GamesBasic

Alpha Version 1.0

Commands And Functions

'This is a technical guide, expect no user guide :) ' – D. Hanna
'If you find any mistakes, please let us know' – J.Hunter [Author]

Table Of Contents

Introduction

Before you begin..

GamesBasic is still in Alpha – This document is also in Alpha, it does not fully explain all commands, errors. Omission >DO< exist within this documentation. If you do not wish to use a alpha program, destroy all copies of this program and documentation.

If you wish to locate a new version of this documentation please download the latest version of GamesBasic via www.GamesBasic.com.

Enjoy GamesBasic – I enjoying making it, and hope you enjoy using it!!

James

Copyright and ownership

Overview

GamesBasic is a privately owned and copyrighted computer program / application. Copyright, ownership and titleship belong solely to James Hunter, Lisburn, Northern Ireland. James Hunter also owns the idea on which GamesBasic has been created, i.e. a programming language which allows easy access to computer hardware. No derivative work should be attempted or carried out. ProSoft Technologies currently holds the licence to distribute GamesBasic.

Warranty and Guarantee

GamesBasic and all supplied materials come with no warranty or guarantee of any kind. You have agreed with all legal material shipped / supplied with GamesBasic and understand that you use this software / documentation entirely at your own risk. James Hunter, ProSoft Technologies or its agents cannot be help responsible for any problems, damage, loss, debt, legal problems, etc.

Further reading

Full terms and conditions which apply to GamesBasic are freely available from ProSoft Technologies, please apply in writing to:

Project Management
ProSoft House
78 Armagh Road
Trandragee
Co. Armagh
BT62 2HS.

GamesBasic - Interface

The GamesBasic Interface – IDE.

GamesBasic comes with a powerful IDE which will require a separate user documentation, and when I have the time I will get one done :)

For now you must navigate it on your own, unless some of you amazing users can write a good document!!

GamesBasic – Features and Bugs

GamesBasic Features

GamesBasic has a impressive list of features, listed below are the ones I can remember :)

- RAD (Rapid Applications Development) system
- Support for ANSI basic syntax
- Simple to use interface (IDE)
- Support for multiple screen effects
- Timers
- Keyboard input
- Sprite (basic)
- User defined procedures
- Comprehensive maths and string handling functions
- Support for WAD-Style projects
- Variables have no type – they are covered automatically
- Support for various loops
- Full graphics commands
- Multiple screens
- Date and Time Functions
- Text Grids
- Immediate Mode
- Screen FX (i.e. Fades)
- Continuous development cycle
- ***FREE!***

GamesBasic Bugs

As with all programming language there are bugs, some of these bugs may never affect you (hopefully)

- GamesBasic does not syntax check (validate) any code before its ran – so if you enter code which is wrong then there is a chance GamesBasic may crash or abort running. Hopefully, you should receive a RTE (run-time error) but some commands expect to receive validated commands (for speed).
- Its not fast enough – Yes, GamesBasic is no where near fast enough for even my standards, and there is one simple reason why this is. Its not optimised. Optimising a program (such as GamesBasic) makes it run faster, a lot faster. However it takes a lot of time, and unless you have the application running 100% correctly you can introduce new bugs and delays in releases.

Expressions

An overview of expressions

What is an expression?

An expression is a number of values, linked together by operators to create a bigger (more powerful) value.

A Simple Expression

Take the following example '100 + 100', this contains two value and an operator. The operator in this case is a '+' for addition, and the result would be 200.
GamesBasic fully supports these standard expressions, as well as ones containing brackets for parenthesis, EG. '(100 + (5 * 2) + 5)'

Complex Expressions

It also supports variables, see later on how to declare and use them. A expression containing a variable may look like the following.. (100 + Counter + (5 * 3))
GamesBasic has a number of built in functions (and you can define your own), so its simply a mater of including the function name and any parameters its requires. EG. (4100 + Average(100,100,200) + Counter) – This shows a function (Average) with 3 parameters, a value (4100) and a variables (Counter), all inside one expression.

GamesBasic understands and performs the rules of precedence correctly.

Expression Operators

Only mathematical expressions have been used, such as add (+) or divide (/). GamesBasic supports a large number of operators and this include :

- AND – Logical AND, both values passed to an AND must be either True or False. I.e. The expression 'Finished = CouldBe And True' – Could require CouldBe to be a variable containing the value True or False. Logical AND will return true if both sides of the statement are true.
- OR – Logical OR, uses the same syntax of AND. But returns true as long as one of the values passed is true.
- NOT – Logical NOT, this inverts the contents of the passed value (must be Boolean true / false value). A logical NOT requires a simple expression, which must be enclosed in a set of brackets. I.e. 'Let Finished = Not(StillGoing)'
- < (Less than) – Return true is the value on its left is less than the value on its right (I.e. 2 < 5 = True)
- > (Greater Than) – Returns true is the value on its left is greater than the value on its right (I.e. 2 < 5 = False, 5 > 2 = True)
- <> (Not Equal To) – Returns true is the value on the left isn't the same as the value on its right.
- >= (Greater than or Equal to) – Returns true if the value on the left is greater than or equal to the number of the right (I.e. "5 >= 5" is True, and "10 >= 5" is also true.
- <= (Less than or Equal to) – Returns true is the value on the left is less than or equal to the value on the right.
- = (Equal to) – Returns true if the both values are the same.

Variables

An overview of variables

What is an Variables

A variable is a holding place for information within you program, as the name suggests the value is ‘variable’. GamesBasic supports a wide range of variable types, which are automatically converted between types (such as integer, floating point, string, etc).

Variables

Declaring variables

LET Command

Syntax : LET <Variable Name> = <Expression>

The ***LET*** command allows you to create a variable at run-time, using the expression provided. The Variable name can be up to 255 characters long, contain alphanumeric values (however, the first character MUST be a alphabetic character) and a few special symbols (such as '_' , '$' , '!', '#')

Example (Simple Expression)

```
LET CostPerHour = 5
LET HoursPerDay = 8
LET DailyWage = CostPerHour * HoursPerDay
```

Example (Complex Expression)

```
LET CostPerHour = (CostPerHour * HoursPerDay) / Insurance + (15 * 10)
LET AverageCost = Average(JohnsPay, AlansPay, JamesPay, RuthsPay) ;
```

Note : The last example, calls a Function named 'Average', which takes multiple values and works out the numeric average!

Variable Scope

The scope of a variable is who ‘visible’ a variable is to other parts of a program. Two different types of variable scope exist : Local and Global.
Local variables are those which can only be seen by commands inside a procedure, or from other procedures called from the procedure which declared the variable. When you enter a procedure and create a number of variables these exist locally, and can be used inside the procedure. When you exit the procedure, all local variables declared inside it are destroyed.
Global variables are the opposite, these exist no mater where you are within the program and are usually declared in the main block, outside procedures. However, it is possible to turn local variables into global variables (See Global command). Global variables are never destroyed and exist until the program is terminated.

GLOBAL Command

Syntax : GLOBAL <Variable>,<Variable>,<Variable>,….

The ***Global*** command turns a list of variables (separated by semicolons) from local into global variables. If you variables does not exist locally, GamesBasic will create the variable as a zero valued integer.
Globally declaring a variable makes it visible to all sections of your code, whether inside procedures, functions or in the main code block.
Note : GamesBasic will not destroy the contents of any variable converted to a global variable.

LOCAL Command

Syntax : LOCAL <Variable>,<Variable>,<Variable>,….

The ***Local*** command provides a way of declaring variables as local to a procedure or function. Once you have declared a variable as Local it will exist only while execution is within that procedure, or any other procedures it calls. Once a *ENDPROC* or *EXITPROC* is executed, all local variables are destroyed and there contents lost.
Global variables can be turned into local variables, am will be destroyed at the end of a procedures / functions execution.
Note : GamesBasic will not destroy the contents of any variable converted to a local variable.

