

Programming Gambas from Zip

Contents

Programming projects are in Red.

Programming and its Grammar

[Objects, Subs and Events](#)

[Names and Memories](#)

[Properties and Kinds of Things \(Classes\)](#)

[Comments](#)

Computers Can Do Three Things

[Memory](#)

Functions and Subs

Preventing Errors

[Calculate Speed from Distance and Time](#)

[Conditional Execution](#)

If...Then...Else Select...Case...

[Game of HiLo](#)

Key class and checking the keyboard

[Grading Student Marks](#)

Colouring Alternate Rows, colours, adding a Quit menu

[Repetition](#)

For...Next...

Repeat...Until

While...Wend

[Moving Button Animation](#)

[TableView that adds up to 5 numbers](#)

[TableView that adds numbers in a grid](#)

Arrays, Lists, Sorting and Shuffling

[Making a Table of Doubles, Squares and Square Roots](#)

Format a number

[Game of Moo](#)

Assignment Operators like += and &=

Writing on the Hard Drive 1

Saving and Opening Text Files

[Game of Animal](#)

Showing and Hiding Objects

Expandable Forms (Automatic Arranging of Objects)

[A Spreadsheet to Average Student Marks](#)

Contextual Menus

Character Codes and Keeping Time

[Game of Concentration](#)

Radio Buttons and Groups, Parents and Children

Writing on the Hard Drive 2

[Saving Settings](#)

“Me”, Saving a colour, checkbox and tableview
Coloured panels
The If(..., ..., ...) function

Modules and Classes

[Locate a Name in a List by Typing](#)

[Properties, Methods, Events](#)

[Static Classes](#)

[Making a SearchBox class with a New Event Based on a TableView](#)

SQLite Databases

[Tables, Primary Keys, Loading the Database Component of Gambas](#)

[Databases Can Do Four Things:](#) Display(Access), Add, Delete and Modify Records

[Make a database with a single table and fill it with random numbers](#)

[SQL — Structured Query Language](#)

[Begin, Commit and Rollback](#)

[Select, Insert Into, Delete From, Update](#)

[* for All Fields](#)

[WHERE clause for some records only](#)

[ORDER BY clause to set the sort order](#)

[*A Cash Spending Application*](#)

Printing

[Printer Object](#)

[*Print Some Plain Text*](#)

[HTML](#)

[*Print Some Rich Text*](#)

[Text and Images](#)

[*Print an image*](#)

[*Print a Class List*](#)

[*Print a calendar for the month*](#)

Tray Item: Notebook

Appendices

[Did You Know? — from Gambas ONE](#)

[Functions Reference](#)

[Constants](#)

[Operators](#)

[Data Types and Conversions](#)

[Formatting](#)

[Operator Precedence](#)

Afterword

Programming Gambas from Zip

Programming

Programming is making applications. It is also called ‘coding’ because the instructions are written in ‘code’.

There are many programming languages. Ours is Gambas. (Gambas **A**lmost **M**eans **B**ASic). **B**ASIC (**B**eginner's **A**ll-purpose **S**ymbolic **I**nstruction **C**ode) first appeared on 1 May, 1964 (55 years ago in 2019, as I write). Everyone loved **B**ASIC. It made programming possible for us ordinary mugs. Benoît Minisini designed and wrote Gambas and gives it away free. He lives in Paris, France and is 47 (in 2019). He said, “Programming is one of my passions since I was twelve” and “I am using many other languages, but I never forgot that I have learned and done a lot with BASIC.”



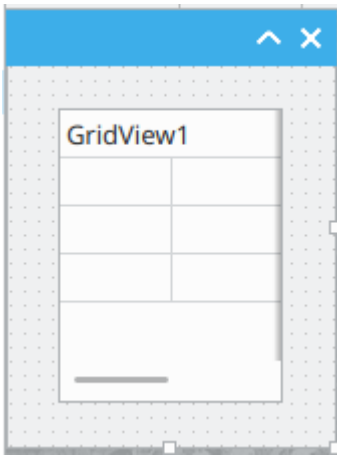
commons.wikimedia.org/wiki/File:Benoît-Minisini.png

Benoît Minisini, 2005—one really great guy. We stand in awe.

This is a piece of code:

```
Public Sub Form_Open()  
    gridview1.Rows.count = 5  
    gridview1.Columns.count = 1  
    For i As Integer = 0 To 4  
        gridview1[i, 0].text = Choose(i + 1, "Red", "Yellow", "Green", "Blue",  
"Gold")  
    Next  
    gridview1.Columns[0].text = "Colour"  
End
```

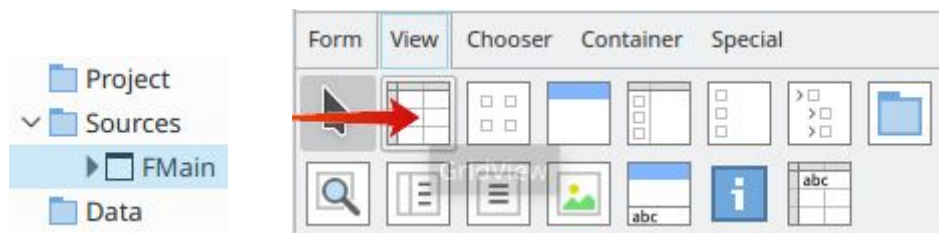
The language is Gambas. This is a **Sub** (also called *Method*, *Procedure* or *Subroutine*. If it gave you some sort of answer or result it could be called a *Function*).



On the left is the design. It is a form. Think of it as a Window. On the right is the application in action. That is what you see when you *run* the program.

Starting an application is called *running* it. The program *runs*. Every statement in the program is *executed*. The **Form_open** sub (above) fills the **gridview** object.

Part 1



Open Gambas.

Double-click FMain in the list on the left. The form opens.

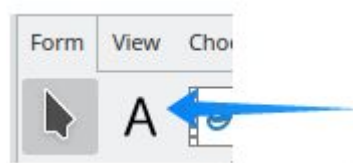
Look for a GridView among the objects at the bottom right. Drag it onto the form. Adjust its size by dragging the handles (small squares on its edge). Adjust the form size too.

Right-click the form (NOT the gridview) > Event menu item > Open.

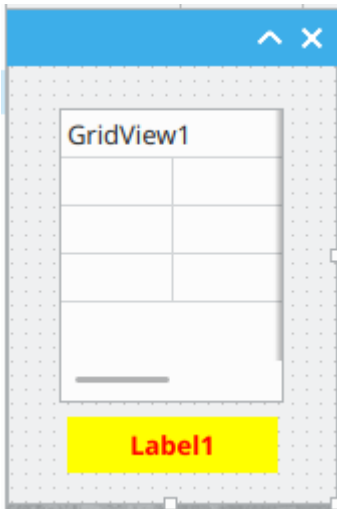
Type or paste in the code for the open event (as above).

Press F5 to run your program.

Part 2



With the same program with the gridview that you just tried, drag a Label onto the form.



Right-click the gridview > Event > Enter, and type this code:

```
Public Sub gridview1_Enter()
    Label1.text = "Hello"
End
```

Right-click the gridview > Event > Leave, and type this code:

```
Public Sub gridview1_Leave()
    Label1.text = ""
End
```

Run the program with F5. Move the mouse in and out of the gridview.

Label1.text = "Hello" is a way of saying "Put the word Hello into the text property of Label1."

Break Complex Things into Parts

This is the most important idea about programming. The only way to write a good program is to "divide and conquer". Start with the Big Idea, then divide it into parts. Piece by piece, write the program. This is called *top-down development*. The opposite is useful, too: *bottom-up development*. Write a simple program and keep adding bits and pieces to it until it does all the things you want it to.

The first programs were simple creatures that did but one thing. Then came *menus*, where you could select one of several things by pressing a key. For example, "Press 1 to get your account balance. Press 2 to recharge. Press 0 to speak to an operator." Nowadays applications do many things. To choose, you have a *menu bar* at the top, *menus* that drop down, and *menu items* that you click on in each menu. You can click a *button*. Buttons can be solitary, or friendly with each other and gather together in a *toolbar*. You can *pop up a menu* anywhere. You can type *shortcut keys* to get things to

happen. There can be several *windows*. Within windows (forms) there can be many *pages* with or without an index tab at the top of each.

Buttons can be on the screen, or the physical keys on your keyboard. The standard keyboard has 101 keys, but they can all be given a second function if you hold down **CTRL-**, the control key, while you type them. That gives you 202. Not enough? Holding down **SHIFT-**, the shift key, makes all the keys different again, giving you 303 buttons. There is **ALT-** (alternative) that makes all the buttons different yet again: 404 of them. The modifier keys held down in combination give you more sets of keys, like more keyboards: **SHIFT+CTRL**, **CTRL+ALT** and **SHIFT-CTRL-ALT**. Now I have lost count. Fortunately for us, no application uses them all.

The key to good programming is to get things into order. Be neat and tidy. Arrange things. Things you do all the time should have their buttons visible; things you do not so often could be hidden away in a popup menu.

Reduce complicated tasks to a series of simple steps. Write lots of little **Subs**, not just one big sub. Write “master subs” that call on lesser subs to do little jobs. The biggest master of them all is the user. You use the program to do this, then that, then something else. The program says, “At your service!” and calls on different subs you have written to do what you want. In the meantime, it waits and listens for your command, checking the keyboard and the mouse, or, if it is not waiting for you, it keeps busy doing something you set it to work on.

Objects, Subs and Events

The program above has two things (called **Objects**): a *form*, and a *gridview*.

Things that happen are called **Events**.

There is an event that “*fires*” or “*is triggered*” when a form opens. It is called (no surprise) the **OPEN** event. If you want something to happen when the form opens, write a sub called *Form_Open*. The instructions there will be carried out when the **OPEN** event fires. The form opens when the application starts up.

You have to think of two things: *what* will happen and *when* it will happen. The “what” is the list of instructions in Subs. The “when” is a case of choosing which events to handle.

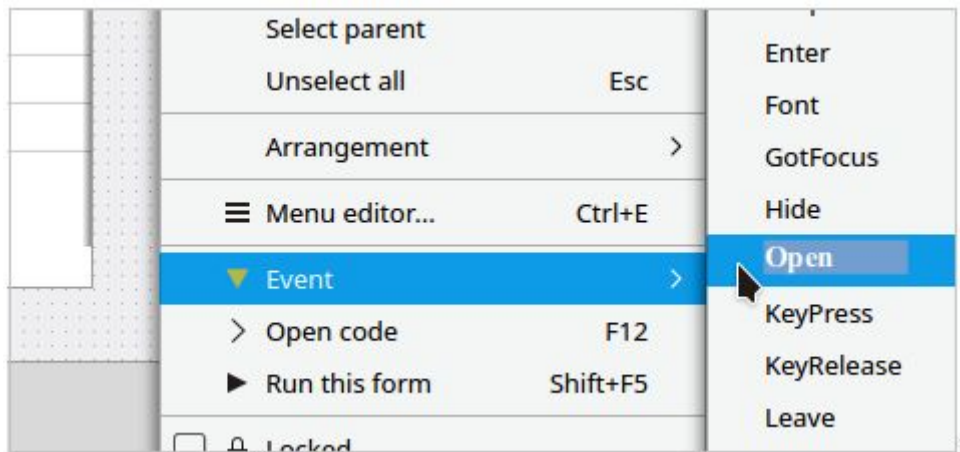
Names and Putting Something into Something

Everything has to have a name. People have names. Forms have names. Buttons have names. You get to choose, using letters and numbers, always starting with a letter, and not having any spaces.

The main form is called **FMain**. Its open event is **Form_Open**.

I like the convention of calling it *FMain*. The “F” says, “Form”. If I had a form that listed people’s addresses, I would call it *FAddress*. When it opens—perhaps at the click of a button—its open event would look for a sub called *FAddress_Open*.

Right-click an object, choose EVENT, and click on the event you want to write some programming for:



Right-click the form > Event > Open > write code for the Open event.

There will be a big confusing list of events. Don't be fazed: only a few are used often. A big part of learning the language is getting used to the events that are most useful to certain objects. Buttons, for example, like to be **Clicked**; you don't use the other events very often. Possibly, entering a button might put some help text in a line at the bottom of the window, and leaving it will clear the message, but not often. I have not seen a double-click handler for a button yet: who double-clicks buttons? You will get used to the favourite events that objects have.

In the program you have written, type or paste this sub. Every time you roll the mouse into the gridview, a message saying "Hello" appears. When you have tried it and it has annoyed you enough, delete the three lines of code.

```
Public Sub Gridview1_Enter()  
    Message("You have entered the gridview ... evil laugh...")  
End
```

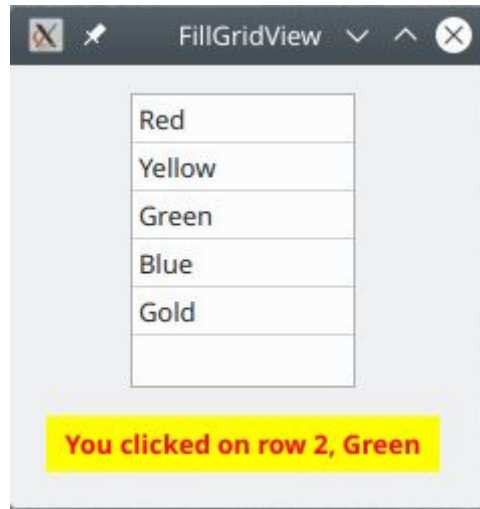
The Activate event happens when you double-click. Enter these lines and double-click any of the lines.

```
Public Sub gridview1_Activate()  
    Message("You double-clicked row " & gridview1.row)  
End
```

gridview1.row is the row you double-clicked on. It is set when you click or double-click. The "&" sign (ampersand) joins two strings of text. (Strings are text ... characters one after the other.)

This next code will do something when you click on gridview1:


```
Public Sub gridview1_Click()  
    Label1.text = "You clicked on row " & gridview1.Row & ", " &  
    gridview1[gridview1.row, 0].text  
End
```



There are two strings. (1) "You clicked on row " and (2) ", ". Strings have double quotes around them.

What is on the right of the *equals* goes into what is on the left.

You don't put something into *Label1*; you put it into the *text* of *Label1*.

Usually languages won't allow this: `Label1.text = 345`. It must be `Label1.text = "345"` or `Label1.text=Str(345)`, but Gambas doesn't mind. *Gridview1.Row* refers to the row number you clicked on. Numbers normally go into things that store numbers, and strings go into things that store strings, but Gambas converts it automatically.

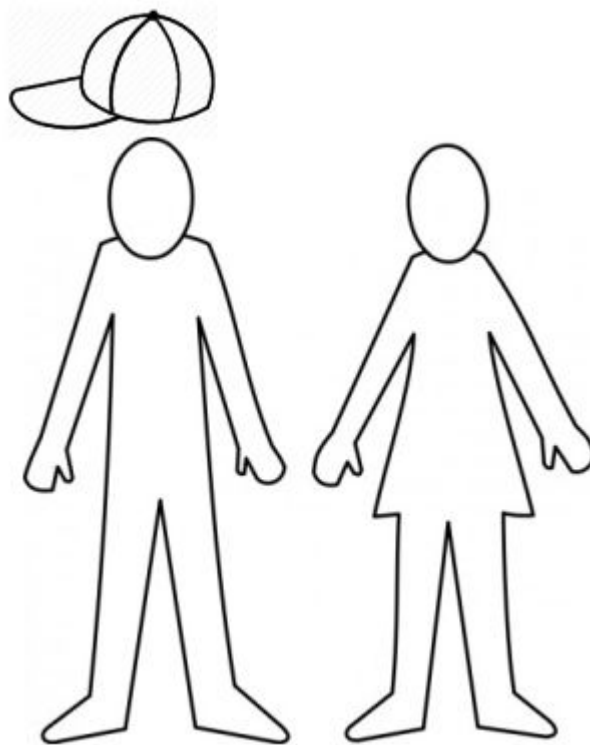
`gridview1[gridview1.row, 0].text` looks complicated, but let's break it down. From the left, it is something to do with *gridview1*. From the right, it is the *text* in something. The part in square brackets is two numbers with a comma in between: the row you clicked on and zero. In the example above, it is [2,0]. Row two, column zero.

Gridview rows and Gridview columns start their numbering at zero.

Properties and Kinds of Things

In this part the examples are not Gambas, but they are written in Gambas style.

Behold John and Joan:



Parts are referred to like **John.arm**, **Joan.head**

Sometimes parts have smaller parts, such as **John.hand.finger**

Parts have properties, like **Joan.height**

Some properties are *integers* (whole numbers), like **Joan.Toes.Count**

Some properties are *floats* (numbers with decimal fractions) like **John.height** . This is not the same as **John.Leg.Height** .

Some properties are *booleans* (true/false, yes/no) like Hat On or Hat Off. We could say Persons, as a class, have a property called HatOn, “**John.HatOn**” or “**Joan.HatOn**”. (It is *true* or *false*.)

They belong to the class called “person”. John is a person; Joan is a person. Persons all have a height, weight, hat-on/hat-off, and other properties.

Persons also have various abilities. The “person” class can sing and wave and smile. Their legs can walk. For example, **John.sing(“Baby Face”)** . Sometimes John will need to know which version, for example, **John.sing(“Baby Face”, “Louis Armstrong”)** or **Joan.wave(“Hello”, “John”, 8)**. This means “Joan! Give a hello wave to John, vigorously!”

There will also be events they can respond to, like a push, or when they hear someone singing. These events are **John_push()** and **Joan_push()**, **John_HearsSinging()** and **Joan_HearsSinging()**

Let’s write a response to these events in the style of Gambas.

John might get annoyed when the Push event takes place:

```
Public Sub John_push()  
    John.Turn(180)  
    Message.shout("Hey! Quit shoving!")  
End
```

Joan might join in the song that she hears and start singing it too:

```
Public Sub Joan_HearsSinging("Ave Maria")  
    Joan.HatOn = False  
    Joan.sing("Ave Maria")  
End
```

Boolean properties are yes/no, true/false, haton/hatoff things. Putting false into the HatOn property means taking her hat off before she sings.

John has the ability to turn around, but we must say how much to turn. He turns 180°, doing an about-face so he is facing the person who pushed him.

Mary can sing, but we must say which song to sing. It is the same one that she is hearing. We say that the *HearsSinging* event is passed the name of the song, "Ave Maria". That song title is passed to Joan's singing ability, which is referred to as *Joan.sing*. The bits of information you pass on so that the action can be done are called **parameters**. Some methods require more than one parameter, and they can be numbers (integers, floats...) or strings (text like "Jingle Bells" or "Fred" or "Quit pushing, will you?!") or booleans (HatOn/HatOff, Complain/Don'tComplain True/False kinds of things), or other objects or, well, anything.

Let's define a sub and this time we'll put some repetition in it. There are two kinds of repetition—one where you repeat a definite number of times, and another where you have to check something to see if it is time to stop. (And the latter comes in two kinds, where you check if it is time to stop before you start or after you have done it—pretested loops or post-tested loops.) Here are the definite and indefinite types:

Five times:

```
Public Sub DoExercises()  
    For i as integer = 1 to 5  
        Hop()  
    Next  
End
```

Until tired:

```
Public Sub DoExercises()  
    Do Until Tired()  
        Hop()  
    Loop  
End
```

While not tired:

```
Public Sub DoExercises()  
    While Not Tired()  
        Hop()  
    Wend  
End
```

In the first, there will be 5 hops and that is all. “i” is an integer that is created in the line **For i as integer = 1 to 5** and it counts from 1 to 5. It is made one bigger by the word **NEXT**. The repeated section is called a *loop*.

Tired() is something that has to be worked out. The procedure for working it out will be in another sub somewhere. At the end of it there will be an answer: are you tired or not? Yes or no, true or false. That will be *returned* as a final value. *Tired()* looks like a single thing, the value returned by a function. **Functions** work something out and give you an answer.

“Hop” might need further explanation if “hop” is not already known. You can use any made-up words you like, provided you explain them in terms that are built-in and already known to Gambas. Here is a sub that explains the *Hop* procedure:

```
Public Sub Hop()  
    GoUp()  
    GoDown()  
End
```

The beauty of breaking procedures down into simpler procedures is that the program explains itself. You are giving names to the complicated tasks without getting lost in the fine detail of how they are done. Not only can you use words that make sense to someone else reading your program but you can track down errors more easily. You can test each part more easily. You can also modify your program more easily. *DoExercises()* can be left as it is, but you can change the definition of hopping with

```
Public Sub Hop()  
    GoUp()  
    ShoutAndPunchTheAir()  
    GoDown()  
End
```

OR

```
Public Sub Hop()  
    GoUp(LeftFoot)  
    GoDown()  
    GoUp(RightFoot)  
    GoDown()  
End
```

Comments

Anything after a single apostrophe is a note to yourself. Gambas disregards it. Put in comments to remind yourself of what you are doing. It can also explain your thinking to someone else who might need to read your program. Here is some more pseudocode:

```
Public Sub Joan_HearsSinging("Ave Maria") 'you must say what the song is
    Joan.HatOn = False 'remove your hat
    'now the fun starts
    Joan.sing("Ave Maria") 'sing the same song
End
```

What Computers Can Do

Now we are back into Gambas. That is enough pseudocode.

There are three things computers can do: *memory*, *repetition* and *conditional execution*. Repetition is doing things over and over like working through millions of numbers or thousands of records. Conditional execution means making choices.

Memory

Calculators sometimes have M+ and MR keys. Whatever number is showing goes into a memory when you press M+. Whenever you need to use that number again, to save typing it, press the *Memory Recall* button, MR. The memory is like a note you have made to yourself to remember this number.

Computers have as many memories as you like and you give them names. Putting something in a memory is as easy as typing **Age = 23**. What is on the right is put into the memory on the left. To see what is in the memory called *Age*, print it or put it in some place where you can see it. **Label1.text = "Your age is " & Age** .

Before you can use some name as a memory you must tell Gambas what kind of thing will be stored in it. This is what the **DIM** statement does. You declare it before it is used. For example, **Dim Age As Integer** or **Dim FirstName as String** . Memories are called *Variables* or *Properties* if they are associated with something.

You can declare a memory and put something into it in one line:

```
Dim FirstName as String = "Michael"
```

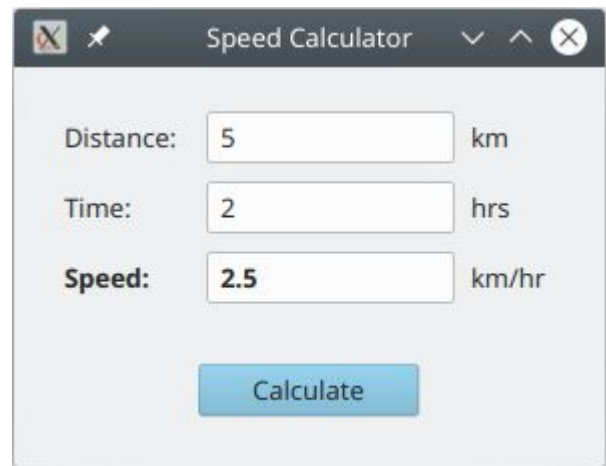
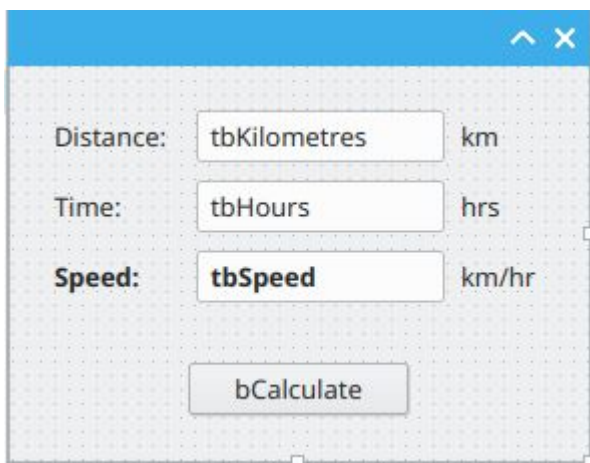
You can use memories to calculate something, and put the answer into another memory:

```
Dim Speed as float = 45
Dim Time as float = 1.5
Dim Distance as Float = Speed * Time
Label1.text = Format(Distance, "##0.##") & "km/hr"
```

To DIMension things you need to know what types are allowed. Here is a list of data types:

Name	Description	Example	De- fault
Boolean	True or False	1 = 2	False
Integer	A whole number from -2147483648 ... +2147483647	123	0
Float	A number with a decimal part, like 2.34	2.34	0.0
Date	Date and time, each stored in an <i>integer</i> .		Null
String	Letters and digits and symbols strung together. Text.	"ABC 12%"	Null
Variant	Any datatype. It has to be converted to some type before it can be used.		Null

Calculate Speed from Distance and Time



There are six labels, three textboxes and one button. The names for the labels do not matter, but the textboxes and the button are named as shown. The program is:

```
Public Sub bCalculate_Click()  
    tbSpeed.text = Val(tbKilometres.text) / Val(tbHours.text)  
End
```

“Val” takes a string and converts it to a number. It means “the value of”.

It could be written this way, using variables, but it would take more lines:

```
Public Sub bCalculate_Click()  
    Dim d As Float = Val(tbKilometres.text)  
    Dim t As Float = Val(tbHours.text)  
    Dim s As Float = d / t
```

```
tbSpeed.text = s
End
```

If you want to give better names to the variables,

```
Public Sub bCalculate_Click()
    Dim distance, time, speed As Float
    distance = Val(tbKilometres.text)
    time = Val(tbHours.text)
    speed = distance / time
    tbSpeed.text = speed
End
```

And you could write a function that takes distance and time and returns the speed. It is good to teach the computer things. Here we have taught the computer that **Speed(5,2)** is 2.5. Our calculate button uses it. We could have a menu item that also uses it.

```
Public Sub bCalculate_Click()
    tbSpeed.text = Speed(Val(tbKilometres.text), Val(tbHours.text))
End

Public Sub Speed(d As Float, t As Float) As Float
    Return d / t
End
```

Now let's trick our program. Don't put in anything for the distance or time. Just click the Calculate button. The poor thing cannot cope. We get this error:

```
Dim d, t, s As Float
d = Val(tbKilometres.text)
t = Val(tbHours.text)
s = d / t
tbSpeed.text = s
```

Type mismatch: wanted Float, got Null instead in Speedo:6.

Problems come at the extremes. In repeated sections, they are most likely to occur in the first or the last repetition. Here, there is an extreme input: nothing, zip, nilch. Gambas cannot get the Val() of that.

tbKilometres.text was *Null*. We should anticipate that someone might click the button without putting in any numbers. Here are two ways handle the situation, and the second one is better because 'prevention is better than cure':

1. Finish early (Return from the sub early)

```
Public Sub bCalculate_Click()
    Dim d, t, s As Float
    If IsNull(Val(tbKilometres.text)) Then
        Message("Hey, you there! Give me a NUMBER for the kilometres.")
        Return
    End If
    s = d / t
    tbSpeed.text = s
End
```

```

    Else
        d = Val(tbKilometres.text)
    Endif
    If IsNull(Val(tbHours.text)) Then
        Message("A number would be nice for the hours. Please try again.")
        Return
    Else
        t = Val(tbHours.text)
    Endif
    s = d / t
    tbSpeed.text = s 'To round to 2 decimal places, change s to format(s,"#.00")
End

```

2. Only enable the button if the calculation can proceed. Set the Enabled property of the button to False to begin with. We need to handle a few events.

```

Public Sub bCalculate_Click()
    tbSpeed.text = Val(tbKilometres.text) / Val(tbHours.text)
    Fmain.SetFocus 'otherwise the button stays highlighted; focus the form
End

Public Sub tbKilometres_Change()
    CheckBothAreNumbers
End

Public Sub tbHours_Change()
    CheckBothAreNumbers
End

Public Sub CheckBothAreNumbers()
    bCalculate.Enabled = Not IsNull(Val(tbKilometres.text)) And Not
    IsNull(Val(tbHours.text))
End

```

bCalculate.Enabled = means ‘set the enabled property of the button to...’

Not IsNull(Val(tbKilometres.text)) means Yes if the text in the kilometres textbox can be converted to a number.

Every textbox has a Change event. It fires when the text in the box changes. Every time you press a key, that Change event is going to see if it’s time to enable the bCalculate button.

And because I cannot leave well enough alone, I apologise for introducing the “Group” property. You may need coffee. Or skip this section. Do you notice that the two textboxes have to each handle the Change event the same way? *tbKilometres* and *tbHours* both check for numbers and enable or disable the button accordingly. Wouldn’t it be nice if both textboxes were like just one textbox? Gambas can do this. Put them in a **Group**. Then this single group will have a *Change* event, and you can handle it just once. Group is a property. Find the Group property for each and set it to “InputBox”. Now your code becomes the simplest yet:

```

Public Sub bCalculate_Click()

```



```

    tbSpeed.text = Val(tbKilometres.text) / Val(tbHours.text)
    Fmain.SetFocus 'otherwise the button stays highlighted; focus the form
End

Public Sub InputBox_Change()
    CheckBothAreNumbers
End

Public Sub CheckBothAreNumbers()
    bCalculate.Enabled = Not IsNull(Val(tbKilometres.text)) And Not
    IsNull(Val(tbHours.text))
End

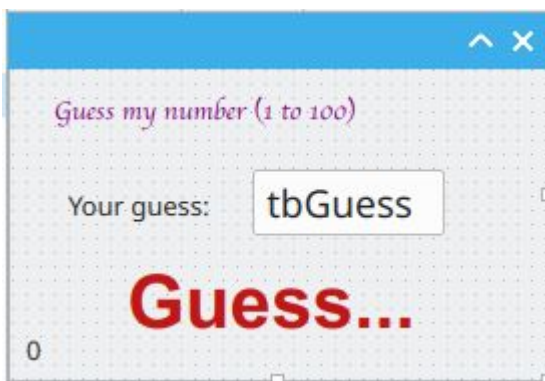
```

It is as if the two textboxes have become inputboxes (a name you just invented), with all the same events. One set of event handlers for several objects.

It's robust (resistant to users who insist on not using your application the way you expect them to). It's concise (3 event handlers, 4 lines of code). It works. One thing remains: a button that says QUIT with one thing in its Click event: the command Quit. Over to you.

If..Then..Else — Game of HiLo

This game is easy to play: one person thinks of a number between 1 and 100. You try to work out the number in as few guesses as possible. Each time you will be told “Too High!” or “Too Low!”.



There are three labels and one textbox. The large label where it says “Guess...” is named *labMyReply*. The textbox is named *tbGuess*. The small label in the bottom left corner is named *labCount*.

```

Public MyNumber As Integer

Public Sub Form_Open()
    MyNumber = Rand(1, 100)
End

```

```

Public Sub tbGuess_KeyPress()
    If Key.Code = Key.Enter Or Key.Code = Key.Return Then GiveMessage
End

Public Sub GiveMessage()

    If IsNull(tbGuess.text) Then 'no guess at all
        labMyReply.text = "Guess..."
        Return
    Endif

    Dim YourGuess As Integer = Val(tbGuess.text)

    If MyNumber = YourGuess Then
        labMyReply.Text = "Right! " & MyNumber
        MyNumber = Rand(1, 100)
        tbGuess.text = ""
        FMain.Background = Color.Green
        labCount.text = "0"
        Return
    Endif

    If YourGuess > MyNumber Then labMyReply.Text = "Too High!" Else
labMyReply.Text = "Too Low!"
        tbGuess.text = ""
        FMain.Background = Color.Pink
        labCount.text = Val(labCount.text) + 1

End

```

The program checks the guess when ENTER or RETURN is pressed in the textbox. The Key class is static, meaning it is always there—you do not have to declare or create it. If you ever need to check for some key, such as SHIFT-C, being pressed, you would write:

```

If Key.SHIFT And Key.Text = "C" Then
    GiveMessage
    Stop Event
Endif

```

The **Stop Event** line is there to prevent capital-C being printed in the textbox, which would normally happen when you type Shift-C in a textbox.

Public MyNumber As Integer means we want a public property called MyNumber. You could say Private instead of public. Private properties are accessible only in the code belonging to that form. If we had other forms (i.e. windows) they cannot see another form's private property. Declaring a property as *private* is a way of telling you, the programmer, that you have only used this property here, in this form's code that you are looking at. Also, another form could have its own property and use the same name.

In the **Event Form_Open()** event handler, Rand(1,100) is a built-in function that represents a random number between 1 and 100.

In the **Event tbGuess_KeyPress()** event handler, the key just typed is compared with the Enter and Return keys. If it was either one, the guess is checked. Every key has a number. Enter is 16777221 and Return is 16777220. We do not need to know the numbers, because they are stored in **Key.Enter** and **Key.Return**. These are constants in the Key class. Because the Key class is static we do not have to make a new example of it: it is always there and we can refer to it by its name. We never need another one of them. There is only ever one keyboard. We never have to say “the Key belonging to the ACER laptop keyboard” or “the Key that was typed on the HP laptop keyboard”.

If IsNull(tbGuess.text) Then ... End avoids the nasty situation of a person pressing Enter without having typed in any number at all.

Dim YourGuess As Integer = Val(tbGuess.text) puts the numeric value of what was typed into a variable called *YourGuess*. This is an integer. If a person typed 34.56, only 34 would be put into *YourGuess*.

If MyNumber = YourGuess Then... checks to see if *YourGuess* matches *MyNumber*. If it is, show the “Right!” message and make the background colour of the form green. Choose another random number ready for the next game and then **Return**, because nothing more needs to be done.

If YourGuess > MyNumber Then labMyReply.Text = "Too High!" Else labMyReply.Text = "Too Low!" puts the appropriate message into the *labMyReply* textbox. This is an *If...Then...Else* statement all on one line. In either case, continue on to make the background pink.

From the Gambas help page, these are the properties and constants for the Key class:

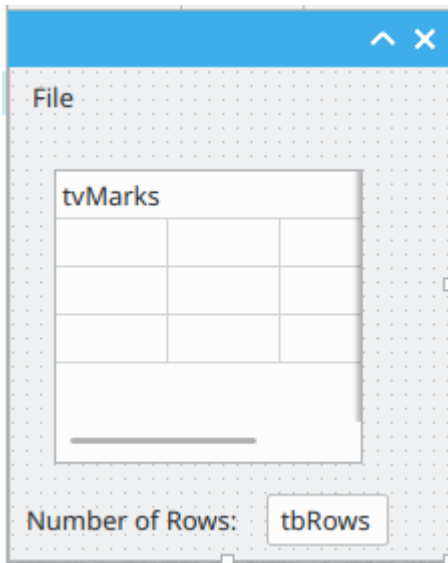
Static properties	Constants
Alt Code Control	AltGrKey AltKey BackSpace BackTab CapsLock
Meta Normal Shift	ControlKey Del Delete Down End Enter Esc
State Text	Escape F1 F10 F11 F12 F13 F14 F15 F16
	F17 F18 F19 F2 F20 F21 F22 F23 F24 F3
	F4 F5 F6 F7 F8 F9 Help Home Ins Insert
	Left Menu MetaKey NumLock PageDown
	PageUp Pause PgDown PgUp Print Return
	Right ScrollLock ShiftKey Space SysReq Tab
	Up

The constants return integer numbers. The properties are what was typed. So you could check if the user typed the PgDown key with **if Key.Code = Key.PgDown then...** Negotiating and reading the help pages is a skill in itself.

Select ... Case ... — Many Choices — Grading Student Marks

Let's type numbers into a TableView and make it work out whether each student earns an A, B, C, D, E or F grade. We'll need two columns. The first will be for the marks, the second for the grades.

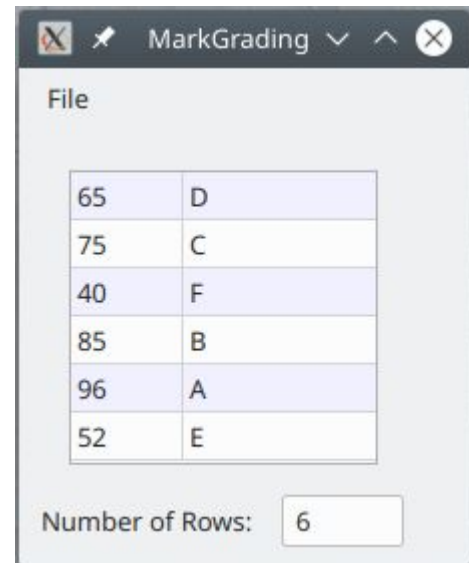
We'll skip having a column for student names for now. And we'll avoid using many **IF...THEN...** statements in favour of its big brother, **SELECT...CASE...**



File

tvMarks	

Number of Rows:



MarkGrading

65	D
75	C
40	F
85	B
96	A
52	E

Number of Rows:

The TableView is named tvMarks. The textbox is named tbRows. Double-click an empty part of the form and enter the code for the form's *open* event:

```
Public Sub Form_Open()  
    tvMarks.Columns.count = 2  
End
```

Double-click the textbox and enter:

```
Public Sub tbRows_KeyPress()  
    If Key.Code = Key.Return Or Key.Code = Key.Enter Then tvMarks.Rows.Count =  
    Val(tbRows.Text)  
End
```

Right-click the TableView > Event > Click, then right-click > Event > Save, and enter these:

```
Public Sub tvMarks_Click()  
    If tvMarks.Column = 0 Then tvMarks.Edit  
End  
  
Public Sub tvMarks_Save(Row As Integer, Column As Integer, Value As String)  
    tvMarks[Row, Column].Text = Value  
    Dim x As Float = Val(Value)  
    Select Case x  
        Case 90 To 100  
            tvMarks[Row, 1].Text = "A"  
        Case 80 To 89  
            tvMarks[Row, 1].Text = "B"  
        Case 70 To 79  
            tvMarks[Row, 1].Text = "C"  
        Case 60 To 69
```

```

        tvMarks[Row, 1].Text = "D"
    Case 50 To 59
        tvMarks[Row, 1].Text = "E"
    Case Else
        tvMarks[Row, 1].Text = "F"
End Select
End

```

The **tvMarks_Click()** handler lets you type in the cell if the column is 0. If you click in column 1 you will not be able to type: nothing happens when you click.

You might think that whatever you type in a cell should show up. It doesn't. It raises the *Save* event. You might want something else to appear other than what was typed. During the *Save* event, actually put the text that was typed into the *text* property of the cell:

tvMarks[Row, Column].Text = Value

The *Save* event comes with three parameters that you can use freely in the course of the event handler: **tvMarks_Save(Row As Integer, Column As Integer, Value As String)** . This line of code puts the value that was typed into the text of the cell. Which cell? *tvMarks[row, column]*. That is the cell.

You refer to a cell by using square brackets: *tvMarks[1,0]* refers to row 1, column 0. (Remember rows and columns are numbered starting with zero.) *tvMarks.Rows[2]* is row 2. *tvMarks.Columns[1]* is the whole of column 1.

A Nice Addition — Colour alternate rows

The *TableView_Data()* event is very useful. It is raised (happens) every time a cell needs to be redrawn on the screen. (It is useful to remember it deals with cells, not rows or columns or the whole table.) Right-click the tableView, then > Event > Data and enter this:

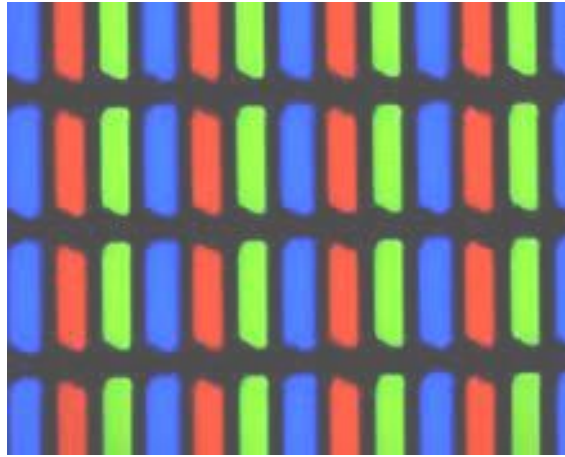
```

Public Sub tvMarks_Data(Row As Integer, Column As Integer)
    If Row Mod 2 = 0 Then tvMarks[Row, Column].Background = &F0F0FF
End

```

This gives alternate rows a light blue colour (very pretty). To explain, cells have a property called "background". It is a colour. Colours can be described in several ways: using a straight number is the simplest. The number is &F0F0FF.

Numbering Colours



A computer screen is full of little lights that light up Red, Green and Blue.

