MODULE 14: DEVICES

On completion of this module you will be able to use, program and interact with the user via the devices such as a mouse, keyboard, joystick, presenter, screen and barcode scanner.

MODULE 14.1: DEVICES

Subject Outcome 1: Introduction

Subject Outcome 2: Mouse

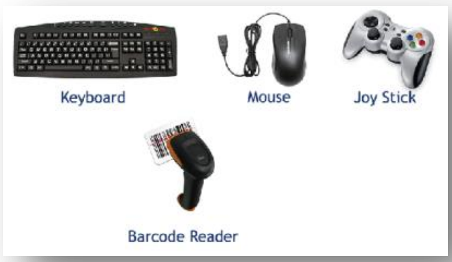
Subject Outcome 3: Keyboard

Subject Outcome 4: JoyStick

Subject Outcome 5: Presenter

Subject Outcome 6: Bar-Code Scanner

Subject Outcome 7: Screen



14.1 INTRODUCTION

Devices are accessories that you plug into your computer that is input (user interacts with computer) or even output drive (ability to alter the capabilities and settings of the accessory). We have the following accessories that we will address and you may use it for the following purposes:

SNO	DEVICE	INPUT DRIVEN	OUTPUT DRIVEN
1	Keyboard	Keys pressed by user. Keypad keyboard.	Keyboard status altered.
2	Mouse	Movement of mouse. Keys pressed of mouse.	Cursor may be placed at coordinates without mouse movement.
3	JoyStick	Buttons pressed by user.	Reprogramming of keys to calibrate joystick.
4	BarCode scanner	Scans as received by scanner is a keyboard entry with hard enter.	
5	Presenter	Buttons pressed by user.	Button codes may be reprogrammed for usages by the program.
6	USB Stick/Memory Stick	Auto play function by stick. Disk storage devices – from stick to computer. USB device ID.	Disk storage devices – info from program to stick. Eject or unauthorized control.
7	Screen	Resolution Duel screen detection	Recoding of resolution
8	Optical Drives	Drive used as storage device.	Eject the drive or play/stop/ff/rewind functions.
9	Com Port	Entries received from communication devices (scans/entries).	Actions to be executed by the COM device (eject drawer, send electrical power to activate, etc.)

14.2 MOUSE

We have already addressed the Mouse's command directives during Module 12, so we will only indicate the main device directives.

ACTION	CODING	EXPLANATION
Screen.Cursor	$TYPECHECK ON $INCLUDE <RapidQ2.inc> $include <qcursor.inc> $resource cur1 as "c:\ms3\mybis\gx\cur3d.cur" dim curme as qcursor curme.loadfromresource(0) create testme as qformex cursor=1	Alter the actual mouse cursor (CUR file) as the file being identified (remember the CURSOR=1 starts from entry 1 and not 0 (zero).

ACTION	CODING	EXPLANATION
	end create testme.showmodal	
Screen.MouseButtons	$TYPECHECK ON $include <rapidq2.inc> showmessage str$(screen.mousebuttons)	Determine how many buttons on the connected mouse.
Screen.MousePresent	$TYPECHECK ON $include <rapidq2.inc> showmessage str$(screen.mousepresent)	Determine if a mouse has been plugged in. If the value is 0 then no mouse, if 1 then mouse is present (includes laptop's touchpad).
Screen.MouseSwap	$TYPECHECK ON $include <rapidq2.inc> showmessage str$(screen.mouseswap)	Report if user has swapped the left and right mouse buttons (left handed people). If 0 then not swapped, if 1 then it has been swapped.
(QSystem).MouseWheelPresent	$TYPECHECK ON $include <rapidq2.inc> dim systm as qsystem showmessage str$(systm.mousewheelpresent)	Determine if the mouse wheel is present.
?.Move(x%,y%)	$TYPECHECK ON $include <rapidq2.inc> $include <qcursor.inc> dim curme as qcursor curme.move(50,50)	This will place the mouse cursor at the indicated position. The X and Y coordinates is relative to the screen and not the window.
?.visible	$TYPECHECK ON $include <rapidq2.inc> $include <qcursor.inc> dim curme as qcursor curme.visible=0 showmessage "invisible" curme.visible=1	Visible=0 will turn the cursor off (invisible) and Visible=1 will make the cursor appear again. The cursor is only invisible over the program's form/window.
MOUSEX MOUSEY	$TYPECHECK ON $include <rapidq2.inc> showmessage "Pos: "+str$(screen.mousex)+" Y: "+str$(screen.mousey)	This will determine the position of the mouse's cursor relative to the entire screen (desktop
SetMouseXY(x%,y%)	$TYPECHECK ON $include <rapidq2.inc> screen.setmousexy(50,50)	This will set the position of the mouse relative to the entire desktop.

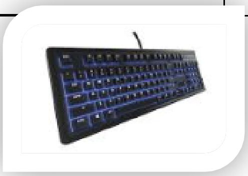
OnMouseWheel(r%,x%,y%,shift%)	$TYPECHECK ON
	$INCLUDE <RapidQ2.inc>
	declare sub dwheel(rotation%,x%,y%,shift%)
	CREATE Form AS QFORMex
	    Caption = "Form"
	    Width = 640:Height = 480:Center
	    onmousewheel=dwheel
	END CREATE
	SetWindowLong(Form.Handle, -8, 0)
	SetWindowLong(Application.Handle, -8, Form.Handle)
	Form.ShowModal
	
	sub dwheel(rotation%,x%,y%,shift%)
	form.caption=Str$(rotation%)+" "+str$(x%)+"/"+str$(y%)
	end sub
		This will enable an event on the turn of the mouse's wheel.

ACTION	CODING	EXPLANATION

ONMOUSEDOWN=sub$
ONMOUSEMOVE=sub$
ONMOUSEUP=sub$
OnClick=sub$
OnDblClick=sub$
	99% of all elements (QEdit, QLabel, etc.) have the ability to enable mouse trapping (maybe not all the events as indicate in the cell to the left, however some of them.
	