Control Structures

Overview of control structures

What is a controls structure

A control structure is a command or number of commands, such affect the running of program. Listed below are the most common types of control structures:

- Loops, these structures allow a number of commands to be repeated until a condition is meet.
- Batch Statements, these structures allow the movement around a program depending on conditions.
- Exceptions, these structures all the processing of an error procedure when something goes wrong with your program.

Control Structures - Loops

Do..Loop Structure

Do Command

Syntax : DO

The ***DO*** command is used to show the starting point of an unconditional loop. This sort of loop will repeat forever, never stopping due to a condition. However using the *Break Loop* command, you can break the loop and continue after the ending *Loop* command.
Note : Every ***DO*** must be accompanied by a terminating *Loop* command, the system will return an error if you have too many/little ***DO***'s or too many/little *Loop's*.

Loop Command

Syntax : LOOP

The ***LOOP*** command, terminates a loop started with a *DO* command, and is the last instruction executed before the loop restarts at the command after the matching *DO* command. See *Break Loop* command for more information.

Repeat..Until Structure

Repeat Command

Syntax : Repeat

The ***Repeat*** command is used to initiate a Repeat..Until loop, this type of loop is used to perform a number of loops until a desired result is achieved.

Until Command

Syntax : Until <Condition>

The ***Until*** command is used to end a Repeat..Until loop by stating the point at which the loop will stop. This <condition> will be tested at the end of each run down the code within the Repeat..Until structure, and until it is TRUE to loop will continue to run.

Example

```
Let Counter = 10
Repeat
    Let Counter = Counter + 1
    Print "The Counter Is " + Counter
Until Counter = 10
```

The above sample code will run until the counter become 10.

While..Wend Structure

While Command

Syntax : While <Condition>

The ***While*** command initiates the While..Wend loop and also state the point at which the loop will complete. While the condition supplied has NOT been met, i.e. it is not true the loop will continue to operate.

Wend Command

Syntax : Wend

The ***Wend*** command is used to show the ending point of a While..Wend loop. When a Wend command is executed, GamesBasic will jump back to its *While* command and test the condition.

Example

```
Let Counter = 10
While Counter < 10
    Let Counter = Counter + 1
    Print “The Counter Is “ + Counter
Wend
```

The above sample code will run until the counter become 10 – Note that “Counter < 10” is used, because the While..Wend loop executes until the condition becomes True.

Control Structures – Loops Special Commands

BREAK LOOP Command

Syntax : BREAK LOOP

The break command has many features, however its importance in the stopping of loops is important. Using ***Break Loop*** you can stop the 'current' loop you are inside, and continue execution after the terminating command (E.g. For a *Do..Loop,* execution continues after the *Loop* Command.).

Control Structures - Jumps

Goto Structure

GOTO Command

Syntax : GOTO <Label>

The ***Goto*** command is used to jump to a label declared elsewhere in your code. This jump is one-way and there is no command to directly return you to the command after the ***Goto*** command (See *Gosub..Return* structure).
The label is declared by placing the label name as the first word on a line, followed by a single colon. For example "Label:" or "JumpHere:" GamesBasic supports both forward and backwards jumps to labels, but will search forwards jumps first (so for extra speed keep ***Goto*** labels in front of ***Goto*** commands).

You can only jump to a label in the same 'block'. A block is a procedure or the main source code block. This means you cannot jump into a procedure, out of a procedure, into loops, out of loops or into control structures (I.e. *If..then..else* commands). GamesBasic will refuse to make such jumps, you can however jump over these commands to a label. Just a long as the label is not inside any of these.

Gosub..Return Structure

GOSUB Command

Syntax : GOSUB <Label>

The ***Gosub*** command is used to jump to a subroutine identified using a Label. ***Gosub*** is similar to *Goto* in the fact it performs a straight jump to the label. However, unlike a *Goto* command you must "Return" back from a sub-routine using the *Return* command.
For every ***Gosub*** command in your code, you should have a equal *Return* command, failure to do this may result in overflowing the ***Gosub*** stack.

RETURN Command

Syntax : RETURN

You must always match a *Gosub* command with a equal ***Return*** command. When GamesBasic encounters a ***Return*** it will jump back to the next command after the original *Gosub* which called the sub-routine.

Example

```
….
Gosub SayHello
Print "Good Bye"
End

SayHello:
Print "Hello World!"
Return
```

IF..Then..Else Structure

Overview

The ***If..Then..Else*** structure is used to make decisions within your code, and allow you to take a different path through it. This control structure can be used to make a choice between drawing a circle or a square, making a sprite walk or run, etc.

IF Command

Syntax : IF <Condition>

The ***If*** command is used to start the ***If..Then..Else*** structure and requires a condition. This condition is tested to see if it returns True Or False, depending on that result either the *Then* or *Else* commands are ran.

If a condition returned True, GamesBasic will run any statements after the *Then* command.
If the condition returns False, and the *Else* statement exists, GamesBasic will run the code after the *Else* command.

THEN Commands and ELSE Commands

Syntax : THEN <Statements>
Syntax : ELSE <statements>

The ***Then*** / ***Else*** statements are executed when a condition returns True or False. Depending on the format of the IF command either 1 or more statements will be executed. The ***If..Then..Else*** structure can have 2 very different formats.

Format One

```
IF <condition> THEN <statement> ELSE <statement>
```

Format Two

```
2) IF <condition> THEN
      <statements>
     ENDIF | [ELSE
       <statements>
     ENDIF]
```

Format One is a single line ***If..Then..Else*** statement and is used when you have one 1 command that you wish executed (depending on the result).

Example

If A = 1 Then B = 1 Else B = 2

Format Two is used to allow multiple statements after the *Then* and *Else* commands. If you use format number two, make sure you include the ENDIF statement to let GamesBasic know the end of the THEN and ELSE statements.

Example

```
If A = 1 Then
   B = 1
   C = 1
  Else
    B = 2
EndIf
```

Graphic Commands

Drawing Simple Shapes

PLOT Command

Syntax : PLOT [SCREEN <no><,>] <x>,<y>[,<colour>]

The plot commands allows the user to plot a single pixel of information onto the display, they provide the command with the desired X and Y locations, and optionally a colour palette index (a number between 0 and 255). This command, like most other graphic commands supports the SCREEN parameter, which allows you to draw directly to screens which are not the current screen.

Example

```
PLOT 100,100,2
PLOT SCREEN 0,100,100,2
```

LINE Command

Syntax 1 : LINE [SCREEN <no><,>] <x1>,<y1> to <x2>,<y2>
Syntax 2 : LINE [SCREEN <no><,>] to <x>,<y>

The line command allows the drawing on a single pixel width line from a starting point to an end point. The line command must always be supplied with a end point (X,Y), however it supports two different ways of supplying the start position. After all drawing operation the 'graphics cursor' is usually located where the last operation ended. Using the LINE TO version (Syntax 2) of the command, you can draw from that last point to a supplied end point. If however, you want to draw a line somewhere else use the fully syntax 1 of the command.
This command contains no optional colour parameter, the line colour is set using the INK command.

Example

```
Line 100,100 To 200,200
Line To 200,200
```

SET LINE Command

Syntax 1 : SET LINE <value>

The set command used with the suffix LINE, allow you to set the style for drawing modes, such as dashed, dotted, etc. The <value> must be between 0 and 5.
Note : DirectX does not support this command, as yet, but its supported in GamesBasic for future use.

BOX Command

Syntax : BOX [SCREEN <no><,>] <x1>,<y1> to <x2>,<y2>

The ***Box*** command provides a way of drawing square or rectangle shapes directly onto any screens. The ***Box*** command requires two points, the upper left corner <x1>,<y1> and the bottom right corner <x2> and <y2>. Using these points GamesBasic will draw on the current screen, a box using the co-ordinates supplied. Adding the optional SCREEN parameter allows you to specify a screen in which you want to draw the box.
The box colour is controlled via the INK command, for a filled rectangle see the ***Bar*** command for more information.