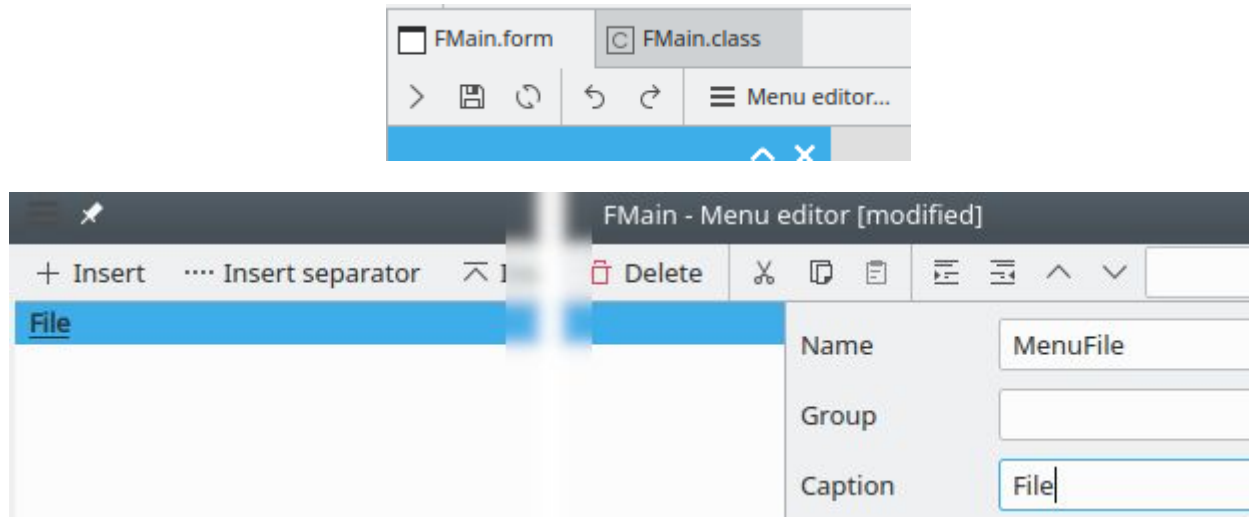
What sort of number is **&F0F0FF**? The “&” sign means the number is written in *hexadecimal*. Normally we use the decimal system, which is Base 10. You count from zero to nine and the digit goes back to zero and you increase the digit to its left by one. Here is normal counting in Base 10: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16... Using hexadecimal you have 16 digits, not ten. Here is counting in Base 16: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 1A, 1B, 1C, 1D, 1E, 1F, 20, 21... The eleventh number in the decimal system is written **11**. The eleventh number in the hexadecimal system is written **B**.

Now, what colour is **&F0F0FF**? The hexadecimal number **F0F0FF** is actually three 2-digit numbers: **F0**, **F0**, **FF**. The first number is how much Red. The second number is how much Green. The third number is how much Blue. That is RGB, red-green-blue. Each goes from **00** to **FF**. **00** in the first place would mean “No red at all”. **FF** in the first place would mean “Maximum red!”. Pure red would be **&FF0000**. Pure green would be **&00FF00**, and pure blue is **&0000FF**. So pure black is **&000000**. Pure white is **&FFFFFF**, which is all the colours as bright as you can make them.

This colour is grey: **&F0F0F0**. The red, green and blue lights are mostly on, but not fully bright. **F0** is not as bright as **FF**. To get a darker grey, lower the numbers but lower them all the same, e.g., **C0C0C0**. Darker again, **B0B0B0**. The tiniest bit darker than that is **AFAFAF**, because after **AF** comes **B0**. The point is, when they are all the same you get shades of grey. So look at **F0F0FF**. It is a very light grey (**F0F0F0**), but the last number, the one for Blue, is a bit brighter. It is **FF**. So the colour is a very light grey with the Blue just a bit brighter. That is, it’s very light blue. It is all about the mix of three colours, red, green and blue. This is pale pink: **FFD0D0**. This is pale green: **D0FFD0**. This is pale yellow: **FFFFD0**. (Red=brightest, Green=brightest, Blue=a little less bright). This is full yellow: **FFFF00**.

It is all about the tiny little LED lights on your screen. There are millions of them, but they are grouped in threes—a red, a green and a blue. You control the brightness of each little light. The brightness goes from **00** to **FF**, or in decimal, from 0 to 255. (**FF**=255). One red, one green and one blue act like one coloured dot. It is called a pixel. A typical laptop screen has a resolution of 1366 x 768 pixels. That is 1,049,088 pixels. Each has three little LED lights, making 3,147,264 lights on your screen, each one with 256 shades of brightness. Amazing!

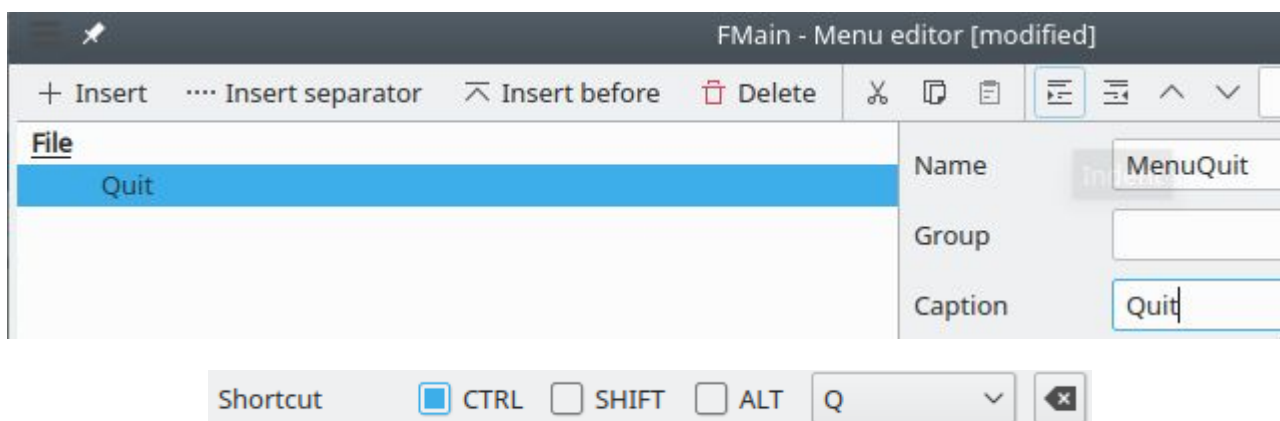
Adding a Quit Menu



Adding a File menu

Click “+Insert”. Name this menu *MenuFile* and caption it *File*. The caption is what the user sees. The name is how we refer to the menu in our programming.

Click “+Insert” again. We want it to be part of the *File* menu, so click the push-right button on the toolbar. While you are at it, give the menu CTRL-Q for a shortcut.



Adding a Quit menuitem to the File menu, with shortcut CTRL-Q

Now write a handler for when the user clicks on *MenuQuit*. (To do this, look for the *File* menu in the form and click it, then click the menuitem *Quit*.)

```
Public Sub MenuQuit_Click()  
    Quit  
End
```

Repetition

Repeated sections of code are called loops. There are a few different ways to repeat.

Gambas help lists them on <http://Gambaswiki.org/wiki/cat/loop>

Counting a fixed number of times:

FOR ... NEXT

```
For i as integer = 1 to 5
    'do something
Next
```

i is an integer that counts 1, 2, 3, 4, 5. Don't put "*as integer*" if you already have **DIM i As Integer**

You can use *i* in the loop, but do not change its value. The word **NEXT** increases it by one and sends the computer back to the start (the **FOR** line) where *i* is checked to see if it is bigger than 5. If it is, it goes to the line following **NEXT**.

REPEAT ... UNTIL

```
Repeat
    'do something
Until x > 50
```

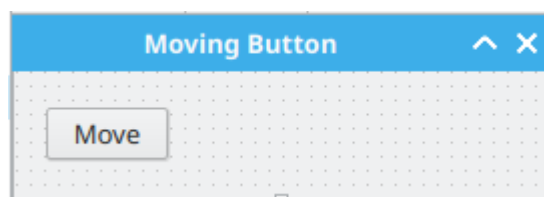
WHILE ... WEND

```
While x < 50
    'do something
Wend
```

To exit from a loop, **BREAK**. To exit from a sub, **RETURN**. To go to the next repetition, **CONTINUE**.

There is also the infinite loop, **DO ... LOOP**, and for items in a numbered list, **FOR EACH ... NEXT**.

The Moving Button



```
Public Sub Button1_Click()
    Do
        Repeat 'move to the right
            Button1.x += 1
            Wait 0.001
        Until Button1.x + Button1.w = FMain.Width

        Repeat 'move to the left
            Button1.x -= 1
```



```

        Wait 0.001
    Until Button1.x = 0
Loop

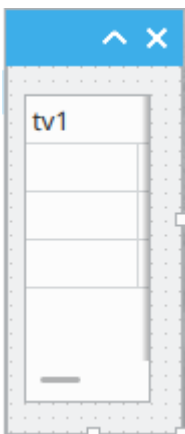
End

Public Sub Form_KeyPress()
    Quit
End

```

The program ends when you press a key. **Wait 0.001** delays progress for one-thousandth of a second. The delay allows the button to be redrawn. **X** and **Y** are traditionally used for *Across* and *Down*. The button moves from left to right and back, so it is Button1's **X** that we need to change. The button keeps moving to the right until its left side plus its width is equal to the width of the form. In other words, it stops moving right when the right side of the button meets the right edge of the form. After that we start subtracting from **X** until it meets the left edge of the form. Try changing the size of the window while the button is in motion: the button still moves to the edge.

A Tableview That Adds Up To 5 Numbers



Type any numbers, ending each with Enter to go to the next line. The total updates when you press Enter.

```

' Gambas class file

Public Sub Form_Open()
    tv1.Rows.Count = 6
    tv1.Columns.Count = 1
End

Public Sub tv1_Save(Row As Integer, Column As Integer, Value As String)

    Dim t As Integer
    tv1[Row, Column].Text = Value
    For i As Integer = 0 To 4

```

```

If IsNull(tv1[i, 0].text) Then Continue
If Not IsNumber(tv1[i, 0].text) Then Continue
t += Val(tv1[i, 0].text)
Next

tv1[5, 0].Background = Color.LightGray
tv1[5, 0].Foreground = Color.Blue
tv1[5, 0].RichText = "<b>" & t & "</b>" 'tv1[5, 0].text = t

End

Public Sub tv1_Click()
If tv1.Row < 5 Then tv1.Edit
End

```

Notice how the totals cell is blue on light grey, and the text is bold. "" & t & "" are tags each side of the total. Rich text is text with tags in it. The first tag is “switch bold on” and the second, with the slash, is “switch bold off”.

The Save event occurs when Enter or Return is pressed. After putting the typed number into the cell with **tv1[Row, Column].Text = Value**, the new total is put into the richtext of cell **tv1[5, 0]**.

Rich Text Tags

Gambas allows these tags in rich text. They are part of **HTML**, *HyperText Markup Language*, which web browsers use to display web pages. Each is switched on first, then switched off at the end, e.g. "<i>This is Bold Italic</i>".

<h1>, <h2>, <h3>, <h4>, <h5>, <h6> → Headlines	<sup> → Superscript
 → Bold font	<small> → Small
<i> → Italic	<p> → Paragraph
<s> → Crossed out	 → Line break
<u> → Underlined	<a> → Link
<sub> → Subscript	 → Font

The Font tag is used this way: "" & "" & t & ""

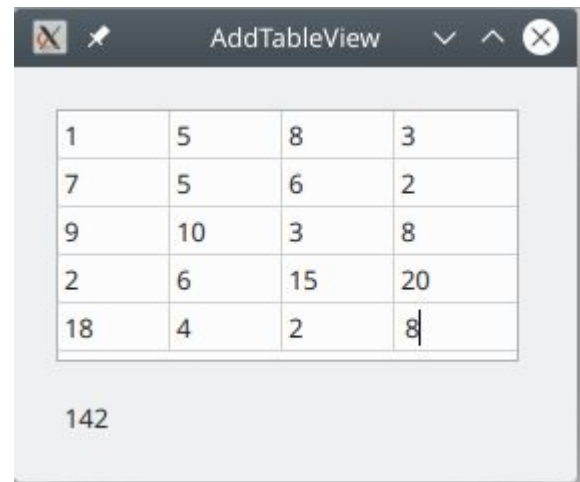
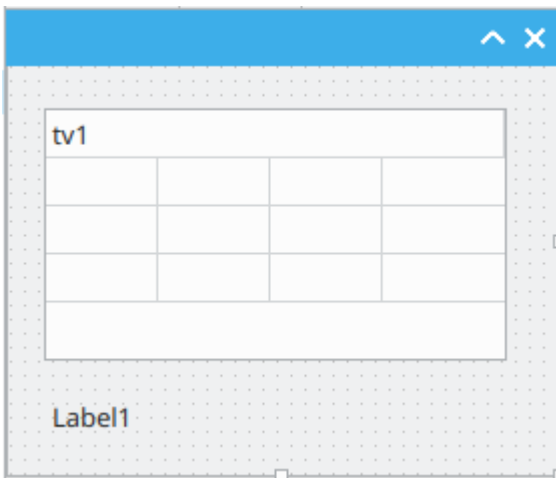
The Paragraph tag denotes paragraphs, which are usually separated by a blank line by browsers. It can used this way: <p align=right>Some Text</p> but I read on one web page, referring to HTML, “The align attribute on <p> tags is obsolete and shouldn't be used”. (<https://developer.mozilla.org/en-US/docs/Web/HTML/Element/p>). In the meantime, it works.

<tt> → single-line source text	<pre> → preformatted text (preserves spacing)
<hr> → horizontal line (no end tag needed)	 → unsorted list
 → list	 → list

JimJohnAnn will give bulleted points in a vertical list.

JimJohnAnn will give numbered points in a vertical list.

A Tableview That Adds Every Number In The Table



This program adds numbers as you type them in a 4 x 5 grid.

```
Public Sub Form_Open()  
    tv1.Rows.Count = 5  
    tv1.Columns.Count = 4  
End  
  
Public Sub tv1_Save(Row As Integer, Column As Integer, Value As String)  
    Dim i, j, t As Integer  
    tv1[Row, Column].Text = Value  
    For i = 0 To tv1.Rows.Max  
        For j = 0 To tv1.Columns.Max  
            If IsNull(tv1[i, j].text) Then Continue  
            If Not IsNumber(tv1[i, j].text) Then Continue  
            t += Val(tv1[i, j].text)  
        Next  
    Next  
    Label1.Text = t  
End  
  
Public Sub tv1_Click()  
    tv1.Edit  
End
```

The loops are inside each other. They mean “For every row, zip across the columns”.

```
For i = 0 To tv1.Rows.Max 'for every row going down...  
    For j = 0 To tv1.Columns.Max 'go across every column  
        ...  
    Next  
Next
```

Arrays, Lists, Sorting and Shuffling

There are times when you need not just separated memories with individual names like Speed, Distance, Time but a list of memories, a collection of variables, that can be numbered. So you might have **Speed[0]**, **Speed[1]**, **Speed[2]** and so on, all storing different speeds, or a list of several times that have names like **Time[0]**, **Time[1]**, **Time[2]** etc. This is called an array.

The elements (items) in the array are numbered, starting from zero, and you use square brackets. Teachers might have an array of student names, **Student[0]**, **Student[1]** ... **Student[Student.Max]**. Arrays have a count (e.g. **Student.Count**) and a max (**Student.Max**). Don't go past *Max* or you will be out of bounds. And that means detention after school for sure.

You can have arrays of just about anything. However, you need to create them with the **NEW** operator when you want them.

An array of strings:

```
Dim Names As New String[]
Names = ["John", "Mary", "Lucy", "Owen", "Peter", "Clare", "Thomas"]
```

An array of integers, using the array's *Add* method:

```
Dim Numbers As New Integer[]
'put the numbers 8 to 28 into an array; Numbers[0] is 8; Numbers[20] is 28
For i as integer = 8 to 28
    Numbers.add(i)
Next
```

Arrays have *Sort* and *Shuffle* methods. You can write any of these. **gb.descent** and its mate are constants in the **gb** component.

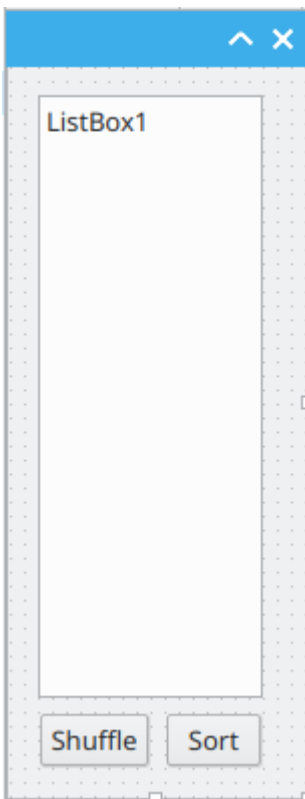
```
Names.sort
Numbers.sort

Names.sort(gb.descent)
Numbers.sort(gb.descent)

Names.shuffle
Numbers.shuffle
```

To see your array, put a listbox on your form then put your array into its list. The array on the right goes into the listbox's list on the left.

```
Listbox1.List = Names
Listbox1.List = Numbers
```



The buttons are called *bShuffle* and *bSort*.

```
Public s As New String[]

Public Sub Form_Open()
    s = ["Apple", "Banana", "Carrot", "Dragonfruit", "Elderberry", "Fig",
"Grape", "Honeydew", "Imbe", "Jackfruit", "Kiwi", "Lime"]
    ListBox1.List = s
End

Public Sub bSort_Click()
    s = ListBox1.List 'copy the list into array s[]
    ListBox1.List = s.Sort() 'put the sorted s[] into the listbox's list
End

Public Sub bShuffle_Click()
    s = ListBox1.List 'copy the list into array s[]
    s.Shuffle 'shuffle the array s[]
    ListBox1.List = s 'put s into the listbox's list
End
```

Notice that you cannot say *ListBox1.List.Shuffle*, even though *ListBox1*'s *List* property acts like an array. Yes, it is an array. No, it doesn't come with the *Shuffle* method.

The shuffle button has an extra line in its *Click* handler compared with the sort button. *s.Sort()* is a thing, a **function**. *s.Shuffle* is a method. It is not a thing but a process, a **procedure**. It is a sub that does not return any value when it is done. You cannot put it into something. If you tried

`ListBox1.List = s.shuffle()` you would get an error message. The Gambas help shows how they are different:

```
Function Sort ( [ Mode As Integer ] ) As String[]
```

Sort the array and return it.

```
Sub Shuffle ( )
```

SINCE 3.13

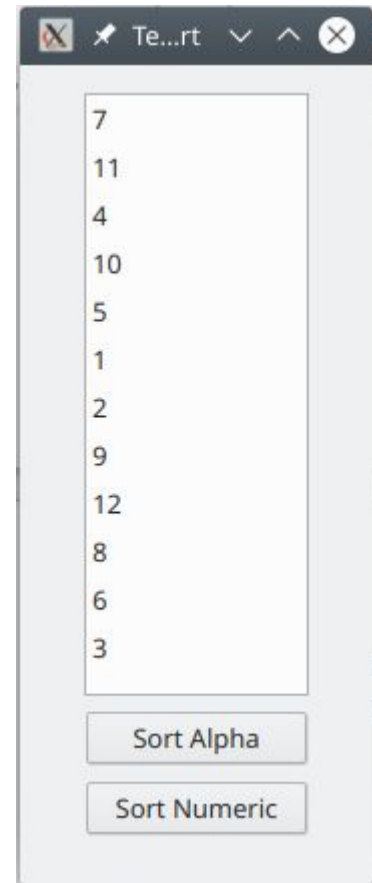
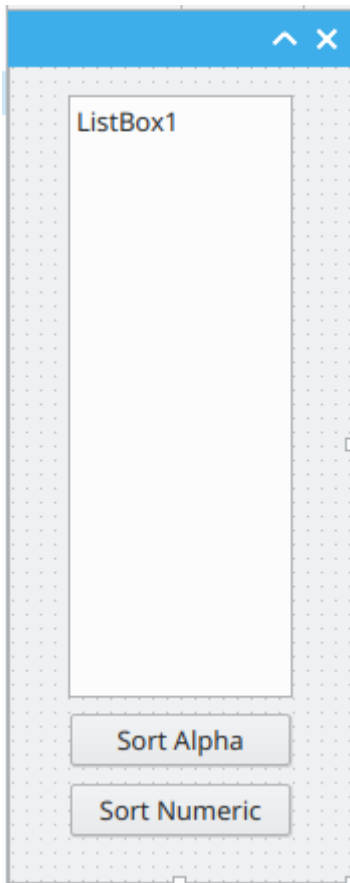
Randomly shuffle the array

Sorting alphabetically and Sorting Numerically

A listbox with numbers needs to be sorted numerically. Sorting number 1 to 12 alphabetically will give you 1, 10, 11, 12, 2, 3, 4, 5, 6, 7, 8, 9 because alphabetically “10” comes before “2”. Copy the list into an integer array and sort that:

```
Public Sub bSort_Click()  
  Dim x As New Integer[]  
  x = ListBox1.List  
  x.Sort  
  ListBox1.List = x  
End
```

You can see this in action:



```
Public z As New Integer[]
Public s As New String[]

Public Sub Form_Open()
    For i As Integer = 1 To 12
        z.Add(i)
    Next
    z.Shuffle
    ListBox1.List = z
End

Public Sub Button1_Click()
    s = ListBox1.List
    ListBox1.List = s.Sort()
End

Public Sub Button2_Click()
    z = ListBox1.List
    ListBox1.List = z.Sort()
End
```

Strings

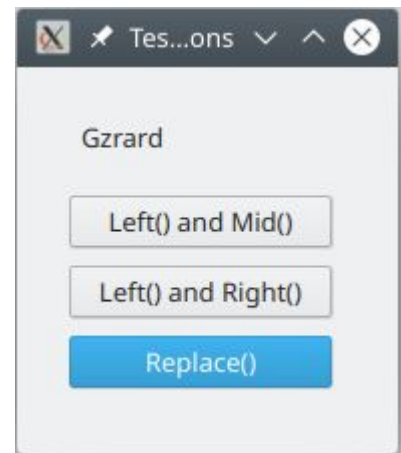
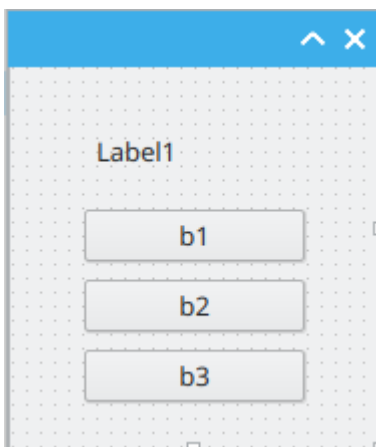
Strings can be treated as arrays of characters. So, if **Nm = "Gerard"** then **Nm[0]** is **G**, **Nm[1]** is **e**, **Nm[2]** is **r** and so on.

However, you cannot change letters like **Nm[2]="x"** to make *Gerard* become *Gexard*. You can *get* the letter, but you can't *put* something into it. You would have to use some of the wonderful functions that strings have. You can do many things with strings. You could use any of these:

Nm = Left(Nm,1) & "x" & Mid(Nm,3)

Nm = Left(Nm,1) & "x" & Right(Nm,4)

Nm = Replace(Nm, "e", "x")



```
Public FirstName As String = "Gerard"

Public Sub b1_Click()
    Label1.Text = Left(FirstName, 1) & "x" & Mid(FirstName, 3)
End

Public Sub b2_Click()
    Label1.Text = Left(FirstName, 1) & "y" & Right(FirstName, 4)
End

Public Sub b3_Click()
    Label1.Text = Replace(FirstName, "e", "z")
End
```

Arrays of Arrays

You can have memories arranged in a list. They would all have the same name, but be numbered like **X[0]**, **X[1]**, **X[2]** etc:

[0]	[1]	[2]	[3]	[4]
-----	-----	-----	-----	-----

You can have memories arranged in a grid (square or rectangle) and refer to them by row and column. This is how the cells are numbered in a GridView or TableView:

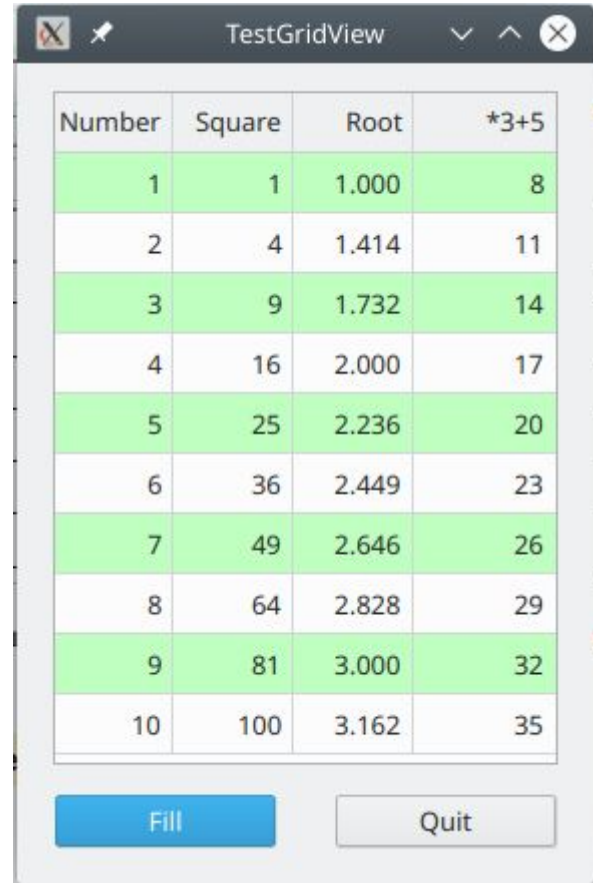
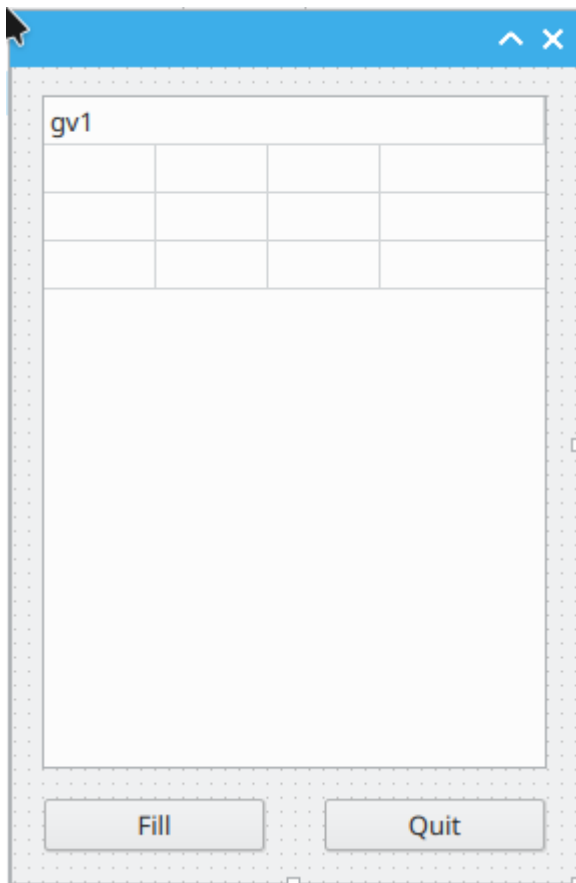
[0,0]	[0,1]	[0,3]	[0,4]
[1,0]	[1,1]	[1,3]	[1,4]
[2,0]	[2,1]	[2,3]	[2,4]

You can have them arranged in a cube or prism, so that there are like layers of rectangles. The third number in brackets tells which layer. [Row, Column, Layer]

[0,0,2]	[0,1,2]	[0,3,2]	[0,4,2]
[1,0,2]	[1,1,2]	[1,3,2]	[1,4,2]
[0,0,1]	[0,1,1]	[0,3,1]	[0,4,1]
[1,0,1]	[1,1,1]	[1,3,1]	[1,4,1]
[0,0,0]	[0,1,0]	[0,3,0]	[0,4,0]
[1,0,0]	[1,1,0]	[1,3,0]	[1,4,0]
[2,0,0]	[2,1,0]	[2,3,0]	[2,4,0]

Yes, you can have multi-dimensional arrays, but simple list arrays are the ones most often used.

Table of Doubles, Squares and Square Roots



This program fills a grid with calculations for the numbers one to ten. It also shows how to adjust properties of a gridview—its columns, rows and cells.

```
Public Sub Form_Open()  
    Me.Show  
    With gv1  
        .Header = GridView.Horizontal  
        .Columns.Count = 4  
        .Columns[0].Width = 50  
        .Columns[1].Width = 60  
        .Columns[2].Width = 60  
        .Columns[3].Width = 50  
        .Columns[0].Text = ("Number")  
        .Columns[1].Text = ("Square")  
        .Columns[2].Text = ("Root")  
        .Columns[3].Text = ("*3+5")  
        For i As Integer = 0 To 3  
            .Columns[i].Alignment = Align.Right  
        Next  
        .Padding = 5  
    End With  
End  
  
Public Sub b1_Click()  
    gv1.Rows.Count = 10  
End
```

```

Public Sub bQuit_Click()
    Quit
End

Public Sub gv1_Data(Row As Integer, Column As Integer)

    Dim x As Integer = Row + 1
    Select Case Column
        Case 0
            gv1.Data.Text = x
        Case 1
            gv1.Data.Text = x ^ 2
        Case 2
            gv1.Data.Text = Format(Sqr(x), "##.000")
        Case 3
            gv1.Data.Text = x * 3 + 5
    End Select
    If Row Mod 2 = 0 Then gv1.Data.Background = &HBFFFFBF
End

```

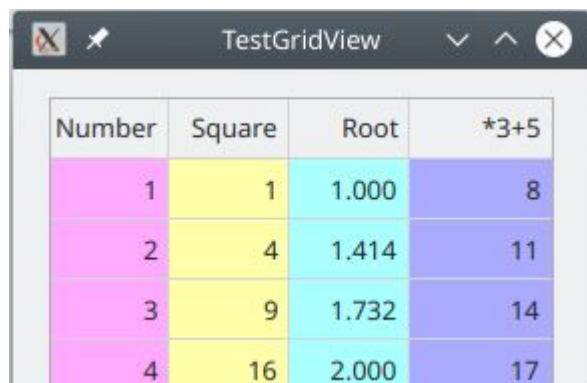
Things to notice are the **With ... End With** lines, setting up the gridview when the form opens, and the **_Data()** event to fill each cell. `gv1.Rows.Count = 10` is enough to trigger the `Data()` event for every cell in those ten rows.

The `_Data()` event occurs when the cell has to be painted. The filling of the cells could be done when the *Fill* button is clicked, but then we would have to use nested *For* statements to count down the rows and across the columns. Gambas already has to do this, because it has to paint each cell. The `_Data()` event happens for each cell in each row, so why not put the text into the cells then?

If `Row Mod 2 = 0 Then gv1.Data.Background = &HBFFFFBF` sets the background of the cell to peppermint green if the row number has a remainder of zero when divided by 2. **Mod** means “the remainder when you divide by ...”. For example 3 Mod 2 is 1, so row 3 has a white background. 4 Mod 2 is 0 because four divided by two equals two remainder zero, so row 4 has a green background.

We could alternate two colours, pink and green, with
gv1.Data.Background = If(Row Mod 2 = 0, &HBFFFFBF, &HFFCFCF)

To colour the columns, try this in the `tv1_Data()` event:



Number	Square	Root	*3+5
1	1	1.000	8
2	4	1.414	11
3	9	1.732	14
4	16	2.000	17

`gv1.Data.Background = Choose(Column + 1, &FFAAFF, &FFFFAA, &AAFFFF, &AAAAFF)`

`Format(Sqr(x), "##.000")` is an interesting expression. The Format function takes a floating point number like 1.41421356237 and formats it according to the pattern supplied. `"##.000"` means *two digits if you need them, a dot, and three decimal places using zeros*. What number to format? `Sqr(x)`. This is the square root of the number `x`. The square root of two appears as 1.414.

Format Function

+	Print the sign of the number.
-	Print the sign of the number only if it is negative.
#	Print a digit only if necessary.
0	Always print a digit, padding with a zero if necessary.
.	Print the decimal separator.
,	Print the thousand separators.
%	Multiply the number by 100 and print a per-cent sign.
E	Introduces the exponential part of a Float number. The sign of the exponent is always printed.

There are many more symbols for formatting dates and currency.

The Game of Moo



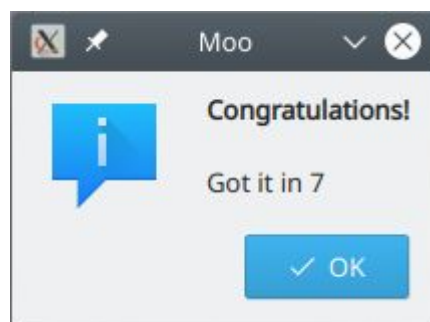
en.wikipedia.org

To play the game of Moo, also called Bulls and Cows, one person (the computer!) chooses a mystery number. It has 4 digits, and all digits are different. Wikipedia says, “The modern game with pegs was invented in 1970 by Mordecai Meirowitz, an Israeli postmaster and telecommunications expert. It resembles an earlier pencil and paper game called Bulls and Cows that may date back a century or more.” “Moo was written in 1970 by J. M. Grochow at MIT in the PL/I computer language.”

After each guess the computer tells you how many bulls you scored and how many cows. A bull is a digit in your number that is in its correct place in the mystery number. A cow is a digit that is present in the mystery number but is in its wrong place. Thus you are aiming for **BBBB**, all four digits in their correct places. **CCCC** means you have the correct digits but they are not in their right

places. **BBC** means two digits are correctly placed, one of the other digits is in the number but in the wrong place, and another digit is not in the number at all. Some people play by the rule that you win if you guess it in ten or fewer turns. Bulls are listed first and cows second. You are not told which digits are the bulls or which are cows.

You need 2 textboxes, a gridview called *gvHistory*, and a button called *bNewGame* with the text property “New Game”.



The *New Game* button is initially invisible.

```
Public MyNumber As String
Public Count As Integer

Public Sub Form_Open()
    gvHistory.Columns.Count = 2
    ChooseNumber
End

Public Sub ChooseNumber()
    Dim Digits As New String[]
    Dim i, p1, p2 As Integer
    Dim d As String
    For i = 0 To 9
        Digits.Add(Str(i))
    Next
    Digits.Shuffle
    MyNumber = Digits[0] & Digits[1] & Digits[2] & Digits[3]
End

Public Sub EvaluateGuess()
    Dim s As String
    Dim i, j As Integer
    Dim YourGuess As String = tbGuess.text
    Count += 1
    For i = 0 To 3 'look for bulls
        If YourGuess[i] = MyNumber[i] Then s &= "B"
    Next
    For i = 0 To 3 'look for cows
        For j = 0 To 3
            If i <> j And YourGuess[i] = MyNumber[j] Then s &= "C"
        Next
    Next
    tbReply.Text = s
    gvHistory.Rows.Count += 1
    gvHistory[gvHistory.Rows.max, 0].text = YourGuess
    gvHistory[gvHistory.Rows.max, 1].text = s
    If s = "BBBB" Then Congratulate
    tbGuess.SelectAll
End

Public Sub Congratulate()
    Message("<b>Congratulations!</b><br><br>Got it in " & count)
    FMain.Background = Color.Yellow '&FFFF00
    bNewGame.Visible = True
End

Public Sub bNewGame_Click()
    FMain.Background = Color.Default
    bNewGame.Visible = False
    gvHistory.Rows.Count = 0
    Count = 0
    ChooseNumber
    tbReply.text = ""
    tbGuess.text = ""
    tbGuess.SetFocus
End

Public Sub tbGuess_Change()
```

```
If Len(tbGuess.text) = 4 Then EvaluateGuess  
End
```

Now for the post-mortem:

There are two public variables. **MyNumber** is the computer's secret number. **Count** keeps count of how many guesses.

Form_Open() On startup, set gridview to 2 columns and choose the secret number.

Public Sub ChooseNumber() Put digits 0 to 9 in a 10-item array called Digits and shuffle. The secret number is the first four digits. They will be random and none is repeated.

Public Sub EvaluateGuess() When you evaluate a guess, it is one more guess so add 1 to *Count*. Look for Bulls: Is the first character in your guess the same as the first character in the number? Check second, third and fourth characters too. Each time two characters match, take what s was and add a *B* to the end of it using **s &= "B"**.

In looking for cows, *i* goes through 0, 1, 2, 3, looking through your number each time. For each one of those digits in your number, check all the digits in the secret number looking for a match, but disregard where the position numbers are the same (like the third digit in the secret number and the third digit in my guess), for that is a bull and has already been found.

S is a variable that contains BBBB or BBCC or B or whatever.

Add a row to the history gridview and put the guess in the first column and its evaluation into the second column.

Public Sub Congratulate() Show a message saying "Congratulations! You got it in 8" or whatever. Change the form's colour to yellow. Make the *New Game* button visible.

Public Sub bNewGame_Click() To start a new game, remove the yellow colour, hide the New Game button, remove everything from the history gridview, set the count of guesses back to zero, choose another number, blank out the two textboxes, and set the focus to the textbox where you type in a guess so you are ready to start typing.

Public Sub tbGuess_Change() When the text in the Guess textbox changes because you have typed another digit, check the length of what is typed and if it is 4 characters long, evaluate the guess.

Adding an "I Give Up" Feature

Let us make it that typing a question mark means "I give up—tell me the answer". We need as new event, `tbGuess_KeyPress()`. As soon as a question mark is typed, this handler will check the static class, `Key`, to see if the character typed was "?". If so, message us the correct answer and start a new game.

Starting a new game is exactly what we have in the `_Click` handler for the New Game button. This code needs to be taken out of the `_Click` handler and put in a sub of its own. We can call on this

NewGame sub when the button is clicked or when the user gives up by typing “?”. Here is the rearranged and extra code:

```
Public Sub bNewGame_Click()
    NewGame
End

Public Sub NewGame()

    FMain.Background = Color.Default
    bNewGame.Visible = False
    gvHistory.Rows.Count = 0
    Count = 0
    ChooseNumber
    tbReply.text = ""
    tbGuess.text = ""
    tbGuess.SetFocus

End

Public Sub tbGuess_KeyPress()

    If Key.Text = "?" Then
        Message("My number was " & MyNumber)
        NewGame
        Stop Event
    Endif

End
```

Stop Event is there to prevent the question mark appearing in the *tbGuess* textbox. It is not necessary.

The important principle is (and you can memorise this or take this to the bank) *if you want to refer to the same lines of code in two or more places, put them in their own sub and call on them by name.*

Assignment Operators

<code>x = x+2</code>	Add 2 to whatever x used to be, then put the answer back into x.
<code>x += 2</code>	The same thing: x becomes whatever it was (=), plus (+) 2.
<code>x = x * 4</code>	Multiply x by 4, and put the answer back into x.
<code>x *= 4</code>	The same thing: x becomes whatever it was (=), times (*) 4.
<code>s = s & "abc"</code>	s becomes whatever s was and (&) “abc” tacked onto the end of it.
<code>s &= "abc"</code>	The same thing: s becomes whatever it was (=) and (&) “abc” onto the end of it.

There are numeric operators including ^ to mean “raised to the power of”. Boolean operators are AND, OR and NOT. There are others but these are the most used and useful.

The Game of Animal

The Animal Game, in which the computer tries to guess the animal you are thinking of by asking you questions, was around well before people had their own computers. It dates to at least the early 1970s. The author of website <http://www.animalgame.com/> says that he/she first saw it in "101 BASIC Computer Games" (ed. by David H. Ahl - Maynard, Mass., Digital Equipment, 1973.) I remember that book. The game was originally developed by Arthur Luehrmann at Dartmouth College. <http://www.smalltime.com/Dictator> has an online version where you guess the dictator instead of an animal.

The computer starts by knowing only two animals, BIRD and FISH and only one question that can tell the two apart, "Does it swim?". Questions can be answered YES or NO. If your animal is neither of these, you are asked for a question that would identify your animal. Gradually a list of animals and questions is built up, and the computer becomes smarter and smarter. It is a simple form of Artificial Intelligence (AI). Some wit once said, "Natural stupidity beats artificial intelligence every time".

To someone starting computing, it may be too complicated. However, there might be bits here and there—saving and loading text from a text file, or the managing of big programs by breaking them into small, meaningful subs, for example—that would be useful. Let it wash over you and get a general feel of things. Practise debugging when you have made a typing error and the program does not run. I was intrigued by it back in the '70's, and it is too good not to include.

Here is a typical dialogue:

Are you thinking of an animal?.....*Yes (No ends the program.)*
It is a fish.....*No.*
The animal you were thinking of was a ...?.....*Dog*
Please type in a question that would distinguish a dog from a fish.....*Does it have legs?*
For a dog the answer would be.....*Yes*
Are you thinking of an animal?.....*Yes*
Can it swim?.....*Yes*
Does it have legs?.....*Yes*
It is a dog.....*Yes*

Here is a data file. It is text only. At the end of every question is the line to go next for yes or no.