	Remember that they all are to call a sub program when the user does the action relative to the element. Each event has addition information as well such as ONMOUSEDOWN(button%, x%,y%, shift%) whereby BUTTON% determines which button was pressed, X position, Y position and did the user add the SHIFT/CTRL or ALT key when the button was pressed?
	
	See each element to review the possible mouse events applicable.	

ONLOSTFOCUS=sub$
ONGETFOCUS=sub$
	These methods will monitor if the status of the activity related to the element in question. If you click with the mouse on a different window, the focus is lost at the current form towards the newly selected form.
	
	The events and monitoring methods should be coded for each element/form.	

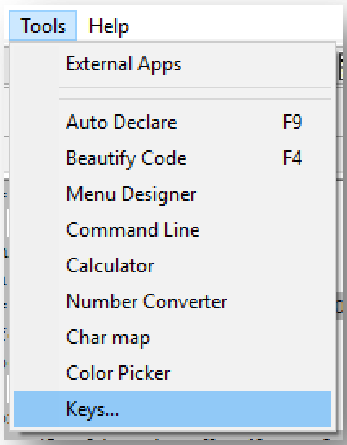
CODING	EXPLANATION
`$TYPECHECK ON`	
`$INCLUDE <RapidQ2.inc>`	
`declare sub losta`	Event sub program when lost.
`declare sub geta`	Event sub program when active.
`declare sub show`	
`CREATE Form AS QFORMex`	
`    Caption = "Form"`	
`    Width = 300:Height = 200:left=20`	
`    onshow=show`	
`END CREATE`	
`CREATE Form2 AS QFORMex`	
`    Caption = "Form2"`	
`    Width = 300:Height = 200:left=400`	
`    onlostfocus=losta:ongetfocus=geta`	If lost then call LOSTA; if focus received then call GETA.
`END CREATE`	
`SetWindowLong(Form.Handle, -8, 0)`	
`SetWindowLong(Application.Handle, -8, Form.Handle)`	
`Form.ShowModal`	
`sub show`	
`form2.visible=1`	
`end sub`	
`sub losta`	
`    form2.caption="bye"`	
`    form.caption="Hallo"`	
`end sub`	
`sub geta`	
`    form.caption="bye"`	
`    form2.caption="Hallo"`	
`end sub`	

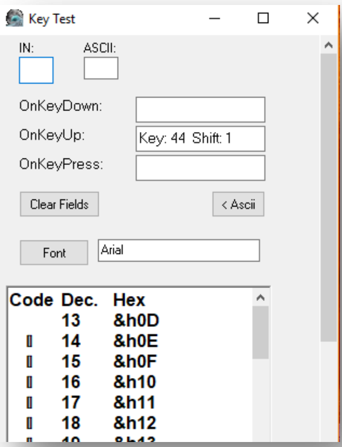


14.3 KEYBOARD

The user has to communicate with your program or enter information into your program. This is done either by a keyboard or mouse. The keyboard is the main method of communication with your computer/program.

All keys on the keyboard are registered per key as KEY SCAN CODES. The value of the key (pressed) is a number and to be able to display the actual key, you need CHR$() to convert it. To see the codes of all the keys; open **FREEQ**; **main** menu **TOOLS**; **KEYS** … simply press the key within the **IN** edit field to see the code.





OnKeyUp: Key: 44 Shift: 1

Font: Arial

```
Code Dec.  Hex
      13   &h0D
  ▯   14   &h0E
  ▯   15   &h0F
  ▯   16   &h10
  ▯   17   &h11
  ▯   18   &h12
```

You have the following key event handles (call a sub-program when the user presses a key – remember that each event must be coded within the element's event handler – ensure that the element may use the event handler).

ACTION	CODING	EXPLANATION

OnKeyPress=Sub$	$TYPECHECK ON
	$INCLUDE <RapidQ2.inc>
	declare sub confirmkey(key as byte)
	
	CREATE Form AS QFORMex
	  Caption = "Form"
	  Width = 300:Height = 200:left=20
	  onkeypress=confirmkey
	  create remark as qlabel
	    left=10:top=20:caption="(key)"
	  end create
	END CREATE
	
	SetWindowLong(Form.Handle, -8, 0)
	SetWindowLong(Application.Handle, -8, Form.Handle)
	Form.ShowModal
	
	sub confirmkey(key as byte)
	remark.caption="Code: "+str$(key)+" … Key: "+chr$(key)
	end sub
		This will monitor any key presses as long as the Window has the FOCUS (active). Note that special keys (HOME, CTRL, etc.) and arrow keys don't work with this event trapper.

If key=n% then	$TYPECHECK ON
	$INCLUDE <RapidQ2.inc>
	declare sub confirmkey(key as byte)
	
	CREATE Form AS QFORMex
	  Caption = "Form"
	  Width = 300:Height = 200:left=20
	  onkeypress=confirmkey
	  create remark as qlabel
	    left=10:top=20:caption="(key)"
	  end create
	END CREATE
	
	SetWindowLong(Form.Handle, -8, 0)
	SetWindowLong(Application.Handle, -8, Form.Handle)
	Form.ShowModal
		React to specific key presses. This will monitor and act on specific key presses.

ACTION	CODING	EXPLANATION

	sub confirmkey(key as byte)
	remark.caption="Code: "+str$(key)+" ... Key: "+chr$(key)
	if key=27 then showmessage "ESC"
	if key=13 then showmessage "ENTER"
	if key=32 then showmessage "SPACE"
	if key=8 then showmessage "Back Space"
	end sub	

Main Menu HotKeys	$TYPECHECK ON
	$INCLUDE <RapidQ2.inc>
	declare sub confirmkey(key as byte)
	
	CREATE Form AS QFORMex
	  Caption = "Form"
	  Width = 300:Height = 200:left=20
	  onkeypress=confirmkey
	  create remark as qlabel
	    left=10:top=20:caption="(key)"
	  end create
	  create mainmenu as qmainmenu
	    create menua as qmenuitem
	      caption="&File"
	    end create
	    create menub as qmenuitem
	      caption="E&dit"
	      create menuba as qmenuitem
	        caption="&Test"
	      end create
	    end create
	  end create
	END CREATE
	
	SetWindowLong(Form.Handle, -8, 0)
	SetWindowLong(Application.Handle, -8, Form.Handle)
	Form.ShowModal
	
	sub confirmkey(key as byte)
	remark.caption="Code: "+str$(key)+" ... Key: "+chr$(key)
	end sub
		This will activate the Main Menu hotkeys (when you press ALT and the short-cut key as pre-programmed). To pre-program it, simply place the & character in front of the letter that you wish to use as the short-cut in conjunction with the ALT key.

