code for these keys easily by clicking on the key button at the bottom right of the Virtual Key List and then pressing the desired key on your physical keyboard.

FIGURE 36 Finding the code for a keystroke in the Virtual Key List.

A hexadecimal code representing the virtual key code will then appear in the Code field. You can then drag the code **using the right**

mouse button to the Keystrokes Editor, and place it just like you would from the Virtual Key List. If it turns out that the code represents one of the predefined virtual keys, the Keystrokes Editor will replace the code with its virtual key name. This would be the case for the example in FIGURE 36 that represents the <*Alt*> key (virtual key name *Menu*).

Chapter 9

The Keystroke Preview

The Keystroke Preview tool window will show you, in detail, which keyboard messages the Runtime Keyboard will produce; and in which order.

When you define keystrokes in the Keystrokes Editor, the Keystrokes Preview automatically updates with a list of the keyboard messages that the Runtime Keyboard will produce. The list shows in detail the order of pressing and releasing the individual keys defined in the Keystrokes Editor. This gives you a good idea of what exactly the target application will receive, so you can avoid "bugs" in your keystroke definitions.

For single keystroke keys the Keystroke Preview will simply show "<Virtual Key> (Press)" and "<Virtual Key> (Release)" to indicate that the virtual key in question will be pressed and released.

For more complex keys, the Keystroke Preview can be very useful. If we reproduce the copyright (©) example from the "Keystrokes Editor" chapter, the output will be as shown in FIGURE 37 below.

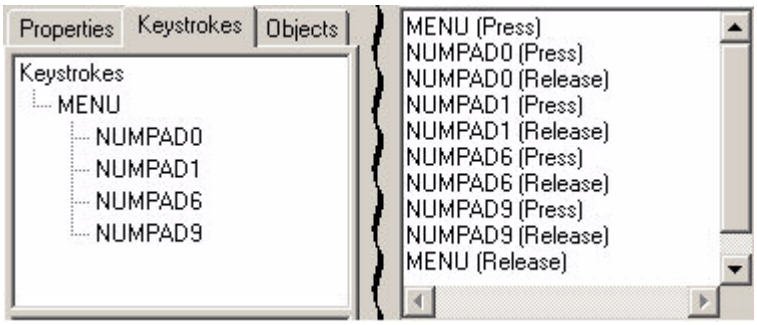

FIGURE 37 The Keystrokes Editor and Keystrokes Preview tool windows showing the definition and result of a copyright (©) key.

As you can see the *Menu* virtual key (<*Alt*>) is pressed, then the four keys on the numeric keypad are pressed and released and finally the *Menu* key is released.

Chapter 10

The Keypool

The Keypool is a collection of predefined keys that you can copy to your own keyboards. You can also edit the Keypool and make your own changes, or replace the whole Keypool with your own.

The Keypool can be a great time saver if you frequently make similar, but still different keyboards. In general terms, it allows you to save a collection of keys and other objects for reuse on other keyboards or pages. The Keypool files are identical to the regular keyboard files, but the Keypool "keyboards" are usually designed with a layout more suitable for the toolbar than for actual use as a keyboard.

FIGURE 38 The standard Keypool showing the "alpha" Page. The other Pages are available as tabs.

Using the Keypool

The mouse cursor will change to the Key Tool cursor when you select an object from the Keypool. When you place the object on a page, the Select Tool is automatically selected.

To copy a Key from the Keypool, simply click on it and then click on a blank spot on the Keyboard Page where you want to place the Key. All properties, except the position, are copied. Changing any properties on the new Key will not affect the original in the Keypool.

You can also copy Label, Image and Indicator objects from the Keypool. The operation is the same as for keys, but unlike keys, these objects don't change shape to indicate that they have been selected.

Modifying the Keypool

When the MountFocus Keyboard Designer starts, it automatically loads the last Keypool you were using. You can open any Keypool file for editing by choosing **File ▸ Open** from the menu and selecting "Keypool files" in the "Files of type" drop down list.

Note

Keyboard files and Keypool files are compatible, meaning that you can open a Keypool file for editing, just like a "normal" keyboard file. Similarly, you can specify to use any standard keyboard file as your Keypool.

The MountFocus Keyboard Designer does not make any file locks on your keyboard or Keypool files while you work with them. This allows you to edit the same Keypool file that is currently open in the Keypool as well as testing the keyboard you are working on with the Runtime Keyboard.

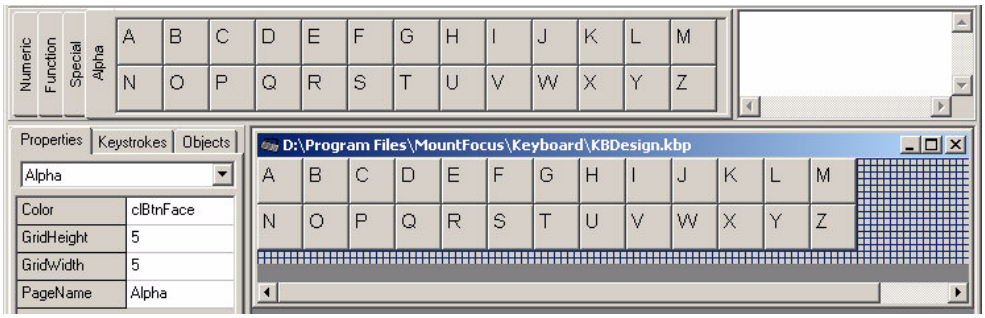

FIGURE 39 A Keypool file can be opened for editing even if it is currently open in the Keypool tool window.

When you design or change a Keypool file, keep in mind that the layout should fit the space the Keypool tool window uses as much as possible. I.e. make the Keypool file wide, and keep its height to a minimum if you place the Keypool on the top or bottom of the main window.

Keypool pages

Given its limited screen real estate, the Keypool can easily be overcrowded. You can solve this problem by designing multi-page Keypool files just as you can for a "normal" keyboard. The pages will appear in the Keypool tool window as tabs showing the page name on each tab. This way you can easily design Keypools with hundreds of objects without getting an overcrowded Keypool

Selecting a different Keypool

To open a different Keypool file in your Keypool tool window, select **File ▸ Select Keypool** from the main menu. In the Open file dialog box, select the Keypool (or keyboard) file you want to open in the Keypool tool window.

All Key, Label, Image and Indicator objects in the Keypool file you selected will become available for copying to other keyboards.

Chapter 11

The Indicator Controller

When designing indicators you can use the Indicator Controller tool window to test how your indicators will work when active in the Runtime Keyboard.

The Indicator Controller allows you to simulate different indicators being switched on and off from within the Keyboard Designer. There are three fields on the Indicator Controller that you can use for this purpose, Interval, Indicator # and State. For details on creating Indicator objects, refer to the "Understanding Indicators" chapter.

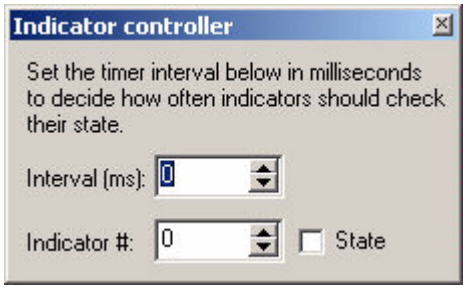

FIGURE 40 The Indicator Controller set to checking the different indicators' state every 50 milliseconds. Also the state for indicator number 2 is set to on.

Setting the interval

The Interval field allows you to set how often the Indicators should check their state. You set the Interval in milliseconds (ms = 1/1000 of a second). When the Interval is 0, indicators will not check their state, thus allowing you to work with them as normal without interference.

Values less than 50 does not create faster changing indicators as Windows' internal timer only updates the time 18 times per second (every 55.55 milliseconds).