Example

```
Box 0,0 To 200,300
Box Screen 10,10,10 To 20,200
```

CIRCLE Command

Syntax : CIRCLE [SCREEN <no><,>] <x>,<y>,<radius>

By use of the ***Circle*** command users can draw perfect circles, only by supplying the centre points of the circle [<x> ,<y>] and the radius (in pixels). Circles are hallow, and are drawn in the current ink colour. To fill a circle, use the *PAINT* command.

TEXT Command

Syntax : TEXT [SCREEN <no><,>]<x1>,<y1>,<expression>

The text command allows the user to quickly provide text information on screen. The X and Y co-ordinates state the upper left corner for the text, while the usage of the expression allows text, variables and functions to be used.
For more information on text commands, see the text oriented commands under the TEXT section.

Example

```
Text 100,100,"Hello World"
Text Screen 1,200,200,"The counter is currently" + Counter
Text X_Location, Y_Location, "You are now at " + X_Location + "," + Y_Location
```

Extended Graphics

BAR command

Syntax : BAR [SCREEN <no><,>] <x1>,<y1> to <x2>,<y2>

The ***BAR*** command is an extension of the standard *Box* command for drawing rectangles or squares. This variation fills the box with the current Brush colour, (see *BRUSH* command for more information), whilst drawing the outline in the Ink Colour (see *INK* command).

ELLIPSE command

Syntax : ELLIPSE [SCREEN <no><,>] <x>,<y>,<radius1>,<radus2>

This command is similar to the *Circle* command, except using an extra radius parameter you can draw ellipse's. See the *Circle* command for more information.

ARC command

Syntax : ARC [SCREEN <no><,>]<x1>,<y1>,<x2>,<y2>,<x3>,<y3>,<x4>,<y4>

Used to draw an arc, use this command. The following description of parameter is technically strong : The arc traverses the perimeter of an ellipse that is bounded by the points (X1,Y1) and (X2,Y2). The arc is drawn following the perimeter of the ellipse, counter clockwise, from the starting point to the ending point. The starting point is defined by the intersection of the ellipse and a line defined by the centre of the ellipse and (X3,Y3). The ending point is defined by the intersection of the ellipse and a line defined by the centre of the ellipse and (X4, Y4).

PIE command

Syntax : PIE [SCREEN <no><,>]<x1>,<y1>,<x2>,<y2>,<x3>,<y3>,<x4>,<y4>

The pie command draw a pie shaped wedge, and is similar to the arc command in that it draws an arc. However at the end of a arc, it will draw lines from the last point to the centre co-ordinates of the arc and then to the start of the arc. Thus, a pie shaped wedge. See *Arc* command for more details..

POLYLINE command

Syntax 1 : POLYLINE [SCREEN <no><,>] <x1>,<y1> (To <x2>,<y2>)
Syntax 2 : POLYLINE [SCREEN <no><,>] (To <x2>,<y2>)

The polyline command is used to draw a number of lines, joined to each other. This command supports a large number of <x> and <y> values, each value state another point in the line. Syntax 1 is used when you want to give the starting point, however if you want to start from the current graphics cursor position use Syntax 2.

Example

```
Polyline 100,100 To 200,200 To 300,300 To 150,150
Polyline Screen 0,200,100 To 200,200
Polyline To 500,500 To 300,200 To 100,100
Polyline Screen 1,To 400,400
```

POLYGON command

Syntax 1 : POLYGON [SCREEN <no><,>] <x1>,<y1> (To <x2>,<y2>)
Syntax 2 : POLYGON [SCREEN <no><,>] (To <x2>,<y2>)

The polygon command is identical to the standard polyline command, however it will draw a line from the end co-ordinates back to the first point, and then fill in the enclosed shape using the brush colour.

Advanced Graphics

GFXLOCATE command

Syntax : [SCREEN <no><,>] <x>,<y>

The Graphics Locate command or ***GFXLocate***, allows you to set the current position of the graphics cursor at run time. By supplying the new <x> and <y> values, and optionally a screen, GFXLocate will move the graphics cursor.

PAINT command

Syntax : PAINT [SCREEN <no><,>]<x1>,<y1>,<colour no.>

The paint command is used to flood fill any closed objects, such as a rectangle, circle or polygon. Using the <x> and <y> co-ordinates you pass this instruction the centre location for the flood fill to begin, the colour no is a palette index value.
Bug: There is a problem where this command paints in black or white, with no colour.

CLIP command

Syntax : CLIP [SCREEN <no><,>] <x1>,<y1> To <x2>,<y2>

The clip command is not currently supported by graphics commands, however it should 'clip' any drawing operations within the boundaries of a rectangle specified by x1,y1 and x2,y2. Calling this command currently will waste CPU cycles.

Graphic Functions

Location Functions

GRLocateX Function

Syntax : GRLocateX

This command returns an integer value presenting the X location of the graphics cursor on the current screen.

GRLocateY Function

Syntax : GRLocateY

This command returns an integer value presenting the Y location of the graphics cursor on the current screen.

Screen colour commands

Colour commands, affecting graphics

INK Command

Syntax : INK [SCREEN <no><,>]<colour no.>

The ***Ink*** command is used to set the colour of the current 'ink' used for drawing operations. The 'ink' colour is used in the drawing of all hallow shapes, whilst it is used in the outline of all solid or filled shapes. The colour parameter is a 'Palette Index' value, between 0 and 255 (due to the 256 colour display). The palette can be changed by using the *Palette* command. See the *Flash* command for information on cycling the values of a colour palette entry.

BRUSH Command

Syntax : BRUSH [SCREEN <no><,>]<colour no.>

Similar to the *Ink* command, this command allows the setting of the current 'brush' used for drawing operations. The 'brush' is used to draw the inside of filled shapes. Please see the *Ink* command for a more detailed explanation.

Display Resolution commands & Functions

Changing Resolution

GamesBasic supports the changing of resolutions to a valid resolution. However, there are no functions yet which return if a resolution can be achieved, these will follow shortly.

RESOLUTION Command

Syntax: RESOLUTION <width>X<height>

By calling the ***Resolution*** command you can change the current resolution of the application. Standard values include 320x240, 640x480, 800x600 and 1024x768.

Example

```
Resolution 800x600
```

Creating and destroying screens

SCREEN OPEN Command

Syntax : SCREEN OPEN <screen no>,<width>,<height>

To open a screen other than the default screen (screen 0), use the ***Screen Open*** command to create a new screen. You supply, to the command, the number of the new screen (there is currently a limit of 16 screens, 0 -> 15), and the width and height values. GamesBasic will create the screen, and display it in the upper left hand side of the display. Screens can have independent widths and heights, but they do share the current resolution and colour palette.
Once you have opened a new screen, GamesBasic assumes it to be the 'current screen', and all drawing operations without the optional SCREEN command will draw to it. See the *SCREEN CURRENT* command to change this.

SCREEN OPEN Command

Syntax : SCREEN CLOSE [<screen no>]

Use this command is close the current screen (if no screen number is given) or using the optional parameter a selectable screen. When you close a screen, GamesBasic will assign the 'current screen' to the screen behind the closed one.

SCREEN CLONE Command

Syntax : SCREEN CLONE [<screen no>]

The clone command allows you to 'clone' the current screen and make an exact copy of it. The copy is then assigned the next free screen number, or the number optionally supplied with the command. A cloned screen will act independently of the current screen, and behaves as a new screen.
Consideration : The graphics present on the old screen are not cloned onto the new screen, this will however be supported before final release.

Managing Screens

SCREEN CURRENT Command

Syntax : SCREEN CURRENT <screen no>

To change the 'current' screen, use this command supplied with a screen number. This command is useful when you do not want to place the keyword SCREEN in front of your graphics commands.

SCREEN TO FRONT Command

Syntax : SCREEN TO FRONT [<screen no>]

As screens are created or cloned, they become the current screen and appear in front of any other screens. If however you want to work on a screen that is behind other screens, you must bring it to the front so its can be correctly seen. Using the SCREEN TO FRONT command allows you to bring the 'current' screen to the front (if you supply no optional screen number), or if you do the selected screen.