OnkeyDown=Sub$	$TYPECHECK ON
	$INCLUDE <RapidQ2.inc>
	declare sub confirmkey(key as word,shift as integer)
	
	CREATE Form AS QFORMex
	  Caption = "Form"
	  Width = 300:Height = 200:left=20
	  onkeydown=confirmkey
	  create remark as qlabel
	    left=10:top=20:caption="(key)"
	  end create
	END CREATE
	SetWindowLong(Form.Handle, -8, 0)
	SetWindowLong(Application.Handle, -8, Form.Handle)
	Form.ShowModal
	
	sub confirmkey(key as word,shift as integer)
	remark.caption="Code: "+str$(key)+" ... Key: "+chr$(key)+": "+str$(shift)
	end sub
		This will trap a key when the user presses the key down (phase before the key is released). In addition you may give depth to the keys being pressed by combining the ALT/CTRL and SHIFT key.
		
		This event handler will also be able to detect the arrow keys being pressed.
		
		The values of the SHIFT responses:
		• Alt – 16
		• CTRL – 1
		• Windows – 91 (key)

OnKeyUp=Sub$	... Onkeyup=confirmkey ...	**This key trapper is when the user releases the key that was pressed.** The coding is the same as ONKEYDOWN.

ACTION	CODING	EXPLANATION

Key=13	`$TYPECHECK ON`
	`$INCLUDE <RapidQ2.inc>`
	
	`declare sub confirmkey(key as word,shift as integer)`
	
	`CREATE Form AS QFORMex`
	`   Caption = "Form"`
	`   Width = 300:Height = 200:left=20`
	`   create remark as qedit`
	`      left=10:top=20:text="(key)"`
	`      onkeyup=confirmkey`
	`   end create`
	`END CREATE`
	
	`SetWindowLong(Form.Handle, -8, 0)`
	`SetWindowLong(Application.Handle, -8, Form.Handle)`
	`Form.ShowModal`
	
	`sub confirmkey(key as word,shift as integer)`
	`if key=13 then`
	`   showmessage remark.text`
	`end if`
	`end sub`
		To trap the ENTER key.
		This function is useful to see if the user is finished entering whatever data and then uses the **ENTER** key to start the next process.
		
		Enter a **name** and **press ENTER**.

If shift=n% and key=n%	`$TYPECHECK ON`
	`$INCLUDE <RapidQ2.inc>`
	
	`declare sub confirmkey(key as word,shift as integer)`
	
	`CREATE Form AS QFORMex`
	`   Caption = "Form"`
	`   Width = 300:Height = 200:left=20`
	`   onkeyup=confirmkey`
	`   create remark as qlabel`
	`      left=10:top=20:caption="(key)"`
	`   end create`
	`END CREATE`
	
	`SetWindowLong(Form.Handle, -8, 0)`
	`SetWindowLong(Application.Handle, -8, Form.Handle)`
	`Form.ShowModal`
	
	`sub confirmkey(key as word,shift as integer)`
	`if shift=1 and key=65 then remark.caption="CTRL && A"`
	`if shift=16 and key=65 then remark.caption="ALT && A"`
	`if shift=256 and key=65 then remark.caption="SHIFT && A"`
	`end sub`
		Trapping SHIFT, ALT and CTRL special keys combined with standard character keys.
		
		You need to type a double && to display this character because of special key event trapping.

If key=40
If key=38
If key=39
If key=37
	`$TYPECHECK ON`
	`$INCLUDE <RapidQ2.inc>`
	
	`declare sub confirmkey(key as word,shift as integer)`
	
	`CREATE Form AS QFORMex`
	`   Caption = "Form"`
	`   Width = 300:Height = 200:center`
	`   onkeyup=confirmkey`
	`END CREATE`
	
	`SetWindowLong(Form.Handle, -8, 0)`
	`SetWindowLong(Application.Handle, -8, Form.Handle)`
	`Form.ShowModal`
	
	`sub confirmkey(key as word,shift as integer)`
	`if key=40 then form.top=form.top+5`
	`if key=38 then form.top=form.top-5`
	`if key=37 then form.left=form.left-5`
	`if key=39 then form.left=form.left+5`
	`end sub`
		Trapping arrow keys.
		Press the arrow keys and see the form move. Take note that you must press and release, don't keep the key in.
		
		Down arrow
		Up arrow
		Left arrow
		Right arrow.

ACTION	CODING	EXPLANATION

KeyPreview=0
KeyPreview=1
	`$TYPECHECK ON`
	`$INCLUDE <RapidQ2.inc>`
	`declare sub confirmkey(key as word,shift as integer)`
	`CREATE Form AS QFORMex`
	`   Caption = "TEST: "`
	`   keypreview=1:onkeyup=confirmkey`
	`   Width = 300:Height = 200:center`
	`   create edita as qedit`
	`   end create`
	`   create editb as qedit`
	`      top=50`
	`   end create`
	`END CREATE`
	`SetWindowLong(Form.Handle, -8, 0)`
	`SetWindowLong(Application.Handle, -8, Form.Handle)`
	`Form.ShowModal`
	
	`sub confirmkey(key as word,shift as integer)`
	`form.caption=form.caption+chr$(key)`
	`end sub`
		Trapping all keys even though focus is not on element. Keys are only trapped (detected) if the element's ONKEY(EVENT) has focus. With this capability all keys being trapped is trapped via the FORM and not the element. You will need to determine which element was meant to trap however to have the correct response.
		
