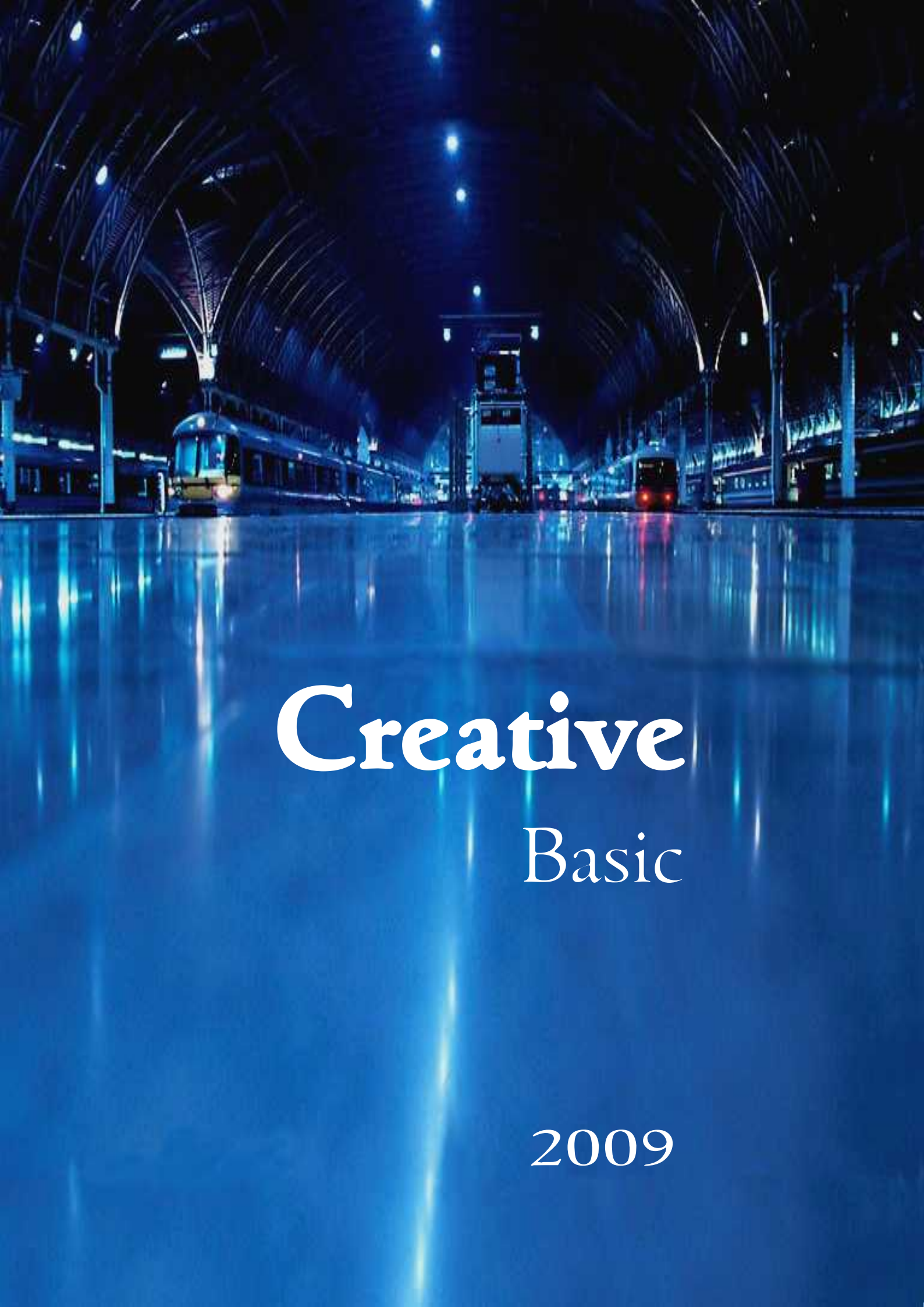
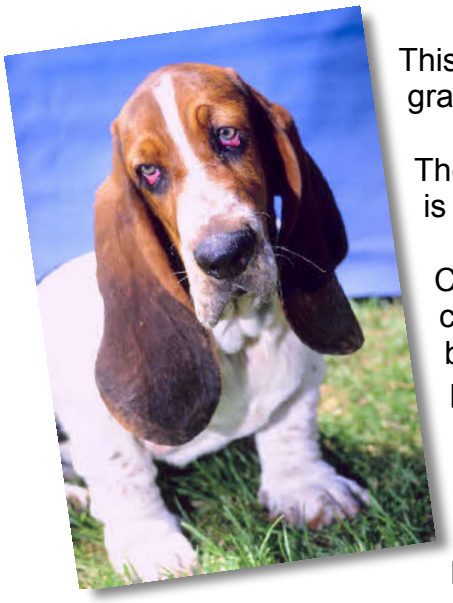


Creative

Basic

2009

Welcome to Creative Basic



This is a friendly version of Basic, developed by master programmer Paul Turley at Ionic Wind software.