Note : Once a screen has been moved to the front, it is automatically made the ‘current screen’.

SCREEN TO BACK Command

Syntax : SCREEN TO BACK [<screen no>]

The SCREEN TO BACK works in a identical way, except moves the screen to the back of all the others.

Note : The screen which has been moved to the back, becomes the ‘current screen’. The screen now at the front is NOT the current screen.

Screen manipulation

SCREEN OFFSET Command

Syntax : SCREEN OFFSET <screen no>,<x offset>,<y offset>

The command allows you to position a screen relative to the upper left corner of a screen (co-ordinates 0,0). By giving an offset from this point, you can position a screen anywhere in space - even off or half-on the computer's display. The usage of negative x and y numbers is allowed, this allows right to left, or down to up movement of a screen.

SCREEN RESIZE Command

Syntax :SCREEN RESIZE <screen no>,<width>,<height>

After you have initially created a screen, you may wish to resize it during its 'lifetime'. By supplying the screen number, along with a new width and height value - GamesBasic will resize the selected screen.
Consideration : GamesBasic does not attempt to remember what was in the screen, and will erase its to the standard screen background colour - this will change in later releases.

Screen update and setting commands

SET AUTO UPDATE Command

Syntax : SET [AUTO UPDATE <on>/<off>]

GamesBasic updates the screen automatically every 20ms (or 50 times a second), allowing for smooth graphics and no flickering. If you don't want GamesBasic to update the display at this regular interval, you need to turn the system off. Using the SET AUTO UPDATE command, you can specify whether GamesBasic automatically updates the display, or whether you have to do it manually. To manually update the display at the required interval, called the *Display Update* Command.

Example

```
Set AutoUpdate off
.... Do some instructions ...
Display Update
```

DISPLAY UPDATE Command

Syntax : DISPLAY [UPDATE <VB Wait>]

When AUTO UPDATE is off (using SET command), the display will not be updated by Games Basic automatically. You must call the DISPLAY UPDATE command to update the contents of the current screen.
The optional parameters <VB Wait> is used to make GamesBasic wait until a vertical blank before updating the display. Supply either 'True' or 'False', if no option is supplied 'False' is assumed. Normally you do no have to wait for the vertical blank, however if you experience flickering when calling this command, see <VB Wait> to True.

Example

```
Display Update
Display Update True
```

CLS Command

Syntax : CLS [SCREEN <no>[<,>]][<colour no>]

This command clears the current screen (if no optional screen is given) with the current Brush colour (if no optional colour palette index is given)

Example

CLS	- Clears current screen to default colour
CLS SCREEN 1	- Clears screen 1 to default colour
CLS SCREEN 1,124	- Clears screen 1 to colour 124
CLS 124	- Clears current screen to colour 124

Mathematical commands

Different types of results

Overview

Most mathematical functions, such as Cos and Sin, can return results in different format i.e. Cosine can return its result in degree’s or radian’s. Use the commands listed here to change the results returned by a mathematical functions.

DEGREE Command

Syntax : DEGREE

The ***Degree*** command sets the type of result returned by functions which can return either degree of radian values. Cosine and Sine are prime example. By calling this command, GamesBasic will return the results in a degree’s.

RADIAN Command

Syntax : RADIAN

Similar to the *Degree* command, ***Radian*** is used to tell GamesBasic to return Radian’s.

Number manipulation commands

Overview

This section details the manipulation of numbers. Most of these commands can be used as substitutes for mathematical expressions, i.e. ‘A =A + 1’, can be performed 35% faster by calling the ‘Inc (A)’ command. As shown, the main reason for using these commands is speed, however their use is limited and sometimes required proper setting-up (i.e. variables must exist).

INC Command

Syntax : INC <Variable>

The ***Inc*** (increment) command allows you to perform fast integer based addition. ***Inc*** performs the same function as ‘Variable = Variable + 1’, but almost 35% faster. Variables must already be declared and/or used, because ***Inc*** will not create a undeclared variable.

DEC Command

Syntax : DEC <Variable>

The ***Dec*** (decrement) command is the reverse of the *Inc* command, taking 1 (one) away from a supplied variable. Again, make sure its declared before calling this command.

ADD Command

Syntax : ADD <Variable>,<expression>[,<base> to <top>]

The ***Add*** command performs a fast addition to a variable. The variable must exist before you call this command, and the expression must resolve to a valid number. The <base> and <top> optional parameters are used to allow a boundary for the addition. Using <base> and <top> have the same purpose as calling ‘IF VARIABLE < BASE THEN VARIABLE=TOP ; IF VARIABLE > TOP THEN VARIABLE=BASE’.

SUB Command

Syntax : SUB <Variable>,<expression>[,<base> to <top>]

The ***Sub*** command is used to subtract an expression from a variable, this command is much faster than using Variable = Variable – Expression, and has the added advantage of the <base> and <top> parameters – See the *Add* command for more details.

RANDOMIZE Command

Syntax : RANDOMIZE <seed>

The ***Randomize*** command initalizes the built-in random number generator with a random value (obtained from the system clock). By using the optional <seed> expression you can repetitively generate a specific sequence of random numbers. This is useful for applications that deal with data simulations.
However, it is not recommended that you use this command to generate a set of random numbers for data encryption or similar purposes that require reproducible sequences of pseudo-random numbers. The implementation of the Random function (which returns the random number's) may change between revisions of Games Basic.

Mathematical functions

Simple functions

Overview

Mathematical functions are an important part of any computer language, and GamesBasic is not without its impressive list of functions. Please remember, for applications which require high-speed execution (such as games or demos) mathematical functions do carry a high CPU load!

ABS Function

Syntax : *Result* = ABS (value)

The ***Abs*** function is used to convert a value into an absolute value, i.e. it makes the number positive.

INT Function

Syntax : *Result* = INT (real value)

When dealing with real (floating point) numbers, it is sometimes necessary to view only the integer (whole) section of the number. Use ***Int*** to return just the integer part of the number, rounded towards zero.

PI Function

Syntax : *Result* = PI

The ***Pi*** function is used to produce *pi* for all mathematical calculations that require it. *Pi* is the ratio of a circle's circumference to its diameter. Pi is approximated as 3.14159265358979323 85.

Extended mathematical functions

MAX Function

Syntax : *Result* = MAX (value1,value2)

The ***Max*** function compares two values (expressions) and returns which one is larger. The return value of '–1' indicates that the left size was largest, however a return value of '+1' or '1' indicates the right side was largest. If both values are the same, a '0' is returned.

MIN Function

Syntax : *Result* = MIN (value1,value2)

Identical to the *Max* function, however returns the minimum of two values. See *Max* function for more details.

SGN Function

Syntax : *Result* = SGN (value)

The function ***Sgn*** returns the sign of the value or expression passed to it. This function will return '-1' if the value is negative, '0' is the value is zero or '1' if the value is positive.

SQR Function

Syntax : *Result* = SQR (value)

The ***Sqr*** function returns the square of the value passed.

EXP Function

Syntax : *Result* = EXP (value)

The ***Exp*** function returns a value, raised to the power of the value supplied. Where the value returned is the base of natural logarithms.

LOG Function

Syntax : *Result* = LOG (value)

The ***Log*** function returns the log base 10 of the supplied value.

LN Function

Syntax : *Result* = LN (value)

The ***LN*** function returns the natural logarithm of the value supplied.

Trigonometry functions

COS Function (Cosine)

Syntax : *Number* = COS (angle)

The ***Cos*** function returns the cosine of a given angle.

SIN Function (Sine)

Syntax : *Number* = SIN (angle)

The ***Sin*** function returns the sine of a given angle.

TAN Function (Tangent)

Syntax : *Number* = TAN (angle)

The ***Tan*** function return the tangent of the supplied value.

ACOS Function (Arc Cosine)

Syntax : *Number* = ACOS (angle)

The ***ACos*** function returns the Arc Cosine of a given angle.