0. Can it swim?/1/2
1. Does it have legs?/3/4
2. Does it have 2 big ears?/5/6
3. dog
4. Does it blow air?/9/10
5. rabbit
6. It is little and does it bite you?/7/8
7. mosquito
8. Is it small?/11/12

9. whale
10. fish
11. fly
12. bird

Play **Guess the Animal**

★ ★ ★ ★

Are you thinking of an animal?

The animal you were thinking of was a ...

Please type in a question that would distinguish a 1111 from a 2222

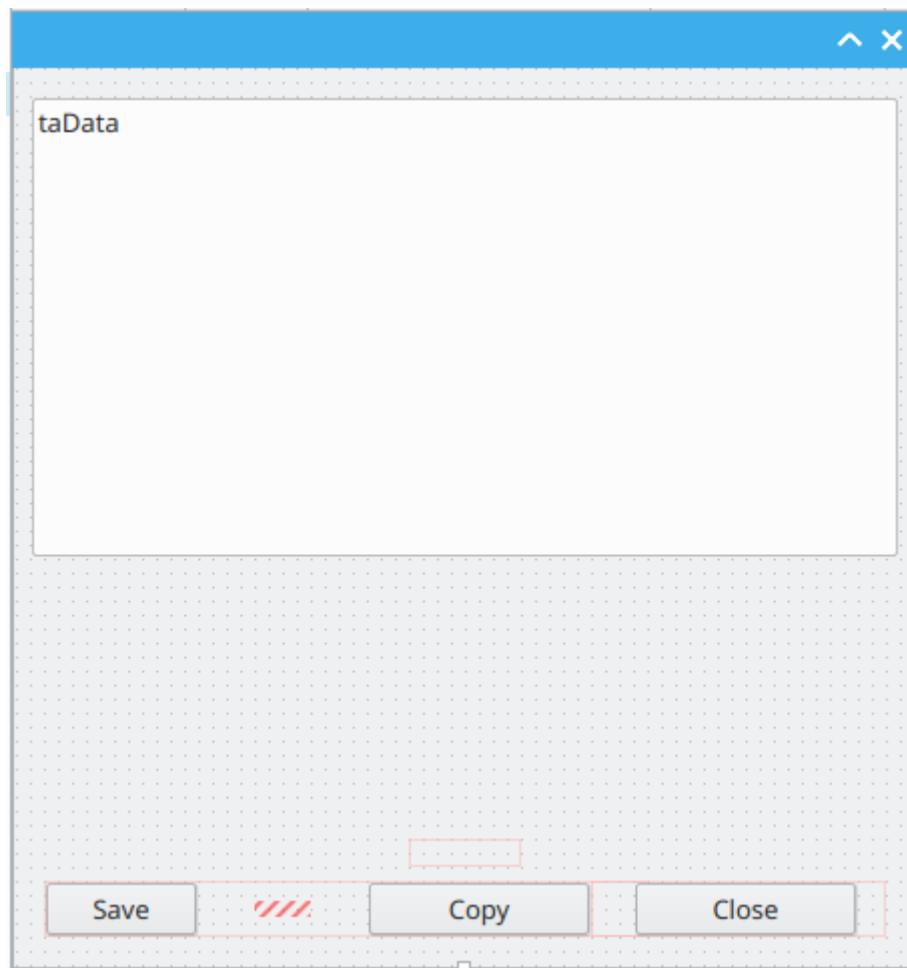
(e.g. Does it have...? Can it...? Is it...?):

labQuestion

For a 1111 the answer would be...

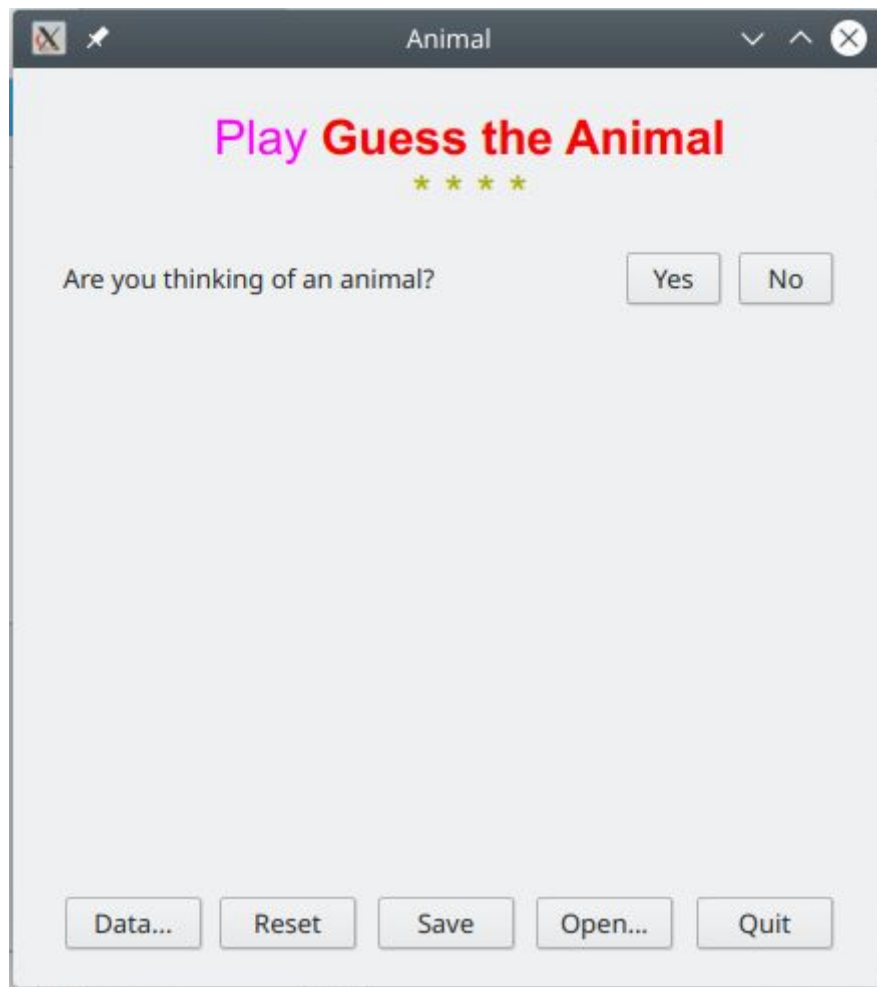
The code refers to these objects by the following names. Here goes. The coloured text at the top is *TextLabel1*. The “Are you thinking of an animal?” line has label *LabPrompt* and buttons *bYes* and *bNo*. The “The animal you were thinking of was a ...” has label *LabPromptForAnimal* and textbox *tbNewAnimal*. The “Please type in a question...” line has label *LabQuestionPrompt*. That is followed by the long textbox *tbQuestion*. Under it is another label *LabQuestion*. The last line, “For a 1111 the answer would be...” has label *LabPrompt2* and buttons *bYes2* and *bNo2*. The bottom line of buttons are named, from the left, *bShowData*, *bReset*, *bSave*, *bOpen* and *bQuit*.

Next is another form that allows you to look at the data. It shows by clicking the *Data...* button.



The form is called *FData*. It will automatically adjust the positions of objects on it when it is resized by dragging its corner handles. To do this, set its *Arrangement* property to *Vertical*. This makes its objects stack from top to bottom. The text area, *taData*, has its *expand* property set to *True* so that its size will expand to fill the space available.

The three buttons, *bSave*, *bCopy* and *bClose*, are in a red rectangle the is a *HBox*. *HBoxes* spread their contents horizontally. This one is called *HBox1*. There are a couple of panels called *Panel1* and *Panel2*. One is above the hbox and one is between the *Copy* and *Close* buttons, to separate them. There is a neat little spring called *Spring1* that pushes the *Save* button to the left and the *Copy* button, tiny separator panel and *Close* button to the right.



When the game starts, these are the only visible objects. All others have their visible property set to False. The program sets their visibility to true later. There are four areas. Depending on the stage of the game you are up to, one by one they are made visible and the others are hidden.

Look again at the sample data file:

0. Can it swim?/1/2
1. Does it have legs?/3/4
2. Does it have 2 big ears?/5/6
3. dog
4. Does it blow air?/9/10
5. rabbit
6. It is little and does it bite you?/7/8
7. mosquito
8. Is it small?/11/12
9. whale
10. fish
11. fly
12. bird

If the line counter *L* arrives at a line with an animal name and that name is rejected, the program replaces that line by the new question you give it and the new animal and the wrong animal are added to the end of the list.

The variables declared as *Public* and *Private* right at the start are there so that they can be accessed by several different subs. They don't just last for the duration of the sub. They belong to the form rather than any particular sub. (*Z[]* needs to be accessed by the other form, too, so it is *Public*.)

```
' Gambas class file
```

```
Public z As New String[] 'database
Private Right As Integer 'line to go to if Yes is clicked
Private Wrong As Integer 'line to go to if No is clicked
Private QuestionPrompt As String
Private AnimalPrompt As String
Private NewAnimal As String
Private AskedAQuestion As Boolean 'true if we've just said, "Is it a...?"
Private NewQuestion As String
Private L As Integer 'which line in database?
Private FromYesLink As Boolean

Public Sub bQuit_Click()
    Quit
End

Public Sub bShowData_Click()
    FData.ShowModal
    FMain.SetFocus
End

Public Sub Form_Open()
    'Make sure the NoTabFocus property is set to true for all buttons, otherwise
    one will be highlighted when you start.
    QuestionPrompt = "Please type in a question that would distinguish
a<br><b>1111</b> from a <b>2222.</b> <br><br>(e.g. Does it have...? Can
it...? Is it...? ):"
    AnimalPrompt = "For a <b>1111</b> the answer would be..."
    StartGame(True) 'clears data too
End

Public Sub bSave_Click()
    SaveData
End

Public Sub SaveData()

    Dialog.Filter = ["*.txt", "Text Files"]
    If Dialog.SaveFile() Then Return
    File.Name = File.BaseName & ".txt"
    File.Save(Dialog.Path, z.Join(Chr(13)))
    FMain.SetFocus
Catch
    Message.Info(Error.Text)
End

Public Sub bOpen_Click()
```

```

Dialog.Filter = [ "*.txt", "Text Files" ]
If Dialog.OpenFile() Then Return
Dim s As String = File.Load(Dialog.Path)
z = Split(s, Chr(13))
FMain.SetFocus
Catch
    Message.Info(Error.Text)

End

Public Sub ShowStep(Stage As Integer) '1, 2, 3 or 4

    labPrompt.Visible = (Stage = 1)
    bYes.Visible = (Stage = 1)
    bNo.Visible = (Stage = 1)

    labPromptForAnimal.Visible = (Stage = 2)
    tbNewAnimal.Visible = (Stage = 2)

    labQuestionPrompt.visible = (Stage = 3)
    tbQuestion.Visible = (Stage = 3)

    labQuestion.Visible = (Stage = 4)
    labPrompt2.Visible = (Stage = 4)
    bYes2.Visible = (Stage = 4)
    bNo2.Visible = (Stage = 4)
    Select Case Stage
        Case 2
            tbNewAnimal.SetFocus
        Case 3
            tbQuestion.SetFocus
        Case Else
            FMain.SetFocus 'a futile attempt to stop buttons being highlighted
    End Select

End

Public Sub bReset_Click()
    StartGame(True) 'clear all data too
    FMain.SetFocus
End

Public Sub StartGame(ClearDataToo As Boolean)

    If ClearDataToo Then
        z.Clear
        z.add("Can it swim?/1/2")
        z.add("fish")
        z.Add("bird")
    Endif
    L = 0
    Right = 0
    Wrong = 0
    AskedAQuestion = True
    labPrompt.text = "Are you thinking of an animal?"
    tbQuestion.Text = ""
    tbNewAnimal.Text = ""
    ShowStep(1)

End

```

```

Public Sub tbNewAnimal_KeyPress() 'Enter should cause the LostFocus event
    If Key.Code = Key.Enter Or Key.Code = Key.Return Then FMain.SetFocus
End

Public Sub tbNewAnimal_LostFocus() 'by pressing Enter or clicking elsewhere

    NewAnimal = LCase(tbNewAnimal.text)
    If IsNull(NewAnimal) Then Return 'user pressed enter without typing
    anything; don't proceed.
    Dim s As String = Replace(QuestionPrompt, "1111", NewAnimal) 'Please type in
    a question...
    s = Replace(s, "2222", z[L])
    labQuestionPrompt.text = s
    ShowStep(3)

End

Public Sub tbQuestion_KeyPress()
    If Key.Code = Key.Enter Or Key.Code = Key.Return Then FMain.SetFocus
End

Public Sub tbQuestion_LostFocus()

    NewQuestion = tbQuestion.Text
    If IsNull(NewQuestion) Then Return 'user pressed enter without typing
    anything; don't proceed.
    NewQuestion = UCase(NewQuestion[0]) & Right(NewQuestion, -1) 'capitalise
    first letter
    If Right(NewQuestion, 1) <> "?" Then NewQuestion &= "?"
    labQuestion.Text = NewQuestion
    labPrompt2.Text = Replace(AnimalPrompt, "1111", NewAnimal) 'For a gorilla
    the answer would be...
    ShowStep(4)

End

Public Sub AskQuestion()

    Dim k As New String[]
    k = Split(z[L], "/")
    labPrompt.text = k[0]
    Right = Val(k[1])
    Wrong = Val(k[2])
    AskedAQuestion = True
    tbNewAnimal.SetFocus

End

Public Sub MakeGuess()
    labPrompt.Text = "It is " & If(InStr(Left(z[L], 1), "aeiou") > 0, "an ", "a
    ") & z[L] & "."
    AskedAQuestion = False
End

Public Sub bYes_Click() 'Yes has been clicked

    If AskedAQuestion Then
        L = Right 'take the right fork
        If InStr(z[L], "/") > 0 Then AskQuestion Else MakeGuess

```

```

Else 'made a guess...
    StartGame(False)
Endif
FMain.SetFocus

End

Public Sub bNo_Click() 'No has been clicked

    If labPrompt.text = "Are you thinking of an animal?" Then Quit 'If you won't
    play, I quit!
    If AskedAQuestion Then
        L = Wrong 'take the left fork
        If InStr(z[L], "/") > 0 Then AskQuestion Else MakeGuess
    Else 'made a wrong guess, so add an animal
        ShowStep(2)
    Endif

End

Public Sub bYes2_Click()

    Dim s As String = NewQuestion & "/" & Str(z.max + 1) & "/" & Str(z.max + 2)

    z.Add(NewAnimal) 'the right animal
    z.Add(z[L]) 'the wrong animal
    z[L] = s 'replace the earlier wrong animal with the new question
    StartGame(False)

End

Public Sub bNo2_Click()

    Dim s As String = NewQuestion & "/" & Str(z.max + 1) & "/" & Str(z.max + 2)
    z.Add(z[Wrong]) 'the wrong animal
    z.Add(NewAnimal) 'the right animal
    z[Wrong] = s 'replace the earlier wrong animal with the new question
    StartGame(False)

End

```

The code for the *FData* form is:

```

Public Sub Form_Open()
    taData.Text = FMain.z.Join(Chr(13)) 'Chr(13) is Return
End

Public Sub bClose_Click()
    Me.close
End

Public Sub bCopy_Click()
    Clipboard.Copy(taData.text)
End

Public Sub bSave_Click()
    FMain.SaveData
End

```

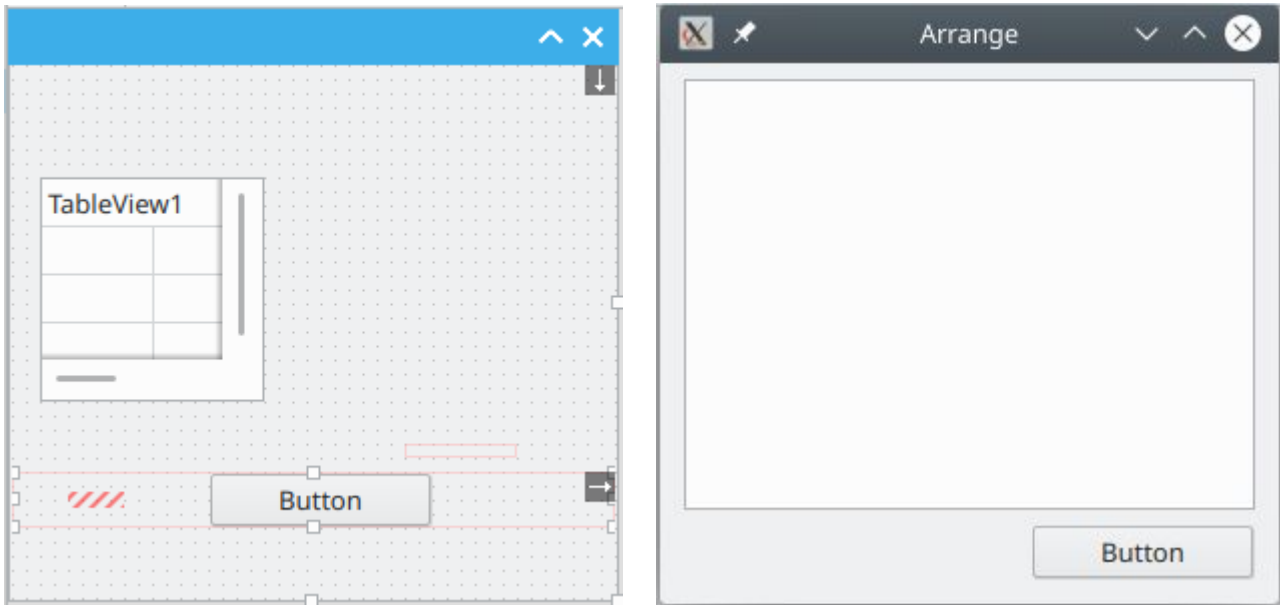

Notes:

Fmain.SetFocus	When you click a button it highlights. To make the highlight go away at the end of doing whatever it does, set the focus on the form. It seems to be needed.
Dim z as new string[] z = split("a/b/c" , "/")	z will get three rows. z[0] = "a", z[1] = "b", z[2] = "c" You can specify the separator, and it may be more than one character long.
Dim s as string s= z.join(" ")	Joins the array z into one string with spaces in between
IsNull(s)	This is true if the string s is empty (no characters in it)
Chr(13)	Every character has a code number. 13 is the number for Return. If you are interested, A is chr(65) and a is char(97). The code is called ASCII.
	<p>Public Sub ShowStep(Stage As Integer) labPrompt.Visible = (Stage = 1)</p> <p>This sub needs a number when it gets called. So you might say ShowStep(1) or ShowStep(2).</p> <p>If LabPrompt.Visible is true the label will be visible. If false, it is invisible. LabPrompt's visibility depends on the number you supply being 1.</p>
if InStr(z[L], "/") > 0 Then	If the string z[L] has a slash in it, then do something. InStr (LookInThis, ForThis) gives the position of the second string in the first.

Making Controls Expand When a Form is Resized

Gridviews and TableViews are often made to stretch and shrink when the window they are in is resized. A form, which is a window, knows how to resize and arrange the controls that are in it. Even when the form opens any controls that can be arranged and expanded will be.

Make a small form with one TableView.



Set the **arrangement** property of the form to Vertical. Set the **expand** property of the tableview to True. Run the program (f5). Change the size of the window and notice how the size of the tableview changes with it.

Adjust the **padding** property of the form to 8. Run the program again (F5). The space between the tableview and the edge of the form has increased. The margin inside the form is the padding. Table cells also have padding.

Add a button under the tableview. Run the program again. Move the button to one side of the tableview. Again run the program. Change the arrangement property to Horizontal and try again. Change it back to Vertical.

Delete the button. Add a HBox. Add a button to the HBox. (Alternately, Right-click the button and choose “*Embed in a container*”, then right-click the container—which is a panel—and *Change into...* a HBox.)

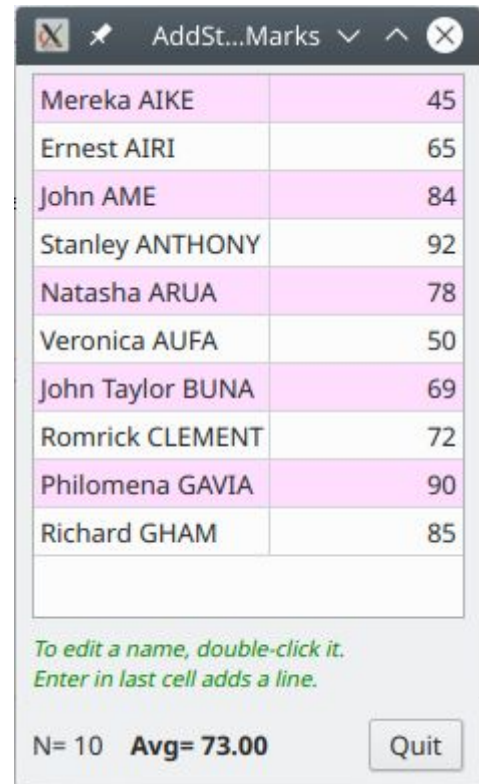
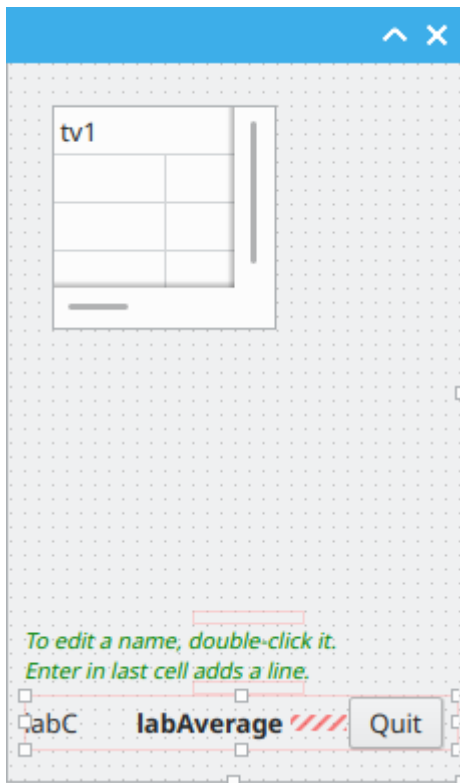
Controls in a HBox are arranged horizontally.

The spring will push the button to the right as far as it will go. Experiment with and without the spring.

If you make the button wider or narrower it stays whatever width you give it in the HBox.

The button expands vertically to fill the HBox from top to bottom. Change HBox’s height and see. HBoxes expand their controls to fill their height. In panels the button stays the same height but you cannot use springs.

A Spreadsheet to Average Student Marks



tv1 is a tableview. Its **expand** property is set to True.

At the bottom is a HBox. Inside, from left to right, is a label *labC*, a boldface label *labAverage*, a spring, and a Quit button called *bQuit* whose text property is “Quit”.

As each number is entered the average is recalculated. Blank cells are skipped.

LabC shows the count and *labAverage* the average.

```
Public Names As New String[]
Public Scores As New Float[]

Public Sub bQuit_Click()
    Quit
End

Public Sub Form_Open()

    Names = ["Mereka AIKE", "Ernest AIRI", "John AME", "Stanley ANTHONY",
"Natasha AUA", "Veronica AUFA", "John Taylor BUNA", "Romrick CLEMENT",
"Philomena GAVIA", "Richard GHAM"]
    tv1.Rows.Count = Names.count
    Scores.Resize(Names.Count)
    tv1.Columns.Count = 2
    tv1.Columns[1].Alignment = Align.Right

End

Public Sub CalculateAverage()
```

```

Dim i, n As Integer
Dim t As Float

For i = 0 To Scores.Max
    If Scores[i] = -1 Or IsNull(tv1[i, 1].text) Then Continue 'skip new but
unfilled-in lines
    n += 1 'number of scores
    t += Scores[i] 'total
Next
If n = 0 Then Return
labC.Text = "N= " & n
labAverage.Text = "Avg= " & Format(t / n, "#0.00")

End

Public Sub tv1_Click()
    If tv1.Column = 1 Then tv1.Edit 'numbers column is editable; Enter goes down
End

Public Sub tv1_Activate() 'double-clicked a cell
    If tv1.Column = 0 Then tv1.Edit 'edit a name
End

Public Sub tv1_Save(Row As Integer, Column As Integer, Value As String)

    tv1[Row, Column].text = Value
    If Column = 1 Then
        Scores[Row] = Value
        CalculateAverage
    Else
        Names[Row] = Value
    Endif

End

Public Sub tv1_Data(Row As Integer, Column As Integer)

    If Column = 0 Then tv1[row, 0].text = Names[Row]
    If Row Mod 2 = 0 Then tv1[Row, Column].Background = &hDDDDFF 'light blue
    tv1.Columns[0].Width = -1 'Automatically set width based on contents

End

Public Sub tv1_Insert()

    tv1.Rows.Count += 1
    Scores.Add(-1)
    Names.Add("")
    tv1.MoveTo(tv1.Rows.max, 0)
    tv1.Edit

End

```

Right at the start two public arrays are created. *Names[]* is a list of the student names. *Scores[]* is a list of their results. They match: the first mark goes with the first name, the second mark with the second name, and so on.

The **tv1_Data(Row As Integer, Column As Integer)** event fires every time a cell needs to be redrawn. It supplies you with Row and Column. It can be thought of as painting the cell.

There is a special consideration with the **_DATA** event: it does not paint cells that it doesn't need to. This is great for displaying large numbers of lines. If there are 100,000 lines and you are only showing 15 of them, only the cells on those 15 lines fire the **_DATA** event, not all hundred thousand. Be careful if you used the **_DATA** event to put the numbers into the cells! There may be data in only 15 of those 100,000 cells. Here there is no problem, because we are typing the numbers ourselves and every time we finish typing a new number it is put into the cell in the **_SAVE** event. (**tv1[Row, Column].text = Value**). When we come to putting values in using the **_DATA** event from a database, though, we shall only put data into the cells we see. Then we have to remember to do calculations on the internally-held data, not on the displayed contents of cells. To get into good habits, I have used the **DATA[]** array to hold the scores, and this is used in the calculation of averages. It comes down to this: *if you are sure all the data is in the cells, use them; if not, use the data where you know for sure it is.*

The following lines create a contextual menu for the tableview with four entries:

```
Public Sub tv1_Menu()  
  
    Dim mn, su As Menu 'main menu and submenu  
  
    mn = New Menu(Me) 'brackets contain the parent, the main window  
    su = New Menu(mn) As "MenuCopyTable" 'submenu of mn; alias is MenuCopyTable  
    su.Text = "Copy table..." 'first submenu's text  
    su = New Menu(mn) As "MenuCopyNames"  
    su.Text = "Copy names..." 'second submenu's text  
    su = New Menu(mn) As "MenuDeleteRow"  
    su.Text = "Delete Row" 'third submenu's text  
    su = New Menu(mn) As "MenuRefresh"  
    su.Text = "Refresh" 'fourth submenu's text  
    mn.Popup  
  
End  
  
Public Sub MenuDeleteRow_Click()  
  
    Names.Remove(tv1.Row)  
    Scores.Remove(tv1.Row)  
    tv1.Rows.Remove(tv1.Row)  
  
End  
  
Public Sub MenuCopyTable_Click() 'clicked the Copy Table menu item  
  
    Dim z As String  
  
    For i As Integer = 0 To Names.Max  
        If Scores[i] = -1 Then Continue  
        z = If(IsNull(z), "", z & gb.NewLine) & Names[i] & gb.Tab & Scores[i]  
    Next  
    Clipboard.Copy(z)  
    Message("Table copied")  
  
End
```

```

Public Sub MenuCopyNames_Click() 'clicked the Copy Names menu item

    Dim z As String
    For i As Integer = 0 To Names.Max
        If IsNull(Names[i]) Then Continue
        z = If(IsNull(z), "", z & gb.NewLine) & Names[i]
    Next
    Clipboard.Copy(z)
    Message("Names copied")

End

Public Sub MenuRefresh_Click()
    tv1.Clear 'clear the data
    tv1.Rows.Count = Names.Count 'reset number of rows to match Names[ ]
    For i As Integer = 0 To Names.Max
        tv1[i, 0].Text = Names[i]
        tv1[i, 1].Text = Scores[i]
        If i Mod 2 = 0 Then
            tv1[i, 0].Background = &hFFDDFF
            tv1[i, 1].Background = &hFFDDFF
        Endif
    Next
End

```

The *_Menu()* event belongs to *tv1*, the tableview. This event fires when the object is right-clicked (to show a menu). The *_Click()* event belongs to *TableviewMenu*. What is that? It is the alias by which the menu **su** is known. Aliases make one think of secretive men in dark trenchcoats, but it is just the name by which it is known.



commons.wikimedia.org/wiki/File:Font_Awesome_5_solid_user-secret.svg

This man is known by an alias—as menus are.

mn and **su** only exist for the duration of the popup menu because they are in the *_Menu()* event. As soon as you click any item on the popup menu, that sub finishes and *mn* and *su* disappear. Luckily, we have said that *su* is also known as *TableviewMenu*. The menu itself, when it was created with *New*, has that name. So any clicks the menu gets are handled by *TableviewMenu_Click()*.

Menus—whether main menus or submenus or menu items— have several events, methods and properties. See <http://Gambaswiki.org/wiki/comp/gb.qt4/menu> :

Methods

[Close](#) [Delete](#) [Hide](#)
[Popup](#) [Show](#)

Events

[Click](#) [Hide](#)
[Show](#)

1

You can tell a menu to Close, Hide, Popup, Show or Delete.

You can tell the program to do things when the menu is clicked, after it hides or just before it shows. Usually it is when the menu is clicked that the menu gets to work.

Properties

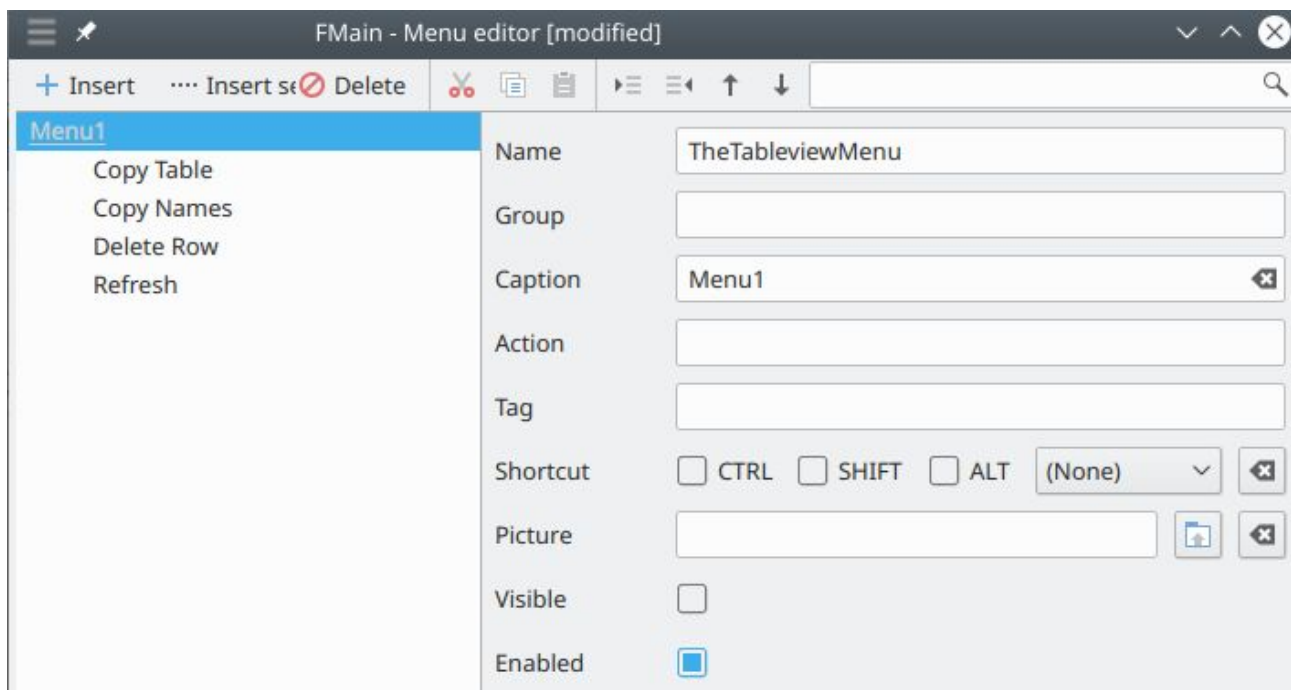
[Action](#) [Caption](#) [Checked](#) [Children](#) [Closed](#) [Enabled](#) [Name](#) [Picture](#) [Proxy](#) [Radio](#)
[Shortcut](#) [Tag](#) [Text](#) [Toggle](#) [Value](#) [Visible](#) [Window](#)

Some properties are boolean (e.g. *enabled*, *checked* and *visible*), others are strings (e.g. *text*, *name*), pictures (*picture*) or variants, which are any type (*tag*).

Making Contextual Menus at Design Time

There is another way to create a contextual menu for when you right-click the tableview: not in code as we have done, but at design time. You still have to write code to handle the menu item clicks, but you avoid all those New MenuItem statements like **mn = New MenuItem(Me)** and **su = New MenuItem(mn) As "MenuCopyTable"**. Simply use the Menu Editor to create a new menu with all its menu items. This menu will appear on the menubar, so to avoid that happening make it invisible (see the **Visible** checkbox below).

This is an easier way to make a contextual menu. It works if you know what the menu is going to be before you start the application. If you need to make a menu depending on what is typed in, you have to create the menu in code and use the **New** operator.



Sorting a GridView or TableView

There is no built-in sorting. I wish there were a method attached to each TableView like TableView_Sort(Column, Direction, NumericOrNot), but there isn't. There is a good way to sort in the online wiki,

<http://Gambaswiki.org/wiki/comp/gb.qt4/gridview/sorted>

but alas, it doesn't sort correctly for columns containing numbers. 10 comes before 2, for example, because "1" would come before "2" in a dictionary. The string "10" is less than the string "2" even though the number ten is greater than the number two.

(Thanks to fgores, Lee Davidson and Gianluigi of the forum

<http://Gambas.8142.n7.nabble.com/How-to-sort-a-TableView-td55814.html>)

So here is my method—somewhat agricultural, but it works. The idea is to go through the tableview row by row, gathering the cells in that row into a string with a separator between each, and putting the item that will determine the sorting right at the start of each. Sort the array, then unpack each row back into the tableview. In other words, pack each row into a string, sort the strings, and unpack each row. Use any rare and unusual character to separate the fields. Here a tilde (~) is used.

```
Public Sub tv1_ColumnClick(Column As Integer)
    tv1.Save 'Calls the Save event in case the user clicked col heading without
    pressing Enter
    SortTable(tv1, Column, tv1.Columns.Ascending, (Column = 1))
End

Public Sub SortTable(TheTable As TableView, Column As Integer, Ascending As
Boolean, Numeric As Boolean)

    Dim z As New String[]
    Dim y As New String[]
    Dim s As String
    Dim i, j As Integer

    For i = 0 To TheTable.Rows.Max
        If Numeric Then
            s = TheTable[i, Column].text 'next line pads it with leading zeros
            s = String$(5 - Len(s), "0") 'So 23 becomes 00023, for example. Works
up to 99999
        Else
            s = TheTable[i, Column].text
        Endif
        For j = 0 To TheTable.Columns.Max
            s &= "~" & TheTable[i, j].text
        Next
        z.add(s)
    Next

    If Ascending Then z.Sort(gb.Ascent) Else z.Sort(gb.Descent) 'sort

    For i = 0 To z.Max 'unpack the array
        y = Split(z[i], "~")
        For j = 1 To y.Max 'skip the first item, which is the sort key
            TheTable[i, j - 1].text = y[j] 'but fill the first column
```


Next
Next
End

Game of Concentration

You know the game: cards are arranged face down, you turn over a card, then you try to remember where you have seen its match. It's somewhere...thinks...yes, it was here! You turn it over to find ... no match! It was somewhere else. You turn the cards over and your friend takes a turn. If you do turn over matching cards, you take the cards. Whoever has the more cards at the end wins.



[en.wikipedia.org/wiki/Concentration_\(card_game\)](https://en.wikipedia.org/wiki/Concentration_(card_game))

In this version, there is only one player and you are racing against the clock to match all the cards.

The form is 508 wide and 408 high, but that is what I needed when I used Cooper Black for the font and +12 for the increased font size. Cooper Black has nice big black letters.

The *File* menu has three items, Give Up, New Game and Quit.

The form has its *arrangement* set to *Fill* so that the solitary gridview *gv1* fills the whole window. It shouldn't be resized, though, so set *resizeable* to *False*.

What we will do is have a 6x6 grid. All squares will be pale yellow. In memory is a kind of mirror image of the grid in an array called *z*. Arrays we have used up to now have been lists. A list is a series of items in a line. Lines are one-dimensional. The one dimension is their length. Grids are 2-dimensional and their two dimensions are length and width. For gridviews there are rows and columns. In our case *z* has rows and columns just as the gridview *gv1* has. **Public z As New String[6, 6]** will create *z* as this in-memory grid. The top left cell of the gridview is *gv1[0,0]*. The top left corner of *z* is *z[0,0]*. The bottom right cell of the gridview is *gv1[5,5]*, and the bottom right corner of *z* is *z[5,5]*. 36 cells in the gridview, like the 36 memories in *z*, are arranged in rows and columns.

The gridview shows the “cards” as we turn them over. The array *z* has the “underneath sides of the cards”. Pictures of your favourite relatives would be nice, but for now they will have big, Cooper Black letters, one on each.

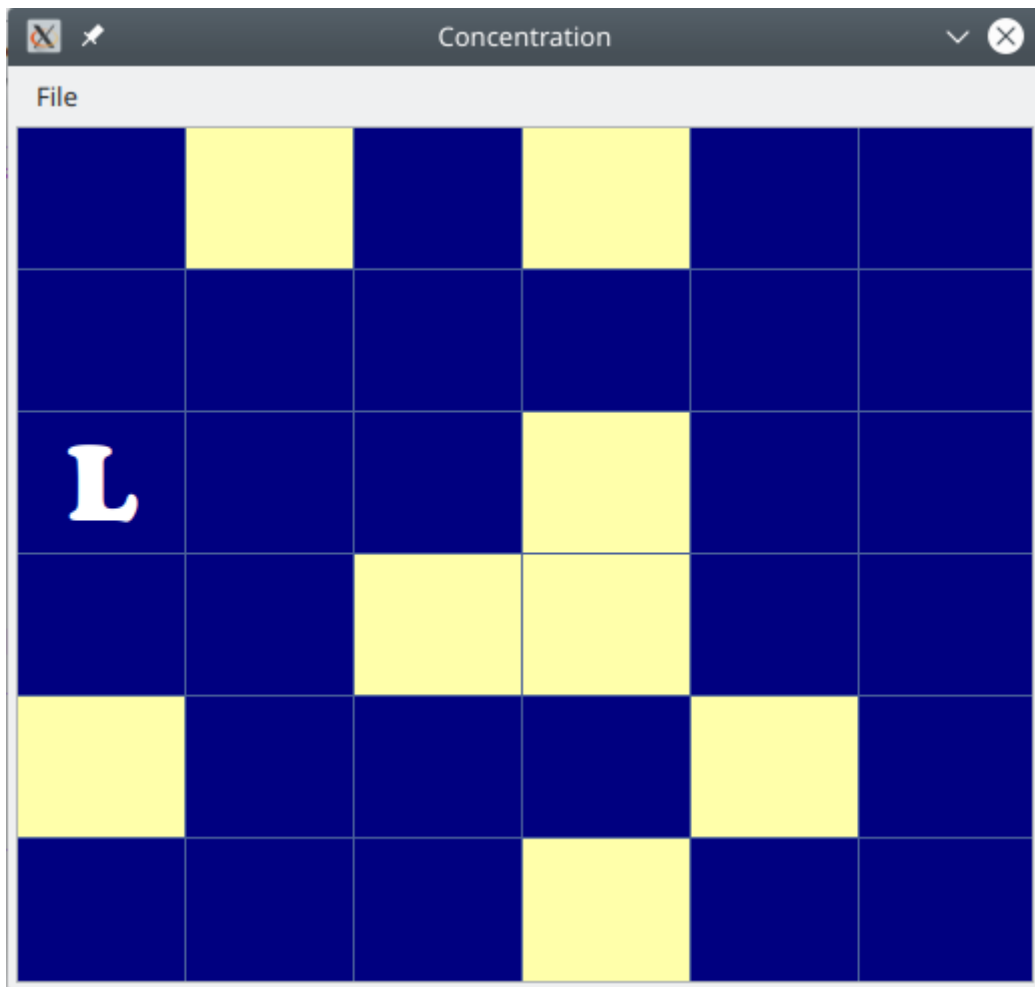
You click a grid cell. The card turns over: we show the letter that we have hidden in *z*. If we kept doing this every time we clicked a grid cell there would be no game. We would just gradually reveal all the letters. So when you show a letter we raise a flag that says “One letter is showing”. If that flag is up, the next time you click a cell we shall know that a second card has been turned over and it is time to check for a match. No match? Hide the letters and lower the flag—one letter is NOT showing. If we have a match perhaps it is the very last card and you have finished the game. Or perhaps it is not the very last card, and you can leave the cards turned over (their letters showing) and play on, remembering to lower that flag because the next click will again be clicking the first-card-of-the-pair. The flag is a boolean (true/false, hat on/hat off) variable called *OneShowing*.