		=1 will enable this function. It does not matter if you have no ONKEYUP/DOWN event trappers – as long as there is only one with the QFORM. You may type on any element, it will register via the FORM and not that element.

14.4 JOYSTICK

A joystick is a device used to play games or use as a controller for control application. Joysticks are however not that simple to code as each type of joystick (model and make) differs and have different amount and types of buttons. This is why all games capable of using joysticks have default settings and custom calibration settings whereby the user must click to confirm the buttons with which action.

The normal UP, DOWN, LEFT and RIGHT are standard settings however. It is the action buttons that causes the complication. The buttons are any of the following codes:

- 1, 2, 4, 8, 16, 32, 64, 128, 256 …

You may call up the controller settings via your Window's System:

```
$TYPECHECK ON
$Include <rapidq2.inc>
defint a
dim systm as qsystem
systm.showcontrolpanel("game",1)
```

As you can see the value increases by the ^2 (power of 2). This is why it is important to be able to configure your joystick before starting to play the game (etc.) With the following example we will detect the directions and buttons pressed:

CODING	EXPLANATION
$TYPECHECK ON	
$INCLUDE <RapidQ2.inc>	
declare sub checkjoy	Call sub program.
Const JOYSTICKID1 = 0	All of the following coding is used to configure the joystick within your FREEQ program.
Const JOYSTICKID2 = 1	
Const JOY_POVCENTERED = -1	
Const JOY_POVFORWARD = 0	
Const JOY_POVRIGHT = 9000	
Const JOY_POVLEFT = 27000	
Const JOY_RETURNX = &H1	
Const JOY_RETURNY = &H2	
Const JOY_RETURNZ = &H4	
Const JOY_RETURNR = &H8	
Const JOY_RETURNU = &H10	
Const JOY_RETURNV = &H20	
Const JOY_RETURNPOV = &H40	
Const JOY_RETURNBUTTONS = &H80	
Const JOY_RETURNRAWDATA = &H100	
Const JOY_RETURNPOVCTS = &H200	
Const JOY_RETURNCENTERED = &H400	
Const JOY_USEDEADZONE = &H800	
Const JOY_RETURNALL = JOY_RETURNX Or JOY_RETURNY Or JOY_RETURNZ Or JOY_RETURNR Or	
JOY_RETURNU Or JOY_RETURNV Or JOY_RETURNPOV Or JOY_RETURNBUTTONS	
Const JOY_CAL_READALWAYS = &H10000	
Const JOY_CAL_READRONLY = &H2000000	
Const JOY_CAL_READ3 = &H40000	
Const JOY_CAL_READ4 = &H80000	
Const JOY_CAL_READXONLY = &H100000	
Const JOY_CAL_READYONLY = &H200000	
Const JOY_CAL_READ5 = &H400000	
Const JOY_CAL_READ6 = &H800000	
Const JOY_CAL_READZONLY = &H1000000	
Const JOY_CAL_READUONLY = &H4000000	
Const JOY_CAL_READVONLY = &H8000000	
Const JOY_OFFSET = 10000	
TYPE TJOYINFOEX	
dwSize AS LONG	
dwFlags AS LONG	
dwXpos AS LONG	
dwYpos AS LONG	
dwZpos AS LONG	
dwRpos AS LONG	
dwUpos AS LONG	
dwVpos AS LONG	
dwButtons AS LONG	
dwButtonNubmer AS LONG	
dwPOV AS LONG	
dwReserved1 AS LONG	
dwReserved2 AS LONG	
END TYPE	
DECLARE FUNCTION JoyGetPosEx LIB "WINMM" ALIAS "joyGetPosEx" _	
(uJoyID AS LONG, JoyInfo AS TJOYINFOEX) AS LONG	
DECLARE FUNCTION JoyReleaseCapture LIB "WINMM" ALIAS "joyReleaseCapture" _	
(uJoyID AS LONG) AS LONG	
DECLARE FUNCTION JoySetCapture LIB "WINMM" ALIAS "joySetCapture" _	
(hWnd AS LONG, uJoyID AS LONG, uPeriod AS LONG, fChanged AS LONG) AS LONG	
DIM JoyInfoEx AS TJOYINFOEX	
DIM JoyX AS LONG, JoyY AS LONG	
JoySetCapture(Application.Handle, JOYSTICKID1, 1, 0)	
JoyReleaseCapture(JOYSTICKID1)	
JoyInfoEx.dwSize = 64	
JoyInfoEx.dwFlags = JOY_RETURNALL	
JoyInfoEx.dwSize = 64	

CODING	EXPLANATION
JoyInfoEx.dwFlags = JOY_RETURNALL	
JoyGetPosEx(JOYSTICKID1, JoyInfoEx)	
JoyX = JoyInfoEx.dwXpos	
JoyY = JoyInfoEx.dwYpos	
dim timer1 as qtimer:timer1.enabled=0:timer1.interval=10	Establish a timer that will confirm any joystick activity.
timer1.ontimer=checkjoy	
CREATE Form AS QFORMex	
Caption = "Form"	
Width = 300:Height = 200:left=20	
create remark as qlistbox	The QList will be used to display the results should the joystick be used.
left=10:top=20	
end create	
END CREATE	
SetWindowLong(Form.Handle, -8, 0)	
SetWindowLong(Application.Handle, -8, Form.Handle)	
timer1.enabled=1	**Start the timer event handler (start checking for joystick activity.**
Form.ShowModal	
sub checkjoy	
JoyGetPosEx(JOYSTICKID1, JoyInfoEx)	
IF JoyInfoEx.dwXpos > JoyX+JOY_OFFSET THEN	If joystick right dir.
remark.additems "Right"	
end if	
IF JoyInfoEx.dwXpos < JoyX-JOY_OFFSET THEN	
remark.additems "Left"	If joystick left dir.
end if	
IF JoyInfoEx.dwYpos > JoyY+JOY_OFFSET THEN	
remark.additems "Down"	If joystick down dir.
end if	
IF JoyInfoEx.dwYpos < JoyY-JOY_OFFSET THEN	If joystick up dir.
remark.additems "Up"	
end if	
IF JoyInfoEx.dwButtons > - THEN	If any other button is pressed, reveal code.
remark.additems "Button "+ str$(JoyInfoEx.dwButtons)+ " pressed."	
end if	**Display the last item within the listbox.**
remark.itemindex=remark.itemcount-1	
end sub	

Now let's add some edit fields to record the code for each button (custom configuration – calibration).