When you set the Interval to a positive value, for example 50, the indicators will check their state every 50 milliseconds (or whatever you specify). If you have Indicator objects on your active Keyboard Page that are connected to any of the system variables (<*Caps Lock*>, <*Num Lock*> or <*Scroll Lock*>), you can see them change state by pressing the corresponding key on your physical keyboard.

Testing user defined indicators

User defined indicators have values ranging from 0 to 250, while system variables are –1 for <*Caps Lock*>, –2 for <*Num Lock*> and –3 for <*Scroll Lock*>. See the "Understanding Indicators" chapter for details.

If you have defined your own indicators that are not connected to either of the system variables you can set their state (on or off) by selecting the indicator number in the Indicator # field in the Indicator Controller tool window. The State check box will then be checked if the selected indicator number is on and unchecked if it is off. Clicking the State field will toggle the selected indicator number's state and (if the Interval is set) update all visible Indicator objects related to the selected indicator number.

Note

Indicators in the Keypool are not simulated. They remain in their off state until copied to a keyboard page.

Chapter 12

The Object List

Using the Object List, you can quickly get an overview of all objects on the current Keyboard Page.

When you click on a Keyboard Page, Key or Indicator, the Object List is updated with a list of all objects that belong to the selected object. This way you can easily get an overview of the available objects. The Object List also allows you to select one or more objects from the list and work with them as if you selected them using the mouse.

Note

To operate on objects placed on a Key or Indicator, the corresponding Key or Indicator must be in edit mode.

The Object List toolbar

When you select one or more objects from the Object List, you can use the tools on the toolbar to Select the selected objects, Bring to front or Send to back.

FIGURE 41 The Object List tools Select, Bring to front and Sent to back.

To select multiple objects in the Object List hold <*Shift*> or <*Ctrl*> pressed while you select (same as Windows Explorer).

Section

4

MOUNTFOCUS INFORMATION SYSTEMS LTD.

Keyboard Designer User's Guide

Object Reference

Chapter 13

Understanding the Keyboard

Only one Keyboard object exists for each keyboard file. The Keyboard object holds one or more Keyboard Pages and determines the size of the Runtime Keyboard window.

You can consider the Keyboard object to be the master container that keeps track of all the other objects in your keyboard file. Therefore, the Keyboard object is used to store global properties that apply to the keyboard file as a whole.

What you see and work with in the Keyboard Designer are Keyboard Pages. See the "Understanding Keyboard Pages" chapter for details.

Since the Keyboard object is invisible, you cannot click on it to select it. To make the Keyboard object appear in the Property Editor, select the "Keyboard" item from the drop down list at the top of the Property Editor.

Keyboard properties

KeySound

By loading a .wav file from the Property Editor into the KeySound property, you specify that this sound should be played each time a key is pressed in the Runtime Keyboard. The sound is also played in the Keyboard Designer, when you enter edit mode for a key.

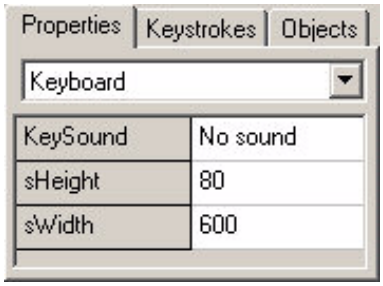

FIGURE 42 The Keyboard objects's properties.

Just like images, the sound file is integrated into your keyboard file so you do not need to distribute the sound file with your keyboard files. In other words: the Runtime Keyboard is capable of running any keyboard file without any other supporting files.

sHeight and sWidth

Unlike the previous versions of the MountFocus Keyboard Designer, the keyboard is not resized when you resize the working window in the Keyboard Designer. You have to set the size of the keyboard yourself in the Property Editor. You do this by specifying the keyboard width in pixels in the sWidth property, and the height in pixels in the sHeight property. There are no left or top properties as this is determined by the Runtime Keyboard settings.

The size you specify here will determine the working space in the Keyboard Designer as well as the physical size of the Runtime Keyboard window.

You can make each page in your keyboard appear to have a different size by using regions. See the "Understanding Regions" chapter for details.

Chapter 14

Understanding Keyboard Pages

Utilizing Keyboard Pages allows you to define keyboards that change appearance and function depending on the situation.

The changing of pages can be controlled from any other application by using OLE, or directly from your keyboard by creating special keys that change the current page. While using the OLE approach provides the greatest flexibility, allowing your application to automatically change pages, thus making a "context sensitive" keyboard; this approach does require programming changes to your existing application.

The purpose of Pages

Pages can be used in a number of ways, but as mentioned initially, their main purpose is to allow you to create a keyboard that can change in appearance and function depending on the situation.

For details on making keys change pages, see the "Understanding Keys" chapter.

One example of such a situation is if you have limited screen real estate available for your virtual keyboard. You can then create one page with alphabetic keys only, one page with numeric and function keys and so on. On each of the pages you can then place a key that sends no keystrokes, but instead changes the active page to one of the others, thus letting the user select which page to use.

Another example is if you are making an application where you have a number of special functions that are available in different situations. Let's say you are making a POS (Point Of Sale) application and you want different keys to be available when the user is logged on and ready to specify items, than when the user is logged off and should only be able to log on or shut down the application. You can then use OLE to make your application decide which page to show, thus limiting the users available keys to those appropriate to the situation.

Page properties

Color

Changing the page Color property changes the background color for the entire page. You can select any of the standard colors from the Color property drop down list in the Property Editor, or you can right click on the current Page and choose **Set background color** from the popup menu to choose any RGB color value for your Page.

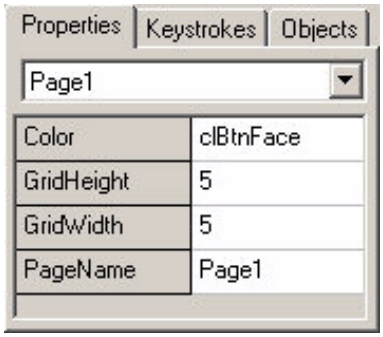

FIGURE 43 The Keyboard Page objects's properties.

GridHeight and GridWidth

The grid is the horizontal and vertical lines that you see on all the Keyboard Pages (if the **View ▸ Show grid** main menu option is checked). The height and width of this grid can be set individually for each page in your keyboard by changing these properties in the Property Editor. In addition to providing a means of visual measurement, you can also check the main menu option **View ▸ Snap to grid** to make all objects move and size in grid increments. I.e. if your grid width is 5, all objects will have a left position divisible by 5, and their width will be divisible by 5. Both the GridHeight and the GridWidth properties are specified in pixels.

> **Note**
>
> Objects that have their AutoSize property set to "True" will be sized irrespective of the grid size.

PageName

The PageName property allows you to give each page a meaningful name. The MountFocus Keyboard Designer allows you to use any character in the PageName property, but we recommend you use only alphanumeric characters and avoid using spaces. This makes it a lot easier to program page changes via OLE. The page name is never displayed by the Runtime Keyboard, so it will only be visible in the Keyboard Designer in the Property Editor (drop down list at the top and the Key object's SelectPage property) and in the Keypool.

Adding new Pages

To add a new page to your keyboard, right click the current Page and choose **New page** from the popup menu. Alternatively you can choose **Tools ▸ New page** from the main menu.

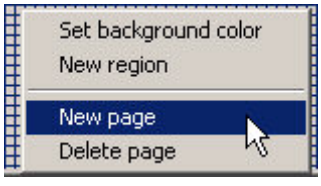

FIGURE 44 The Keyboard Page popup menu, creating a new page.