How do we know the game is finished? Every time we have a match, add 2 to a running total of how many cards are out of the game. When that total reaches 36, all cards have been matched. The variable that keeps count is called *TurnedOver* and it is an integer. It is a public (or private, as in “private to the form”—it doesn’t matter) variable, declared right at the start with all the other variables that need to exist for the duration of the form and not disappear when a sub finishes.

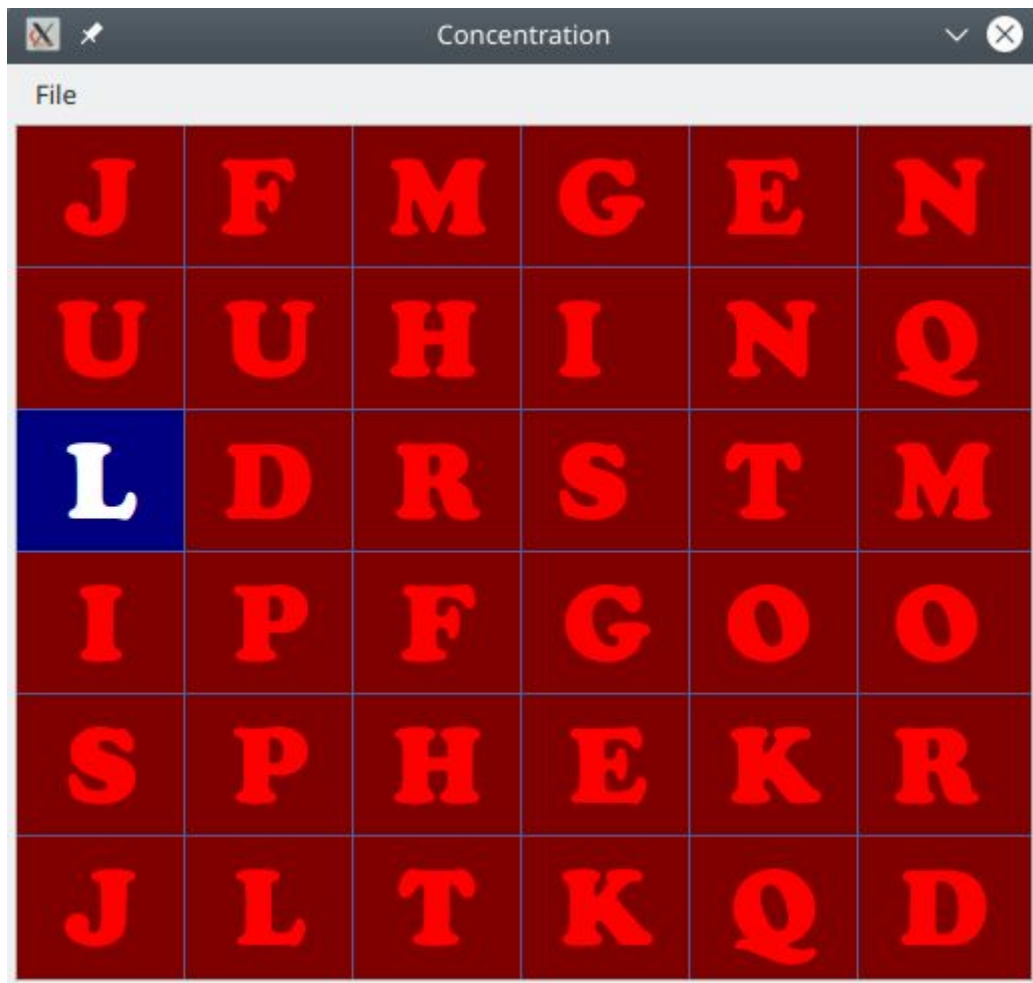
There is a timer included. The built-in function *Timer()* represents the number of seconds since the application started. As soon as you make your first click the time is stored in *StartTime*. How long you took to play is put into the variable *secs*.

Dim secs As Integer = Timer() - StartTime

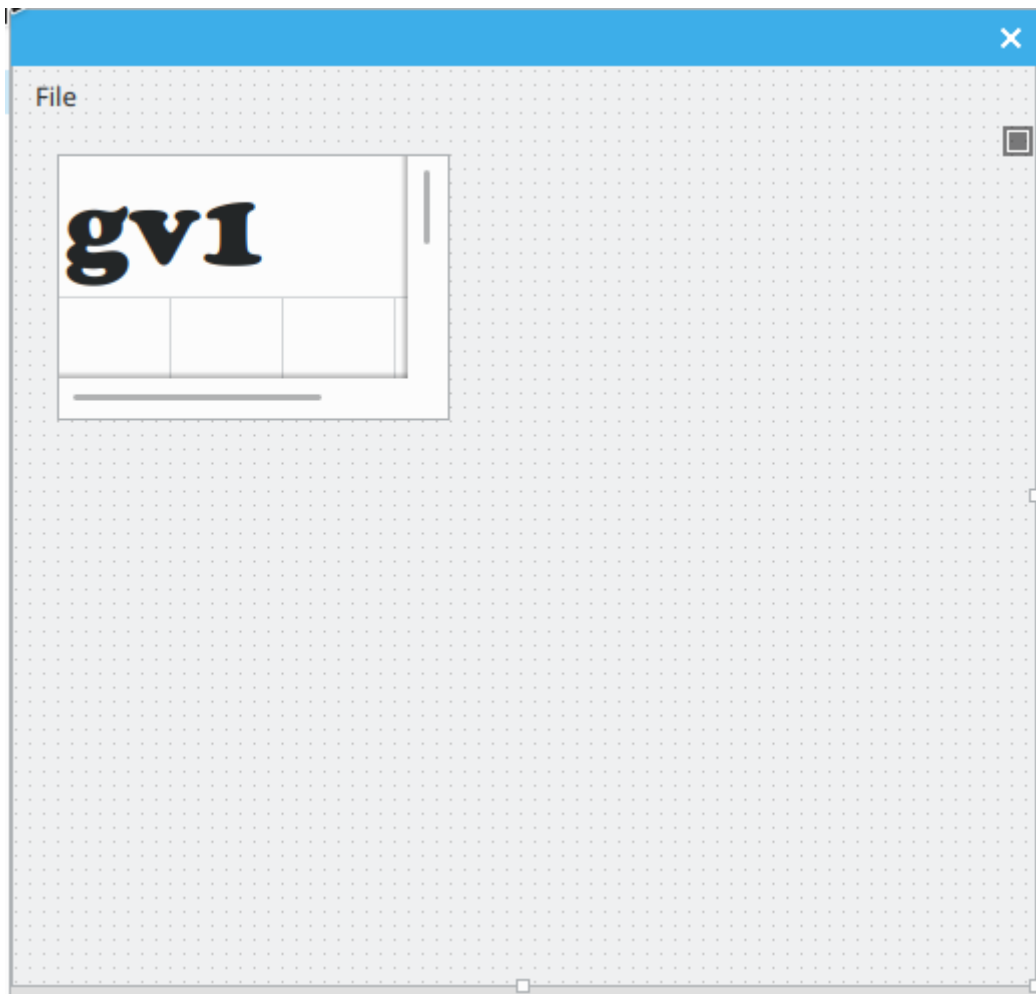
The game board is set up in the *Initialise* sub. It zeroes the things that have to be zeroed. It calls on *GetRandomLetters* to make a list called *s[]* of the letters that will be distributed to the cells of *z* with its 6 rows and 6 columns. *s* needs to have 18 random letters, each repeated so there are matching pairs. It has 36 items altogether. Here are the screenshots and the code.



Looking for matches



“Give Up” shows where they all were.



The 508 x 458 form. *Arrangement* property = *Full*. *gv1* has *expand* set to *True*.
The *File* menu has 3 items: *Give Up*, *New Game* and *Quit*.

```
' Gambas class file

Const nRows As Integer = 6
Const nCols As Integer = 6
Const PaleYellow As Integer = &hFFFAA
Public TurnedOver As Integer
Public z As New String[nRows, nCols]
Public s As New String[]
Public OneShowing As Boolean
Public FirstRow As Integer
Public FirstColumn As Integer
Public StartTime As Float

Public Sub Form_Open()

  Dim i, j As Integer
  gv1.Columns.Count = nCols
  gv1.Rows.Count = nRows
  gv1.Background = Color.DarkBlue
  For i = 0 To gv1.Columns.max
    gv1.Columns[i].Alignment = Align.Center
```

```

        gv1.Columns[i].Width = 84
    For j = 0 To gv1.Rows.max
        gv1[i, j].Padding = 16
    Next
Next
Initialise

End

Public Sub Initialise()

    Dim i, j, n, nCol, nRow As Integer
    Dim c As String

    nCol = gv1.Columns.max
    nRow = gv1.Rows.Max
    GetRandomLetters(18) 'each letter twice
    For i = 0 To nRow
        For j = 0 To nCol
            c = s[n]
            z[i, j] = c
            gv1[i, j].ForeColor = Color.Black
            gv1[i, j].Text = ""
            gv1[i, j].Background = Color.DarkBlue
            n = n + 1
        Next
    Next
    TurnedOver = 0

End

Public Sub GetRandomLetters(Count As Integer)

    Dim i, p, r1, r2 As Integer
    Randomize 'different random numbers every time
    s.clear
    p = Rand(Asc("A"), Asc("Z")) 'start with any letter
    Do Until s.count >= 2 * Count
        s.Add(Chr(p))
        s.Add(Chr(p)) 'other one in the pair
        p += 1
        If p > Asc("Z") Then p = Asc("A") 'back to the start
    Loop
    For i = 0 To s.Count 'c.shuffle() 'When I update to 3.13 I can use this!
        r1 = Rand(0, s.max)
        r2 = Rand(0, s.max)
        Swap s[r1], s[r2]
    Next

End

Public Sub MenuNew_Click()
    Initialise
End

Public Sub MenuQuit_Click()
    Quit
End

Public Sub gv1_Click()

```

```

If TurnedOver = 0 Then StartTime = Timer 'begin timing from the first click
If OneShowing Then
    gv1[gv1.row, gv1.Column].Background = Color.DarkBlue
    gv1[gv1.row, gv1.Column].Foreground = Color.White
    gv1[gv1.row, gv1.Column].Text = z[gv1.row, gv1.Column]
    gv1.Refresh
    Wait 'finish pending operations and do the refresh
    Evaluate(gv1.row, gv1.Column)
Else
    FirstRow = gv1.row
    FirstColumn = gv1.Column
    gv1[FirstRow, FirstColumn].Background = Color.DarkBlue
    gv1[FirstRow, FirstColumn].Foreground = Color.White
    gv1[FirstRow, FirstColumn].Text = z[FirstRow, FirstColumn]
    OneShowing = True
Endif

```

End

Public Sub Evaluate(row **As Integer**, column **As Integer**)

```

If z[FirstRow, FirstColumn] = gv1[row, column].Text Then 'a match
    TurnedOver += 2
    If TurnedOver = nRows * nCols Then
        Dim t As String
        t = TheTime()
        Message("Well done!<br>You took " & t)
        Initialise
    Else
        Wait 0.5
        gv1[FirstRow, FirstColumn].Text = ""
        gv1[row, column].Text = ""
        gv1[FirstRow, FirstColumn].Background = PaleYellow
        gv1[row, column].Background = PaleYellow
    Endif

Else 'no match
    Wait 1 'second
    gv1[FirstRow, FirstColumn].Text = ""
    gv1[row, column].Text = ""
    gv1[FirstRow, FirstColumn].Background = Color.DarkBlue
    gv1[row, column].Background = Color.DarkBlue

Endif
OneShowing = False

```

End

Public Sub TheTime() **As String**

```

    Dim secs As Integer = Timer() - StartTime
    Dim h As Integer = secs / 60 / 60

    Secs -= h * 60 * 60
    Dim m As Integer = secs / 60
    Secs -= m * 60
    Return If(h > 0, Str(h) & "h ", "") & If(m > 0, Str(m) & "m ", "") &
Str(secs) & "s"

```

```

End

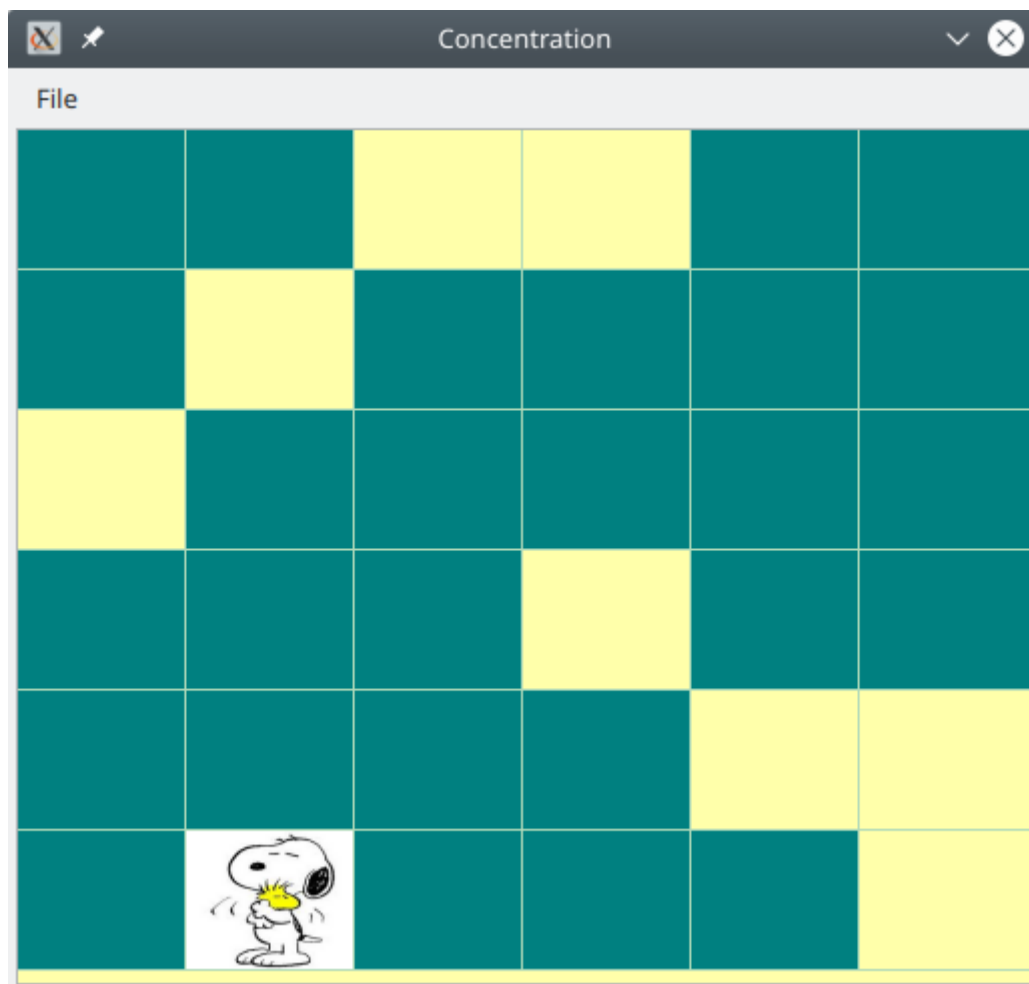
Public Sub MenuGiveUp_Click()

    For i As Integer = 0 To nRows - 1
        For j As Integer = 0 To nCols - 1
            If gv1[i, j].Text = "" Then
                gv1[i, j].ForeColor = Color.Red
                gv1[i, j].Text = z[i, j]
                gv1[i, j].Background = Color.DarkRed
            End If
        Next
    Next

End

```

If you prefer to work with clicking pictures, you will need a folder called Pix located in your Pictures folder. You need to put 18 pictures in it (jpg or png).



Now, where is the other Snoopy?



“Give up” shows where they all were.

In the code that follows, the picture files are read into an array of images. Images and Pictures in Gambas differ in what part of memory they are stored in. Images, unlike pictures, can be stretched to fit in an area with any given width and height. So the images in the array of eighteen are, when one is needed, put into a single Image called `Img`, stretched to fit one of the grid cells. The resulting image, now the right size, is converted to a picture using the `Picture` method that images have.

To show an image the cell's `Picture` property is set to the converted image. To hide it the picture property is set to `Null`. This happens when you click on a cell.

There is a corresponding two-dimensional (rows/columns) array of picture names. To see if there is a match, the names of the pictures in the two clicked-on cells are compared.

It is time to congratulate the winner when 18 cells have been correctly matched (two at a time).

' Gambas class file

```
Const nRows As Integer = 6
Const nCols As Integer = 6
Const PaleYellow As Integer = &hFFFFAA
Public TurnedOver As Integer
Public z As New String[nRows, nCols] 'names of the pictures
Public Images As New Image[nRows, nCols] 'the images themselves
```

```

Public Img As Image
Public s As New String[]
Public OneShowing As Boolean
Public FirstRow As Integer
Public FirstColumn As Integer
Public StartTime As Float

```

```

Public Sub Form_Open()

```

```

    Dim i, j As Integer

```

```

    gv1.Columns.count = nCols
    gv1.Rows.Count = nRows
    gv1.Background = PaleYellow
    For i = 0 To gv1.Columns.max
        gv1.Columns[i].Alignment = Align.Center
        gv1.Columns[i].Width = 84
    Next
    Initialise

```

```

End

```

```

Public Sub Initialise()

```

```

    Dim i, j, n, nCol, nRow As Integer
    Dim c As String

```

```

    nCol = gv1.Columns.max
    nRow = gv1.Rows.Max
    GetRandomLetters 'each picture twice
    For i = 0 To nRow
        gv1.Rows[i].Height = 70
        For j = 0 To nCol
            c = s[n]
            z[i, j] = c
            gv1[i, j].Picture = Null
            gv1[i, j].Background = Color.DarkCyan
            n = n + 1
        Next
    Next
    TurnedOver = 0

```

```

End

```

```

Public Sub GetRandomLetters()

```

```

    Dim i, j, n As Integer
    Dim path As String = User.Home & "Pictures/Pix/" 'must be 18 pictures in
here
    Dim cellW As Float = gv1[0, 0].Width
    Dim cellH As Float = gv1[0, 0].Height
    Dim scale As Float = Min(CellW, CellH)

    If Not Exist(Path) Then
        Message("Please create a folder called Pix in your Pictures folder.<br>Put
18 pictures in it.")
        Quit
    Endif
    s = Dir(path, "*.png")
    s.Insert(Dir(path, "*.jpg"))

```

```

If s.Count < 18 Then
    Message("Please put 18 pictures in the Pix folder inside your Pictures
folder.<br>There were only " & s.Count)
    Quit
Endif
s.Insert(s) 'second copy
s.Shuffle
For i = 0 To gv1.Rows.Max
    For j = 0 To gv1.Columns.Max
        Images[i, j] = Image.Load(path & s[n])
        n += 1
    Next
Next

End

Public Sub MenuNew_Click()

    Initialise

End

Public Sub MenuQuit_Click()

    Quit

End

Public Sub gv1_Click()

    If TurnedOver = 0 Then StartTime = Timer 'begin timing from the first click
    If OneShowing Then
        gv1[gv1.row, gv1.Column].Background = Color.White
        Img = Images[gv1.row, gv1.Column].stretch(70, 70)
        gv1[gv1.row, gv1.Column].Picture = Img.Picture
        gv1.Refresh
        Wait 'finish pending operations and do the refresh
        Evaluate(gv1.row, gv1.Column)
    Else
        FirstRow = gv1.row
        FirstColumn = gv1.Column
        gv1[FirstRow, FirstColumn].Background = Color.White
        Img = Images[FirstRow, FirstColumn].stretch(70, 70)
        gv1[FirstRow, FirstColumn].Picture = Img.Picture
        OneShowing = True
    Endif

End

Public Sub Evaluate(row As Integer, column As Integer)

    If z[FirstRow, FirstColumn] = z[row, column] Then 'a match
        TurnedOver += 2
        If TurnedOver = nRows * nCols Then
            Dim t As String
            t = TheTime()
            Message("Well done!<br>You took " & t)
            Initialise
        Else
            Wait 0.5 'half second

```

```

        gv1[FirstRow, FirstColumn].Picture = Null
        gv1[row, column].Picture = Null
        gv1[FirstRow, FirstColumn].Background = PaleYellow
        gv1[row, column].Background = PaleYellow
    Endif

Else 'no match
    Wait 1 'second
    gv1[FirstRow, FirstColumn].Picture = Null
    gv1[row, column].Picture = Null
    gv1[FirstRow, FirstColumn].Background = Color.DarkCyan
    gv1[row, column].Background = Color.DarkCyan

Endif
OneShowing = False

End

Public Sub TheTime() As String

    Dim secs As Integer = Timer() - StartTime
    Dim h As Integer = secs / 60 / 60

    Secs -= h * 60 * 60
    Dim m As Integer = secs / 60
    Secs -= m * 60
    Return If(h > 0, Str(h) & "h ", "") & If(m > 0, Str(m) & "m ", "") &
    Str(secs) & "s"

End

Public Sub MenuGiveUp_Click()

    For i As Integer = 0 To nRows - 1
        For j As Integer = 0 To nCols - 1
            If gv1[i, j].Picture = Null Then
                Img = Images[i, j].Stretch(70, 70)
                gv1[i, j].Picture = Img.Picture
                gv1[i, j].Background = Color.White
            End If
        Next
    Next

End

```

ASCII Codes

Characters (letters, digits, punctuation symbols) are stored in a computer's memory by numbers. The most widely used system is ASCII, *American Standard Code for Information Interchange*. It was developed in the United States, and Wikipedia tells me the governing body prefers to call it US-ASCII because it uses the American dollar sign (\$) and the Latin alphabet. Whenever you hit a key on the keyboard one of those code numbers goes into the computer. Even the non-printing characters have ASCII codes. The *spacebar* is 32. Hit the *Delete* key and 127 would go in. The *Backspace* key sent the number 8. To confuse you, the ASCII code for the digit '1' is 49. What the

computer does with these code numbers is up to the application. And *you* are writing the applications. In the above program, type what you like and nothing happens at all (except for CTRL-G which I have for “Give Up”, CTRL-N which is “New Game” and CTRL-Q, which is the shortcut for Quit).



On the old manual typewriters at the end of a line you had to flick a lever and the roller with the paper going around it would zip back to the start of the line (*Return*) and pull the paper up a line (*Linefeed*) ready to start typing the next line. *Return* is 13. ASCII 13 is also Control-M (written **^M**) and in programming languages is sometimes written `\r`. Linefeed is 10 and is also Control-J (**^J**). There is a *Formfeed* control, Control-L, that used to go to a new page (ASCII 12). You wouldn't remember manual typewriters unless you spend time in museums, but for me it is like yesterday (sigh). Nowadays ASCII is largely replaced by Unicode. ASCII was limited to 128 characters. Unicode can display 137,993 characters, says Wikipedia—enough for all sorts of non-English characters and all the emojis you could ever want.

Even the original ASCII gave problems for people who spoke languages other than English. Wikipedia has the amusing example of ‘a Swedish programmer mailing another programmer asking if they should go for lunch, could get "N{ jag har sm|rg}sar" as the answer, which should be "Nä jag har smörgåsar" meaning "No I've got sandwiches" ’ and he or she would just have to put up with it. <https://en.wikipedia.org/wiki/ASCII> .

Radio Buttons and Groups



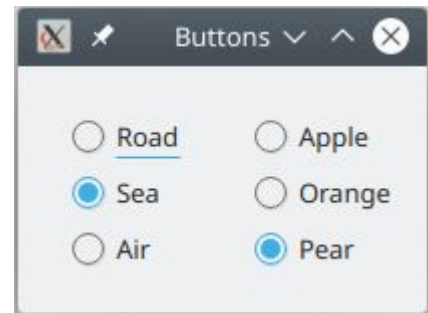
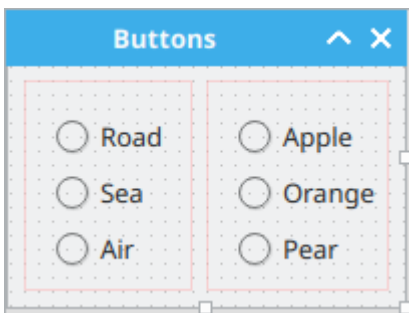
Radio Buttons—one goes down, the others pop up

Radio buttons are like the buttons on the old cassette players. When you press down on one the others pop up. They are used for selecting one option among many.

If you have some radio buttons in a form, only one can be highlighted. Click one and the others clear. Even in code if you set one button's highlight the others will unhighlight by themselves. The *value* property (boolean) indicates if the button is highlighted. When you click a radio button *rb1* **rb1.value = true** happens automatically. When you click another button **rb1.value = false** happens automatically.

You might need two sets of radio buttons. To keep them separate, create them in a panel or some other container. Put the panel there first, and then make radio buttons in it, or select all the buttons you want to work together, right-click, and choose *Embed in a container*.

Another trick is to make them all **share their events**. Click any of the buttons and one and only one *_Click* event will fire. This avoids writing separate *_Click* handlers for each button. One handler does them all. But how do you know which button was clicked? Your one handler will probably want to do something based on which button was selected. This is where **Last** comes in. *Last* is the very last object that something happened to or that did something.



There are two sets of buttons, with each set in their own panel. *rbRoad*, *rbSea* and *rbAir* are in *Panel1*. *rbApple*, *rbOrange* and *rbPear* are in *Panel2*. The panels are the **parents** of their buttons. The buttons are their **children**.

The **Group** property for the road, sea and air buttons is set to *rbTransport*. It is as if they are acting like they are one single radio button, *rbTransport*.

The *Group* property for the apple, orange and pear buttons is set to *rbFruit*. It is as if they are one radio button, *rbFruit*.

Double-click one of the transport buttons (any one). You will find yourself writing a handler for the *rbTransport* group of buttons. Likewise, double-click one of the fruit buttons and you will find yourself writing a handler that *rbApple*, *rbOrange* and *rbPear* all respond to.

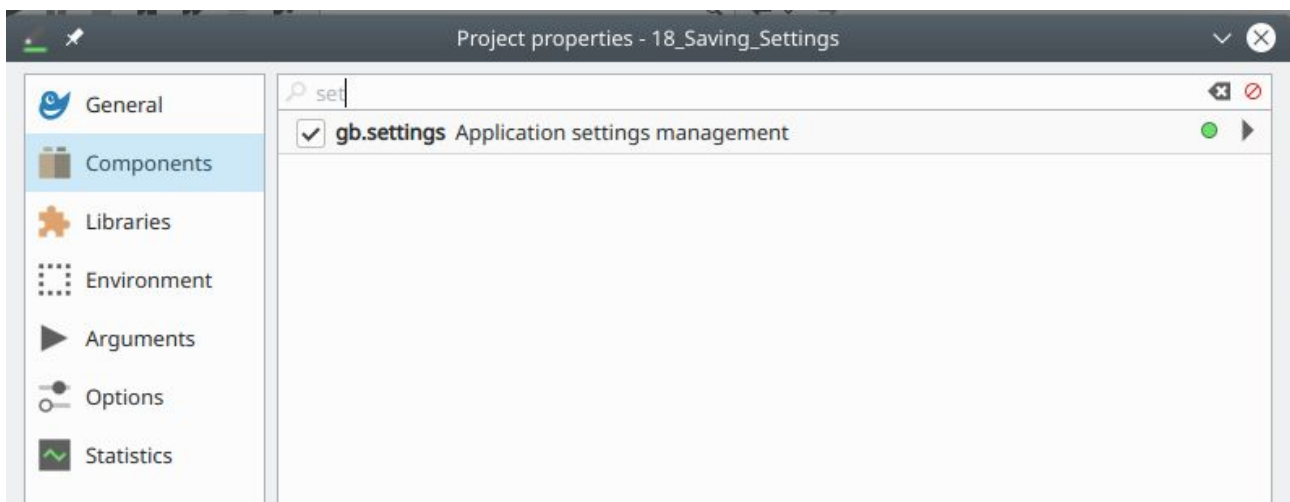
```
Public Sub rbTransport_Click()  
    Message("You choose to travel by " & Last.text)  
End  
  
Public Sub rbFruit_Click()  
    Message("I like " & LCase(Last.text) & "s too!")  
End
```

The LCase function makes the text inside the brackets lower case. Run the program and click on buttons.

Add Settings Saving to the Radio Buttons

Gambas provides a neat way to save settings. Settings can be the path to the last data file, so it does not have to be relocated the next time the program starts. They can be anything the user typed or chose that you want to remember for next time. Here we shall save the selected radio buttons.

First, make sure the Settings component is enabled as part of your project. After starting a new QT graphical project, select Project Menu > Properties..., look through for the gb.settings component and tick it:



Use the same form as above with the fruit and transport buttons, but change the code to this:

```
Public Sub rbTransport_Click()  
    Settings["Radiobuttons/Transport"] = Last.Text  
End  
  
Public Sub rbFruit_Click()  
    Settings["Radiobuttons/Fruit"] = Last.Text  
End  
  
Public Sub Form_Open()  
  
    Select Case Settings["Radiobuttons/Transport"]  
        Case "Road"  
            rbRoad.value = True  
        Case "Sea"  
            rbSea.Value = True  
        Case "Air"  
            rbAir.value = True  
    End Select  
  
    Select Case Settings["Radiobuttons/Fruit"]  
        Case "Apple"  
            rbApple.value = True  
        Case "Orange"
```



```

        rbOrange.Value = True
    Case "Pear"
        rbPear.value = True
    End Select
End

```

Run the program. Select a transport and fruit. Close the program. Run it again: your choices have been restored.

You could have your settings saved when the form closes. Gambas wiki has this example, showing how you can restore the window to whatever place it was last dragged to and whatever size it was resized to when last the program ran:

```

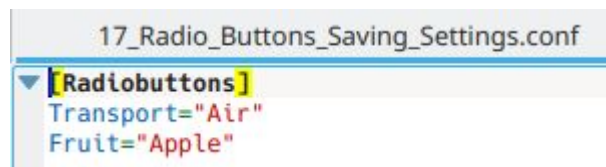
Public Sub Form_Open() 'Restore settings
    Me.Top = Settings["Window/Top", Me.Top]
    Me.Left = Settings["Window/Left", Me.Left]
    Me.Height = Settings["Window/Height", Me.Height]
    Me.Width = Settings["Window/Width", Me.Width]
End

Public Sub Form_Close() 'Save settings
    Settings["Window/Top"] = Me.Top
    Settings["Window/Left"] = Me.Left
    Settings["Window/Height"] = Me.Height
    Settings["Window/Width"] = Me.Width
End

```

Me means the current form.

Where are these settings actually stored? In your home folder is a hidden folder for settings called *.config* . In Linux any file or folder whose name starts with a dot is hidden. Look in *.config* for the Gambas3 folder. In it you will find a text file with the same name as your program. Open it and you will see the settings file.

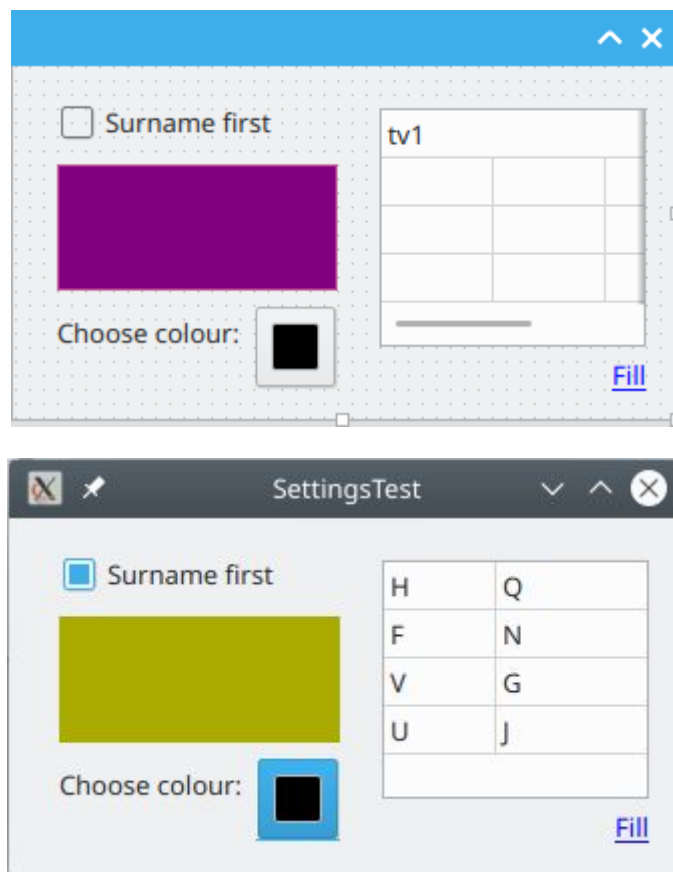


The settings text file for the application called 17_Radio_Buttons_Saving_Settings.

Settings are neatly arranged under headings. Now you can see the significance of the string that has the slash in it: the first item is the heading. **Settings["Radiobuttons/Fruit"]** is the *Fruit* setting under the *Radiobuttons* heading.

You need to be careful: the very first time you run your program there may not be a settings file. If your form opens and looks for a particular setting when no settings file exists there will be problems. Test for empty (null) strings.

Saving a colour, a checkbox and the contents of a TableView



On the form is a checkbox *cbSurnameFirst*, a panel *Panel1*, a label with the text “Choose colour:”, a colorbutton *ColorButton1*, a label *Label1* whose text is “Fill”, colour blue and underlined, and a tableview *tv1*.

Run the program. Fill the tableview with random letters. Choose a colour. Highlight the completely useless button “Surname first”. Close the program. Run the program again. Settings are restored.

```
Public Sub ColorButton1_Change()  
    Panel1.Background = ColorButton1.Color  
    Settings["Colours/PanelColour"] = Panel1.Background  
End  
  
Public Sub Label1_MouseDown()  
  
    tv1.Columns.count = 2  
    Settings["TableView/Columns"] = tv1.Columns.count  
    tv1.Rows.count = 4  
    Settings["TableView/Rows"] = tv1.Rows.count  
    For i As Integer = 0 To tv1.Rows.Max  
        For j As Integer = 0 To tv1.Columns.Max  
            tv1[i, j].text = Chr(Rand(Asc("A"), Asc("Z")))   
            Settings["TableView/" & i & "," & j] = tv1[i, j].text  
        Next  
    Next  
End
```

```

Public Sub cbSurnameFirst_Click()
    Settings["Names/SurnameFirst"] = cbSurnameFirst.Value
End

Public Sub Form_Open() 'restore settings

    Dim Surname As String = Settings["Names/SurnameFirst"]
    cbSurnameFirst.Value = If(IsNull(Surname), False, Surname)
    Dim nCols As String = Settings["TableView/Columns"]
    tv1.Columns.count = If(IsNull(nCols), 2, nCols)
    Dim nRows As String = Settings["TableView/Rows"]
    tv1.Rows.count = If(IsNull(nRows), 4, nRows)
    For i As Integer = 0 To tv1.Rows.Max
        For j As Integer = 0 To tv1.Columns.Max
            tv1[i, j].text = Settings["TableView/" & i & "," & j]
        Next
    Next
    Dim colour As String = Settings["Colours/PanelColour"]
    Panel1.Background = If(IsNull(colour), &hFFFFFF, colour)
End

```

IF Function

There is a special form of the IF...THEN...ELSE statement that saves writing several lines of code. It is in the form of a function. These two are equivalent:

```

if IsNull(colour) Then
    Panel1.Background = &hFFFFFF 'white
Else
    Panel1.Background = colour
EndIf

```

and

```

Panel1.Background = If(IsNull(colour), &hFFFFFF, colour)

```

In the one-line statement, the `If(IsNull(colour), &hFFFFFF, colour)` is one single thing. It is a number representing a color. Which colour? In the brackets are three items: a test that is either *true* or *false*, the answer if the test comes up *true* and the answer if the test comes up *false*. The pattern is *if(TrueOrFalseThing, ValueIfTrue, ValueIfFalse)*. `&hFFFFFF` is the hexadecimal number for *White* (all red, green and blue LED lights fully on).

This is a sample settings file:

```
[Colours]
PanelColour=11184640

[Names]
SurnameFirst=0

[TableView]
Columns=2
Rows=4
0,0="H"
0,1="Q"
1,0="F"
1,1="N"
2,0="V"
2,1="G"
3,0="U"
3,1="J"
```

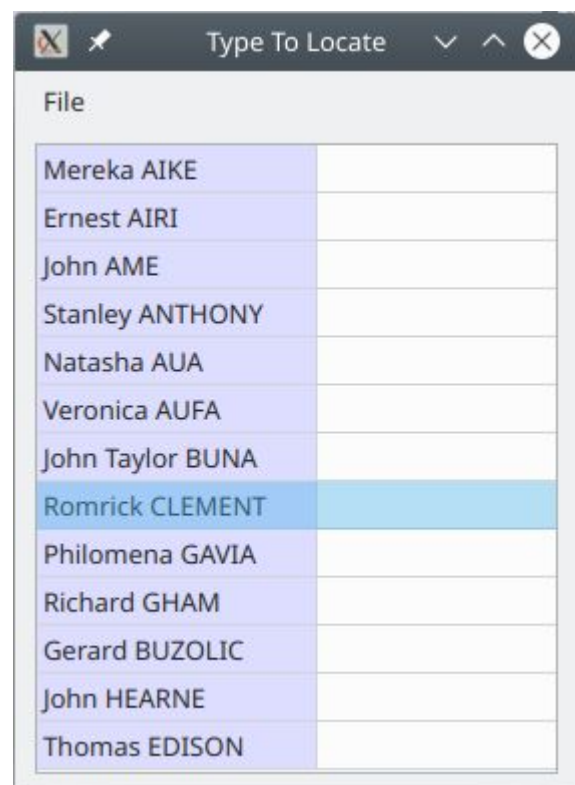
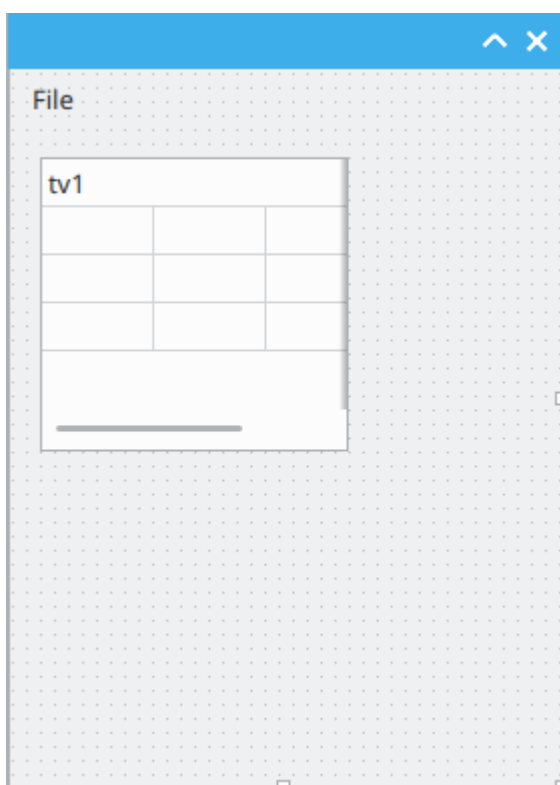
```
[Colours]
PanelColour=11184640

[Names]
SurnameFirst=-1

[TableView]
Columns=2
Rows=4
0,0="H"
0,1="Q"
1,0="F"
1,1="N"
2,0="V"
2,1="G"
3,0="U"
3,1="J"
```

On the left, the checkbox is unchecked. On the right, the checkbox is checked.

A SearchBox to Locate Names by Typing



On the right, Romrick Clement's name was found as soon as CL was typed.

If you are a teacher entering marks you might have a pile of a hundred assignments, in random order. It would be good to be able to locate a name by typing a few letters. When the name is located, press *Enter* so you can type the mark. Press *Enter* again and you are ready to locate the next name.