CODING	EXPLANATION
$TYPECHECK ON	**You need select one of the two edit fields, then press the button on the joystick applicable for that action.**
$INCLUDE <RapidQ2.inc>	
declare sub checkjoy	
Const JOYSTICKID1 = 0	
Const JOYSTICKID2 = 1	
Const JOY_POVCENTERED = -1	
Const JOY_POVFORWARD = 0	
Const JOY_POVRIGHT = 9000	
Const JOY_POVLEFT = 27000	
Const JOY_RETURNX = &H1	
Const JOY_RETURNY = &H2	
Const JOY_RETURNZ = &H4	
Const JOY_RETURNR = &H8	
Const JOY_RETURNU = &H10	
Const JOY_RETURNV = &H20	
Const JOY_RETURNPOV = &H40	
Const JOY_RETURNBUTTONS = &H80	
Const JOY_RETURNRAWDATA = &H100	
Const JOY_RETURNPOVCTS = &H200	
Const JOY_RETURNCENTERED = &H400	
Const JOY_USEDEADZONE = &H800	
Const JOY_RETURNALL = JOY_RETURNX Or JOY_RETURNY Or JOY_RETURNZ Or JOY_RETURNR Or JOY_RETURNU Or JOY_RETURNV Or JOY_RETURNPOV Or JOY_RETURNBUTTONS	

CODING	EXPLANATION

```
Const JOY_CAL_READALWAYS = &H10000
Const JOY_CAL_READRONLY = &H2000000
Const JOY_CAL_READ3 = &H40000
Const JOY_CAL_READ4 = &H80000
Const JOY_CAL_READXONLY = &H100000
Const JOY_CAL_READYONLY = &H200000
Const JOY_CAL_READ5 = &H400000
Const JOY_CAL_READ6 = &H800000
Const JOY_CAL_READZONLY = &H1000000
Const JOY_CAL_READUONLY = &H4000000
Const JOY_CAL_READVONLY = &H8000000
Const JOY_OFFSET = 10000

TYPE TJOYINFOEX
   dwSize AS LONG
   dwFlags AS LONG
   dwXpos AS LONG
   dwYpos AS LONG
   dwZpos AS LONG
   dwRpos AS LONG
   dwUpos AS LONG
   dwVpos AS LONG
   dwButtons AS LONG
   dwButtonNubmer AS LONG
   dwPOV AS LONG
   dwReserved1 AS LONG
   dwReserved2 AS LONG
END TYPE
DECLARE FUNCTION JoyGetPosEx LIB "WINMM" ALIAS "joyGetPosEx" _
            (uJoyID AS LONG, JoyInfo AS TJOYINFOEX) AS LONG
DECLARE FUNCTION JoyReleaseCapture LIB "WINMM" ALIAS "joyReleaseCapture" _
            (uJoyID AS LONG) AS LONG
DECLARE FUNCTION JoySetCapture LIB "WINMM" ALIAS "joySetCapture" _
            (hWnd AS LONG, uJoyID AS LONG, uPeriod AS LONG, fChanged AS LONG) AS LONG

DIM JoyInfoEx AS TJOYINFOEX
DIM JoyX AS LONG, JoyY AS LONG
JoySetCapture(Application.Handle, JOYSTICKID1, 1, 0)
JoyReleaseCapture(JOYSTICKID1)
JoyInfoEx.dwSize = 64

JoyInfoEx.dwFlags = JOY_RETURNALL
JoyGetPosEx(JOYSTICKID1, JoyInfoEx)
JoyX = JoyInfoEx.dwXpos
JoyY = JoyInfoEx.dwYpos
dim timer1 as qtimer:timer1.enabled=0:timer1.interval=10
timer1.ontimer=checkjoy
CREATE Form AS QFORMex
   Caption = "Form"
   Width = 500:Height = 200:left=20
   create remark as qlistbox
      left=10:top=20
   end create
   create edt1 as qedit
      left=200:top=48:showhint=1
      hint="kick button"
   end create
   create edt2 as qedit
      left=200:top=78:showhint=1
      hint="jump button"
   end create
END CREATE
SetWindowLong(Form.Handle, -8, 0)
SetWindowLong(Application.Handle, -8, Form.Handle)
timer1.enabled=1
Form.ShowModal
```

CODING	EXPLANATION

sub checkjoy
  JoyGetPosEx(JOYSTICKID1, JoyInfoEx)
  IF JoyInfoEx.dwXpos > JoyX+JOY_OFFSET THEN
    remark.additems "Right"
  end if
  IF JoyInfoEx.dwXpos < JoyX-JOY_OFFSET THEN
    remark.additems "Left"
  end if
  IF JoyInfoEx.dwYpos > JoyY+JOY_OFFSET THEN
    remark.additems "Down"
  end if
  IF JoyInfoEx.dwYpos < JoyY-JOY_OFFSET THEN
    remark.additems "Up"
  end if
  IF JoyInfoEx.dwButtons > - THEN
    IF edt1.Handle = GETFOCUS () THEN
      edt1.text=str$(JoyInfoEx.dwButtons)
      setfocus(remark.handle)
      goto skiprest
    end if
    IF edt2.Handle = GETFOCUS () THEN
      edt2.text=str$(JoyInfoEx.dwButtons)
      setfocus(remark.handle)
      goto skiprest
    end if
    remark.additems "Button "+ str$(JoyInfoEx.dwButtons)+ " pressed."
  end if
  skiprest:
  remark.itemindex=remark.itemcount-1
end sub
	If cursor is active in EDIT field 1, then add joystick value, then exit this sub program
	
	If the cursor is active in EDIT field 2 then add the result of the joystick button press to this EDIT field.