ASIN Function (Arc Sine)

Syntax : *Number* = ASIN (angle)

The ***ASin*** function returns the Arc Sine of a given angle.

ATAN Function (Arc Tangent)

Syntax : *Number* = ATAN (angle)

The ***ATan*** function return the Arc Tangent of the supplied value.

HCOS Function (Hyperbolic Cosine)

Syntax : *Number* = HCOS (angle)

The ***HCos*** function returns the Hyperbolic Cosine of a given angle.

HSIN Function (Hyperbolic Sine)

Syntax : *Number* = HSIN (angle)

The ***HSin*** function returns the Hyperbolic Sine of a given angle.

HTAN Function (Arc Tangent)

Syntax : *Number* = HTAN (angle)

The ***HTan*** function return the Hyperbolic Tangent of the supplied value.

String Functions

Simple Functions

LEFT Function

Syntax : *String* = LEFT (source string, count)

The ***left*** function reads a number of characters for the source string, starting at the left-hand side. The count should be >0 and has no 'real' upper limit (i.e. its can go up to the maximum integer value supported). If you supply a count of 0, ***Left*** returns a empty string.

RIGHT Function

Syntax : *String* = RIGHT (source string, count)

The ***right*** function starts at the right hand side of the source string, and returns the 'count' number of characters. Count should be, again, greater than zero. However, if you supply 0 or a negative number, ***right*** returns a empty free.

MID Function

Syntax : *String* = MID (source string, start position, character count)

The ***Mid*** function is used to return a number of characters from the middle of a string. You supply the starting position and the number of character you want returned.

INSTR Function

Syntax : *number* = INSTR (guest string, host string [,start position])

The ***Instr*** function is used to return the position of a 'host' string, within a 'guest' string. For example, the string 'And' occurs at the 6th position in the following string 'Hello And Welcome'. This function IS case-sensitive, to convert it to a non-case-sensitive operation, uppercase or lowercase both strings before performing the ***INSTR*** function.
The optional start position is used to offset where GamesBasic starts to look for the 'host' string within the 'guest' string. If GamesBasic doesn't find the string, -1 (false) is returned.

Case conversion.

The case conversion functions, uppercase and lowercase, are used frequently when dealing with interpreting user-input. By converting the user-input into standard format (E.g. lowercase) you can easily test results and save speed in comparing both upper and lower case characters.

UPPERCASE Function

Syntax : *String* = UPPERCASE (String)

This function converts the characters in a string into upper case (capital) letters, and places the results into a new string.

LOWERCASE Function

Syntax : *String* = LOWERCASE (String)

This works in a similar way to the uppercase function, however the new string created contains lowercase letters.

Manipulating Strings

Sometimes you may want to handle your strings for special purposes, such as printing large volumes of continuous information.

STRING Function

Syntax : *String* = STRING (character, count)

The ***String*** function allow you to create a new string using the first character of a existing / supplied string. By supplying the character and a count value, ***String*** returns a new string containing 'count' number of the first character.

Example

A = String("A",10)	, creates a string = 'AAAAAAAAAA'
B = String ("Happy",5)	, creates a string = 'HHHHH'

SPACE Function

Syntax : *String* = SPACE (count)

The ***Space*** function is extremely similar to the *String* function, however it creates a string contain 'count' number of spaces. This function is provided, due to the large amount of spacing required by some programs. This function is much faster than using the *String* function to do the same role.

Example

A = Space(10) , creates a string = '^ ^ ^ ^ ^ ^ ^ ^ ^ ^' ['^' = a space]

FLIP Function

Syntax : *String* = FLIP (source string)

The ***Flip*** command is used to flip or reverse the order of characters in a string.

Example

New = Flip("!BG gnippilf")   , creates a string = 'flipping GB!'

REPEAT Function

Syntax : *String* = REPEAT (source string, count)

To repeat the same string (not character) use the ***Repeat*** function. This builds a new string, contains 'count' number of copies of the source string.

Example

New = Repeat('HI!',3)   , creates a string = 'HI!HI!HI!'

CHR Function

Syntax : *String* = CHR (number)

By passing a number to this function, ***Chr*** will return a single character representing the ASCII value of that number .

Example

New = Space(65)   , The variable New – contains the character 'a'

ASC Function

Syntax : *Number* = ASC (string)

The ***Asc*** function performs the reverse of the *CHR* function, its will return the ASCII value of a supplied character.

LEN Function

Syntax : *Number* = LEN (String)

The ***Len*** command returns the length of a supplied string.

Text commands and functions

Overview

What are text commands and functions?

GamesBasic provides a way of placing text onto screens in a quick and easy way. The Text commands and function provide you with a range of commands to perform the operations, and functions to return results about these commands. This section assumes you understand fonts and their details (such as size, name, etc.)

Scope of commands and functions

Each of the font commands and functions will set the font only on the current screen, so screens can have different font details.

Font Manipulation Commands

Overview

The commands in this section allow you to change the various different characteristics of fonts, such as their size, style, colour and drawing mode.

FONT Command

Syntax : *FONT <section> <value>*

The ***Font*** command allows you to change the many different settings associated with fonts. This one command, by use of the section parameter, can be used to set the current font's size, name, style, colour, etc.

FONT SIZE Command

Syntax : *FONT SIZE <value>*

This variation of the ***Font*** command is used to set the size of the font. For monospaced or fixed-width fonts the size may have to be a set value or the resulting font may look blocky. For Proportionally spaced or TrueType fonts, the size is less important as GamesBasic can scale the font with good accuracy.

FONT NAME Command

Syntax : *FONT NAME <value>*

By using the *NAME* variation it is possible to change the font's face (or name). If you supply a font name not installed on the system, GamesBasic will look for the closest match (by looking at all other fonts with similar styles, sizes, pitch, etc.)

Example

```
Font Name "Tohama"
Font Name Arial

Let New_Font = "Courier"
Font Name New_Font
```

FONT STYLE Command

Syntax : *FONT STYLE <value>*

The *Font Style* command is used to set features such as bold, italics, underline or strikeout. However, the value passed to font style is a binary value. The following table show bit allocations.

Bit No	Meaning	Value
1	Bold	1
2	Italics	2
3	Underline	4
4	Strike Out	8

Example

Font Style 1	, 0 = Binary '0000'	no styles (normal)
Font Style 1	, 1 = Binary '0001'	thus bold is set
Font Style 3	, 3 = Binary '0011'	thus bold and italics is set
Font Style 14	, 14 = Binary '1110'	all styles are on, except bold
Font Style 15	, 14 = Binary '1111'	all styles

FONT COLOR Command

Syntax : *FONT COLOR <value>*

The ***Font Color*** command is used to set the colour of the current font, which is expressed in a RGB Value.
Note : Before the font is drawn, GamesBasic looks up the palette for the closest match to the RGB value entered.

Example

Font Color 255	, full Blue
Font Color 0	, black
Font Color 16777215	, white
Font Color 65025	, yellow

FONT DMODE Command

Syntax : *FONT DMODE <value>*

The ***Font DMode*** command is used to set how any text, printed using the font, will be added to the existing graphics on the current screen. The system caters for a number of different modes, use the following table to set the <value>.

Value	Name	Description
1	Replace	This mode will place the text onto the current screen by : 1. Clearing the area beneath the text, to the colour set by <value2> (background colour) by the *Font Color* command 2. Draw the text into that area
2	Transparent	This mode will draw text transparently onto the current screen, and will not destroy any area beneath it.
3	AND	The text will be logically AND'ed with the current screen.
4	XOR	The text is Exclusive OR'ed with the current screen data.
5	IGNORE	All font based operations, are now ignored.

Font Manipulation Functions

Overview

These commands allow you to query details about the current fonts begin used and available for use.

FONTNAME Function

Syntax : String = FONTNAME <integer index>

The ***FontName*** function returns the name of the font at location <integer index> within the current font list. The <integer index> runs between zero (0) and the value returned by *FontMax.*

Example

```
For Counter = 0 To FontMax
        Print FontName(Counter)
Next Counter
```

FONTMAX Function

Syntax : *Integer* = FontMax

The ***FontMax*** function returns the number of fonts currently available to the user.