This program has a tableview that allows you to locate names by typing. The letters do not have to be from the start of the name; they can be anywhere in it. To find the next occurrence of the string of letters you have typed (e.g. the next 'John'), press *tab* or *right-arrow*. To find the previous occurrence of the string, *shift-tab* or *left-arrow*. To clear the search string and start again, press *esc*, *backspace* or *delete*.

To stop making *Enter* leave the cell and allow you to search for another name, and instead move to the cell below (as you would if typing marks down the list in sequence), select *Down* from the *File* menu (shortcut, Ctrl-D). Enter then moves you *down* the list.

```
' Gambas class file

Static ss As String

Public Sub tv1_KeyPress()

    Select Case Key.Code
        Case Key.Esc, Key.BackSpace, Key.Del
            ss = ""
            tv1.UnselectAll
        Case Key.Enter, Key.Return
            EnterOnLine 'action on pressing Enter
            ss = ""
            tv1.UnselectAll
        Case Key.Tab
            SearchDown
        Case Key.BackTab
            SearchUp
        Case Else
            ss &= Key.Text
            SearchDown
    End Select

End

Public Sub SearchUp()

    Dim i, Start As Integer
    If tv1.Rows.Selection.Count = 0 Then Start = -1 Else Start =
tv1.Rows.Selection[0] 'the selected line
    For i = Start - 1 DownTo 0
        If InStr(LCase(tv1[i, 0].text), LCase(ss)) Then
            tv1.Rows.Select(i)
            Return
        Endif
    Next
    For i = tv1.Rows.max DownTo Start
        If InStr(LCase(tv1[i, 0].text), LCase(ss)) Then
            tv1.Rows.Select(i)
            Return
        Endif
    Next

End

Public Sub SearchDown()
```

```

    Dim i, Start As Integer
    If tv1.Rows.Selection.Count = 0 Then Start = -1 Else Start =
tv1.Rows.Selection[0]
    For i = Start + 1 To tv1.Rows.Max 'if no selected line, start at top, else
start at next line
        If InStr(LCase(tv1[i, 0].text), LCase(ss)) > 0 Then
            tv1.Rows.Select(i)
            Return
        Endif
    Next
    For i = 0 To Start 'if no more occurrences, you will end up at the line you
are on
        If InStr(LCase(tv1[i, 0].text), LCase(ss)) > 0 Then
            tv1.Rows.Select(i)
            Return
        Endif
    Next

End

Public Sub EnterOnLine()
    tv1.Column = 1
    tv1.Edit
End

Public Sub Form_Open()

    Dim Names As New String[]
    Dim i As Integer
    Names = ["Mereka AIKE", "Ernest AIRI", "John AME", "Stanley ANTHONY",
"Natasha AUA", "Veronica AUFA", "John Taylor BUNA", "Romrick CLEMENT",
"Philomena GAVIA", "Richard GHAM", "Gerard BUZOLIC", "John HEARNE", "Thomas
EDISON"]
    tv1.Rows.Count = Names.count
    tv1.Columns.Count = 2
    tv1.Columns[0].Background = &hDDDDFF
    For i = 0 To Names.Max
        tv1[i, 0].text = Names[i]
    Next
    tv1.Columns[0].Width = 140 '-1 for max needed width
    tv1.Mode = Select.Single
    tv1.NoKeyboard = True

End

Public Sub tv1_Save(Row As Integer, Column As Integer, Value As String)
    tv1[Row, Column].text = Value
End

Public Sub MenuQuit_Click()
    Quit
End

Public Sub tv1_Click()

    If tv1.Column > 0 Then 'first column is not editable
        If MenuDown.Checked Then
            tv1.Edit
        Else
            ss = ""

```

```

        tv1.SetFocus
        tv1.Edit
    Return
Endif
Endif

End

Public Sub tv1_DblClick()
    tv1.Edit 'to edit the names in the first column, double-click one of them
End

Public Sub MenuDown_Click()
    MenuDown.Checked = Not MenuDown.Checked
    tv1.NoKeyboard = Not MenuDown.Checked
End

```

Modules and Classes

Programs tend to become large as more features are added to them. More menu items mean more menu handlers. More buttons, more lists, more tables—all mean more subs. There needs to be a way to organise them, and there is. The files in a computer are organised into folders. The subs in a program are organised into modules and classes.

Modules are like containers. **Classes** are like animals of various species.

You can put what you like into containers. You can collect all the subs that are related in some way and put them into a module. For example, you might make a module called Time and put in it all the bits of program related to times and dates. In it you might put that great function you wrote to work out a person's age given their date of birth. You called it

Public Sub AgeThisYear(DOB as date) as string

And with it you could put that function that takes how many seconds you took to complete a puzzle and convert it into minutes and seconds format:

Public Sub TidyTime(Secs as integer) as string

They could go into the Time module to save cluttering up the form's code. There will be enough event handlers to fill it up without these functions as well. You cannot move those event handlers into a module. They have to be in the form's code, waiting for something on the form to do something to make them fire.

How do you call on subs that have been put into a module? You have to refer to the module they have been put in, and the name of the sub. It is like having a crowd of people all gathered together in a park. You can call out "John!" or "Mary!" and John or Mary will step forward. Once you start putting people into houses, though, it has to be "HollyCottage.John" or "FernyVale.Mary". There might be several Johns or Marys, for one thing. So we would refer to

Time.AgeThisYear("1/6/1955")

and

Time.TidyTime(258)

Anything in a module is available throughout your program. Modules are like boxes or folders or filing cabinets: just places you can park your subs. To call them, put the module name, a dot, then the sub.

Classes, though, are like kinds of animals. In the animal world, species are grouped into *genuses*, *genuses* into *families*, and so on up to *Kingdom* (and now one higher level, *Domain*), which used to be *Animal* or *Plant* but now includes others (*Monera* which is mainly bacteria, *Protists* which include algae and protozoans, and *Fungi* which is mushrooms, moulds and yeasts). Every animal and plant has a two-part name made up of *Genus* and *Species*. “Genus” means “General” and “Species” means “Specific”. My name is, let us say, Luke Bilgewater. There are many different individuals in the Bilgewater family, but only one Luke Bilgewater. Luke is the specific name and Bilgewater is the general or generic name. The classification system goes this way:

DOMAIN → KINGDOM → PHYLUM → CLASS → ORDER → FAMILY → GENUS → SPECIES.

You will notice that “*Class*” comes in there. In programming, classes are things that you can have examples of. A tableview is a class. You can have many tableviews in a program. A button is a class. You can have many buttons. A menu is a class. You can have many menus. A form is a class. You can have many forms.

You can also derive classes from other classes.

Let’s take the horse. Wikipedia says, “At present, the domesticated and wild horses are considered a single species, with the valid scientific name for the horse species being *Equus ferus*”. Imagine a horse with wings. It would have everything that regular horses have (properties, like tails and hooves) and can do everything that regular horses do (methods, like gallop and neigh) and respond to things regular horses respond to (events, like approaching a water trough or getting saddled up). However, in addition to these, it will have wings. This new class, *Equus Pegasus*, will inherit everything that an *Equus* has, but will also have **Public Sub Wings() ... End** specific to this special type of *Equus*.

Classes (unlike modules) can have as many real live examples as you want. Each example will have its own wings. Each will have its own name. “Make a new example of one of these kinds of animals” is, in programming language, “*instantiate*” or “make a new instance”.

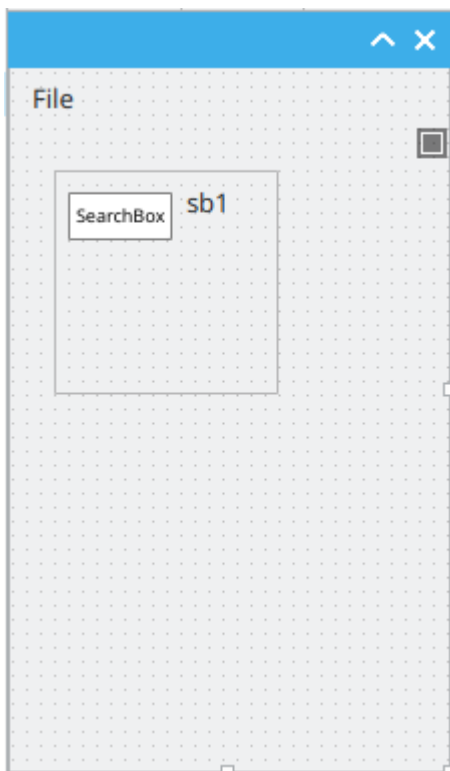
A class is an abstract thing (like a type of animal). You need instances of the class to have anything you can work with.

In saying that a class definition is like a blueprint, that is the usual case. If you want the class to exist as a one-of-a-kind animal, you can do that by making it **Static**. Gambas has static classes. They are classes that are always there. Modules are static classes: always there, and you only have one of them.

An example of a static class is *Key*. In the online help it says, about *Key*, “This class is used for getting information about a key event, and contains the constants that represent the keys. ... This class is static.” It is always there; it is a once-off thing; refer to it as if it were the name of a beast, not just the *kind* of beast that it is. So the key that was just typed is *Key.Text* and the number code for that key is *Key.Code*. Just as sure as horses have legs, *Key* has several constants you can refer to, like *Key.Enter* and *Key.Return* and *Key.Del* and *Key.Esc* that are the code numbers for those keys. And just as sure as horses can have a saddle on or a saddle off, there is the property *Key.Shift* and *Key.Control* that can be up or down, that is to say, *true* or *false*.

Let’s take a tableview and give it wings. Our new class will be everything that a tableview is, but with the additional ability to locate lines by typing in a few of the letters. It is our *SearchBox* again, only this time we shall make it a class. Then we can make as many new *SearchBoxes* as we like. We only need design the prototype for the new car; after that we can have as many rolling off the assembly line as we like.

Making a SearchBox Class



File	
Mereka AIKE	
Ernest AIRI	
John AME	78
Stanley ANTHONY	
Natasha AUA	
Veronica AUFA	
John Taylor BUNA	
Romrick CLEMENT	
Philomena GAVIA	
Richard GHAM	86
Gerard BUZOLIC	
John HEARNE	
Thomas EDISON	

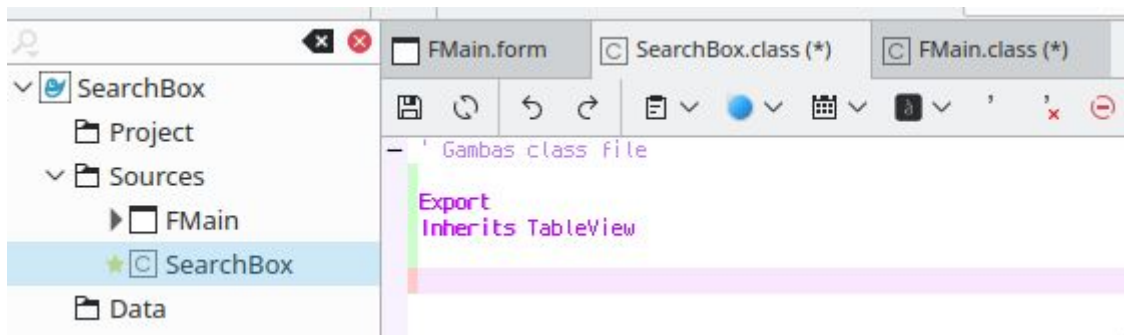
The File Menu has two entries, *Go Down* and *Quit*. Don’t give *Go Down* a keyboard shortcut. (If typed while in a cell you can find yourself in an endless loop—nothing happens, nothing responds.) The menu items are called *MenuGoingDown* and *MenuQuit* respectively.

The menu item *Go Down* uses its checkbox. If ticked, searching for names by typing is switched off. Enter moves the cursor down to the next cell below as usual. If unticked (the program starts this way), type a few letters to locate a person’s name and press Enter to enter the score. Then press

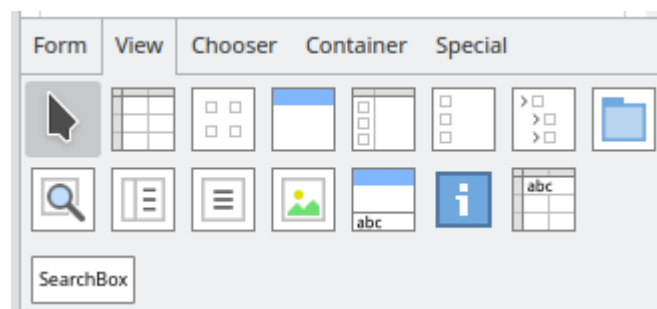
Enter again to leave the cell and be ready to search for the next name. The tableview property that is critical here is *NoKeyboard*. The menu item sets or unsets it.

Steps:

1. Start a new graphical application.
2. Create a new class, make it Exported and call it “SearchBox”. (Rt-click the Sources folder > New > Class...)
3. Enter the line `INHERITS TableView` right at the top of the class.



4. Press F5 to run the application. Quit the program straight away.



Now there should be a SearchBox among the classes. Drag one onto the form *FMain*. You have just made a new instance. You can peel off as many copies as you want. For the moment, we only need the one.

Now we teach our horse to fly. This code goes in the SearchBox class:

```
Export 'Without this you will not see the class in the toolbar of classes
Inherits TableView
```

```
Private ss As String
Public SearchColumn As Integer
Event EnterOnLine
```

```
Public Sub CheckKey()
```

```
    Select Case Key.Code
    Case Key.Esc, Key.BackSpace, Key.Delete
        ss = ""
        Me.UnselectAll
```

```

    Case Key.Enter, Key.Return
        Raise EnterOnLine 'action on pressing Enter
        ss = ""
    Case Key.Tab
        SearchDown
    Case Key.BackTab
        SearchUp
    Case Else
        ss &= Key.Text
        SearchDown
    End Select

End

Private Sub SearchUp()

    Dim i, Start As Integer
    If Me.Rows.Selection.Count = 0 Then Start = -1 Else Start =
Me.Rows.Selection[0] 'the selected line
    For i = Start - 1 DownTo 0
        If InStr(LCase(Me[i, SearchColumn].text), LCase(ss)) Then
            Me.Rows.Select(i)
            Return
        Endif
    Next
    For i = Me.Rows.max DownTo Start
        If InStr(LCase(Me[i, SearchColumn].text), LCase(ss)) Then
            Me.Rows.Select(i)
            Return
        Endif
    Next

End

Private Sub SearchDown()

    Dim i, Start As Integer
    If Me.Rows.Selection.Count = 0 Then Start = -1 Else Start =
Me.Rows.Selection[0]
    For i = Start + 1 To Me.Rows.Max 'if no selected line, start at top, else
start at next line
        If InStr(LCase(Me[i, SearchColumn].text), LCase(ss)) > 0 Then
            Me.Rows.Select(i)
            Return
        Endif
    Next
    For i = 0 To Start 'if no more occurrences, you will end up at the line you
are on
        If InStr(LCase(Me[i, SearchColumn].text), LCase(ss)) > 0 Then
            Me.Rows.Select(i)
            Return
        Endif
    Next

End

Public Sub HandleClick()
    If Me.Column = SearchColumn Then Return 'searchable column is not editable
by clicking
    ss = ""

```

```
Me.Edit
End
```

This above code is now part of all searchboxes. It still has to be called upon at the right times. If not, it will never get done. So here is the code for the main form *FMain*:

```
Public Sub sb1_EnterOnLine()
    sb1.Column = 1
    sb1.Edit
End

Public Sub Form_Open()

    Dim Names As New String[]
    Dim i As Integer
    Names = ["Mereka AIKE", "Ernest AIRI", "John AME", "Stanley ANTHONY",
"Natasha AUA", "Veronica AUFA", "John Taylor BUNA", "Romrick CLEMENT",
"Philomena GAVIA", "Richard GHAM", "Gerard BUZOLIC", "John HEARNE", "Thomas
EDISON"]
    sb1.Rows.Count = Names.count
    sb1.Columns.Count = 2
    For i = 0 To Names.Max
        sb1[i, 0].text = Names[i]
        If i Mod 2 = 0 Then
            sb1[i, 0].Background = &hDDDDFF
            sb1[i, 1].Background = &hDDDDFF
        End If
    Next
    sb1.Columns[0].Width = 140 '-1 for max needed width
    sb1.Mode = Select.Single
    sb1.NoKeyboard = True 'start with sb1 selecting the line when Enter is
pressed in a cell
    sb1.Expand = True
    sb1.SetFocus

End

Public Sub MenuQuit_Click() 'Yes, I put in a Quit menuitem in a File menu.
    Quit
End

Public Sub sb1_KeyPress()
    sb1.CheckKey()
End

Public Sub sb1_DblClick()
    sb1.Edit 'to edit the names in the searchable column, double-click one of
them
End

Public Sub sb1_Save(Row As Integer, Column As Integer, Value As String)
    sb1[Row, Column].text = Value
End

Public Sub sb1_Click()
    sb1.HandleClick
End
```

```
Public Sub MenuGoingDown_Click()  
    MenuGoingDown.Checked = Not MenuGoingDown.Checked  
    sb1.NoKeyboard = Not sb1.NoKeyboard  
End
```

This horse knows how to fly. This tableview, now known by the illustrious and noble name of *SearchBox*, knows how to search for occurrences of what you type.

You talk to the horse when the horse is listening. You talk to the *SearchBox* when it gives you events that you can intercept. Our particular *SearchBox*, *sb1*, gives us all the events that tableviews do, and one more. It has a homemade event *EnterOnLine*.

The horse does what you tell it if you use words it understands. When the form gets a *keypress* event from the searchbox, tell it to *CheckKey*. *SearchBox* knows how to check your key. *SearchBox* will happily scan upwards or downwards looking for the next occurrence of what you typed. If you are happy with the selected line it presents to you, it will notify you with the *EnterInLine* event. Otherwise it adds the key to the string of letters you have already typed and does some more searching, starting with the next line.

EnterOnLine is raised when the key you type gets checked. If you typed *Enter* or *Return*, the *EnterOnLine* event occurs. In the main window you decide what you are going to do about it (if anything). In our case, it means we have found the line we are after and we want to type in the second column.

In a sense, you talk to the class through events and the class talks to you through events that it itself raises.

Homemade events that your classes give you might be like a simple greeting (“Hello!”) or they can convey parameters (“Hello, I’ll be there in 5 minutes!”). Our *EnterOnLine* might tell us which line you pressed *Enter* on. Then the Event definition would then read

```
Event EnterOnLine(LinNum as integer)
```

And the event handler might read

```
Public Sub sb1_EnterOnLine(LinNum As Integer)  
    sb1.Row = LinNum  
    sb1.Column = 1  
    sb1.Edit  
End
```

When a line is highlighted its *Row* property is set to that row anyway, so it is not necessary. Like, why give a command to move to a row when you are already at that row? So the *LinNum* parameter is not necessary, but the horse talks to you through events it raises and it *could* tell you what line you are on when it sends you an event *if* you wanted it to.

Setting Up a Class

If there are things to do when a new instance of a class is created, like setting up how many rows and columns there should be in a tableview or the names of the column headings, the place to do it is in a special sub called `_new()`. Whenever a new example of the class is made with the *New* operator, this event is called.

In the SearchBox program above, we can take all the setting up that is done in the *Form_Open()* event handler and move it into the class itself. Delete the *Form_Open()* event, and in the SearchBox class put the code there. When you run the program it works exactly the same.

Now the class sets itself up. The form does not have to set up each one.

```
Public Sub _new()  
  
    Dim Names As New String[]  
    Dim i As Integer  
    Names = ["Mereka AIKE", "Ernest AIRI", "John AME", "Stanley ANTHONY",  
"Natasha AUA", "Veronica AUFA", "John Taylor BUNA", "Romrick CLEMENT",  
"Philomena GAVIA", "Richard GHAM", "Gerard BUZOLIC", "John HEARNE", "Thomas  
EDISON"]  
    Me.Rows.Count = Names.count  
    Me.Columns.Count = 2  
    For i = 0 To Names.Max  
        Me[i, 0].text = Names[i]  
        If i Mod 2 = 0 Then  
            Me[i, 0].Background = &hDDDDFF  
            Me[i, 1].Background = &hDDDDFF  
        End If  
    Next  
    Me.Columns[0].Width = 140 '-1 for max needed width  
    Me.Mode = Select.Single  
    Me.NoKeyboard = True 'start with sb1 selecting the line when Enter is  
pressed in a cell  
    Me.Expand = True  
    Me.SetFocus  
  
End
```

SQLite Databases



A database is a file on the hard drive that has a structure to it so that it can hold large amounts of information and access it quickly.

SQLite is one type of database. It was written by Dwayne Richard Hipp (born 1961 in North Carolina). It was first released in August 2000. It is public domain, meaning anyone can use it free of charge. Google Chrome, Firefox, the Android operating system for smartphones, Skype, Adobe Reader and the iPhone all use **SQLite**. It's just nice. And you pronounce it "S Q L Lite", so saith Wikipedia.

Databases store information in tables. Gambas has a tableview. This, too, has rows and columns. You can think of a database table as an invisible tableview in the database file.

Rows are called *Records*. Columns are called *Fields*.

For example, a teaching might have a database with a Students table. In that table there is a row for every student. Looking across the row you see *StudentID*, *FirstName*, *LastName*, *Sex*, *DateOfBirth*, *Address*, *PhoneNumber*. These are the fields. They are the columns.

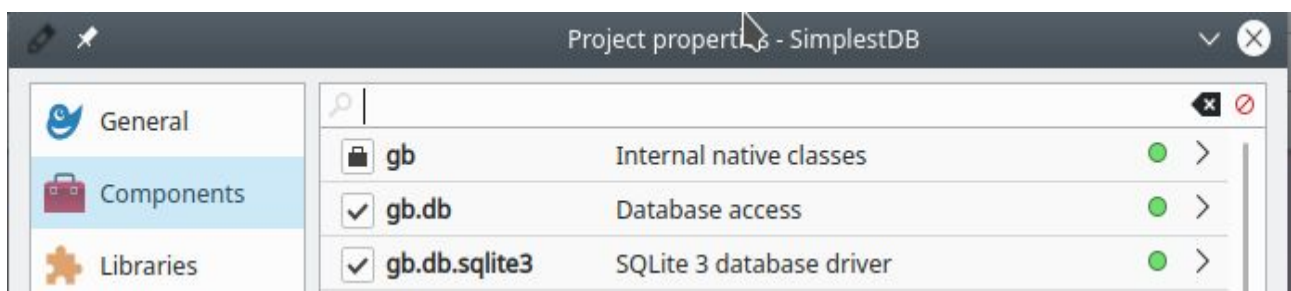
StudentID	FirstName	LastName	Sex	DateOfBirth	Address	PhoneNumber
2019001	Mary	Smith	F	2008-06-23	21 Holly Crt, Bundaberg	07324657
2019002	Jim	Jones	M	2003-02-19	14 Primrose St, Bundaberg	07123456
2019003	Lucy	Watkins	F	2003-10-05	5 Flower St, Bundaberg	07938276

This could be a TableView, or a Table in a Database file.

Every database table has to have a Primary Key. Every record must have a unique value for this field: one that no one else shares. The simplest is to call it RecID and number 1, 2, 3... etc. In the table above, the primary key is going to be the StudentID and it is an integer. The first four digits are the year of enrolment. (We could have another column for YearOfEnrolment and just use a sequence number for the StudentID.)

In SQLite all data is stored as strings, even though you might specify some columns as integers, others as strings and others as dates. SQLite is very forgiving: you can put things that aren't numbers into integer columns and so on, but try not to. Empty cells are NULL. Try to avoid those, too. When you make a new blank record, initialise values to the empty string, "".

Including the Database Facility



SQLite is a component (optional part) of Gambas. There is also a Database access component. On the Project Menu > Properties... > Components page, be sure to tick **gb.db** and **gb.db.sqlite3**. Without these components in your project you will get errors as soon as you try to run your program.

SQL — Structured Query Language

You send messages to **SQLite** and it sends answers back to you using a special language called **SQL** (“S Q L” or “sequel”, pronounce it either way.) This means learning another language, but the simple statements that are used most frequently are not difficult to learn. They are the only ones I know, anyway. **SQL** was invented by Donald D. Chamberlin and Raymond F. Boyce and first appeared in 1974. **SQL** is so universal that everyone who writes databases knows of it. It is an international standard. **SQLite** is one implementation of it.

For example, you might send a message to **SQLite** saying

```
SELECT * FROM Students
```

This says, “select everything from the Students table”. This gives you the whole table. Or you might only want the students who are male:

```
SELECT * FROM Students WHERE Sex = 'M'
```

Perhaps you want everyone, but you want the females first and the males second:

```
SELECT * FROM Students ORDER BY Sex
```

That will get the females first, because “F” comes before “M”. The females will all be in random order and likewise the males unless you write

```
SELECT * FROM Students ORDER BY Sex DESC, LastName ASC
```

This returns a table to you with males first (alphabetically by surname) followed by females (alphabetically by surname).

You might only want the students names, so you could write

```
SELECT FirstName, LastName FROM Students ORDER BY LastName
```

Perhaps you want only those students who were enrolled in 2019. Now, this is part of the StudentID. You want only those students whose StudentID number starts with “2019”. You use a “wildcard”. The percent (%) sign means “anything here will do”.

```
SELECT FirstName, LastName FROM Students WHERE StudentID LIKE '2019%'
```

When you send these **SELECT** statements to the database, **SQLite** will send you back a table. Gambas calls it a **RESULT**.

Suppose you have a database called *db1* (as far as Gambas is concerned) and it is attached to *MyStudentDatabase.sqlite* which is the actual database file on your hard drive. You need a result to store the reply:

```
Dim res as Result  
res = db1.exec("SELECT * FROM Students")
```

res has the information you asked for. You might want to print the information, or show it in a tableview, or hold it internally in arrays so you can do calculations on it. You need to cycle through the records thus:

```
While res.Available  
  'do something with res!FirstName, res!LastName and res!DateOfBirth etc  
  res.MoveNext  
Wend
```

For displaying information in a tableview there is a special event that is triggered each time a cell has to have its contents painted on the screen. It is particularly useful if your set of records is large. The tableview does not have to hold all the information from all the records in itself. It can get the information as it needs it for when it has to be displayed. Be a little careful here: if you are depending on all the information being in the tableview, it may or it may not be all there. This is an example of using the *_Data* event, getting the information from the result table *res* when it is needed to display a particular cell in the tableview:

```
Public Sub TableView1_Data(Row As Integer, Column As Integer)  
  res.MoveTo(row)  
  If Column = 0 Then  
    TableView1.Data.Text = res!FirstName  
  Else  
    TableView1.Data.Text = res!LastName  
  Endif  
End
```

Notice the use of *TableView1.Data.Text* , which represents the text in the cell.

Notice we have **result.MoveTo** to go to a particular record, **result.MoveNext** if we are stepping through them one at a time, and **result.Available** to check to see if there is another record to *MoveNext* to. Useful in setting the number of rows to have in your tableview is **result.RecordCount**.

Besides accessing the information in the database, with databases you want to be able to:

1. *Add* records
2. *Delete* records
3. *Modify* records

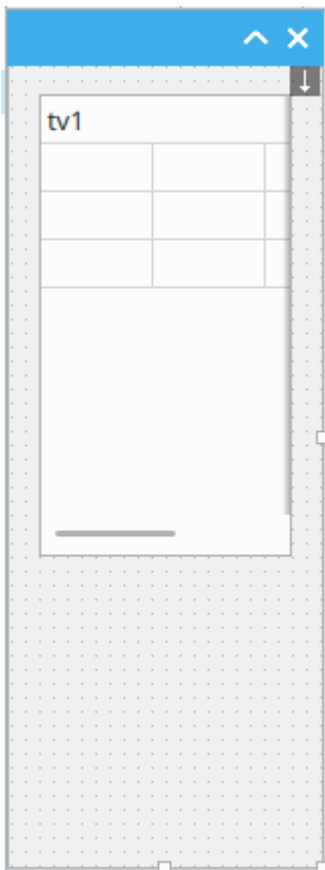
All but the simplest databases have more than one table in them. Tables can be linked to each other, so records can have signposts in them to indicate lines in other tables that apply. The signpost is the **record ID** or other **primary key** of a record in another table. For example, a database of political candidates might have a signpost to the party they belong to. **SQL** is so smart it can look up the two tables at once to provide you with the information you need, for example this ‘join’ of two tables. (Candidates are in a particular party, and it is the parties that have policies on a variety of issues.)

```
SELECT Candidate, PolicyOnPensions FROM Candidates, Parties WHERE  
Candidate.PartyID = Parties.PartyID AND Candidates.Electorate="Fairfax"
```

Database with a Single Table, to be filled with Random Numbers

The next program comes from <https://kalaharix.wordpress.com/Gambas/creating-a-databases-and-tables-from-Gambas/> slightly rearranged. It creates a database in your home folder called *Test.sqlite*, fills it with random two-digit numbers, then accesses the database to show them in a tableview.

You need a form with a tableview called *tv1*. Make it long and thin, as it has 2 columns.



RecID	Value
9985	13
9986	10
9987	83
9988	30
9989	97
9990	89
9991	58
9992	36
9993	23
9994	15
9995	17
9996	75
9997	52
9998	21
9999	44
10000	71

' Gambas class file

```
Private db1 As New Connection
Private rs As Result
```

```
Public Sub SetupTableView()
```

```
    tv1.header = GridView.Horizontal
    tv1.grid = True
    tv1.Rows.count = 0
    tv1.Columns.count = 2
    tv1.Columns[0].text = "RecID"
    tv1.Columns[1].text = "Value"
    tv1.Columns[0].width = 55
    tv1.Columns[1].width = 55
```

```
End
```

```
Public Sub CreateDatabase()
```

```
    db1.Type = "sqlite"
    db1.host = User.home
    db1.name = ""
```

```
    'delete an existing test.sqlite
```

```
    If Exist(User.home & "/Test.sqlite") Then
        Kill User.home & "/Test.sqlite"
    Endif
```

```
    'create test.sqlite
    db1.Open
```

```

    db1.Databases.Add("Test.sqlite")
    db1.Close

End

Public Sub MakeTable()

    Dim hTable As Table

    db1.name = "Test.sqlite"
    db1.Open
    hTable = db1.Tables.Add("RandomNumbers")
    hTable.Fields.Add("RecID", db.Integer)
    hTable.Fields.Add("Value", db.Integer)
    hTable.PrimaryKey = ["RecID"]
    hTable.Update

End

Public Sub FillTable()

    Dim i As Integer
    Dim rs1 As Result

    db1.Begin
    rs1 = db1.Create("RandomNumbers")
    For i = 1 To 10000
        rs1!RecID = i
        rs1!Value = Rand(10, 99)
        rs1.Update
    Next
    db1.Commit

Catch
    db1.Rollback
    Message.Error(Error.Text)

End

Public Sub ReadData()
    'read the database
    Dim SQL As String = "SELECT * FROM RandomNumbers"
    rs = db1.Exec(SQL)
End

Public Sub Form_Open()
    SetupTableView
    CreateDatabase
    MakeTable
    FillTable
    ReadData
End

Public Sub Form_Activate()
    'change the rowcount of the gridview from 0 to the number of records.
    'This triggers the data handling event
    tv1.Rows.Count = rs.Count
End

Public Sub tv1_Data(Row As Integer, Column As Integer)

```

```

rs.moveTo(row)
If Column = 0 Then tv1.Data.Text = rs!RecID Else tv1.Data.Text = rs!Value
'If Column = 0 Then tv1.Data.Text = Str(rs["RecID"]) Else tv1.Data.Text =
Str(rs["Value"])
'Either of these two lines will do it.
End

Public Sub Form_Close()
    db1.Close
End

```

When you work with a database a temporary “journal” file is created. That file is incorporated into the database when it is “committed”. If you don’t want to commit, you “rollback” the database to what it was before you made these latest changes. The temporary file contains the “transaction”, meaning the latest work you have just done to change the database. That is what the *db1.Begin*, *db1.Commit* and *db1.Rollback* mean.

The above program is a good template to adapt when making a database.

A Cash Spending Application

This application saves records of cash spending. You can allocate each expenditure to a category. Each time you allocate to a category, totals are worked out for the categories and you can see what fraction of your spending went to each of the categories.

If you know you spent, say, €100, and you can only account for, say €85, you can distribute the remaining €15 among the categories.

Before letting loose on the code and after a look at the form we shall take a look at the process of designing such an application.

The screenshot shows the FMain application window with a menu bar (File, Data, Help) and a status bar (0 * 0, 833 x 511). The window is divided into two main sections: 'Spending' (pink background) and 'Categories' (purple background). Each section contains a table with 4 rows and 5 columns. The 'Spending' table has a header row with 'labSpendingTotal' and a data row with 'tv1'. The 'Categories' table has a header row with 'labCategoriesTotal' and a data row with 'tvCategories'. Below each table is a 'Target' field with a 'tbTarget' button, an '= Done:' field with a 'tbDone' button, and a '+ Still to do:' field with a 'tbToDo' button. The 'Categories' section also has an 'Amount:' field with a 'Distribute' button.

The screenshot shows the File menu with the following items: New Data File..., Open Data File... (Ctrl+O), and Quit (Ctrl+Q).

The screenshot shows the Data menu with the following items: New Spending (Ctrl+N), New Category (Ctrl+K), Clear Spending, Clear Categories, Insert Default Categories, Round Category Totals (Ctrl+R), No Selection (Ctrl+Space), Recalculate (F4), and Copy (Ctrl+C).

The *File* menu has items *MenuNewDatabase*, *MenuOpen* and *MenuQuit*.

The *Data* Menu has items *MenuNewSpending*, *MenuNewCategory*, *MenuClearSpending*, *MenuClearCategories*, *MenuRound*, *MenuUnselectAll*, *MenuCalculate* and *MenuCopy*.

The *Help* menu is optional.

The textbox whose name you cannot quite see above is *tbDistribute*.

The screenshot shows a window titled "/home/gerard/TestSpending.db" with a menu bar (File, Data, Help). It contains two table views side-by-side.

Spending Table View: The title bar shows "Spending" and a total of "85.00". The table has 5 columns: an index, Date, Amount, Category, and Comment. It contains 8 rows of data.

	Date	Amount	Category	Comment
1	1-2-2019	20.00	Papers etc	Linux magazine
2	2-3-2019	10.00	Medical	Cough medicine
3	3-4-2019	15.00	Travel	Train fares
4	4-5-2019	5.00	Papers etc	Newspaper
5	5-6-2019	12.00	Provisions	Takeaways
6	6-7-2019	8.00	Provisions	Milkshake
7	7-8-2019	10.00	Clothes	Cap
8	8-9-2019	5.00	Papers etc	Newspaper

Categories Table View: The title bar shows "Categories" and a total of "85.00". The table has 4 columns: an index, Total, %, and Category. It contains 9 rows of data.

	Total	%	Category
1	20	24	Provisions
2	15	18	Travel
3	10	12	Medical
4			Donations
5	30	35	Papers etc
6	10	12	Clothes
7			Personal
8			Phone
9			Repairs

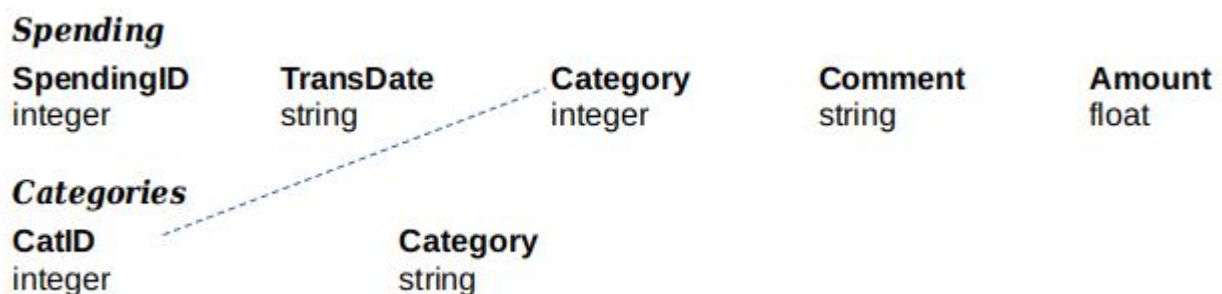
At the bottom, there are input fields for "Target: 100.00", "= Done: 85.00", "+ Still to do: 15.00", and "Amount: 15.00" with a "Distribute" button.

The program starts by opening the last database file that was open, or prompting to make a new one if it is the first time, or locating it if you sneakily moved it since the last time it was open. It also starts with a blank row in the *Spending* and *Categories* tableviews.

When a category is chosen for the selected spending line (click a category line anywhere except in the name column and press Enter) the category totals and percentages are recalculated.

Typing in the *Target* textbox is optional. If there is a number in it, "Still to do" will be calculated.

Internally, the database has two tables called *Spending* and *Categories*. You can see two tableviews corresponding to the two database tables. These are the fields in each table:



The two primary keys are *SpendingID* and *CatID*. They number the records in sequence (1, 2, 3...)

The *Spending* table's *Category* field contains a number which, when you look it up in the *Categories* table, gives you the category name. This is good: if you change the spelling of a category name you only have to change it once.

Hidden Columns

The user does not need to see the record IDs. They are internal to the database. They have to be unique: each record must have its own record ID. They are the primary keys of the *Spending* and *Categories* tables. They will be the very first columns in the tableviews, but they will be hidden from view (zero width). Also, in the *Spending* table, the user does not want to see the Category ID (the reference to one of the categories). It will be the last column in the *Spending* table, and also zero width. The columns start from zero, so it is column 5, just to the right of the *Amount* column.

List the Jobs

Having sketched out a form design and planned the tables and fields with pencil and paper, we next think of what we want the program to do. It is good to keep in mind the things databases do: *Add*, *Delete*, *Modify* (as well as display the data). Here is a list. These are going to be the subs.

Database

NewDatabase	Create a new database file on disk with its two tables
OpenDatabase	Open the database and display what is in it on startup

General

Calculate	Add up totals and work out percentages for each category
DoTotals	Grand totals for amounts in spending and categories tables
SetupTableViews	The right number of rows and columns and column headings

Spending Table

NewSpending	Add a record to the Spending table
ShowSpending	Display what is in the Spending table in tv1, the tableview
TidySpendingTable	The final part of ShowSpending, really. Alternating blue lines
SumSpending	Part of "DoTotals"; add up all the spending totals
	Clear a category (make it a right-click menu)
	Delete a record when you press DEL or BACKSPACE on a selected line

Categories Table

NewCategory	Add a record to the Categories table
ShowCategories	Display what is in the Categories table in tvCategories
TidyCategoriesTable	The final part of ShowCategories. Alternating blue lines.
SumCategories	Part of "DoTotals"; add up all the category amounts
	Insert default categories into the categories table (a menu item)
EnterOnCategoryLine	Enter on a line inserts category on the selected spending line.

	Delete a record when you press DEL or BACKSPACE on a selected line
--	--

Other Features

	Work out how much is left to allocate
	Distribute what is left among the categories
	A Help window
	Save what database we are using in Settings for next time
	Copy everything as text, to paste into a word processing document
	Round numbers to whole euros (and check totals are not out by one)
	A Quit menu item to close the program

Useful Functions

CategoryNameFromID	Given the CatID number, return the Category Name (a string)
Massage	Given the user's choice of filename, remove bad characters

Now it is time to program. Write the subs. Work out when they will be called on to do their work. Some can be consigned to menus. Some can happen when you click things. You are the one who is going to use this program: Do you want to click buttons? Do you want windows to pop up when you add a new category or a new spending transaction? Are there *nice* ways of doing things—intuitive ways—so things can happen naturally, as a new user might expect them to happen? We do some thinking and come up with some ideas:

- We can do with a blank row in each table to start with, that you can type in.
- When you finish typing in a cell, save that cell. Avoid having to click a Save button.
- When you press *Enter* in the last cell of the line, make a new line.
- When a category line is selected and the user presses *Enter*, put that category into whatever line in the spending table that is selected (highlighted). Move to the next spending line that doesn't have a category so you can click a category line and *Enter* it. So you can enter categories for all the lines at the end, after you have entered everything else.
- When you start, open the same database you had open last time. If none, give a choice of creating a new database or browsing to find the database that you moved or somebody may have given to you on a USB or in an email.
- Edit a category by clicking on it.
- Edit a cell in the spending table by clicking on it (except the category — just *Enter* on a line in the categories table to put a new one in.)
- When you allocate a spending line to a category, recalculate the percentages for all categories.

- When you change the total in the *Target* textbox, do a subtraction to find out how much you still have left to allocate.
- Put blanks into cells that have nothing in them rather than zeros.
- Pressing *Delete* or *backspace* in either of the tableviews will delete the selected (highlighted) line and delete its record from the database. No questions, no confirmation request—it just does it. Only one line can be deleted at a time, and it is easy enough to re-enter if you press *Delete* by mistake.
- If the first cell on a tableview row has a record ID number in it, the record exists and saving just has to update it. If it is blank, the database has to first create a new record, giving it the next highest record number, put its record number in the first cell, and then update it.