14.5 PRESENTER

A presenter is used to work with presentation software such as MS PowerPoint ®. This is however the limit to users, for a programmer this is only another programming tool that may be incorporated into your program. This device is just another communication tool with codes as you press the buttons. Therefore you can program this device to work with your software. Just like the joystick however, you need to configure the key codes as each model and make differs.

$TYPECHECK ON
$INCLUDE <RapidQ2.inc>
declare sub confirmkey(key as word,shift as integer)
 	
CREATE Form AS QFORMex
  Caption = "Form"
  Width = 300:Height = 200:left=20
  onkeydown=confirmkey
  create remark as qlabel
    left=10:top=20:caption="(key)"
  end create
END CREATE
SetWindowLong(Form.Handle, -8, 0)
SetWindowLong(Application.Handle, -8, Form.Handle)
Form.ShowModal
 	
sub confirmkey(key as word,shift as integer)
remark.caption="Code: "+str$(key)+" ... Key: "+chr$(key)+": "+str$(shift)
end sub
	Use this program to detect and indicate the values of the keys. Some of the button codes will include the SHIFT value.
	
	If you wish to use this tool within your program, then you will need to record the values each button (probably save it for auto load in future). When we address the SOUND & MUSIC module, then we will code a program using the presenter as a remote for the music player.

14.6 BARCODE SCANNER

A barcode scanner is nothing but a laser light that reads lines and retype them in sensible characters (numbers and letters), and then when reading the last digit, presses a HARD ENTER. So it is actually a smart keyboard. Bar code scanners are used to scan a barcode and then search a database quickly to display or address which ever product it is (stock counting, verification, ID's, etc.) The reasons to use bar codes are endless.

<u>You get two types of scans</u>:

- Bar Code

- QR Scans.

We will be working with standard bar codes. A bar code number is devided into different parts:

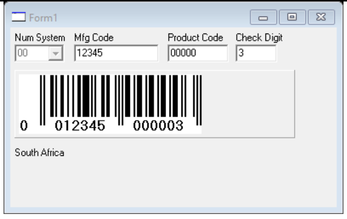
123 1234 12345 1

- Num System (Country of origin) – South Africa being 600 & 601 *user setting
- MFG Code (for instance your company code) *user setting
- Product Code * user setting
- Check Digit * system setting – this digit is determined bmo mathematics to determine the end of the bar code.

This lesson will have two parts, part 1 how to create a bar code (you own) and the second, how to read the bar code. 95% of all products have bar codes, therefore you simply scan them in, however you may create your own bar code as long as you only alter the *user settings.

14.6.1 Create a Barcode

The barcode system used is EAN-13. See http://www.barcodeisland.com for other formats (also those including text characters).

CODING	EXPLANATION
$TYPECHECK ON $INCLUDE "RAPIDQ2.INC" dim i as single DECLARE SUB ComboBox1Change (Sender AS QCOMBOBOX) DECLARE SUB Edit2KeyDown (Key AS WORD, Shift AS LONG, Sender AS QEDIT) DECLARE SUB Edit3KeyDown (Key AS WORD, Shift AS LONG, Sender AS QEDIT) DECLARE SUB SetFocus LIB "USER32" ALIAS "SetFocus" (HWnd AS LONG) DECLARE FUNCTION CODE2OF5(tekst AS STRING) AS STRING DECLARE SUB makebmp (barcode AS STRING,tekst AS STRING,BLineWidth AS INTEGER,filename AS STRING) DIM barcod AS STRING DIM bcodel(0 TO 13) AS STRING*8 DIM bcoder(0 TO 13) AS STRING*8 DIM bcodea(0 TO 13) AS STRING*8 DIM bcodep(0 TO 13) AS STRING*8 DIM Sysltem$(200) AS STRING DIM BITMAP AS STRING DIM TEKSTX AS STRING DIM x AS INTEGER DIM barlengte AS INTEGER DIM LANG AS INTEGER DIM WEIGHT AS INTEGER DIM CHK AS INTEGER DIM CHECK AS INTEGER DIM barcode AS STRING DIM tekst AS STRING DIM BLineWidth AS INTEGER DIM filename AS STRING DIM bmp AS QBitmap DIM breedte AS INTEGER DIM BCount AS INTEGER DIM BXPOS AS INTEGER DIM TEKSTPOS AS INTEGER DIM Bcolor AS INTEGER DIM NumberSistemPri$ AS STRING DIM NumberSistemSec$ AS STRING DIM File AS QFileStream BLineWidth=2 filename=curdir$+"\gx\2OF5.bmp" CREATE Form AS QFORM Caption = "BARCODER" Width = 400:Height = 250:Center CREATE Label1 AS QLABEL Caption = "Num System":Left = 5:Top = 5:Width = 72 END CREATE CREATE Label2 AS QLABEL Caption = "Mfg Code" Left = 75:Top = 5 Width = 104:Transparent = 1 END CREATE CREATE Label3 AS QLABEL Caption = "Product Code" Left = 185:Top = 5:Width = 80:Transparent = 1 END CREATE CREATE Label4 AS QLABEL Caption = "Check Digit":Left = 265:Top = 5:Width = 72 END CREATE	 Form1 Num System: 00 · Mfg Code: 12345 · Product Code: 00000 · Check Digit: 3 0 012345 000003 South Africa For this program to work you need to ensure that the **2oF5.bmp** file is present within the application's directory/path. Simply enter the numbers and press ENTER at the last character within the Product Code. You must copy the source code as is into your program. All the coding within yellow highlight may be copied as is into your program – ensure no double elements names that could cause the system to crash.

CREATE ComboBox1 AS QCOMBOBOX
    Text = "00":Left = 5
    Top = 20:Width = 59:enabled=0
  END CREATE
	The Num System holder.