FONTHEIGHT Function

FONTWIDTH Function

Syntax : *Integer* = FontHeight
Syntax : *Integer* = FontWidth

As their names suggest ***FontHeight*** and ***FontWidth*** both return the current font's height and width in pixels.

TEXTWIDTH Function

Syntax : *Integer* = TextWidth(Expression)

By supplying a string expression to ***TextWidth***, GamesBasic will return the number of pixels required to display this text in the current font.

MAXCHARSWIDE Function

MAXCHARSHIGH Function

Syntax : *Integer* = MaxCharsWide
Syntax : *Integer* = MaxCharsHigh

These two functions both return the number of characters which will fit horizontally and vertically in the current screen. Accurate values can only be returned for mono-spaced fonts because each characters is set to a standard size.

FONTSTYLE Function

Syntax : *Integer* = FontStyle

This command returns the current style in use in the form of a number. Please see the *Font Style* command for more information on the value returned.

XCURS Function

YCURS Function

Syntax : *Integer* = XCurs
Syntax : *Integer* = YCurs

Returns the current position of the text cursor.

Direct Font Manipulation Commands

Overview

These commands provide a way of directly setting some of the font styles and drawing modes for the current screen. These commands are provided, due to the fact that they are faster than called the *Font* command, and can better understood.

BOLD Command

Syntax : *BOLD <on> ! <off>*

The ***Bold*** command allows uses to turn on and off the bold style for the current screen. This command performs the same function as the *Font Style* command, only faster.

ITALICS Command

Syntax : ITALICS *<on> ! <off>*

The ***Italics*** command allows uses to turn on and off the italics style for the current screen. This command performs the same function as the *Font Style* command, only faster.

UNDER Command

Syntax : UNDER *<on> ! <off>*

By using the ***Under*** command users can turn the underline style for the current screen on and off. This command performs the same function as the *Font Style* command, only faster.

STRIKE Command

Syntax : STRIKE *<on> ! <off>*

By using the ***Strike*** command users can turn the strikeout style for the current screen on and off. This command performs the same function as the *Font Style* command, only faster.

INVERSE Command

Syntax : INVERSE *<on> ! <off>*

Swaps the paper (brush) colour and the font colour around, does not affect normal graphics commands. This commands performs the same function as the *Font DMode* command, only faster. Setting inverse to <off>, sets the Draw Mode (DMode) to Replace.

SHADE Command

Syntax : SHADE *<on> ! <off>*

This function turns the shading of text on and off. When on, all text will be drawn in a grey colour. If the current palette does not contain a grey colour, GamesBasic looks for the nearest match. This commands performs the same function as the *Font DMode* command, only faster. Setting to <off>, sets the Draw Mode (DMode) to Replace.

Text Commands

Overview

GamesBasic provides a host of commands which allow you to easily output text in various formats to the current screen.

Note : This commands are powerful and versatile, however they do require a large amount of time to execute and thus are not recommended for use in applications which require high-speed command execution.

CENTER Command

Syntax : CENTER <expression>

The ***Center*** offers a quick and easy way of centring any text onto the current screen. The center command starts drawing text at the current graphics location, which can be set using *GFXLocate* command – Since we center text, you need only set the vertical parameter – the horizontal value is not important.

Example

```
Screen Open 1,400,400
GFXLocate 0,100
Center “Hi this is center in screen”
```

PRINT Command

Syntax : PRINT <expression>

The ***PRINT*** command is the most powerful of GamesBasic’s text orientated commands, allowing some complex operators. The ***Print*** command supports text grids (where text is aligned to a special invisible grid) and text preformatting (where GamesBasic will read imbedded escape codes within an expression to change font styles, sizes, colours, etc.). Please refer to the sections of *Text Grids* and *Text Preformatting* for more information.

? Command

See *Print* command for more information

Text Grids

Overview

Text Grids offer an excellent way of displaying information on screen, and always ‘knowing’ exactly where the any character has been or is going to be placed. The Text Grid system only works with Mono-spaced fonts, due to the need for GamesBasic to know each fonts width and height. If you are currently using a variable width font when you turn on the text grids, GamesBasic will attempt to find a fixed width font which matches it. There is no guarantee that GamesBasic will find a fixed font. If you continue to use Text Grids without setting a fixed width font, GamesBasic will not be able to ‘know’ the position of any characters, and will return unpredictable results.

SET TEXT GRID Command

Syntax : *SET TEXT GRID <on> / <off>*

The ***Set Text Grid*** command is used to turn the text grid system on and off. When you turn this system on, GamesBasic will draw all monospaced fonts in a grid. Allowing you to navigate around this grid using commands such as *Cup* and *CDown*.
Note : Turning this on when you are using a variable width font will result in GamesBasic looking for a fixed width font which matches *ALL* attributes. If GB doesn’t find a found with these attributes, the font stays unchanged. Thus, all operations will be unpredictable due to the variable width font.

Text Grid - Navigation Commands

This section details what commands to use for navigating around text grids, these commands only correctly work when the *Set Text Grid On* command is issued, and a mono-spaced font is used.

HOME Command

Syntax : *HOME*

The ***Home*** command is used to place the text cursor back to the upper left hand corner of the current screen. This upper corner is refereed to as co-ordinates (0,0). The ***Home*** command performs the same function as ***Locate 0,0***.

LOCATE Command

Syntax : *LOCATE <x>,<y>*

The ***Locate*** command allows absolute control over the text cursors position on the current screen. ***Locate*** requires 2 parameters, an X and Y Value. These values are the grid co-ordinates you wish to place the text cursor at.
Note : These X and Y values are not screen co-ordinates.

MEMORIZE Command

Syntax : *MEMORIZE <what>*

The ***Memorize*** command is used to allow you to record the current X and/or Y text cursor location. By supplying either 'X' or 'Y' or 'XY', you can instruction GamesBasic to record either the X or Y value, or both.

REMEMBER Command

Syntax : *REMEMBER <what>*

The ***Remember*** command is used in conjunction with the M*emorize* command to recall any 'saved' or memorised text co-ordinates. By supplying either 'X' or 'Y' or 'XY' you will tell GamesBasic which of the co-ordinates to recall.

CMOVE Command

Syntax : *CMOVE <x>,<y>*

Similar to the *Locate* command the ***CMove*** command allows you to move the text cursor. Instead of supplying an absolute value, you supply an offset value from the current text cursor's position. For example *Locate 10,20* would move the text cursor to 10,20. If after this you called ***CMove 10,20*** GamesBasic would move the cursor to 20,40. It thus adds the value 10 to the existing X location of 10, and 20 to the existing Y location of 20.

CUP Command

Syntax : *CUP*

This command moves the text cursor up one place, the same as calling *CMove 0,-1*

CDOWN Command

Syntax : *CDOWN*

This command moves the text cursor down one place, the same as calling *CMove 0,1*

CLEFT Command

Syntax : *CLEFT*

This command moves the text cursor left one place, the same as calling *CMove -1,0*

CRIGHT Command

Syntax : *CRIGHT*

This command moves the text cursor right one place, the same as calling *CMove 1,0*

Text Grid - Scrolling

GamesBasic allows you to easily scroll the contents of a line vertically or horizontally. By using the following two commands, text grids can be scrolled to produce text that roles onto the screen.

HSCROLL Command

Syntax: *HSCROLL (<L>|<S>,<L>|<R>)*

The ***HScroll*** command is used to scroll a line or the entire screen horizontally 1 text Grid Square (i.e. Width of the current font). You have to supply two parameters. The first parameter is the letter 'L' or 'S' – Stating the **L**ine or **S**creen is to be scrolled. The second parameter is 'L' or 'R' for the direction, **L**eft or **R**ight.