Here are the names of the objects on the form *FMain*:

Panels: Panel1 (pink), Panel2 (blue)

Labels saying “Spending”, “Categories”, “Target:”, “= Done:”, “+ Still to do:”, “Amount:”

Labels called “LabSpendingTotal” and “LabCategoriesTotal” top right of the tableviews.

TableViews: tv1 for spending and tvCategories

TextBoxes: tbTarget, tbDone, tbToDo, tbDistribute

Button: bDistribute

File Menu: MenuNewDatabase, MenuOpen (Ctrl-O), MenuQuit (Ctrl-Q)

Data Menu: MenuNewSpending (Ctrl-N), MenuNewCategory (Ctrl-K), MenuClearSpending, MenuClearCategories, MenuDefaultCategories, MenuRound (Ctrl-R), MenuUnselectAll (Ctrl-Space), MenuCalculate (F4), MenuCopy (Ctrl-C)

Help Menu: Help and Instructions (F1) (Opens a separate form called Help. Put on it what you like.)

Category Menu (invisible, so it is not on the main menubar): *MenuClearCategory* (This one pops up with you right-click a category cell in the spending table.)

Here is the code. Following it is an explanation of the SQL statements.

```
Public fdb As New Connection 'finance database
Public rs As Result 'result set after querying database
Public SQL As String

Public Sub Form_Open()

    SetUpTableViews
    If IsNull(Settings["Database/host"]) Then
        Select Case Message.Question("Create a new data file, or open an existing
one?", "New...", "Open...", "Quit")
            Case 1 'new
```

```

        NewDatabase
    Case 2 'open
        OpenDatabase(Null, Null)
    Case Else
        Quit
    End Select
Else
    OpenDatabase(Settings["Database/host"], Settings["Database/name"])
Endif

End

Public Sub Form_Close()
    fdb.Close 'close connection
End

Public Sub OpenDatabase(dbHost As String, dbName As String) 'if these are
null, ask where the database is

    If Not Exist(dbHost & "/" & dbName) Or IsNull(dbHost) Then 'it's not where it was
last time, or path not supplied
        Dialog.Title = "Where is the database?"
        Dialog.Filter = ["*.db"]
        Dialog.Path = User.Home & "/Documents/"
        If Dialog.OpenFile() Then Return ' User pressed Cancel; still can't open a
database
        Dim s As String = Dialog.Path
        Dim p As Integer = RInStr(s, "/") 'position of last slash
        fdb.host = Left(s, p)
        fdb.Name = Mid(s, p + 1)
    Else
        fdb.host = dbHost
        fdb.Name = dbName
    End If

    Try fdb.Close
    fdb.type = "sqlite3"
    Try fdb.Open
    If fdb.Opened Then
        FMain.Caption = fdb.host & "/" & fdb.Name
        Settings["Database/host"] = fdb.host
        Settings["Database/name"] = fdb.Name
    Else
        Message.Info("<b>Couldn't connect.</b><br><br>... please try again or
create a new database.")
        Return
    Endif

    ShowSpending
    ShowCategories
    Calculate

End

Public Sub NewDatabase()

    Dialog.Path = User.Home & "/" 'setting it to "~/" doesn't work
    Dialog.Title = "Create a New Database"
    If Dialog.SaveFile() Then Return 'clicked Cancel
    Dim s As String = Dialog.Path & ".db"

```

```

Dim p As Integer = RInStr(s, "/") 'position of last slash
Dim FName As String = Mid(s, p + 1)
fdb.host = Left(s, p)
fdb.Name = "" 'This MUST be left blank. If not, database file will not be
created
fdb.Type = "sqlite3"

If Exist(s) Then Kill s 'delete existing file of that name
fdb.Close
Try fdb.Open 'opens a connection to the database; do this after setting
properties and before creating
If Error Then
    Message("Unable to open the database file<br><br>" & Error.Text)
    Return
Endif
fdb.Databases.Add(FName) 'does the creating

fdb.Close
Dim dbTable As Table
fdb.name = FName
Try fdb.Open
If Not fdb.opened Then
    Message("Unable to open the data file")
    Return
Endif
dbTable = fdb.Tables.Add("Spending")
dbTable.Fields.Add("SpendingID", db.Integer)
dbTable.Fields.Add("TransDate", db.String)
dbTable.Fields.Add("Category", db.Integer)
dbTable.Fields.Add("Comment", db.String)
dbTable.Fields.Add("Amount", db.Float)
dbTable.PrimaryKey = ["SpendingID"]
dbTable.Update
rs = fdb.Create("Spending")
If fdb.Error Then
    Message("Couldn't create the Spending table.<br><br>: " & Error.Text)
    Return
Endif

rs!SpendingID = 1
rs!TransDate = ""
rs!Category = 0
rs!Comment = ""
rs!Amount = 0.0
rs.Update
fdb.Commit
If fdb.Error Then
    Message("Couldn't save a first record in the Spending table.<br><br>: " &
Error.Text)
    Return
Endif

fdb.Close
fdb.name = FName
Try fdb.Open
If Not fdb.opened Then
    Message("Unable to open the data file")
    Return
Endif
dbTable = fdb.Tables.Add("Categories")

```

```

dbTable.Fields.Add("CatID", db.Integer)
dbTable.Fields.Add("Category", db.String)
dbTable.PrimaryKey = ["CatID"]
dbTable.Update
rs = fdb.Create("Categories")
If fdb.Error Then
    Message("Couldn't create the Categories table.<br><br>: " & Error.Text)
    Return
Endif

rs!CatID = 1
rs!Category = ""
rs.Update
fdb.Commit
If fdb.Error Then
    Message("Couldn't save a first record in the Categories table.<br><br>: "
& Error.Text)
    Return
Endif

End

Public Sub DoTotals()

    labCategoriesTotal.Text = SumTheCategories()
    labSpendingTotal.text = SumTheSpending()
    tbDone.Text = labSpendingTotal.Text

End

Public Sub ShowSpending()

    rs = fdb.Exec("SELECT * FROM Spending")
    Dim L, CatID As Integer
    Dim CatName As String
    tv1.Rows.Count = 0 'clear
    If Not IsNull(rs) Then
        While rs.Available
            tv1.Rows.Count += 1
            L = tv1.Rows.max
            tv1[L, 0].text = rs!SpendingID
            tv1[L, 1].Text = rs!TransDate
            tv1[L, 2].Text = Format(rs!Amount, "0.00")
            CatName = rs!Category
            If Not IsNull(CatName) Then
                CatID = If(IsNull(Val(CatName)), -1, Val(CatName))
                If CatID > -1 Then tv1[L, 3].Text = CategoryNameFromID(CatID)
            Endif
            tv1[L, 4].Text = rs!Comment
            tv1[L, 5].Text = rs!Category 'Category ID in this hidden column
            rs.MoveNext
        Wend
    Endif
    If tv1.Rows.Count = 0 Then tv1.Rows.Count = 1
    TidySpendingTable

End

Public Sub ShowCategories()

```

```

rs = fdb.Exec("SELECT * FROM Categories")
Dim L As Integer
Dim t As Float
tvCategories.Rows.Count = 0 'clear
If Not IsNull(rs) Then
    While rs.Available
        tvCategories.Rows.Count += 1
        L = tvCategories.Rows.max
        tvCategories[L, 0].text = rs!CatID
        tvCategories[L, 3].Text = rs!Category
        rs.MoveNext
    Wend
Endif
If tvCategories.Rows.Count = 0 Then tvCategories.Rows.Count = 1
TidyCategoriesTable

End

Public Sub NewSpending()
    tv1.Rows.count = tv1.Rows.count + 1
    tv1.MoveTo(tv1.Rows.Max, 1)
    tv1.Edit
End

Public Sub NewCategory()
    tvCategories.Rows.count = tvCategories.Rows.count + 1
    tvCategories.row += 1
    tvCategories.Edit
End

Public Sub tv1_Insert()
    NewSpending
End

Public Sub tvCategories_Insert()
    NewCategory
End

Public Sub tv1_Click()

    Select Case tv1.Column
        Case 1, 2, 4
            tv1.Edit
        Case 3
            If tvCategories.Rows.Count > 0 Then
                tvCategories.SetFocus
                tvCategories.Rows[0].Selected = True
            Endif
        End Select
    End

End

Public Sub tvCategories_Click()
    If tvCategories.Column = 3 Then tvCategories.Edit
End

Public Sub SetUpTableViews()

    Dim i As Integer
    tv1.Columns.count = 6

```

```

tv1.Rows.count = 1
tv1.Columns[0].Width = 0
tv1.Columns[1].Alignment = Align.Center
tv1.Columns[2].Alignment = Align.Right
For i = 1 To tv1.Columns.Max - 1
    tv1.Columns[i].Width = Choose(i, 80, 80, 130, tv1.Width - tv1.ClientW -
306)
    tv1.Columns[i].Text = Choose(i, "Date", "Amount", "Category", "Comment")
Next
tvCategories.Columns.count = 4
tvCategories.Rows.count = 1
tvCategories.Columns[0].Width = 0
For i = 1 To tvCategories.Columns.Max
    tvCategories.Columns[i].Width = Choose(i, 60, 60, tvCategories.Width -
tvCategories.ClientW - 350)
    tvCategories.Columns[i].Text = Choose(i, "Total", "%", "Category")
Next
tvCategories.Columns[1].Alignment = Align.right
tvCategories.Columns[2].Alignment = Align.Center
tv1.Columns[5].Width = 0

End

Public Sub TidySpendingTable()

    For i As Integer = 0 To tv1.Rows.Max
        For j As Integer = 0 To tv1.Columns.Max
            If j = 2 Or j = 3 Then tv1[i, j].Padding = 4
            If i Mod 2 = 1 Then tv1[i, j].Background = &hF0F0FF
        Next
    Next

End

Public Sub TidyCategoriesTable()

    For i As Integer = 0 To tvCategories.Rows.Max
        For j As Integer = 1 To tvCategories.Columns.Max
            tvCategories[i, j].Padding = 4
            If i Mod 2 = 1 Then tvCategories[i, j].Background = &hF0F0FF
        Next
    Next

End

Public Sub Massage(s As String) As String
    'Doesn't like spaces or hyphens in file names. Doesn't complain; just
    doesn't create the file.

    Dim z As String

    For i As Integer = 0 To Len(s) - 1
        If IsLetter(s[i]) Or IsDigit(s[i]) Or s[i] = "_" Or s[i] = "." Then z &=
s[i] Else z &= "_"
    Next
    Return z

End

```

```

Public Sub tvCategories_Save(Row As Integer, Column As Integer, Value As String)

    Dim RecID As Integer
    Dim OriginalValue As String = tvCategories[Row, Column].Text

    tvCategories[Row, Column].Text = Value
    If IsNull(tvCategories[Row, 0].Text) Then 'no record ID, so we need a new record
        Dim Res As Result
        SQL = "SELECT MAX(CatID) as 'TheMax' FROM Categories"
        Res = fdb.Exec(SQL)
        If IsNull(Res!TheMax) Then RecID = 1 Else RecID = Res!TheMax + 1
        tvCategories[Row, 0].Text = RecID
        SQL = "INSERT INTO Categories(CatID,Category) VALUES(" & RecID & ", '" & Value & "')"
        fdb.Exec(SQL)
        If fdb.Error Then Message("Couldn't save:<br><br>" & SQL & "<br><br>" & Error.Text)
    Endif
    'update the record
    RecID = tvCategories[Row, 0].Text
    SQL = "UPDATE Categories SET Category = '" & Value & "' WHERE CatID='" & RecID & "'"
    Try fdb.Exec(SQL)
    If fdb.Error Then Message("Couldn't save:" & SQL & "<br><br>" & Error.Text)
    If Value <> OriginalValue Then ShowSpending 'category name was changed
End

Public Sub tv1_Save(Row As Integer, Column As Integer, Value As String)

    Dim RecID As Integer

    Dim FieldName As String = Choose(Column, "TransDate", "Amount", "Category", "Comment")

    If IsNull(tv1[Row, 0].Text) Then 'There's no Record ID, so insert a new record
        Dim Res As Result
        SQL = "SELECT MAX(SpendingID) as 'TheMax' FROM Spending"
        Try Res = fdb.Exec(SQL)
        If IsNull(Res!TheMax) Then RecID = 1 Else RecID = Res!TheMax + 1
        tv1[Row, 0].Text = RecID
        SQL = "INSERT INTO Spending(SpendingID,TransDate,Amount,Category,Comment) VALUES('" & RecID & "', ' ', ' ', ' ', ' ')"
        Try fdb.Exec(SQL)
        If Error Then
            Message("Couldn't save: " & Error.Text)
            Return
        Endif
    Endif
    'update record
    RecID = tv1[Row, 0].Text
    SQL = "UPDATE Spending SET " & FieldName & " = '" & Value & "' WHERE SpendingID='" & RecID & "'"
    Try fdb.Exec(SQL)
    If Error Then
        Message("Couldn't save:" & SQL & "<br><br>" & Error.Text)
        Return
    Endif
End

```

```

If Column = 2 Then
    tv1[Row, Column].Text = Format(Val(Value), "###0.00")
    Calculate 'amount has changed
Else
    tv1[Row, Column].Text = Value
Endif

Catch
    Message("Couldn't save ... have you created and opened a database yet?")
    Stop Event 'Don't go automatically to the next cell. If you do, you'll get
    this message twice.

End

Public Sub tv1_KeyPress()

    Select Case Key.Code
        Case Key.BackSpace, Key.Del 'remove record
            Dim RecID As Integer = tv1[tv1.Row, 0].Text
            SQL = "DELETE FROM Spending WHERE SpendingID='" & RecID & "'"
            Try fdb.Exec(SQL)
            If Error Then
                Message("Couldn't delete<br><br>" & Error.Text)
            Else
                tv1.Rows.Remove(tv1.Row)
                If tv1.Rows.Count = 0 Then tv1.Rows.Count = 1
            Endif
        Case Key.Enter, Key.Return
            If tvCategories.Rows.Count > 0 Then
                tvCategories.SetFocus
                tvCategories.Rows[0].Selected = True
            Endif
    End Select

End

Public Sub tvCategories_KeyPress()

    Select Case Key.Code
        Case Key.BackSpace, Key.Del 'remove record
            Dim RecID As Integer = tvCategories[tvCategories.Row, 0].Text
            SQL = "DELETE FROM Categories WHERE CatID='" & RecID & "'"
            Try fdb.Exec(SQL)
            If Error Then
                Message("Couldn't delete<br><br>" & Error.Text)
            Else
                tvCategories.Rows.Remove(tvCategories.Row)
            Endif
        Case Key.Enter, Key.Return
            EnterOnCategoryLine 'action on pressing Enter
            tvCategories.UnselectAll
    End Select

End

Public Sub MenuClearSpending_Click()
    fdb.Exec("DELETE FROM Spending")
    tv1.Rows.count = 1
    tv1.Clear
End

```



```

Public Sub MenuClearCategories_Click()

    fdb.Exec("DELETE FROM Categories")
    tvCategories.Rows.count = 1
    tvCategories.Clear

End

Public Sub CategoryNameFromID(ID As Integer) As String

    Dim res As Result = fdb.Exec("SELECT Category FROM Categories WHERE CatID="
    & ID)

    If Not res.Available Then Return "?"
    If IsNull(res!Category) Then Return "-"
    Return res!Category

End

Public Sub EnterOnCategoryLine() 'apply this category to the selected
    Spending line

    If tv1.row < 0 Then Return
    If IsNull(tv1[tv1.row, 0].text) Then
        Message("Please save this spending record first by entering some other
        item of data; there's no record ID yet.")
        Return
    Endif
    tv1[tv1.row, 3].text = tvCategories[tvCategories.row, 3].Text
    Dim CategoryID As String = tvCategories[tvCategories.row, 0].Text
    Dim SpendingID As String = tv1[tv1.row, 0].text
    tv1[tv1.row, 5].text = CategoryID
    SQL = "UPDATE Spending SET Category='" & CategoryID & "' WHERE SpendingID='"
    & SpendingID & "'"
    Try fdb.Exec(SQL)
    If Error Then
        Message("Couldn't save the category<br><br>" & SQL & "<br><br>" &
        Error.text)
        Return
    Endif
    Calculate
    For i As Integer = tv1.row To tv1.Rows.Max
        If IsNull(tv1[i, 3].text) Then
            tv1.Rows[i].Selected = True 'select the next Spending row that needs a
            category
            tvCategories.SetFocus
            Return
        Endif
    Next
    tv1.SetFocus

End

Public Sub Calculate()

    Dim i, j, CategoryID As Integer
    Dim t, GrandTotal As Float
    Dim res As Result
    Dim s As String

```

```

For i = 0 To tvCategories.Rows.Max 'every category
    If IsNull(tvCategories[i, 0].Text) Then Continue
    CategoryID = tvCategories[i, 0].Text
    Try Res = fdb.Exec("SELECT Total(Amount) AS TotalAmount FROM Spending
WHERE Category=" & CategoryID)
    If Error Then
        Message("Couldn't total<br><br>" & Error.Text)
        Continue
    Endif
    While res.Available
        t = res!TotalAmount
        GrandTotal += t
        If t = 0 Then tvCategories[i, 1].Text = "" Else tvCategories[i, 1].Text
= Format(t, "##0.00")
        res.MoveNext
    Wend
Next
If GrandTotal = 0 Then Return
For i = 0 To tvCategories.Rows.Max
    s = tvCategories[i, 1].Text
    If Not IsNull(s) And If Val(s) > 0 Then tvCategories[i, 2].Text =
Format(100 * Val(s) / GrandTotal, "##0.##") Else tvCategories[i, 2].Text = ""
Next
tbDone.Text = Format(GrandTotal, "##0.00")
labSpendingTotal.Text = tbDone.Text
labCategoriesTotal.Text = SumTheCategories()
If Not IsNull(tbTarget.Text) Then
    tbToDo.Text = Format(Val(tbTarget.Text) - GrandTotal, "##0.00")
    tbDistribute.Text = tbToDo.Text
Endif

End

Public Sub SaveCategoriesTable()
    For i As Integer = 0 To tvCategories.Rows.Max
        SaveCategoryLine(i)
    Next
End

Public Sub SaveCategoryLine(i As Integer) 'i is the line number

    Dim RecID As Integer
    Dim t, pct As Float
    Dim s, CategoryName As String
    Dim res As Result

    RecID = Val(tvCategories[i, 0].Text)
    CategoryName = tvCategories[i, 3].Text
    t = If(IsNull(tvCategories[i, 1].Text), 0, Val(tvCategories[i, 1].Text))
    s = tvCategories[i, 2].Text
    pct = If(IsNull(s), 0, Val(s))
    If IsNull(RecID) Then 'new record needed
        res = fdb.Exec("SELECT Max(CatID) AS MaxCatID FROM Categories")
        RecID = res!MaxCatID + 1
        SQL = "INSERT INTO Categories(CatID,Category) VALUES(" & RecID & "," &
CategoryName & ")"
        fdb.Exec(SQL)
    If Error Then

```

```

        Message("Couldn't insert a new record<br><br>" & SQL & "<br><br>" &
Error.text)
        Return
    Endif
Else
        SQL = "UPDATE Categories SET Category='" & CategoryName & "' WHERE CatID="
& RecID
        Try fdb.Exec(SQL)
        'before checking Error, don't forget to use TRY. Otherwise Error will be
set and you'll seem to have an error when you don't
        If Error Then
            Message("Couldn't update a record<br><br>" & SQL & "<br><br>" &
Error.text)
            Return
        Endif
    Endif

End

Public Sub SumTheCategories() As String

    Dim t As Float
    Dim s As String

    For i As Integer = 0 To tvCategories.Rows.Max
        s = tvCategories[i, 1].Text
        If Not IsNull(s) Then t += Val(s)
    Next
    Return Format(t, "##0.00")

End

Public Sub SumTheSpending() As String

    Dim t As Float
    Dim s As String
    For i As Integer = 0 To tv1.Rows.Max
        s = tv1[i, 2].Text
        If Not IsNull(s) Then t += Val(s)
    Next
    Return Format(t, "##0.00")

End

Public Sub MenuCalculate_Click()
    Calculate
End

Public Sub tbTarget_LostFocus()

    If Not IsNull(tbTarget.text) Then tbTarget.Text = Format(Val(tbTarget.Text),
"##0.00") Else tbTarget.Text = ""
    Calculate

End

Public Sub tbTarget_KeyPress()
    If Key.Code = Key.Enter Or Key.Code = Key.Return Then FMain.SetFocus
End

```

```

Public Sub bDistribute_Click()

    Dim t, pct, y, z As Float

    If IsNull(tbDistribute.Text) Then Return
    Dim x As Float = Val(tbDistribute.Text)
    For i As Integer = 0 To tvCategories.Rows.Max
        If IsNull(tvCategories[i, 1].Text) Then Continue
        If IsNull(tvCategories[i, 2].Text) Then Continue
        t = Val(tvCategories[i, 1].Text)
        pct = Val(tvCategories[i, 2].Text)
        y = t + pct / 100 * x
        z += y 'running total
        If y = 0 Then tvCategories[i, 1].Text = "" Else tvCategories[i, 1].Text =
Format(y, "##0.00")
        SaveCategoryLine(i)
    Next
    labCategoriesTotal.text = Format(z, "##0.00")
    FMain.SetFocus

End

Public Sub tbDistribute_LostFocus() 'when leaving, fix the appearance

    If Not IsNull(tbDistribute.text) Then tbDistribute.Text =
Format(Val(tbDistribute.Text), "##0.00") Else tbDistribute.Text = ""

End

Public Sub tbDistribute_KeyPress() 'enter leaves the textbox
    If Key.Code = Key.Enter Or Key.Code = Key.Return Then FMain.SetFocus
End

Public Sub MenuDefaultCategories_Click()

    Try fdb.Exec("DELETE FROM Categories") 'it might be already cleared
    tvCategories.Rows.Count = 9
    tvCategories.Clear
    Dim s As String
    For i As Integer = 0 To 8
        s = Choose(i + 1, "Provisions", "Travel", "Medical", "Donations", "Papers
etc", "Clothes", "Personal", "Phone", "Repairs")
        tvCategories[i, 3].text = s
        tvCategories[i, 0].text = i + 1
        SQL = "INSERT INTO Categories(CatID,Category) VALUES(" & Str(i + 1) & ", '"
& s & "')"
        Try fdb.Exec(SQL)
        If Error Then Message("Couldn't insert a new record in the categories
table.<br><br>" & SQL & "<br><br>" & Error.Text)
    Next
    labCategoriesTotal.text = ""

End

Public Sub MenuRound_Click()

    Dim s As String
    Dim x, t As Float

    For i As Integer = 0 To tvCategories.Rows.Max

```

```

    s = tvCategories[i, 1].Text
    If IsNull(s) Then
        tvCategories[i, 1].Text = ""
    Else
        x = Round(Val(s))
        t = t + x
        tvCategories[i, 1].Text = x
    Endif
Next
labCategoriesTotal.Text = Format(t, "##0.00")
For i As Integer = 0 To tvCategories.Rows.Max
    s = tvCategories[i, 2].Text
    If Not IsNull(s) Then tvCategories[i, 2].Text = Round(Val(s))
Next

End

Public Sub MenuOpen_Click()
    OpenDatabase(Null, Null)
End

Public Sub MenuNewDatabase_Click()
    NewDatabase
End

Public Sub MenuNewSpending_Click()
    NewSpending
End

Public Sub MenuNewCategory_Click()
    NewCategory
End

Public Sub MenuQuit_Click()
    Quit
End

Public Sub MenuCopy_Click()
    Dim s, z As String
    Dim i, j As Integer

    For i = 0 To tv1.Rows.Max
        s = tv1[i, 1].Text
        For j = 2 To 4
            s &= gb.Tab & tv1[i, j].Text
        Next
        z &= If(IsNull(z), "", gb.NewLine) & s
    Next
    z &= gb.NewLine
    For i = 0 To tvCategories.Rows.Max
        s = tvCategories[i, 1].Text
        For j = 2 To 3
            s &= gb.Tab & tvCategories[i, j].Text
        Next
        z &= If(IsNull(z), "", gb.NewLine) & s
    Next
    z &= gb.NewLine & gb.NewLine & "Total Withdrawn: " & tbTarget.Text & gb.tab
    & " = Total Accounted For: " & tbDone.Text & gb.tab & " + Cash on hand: " &
    tbToDo.Text
    Clipboard.Copy(z)

```

```

End

Public Sub MenuShowHelp_Click()

    Help.ShowModal

End

Public Sub MenuClearCategory_Click()

    Dim RecID As Integer = tv1[tv1.row, 0].Text

    fdb.Exec("UPDATE Spending Set Category=' ' WHERE SpendingID=" & RecID)
    tv1[tv1.row, 3].Text = "" 'Cat text
    tv1[tv1.row, 5].Text = "" 'Cat ID
    Calculate

End

Public Sub MenuUnselectAll_Click()

    tv1.Rows.UnselectAll
    tvCategories.Rows.UnselectAll

End

```

SQL Statements

Some of these statements are used as they appear. Others are a string that is built up from parts. You might see SQL = Bits of the statement are SQL and the field name might be added to it in the right place and be stored in a variable, for example. Or perhaps the record ID might be in a variable called RecID. Use single quotes in the string that is sent to **SQLite**. Use double quotes when assembling the statement in Gambas.

SELECT * FROM Spending	Select everything from the Spending table
SELECT * FROM Categories	Select everything from the Categories table
SELECT MAX(CatID) as 'TheMax' FROM Categories	Get the highest CatID from the Categories table and call it "TheMax".
INSERT INTO Categories(CatID,Category) VALUES(123,'Entertainment')	Create a new record in the Categories table. Put 123 into the CatID field and Entertainment into the Category field.
UPDATE Categories SET Category = 'Entertainment' WHERE CatID='123'	The Categories table has to be updated. In the record with CatID equal to 123 , put Entertainment in the Category field.
SELECT MAX(SpendingID) as 'TheMax' FROM Spending	Find the biggest SpendingID in the Spending table and call it "TheMax".
INSERT INTO Spending(SpendingID,TransDate,Amount,Category,Co mment) VALUES('123',' ',' ',' ',' ')	Create a new record in the Spending table. SpendingID = 123 TransDate = a blank

	Amount = a blank Category = a blank Comment = a blank
UPDATE Spending SET TransDate = '4-11-2019' WHERE SpendingID='123'	Put "4-11-2019" into the TransDate field of the record in the Spending table that has a SpendingID of 123.
DELETE FROM Spending WHERE SpendingID='123'	Delete the record in the Spending table that has a record ID of 123.
DELETE FROM Categories WHERE CatID='123'	Delete the record in the Categories table that has a record ID of 123.
DELETE FROM Spending	Delete every record from the Spending table. All the data disappears, never to be seen again.
DELETE FROM Categories	Delete every record from the Categories table. All the category records, gone forever.
SELECT Category FROM Categories WHERE CatID=123	Give me the name of the category that goes with the CatID record number 123.
UPDATE Spending SET Category='4' WHERE SpendingID='123'	Set the Category field of record 123 of the Spending table to 4. This spending item goes into the fourth category, whatever that is. To find out what the fourth category is, look up the Categories table and find the record with CatID =4
SELECT Total(Amount) AS TotalAmount FROM Spending WHERE Category='4'	Get the sum of all the numbers in the Amount fields of all the records in the Spending table that have 4 in their Category field. Simply, add up all the amounts spent in category 4. Call the answer "TotalAmount"
SELECT Max(CatID) AS MaxCatID FROM Categories	Get the highest CatID from the Categories table. Call it MaxCatID .
SQL = "INSERT INTO Categories(CatID,Category) VALUES(4,Travel)"	Create a new Categories record. Set the CatID field equal to 4 and the Category to "Travel".
UPDATE Categories SET Category='Travel' WHERE CatID=4	Update the Categories record that has a record ID of 4. Put "Travel" into the Category field.
UPDATE Spending Set Category=' ' WHERE SpendingID=123	Put a blank into the Category field of Spending record 123

The statements are either **SELECT**, **INSERT**, **DELETE** or **UPDATE**.

The patterns are:

```

SELECT fields FROM table
SELECT fields FROM table WHERE field = something
SELECT * FROM table
SELECT Total(field) AS nameForIt FROM table
SELECT Max(field) AS nameForIt FROM table

```

```
INSERT INTO table(fields) VALUES(values)
DELETE FROM table
DELETE FROM table WHERE field = something
UPDATE table SET field = something WHERE keyfield = something
```

These are not the only SQL statements: there are many more. They are enough to get a working knowledge of SQL. Online help for SQLite can be found at

<http://www.sqlitetutorial.net/>

<https://www.tutorialspoint.com/sqlite/>

<http://sqlitetutorials.com/>

<https://www.w3schools.com/sql/>

A most important point about using the UPDATE statement:

**Be careful when updating records.
If you omit the WHERE clause, ALL records will be updated!**

For example, do not write this: **UPDATE Spending SET Amount=12.50** .This puts 12.50 into the *Amount* field of *every* record. *All* amounts become 12.50. You should say **UPDATE Spending SET Amount=12.50 WHERE SpendingID=42** .

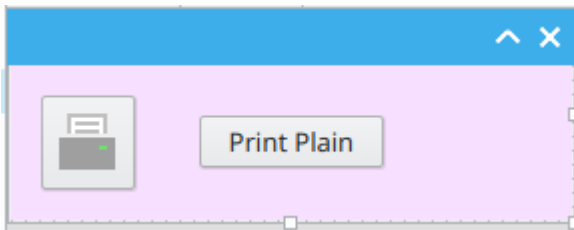
Printing

When practising printing, printing “to a file” will save paper. You can open the resulting PDF (*Portable Document Format*) file in your favourite PDF reader, such as *Okular*, and see on screen what you would get on paper.

This is about the simplest demonstration of printing. In your program you need a “**printer**”. We have used objects like buttons and tableviews. You can see them. A printer, though, is invisible. There is a printer class, just as there is a button class. You drag a printer onto your form just the same as you would drag a button or any other object. On the form it looks like a printer, but when the program runs it cannot be seen. It is really just a lump of code that is built into Gambas and does the things that printers are supposed to do, namely print and look after page sizes and orientation and so on. **Printer** is a clever little object.

First you tell your *Printer* object to configure, then you tell it to print. (“Printer, print!”, or as we write it in Gambas, *prn1.print*). When you tell it to print it will issue the *Draw* event. In the *draw* event you put things on the page that you want printed. You do this with all the abilities that another class has, the **Paint** class. The *Paint* class can put things onto the page for printing, but it has other uses too, such as painting into *DrawingAreas* or *ScrollAreas* on the form. Right: here we go!

Printing Some Plain Text



This small form has a *Printer* object and a button called *bPlainPrint*.

```
Public SimpleText As String
Public Sub Form_Open()

    SimpleText = "Countries of the World<br><br>Papua New Guinea<br><br>Papua
New Guinea is a country that is north of Australia. It has much green
rainforest. It has beautiful blue seas. Its capital, located along its
southeastern coast, is Port Moresby.<br><br>This is plain text in Linux
Libertine 12 point.<br><br>John Smith, editor"

End

Public Sub pr1_Draw()
    Paint.Font = Font["Linux Libertine,12"]
    Paint.DrawRichText(SimpleText, 960, 960, Paint.Width - 2 * 960)
End

Public Sub bPrintPlain_Click()
    If pr1.Configure() Then Return 'if user clicks Cancel, don't continue
    pr1.Print
End
```

When the form opens, some text is put in a variable called *SimpleText* for printing.

When the button is clicked the printer *pr1* is told to configure itself. If the user clicks the *Cancel* button this returns the value of *True*, so we should do nothing more. Otherwise, dear friendly printer object, please print.

The printer object *pr1* sends us the *Draw* event. It is saying, “I want to draw something! Please tell me what to paint on the page!”. We oblige by saying

```
Paint.Font = Font["Linux Libertine,12"]
Paint.DrawRichText(SimpleText, 960, 960, Paint.Width - 2 * 960)
```

Paint.Font is a property describing the font. It is a property with parts to it. We assemble those parts using *Font[something]*. The something is a string. For example, **Font["Bitstream Vera Serif,Bold,24"]** means “assemble a font that is Bitstream Vera Serif, bold, 24 point”. That is put in the *Font* property of the *Paint* thing. Actually the *Paint* thing is just a collection of skills. It is nothing you can see. It is another invisible class. **Be careful not to put spaces in that string unless part of the font name.** Gambas *Help* warns you of this. No spaces either side of the commas!

Paint.DrawRichText(something) is one of *paint*’s skills. It is a method it knows how to do. It needs at least three things in brackets. It can take a few more. Here we have four “arguments”, or “things in brackets”. First item: what to print. Second item: how far across to begin printing. Third item: how far down to begin printing. The 960 will give an inch margin. 96 dots per inch is a typical default printer resolution. The number is “tenths of a dot”. (I hope I have that right.) Fourth item: how wide is my printing going to be? Answer: the full width that *Paint* will allow less an inch on the left and an inch on the right. Each inch is 960. Take away two of them.

**
** means “break”, which goes to the next line. **

** means go to a new line, then go to a new line again. It gives us a blank line.

Rich Text understands **
. It also understands quite a few other symbols planted in the text. There are symbols to make it print using “Heading 1” style, and “Heading 2” and so on. You cannot change what these styles look like, though. They are built in and that is that. You can also change fonts and print size and colour anywhere in your text. These codes make the print come out a certain way. In fact, it is a language in itself: *HyperText Markup Language*, or **HTML. For example, to switch on *Bold*, put in this tag: ****. When you want to switch *Bold* off, put in this one: ****.

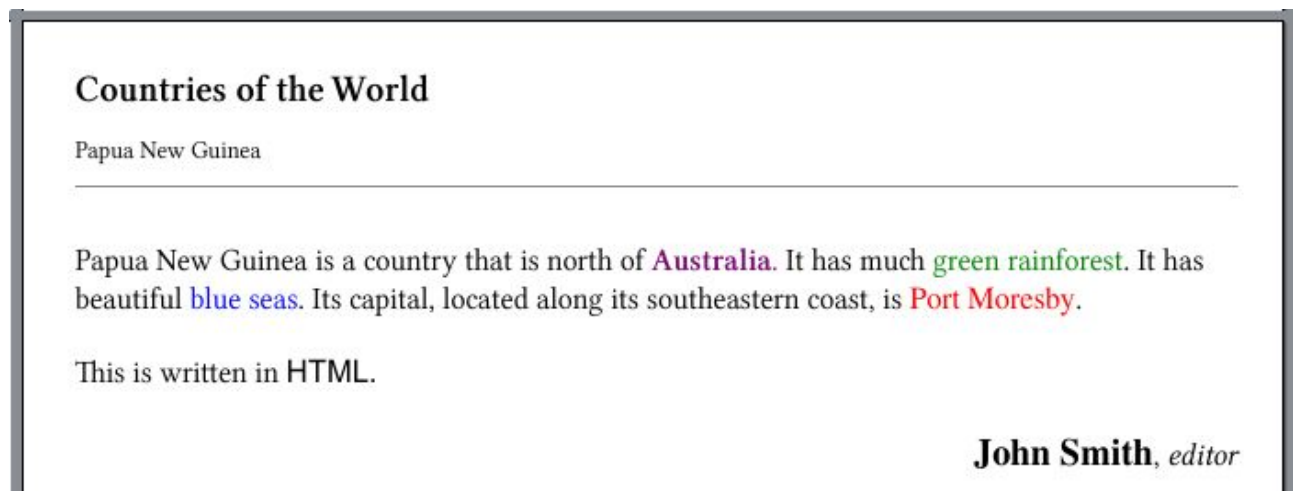
Printing Rich Text (with HTML tags in it)

Instead of *PlainText*, get this to print:

```
FancyText = "<h3>Countries of the World</h3><Small>Papua New
Guinea</Small><hr><br>Papua New Guinea is a country that is north of <font color =
#780373><b>Australia</b>.</font><font color = black> It has much</font><font color = green>
green rainforest</font><font color = black>. It has beautiful <font color = blue>blue
seas</font><font color = black>. Its capital, located along its southeastern coast, is <Font
Face=Times New Roman, Color=#FF0000>Port Moresby</font>.<br><br>This is written in <font
face = Arial>HTML.<br></font><p align = right><font face = Times New Roman,
Size=+1><b>John Smith</b></font>, <i>editor</i></p>"
```

Incidentally, that text, if saved in a text file with the extension .html, will open and display in a web browser, such as FireFox. You can try it.

The result will be:



I used Heading 3 (`<h3> ... </h3>`) because Heading 1 was gross.

There are many tags in the text to make it look like that. Gambas allows these tags. It is only a small selection from the full **HTML**. Save a document in **HTML** in your word processor, open it in a text editor like *Kate*, and be amazed.

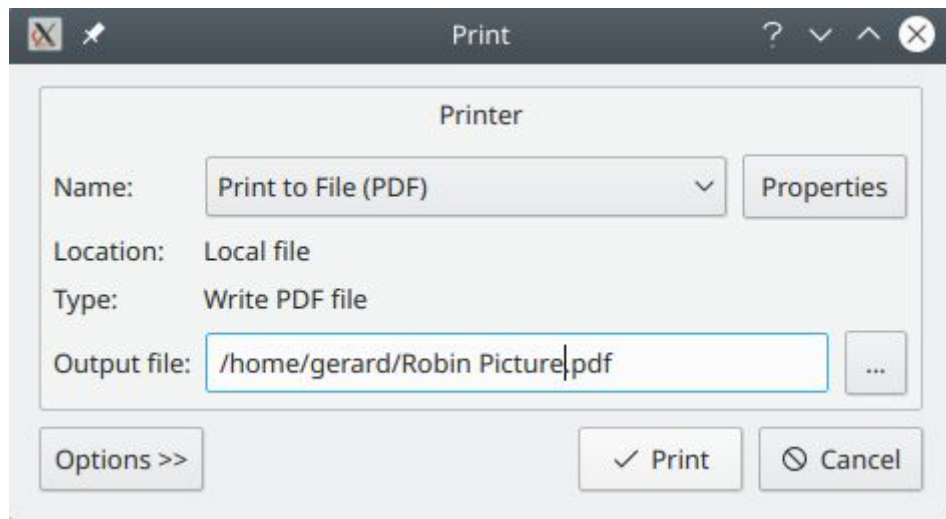
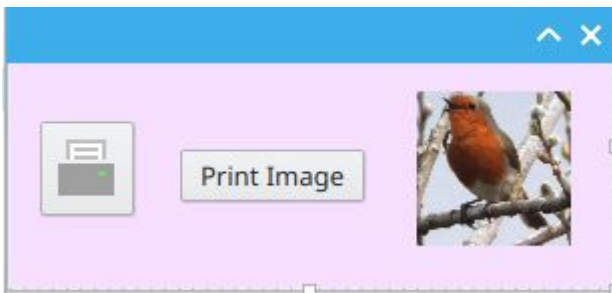
Tags that can be used in Rich Text

<code><h1></code> , <code><h2></code> , <code><h3></code> , <code><h4></code> , <code><h5></code> , <code><h6></code> → Headlines	<code><sup></code> → Superscript
<code></code> → Bold font	<code><small></code> → Small
<code><i></code> → Italic	<code><p></code> → Paragraph
<code><s></code> → Crossed out	<code>
</code> → Line break
<code><u></code> → Underlined	<code><a></code> → Link
<code><sub></code> → Subscript	<code></code> → Font

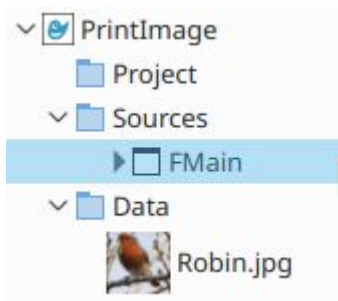
Examples to change text colour and align a paragraph:

<code> ... </code>	<code><p align=right> ... </p></code>
---	---

Print an Image on a Page



Print.Configure()



Properties	Hierarchy
PictureBox1	PictureBox
Picture	Robin.jpg
PopupMenu	
Public	False
Stretch	True

Drag a picture file onto the *Data* folder. Set the *Picture* property of the PictureBox to it.