CODING	EXPLANATION
`  CREATE Edit2 AS QEDIT`	The MFG code.
`    Text = "":Left = 75:Top = 20`	
`    Width = 100:MaxLength=5`	
`    OnKeyDown = Edit2KeyDown`	
`  END CREATE`	
`  CREATE Edit3 AS QEDIT`	The product code.
`    Text = "":Left = 185`	
`    Top = 20:Width = 72`	
`    MaxLength=5:OnKeyDown = Edit3KeyDown`	
`  END CREATE`	
`  CREATE Edit4 AS QEDIT`	The digital check.
`    Text = "":Left = 265:Top = 20:Width = 49`	
`    MaxLength=1:enabled=0`	
`  END CREATE`	
`  CREATE Panel1 AS QPANEL`	
`    Left = 5:Top = 50:Caption = ""`	
`    Width = 330:Height = 80`	
`    CREATE Image AS QImage`	
`      Autosize=0:Top = 5:Left =5`	Preview of the actual bar code.
`      Width = 340:Height = 70`	
`      BMP=curdir$+"\gx\20F5.bmp"`	
`    END CREATE`	
`  END CREATE`	
`END CREATE`	
`NumberSistemPri$=LEFT$(ComboBox1.Text,LEN(ComboBox1.Text)-1)`	Assign as image.
`NumberSistemSec$=RIGHT$(ComboBox1.Text,1)`	
`SetFocus(Edit2.Handle)`	
`Form.ShowModal`	
`SUB ComboBox1Change (Sender AS QCOMBOBOX)`	If the user alters the country of origin.
`   NumberSistemPri$=LEFT$(ComboBox1.Text,LEN(ComboBox1.Text)-1)`	
`   NumberSistemSec$=RIGHT$(ComboBox1.Text,1)`	
`   SetFocus (Edit2.Handle)`	
`END SUB`	
`SUB Edit2KeyDown (Key AS WORD, Shift AS LONG, Sender AS QEDIT)`	If the user entered the Product code.
` IF Key = 13 THEN`	
`   IF LEN(Edit2.Text)=5 THEN SetFocus (Edit3.Handle)`	
` ELSE`	
`   IF LEN(Edit2.Text)=5 THEN SetFocus (Edit3.Handle)`	
` END IF`	
`END SUB`	
`SUB Edit3KeyDown (Key AS WORD, Shift AS LONG, Sender AS QEDIT)`	The digital check.
`dim summ as single`	
`summ = 0`	
` IF Key = 13 THEN`	
`  dim kod$ as string`	
`   Kod$=ComboBox1.Text+Edit2.Text+Edit3.Text`	
`   FOR i=0 TO LEN(Kod$)`	
`     IF i=LEN(Kod$) THEN exit for`	
`      dim digit as single`	
`     Digit=VAL(LEFT$(RIGHT$(Kod$,LEN(Kod$)-i),1))`	
`     Select Case i`	
`     Case 0,2,4,6,8,10,12`	
`       summ=summ+Digit*1`	
`     Case 1,3,5,7,9,11,13`	
`       summ=summ+Digit*3`	
`     End select`	
`   NEXT`	
`   IF RIGHT$(STR$(summ),1)="0" THEN`	
`    Edit4.Text="0"`	
`   ELSE`	
`    Edit4.Text=STR$(10-VAL(RIGHT$(STR$(summ),1)))`	
`   END IF`	
`   TEKSTX = ComboBox1.Text+Edit2.Text+Edit3.Text+Edit4.Text`	
`   barcod=CODE20F5(TEKSTX)`	
`   makebmp (barcod,TEKSTX,BLineWidth,filename)`	
` END IF`	

CODING	EXPLANATION
END SUB	
FUNCTION CODE2OF5(tekst AS STRING) AS STRING	Establish image with bar code (codes).
BCodel(0) = "0100111" BCodel(1) = "0110011" BCodel(2) = "0011011" BCodel(3) = "0100001" BCodel(4) = "0011101" BCodel(5) = "0111001" BCodel(6) = "0000101" BCodel(7) = "0010001" BCodel(8) = "0001001" BCodel(9) = "0010111" BCoder(0) = "0001101" BCoder(1) = "0011001" BCoder(2) = "0010011" BCoder(3) = "0111101" BCoder(4) = "0100011" BCoder(5) = "0110001" BCoder(6) = "0101111" BCoder(7) = "0111011" BCoder(8) = "0110111" BCoder(9) = "0001011" BCodea(0) = "1110010" BCodea(1) = "1100110" BCodea(2) = "1101100" BCodea(3) = "1000010" BCodea(4) = "1011100" BCodea(5) = "1001110" BCodea(6) = "1010000" BCodea(7) = "1000100" BCodea(8) = "1001000" BCodea(9) = "1110100" BCodep(0) = "111111" BCodep(1) = "110100" BCodep(2) = "110010" BCodep(3) = "110001" BCodep(4) = "101100" BCodep(5) = "100110" BCodep(6) = "100011" BCodep(7) = "101010" BCodep(8) = "101001" BCodep(9) = "100101" TEKSTX = RIGHT$(TEKST,LEN(TEKST)-1) +CHR$(CHK) TEKSTX = TEKST + CHR$(CHK) barlengte=len(TEKSTX) dim bitf$ as string:dim bitp$ as string BITf$="":BITp$="" FOR x = 1 TO 14 dim barchar as single BARCHAR=ASC(MID$(TEKSTX,x,1))-48 SELECT CASE x CASE 1 dim parity$ as string parity$=BCodep(BARCHAR) CASE 2 TO 7 IF LEFT$(RIGHT$(parity$,LEN(parity$)-(x-2)),1)="1" THEN BITf$=BITf$+LEFT$(BCODER(BARCHAR),7) ELSE BITf$=BITf$+LEFT$(BCODEL(BARCHAR),7) END IF CASE 8 TO 13 BITp$=BITp$+LEFT$(BCODEA(BARCHAR),7) END SELECT NEXT x CODE2OF5="000000000000101"+BITf$+"01010"+BITp$+"101" END FUNCTION	
sub makebmp (barcode as string,tekst as string,BLineWidth as integer,filename as string)	Create the bar code.