Deleting a Page

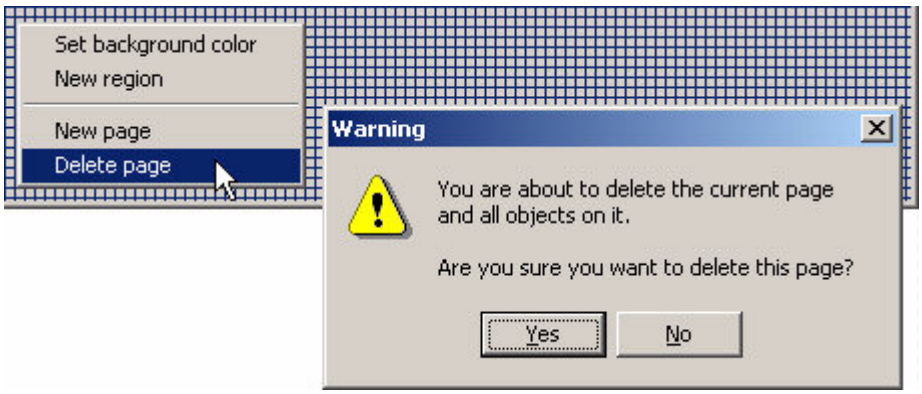

FIGURE 45 The Keyboard Page popup menu and Page delete confirmation window.

To delete the active Page, right click the page and choose **Delete page** from the popup menu. Since deleting a Page also deletes all objects on that page, you will be asked to confirm the Page delete command.

Selecting the working Page

To select which Page to work with, select the Page you want from the drop down list at the top of the Property Editor.

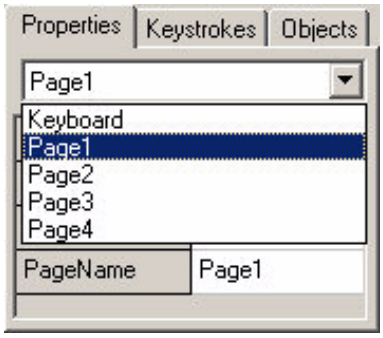

FIGURE 46 Selecting the working Page from the Property Editor drop down list.

Chapter 15

Understanding Labels

Label objects are used to present text on a Key, Indicator or Keyboard Page.

The Label object is used for presenting text in a vast variety of ways. Although it is apparently a simple object, it can be used to create many special effects by using symbol fonts and experimenting a bit with the properties.

FIGURE 47 These four Label objects show that you can create cool keyboards without adding graphics. Two of the Label objects (the "world" and the "webs") use the Webdings font that ships with Windows 2000.

Label properties

AutoSize

When the AutoSize property for a Label object is set to, "True" (recommended) the Label's sHeight and sWidth properties will be set automatically by the Keyboard Designer whenever the Text or Font properties change. You can still override the automatically set values by resizing the Label object or changing its sHeight or sWidth properties in the Property Editor.

When the AutoSize property is set to "False", no automatic calculation is performed.

Color

The Label object's Color property decides the Label's background color. You can choose a standard color from the Color property's drop down list box in the Property Editor, or

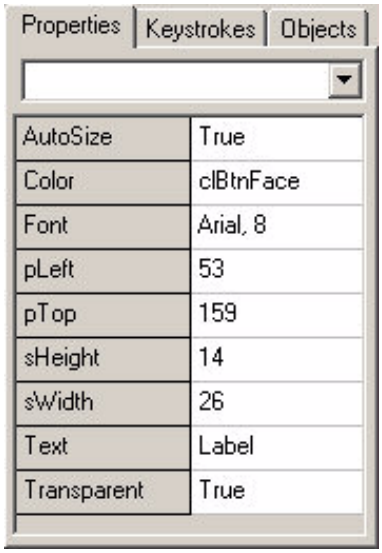

FIGURE 48 The Label objects's properties in the Property Editor.

you can select the Label object and then right click on it and choose **Set background color** from the popup menu.

> **Note**
>
> If the Transparent property of a Label object is set to "True", the background, and thus the background color, is not painted. To make the background color visible, set Transparent to "False".

Font

The Font property dictates how the text for your Label is presented. In the Property Editor only the Font's, name and size is displayed. To see details about the Font settings, or to make changes, click on the ellipsis (…) button for the Font property in the Property Editor. This opens the Font dialog box where you can change the font settings to your liking.

pLeft and pTop

The pLeft and pTop properties represent the left and top position of the Label object. Both properties are specified in pixels starting at the left and top edges of the parent object respectively. The parent object is the Keyboard Page, Key or Indicator on which the Label object is placed.

sHeight and sWidth

Similarly, the sHeight and sWidth properties specify the Label object's height and width in pixels. Usually, these properties are set automatically (if AutoSize is set to "True") and you don't need to change them.

Text

The Text property, naturally, specifies the text to show in the label. You can write as many lines of text as you wish in this property, separating each line with a vertical bar (|) character. An easier way to write long Labels is to select the label and right click on it and then choosing **Edit text** from the popup menu. This opens the Edit Text dialog box where you can enter multiple lines of text in a more sensible fashion.

> **Note**
>
> Since the vertical bar (|) character is used to specify a new line, you cannot make a Label display a vertical bar. To solve this issue you can use an Image object with the Dot.bmp bitmap that comes with the MountFocus Keyboard Designer. To make the Dot.bmp look like a vertical bar, set its Transparent property to "False", AutoSize to "False" and use its sHeight and sWidth properties to specify the size of the vertical bar.

Transparent

Setting the Transparent property of a Label object to "True", causes the label to paint itself without a background, thus letting any objects behind it shine through. When the Transparent property is set to "False", the Label will fill its entire background with its background color before painting the text.

Chapter 16

Understanding Images

Image objects are used to place bitmaps on Keys, Indicators and Keyboard Pages.

The Image object is used as a placeholder for a bitmap. You can load any bitmap file (.bmp) into an Image object. To manipulate the graphic itself you have to use a graphics design program, like the Paint applet that ships with Windows. To load graphics of other file formats, you have to use a graphics conversion program (or the SaveAs or Export functions in some graphics design programs) and save your bitmap as a .bmp file.

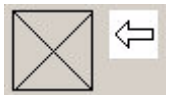

FIGURE 49 Two image objects, the first one being empty (no bitmap loaded).

All bitmaps you load are included in the keyboard file. This means that when you want to distribute your keyboard, you do not need to distribute any bitmap files with it.

Image properties

AutoSize

When the AutoSize property for an Image object is set to, "True", the sHeight and sWidth properties are set automatically equal to the bitmap's height and width in pixels. This ensures that the Image will always display exactly as designed.

When the AutoSize property is set to "False", you can resize the Image object and the bitmap will be stretched to fit the Image object.

AutoSize has no effect on an Image object until a bitmap has been loaded.

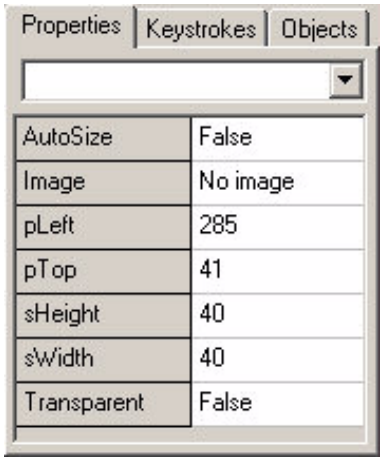

FIGURE 50 The Image objects's properties in the Property Editor.

Image

Before you load a bitmap into the Image object, the text "No image" is displayed in the Property Editor. To load a bitmap file, click on the ellipsis button (…) on the Image property in the Property Editor to open the Open dialog box. From there, select the bitmap you want to load and

click Ok. This operation will also clear any existing bitmap from the Image object, if present.