The Printer is named *pr1*.

```

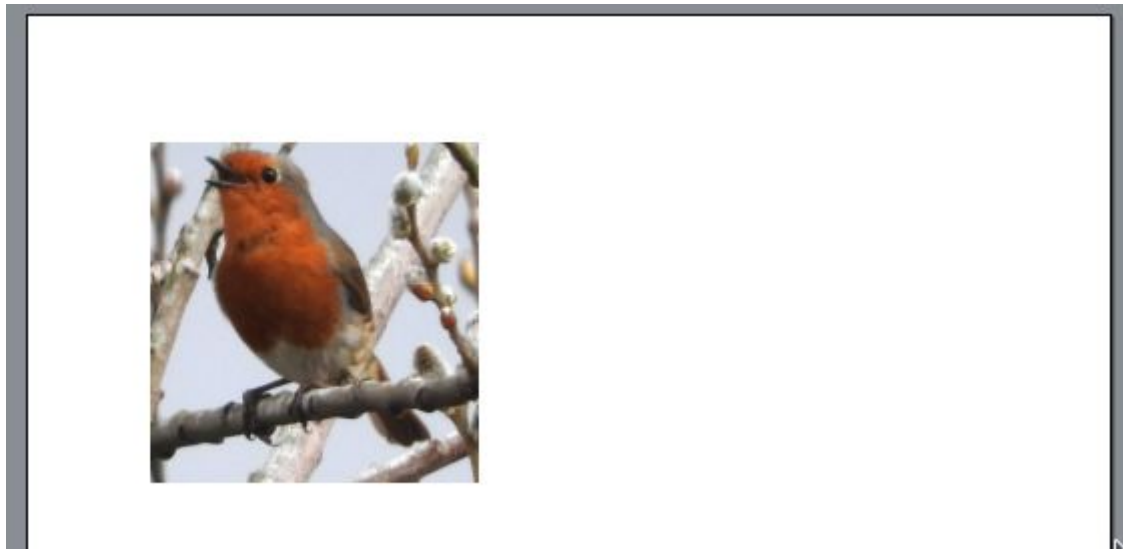
Public Pic As Picture

Public Sub pr1_Draw()
    Paint.DrawPicture(Pic, 960, 960, 3000, 3000)
End

Public Sub bPrint_Click()
    If pr1.Configure() Then Return 'if user clicks Cancel, don't continue
    Pic = PictureBox1.Picture
    pr1.Print
End

```

The picture is scaled to be 3000 x 3000. When I print to a file, the resolution is 96 dots per inch (96 dpi). The picture is printed 1 inch from the top and left margins and is scaled to fit into about 3 inches x 3 inches (3000x3000).



Print a Class List

Jessica Smith																			
William Moreton																			
Olivia Roberts																			
Amelia Taylor																			
Jacob Wilson																			
Isla Martin																			

In this program, 40 names are invented and put in an array called *Z[]*. If you were serious, the list of names could be typed in by the user or loaded from a names file or obtained from a database.

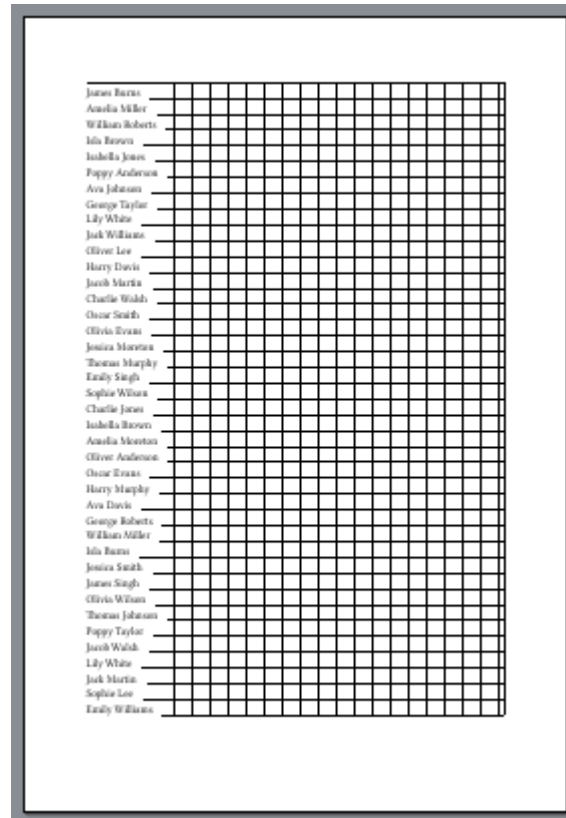
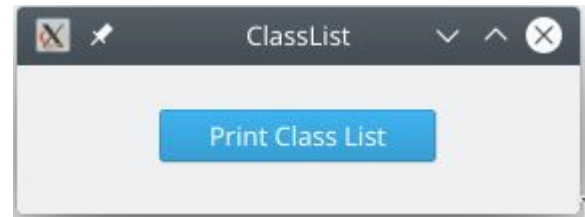
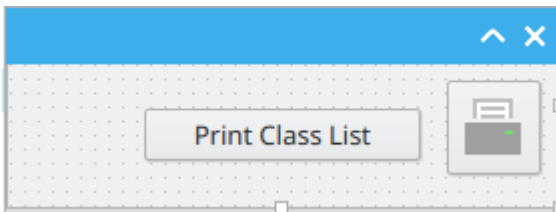
The names are printed down the page. There needs to be a side margin, and here it is set to an inch (960 dots when printing to **PDF**). It is stored in the variable (private property of the form) *SideMargin*. It is the same on the left and the right. The top margin is *TopMargin*.

When you print a name, how far down do you go before printing the next? *LineSpacing* is set to 280. That works out at about 0.3 of an inch. (960 is an inch).

The plan is: Print a name. However long that name is, move along a bit. That is the starting point for a horizontal line. Line as far as the page width less the right side margin. Draw the line. Go down a linespacing. Print the next name. Draw its line. Go down. Print a name. Draw its line, and so on.

Then draw the vertical lines to make the boxes. Start a little to the right of the width of the longest name. Step 330 dots, draw a vertical line, step another 330 dots, draw the next line, and so on.

Don't go past the end point of the horizontal lines. Finally, to make the right hand edge neat, draw a final vertical line. The Printer is called *Prn*. The button is *bPrint*.



```
Private z As New String[]
Private LineSpacing As Integer = 280
Private TopMargin As Integer = 960
Private SideMargin As Integer = 960

Public Sub Prn_Draw()

    Dim s As String
    Dim i As Integer
    Dim NameWidth, HowFarDown, MaxNameWidth, MaxDistanceDown As Float
    Dim MaxWidth As Float = Paint.Width - 2 * SideMargin

    Paint.Font = Font["Linux Libertine,12"]
    Paint.Color(Color.Black)
    Paint.MoveTo(SideMargin, TopMargin) 'start here
    Paint.LineTo(Paint.Width - SideMargin, TopMargin) 'draw to here
    Paint.Stroke 'paint the top line
    For i = 0 To z.Max
        s = z[i]
        NameWidth = Paint.TextExtents(s).Width + 180 'gap at the end about 1/5
    inch
```

```

        MaxNameWidth = Max(MaxNameWidth, NameWidth) 'remember the width of the
longest name
        HowFarDown = TopMargin + (LineSpacing * (i + 1))
        Paint.DrawText(s, SideMargin, HowFarDown)
        Paint.MoveTo(SideMargin + NameWidth, HowFarDown) 'starting position
        Paint.LineTo(Paint.Width - SideMargin, HowFarDown) 'finishing position
        Paint.Stroke 'draw the line
    Next
    MaxDistanceDown = TopMargin + z.Count * LineSpacing 'vertical lines go down
to here
    For i = SideMargin + MaxNameWidth + 100 To Paint.Width - SideMargin Step 330
'step across the page every 1/3 inch
        Paint.MoveTo(i, TopMargin)
        Paint.LineTo(i, MaxDistanceDown)
        Paint.Stroke
    Next
    Paint.MoveTo(Paint.Width - SideMargin, TopMargin)
    Paint.LineTo(Paint.Width - SideMargin, MaxDistanceDown)
    Paint.Stroke 'final line on right

End

Public Sub GetNames()

    Dim FN As String[] = ["Oliver", "Jack", "Harry", "Jacob", "Charlie",
"Thomas", "George", "Oscar", "James", "William", "Amelia", "Olivia", "Isla",
"Emily", "Poppy", "Ava", "Isabella", "Jessica", "Lily", "Sophie"]
    Dim SN As String[] = ["Smith", "Jones", "Williams", "Brown", "Wilson",
"Taylor", "Moreton", "White", "Martin", "Anderson", "Johnson", "Walsh",
"Miller", "Davis", "Burns", "Murphy", "Lee", "Roberts", "Singh", "Evans"]

    FN.Shuffle
    SN.Shuffle
    Dim i, n As Integer
    For i = 1 To 40
        z.Add(FN[n] & " " & SN[n])
        n += 1
        If n > FN.Max Then
            FN.Shuffle
            SN.Shuffle
            n = 0
        Endif
    Next

End

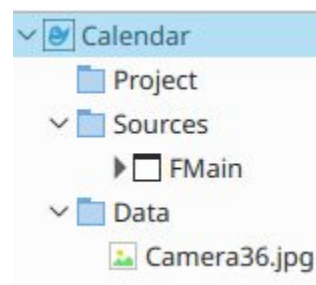
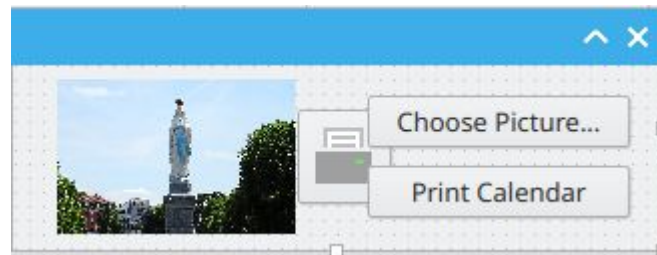
Public Sub bPrint_Click()

    Prn.OutputFile = User.Home & / "Names.pdf" 'I'm printing to a pdf file
    If Prn.Configure() Then Return
    GetNames
    Prn.Print()

End

```

Print a Calendar



The form contains PictureBox1, a printer called Prn, and two buttons called bPicture and bPrint.

This program prints a calendar for the current month. When you look at the page you want to print you will see the “things” that have to be printed in various places. There are three things that call for repetition: the boxes, the numbers in the top left corner of each, and the names of the days of the week.

I gave *PictureBox1* a default picture (that shows as soon as the program is run). First I dragged a photo onto the *Data* folder. Then I set the *Picture* property of the picturebox to it.

If you do not have a picture to begin with, the user needs to click the *Choose Picture...* button before clicking *Print*. The picture is stored in a property called *Pic*. If it is null printing does not proceed.

```
Public Pic As Picture

Public Sub LoadPicture()

    Dim Path As String

    Dialog.Title = "Please Select a picture"
    Dialog.Filter = ["*.jpg", "*.png", "Image Files", "*", "All files"]
    Dialog.Path = User.Home
    If Dialog.OpenFile() Then Return
    Pic = Picture.Load(Dialog.Path)
    PictureBox1.Picture = Pic

End

Public Sub bPicture_Click()
    LoadPicture
    FMain.SetFocus
End

Public Sub bPrint_Click()

    Pic = PictureBox1.Picture 'This line can be deleted if you don't give your
PictureBox a default picture.
    If IsNull(Pic) Then
        Message("Please select a photo first.")
    Else
        Prn.OutputFile = User.Home & "Calendar.pdf"
        If Prn.Configure() Then Return
        Prn.Print
    Endif

End

Public Sub Prn_Draw()

    Dim LeftMargin As Float = 480 'half inch
    Dim TopMargin As Float = 1200 'inch and a bit
    Dim row, col, DayNum, CellNum As Integer
    Dim s As String
    Dim ThisMonth As Integer = Month(Date(Now)) 'the month number of the date
part of the NOW function; 1 to 12
    Dim ThisYear As Integer = Year(Date(Now)) 'current year
    Dim FirstOfMonth As Date = Date(ThisYear, ThisMonth, 1)
```

```

    Dim StartDay As Integer = WeekDay(Date(FirstOfMonth)) 'the weekday of the
first of the month
    Dim TextHeight, TextWidth, GridTop As Float

    GridTop = 7.2 * 960

    'Big Photo
    Paint.DrawPicture(Pic, LeftMargin, TopMargin / 2, Paint.Width - 2 *
LeftMargin, 5 * 960) '5 inch height

    'Month and Year title
    Paint.Font = Font["Copperplate33bc,32"]
    TextHeight = Paint.TextExtents("S").Height 'the height of a character
    s = Choose(ThisMonth, "January", "February", "March", "April", "May",
"June", "July", "August", "September", "October", "November", "December") & "
" & ThisYear
    Paint.DrawText(s, 0, GridTop - 1000, Paint.Width, TextHeight, Align.Center)
'inch above grid top

    'Grid
    Dim Side As Float = (Paint.Width - 2 * LeftMargin) / 7 'one-seventh of the
width between margins
    For row = 0 To 4
        For col = 0 To 6
            Paint.DrawRect(LeftMargin + Side * Col, GridTop + Side * Row, Side,
Side, Color.Black) 'each square
        Next
    Next

    'Days of the Week headings
    Paint.Font = Font["Apple Chancery,12"]
    TextHeight = Paint.TextExtents("S").Height 'the height of a character
    For col = 0 To 7
        s = Choose(col + 1, "Sunday", "Monday", "Tuesday", "Wednesday",
"Thursday", "Friday", "Saturday")
        Paint.DrawText(s, LeftMargin + Side * Col, GridTop - TextHeight - 96,
Side, TextHeight, Align.Center)
    Next

    'Dates
    Dim DaysInMonth As Integer
    If ThisYear Mod 4 = 0 And ThisMonth = 2 Then DaysInMonth = 29 Else
DaysInMonth = Choose(ThisMonth, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30,
31)
    Paint.Font = Font["Linux Libertine,20"]
    TextHeight = Paint.TextExtents("1").Height 'the height of a digit
    For row = 0 To 4
        For col = 0 To 6
            CellNum = 7 * row + col
            If CellNum >= StartDay Then
                DayNum += 1
                If DayNum > DaysInMonth Then Return 'Don't go to 35 days in the month!
                s = If(Col = 0, "<font color=#DD0000>" & DayNum & "</Font>", "<font
color=#000000>" & DayNum & "</Font>")
                Paint.DrawRichText(s, LeftMargin + Side * Col + 96, GridTop + Side *
Row + TextHeight + 96)
            Endif
        Next
    Next
    Row = 0

```

```

Col = 0
While DayNum < DaysInMonth 'Put extra dates up in the top left of the grid.
    DayNum += 1
    s = If(Col = 0, "<font color=#DD0000>" & DayNum & "</Font>", "<font
color=#000000>" & DayNum & "</Font>")
    Paint.DrawRichText(s, LeftMargin + Side * Col + 96, GridTop + Side * Row +
    TextHeight + 96)
    Col += 1 'next column
Wend
End

```

Design a Notebook Tray Icon

The idea for this program is simple: whatever text you have on the clipboard, click on a handy little tray icon and it will be saved to an SQLite database. Nothing seems to happen, but the text is saved in a new record.

There is also the means to search for notes already saved. This is done in a window that appears when you middle-click the tray icon. I would have preferred it to appear on right-clicking the icon, but right-clicking can only make a menu appear. That is what right-clicks do: they show contextual menus. Now, middle-clicking is fine if you have a mouse with a middle wheel or button that can be clicked. For those of us who use a laptop's trackpad, middle-clicking is simulated by clicking both left and right buttons simultaneously. On KDE there is an option in System Settings that can disable this, but by default it is enabled.

A strange and curious thing occurs when you middle-click for the first time: nothing. The second middle-click makes the window appear. I do not know why this happens, but when experimenting with the tray icon menu that can appear when you right-click if you set up a menu and handle the right-click event, the same thing happened. It needed two right-clicks for the menu to appear. I have mentally filed it under Gambas, Strange and Curious Things.

Experiment with the main window being hidden when the application starts if you like, or not the *skiptaskbar* property so the minimised main window does not get listed with the other open applications in the panel, but the effects did not appeal at all so they have been left at their defaults.

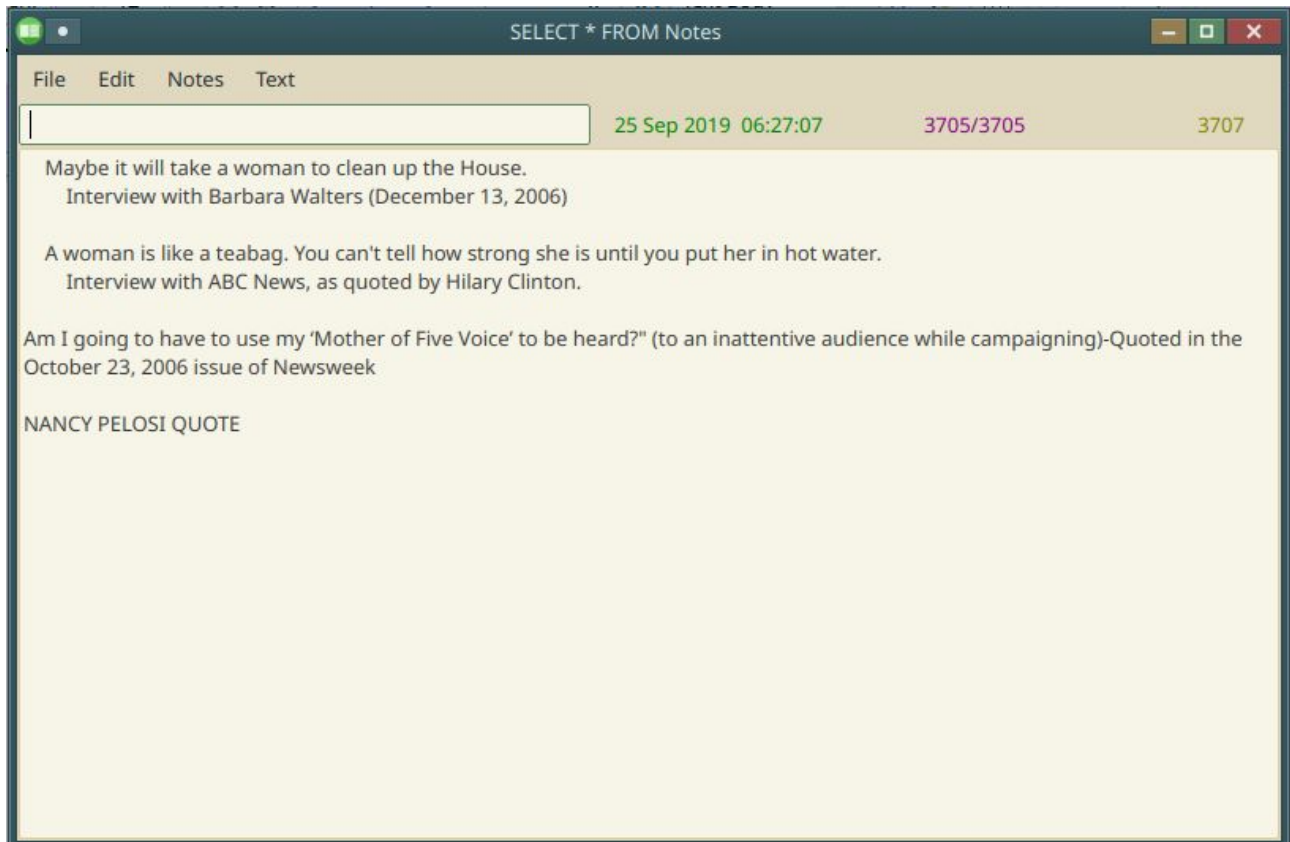
Slightly disappointing was not finding a way to autostart this Gambas program when the computer starts up. I have a widget on the panel to the compiled program, and it is only a simple click to get it going, but the usual KDE autostart process was unable to get it going.

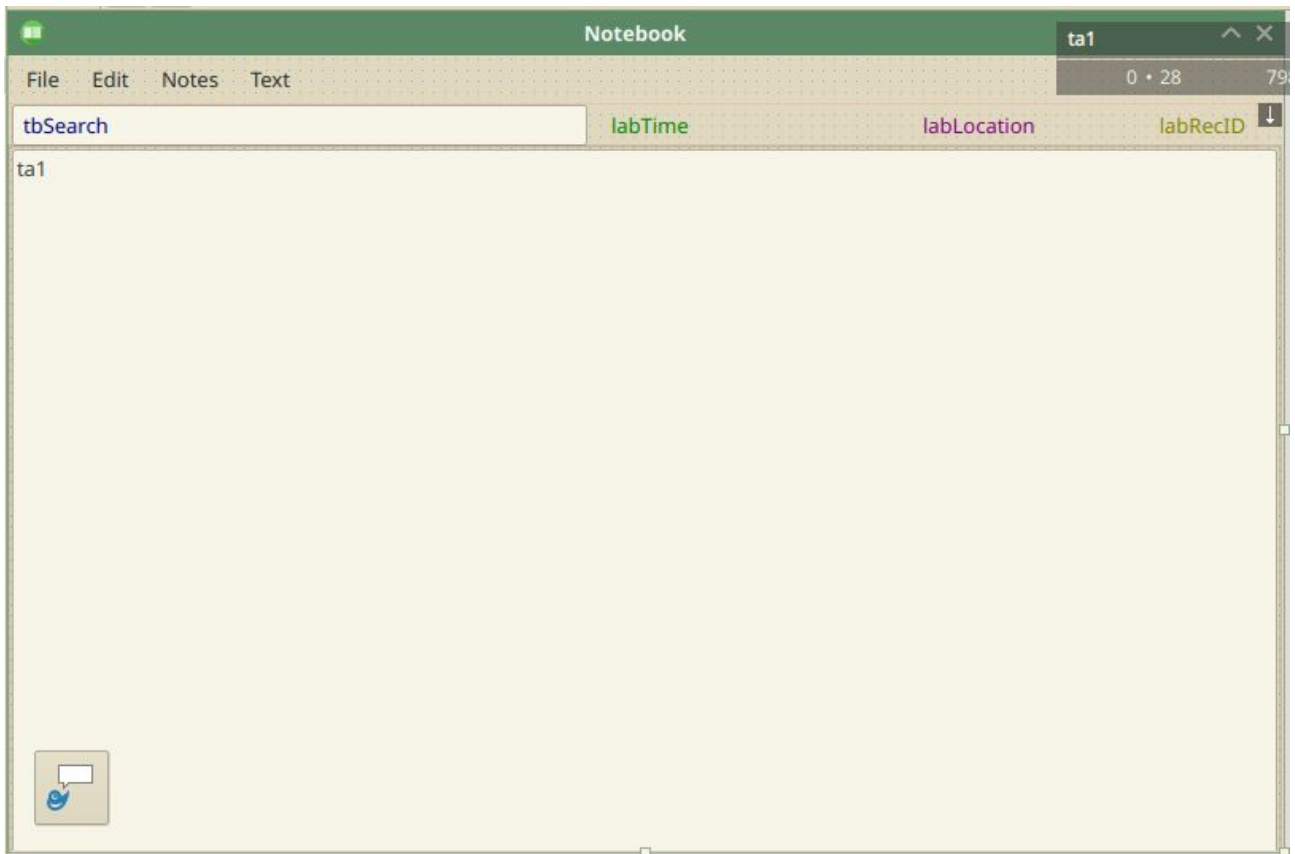
I found a nice notebook icon with a Google image search:



It does not need to be any particular size. It scales itself nicely when the program runs.

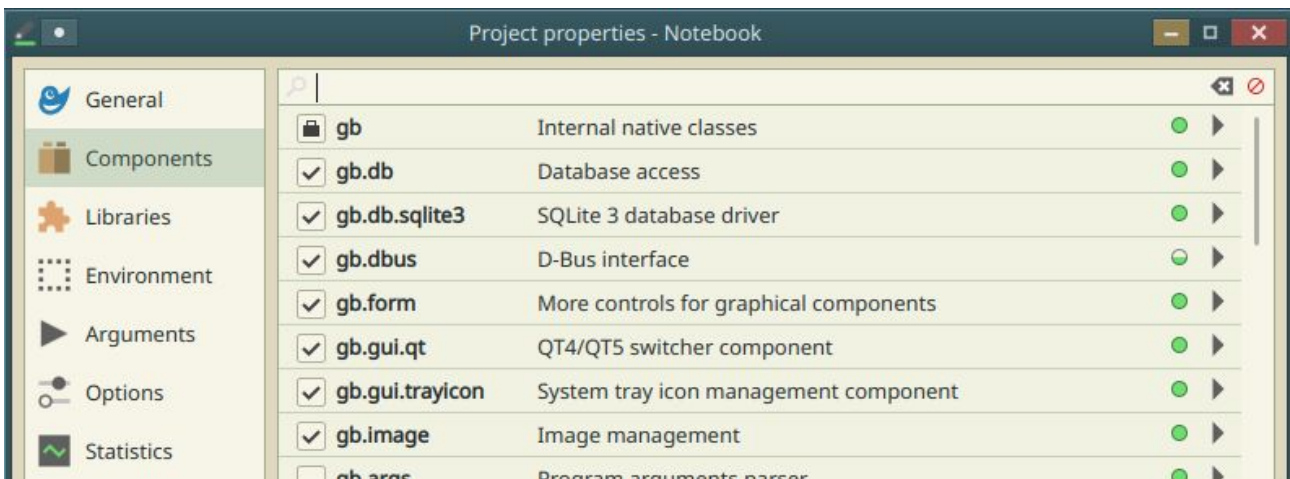
The window has a large text area to show a note, and a textbox at the top left in which to type search text. There are also three labels that display the date/time when the note was saved, the position of the note among all the notes that have been found with that search text in them, and, for interest, the record ID of the note.





The tray icon (bottom left corner) can be placed anywhere. It does not appear in the window. It appears in the system tray (usually at the bottom right corner of the screen, in KDE's default panel).

The tray icon comes in its own component, so check Project > Properties to see that it and the database components are included:



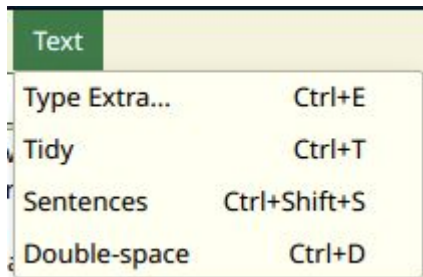
In summary, to use this notebook:

1. Copy any text to the clipboard.
2. Click the tray icon and your text is saved.
3. Click the tray icon with both left and right mouse buttons at the same time to search for a note.

Sometimes it is useful to save the text then bring up the window and add some key words that will help you find the note again. I sometimes add the words JOKE or QUOTE or FAMILY HISTORY. That way, by typing “QUOTE” in the search box, all my quotes appear and I can step through them one at a time. Copying all selected notes would be a useful feature to add.

The SQL select statement appears in the window caption, for interest.

For good measure, a TEXT menu has four entries for adjusting text:



Type Extra... positions the cursor at the end of the text of the visible note, ready for you to type something extra.

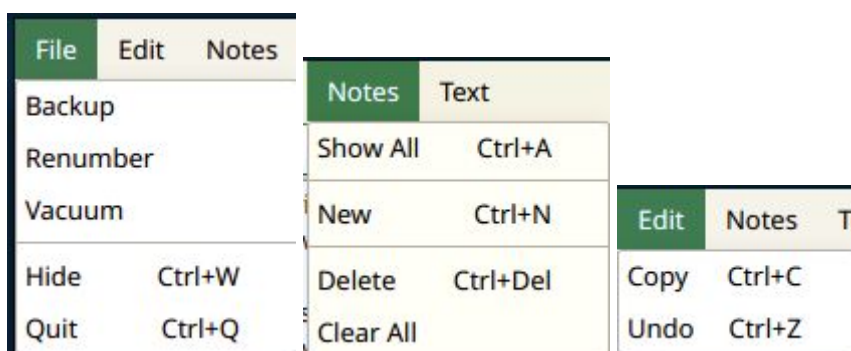
Tidy gets rid of multiple spaces, multiple tabs and multiple blank lines, and removes leading and trailing spaces.

Sentences joins broken sentences. Text copied from emails often has distinct lines.

Double-space separates the paragraphs with a blank line.

These text operations work on the whole of the text if there is no selection, or on the selected text if there is a selection. The shortcuts are there because often I find myself doing the last three in quick succession to get the text looking decent.

The other menus are:



The database is an SQLite file. You might wonder where the OPEN and NEW menu items are. For simplicity, the program creates a new database when it opens if none exists with the name and in the place it expects to find it. It is called *Notebook.sqlite* and it is in the Home directory. It is easy enough to change the code to make it in the Documents directory if you wish.

File > Backup is a useful menuitem that copies the database file to a Backups folder in the Documents folder, date-stamping the file name to show when it was created and to not interfere with earlier backups.

File > Renumber makes the record ID's sequential, with no gaps. It, and the Vacuum item below it, are unnecessary. Vacuum tidies the internal structure of the database file. It uses SQLite's inbuilt vacuum command.

File > Quit closes the application. *File > Hide* makes the window invisible. Clicking the close box on the window also only hides the window; it does not close the application. For this trick, the window's **Persistent** property is set to *true*. The window persists, invisibly, when its close box is clicked.

When you type in the search textbox an SQL Select statement selects all the notes that contain that text. It is a simple search: it does not search for notes containing any of the words, ordered from best to least matching. It searches for exactly the text that is typed. As you type each letter the search is performed. A public property in the *mdb* module, *Public rs As Result*, stores the results of the search. *Notes > Show All* clears the search and finds all the notes. This happens when the program starts: all notes are selected.

Notes > New... lets you type a new note that is saved when you leave the text area.

Notes > Delete deletes the note currently displayed from the database.

Notes > Clear All clears all notes from the database. There is no undo but there is a chance to bail.

Window properties are:

Arrangement = Vertical

Stacking = Above

Width = 800

Height = 500

The Tray Icon (*ti1*) properties that need to be set are:

Tooltip = Click to save clipboard text

Visible = True

Be sure to set it to visible or you will not see the tray icon when the program is run, and there will be no elegant way of quitting the program when it is running after you hide the window that shows initially.

The *Stacking* property ensures that the window remains above other windows. If you click in the window belonging to your web browser, for example, the Notebook window does not get covered by the browser window but remains on top, floating above it. This can be useful for utility type programs.

The last thing to note before getting to the code is the use of the keyboard's arrow keys. UP and DOWN take you to the first and last of the found notes respectively. LEFT and RIGHT step you

back or forwards through the found notes. “Found notes” means those that are found to have the string of text in them that you typed in the textbox, or, if nothing has been typed, all the notes.

A few comments follow the code.

Code relating to the database is collected in a module called *mdb*:

```
' Gambas module file

Public db1 As New Connection
Public rs As Result
Public SQLSel As String

Public Sub CreateDatabase()

    db1.Type = "sqlite"
    db1.host = User.home
    db1.name = ""

    'delete an existing Notebook.sqlite
    If Exist(User.home & "/Notebook.sqlite") Then
        Kill User.home & "/Notebook.sqlite"
    Endif

    'create Notebook.sqlite
    db1.Open
    db1.Databases.Add("Notebook.sqlite")
    db1.Close

End

Public Sub ConnectDatabase()

    db1.Type = "sqlite"
    db1.host = User.home
    db1.name = "Notebook.sqlite"
    db1.Open
    SelectAllNotes
    'Message("Notes file connected:<br>" & db1.Host & "/" & db1.Name & "<br><br>" &
    rs.Count & " records")

End

Public Sub MakeTable()

    Dim hTable As Table

    db1.name = "Notebook.sqlite"
    db1.Open
    hTable = db1.Tables.Add("Notes")
    hTable.Fields.Add("KeyID", db.Integer)
    hTable.Fields.Add("Created", db.Date)
    hTable.Fields.Add("Note", db.String)
    hTable.PrimaryKey = ["KeyID"]
    hTable.Update
    Message("Notes file created:<br>" & db1.Host & "/" & db1.Name)

End

Public Sub SelectAllNotes()
```



```

rs = db1.Exec("SELECT * FROM Notes")
FMain.Caption = "SELECT * FROM Notes"
rs.MoveLast

```

End

Public Sub Message(z As String) As String

```

While InStr(z, "'") > 0 'this avoids a build-up of single apostrophes
    Replace(z, "'", "''")
Wend
Return Replace(z, "'", "''")

```

End

Public Sub AddRecord(s As String, t As Date) As String

```

Dim rs1 As Result
Dim NextID As Integer

```

```

If rs.Max = -1 Then NextID = 1 Else NextID = db1.Exec("SELECT Max(KeyID) AS
TheMax FROM Notes")!TheMax + 1

```

```

db1.Begin
rs1 = db1.Create("Notes")
rs1!KeyID = NextID
rs1!Created = t 'time
rs1!Note = Message(s)
rs1.Update
db1.Commit
SelectAllNotes
Return NextID

```

Catch

```

db1.Rollback
Message.Error(Error.Text)

```

End

Public Sub UpdateRecord(RecNum As Integer, NewText As String)

```

db1.Exec("UPDATE Notes SET Note='" & Message(NewText) & "' WHERE KeyID=" &
RecNum)

```

```

Dim pos As Integer = rs.Index

```

'Refresh the result cursor, so the text in it is updated as well as in the database file. This is tricky.

```

If IsNull(SQLSel) Then rs = db.Exec("SELECT * FROM Notes") Else rs =
db.Exec(SQLSel) 'SQLSel is the last search, set by typing in tbSearch
rs.MoveTo(pos) 'Ooooh yes! It did it.

```

Catch

```

Message.Error("<b>Update error.</b><br><br>" & Error.Text)

```

End

Public Sub MoveRecord(KeyCode As Integer) As Boolean

```

If rs.Count = 0 Then Return False
Select Case KeyCode

```

```

    Case Key.Left
        If rs.Index > 0 Then rs.MovePrevious Else rs.MoveLast
        Return True
    Case Key.Right
        If rs.Index < rs.Max Then rs.MoveNext Else rs.MoveFirst
        Return True
    Case Key.Up
        rs.MoveFirst
        Return True
    Case Key.Down
        rs.MoveLast
        Return True
End Select
Return False

End

Public Sub ClearAll()

    If Message.Warning("Delete all notes? This cannot be undone.", "Ok",
"Cancel") = 1 Then
        db1.Exec("DELETE FROM Notes")
        SelectAllNotes
    Endif

End

Public Sub DeleteRecord(RecNum As Integer)

    db1.Exec("DELETE FROM Notes WHERE KeyID='" & RecNum & "'")
    SelectAllNotes

End

Public Sub SearchFor(s As String)

    SQLSel = "SELECT * FROM Notes WHERE Note LIKE '%" & Message(s) & "%'"

    If IsNull(s) Then
        SelectAllNotes
        SQLSel = ""
    Else
        FMain.Caption = SQLSel
        rs = db1.Exec(SQLSel)
    Endif

End

Public Sub Renumber()

    Dim res As Result = db.Exec("SELECT * FROM Notes ORDER BY KeyID")
    Dim i As Integer = 1
    Dim x As Integer

    Application.Busy += 1
    While res.Available
        x = res!KeyID
        db.Exec("UPDATE Notes SET KeyID=" & i & " WHERE KeyID=" & x)
        i += 1
        res.MoveNext
    End While
End Sub

```

```

Wend
SelectAllNotes
Application.Busy -= 1

End

Public Sub Vacuum() As String

    Dim fSize1, fSize2 As Float

    fSize1 = Stat(db1.Host & db1.Name).Size / 1000 'kB
    db1.Exec("Vacuum")
    fSize2 = Stat(db1.Host & db1.Name).Size / 1000 'kB
    Dim Units As String = "kB"
    If fSize1 > 1000 Then 'megabyte range
        fSize1 /= 1000
        fSize2 /= 1000
        Units = "MB"
    Endif
    Return Format(fSize1, "#.0") & Units & " -> " & Format(fSize2, "#.0") &
Units & " (" & Format(fSize1 - fSize2, "#.00") & Units & ")"

End

Public Sub Backup() As String

    If Not Exist(User.Home & "Documents/Backups/") Then Mkdir User.Home &
"Documents/Backups"
    Dim fn As String = "Notebook " & Format(Now, "yyyy-mm-dd hh-nn")
    Dim source As String = db1.Host & db1.Name
    Dim dest As String = User.Home & "Documents/Backups/" & fn
    Try Copy source To dest
    If Error Then Return "Couldn't save -> " & Error.Text Else Return "Saved ->
/Documents/Backups/" & fn

End

```

The main form's code, the code for the FMain class:

```

' Gambas class file

Public OriginalText As String

Public Sub ti1_Click()

    Dim TimeAdded As String

    If Clipboard.Type = Clipboard.Text Then TimeAdded =
mdb.AddRecord(Clipboard.Paste("text/plain"), Now())

End

Public Sub Form_Open()

    If Not Exist(User.Home & "Notebook.sqlite") Then 'create notebook data file
        mdb.CreateDatabase
        mdb.MakeTable
        mdb.SelectAllNotes
    Else

```

```

        mdb.ConnectDatabase
        ShowRecord
    Endif

```

End

Public Sub MenuQuit_Click()

```

    mdb.db1.Close
    ti1.Delete
    Quit

```

End

Public Sub Form_KeyPress()

```

    If mdb.MoveRecord(Key.Code) Then
        ShowRecord
        Stop Event
    Endif

```

End

Public Sub ShowRecord()

```

    If mdb.rs.count = 0 Then
        ClearFields
        Return
    Endif
    ta1.Text = Replace(mdb.rs!Note, "'", "'")
    Dim d As Date = mdb.rs!Created
    labTime.text = Format(d, gb.MediumDate) & " " & Format(d, gb.LongTime)
    labRecID.text = mdb.rs!KeyID
    labLocation.text = Str(mdb.rs.Index + 1) & "/" & mdb.rs.Count
    OriginalText = ta1.Text

```

End

Public Sub MenuClear_Click()

```

    mdb.ClearAll
    ClearFields
    labTime.Text = "No records"

```

End

Public Sub MenuCopy_Click()

```

    Clipboard.Copy(ta1.Text)

```

End

Public Sub MenuDeleteNote_Click()

```

    Dim RecNum As Integer = Val(labRecID.Text)

    mdb.DeleteRecord(RecNum) 'after which all records selected; now to
    relocate...
    Dim res As Result = db.Exec("SELECT * FROM Notes WHERE KeyID<" & RecNum)
    res.MoveLast

```

```
Dim i As Integer = res.Index
mdb.rs.MoveTo(i)
ShowRecord
```

End

Public Sub ClearFields()

```
ta1.Text = ""
labRecID.Text = ""
labTime.Text = ""
labLocation.Text = ""
```

End

Public Sub MenuNewNote_Click()

```
ClearFields
ta1.SetFocus
```

End

Public Sub ta1_GotFocus()

```
OriginalText = ta1.Text
```

End

Public Sub ta1_LostFocus()

```
If ta1.Text = OriginalText Then Return 'no change
SaveOrUpdate
```

End

Public Sub tbSearch_Change()

```
mdb.SearchFor(tbSearch.Text)
ShowRecord
```

Catch

```
Message.Error(Error.Text)
```

End

Public Sub ta1_KeyPress()

```
If Key.Code = Key.Esc Then Me.SetFocus 'clear focus from textarea; this
triggers a record update
```

End

Public Sub tbSearch_KeyPress()

```
If Key.Code = Key.Esc Then Me.SetFocus
```

End

Public Sub MenuShowAll_Click()

```
mdb.SelectAllNotes  
ShowRecord
```

End

```
Public Sub KeepReplacing(InThis As String, LookFor As String, Becomes As String) As String
```

```
    Dim z As String = InThis  
  
    While InStr(z, LookFor) > 0  
        z = Replace(z, LookFor, Becomes)  
    Wend  
    Return z
```

End

```
Public Sub SaveOrUpdate()
```

```
    If IsNull(labRecID.Text) Then 'new record  
        If IsNull(ta1.Text) Then Return  
        Dim d As Date = Now()  
        labRecID.Text = mdb.AddRecord(ta1.Text, d)  
        labTime.text = Format(d, gb.MediumDate) & " " & Format(d, gb.LongTime)  
    Else 'update  
        If IsNull(ta1.Text) Then  
            mdb.DeleteRecord(Val(labRecID.Text))  
            ClearFields 'maybe leave everything empty?  
        Else  
            mdb.UpdateRecord(Val(labRecID.Text), ta1.Text)  
        Endif  
    Endif
```

End

```
Public Sub MenuTidy_Click()
```

```
    OriginalText = ta1.Text  
    If IsNull(ta1.Text) Then Return  
    Dim z As String = If(ta1.Selection.Length = 0, Trim(ta1.Text),  
Trim(ta1.Selection.Text))  
    z = KeepReplacing(z, gb.NewLine, "|")  
    z = KeepReplacing(z, gb.Tab & gb.Tab, gb.Tab)  
    z = KeepReplacing(z, " ", "|")  
    z = KeepReplacing(z, "| ", "|")  
    z = KeepReplacing(z, "|" & gb.tab, "|")  
    z = KeepReplacing(z, "||", "|")  
    z = KeepReplacing(z, "|", gb.NewLine)  
    If ta1.Selection.Length = 0 Then ta1.Text = z Else ta1.Selection.Text = z  
    SaveOrUpdate
```

End

```
Public Sub MenuSentences_Click()
```

```
    OriginalText = ta1.Text  
    If IsNull(ta1.Text) Then Return  
    Dim z As String = If(ta1.Selection.Length = 0, Trim(ta1.Text),  
Trim(ta1.Selection.Text))  
    z = KeepReplacing(z, gb.NewLine, "~")
```

```

z = KeepReplacing(z, "~ ", "~")
z = KeepReplacing(z, ".~", ".|")
z = KeepReplacing(z, "~", " ")
z = KeepReplacing(z, " ", " ")
z = KeepReplacing(z, "|", "." & gb.NewLine)
If ta1.Selection.Length = 0 Then ta1.Text = z Else ta1.Selection.Text = z
SaveOrUpdate

```

End

Public Sub MenuUndo_Click()

```

Dim z As String = ta1.Text

ta1.Text = OriginalText
OriginalText = z
SaveOrUpdate

```

End

Public Sub MenuRenumber_Click()

```

mdb.Renumber
ShowRecord

```

End

Public Sub MenuVacuum_Click()

```

Me.Caption = "File size -> " & mdb.Vacuum()

```

End

Public Sub MenuBackup_Click()

```

Me.Caption = mdb.Backup()

```

End

Public Sub MenuTypeExtra_Click()

```

ta1.SetFocus
ta1.Text &= gb.NewLine & gb.NewLine
ta1.Select(ta1.Text.Len)

```

End

Public Sub MenuDoubleSpace_Click()

```

OriginalText = ta1.Text
If IsNull(ta1.Text) Then Return
Dim z As String = If(ta1.Selection.Length = 0, Trim(ta1.Text),
Trim(ta1.Selection.Text))
z = KeepReplacing(z, gb.NewLine, "|")
z = KeepReplacing(z, "||", "|")
z = KeepReplacing(z, "|", gb.NewLine & gb.NewLine)
If ta1.Selection.Length = 0 Then ta1.Text = z Else ta1.Selection.Text = z
SaveOrUpdate

```

End

```

Public Sub ti1_MiddleClick()

    Me.Show
    Me.Activate
    tbSearch.Text = ""
    mdb.SelectAllNotes
    ShowRecord

End

Public Sub MenuHide_Click()

    Me.Hide

End

```

The ***Massage(string)*** function is necessary for handling the saving of text that has single apostrophes in it. SQL statements use single apostrophes to surround strings. A single apostrophe in the string will terminate the string and what follows will be a syntax error as it will be incomprehensible. To include an apostrophe it has to be doubled. For example, to save the string *Fred's house* it has to first be converted ("massaged") to *Fred''s house*.