VSCROLL Command

Syntax: V*SCROLL (<L>|<S>,<U>|<D>)*

The ***VScroll*** command is used to scroll a line or the entire screen vertically 1 text Grid Square (i.e. Height of the current font). You have to supply two parameters. The first parameter is the letter 'L' or 'S' – Stating the **L**ine or **S**creen is to be scrolled. The second parameter is 'U' or 'D' for the direction, **U**p or **D**own.

Text Preformatting

Text preformatting is a powerful feature of GamesBasic allowing you to change font sizes, colours, styles, includes tab's and even move the text cursor all using imbedded codes within expressions.

SET TEXT PREFORMATTING Command

Syntax : *SET TEXT PREFORMATTING <on>|<off>*

This command allow you to specify whether GamesBasic will preformat text before printing it on screen. This preformatting only works with the *PRINT* command, but allows complex commands to be imbedded in normal text. Imbedded codes can allow you to change the text size, style, colour, etc.

SET TAB Command

Syntax : *SET TAB <expression>*

This command allows you to supply at which column width a tab exists. The default value is 8, so if a tab is inserted the next text will be printed at the start of the next column (i.e. position 8 or 16 or 24 or 32 and so on..)

Text Preformatting – Functions and escape codes

The following functions can be used to imbed escape codes into expressions, along with each function you will find an the escape sequence that it generates – allowing you to create your own escape codes without calling the functions (please note all escape code characters are in UPPER case).

PreSTYLE Function

Syntax : *Escape Sequence* = PreStyle (<Bold>|<Italics>|<Underline>|<Strikeout>)

The ***PreStyle*** function generates a escape sequence which sets the current screens font style. By passing a simple keyword (not an expression) you can toggle the style on or off.

Example

Print PreStyle(BOLD) + "THIS IS IN HOLD!"
Print PreStyle(Variable_Containing_The_String_BOLD) + "THIS IS INVALID!"

The second version is invalid, PreStyle does not handle expressions.

Escape Sequence Generated

Chr(27) + "Fx"

X = B (for bold)
X = I (for italics)
X = U (for underline)
X = S (for strikeout)

PreINK Function

Syntax : *Escape Sequence* = PreInk (expression)

The ***PreInk*** function changes the current screens font colour to the palette index supplied. The ***PreInk*** function can handle complex expressions that return a valid integer value.

Example

Print PreInk(1) + “This is in red!”
Print PreInk(512 \ 2) + “This is in white “ + PreInk(1) + “but I'm seeing red!”

Escape Sequence Generated

Chr(27) + “Ix” + “ “

X = A valid integer string, representing the colour index.

Note : There must be an terminating space, or GB will not see the end of the number! (The space is not printed).

PreMOVE Function

Syntax : *Escape Sequence* = PreMove (X,Y)

The ***PreMove*** function generates a escape sequence which will allow you to move the cursor by X and Y amount. The X and Y parameters can be expression, as long as they return a valid integer value. X and Y can also be negative.

Example

```
Home
Print PreMove(10,10) + “I'm sitting at position (10,10)!”
Print PreMove(2,2) + “Bit I'm at 12,12 because 10 (old position) + 2 = 12!”
```

Escape Sequence Generated

Chr(27) + “Mdx” + “ “

D = Movement direction, values : “U” – Up, “D” – Down, “L” – Left & “R” - Right
X = A valid integer string, stating amount of moment!

Note : The PreMove function will generate 2 escape sequences, one for the Up/Down movement, the other for the Left/Right movement.
Note : There must be an terminating space, or GB will not see the end of the number! (The space is not printed).

PreLOCATE Function

Syntax : *Escape Sequence* = PreLocate (X,Y)

The ***PreLocate*** function generates a escape sequence which will set the location of the text cursor to the X and Y values supplied, rather than move them by the X and Y values. If you require movement by X and Y amount use ***PreMove***, if you want them set absolutely use ***PreLocate***. This function accepts expressions which resolve to positive or negative integers.

Example

Locate 20,20
Print PreLocate(10,10) + “I'm sitting at position (10,10)!”
Print PreLocate(12,2) + “I'm sitting at position (12,1)” + PreLocate(1,1) + “I'm at 1,1”

Escape Sequence Generated

Chr(27) + “Xz “ + “ “ + Chr(27) + “Yz” + “ “

z = Valid integer string.

Note : After each integer value there must be a space, without this GB will not be able to see the end of the integer value.

TAB Function

Syntax : *Escape Sequence* = TAB

The ***Tab*** function simply returns the tab escape code, use the *Set Tab* command to change the width of the tab.

Example

Set Tab 8
Print “Hello” + Tab + “I'm aligned to the 8th column!”

Escape Sequence Generated

Chr(27) + “T“

Note : No terminating space.

PreCDown Function

PreCUP Function

PreCLeft Function

PreCRight Function

Syntax : *Escape Sequence* = PreCDown(Amount)

All these functions move the cursor by the supplied amount. See the *CDown, CUp, CLeft* and *CRight* commands for more information. The amount can be a expression which resolves to a valid positive or negative number, GamesBasic will automatically convert (for example) a negative ***PreCRight*** into a positive ***CLeft*** function.

Example

Print “GamesBasic By James Happy” + PreCLeft(5) + “Hunter”

Escape Sequence Generated

Chr(27) + “Mdx” + “ “

D = Movement direction, values : “U” – Up, “D” – Down, “L” – Left & “R” - Right
X = A valid integer string, stating amount of moment!

Note : There must be an terminating space, or GB will not see the end of the number! (The space is not printed).

Text grid - functions

GRID Function

Syntax : *Boolean* = Grid

The ***Grid*** functions return true or false depending on whether text grids are currently enabled or not.

Date Time commands and functions

Overview

What are date and time commands / functions?

GamesBasic allows you the programmer to use date and times easy and effectively inside your programs. GamesBasic supports a universal standard to handle date and times, and allow you to format them into many different user-definable ways.

Date Commands and Functions

Overview

These commands and functions deal only with the reading and formatting of dates.

CURRENTDATE Function

Syntax : Result = *CurrentDate*

The ***CurrentDate*** function returns a real number which represents the number of days from 12/30/1899. This system is used to maintain compatibility with Windows and OLE 2.0 Automation.

DATETOSTR Function

Syntax : Result = *DATETOSTR(<expression>)*

The ***DateToStr*** function converts a real number into a valid date in the format specified in the ShortDateFormat global variable (in windows). The usual format is 'dd/mm/yy'

STRTODATE Function

Syntax : Result = *STRTODATE(<expression>)*

The reverse of the *DateToStr* function, ***StrToDate*** takes a string expression and tries to return a valid date real number. Failure to include a valid date in the string will result in an error.

Time Commands and Functions

Overview

These commands and functions deal only with the reading and formatting of times.

CURRENTTIME Function

Syntax : Result = *CurrentTime*

The ***CurrentTime*** function returns a real number which represents the number of days from 12/30/1899. This system is used to maintain compatibility with Windows and OLE 2.0 Automation.

TIMETOSTR Function

Syntax : Result = *TIMETOSTR(<expression>)*

The ***TimeToStr*** function converts a real number into a valid TIME in the format specified in the LongTimeFormat global variable (in windows). The usual format is 'hh:mm:ss'

STRTOTIME Function

Syntax : Result = *STRTOTIME(<expression>)*

The reverse of the *TimeToStr* function, ***StrToTime*** takes a string expression and tries to return a valid time real number. Failure to include a valid time in the string will result in an error.