To clear the loaded bitmap from the Image object, select the Image property in the Property Editor and press <*Delete*> on your keyboard.

pLeft and pTop

The pLeft and pTop properties represent the left and top position of the Image object. Both properties are specified in pixels starting at the left and top edges of the parent object respectively. The parent object is the Keyboard Page, Key or Indicator on which the Image object is placed.

sHeight and sWidth

Similarly, the sHeight and sWidth properties specify the Image object's height and width in pixels. If the AutoSize property for the Image object is set to "True" and a bitmap has been loaded, you will not be allowed to change these properties.

Transparent

When the Transparent property for an Image object is set to "True", the MountFocus Keyboard Designer selects the color present on the pixel in the **lower, left corner** of the bitmap as the "transparent color". All other pixels on the bitmap that have the same color as the transparent color will not be displayed.

To place an object in front or behind another, select the object, right click and choose **Bring to front** or **Send to back** from the popup menu.

To see a good example of this, create an Image object and load the Backsp.bmp file that comes with the Keyboard Designer into it. This bitmap shows a black arrow on a white background. Setting Transparent to "True" chooses white as the transparent color (since the lower, left pixel is white) and only displays the black arrow. Any object existing behind the Image object will shine through the "transparent" area of the Image.

Setting the Transparent property to "False" causes the entire Image object to be filled by the bitmap "as is".

Image considerations

It's usually a good idea to stick to small, simple bitmaps. First of all this keeps your keyboard files in a reasonable size, but large images also takes longer to draw thus slowing down the displaying of your keyboard. Remember that the end user needs a keyboard that is intuitive and easy to use – not a piece of art.

Also, when you design your keyboard graphics it can be a good idea to make the background color something hideous that you will never use as a part of the graphics itself. When you load the graphics into the Keyboard Designer (and set Transparent to "True") the background color will not show anyway.

Chapter 17

Understanding Indicators

The Indicator object can change appearance depending on certain external factors.

By using Indicator objects on your keyboards, you can create keyboards that can respond to external factors by changing appearance. The most common use of Indicators is to indicate the state of the <*Caps Lock*>, <*Num Lock*> and <*Scroll Lock*> keys (on or off). However, the MountFocus Keyboard Designer also allows you to define your own indicators to indicate any special state (on or off) in your own program.

FIGURE 51 Three identical Indicator objects connected to different indicator numbers.

The indicator range

The MountFocus Keyboard Designer and Runtime Keyboard both monitor 254 different indicators each identified by a number ranging from –3 to 250. The first three (–3 to –1) monitor the system variables <*Scroll Lock*> (–3), <*Num Lock*> (–2) and <*Caps Lock*> (–1). The rest (0 – 250) are left for you to define as you please.

Note

Don't confuse indicators in the indicator range (numbered –3 to 250) with Indicator objects. Each Indicator object can act on one (and only one) indicator in the indicator range. For example, you can make an Indicator object to act on the <*Caps Lock*> indicator (indicator number –3). This particular Indicator object will only monitor that one indicator, but you can make as many Indicator objects as you wish to monitor the <*Caps Lock*> indicator (or any other indicator).

Defining your own indicators in the indicator range only makes sense if you are going to control the Runtime Keyboard from your own application via OLE. Using the OLE interface is the only way to switch user defined indicators on or off. The three system variables, however, work without using OLE.

Designing Indicators

When you are working with an Indicator object in the Keyboard Designer, it is usually a good idea to set the background color to something visible (i.e. a different color from the background and other objects nearby) and set the Transparent property to "False". This gives you a good visible indication on the Indicator object's boundaries.

When you place a new Indicator object on a Keyboard Page it will be displayed in it's off state. As long as the Indicator object is empty (has no Label or Image objects on it) it will display as a frame with black interior for the off state and white interior for the on state.

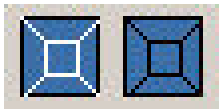

FIGURE 52 Empty Indicator objects showing the On caption (left) and Off caption (right)

Note

The Indicator object itself has no visible parts (except the background color). You design the visual look of an Indicator object by placing Label and Image objects on the Indicator object. The Indicator object provides two separate "captions" (similar to "mini Keyboard Pages") for this purpose. One for the on state of the Indicator object and one for the off state. The next couple of paragraphs explain how to design your Indicator.

Before you can place Label or Image objects on the Indicator object, you have to put the Indicator object in "edit mode". You do this by holding the <*Shift*> key pressed while clicking on the Indicator. You can now place Label or Image objects on the Indicator by choosing the appropriate tool and clicking on a blank spot inside the Indicator object's boundaries. Now, you can click on the new object to set its properties as usual. The Indicator object will remain in edit mode until you click on any object outside it (any object that does not belong to the Indicator being edited).

To change the state of the Indicator object so you can design the on (or off) state, first exit edit mode by clicking outside the Indicator object. Then select it again, right click on it, and choose **Show on caption** (or **Show off caption**). You can now design the other state of the Indicator object in the same way as you did the first.

Indicator properties

Color

The Indicator object's Color property decides the Indicator object's background color. You can choose a standard color from the Color property's drop down list box in the Property Editor, or you can select the Indicator object and then right click on it and choose **Set background color** from the popup menu. The background color of an Indicator object remains the same regardless of the indicator's state.

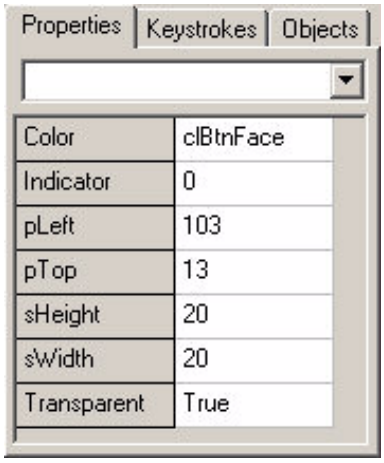

FIGURE 53 The Indicator object's properties in the Property Editor

Note

If the Transparent property of an Indicator object is set to "True", the background, and thus the background color, is not painted. To make the background color visible, set Transparent to "False".

Indicator

This property specifies which indicator number in the indicator range the Indicator object should monitor. The Indicator property accepts numeric values ranging from –3 to 250 defined as follows:

–3 = Monitor the <*Scroll Lock*> state

–2 = Monitor the <*Num Lock*> state

–1 = Monitor the <*Caps Lock*> state

0–250 = Monitor user defined indicators. These indicators must be set on or off from another application via OLE.

pLeft and pTop

The pLeft and pTop properties represent the left and top position of the Indicator object. Both properties are specified in pixels starting at the left and top edges of the parent object respectively. The parent object is the Keyboard Page or Key on which the Indicator object is placed.

sHeight and sWidth

Similarly, the sHeight and sWidth properties specify the Indicator object's height and width in pixels.

Transparent

Setting the Transparent property of an Indicator object to "True", causes the Indicator object to paint itself without a background, thus letting any objects behind it shine through in areas not covered by Label or Image objects that are part of the caption. When the Transparent property is set to "False", the Indicator

object will fill its entire background with its background color before painting the text.

Chapter 18

Understanding Keys

The Key object is, of course, what it's all about. In addition to sending keystrokes, the Key object can also be used for other purposes.

Naturally, the Key object's primary function is to send keystrokes to the target application. When you select a Key object, the Keystrokes Editor becomes active and allows you to define just about any keystroke combination imaginable. However, a Key object can also be configured to change the active Keyboard Page or to shut down the Runtime Keyboard program. You design the visual appearance of a Key object by placing Label, Image and Indicator objects on it, thus allowing you to be as creative as you want when it comes to the look of your Keys.

Designing Keys

FIGURE 54 By placing Label, Image or Indicator objects on a Key you can be as creative as you like.