There is also a companion version 'Emergence Basic', which is intended for professional software developers.

Creative Basic provides you with all the tools necessary to create serious Windows programs and fun games. It should be particularly suitable for those just learning Windows programming, and for hobby programmers.

Although there are many other Windows programming languages available to use, most are too expensive, or too hard to learn for the average computer user. Creative Basic fills the 'gap' with its easy to use syntax and low price.

System Requirements

No .. you won't need a supercomputer ..

Creative Basic places no significant requirements on any modern machine.

If you wish to use any of the DirectX graphics or Direct3D features of the language, you will need DirectX 7.0 or greater installed.

Creative Basic programs will run on any Windows system from Windows 95 onwards.

Installation and Uninstalling

To install Creative Basic, copy the setup program **cbsetup.exe** to a convenient folder and run it. This will allow you to install the Creative Basic system in a folder of your choice.

When you first run Creative Basic, you will be asked to enter your name and authorisation key.

To un-install Creative Basic (which is recommended when you are installing a new version to replace a previous version), go to **'Start - Settings - Control Panel - Add or Remove Programs'** and choose **'Remove'.**

Welcome to the fascinating world of computer programming.. There's a lot of satisfaction to be had from getting a computer to perform a task for you - whether it be a calculation, building a database of your record collection, or predicting the football pool results.

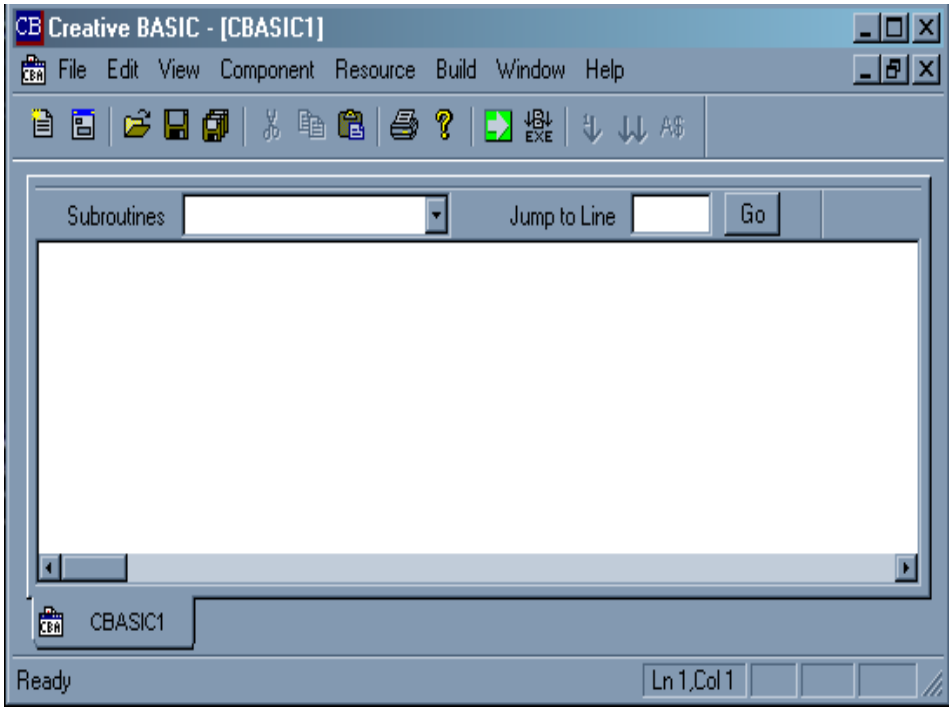
You've made a great choice by selecting Creative Basic as your preferred language. There are many programming languages available, but not all are as simple and friendly to use as Creative Basic.

Each programming language comes with a set of **instructions**. These are the tools you use to tell the computer what you want it to do.

We will be dealing with programming principles rather than examining all the features of Creative Basic, which are fully covered in the Creative Basic User's Guide.

A First Look at Creative Basic

When you first start Creative Basic, this is what you will see -



You have a blank canvas waiting for you to type a masterpiece.

At the cursor, type :

```
Print "Hello"
```

Click the green arrow on the toolbar to run the program. Press any key to close the program.

That was easy, wasn't it? The result is not very impressive, and we can do much better than that.

Take a moment to look at the Help menu. You will find the **User's Guide** an essential reference to programming with Creative Basic.