The ***KeepReplacing(InThis, LookFor, ReplaceWithThis)*** function performs replacements until the *LookFor* string is no longer present. For example, if you wanted to remove multiple x's from *abcxxxxdef* and just have one single x you cannot just use *Replace("abcxxxxdef", "xx", "x")*, for this would produce *abcxxdef*. The first double-x becomes a single x, and the second double-x becomes a single x. You still have a double-x. You have to keep replacing until there are no more double-x's.

That's all, folks. May I finish where I began, with a word of thanks to Benoît Minisini. This programming environment is a delight to use. With the gratitude of all of us users we sing, glass in hand, "For he's a jolly good fellow, and so say all of us".

Thanks to <https://tohtml.com/vbasic/> for highlighting the code.

Gerard Buzolic

25 September 2019

Appendix 1

Did You Know? — From Gambas ONE

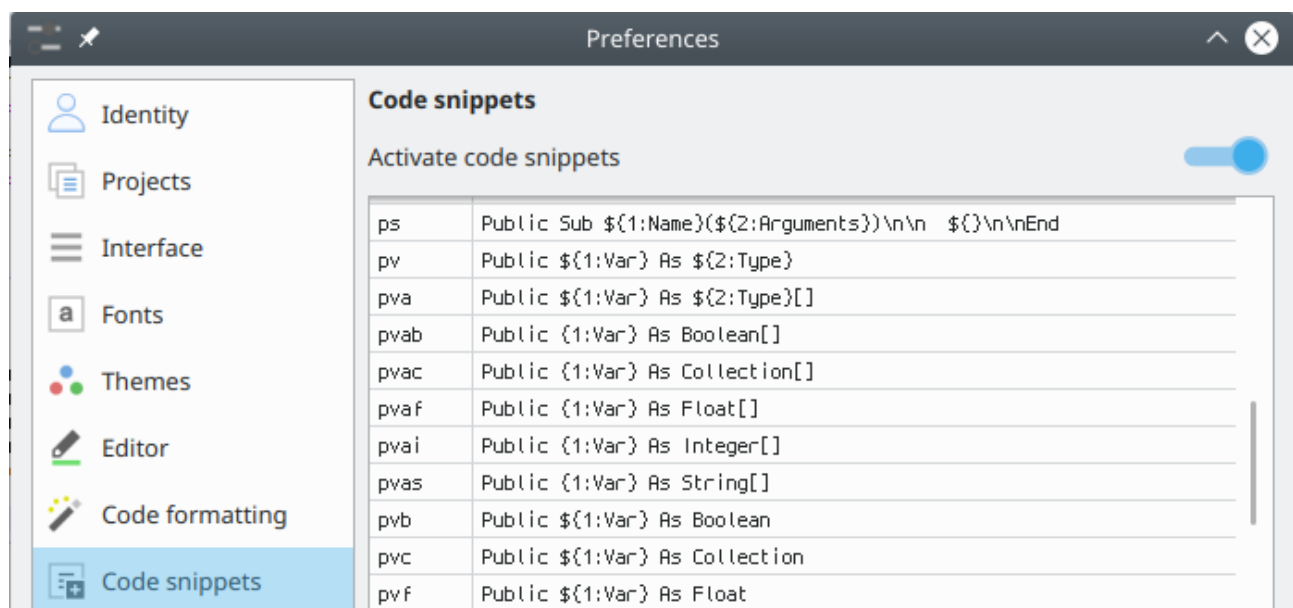
Shortcuts in Writing Code

Thanks to cogier, stevedee and jormmo on Gambas One. <https://forum.Gambas.one/viewtopic.php?f=4&t=489>

```
Public Sub Form_Open()  
.  
End
```

Double-click a blank area of the form to start typing code for the Public Sub Form_Open() event. Double-click a button to start typing code for Public Sub Button_Click(). Otherwise, right-click the object or form > click EVENT... > choose the event you want to write code for.

Expansions



If you want to start writing a new sub, type **ps<tab>** and you will see already typed for you:

```
Public Sub Name(Arguments)
```

```
End
```

v<tab> is changed into `Private $Var As Type`

ds<tab>, **df<tab>** and **di<tab>** are changed into

Dim sVar As String

Dim fVar As Float

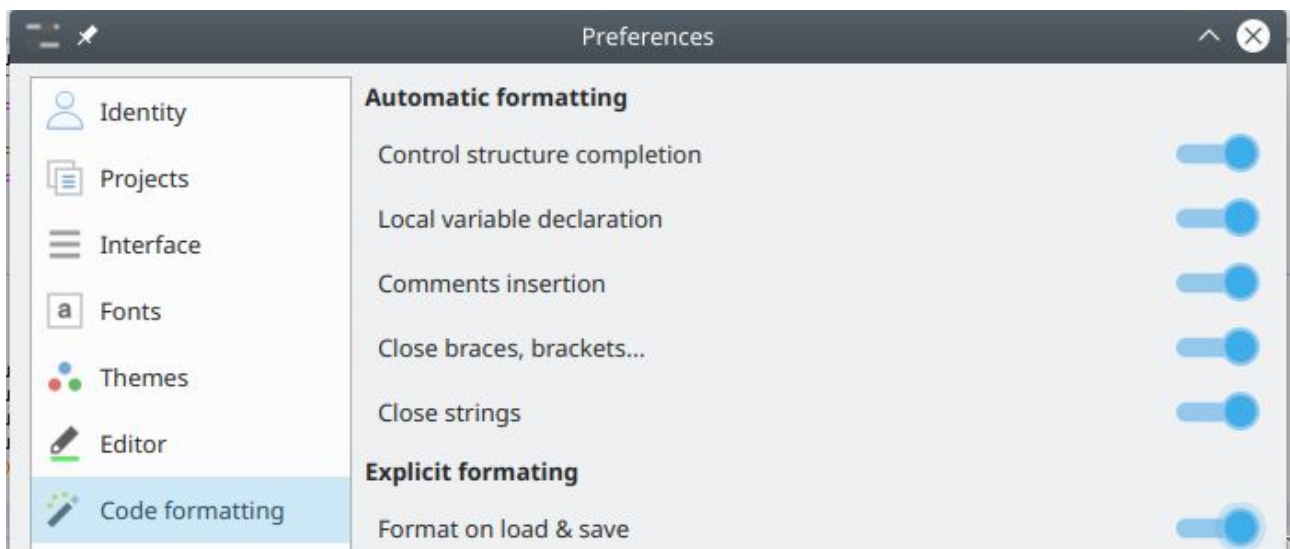
Dim iVar As Integer

Help

If you hold down the CTRL key and click on a Gambas reserved word (e.g. Public, New, For, etc) the relevant help page is displayed. Selecting a keyword and pressing F2 also does it.

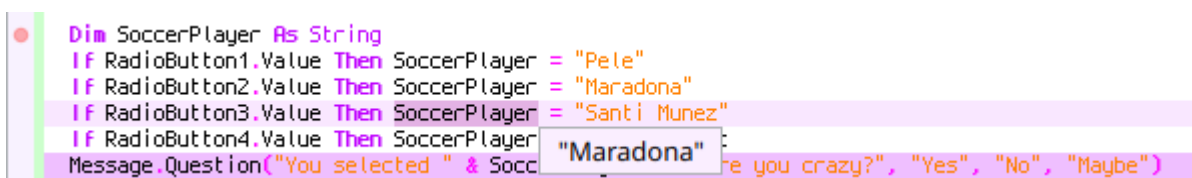
Right-click a tool in the toolbox and help will appear.

Declaring Variables



In preferences *Ctrl-Alt-P*, switch on *Local variable declaration* then in your empty 'Sub' type **iCount = 6** and a **Dim iCount as Integer** automatically appears.

Pausing and Looking at Variables



Breakpoint set on the Dim statement. SoccerPlayer has "Maradona" in it.

If the program is paused (you put in a “breakpoint” in a certain place in the code, and the execution reached this place, or you click the *Pause* button) you can select a variable (drag over it, or double-click it) and you will see its value.

Widen, Shorten and Move Things

Width	126
X	343
Y	224

If you select the width or height properties of a control, the up or down arrows increase or reduce it by 7 pixels.

If you select the X property of a control, up-arrow moves right, down-arrow moves left by 7 pixels. Similarly in the Y property value box, up-arrow moves down and down-arrow moves up (work that one out) by 7 pixels.

If you want a line or selected lines of code to move up, click in the line and press Alt-UpArrow. Alt-DownArrow moves it or them down.

Many Random Numbers

```
Dim siRand1, siRand2, siRand3, siRand4 As Short = Rand(0, 9)
```

This declares four short integer variables and puts a random digit between 0 and 9 into each.

Deleting a Whole Line of Code

Press **Shift-Delete** anywhere on the line.

Commenting and Uncommenting Lines of Code

<pre>Dim SoccerPlayer As String If RadioButton1.Value Then SoccerPlayer = "Pele" If RadioButton2.Value Then SoccerPlayer = "Maradona" If RadioButton3.Value Then SoccerPlayer = "Santi Mune If RadioButton4.Value Then SoccerPlayer = TextBox1.Te</pre>	<pre>' Dim SoccerPlayer As String ' If RadioButton1.Value Then SoccerPlayer = "Pele" ' If RadioButton2.Value Then SoccerPlayer = "Maradona" ' If RadioButton3.Value Then SoccerPlayer = "Santi Mune ' If RadioButton4.Value Then SoccerPlayer = TextBox1.Te</pre>
---	---

Code is active on the left, commented out on the right.

Select the lines > Press **Ctrl-K** to “komment-out” the lines. **Ctrl-U** with lines selected will un-comment them. (Commenting out means making them into comments so they are not executed.)

Non-Case-Sensitive Comparisons

Use a double equals sign to disregard case. For example, `"Hello" == "HELLO"` is true. `"Hello" = "HELLO"` is false.

You can also use a function to compare two strings `String.Comp("Hello", "HELLO", gb.IgnoreCase) = 0`. `String.Comp("Hello", "HELLO") <> 0` is true.

Appendix 2

Functions Reference

String Functions

Character Codes

Asc	Returns the ASCII code of a character	Every character has an ASCII code, eg Asc("A") is 65.
Chr	Returns a character from its ASCII code	Chr(65) is "A" . Chr(32) is the space. Chr(1) is Ctrl-A When checking to see if a key was pressed, use Key["A"] or Key[Key.ShiftKey] instead for the number of a key on the keyboard, rather than the number of a character.

Parts of Strings

Left	The left-hand end of a string	Left("Hello", 4) is "Hell"
Mid	The "middle" part of a string Mid(String, StartingAt, HowLong)	Mid("Gambas", 3, 2) is <i>mb</i> because that is the string that starts at position 3 and is 2 characters long.
Right	The right-hand end of a string	Right("String", 4) is "ring"

Find and Replace

InStr	The position of the second string in the first InStr(String, LookFor) InStr(String, LookFor, StartingAt)	InStr("Gambas is basic", "bas") is 4, because <i>bas</i> starts at position 4. InStr("Gambas is basic", "bas", 6) is 11, because InStr only starts looking for bas from position 6. If the string is not found it returns 0. InStr("Gambas is basic", "bas", -5) is 11. The -5 means only start looking five from the end until the end, i.e. at position 10, the second space. Note that the last letter, c, is not 1 from the end. It is the end. <i>i</i> is 1 from the end.
Replace	The string with every occurrence of something replaced by something else Replace(String, LookFor, WithThis)	Replace("Abracadabra", "a", "*") is Abr*c*d*br* Message(Replace(LetterText, "<name>", FirstName&" "&LastName))
Subst	A string with its markers replaced by	Subst("Tell &1 to arrive at &2 PM", "Mark", "4")

	other strings	is "Tell Mark to arrive at 4 PM"
--	---------------	-------------------------------------

Change Case

UCase	Converts a string to uppercase.	UCase("gerard") is "GERARD"
LCase	The string converted to lower case	LCase("GERARD") is "gerard"

Trimming

Trim	A string with no spaces at either end	Trim(" Hello ") is "Hello"
LTrim	Left Trim The string with spaces removed from the start	LTrim(" Hello") is "Hello"
RTrim	Right Trim The string with spaces removed from the end	RTrim("Hello ") is "Hello"

Building Strings

String	A string made of the same string repeated many times. String(HowMany, WhichString)	String(6, "*") is "*****" String(2, "Gambas") is "GambasGambas"
Space	A string of spaces	Space(8) is eight spaces
Quote	The string in quotation marks The UnQuote function removes them.	Quote("Hello,") & " she said." is "Hello," she said.

Miscellaneous

Len	The length of a string	Len("Gambas is basic") is 15
Comp	Compares two strings Comp(FirstString, SecondString) Comp(FirstString, SecondString, Mode)	If the two strings are the same, it returns 0. If the first string comes after the second in a dictionary, it returns +1 If the first string comes before the second it is -1 If Comp(UserName, "Akiti", gb.IgnoreCase) = 0 Then Message("Welcome Akiti!") else Message("You're not Akiti!")
Split	An array made from a string Split(String, Separator)	Dim z As String[] = Split("John,Mary,Paul,Gemma,Lucille,Hamish", ",") The string is split at the commas.

Numeric Functions

Positives and Negatives

Abs	Absolute value of a number — remove the negative sign if any.	Abs(7) = 7 Abs(-7) = 7
Sgn	Sign (+, 0, -) of a number	Sign(3.14) = +1 Sign(-6) = -1 Sign(0) = 0

To Whole Numbers

Int	Integer part of a number Negative numbers are rounded down; positive numbers abandon the fraction part.	Int(3.2) = 3 Int(3.9) = 3 Int(6) = 6 Int(-7.2) = -8 Int(-7.9) = -8
Round	Round a number to the nearest integer or power of ten e.g. 10^2 is 100, so Round(x,2) is to the nearest 100. Negative powers are how many decimal places	Round(Pi, -2) = 3.14 Print Round(1972, 2) = 2000
Frac	The fractional part of a number The fractional part of a date is the time.	Frac(3.14) = 0.14 Frac(Date(1999, 12, 13, 14, 20, 00)) = 0.597222222015 CDate(Frac(Date(1999, 12, 13, 14, 20, 00))) = 14:20:00
Fix	Integer part of a number—remove any fractional part	Fix(3.14) = 3 Fix(-3.14) = -3
CInt	Convert to an integer Abandon the fraction part completely.	Int(3.2) = 3 Int(3.9) = 3 Int(6) = 6 Int(-7.2) = -7 Int(-7.9) = -7
Floor	Round down to the next lowest whole number	Floor(3.14) = 3 Floor(-3.14) = -4

Ceil	Ceiling: round up to the next highest whole number	Ceil(3.14) = 4 Ceil(-3.14) = 3
------	--	-----------------------------------

Comparing

Max	Whichever of two numbers is greater Also works with dates	Max(6,4) = 6
Min	Whichever of two numbers is lesser Also works with dates	Min(6,4) = 4

Increment and Decrement

INC	Increments a variable; same as $x += 1$ or $x = x+1$ This is an procedure rather than a function (no brackets; it's a verb, not a noun)	INC x
DEC	Decrement a variable; same as $x -= 1$ or $x = x-1$ This is an procedure rather than a function.	DEC x

Character Test Functions

The empty string is **NULL**. IsNull("") is *True*. IsNull("M") is *False*.

IsAscii	Tests if a string contains only ASCII characters.
IsBlank	Tests if a string contains only blank characters.
IsDigit	Tests if a string contains only digits.
IsHexa	Tests if a string contains only hexadecimal digits.
IsLCase	Tests if a string contains only lowercase letters.
IsLetter	Tests if a string contains only letters.
IsPunct	Tests if a string contains only printable non-alphanumeric characters.
IsSpace	Tests if a string contains only space characters.
IsUCase	Tests if a string contains only uppercase letters.

Random Numbers

Randomize	Without it you get the same series of random numbers each time your program is run.	Randomize Just the word by itself.
Rand	A random integer between two numbers.	Rand(1,6) is a random number between 1 and 6 inclusively. This could be a dice throw. Rand(1,2) is either a 1 or a 2.
Rnd	A random floating point number.	Rnd() is between 0 and 1 Rnd(9) is between 0 and 9, but never 9 itself. Rnd(10, 20) is between 10 and 20, but never 20.

Time And Date

A given date and time is stored internally as two integers, the first being the date and the second being the time. The following examples may explain what you can do with dates and times.

Examples:

Now	07/08/2019 19:18:54.929	8 July 2019, almost 8:19 pm The current date and time
Format(Now, "dddd dd/mm/yyyy hh:nn:ss")	Tuesday 09/07/2019 20:45:13	More on Format() later. Use it to present a date and/or time, or indeed any number, in the way you want. And it is now 8:46pm on July 9, by the way.
Date()	07/07/2019 23:00:00	When today started, less one hour for Daylight Saving Time.
Date(1955, 6, 1)	05/31/1955 23:00:00	Someone's birthday, less one hour for Daylight Saving Time. The date is assembled from Year, Month, Day.
Date(2019, 12, 25, 6, 15, 0)	12/25/2019 05:15:00	6:15 am on Christmas Day, 2019. Time for opening Christmas presents. Year, Month, Day, Hours, Minutes, Seconds
DateAdd(Date(1955, 6, 1), gb.Day, 14)	06/14/1955 23:00:00	14 days after 1 June 1955. DS again. You can add other things besides days: gb.Second, gb.Minute, gb.Hour, gb.Day, gb.Week, gb.WeekDay

		(ignore Saturday and Sunday), gb.Month, gb.Quarter, gb.Year
DateDiff(Date(2019, 7, 6), Date(Now), gb.Day)	2	Days between two days ago and now
DateDiff(Date(Now), Date(2018, 12, 25), gb.Day)	195	How many days since Christmas? (It's now 2019. Last Xmas was 2018.) DateDiff(RecentDate, PastDate, Units)
DateDiff(Date(Now), Date(2019, 12, 25), gb.Day)	170	How many days until Christmas? (It's now 2019.)
DateDiff(Date(1955, 6, 1), Now, gb.Second)	2022959392	How many seconds I have been alive
DateDiff(Date(1955, 6, 1), Now, gb.Day)	23413	How many days I have been alive
CLong(DateDiff(Date(1955, 6, 1), Now, gb.Minute)) * 72	2427551928 ... no, wait, it's 2427552144 ... no, wait, it's 2427552216 ...	How many heartbeats since my birth. Had to convert the DateDiff to Long Integer because without it there is an overflow problem: it shows as a negative number. Longs can hold very big numbers.

Parts of Dates and Times

Hour(Now) & " hrs " & Minute(Now) & " mins"	14 hrs 39 mins	It is now 2:39 pm. You can also use these: Day(), Hour(), Minute(), Month() Second(), Time(), Week(), Weekday() Year()
"Rip van Winkle, this is " & Year(Now)	Rip van Winkle, this is 2019	The year part of a date
WeekDay(Now)	2	0 = Sunday, 6 = Saturday 2 means it is Tuesday.
If WeekDay(Now) = gb.Tuesday then label1.text = "It is Tuesday." else label1.text = "It is not Tuesday."	It is Tuesday.	gb.Tuesday is a constant whose value is 2. Using the constant means you need not remember Tuesday is 2. Others are gb.Monday,

		gb.Wednesday, gb.Thursday, gb.Friday, gb.Saturday and gb.Sunday
Time(Now)	14:58:44.654	The time part of a date (Now is the current date and time)
Time(14, 08, 25)	14:08:25	Assemble a time from its parts <i>Time(Hour, Minutes, Seconds)</i>
Time()	15:05:09.515	The time right now
Conversions		
Val("1/6/55")	05/31/1955 23:00:00	Converts the string to a date The 1-hour difference is due to daylight saving time.
Val("1/6/55 2:00")	06/01/1955 01:00:00	2 am on 1 June 1955 The 1-hour difference is due to daylight saving time.
Val("1-6-55 2:00")	nothing	Very sensitive to the format... ignored
Str(2484515)	2484515	If you supply a number, Str() will convert that number to a string.
Dim D As Date = 2484515 label1.text = Str(D)	16/05/2002 01:00:00	The number has changed to a date. Str() converts it into local date format
Dim D As Date = 2484515 label1.text = D	05/16/2002	United States date format
CDate("1-6-55 2:00")	Error	"Wanted date, got string instead" Not only sensitive, but offended!

Appendix 3

Constants

String constants

<http://Gambaswiki.org/wiki/cat/constant>

gb.NewLine	Newline character. This is Chr(10).
gb.Cr	Carriage return character. This is Chr(13). The line terminator on old Macintoshes.
gb.Lf	Linefeed character. This is Chr(10). The line terminator on Linux and new Macs.
gb.CrLf	Carriage return followed by linefeed. The line terminator on Windows and network protocols such as HTTP.
gb.Tab	Tab character. This is Chr(9).

Sort Order

gb.Ascent	Ascending sort (This is default).
gb.Descent	Descending sort.

Alignment

Because these are part of the *Align* class, refer to these as *Align.Center*, *Align.Left* etc.

Bottom	BottomRight	Left	Top	TopRight
BottomLeft	Center	Normal	TopLeft	
BottomNormal	Justify	Right	TopNormal	

Appendix 4

Operators

Arithmetic

<http://Gambaswiki.org/wiki/lang/arithop> by example:

- 6 $x = -x$	Unary minus Changes number to a negative	$5 / 2 = 2.5$	Divide
$3 + 2 = 5$	Add	$3 ^ 2 = 9$	Power of
$7 - 4 = 3$	Subtract	$13 \setminus 2 = 6$ $13 \text{ DIV } 2 = 6$	Integer division
$5 * 2 = 10$	Multiply	$12 \% 7 = 5$ $12 \text{ MOD } 7 = 5$	Remainder after dividing

Boolean

Two things combined, or one thing operated on	Overall Value	Examples
SomethingTrue AND SomethingTrue	True	(1=1) AND (2=2) is TRUE (1<2) AND (5>4) is TRUE Both have to be true to make the whole thing true.
SomethingTrue AND SomethingFalse	False	(1=1) AND (2=3) is FALSE (6>5) AND (4<3) is FALSE
SomethingFalse AND SomethingTrue	False	(5=6) AND (4=4) is FALSE
SomethingFalse AND SomethingFalse	False	(5=6) AND (2=3) is FALSE
NOT SomethingTrue	False	NOT (8=8) is FALSE NOT (1 > -1) is FALSE "Not" means "the opposite of"
NOT SomethingFalse	True	NOT (1=2) is TRUE NOT ("apple" > "banana") is TRUE

		i.e., the reverse of (a comes after b)
SomethingTrue OR SomethingTrue	True	(1=1) OR (2=2) is TRUE Either one being true will make the whole thing true.
SomethingTrue OR SomethingFalse	True	(1=1) OR (2=3) is TRUE
SomethingFalse OR SomethingTrue	True	(7=3) OR (3=3) is TRUE
SomethingFalse OR SomethingFalse	False	(1=1) OR (2=3) is FALSE
SomethingTrue XOR SomethingTrue	False	(1=1) XOR (4=4) is FALSE
SomethingTrue XOR SomethingFalse	True	(1=1) XOR (4=5) is TRUE
SomethingFalse XOR SomethingTrue	True	(1=5) XOR (4=4) is TRUE
SomethingFalse XOR SomethingFalse	False	(1=2) XOR (4=5) is FALSE

AND = both

OR = either

XOR = either, but not both (“exclusive OR”)

String Operators

Joining (Concatenation)

String & String	Concatenates two strings.
String &/ String	Concatenate two strings that contain file names. Add a path separator between the two strings if necessary. An example of a path is /home/gerard/Documents/Gambas/ How to get it: User.Home &/ "Documents" &/ "Gambas/"

Comparison

String = String	Returns if two strings are equal.
String == String	Case-insensitive comparison Returns if two strings are equal, regardless of upper or lower case
String LIKE String	Checks if a string matches a pattern. Is the first string like the second? There are special codes for the pattern string. Refer to the wiki for more

	<p>codes. http://Gambaswiki.org/wiki/lang/like</p> <p>* means any character or string of characters</p> <p>"Gambas" Like "G" means "Does Gambas begin with a G?"</p> <p>? means any single character; [] means either/or the things in brackets:</p> <p>"Gambas" Like "[Aa]" means "Does Gambas have a capital or small A as its second letter?"</p> <p>"Gambas" Like "G[^Aa]" means "Does Gambas <i>not</i> have a capital or small A as its second letter?"</p> <p>Dim Fruit As String = "pear"</p> <p>Label1.text = Fruit Like "{apple,pear,lemon}"</p> <p>shows <i>True</i>, but</p> <p>Label1.text = Fruit Like "{apple, pear, lemon}"</p> <p>shows <i>False</i> because spaces matter.</p>
String MATCH String	<p>Checks if a string matches a PCRE regular expression. Refer to http://Gambaswiki.org/wiki/doc/pcre</p> <p>PCRE means Perl Compatible Regular Expressions</p> <p>Regular expressions are the ultimate in finding things in strings.</p>
String BEGINS String	<p>Checks if a string begins a certain way</p> <p>"Gerard" Begins "G" means "Does Gerard begin with a G?"</p>
String ENDS String	<p>Checks if a string ends a certain way</p> <p>"Benôit" Ends "t" means "Does Benôit end with a t?"</p>
String <> String	Does not equal, or "is not the same as"
String1 < String2	Does string1 come alphabetically before string2?
String1 > String2	Does string1 come alphabetically after string2?
String1 <= String2	Does string1 come alphabetically before, or is it the same as, string2?
String1 >= String2	Does string1 come alphabetically after, or is it the same as, string2?

Appendix 5

Data Types and Conversions

Data Types

One byte is the amount of memory required to store a single character such as the letter “A” or the digit “1”. It is eight bits (1’s or 0’s, like little switches that are on or off, up or down). 8 bits = 1 byte. 4 bits = 1 nibble, but that one is not used much.

<http://Gambaswiki.org/wiki/lang/type>

	Description and Limits	Default value	Size in memory
Boolean	True or false	FALSE	1 byte
Byte	0 to 255	0	1 byte
Short	-32,768 to +32,767	0	2 bytes
Integer	-2,147,483,648 to +2,147,483,647	0	4 bytes
Long	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	0	8 bytes
Single	Single precision	0.0	4 bytes
Float	Floating point or double precision	0.0	8 bytes
Date	Date and time, each stored in an integer.	NULL	8 bytes
String	A variable length string of characters.	NULL	4 bytes on 32 bits systems, 8 bytes on 64 bits systems
Variant	Any datatype.	NULL	12 bytes on 32 bits systems, 16 bytes on 64 bits systems
Object	Anonymous reference to any object.	NULL	4 bytes on 32 bits systems, 8 bytes on 64 bits systems.
Pointer	A memory address.	0	4 bytes on 32 bits systems, 8 bytes on 64 bits systems

Conversions

Some conversions are omitted. Many of these conversions are done automatically as required. For example, these are fine without explicitly having to write the functions:

label1.text = 54.65

Dim d As Date = "09/06/1972 01:45:12"

Dim d As Date = 2484515 gives d the date 05/16/2002

CBool	to a Boolean	<p>An expression is false if it is</p> <ul style="list-style-type: none"> A false boolean A zero number A zero length string A null date A null object <p>CBool(0) is false CBool(1) is true CBool("Gambas") is true CBool("") is false CBool(Null) is false</p> <p>0 → False Anything else → True e.g. 6 is True; -4 is True; 3-(1+2) is False.</p>
CDate	to a Date	<p>CDate("09/06/1972 01:45:12") CDate(2484515)</p>
CFloat or CFlt	to a Float	<p>CFloat("+3.1416") Cfloat("1.0E+3") is 1000</p>
CInt or CInteger	to an Integer	<p>CInt("17") is the number 17 CInt(True) is -1 CInt(Now) is 2490779 CInt(3.2) is 3 CInt(3.9) is 3 CInt(6) is 6 CInt(-7.2) is -7 CInt(-7.9) is -7</p> <p>TRUE → -1 FALSE → 0</p>
CStr or CString	to a String	<p>CStr(-1972) is -1972</p>

		CStr(Now) is 05/16/2002 15:08:31 CStr(Pi) is 3.14159265359
Str	a number or a date into a string	The opposite of Val() Use Format() to have control of what form the number or date takes.
Val	a string into a number or a date	Conversion follows these steps until it finds something it can convert: Look for a date Look for a long Look for an integer Look for a true or false If none, return NULL. IsNull(Val("Gambas")) is True

Appendix 6

Formatting

Formatting Numbers

Example: Label1.text = Format(123456.789, ",#.00") shows as 123,456.79

Format(123456.789) shows as if you used gb.Standard

<http://Gambaswiki.org/wiki/cat/constant>

Formatting Constants

gb.Standard	Uses gb.GeneralNumber for formatting numbers and gb.GeneralDate for formatting dates and times.
gb.GeneralNumber	Writes a number with twelve decimal digits. Uses scientific format if its absolute value is lower than 10^{-4} (0.0001) or greater than 10^7 (1 million).
gb.Fixed	Equivalent to " 0.00 "
gb.Percent	Equivalent to " ###% "
gb.Scientific	Write a number with its exponent (power of ten) and eighteen decimal digits.

Symbols in the Format Strings

Symbols other than these print as they appear. For example, \$ prints as is.

+	Print the sign of the number.	Format(Pi, "+#.####")	+3.142
-	Print the sign of the number only if it is negative.	Format(Pi, "-#.####")	3.142
#	Print a digit only if necessary. One # before the decimal point is all that is needed. After the decimal point, as many #'s as you want decimal places.	Format(123.456789, "#.####")	123.457
0	Always print a digit, padding with a zero if necessary.	Format(24.5, "\$#.00")	\$24.50
.	Print the decimal point	Format(123.456, "#.0")	123.5
,	Separate the thousands	Format(1234567890, "#,")	1,234,567,890

		Format(1234567890, "#")	
%	Multiply the number by 100 and print a percent sign.	Format(0.25, "%%")	25%
E	<p>This is <i>Scientific Notation</i>, which is "Something-times-ten-to-the-power-of-something".</p> <p>"E" means "times ten to the power of..."</p> <p>1.2E+3 means 1.200 with the decimal point moved three places to the right (get bigger x 1000)</p> <p>Negative numbers after the "E" mean move the decimal point to the left.</p>	Format(1234.5, "##E##") Format(0.1234, "##E##")	1.2E+3 1.2E-1
\$	The national currency symbol (according to the country as set on your computer)	Format(-1234.56, "\$,###")	-\$1,234.56
\$\$	The international currency symbol (according to the country as set on your computer)	Format(-1234.56, "\$\$,###")	-AUD 1,234.56
()	Negative numbers represented with brackets, which is what finance people use.	Format(-123.4, "(\$\$,#.00)")	(AUD 123.40)

Formatting Dates

Example: **Format(Now, gb.Standard)** shows as **10/07/2019 21:07:26**

Formatting Constants

gb.GeneralDate	Write a date only if the date and time value has a date part, and write a time only if it has a date part. Writes nothing for a <i>null</i> date or a short time when there is no date, and writes the date and time for all other cases.	Format(Now, gb.GeneralDate) is 10/07/2019 21:17:45
gb.Standard	Uses gb.GeneralNumber for formatting numbers and gb.GeneralDate for formatting dates and times.	10/07/2019 21:20:45
gb.LongDate	Long date format	Wednesday 10 July 2019
gb.MediumDate	Medium date format	10 Jul 2019
gb.ShortDate	Short date format	10/07/2019
gb.LongTime	Long time format	21:22:35

gb.MediumTime	Medium time format	09:23 PM
gb.ShortTime	Short time format	21:23

Format String Symbols

Label1.text = **Format(Now, "dddd dd/mm/yyyy hh:nn:ss")** shows as **Tuesday 09/07/2019 20:45:13**

yy	The year in two digits	h	The hour
yyyy	The year in four digits	hh	The hour in two digits.
m	The month	n	The minutes.
mm	The month in two digits.	nn	The minutes in two digits
mmm	Abbreviated month	s	The seconds
mmmm	Full name of the month	ss	The seconds in two digits
d	The day	:	The time separator
dd	The day in two digits	u	A point and the milliseconds, if non-zero
ddd	Abbreviated weekday	uu	A point and the milliseconds in three digits.
dddd	Full name of the weekday	t	The timezone alphabetic abbreviation
/	The date separator	tt	The timezone in HHMM format
		AM/PM	The AM or PM symbol

Formatting Currency

For the symbols in a format string, see above (numbers).

gb.Currency	Uses the national currency symbol.	Format(14.50, gb.Currency) shows as \$ 14.5
gb.International	Uses the international currency symbol.	Format(14.50, gb.International) shows as AUD 14.5

Operator Precedence

In an expression, which part gets worked out first, then which operations are worked out next?

For example, is $2 + 3 * 4$ equal to 20 (+ first, * second) or is it 14 (* first, then +)? Multiplication precedes addition, so the expression comes down to **14**.

Rules:

Things are worked out before they are compared.

Anything in brackets is worked out first.

The highest priority is changing the sign (-) or reversing true/false with **NOT**.

Strings are joined before paths are assembled. (& is done before &/).

Powers are done before **multiplication or division**, which are done before **addition or subtraction**.

Where there are several operations and they all have the same precedence, the order is left to right but it does not matter because $3 * (4 / 2)$ is the same as $(3 * 4) / 2$

Comparisons are worked out before they are **ANDed**, **ORed** or **XORed** with other comparisons.

Examples:

$4 ^ 2 * 3 ^ 3$ is the same as $(4 ^ 2) * (3 ^ 3)$

$a > 10$ **AND** $a < 20$ is the same as $(a > 10)$ **AND** $(a < 20)$

$4 * 2 + 3 * 3$ is the same as $(4 * 2) + (3 * 3)$

$4 + 2 = 5 + 1$ is the same as $(4 + 2) = (5 + 1)$

Afterword

This book is the work of someone who has only picked up Gambas in the last six months. It began as a Christmas holidays project. The only object-oriented programming experience I have had prior to this is with Xojo, another delightful language in which to program, though, unlike Gambas, commercial. Neither has this book been proof read, so there will be errors for sure. Programming is about errors every step of the way. The sample programs have been tested and they work, but even there something may have crept through that needs fixing.

My interest in computers began in 1974 while a student at a teachers college in Brisbane. At that time I was one of a small group whom a lecturer invited to learn programming after hours from an acoustic coupler. The telephone handset was fastened with rubber clips to a teletype and messages went to and from from the University of Queensland computer, a massive thing with a whopping 20 megabytes of memory that could handle 64 remote users concurrently. After graduation I went on staff in a primary school in north Queensland, and would often bike over to the secondary school to check out the latest version of a computer language called MBASIC written by some young bloke called Bill Gates.

That was in 1976 and the school was the first in the state to have a computer, a DEC-10. There were no floppy disks in those days: it was all stored on paper tape. It was fascinating to type a command (even the word made you feel powerful) on a keyboard and the paper tape writer somewhere else in the room would punch out confetti from a zigzag strip of paper. It could be read by lights shining through the small holes as they were pulled over the light sensors at the dazzling speed of 300 characters per second.

In one school a staff member couldn't see what was difficult about using computers to print student reports. "Can't you just press the PRINT button?" Forty years on I am still writing PRINT buttons. We all pursue the goal of finding the holy grail of programming: one button to rule them all, one button to do everything. Programming has only ever been a hobby, though. That there are people who write languages, database engines and operating systems is awe-inspiring. There are wizards out there. They walk among us. They look like ordinary people.

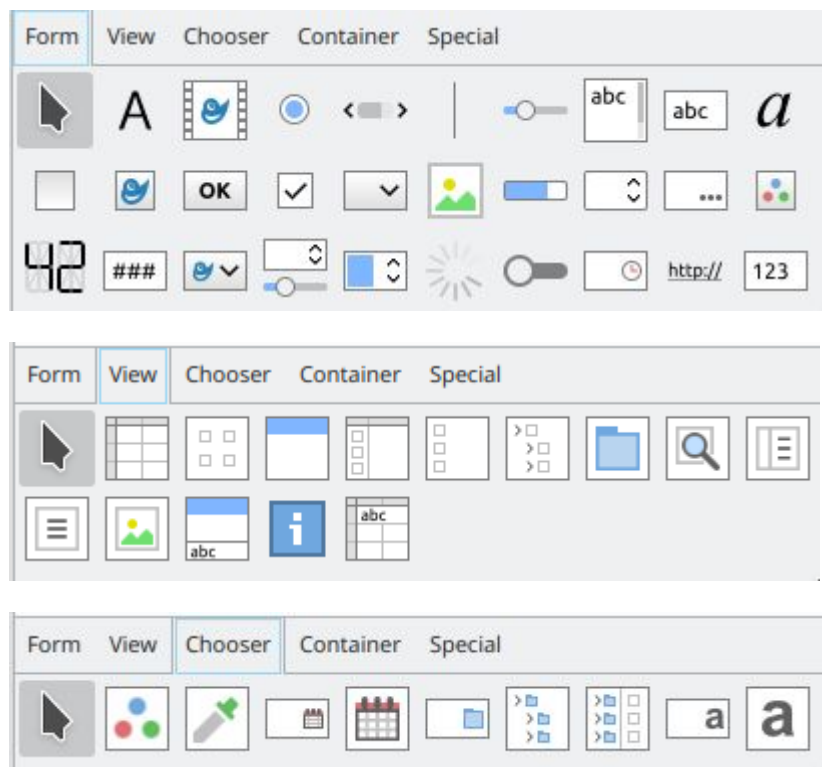
With that background, can you forgive me for using *i* and *j* as names for integer variables? For those who want to, there is <http://Gambaswiki.org/wiki/doc/naming> where my agricultural standards can be refined. For example,

```
Private $iLast As Integer
Private $sLast As String
Private $hEditor As Object
Private $sOldVal As String
Private $bFreeze As Boolean

Public Sub Form_Resize()

    Dim iWidth As Integer
```

Another shortcoming in this book is a lack of introduction to many tools:



Look at them all. All this book provides are buttons, textboxes, labels, grid- and tableviews and forms. The goodies remain in their boxes, unopened, under the Christmas tree. I may find out about them myself someday. They certainly look exciting.

To the person learning Gambas and programming for the first time, good luck.

Thanks again, Benoît and all the writers on forums.

Gerard

Waterford, Ireland, 2019