Before you can place Label, Image or Indicator objects on a Key, you must put the Key in "edit mode". You do this by holding the <*Shift*> key pressed while clicking on the Key. You can now place Label, Image or Indicator objects on the Key by choosing the appropriate tool and clicking on a blank spot inside the Key objects's boundaries. Now, you can click on the new object to set its properties as usual. The Key will remain in edit mode until you click on any object outside it (any object that does not belong to the Key being edited).

In FIGURE 54, the "A" and "©" Keys are designed by placing a Label object in the upper left corner of the Key. The <*Shift*> Key is designed by placing an Image object in the upper left corner and loading the "Shift.bmp" bitmap into it. Finally the <*Caps Lock*> Key first has a Label object with the text "Caps|Lock" and then an Indicator object connected to the <*Caps Lock*> system variable (–1) on the right side. In turn the Indicator object has an Image object on both captions (on and off) where the "Off state" Image loads the "IndOff.bmp" bitmap and the "On state" Image loads the "IndOn.bmp" bitmap.

This demonstrates that you can design Keys which uses other objects as its "caption" and that these objects themselves (being Indicator objects) can in turn hold other Objects to create a "live" Key.

Defining keystrokes

You use the three tool windows Keystrokes Editor, Virtual Key List and Keystroke Preview together to define keystrokes for a Key. Each of these tool windows have their own chapters in section 3, dedicated to explaining how to define keystrokes for a key.

Note

If you use a Key to change the active Keyboard Page or shut down the Runtime Keyboard, it can still send keystrokes. When a Key is pressed in the Runtime Keyboard, all defined keystrokes are sent to the target application before Keyboard Page changes or Runtime Keyboard shutdown is performed.

Key properties

Background

In addition to the background color, the Key object allows you to load a bitmap to use as the background for the Key. The Key Background bitmap works mostly like an Image object, except that the Key Background bitmap is always stretched to fit the entire Key surface. The only manipulation you can do with the Key Background bitmap is to set the Key object's Transparent property on or off.

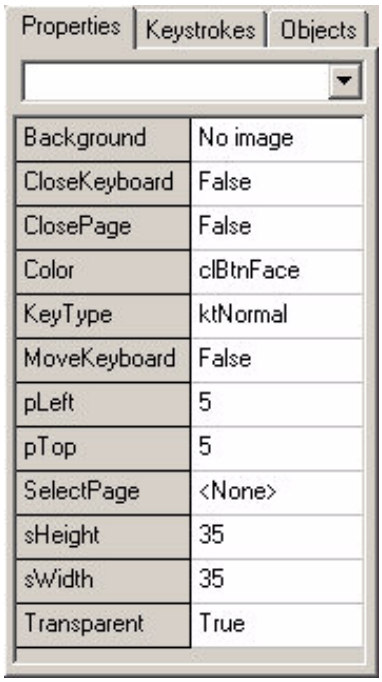

FIGURE 55 The Key objects's properties in the Property Editor.

CloseKeyboard

When you set the CloseKeyboard property to "True", pressing the Key in the Runtime Keyboard will cause the Runtime Keyboard to shut down. In addition, setting CloseKeyboard to "True" will cause the ClosePage and MoveKeyboard properties to be set to "False" and the SelectPage property to be set to "<None>" automatically.

ClosePage

The Runtime Keyboard program keeps track of the last 1000 page changes you have made since the program was started. When you "close" a Keyboard Page, you actually re-activate the previous Keyboard Page that was active. In some cases, this can make more sense than activating a specific Keyboard Page. Setting the ClosePage property to "True", will cause the Runtime Keyboard to perform a "close Keyboard Page" command. In the Keyboard Designer, setting this property to "True" will set the CloseKeyboard and MoveKeyboard properties to "False" and the SelectPage property to "<None>" automatically.

Color The Key object's Color property decides the Key object's background color. You can choose a standard color from the Color property's drop down list box in the Property Editor, or you can select the Key object and then right click on it and choose **Set background color** from the popup menu.

KeyType The KeyType property can be set to either of two values: ktNormal or ktShift. Keys that have their KeyType set to ktShift will not immediately send their keystrokes to the target application, but instead remain in a "pressed" state until a "normal" Key (a Key with its KeyType set to "ktNormal") is pressed. Then the pressed "shift" Key's keystrokes will be sent before and after the "normal" Key's keystrokes. See below for a detailed discussion of this mechanism.

MoveKeyboard When the MoveKeyboard property is set to "True", pressing the key in the Runtime Keyboard causes an immediate change to "Move Anywhere" mode. This allows the user to move the Runtime Keyboard once. See the "Runtime Reference" section for more information. When the MoveKeyboard property is set to "True", the CloseKeyboard and ClosePage properties are set to "False" and the SelectPage property is set to "<None>".

pLeft and pTop The pLeft and pTop properties represent the left and top position of the Key object. Both properties are specified in pixels starting at the left and top edges respectively of the Keyboard Page on which the Key object is placed.

SelectPage If you set the SelectPage property to the name of a Keyboard Page, pressing the Key in the Runtime Keyboard will cause that Keyboard Page to become active. When the SelectPage property is set (not "<None>"), the CloseKeyboard, ClosePage and MoveKeyboard properties are all set to "False".

sHeight and sWidth The sHeight and sWidth properties specify the Key object's height and width in pixels.

Transparent The Key object's Transparent property makes the Key's background bitmap Transparent when set to "True", or opaque when set to "False". For details on bitmap transparency, refer to the Image object's Transparent property.

Managing Keyboard Page changes

For a discussion of Keyboard Pages, refer to the "Understanding Keyboard Pages" chapter.

In earlier versions of the MountFocus Keyboard Designer and Runtime Keyboard you could only change the active Keyboard Page via OLE from another application or by using the system tray popup menu. New in this version is the ability to let Keys change pages when they are pressed in the Runtime Keyboard.

One example of where this functionality could be useful is if you want to make a keyboard that is minimizable. To do this, place a "minimize" Key on every page (design some sensible caption) and create a Keyboard Page that you call "Minimized". Now set the SelectPage property for all your minimize Keys to "Minimized". This will cause the Runtime Keyboard to make the Minimized Keyboard Page active whenever a minimize Key is pressed.

Refer to the "Understanding Regions" chapter for details on Regions and how they work.

On the Minimized page you simply create only one "maximize" Key, which has its ClosePage property set to "True". By using a Region object to limit the visible size of the keyboard, you can make the Minimized Keyboard Page appear to be very small. By using the ClosePage property instead of the SelectPage property, you ensure that pressing the maximize Key will return to whichever Keyboard Page you came from instead of always maximizing to a given Keyboard Page.

A more expanded example could be if you were making a keyboard for a POS application in a fast food restaurant. Instead of placing one Key for each available sales item all on one Keyboard Page, you could make a "Main" page with Keys for "Drinks", "Burgers", "Meals" etc. and make each of those Keys change the active Keyboard Page to one with Keys for "Coffee", "Tea" etc. The "Coffee" Key would send the appropriate keystrokes (item code for coffee) to the POS application and in addition close the active Keyboard Page by setting the ClosePage property to "True".

In such a setup, no item code would ever be more than two "virtual keystrokes" away and at the same time saving a lot of screen real estate.

Working with "shift" Keys

The KeyType property on the Key object allows you to define a Key as "normal" (ktNormal) or "shift" (ktShift). This functionality make it possible to make Keys for <*Shift*>, <*Ctrl*>, <*Alt*> etc. because you can't "hold" one Key while pressing another on a virtual keyboard. Even though you can define any set of keystrokes for a "shift" Key, there are some considerations you must take into account.

When a "shift" Key is pressed in the Runtime Keyboard it sends no keystrokes. Instead the keystrokes are added to an internal list, waiting for a "normal" Key to be pressed. Then the "shift" Key's "key down message" is sent before, and the "key up message" after the "normal" Key's keystrokes.