CBasic comes with many example programs, which you can load using **'File - Open'**. You can run these programs to see what they do, and examine the code to see how they work.

Beginning Programming

I assume you've started CBasic, and have taken a look at the editor screen.

Rather than spend 20 pages at this point discussing how to use all the menu and toolbar facilities, we will leave such things until we need them.

Of course, if you really need to know something specific, look it up in the Help file.

So there you are, the editor screen is open, and we're ready to write a program.

If you've never written a program before that can be a bit daunting. How to get started?

It's not going to achieve much if you just type any old random words. The compiler just won't understand.

You have to type specific and accurate instructions that it will understand.

So what are these instructions? We won't go into the details yet - there's a lot to learn and it would take a long time. Instead, we will investigate CBasic commands as they arise.

The Help menu will come in useful for detailed information. Not even an experienced programmer can remember the details of every instruction.

Open the Help menu, and select **User's Guide - Appendix - Command Reference** .

There are more than 250 commands. Each of them performs a specific task, and many require several other items of information in order to work.

We will become acquainted with them in due course as they are needed.

Look upon the commands as a set of tools to be selected and used when a particular job is to be done.

All programs process 'data' and display some results.

Data can be either numbers or text.

Numbers can be simple integers such as: 5, -25, 0 .. Etc.
Or Decimal values such as: 0.025, 12345.678, 2.5e-12 .. Etc.

Text comprises strings of 'alphanumeric' characters from the ASCII character set.
Examples are: "My name is - Fred Smith.", "$55.23", "#~&^+!" .. etc

Writing a First Program

With all those tools the problem is to know how to proceed?

My advice would be, not to attempt anything too ambitious to begin with.

With any new program, you need to form a clear mental picture of what you are trying to achieve.

Since we want to program in Windows, what better place to start than to create a Window.

We will create a 'skeleton' window - so called because it is just the bare bones of a Windows program. It will be useful to save as the starting point for many other programs.

Before we start, create a new directory in which the new program can be stored – name the directory something like 'CBWork', so that you can recognise it again as the place where useful bits of program are kept.

As a general rule, aim to keep useful programs and code snippets in suitable directories. You can then load them at a later time, and copy and paste useful bits into new programs that you write. It will save you lots of time by not having to re-think how you did useful things.

What we are aiming for is a simple window, with an 'Exit' button with which to close the window.

To start a new program in CBasic, click on **'File - Close'** to close any current program in the editor window.

Click **'File - New - Source File'** to create an empty document. Hopefully, you will find it convenient to copy the program code (see below)

Note that code will be shown in blue coloured boxes

Paste or type the code into the CBasic editor window.

If you would like to keep the program for later reference, you can use the **'File – Save As'** menu to give it a name, and save it to the directory you created previously.

You can now run the program by clicking the green 'Run' arrow.

Before we examine the code, note that we will discuss the instructions 'in context', as we meet them.

The program may look complicated, but we will work through it and reveal it's structure.

It has a pattern, which you will see repeated in all the programs you write. Once you grasp the pattern, you will find it easy to write your own code.

Here's the code for Program 1: Copy and paste it into your editing window.

```
' A Skeleton Window Example ..
' CBasic Users Guide 2009

def w:window
def run, textW, textH:int
def a$:string

autodefine "OFF"

' open the window ..
window w, -700, 0, 700, 450, @SYSMENU, 0, "Skeleton Window", msghandler
setwindowcolor w, rgb(0,100,120)

' define a Button control ..
control w, "B,Exit, (700 - 70)/2, 320, 70, 30, @ctlbtnflat, 1"
setcontrolcolor w, 1, 0, rgb(120,140,220)

' display some text ..
a$ = "Welcome to Creative Basic"
setfont w, "Arial", 30, 700, @sfitalic
frontpen w, rgb(130,130,230)
drawmode w, @transparent
gettextsize w, a$, textW, textH
move w,(700-textW)/2,60
print w, a$


run = 1

waituntil run = 0
closewindow w
end

sub msghandler
select @class
  case @idclosewindow
     run = 0
  case @idcreate
     centerwindow w
  case @idcontrol
     if (@controlid = 1) then run = 0
endselect
return
```

If all has gone well, you should see a nice looking window - and look, you've hardly had to do any work yourself.

So what's significant about this example?

You get a window with a **Coloured background**, an 'Exit' **button** which is coloured and **'Flat'** (not the normal Window's button style which is 3D effect), and some **centred text.**

These effects are easily achieved in CBasic.

CBasic has simplified such techniques for the user, by wrapping difficult aspects of Windows programming into simple statements.