CODING	EXPLANATION
breedte=len(barcode)*blinewidth	
lang=len(barcode)	
BColor=0	
bmp.Monochrome= 1	
bmp.Width=breedte+1	
bmp.Height=70	
BXPos=1	
FOR bcount = 1 to lang	
if mid$(barcode,BCount,1)="1" then bcolor=0	
if mid$(barcode,BCount,1)="0" then bcolor=16777215	
SELECT CASE bcount	
CASE 13 TO 15,58 TO 62,105 TO 107	
bmp.FillRect (BXPos*1 , 1,(BXPos+BLineWidth) ,60 ,BColor)	
CASE 16 TO 57,63 TO 104	
bmp.FillRect (BXPos*1 , 1,(BXPos+BLineWidth) ,50 ,BColor)	
END SELECT	
BXPos=BXPos+BLineWidth	
NEXT BCount	
bcolor=16777215:bmp.Font.size=12:bmp.Font.AddStyles(fsBold)	
tekstpos=(1*1): bmp.TextOut(tekstpos,50,NumberSistemPri$, 0,16777215)	
tekstpos=(21*2): bmp.TextOut(tekstpos,50,NumberSistemSec$+Edit2.Text, 0,16777215)	
tekstpos=(67*2):bmp.TextOut(tekstpos,50,Edit3.Text+Edit4.Text,0,16777215)	
bmp.SaveToFile(filename)	
Image.BMP=curdir$+"\gx\20F5.bmp"	
end sub	

14.6.2 Reading a Barcode with a Barcode Scanner

You need a bar code scanner for this program to work. Simply plug it in the USB Port. No drivers or special settings are required. It must be plugged in before running the program. Remember now that the bar code scanner is actually a keyboard that converts bar code to text (letters and numbers), so as it is reading the barcode, it will send it character by character to the active element (for instance a QEDIT).

CODING	EXPLANATION
$TYPECHECK ON	
$INCLUDE <RapidQ2.inc>	
declare sub checkscannedcode(key as byte)	The sub program that will be called as the bar code is being scanned into the QEDIT.
dim ppscan as single	
dim i as single	
dim iscan as single	
CREATE Form AS QFORM	
Caption = "SCANNER":Width = 500:Height = 450:Center	
create m30 as qpanel	
Top = 20:Left = 5:Height = 380:Width = 450:color=8421504:bevelinner=1:bevelouter=-1	
create scancoder as qedit	The QEDIT that will record the bar code scanning.
left=10:top=10:color=255:TabOrder=1:onkeyup=checkscannedcode	
showhint=1:hint="this edit box must be active and green for the scanner to scan ..."	
end create	
create barscan as qstringgrid	The StringGrid that will record all the bar code scanned entries.
left=20:top=50:colcount=3:rowcount=500:height=300:width=400	
separator=chr$(240):defaultrowheight=16	
cell(0,0)="SNO":colwidths(0)=40:cell(1,0)="BARCODE":colwidths(1)=80	
cell(2,0)="TIME":colwidths(2)=200:enabled=0	
end create	
end create	
END CREATE	
ppscan=1	Start from line 1 within the string grid. Remember the headings is line 0.
SetWindowLong(Form.Handle, -8, 0)	
SetWindowLong(Application.Handle, -8, Form.Handle)	
Form.ShowModal	

CODING	EXPLANATION

sub checkscannedcode(key as byte)
  if key=13 then
    if len(scancoder.text)<13 then
      if scancoder.text<>barscan.cell(3,ppscan-1) then
        barscan.cell(0,ppscan)=str$(ppscan)
        barscan.cell(1,ppscan)=scancoder.text
        barscan.cell(2,ppscan)=time$
        if ppscan>10 then barscan.toprow=ppscan-8
        ppscan=ppscan+1
        scancoder.text=""
      end if
    end if
    scancoder.text=""
  end if
end sub
	Scan and **record** character as **KEY**.
	If **ENTER** then
	If the scanned text is less than 13 then … if the new scan is not the previous scan (avoid double scans) then record the information into the line of the stringgrid.
	Ensure the **line** is **visible** as the entries are recorded into the stringgrid.
	Add to **next line** for **next recording**.
	Clear the text box of the **scancoder QEDIT**.

14.7 SCREEN(s)

The screen is the actual screen that you uses. You may have an onboard screen (laptop), external screen (desktop) and extended screen (additional screen/LCD projector). You have the following functions related to screens to modify or obtain information from:

ACTION	CODING	EXPLANATION
Screen.ClientWidth Screen.ClientHeight	$TYPECHECK ON $Include <rapidq2.inc> showmessage str$(**screen.clientwidth**) showmessage str$(**screen.clientheight**)	Determine width of screen. Determine height of screen. These values are numerical values so you need STR$() to display it.
Screen.GetPixelDepth	$TYPECHECK ON $Include <rapidq2.inc> showmessage str$(**screen.getpixeldepth**)	This will determine the bits/pixel setting of the screen (either 8/16/32 bit) of which 32 bit is the highest.
Screen.Monitors	$TYPECHECK ON $Include <rapidq2.inc> showmessage str$(**screen.monitors**)	This command will determine how many screens are active on your system. 1 = 1 screen, 2 = 2 screens (duel). If you plug in an additional screen (**VGA** or **HDMI**) then you may either duplicate or extend it. To extend will increase the value of the total width, however it will not be visible with **CLIENTWIDTH**. Should you wish to let your program appear on the second screen then **LEFT=SCREEN.CLIENTWIDTH+10**
Screen.SetResolution	$TYPECHECK ON $Include <rapidq2.inc> Screen.SetResolution(1024,768,32,0)	This will alter the screen's resolution. You have the following possible settings (if your screen are able to): • 1366,768 (15") • 1360,768 • 1280,720 (14") • 1280,600 • 1024,768 (netbooks) • 800,600 (Games,LCD) • 640,480 (LCD)

ACTION	CODING	EXPLANATION
		Larger screen settings are available; these are the standard settings for the average computer.
Short-Cut links	$TYPECHECK ON $Include <rapidq2.inc> defint a dim systm as qsystem systm.showcontrolpanel("display",1) systm.showcontrolpanel("power",1)	Link to screen savers Link to screen power settings Screen Saver settings. Power Settings settings.