Let's examine how this works with a <*Ctrl*> Key having its KeyType property set to ktShift. When you press it in the Runtime Keyboard, two keystroke messages are added to the internal list: "<*Ctrl*> down" and "<*Ctrl*> up". Now, if you press the <*C*> Key (which is a "normal" Key), the Runtime Keyboard will first send "<*Ctrl*> down", followed by the "normal" Key's messages, "<*C*> down" and "<*C*> up", and finally the "shift" Key's "<*Ctrl*> up" message. The result is a correctly dispatched <*Ctrl*>+<*C*> keystroke combination.

Pressing multiple "shift" Keys before a "normal" Key will produce the same result. In other words: all "shift" Keys in the internal list are processed both before and after the "normal" Key.

FIGURE 56 Correctly formatted keystrokes for a "shift" Key

When you define the keystrokes for a "shift" Key, make sure all the keystrokes follow a straight line: one keystroke being the **child** of the previous like in FIGURE 56. This is because the Runtime Keyboard sends the **first half** of the keystroke messages **before** and the **last half after** the "normal" Key regardless of the message being a "down" or "up" message.

Chapter 19

Understanding Regions

The Region object can be used to limit the visible portion of a Keyboard Page, thus allowing the Runtime Keyboard to change size and shape when changing the active Keyboard Page.

By using Region objects you can create keyboards that completely change appearance when you change the active Keyboard Page. When no Region objects are present on a Keyboard Page, the visible size of the Runtime Keyboard will be equal to the keyboard size specified by the Keyboard object. However, when one or more Region objects are present on the active Keyboard Page, only the part(s) of the Keyboard Page covered by a Region object will be visible leaving the rest of the keyboard transparent. Other applications will then be visible and accessible in the transparent areas.

Using Regions does not actually change the size of the Runtime Keyboard window. It simply limits the Runtime Keyboard's accessible areas. This becomes apparent if you create a Keyboard Page with one small Region object on it and then try to move the Runtime Keyboard around the edges of the desktop. You will notice that you cannot move the visible part of the Runtime Keyboard all the way to all four sides of the screen. This happens because the Runtime Keyboard does not allow moving outside the visible area of the screen and the size of the Runtime Keyboard has not changed, even though it looks like it has.

Working with Regions

To create a new Region object, right click on the Keyboard Page and choose **New region** from the popup menu, or select **Tools ▸ New region** from the main menu. The new Region object appears in the upper left corner of your Keyboard Page. You can move and size the Region object just like any other object.

When you select a Region object the first time, right click on it and choose **Send to back** from the popup menu. This makes all the other objects on the same Keyboard Page visible on top of the Region. You can still move and size the Region object like before.

As long as a Region object is visible in the Keyboard Designer, you can not add other objects on top of it. This makes working with your design a bit awkward after you have added Region objects. To avoid this problem, add Region objects last, and uncheck the **View ▸ Show regions** main menu item when you are done. This allows you to work with all the other objects on your keyboard without interference from Region objects.

Region properties

pLeft and pTop

The pLeft and pTop properties represent the left and top position of the Region object. Both properties are specified in pixels starting at the left and top edges respectively of the Keyboard Page on which the Region object is placed.

sHeight and sWidth

Similarly, the sHeight and sWidth properties specify the Region object's height and width in pixels.

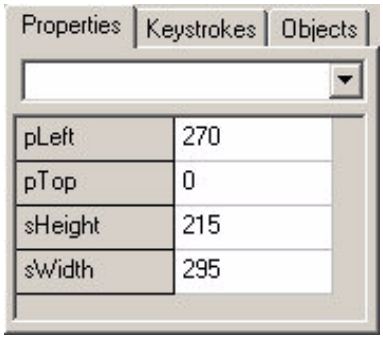

FIGURE 57 The Region object's properties in the Property Editor

Section

5

MOUNTFOCUS INFORMATION SYSTEMS LTD.

Keyboard Designer User's Guide

Runtime Reference

Chapter 20

Starting the Runtime Keyboard

The Runtime Keyboard can be started in many different ways. This chapter describes how, and why you should use different methods.

The Runtime Keyboard can be started from the start menu like other programs, from the command line prompt or via OLE/ActiveX from another program. How you start the Runtime Keyboard depends on how you plan to use it.

Starting from the start menu

The easiest way to start the Runtime Keyboard is, of course, to choose the "Runtime Keyboard" icon from the start menu; just like you start most other programs. Most settings, like position, sound on/off etc. are stored by the Runtime Keyboard so it will load them each time it is started. The start menu shortcut starts the Runtime Keyboard with the –o parameter, which lets you choose the keyboard file you want to work with.

Starting from the command line

You can also start the Runtime Keyboard directly from the command line prompt (or DOS box). The benefit of this method is that you can specify additional **command line parameters** to automatically load a specific keyboard file, set the Runtime Keyboard position, switch sound on or off etc.

You can easily create a shortcut that includes extra parameters. In the Windows Explorer, drag the KBD.EXE (the Runtime Keyboard) file to the directory where you want the shortcut. Then right click on the shortcut and select **Properties** from the popup menu. Under the Shortcut tab you will see the command to start the Runtime Keyboard in the Target edit box. Now you can add your command line parameters after the command. These parameters must be after any closing quotes (") characters.

For details about the available command line parameters, see the "Runtime Keyboard Functions" chapter.

Starting from another application

If you want to start the Runtime Keyboard from your own application, you can opt to use the "quick and dirty" method of just sending the command line as described above to the ShellExecute API function (Run/Start etc. depending on you development tool). While this will start the Runtime Keyboard, it leaves you with no control over the Runtime Keyboard program.

The preferred way is to use OLE or ActiveX. By linking to the Runtime Keyboard using one of these methods, you are able to show, hide, move change the active Keyboard Page and so on, using simple commands in your own programming tool. This means you can force the Runtime Keyboard to change layout (Keyboard Page) depending on the focused window or control in your own application and generally provide a custom made, context sensitive input module, tailor-made for your exact needs.

Details on OLE/ActiveX can be found in the "Using OLE and ActiveX" chapter. There you also find a discussion on the differences between the two methods.

Chapter 21

Deploying the Runtime Keyboard

There are a few things you should keep in mind when you want to deploy keyboards to your users.

As mentioned before in this manual, the keyboard files produced with the Keyboard Designer contains all used bitmaps, sound files and so on. This saves you the trouble of having to keep a list of all supporting files used with your keyboard. The result is that you just need to distribute the Runtime Keyboard executable file, "KBD.EXE" and your keyboard file(s). If you have been using the ActiveX or VCL components, you may have to distribute those component files as well, depending on how your linker has been set up.

Installing the Runtime Keyboard

When you install the Keyboard Designer package, a zipped installation set of the Runtime Keyboard is included. This is the complete Runtime Keyboard installation that can be distributed to your users and installed by running the Setup program. Add your keyboard files in the Runtime Keyboard directory, and you're done.

All installations of the Runtime Keyboard are initially installed as 30-day trials until you enter the license number to make the installation permanent.

Custom install

MountFocus also provides custom install routines. These install programs are made on a per customer basis to suit individual needs. If you plan to distribute a lot of Runtime Keyboard licenses, either internally in your own organization or as an OEM or reseller, let us know and we will be happy to help you.

Licensing

Every installation of the Runtime Keyboard requires its own license number. Licenses can be obtained online from our web site www.mountfocus.com and are available in single license, 10 license, 25 license, 50 license and 100 license packs.

When you order a license pack from our online ordering system, you receive the license keys by email immediately.

Currently the MountFocus Keyboard Designer and Runtime Keyboard programs are sold with a lifetime upgrade. This means that you, and your users, will receive all future upgrades of the software free of charge. Any changes to this policy will only affect licenses sold after the change of policy, so order now – before we change our minds ☺.