How Does it Work

Now we'll take a look at how it works. Firstly, step well back from it, and just look at the program in the editing screen.

Assuming that you've kept the default editor colours, you will see lots of blue bits, a few bright red words, a few orange words, quite a lot of green (comments), and the remaining items will be black.

The blue words are the in-built CBasic commands, so this gives you an idea how often CBasic tools have been used in this application.

The red words, indicate data type definitions. These occur mostly at the beginning of any program or subroutine.

Orange items are what are known as system variables and constants, used mainly when setting up controls, or checking system messages. These are built into CBasic. (see the User Guide - Appendix for a complete list).

You will usually see quite a lot of green words. If you follow my advice, you will include lots of comments to explain what is going on. (I didn't use too many here to keep it short).

You will find you will often refer to the comments in your program in future, to understand the program code.

The black items are interesting, because these represent your own variables and data used for the application.

Now we will look a little closer to establish the significant parts of the program. Don't look at the detail - just try to see the overall structure.

Comments

The first two lines are a small block of comments. Every program should have some comments at the beginning to describe the program, the author and the creation date.

Variable Type Definitions

Then follows a small block of variable type definitions.

You will hardly ever write all of these in one go. As you are writing the program, you will often think - 'Oh, I need a new variable for this part of the program.'

Then you will think of a name for it, and what type of data it will hold. Then you add the name to an appropriate definition list at the beginning of the program.

Goodbye Autodefine ..

Then follows the Autodefine "OFF" statement, which makes sure you don't forget to specify what type of data your variables will hold.

Open a Window

Next we find the Window command to open the window.

This is a key instruction, because without the window we could do nothing more.

Position Controls

Once the window has been requested, we usually add the controls that we need for this window.

In this small application there is just one window and one button - the 'Exit' button.

Various bits of Initialising Code

Next come any sections of program, implementing things you want to do with the window, like colouring the window with a gradient fill, or maybe some nice graphics.

In this simple program we just write some text to the screen.

Wait .. Wait ..

Then comes an important bit. The 'run = 1' and the 'waituntil run = 0' statements.

Once the condition 'run = 0' occurs, the program will close the window, and END.

However, we'll leave discussing what is happening here until we've seen the rest of the program structure.

Process the Messages

The other subroutine 'msghandler' is the most important part of every Windows program.

It won't always be named 'msghandler' though - you can call it whatever you like - it's what it does that's important.

In the next section we will look at the key role the 'msghandler' routine plays in every Windows program.

So there we are - do you see the pattern?

In the main body of the program we have -

- **Comments (describing the program)**
- **Data Definitions**
- Open a **Window**
- Set up **Controls**
- **Initialising** sections
- A '**Wait Until'** statement, followed by
- **Close Window** and
- **END** statements.

Then follows the 'msghandler' subroutine associated with the window.

This is where most of the work to interact with the user is usually done.

Finally, any number of your own **Subroutines** can be placed after the END statement of the main program section ..

Now we move on to look at the role messages play in all Windows programs.

Understanding messages is the key to getting your programs working successfully.

Messages and the Secrets of Windows

You need to concentrate on this bit. If you can see how messages work, you are well on the way to writing any Windows program in CBasic.

When you open a window, Windows begins generating messages such as - 'the window has been created' - 'the window has been re-sized' - 'a control has been clicked' - and so on.

The **Message Handler** subroutine in your program, intercepts relevant messages and executes some code, based on what has occurred. This is how user interaction occurs.

Once it has been set going, the program becomes '**Event Driven**'. It responds to whatever the user does.

Each window and each control will send messages that can be intercepted and acted upon by your message handling subroutine.

This subroutine becomes the heart of your program and will process whatever code, and subroutines you wish based on the user's actions.

If you look again at the subroutine **'msghandler'** in our program, you will see we first process a message (when it occurs) that the window has been closed.

We respond by setting the variable **'run'** equal to '0'.

Now you see the point of the statement '**waituntil run = 0'** in the main part of the program.

The main program has been cycling quietly to itself, just waiting for the condition '**run = 0**' to be true. Then the program drops through to close the window, and END.

When a 'window has been created' message is detected, we respond with a command '**centerwindow w**' to centre the Window on the screen.

In a similar way, we deal with other types of message concerning Timers, a Button being clicked, or a Key being pressed on the keyboard.

There will be many other types of messages which we have chosen to ignore in this simple application.

You will find lots of information about messages associated with windows and controls in the **Help** system.

If you grasp this messaging concept, you will be able to tackle any CBasic program.

There are several types (or styles) of window.

Which you choose will depend on your application and your preference for the user interface you wish to present.

Since one or more windows will be an important choice in all programs you write, we will take a look at some of the options.