Chapter 22

Runtime Keyboard Functions

This chapter aims to give you an overview of all the functions that are available in the Runtime Keyboard, and how to access them from the command line or via OLE or ActiveX.

The Runtime Keyboard has a number of functions that can be set either by command line parameters, via OLE/ActiveX or both. Obviously, command line parameters only allow you to specify options at startup, whereas OLE/ActiveX allows operating on the Runtime Keyboard while it's running.

Runtime Keyboard Functions

In this list OLE is used as a common term for OLE and ActiveX. Properties are specified as "R/W" for "Read and Write" and "RO" for "Read Only". For details about property data types and function parameters, refer to the "Using OLE and ActiveX" chapter.

Command line options are specified as "-option", always using lower case letters. The quotes are not part of the parameter. Parameters that use numeric values expects the value immediately (no spaces) like "-bl180". If opposing (non-compatible) command line options are specified, the last one takes precedence.

ActivePageIndex (OLE R/W property – no command line option)

Sets or gets the active Keyboard Page by numeric index. The first Keyboard Page has number 0, the second Keyboard Page number 1 and so on. The Runtime Keyboard keeps track of all Keyboard Page changes, whether done via OLE or Key object properties. This allows the ClosePage function to backtrack through the same Keyboard Pages.

ActivePageName (OLE R/W property – no command line option)

Works the same way as ActivePageIndex above, except that it works with Keyboard Page names instead of their numeric representation. The Keyboard Page names are not case sensitive.

Blend (OLE R/W property – "–bl<x>" where <x> is the Blend value 0-255)

Blending is only available when the Runtime Keyboard is running on Windows 2000. A Blend value of 0 makes the entire Runtime Keyboard transparent, whereas a Blend value of 255 makes it completely opaque. Values between makes the Runtime Keyboard window more or less transparent, allowing the application behind the keyboard to be visible at the same time as you use the Runtime Keyboard.

ClosePage (OLE function – no command line option)

Calling this function via OLE causes the Runtime Keyboard to backtrack to the previous Keyboard Page. This can be very useful when your application opens a common dialog window that requires its own Keyboard Page. When the window is activated it can select the page it wants, and when deactivated it can call the ClosePage function and thereby reselecting whichever Keyboard Page was displayed previously.

Hide (OLE function – no command line option)

Hide (you guessed it) simply hides the entire Runtime Keyboard.

Height (OLE RO property – no command line option)

This property returns the height in pixels of the full size Runtime Keyboard window. Using regions to hide parts of the active Keyboard Page does not affect this property. The height of the keyboard is set in the Keyboard Designer.

Left (OLE R/W property – "–l<x>" where <x> is the left position in pixels)

The Left property refers to the left side of the Runtime Keyboard window, measured in pixels relative to the left side of your desktop. Certain programs, like the task bar or the Office bar, may change the size of the desktop affecting the placement of the Runtime Keyboard.

LoadKeyboard (OLE function – "–o" or "<filename>")

When using command line options you can load a keyboard file in two ways; the "–o" parameter displays the Open File dialog box and allows you to choose a keyboard file whereas specifying a filename will open that particular file. If you need to specify a filename or path that includes spaces, you can enclose the filename in quotes (").

When loading a keyboard file using the OLE function, make sure you specify the entire path and filename to the keyboard file. This is because you cannot trust the "current directory" to be the same when you start an OLE Automation Server.

MenuOn (OLE R/W property – "–menu" for on or "–nm" for off)

In some cases (especially when using OLE) it may not be desirable to allow the user direct access to the popup menu functions. This option then controls whether or not the system tray icon popup menu is operational or not.

MoveMode (OLE R/W – "–md" for Drag or "–mc" for Click)

Version 2.0 of the Runtime Keyboard allows two different ways for the user to move the Runtime Keyboard: Drag and Click. The Drag mode is what we are used to from the older versions, whereas the Click mode displays a large window covering the entire screen and allows the user to move the keyboard with a single click. This can be appropriate for touch screen environments where dragging across the screen is undesirable. The MoveMode setting only has effect when the Runtime Keyboard is in **"Move Anywhere" mode** (see Position below).

Position (OLE R/W – "–p<x>" where <x> is the position as described below)

The Runtime Keyboard can be moved to five predetermined positions on your desktop, each identified by its own number; Upper Left (1), Upper Right (2), Center (3), Lower Left (4) and Lower Right (5). Additionally placing the Runtime Keyboard in "Move Anywhere" mode (Position 0) lets the user move the Runtime Keyboard once, either by dragging or by clicking (see MoveMode above). Once the Runtime Keyboard is moved it remains stationary until Anywhere mode is entered again.

Version 2.0 also introduces "Movable" mode (Position 6) which allows the user to move the keyboard, by dragging only, at any time.

SetIndicator (OLE function – no command line option)

Available only through OLE, this function allows you to set the state of your user defined indicators. See the "Understanding Indicators" chapter for details on user defined indicators.

SetToTop (OLE function – no command line option)

The Runtime Keyboard stays on top of all other applications. However, if your application also stays on top, you may experience that the Runtime Keyboard ends up behind your application. A call to this function makes sure the Runtime Keyboard appear on top again.

Show (OLE function – no command line option)

A call to this OLE function redisplays the Runtime Keyboard window (if it was hidden – that is).

SoundOn (OLE R/W property – "–sound" for on and "–ns" for off)

Version 2.0 now supports a sound that will be played every time the user presses a key in the Runtime Keyboard. The sound itself is a part of the keyboard file, and while some users may find the sound useful (or cool), others may find it utterly annoying. The SoundOn property allows you to turn sound playing on or off.

Top (OLE R/W property – "–t<x>" where <x> is the top position in pixels)

The Top property refers to the top of the Runtime Keyboard window measured in pixels relative to the top of your desktop. Certain programs, like the task bar or the

Office bar, may change the size of the desktop affecting the placement of the Runtime Keyboard.

TrNumPad (OLE R/W property – "–num" for on or "–nt" for off)

The ten numeric keys on the numeric keypad, plus the decimal key, work in different ways depending on whether or not <*Num Lock*> is active. The Runtime Keyboard detects the <*Num Lock*> state and translates these keys to their correct counterparts if <*Num Lock*> is off. To avoid this translation (sending numbers independent of the <*Num Lock*> state), turn off the TrNumPad property.

Wait (OLE R/W property – "-w<x>" where <x> is the number of milliseconds)

In certain, exceptional cases, the Runtime Keyboard must wait for a short while before sending keystrokes to the target application. If you experience that keystrokes are sent before the target application regains focus, you can set this property to cause the Runtime Keyboard to wait for the specified amount of time. The wait period is set in milliseconds (ms) which is 1/1000 of a second.

Width (OLE RO property – no command line option)

This property returns the width in pixels of the full size Runtime Keyboard window. Using regions to hide parts of the active Keyboard Page does not affect this property. The width of the keyboard is set in the Keyboard Designer.

WinHandle (OLE RO property – no command line option)

This property simply returns the Windows assigned window handle for the Runtime Keyboard window. This can be useful if you want to perform Windows API calls using the Runtime Keyboard window as the target. Naturally, due to the nature of its use, anything you use it for can, and will, be held against you in a support case ☺.

Chapter 23

Using OLE and ActiveX

To gain complete control over the Runtime Keyboard you can use OLE or ActiveX from your own application.

In previous versions of the Runtime Keyboard, only the OLE interface was available. In addition, version 2.0 also includes ActiveX (and Borland VCL) components to make controlling your Runtime Keyboards even easier.

What's the difference?

The Runtime Keyboard program is an **out of process OLE automation server**. This means that the Runtime Keyboard runs in its own memory space, its own thread and acts and operates pretty much like any other stand-alone program. Another effect of this is that you cannot place the Runtime Keyboard as a component or control directly on your own forms. This is done on purpose, because it allows the Runtime Keyboard to send keystrokes to modal dialog boxes as well as other parts of your programs where you cannot easily place a keyboard component.

Through the OLE technology, the Runtime Keyboard exposes a set of properties and functions that other applications can operate on. Using this capability from just about any development tool, is usually pretty easy, and usually poorly documented. In tools like Visual Basic and PowerBuilder it is simply a matter of creating the correct type of variable to hold the reference to the OLE server. Other tools, like Delphi, C++ Builder, Visual C++ etc. allow you to import the Type Library, which describe the OLE functions available, and create a new class that wraps the OLE object's functionality. These tools also include ways to operate directly on the OLE object's Interfaces, which is the "low level" of OLE communication.

VCL or Visual Component Library are special components designed for Borland's Delphi and C++ Builder development tools.

Because using OLE objects directly can involve a bit more research than you may want to do, we have included a set of non-visual ActiveX and VCL "KeyboardCtrl" components that you can use instead. These components simply wrap all the OLE complexities so that you can place the component on your form and access its properties like you do for other components. The KeyboardCtrl component creates one (1) reference to the Runtime Keyboard automation server, which in turn starts one (1) instance of the Runtime Keyboard program. In other words, place the KeyboardCtrl component on one, global form. Using multiple KeyboardCtrl components will cause multiple keyboards to be opened.

Using OLE

OLE is the preferred way of connecting to the Runtime Keyboard. If you know how to do this from your development tool of choice, or you are willing to figure it out, a direct OLE connection leaves fewer layers and thus fewer places where things can go wrong.

The classname for the Runtime Keyboard OLE automation server is "Kbd.mfSoftKeys" (without the quotes). For Visual Basic and PowerBuilder, this is all you need to specify.

C++ and Delphi programmers can also specify this name using the "Type Library Import" (or similar) function in your development environment. This will create a wrapper class that you can instansiate to reference the Runtime Keyboard OLE server. Hardcore OLE experts can also connect directly to the OLE servers "Interface" which is named "ImfSoftkeys".

This psaudocode then describes how you use the OLE connection:

```
// Global declaration
oleobject MyKeyboard // Visual Basic / PowerBuilder
mfSoftkeysClass MyKeyboard // C++ / Delphi etc.

// Application Initialization
Connect MyKeyboard to "Kbd.mfSoftKeys" // VB / PB
Create MyKeyboard // C++ / Delphi etc. - Instansiate object

MyKeyboard.LoadKeyboard( "C:\MyDir\MyFile.kbd" )
MyKeyboard.Position = 6 // Movable mode
MyKeyboard.Left = MainWindow.Left // Align to main form
MyKeyboard.Top = MainWindow.Bottom

// Window activate (for any number of windows)
// If necessary, change the Keyboard Page
MyKeyboard.ActivePageName = This.WindowName

// Window de-activate (for any number of windows)
// If necessary, close the Keyboard Page
MyKeyboard.ClosePage

// Application shutdown
Destroy MyKeyboard
```

As you can see, as soon as you have figured out how to connect to OLE objects or import a Type Library in your development tool, there is not much to it.

Note

When you instansiate an OLE object (connection to the Runtime Keyboard), the Runtime Keyboard program is automatically started. Similarly, the Runtime Keyboard is automatically closed when the object is destroyed.

Using ActiveX and VCL

The ActiveX and VCL "KeyboardCtrl" components gives you an easier way to connect to the Runtime Keyboard. Simply place the KeyboardCtrl component on your main form (or some other global container) and then call the KeyboardCtrl component's properties and functions as you would any other similar object.

The KeyboardCtrl component is non-visual, and contains no design time properties. This means that the component itself will not be visible at runtime and it means you have to write code to access its properties and functions.

When the form that contains the KeyboardCtrl component is created, the Runtime Keyboard is started automatically. Until you call the LoadKeyboard function, the Runtime Keyboard may appear as a blank window. This is because the KeyboardCtrl object is automatically created when the form is created, and then the KeyboardCtrl object creates a link to the Runtime Keyboard OLE server.

If you take a look at the psaudocode for using OLE directly, the only difference is that you don't have to declare a global variable (the KeyboardCtrl component is that variable) and you don't have to explicitly create and destroy the object. The form you placed the KeyboardCtrl component on will do this automatically.

If you want to use the ActiveX version of the KeyboardCtrl component, simply select "mfKeyboardCtrl" from your list of ActiveX (also called OCX) components and place it on your form. Give it a name, and you're up and running.

If you are using Delphi or C++ Builder, you probably want to install the VCL component instead. This involves a couple of steps:

1. Copy the files "mfSoftKeys.bpl", "mfKeyboardCtrl.dcu" and "Kbd_Ctrl.dcu" to your "Lib" directory or any other directory that is in the search path in your development environment.
2. Select **Component ▸ Install Packages** from the menu, press Add, browse to the directory where you placed the files in step 1 and select "mdSoftKeys.bpl".
3. The mfKeyboardCtrl Component is now available in the Component Palette under the "Keyboard" tab.

That's it! You can now fully control any Runtime Keyboard. Have fun!

Properties and functions

Listed below are the properties and functions available using either of the above mentioned methods. This list describes the property datatypes and the function prototypes (parameters etc.), but does not discuss the actual functionality for each property or function. For details about the functionality, see the "Runtime Keyboard Functions" chapter.

Properties (RO = Read Only)

ActivePageIndex (long) Selects the active Keyboard Page (0 to number of Keyboard Pages – 1).

ActivePageName (bstr) Selects the active Keyboard page (any valid Keyboard Page name).

Blend (long) Sets transparency factor (0 to 255).

Height (RO) (long) Returns the Runtime Keyboard window height in pixels.

Left (long) Set the Runtime Keyboard left position (0 to desktop width – keyboard width).

MenuOn (long) Switches on or off the system tray menu (0 for "off" and any other value for "on").

MoveMode (long) Selects how to move the Runtime Keyboard in Anywhere mode (0 for "Drag" and 1 for "Click" – other values map to "Drag").

Position (long) Sets the position of the Runtime Keyboard. Accepts the values listed below. Any other value maps to "Anywhere".

0. Anywhere
1. Upper left corner of desktop
2. Upper right corner of desktop
3. Center of desktop
4. Lower left corner of desktop
5. Lower right corner of desktop
6. Movable

SoundOn (long) Switches sound on or off (0 for "off" and any other value for "on").

Top (long) Sets the Runtime Keyboard top position (0 to desktop height – keyboard height).

TrNumPad (long) Switches on or off translation of numeric keypad keys when <*Num Lock*> is off (0 for "off" and any other value for "on").

Wait (unsigned long) Sets the time in milliseconds to wait before dispatching keystrokes (0 or higher – values over 1000 renders the keyboard more or less useless).

Width (RO) (long) Returns the Runtime Keyboard window width in pixels.

WinHandle (RO) (long) Returns the standard windows handle value.

Functions

ClosePage (no parameters) Closes the current Keyboard Page.

Hide (no parameters) Hides the entire Runtime Keyboard window.

LoadKeyboard (Filename: bstr) Loads the keyboard file specified in the Filename parameter.

SetIndicator (IndicatorNo, State: long) Sets the indicator specified in the IndicatorNo parameter (0 to 250) to "off" if the State parameter is 0 and "on" if the State parameter is non-zero.

SetToTop (no parameters) Forces the Runtime Keyboard window to reset itself to "stay on top", placing it on top of newer windows using the same function.

Show (no parameters) Redisplays the Runtime Keyboard window if it was previously hidden.