Date & Time Misc. Commands and Functions

FORMATDATETIME Function

Syntax : Result = *FormatDateTime(<String>,<real>)*

This function allows the user to format the date and time exactly as he/she see fit. The first expression should return a string which can contain any of the values below, while the second expression should contain a real number describing the date/time..

c	Displays the date using the format given by the ShortDateFormat global variable, followed by the time using the format given by the LongTimeFormat global variable. The time is not displayed if the fractional part of the DateTime value is zero.
d	Displays the day as a number without a leading zero (1-31).
dd	Displays the day as a number with a leading zero (01-31).
ddd	Displays the day as an abbreviation (Sun-Sat) using the strings given by the ShortDayNames global variable.
dddd	Displays the day as a full name (Sunday-Saturday) using the strings given by the LongDayNames global variable.
ddddd	Displays the date using the format given by the ShortDateFormat global variable.
dddddd	Displays the date using the format given by the LongDateFormat global variable.
m	Displays the month as a number without a leading zero (1-12). If the m specifier immediately follows an h or hh specifier, the minute rather than the month is displayed.
mm	Displays the month as a number with a leading zero (01-12). If the mm specifier immediately follows an h or hh specifier, the minute rather than the month is displayed.
mmm	Displays the month as an abbreviation (Jan-Dec) using the strings given by the ShortMonthNames global variable.
mmmm	Displays the month as a full name (January-December) using the strings given by the LongMonthNames global variable.
yy	Displays the year as a two-digit number (00-99).
yyyy	Displays the year as a four-digit number (0000-9999).
h	Displays the hour without a leading zero (0-23).
hh	Displays the hour with a leading zero (00-23).
n	Displays the minute without a leading zero (0-59).
nn	Displays the minute with a leading zero (00-59).
s	Displays the second without a leading zero (0-59).
ss	Displays the second with a leading zero (00-59).
t	Displays the time using the format given by the ShortTimeFormat global variable.
tt	Displays the time using the format given by the LongTimeFormat global variable.
am/pm	Uses the 12-hour clock for the preceding h or hh specifier, and displays 'am' for any hour before noon, and 'pm' for any hour after noon. The am/pm specifier can use lower, upper, or mixed case, and the result is displayed accordingly.
a/p	Uses the 12-hour clock for the preceding h or hh specifier, and displays 'a' for any hour before noon, and 'p' for any hour after noon. The a/p specifier can use lower, upper, or mixed case, and the result is displayed accordingly.
ampm	Uses the 12-hour clock for the preceding h or hh specifier, and displays the contents of the TimeAMString global variable for any hour before noon, and the contents of the TimePMString global variable for any hour after noon.
/	Displays the date separator character given by the DateSeparator global variable.
:	Displays the time separator character given by the TimeSeparator global variable.

```
'xx'/"xx"Characters enclosed in single or double quotes are displayed as-is, and do
         not affect formatting.
```

Input : Keyboard Commands And Functions

Overview

GamesBasic allows you to access the keyboard easily and quickly, so you can get results on what the user is typing.

WAIT KEY Command

Syntax : Wait Key

The ***Wait Key*** command is a variation of the normal ***Wait*** command, but instead of supplying the number of sections you want GamesBasic to pause – you allow GamesBasic to remain paused (waiting) for a key press (any key).

This command does not effect any running sprites, music, effects, etc.

INKEY Function

Syntax : String = Inkey

The ***Inkey*** function allows you to retreive any key strokes which have occurred since your last access of the ***Inkey*** function.
This function is extremely handy, it allows you to forget about checking the keyboard until it really matters. Any keys you have not 'looked' at are stored until you call ***Inkey***. Unlike if you were to use the *KeyDown* functions, where you would have to constantly call the function to see if a new key has been pressed!

KEYDOWN Function

Syntax : Boolean = KeyDown(scancode)

This function allow you to directly read the state of any key on the keyboard. By supplying the scan code of a key to ***KeyDown*** you will receive either True or False. A True result shows you the key was down, False the key has not been pressed.

KEYSHIFT Function

KEYCTRL Function

KEYALT Function

Syntax : Integer = KeyShift
Syntax : Integer = KeyCtrl
Syntax : Integer = KeyAlt

These three functions are used to test the status of 6 special keys on the keyboard. These keys are the Control, Alt and Shift keys (both left and right side of the keyboard). Each function when called will return an integer value which will represent the current state of the key. The following table show you what the 4 integer values mean.

Integer Value	Description
0	Neither the left or right key was down
1	Left key depressed
2	Right key depressed
3	Both keys depressed

Example

```
Print KeyShift
If KeyCtrl = 1 Then Print "Left Control Key Down!"
```

User defined procedure and functions

Overview

GamesBasic provides a powerful way for users to structure there programs using their own user defined procedure and functions. GamesBasic supports procedures and functions (simply called procedures from here forth) listed below are some advantages and features of the system GamesBasic offers..

- Procedures can be placed anywhere in your code.
- You do not need to predefine them, before calling them.
- They support nearly unlimited parameter passing.
- They support the passing of full expressions, not just variables.
- They can be called recursively, until a stack overflow :)
- They support local variables, destroyed on exist from a procedure.
- There names are unlimited in size.

Functions

GamesBasic does not support user-defined functions (yet)..

Procedures

The following commands are for use in the declaration, handling and calling of procedures within GamesBasic.

PROCEDURE Command

Syntax : PROCEDURE <Name> [(<Variable>[,<Variable>])]

The ***Procedure*** command declares the following code, until the *EndProc* command to belong to the named procedure. A Procedure must have a unique name which is not an existing reserved word, variable or procedure name. You can optionally declare a list of variables what much be filled when the procedure is called.

Example

Call Print_Text_Count("Hello")

Procedure Print_Text_Count(TextToCount)
Print "Length of string " + TextToCount + " is " + Len(TextToCount)
EndProc

ENDPROC Command

Syntax : ENDPROC

The ***EndProc*** command is used to declare the end of a procedure. All *Procedure* commands must be terminated with a ***EndProc***.

CALL Command

Syntax : CALL <Proc Name> [(<Variable>[,<Variable>])]

The ***Call*** command is used to jump to a procedure and start executing the code within that procedure. If a procedure requires a number of variables, they must be supplied in brackets after the procedure name and separated by semicolons.
The commands after the call command will be executed when all the code has been ran inside the procedure, or a *ExitProc* command is performed.

EXITPROC Command

Syntax : EXITPROC

The ***ExitProc*** command performs the same role as *EndProc* except inside a procedure. When you need to terminate and jump out of a procedure, the ***ExitProc*** command is used to return control back to the instruction after the calling point.

Debugging Commands & Functions

GamesBasic provides a comprehensive set of debugging utilities. However you may need to gain information from deep within you programs – where normal debugging practices are just not possible! The following commands and functions are useful and can be called anywhere.

OUTPUTDEBUG Command

Syntax: OUTPUTDEBUG <expression>

This command is used in connection with the debug window (available only in the GamesBasic IDE – Not in standalone mode). When you open the debug window any expressions passed to the ***OutputDebug*** command are printed directly on the debug window.

TICKCOUNT Function

Syntax: Integer = TickCount

TickCount returns (in a integer) the number of millisecond that have passed single the computer was turned on. This value is used to see how fast a particular section of code is executing. For example. Take the value of TickCount at the start and end of a block of code, and subtract the two values. The difference is the number of milliseconds required to run the code!

Immedatate Mode

What is Immedaite Mode?

Immediate mode is a window that appears after you have finished running your program that allows you to remain inside the GamesBasic run-time environment and execute commands by typing them on the keyboard.
If for example you are running a program and you press 'ESCAPE' to break it (if that is the current Break Key) GamesBasic will popup the immediate mode window. If you program uses the variable 'Counter' you could easily type 'Print Counter' – which would cause GamesBasic to run the command. Thus the value of Counter variable would be printed where you text cursor is!

What commands can I use?

Due to the limits of typing only one command per line multi-line commands (such as FOR..NEXT or IF..THEN..ELSE statements) can not be used. However all other commands can be used, any errors will be reported inside the immediate mode window.

Current Limits..

There are a current number of limits or draw back to the immediate mode because of GamesBasic alpha state (ie. Its not finished) these will allow be removed in the later releases. The keyboard implementation is not good, it is a little buggy and we know the delete does not work. Also command link history does not work either!

Commands and functions special to Immediate Mode.

The following list of commands and functions are special to the immediate mode and executing them within GamesBasic will produce unreliable results.

LINES Command

Syntax: LINES <number>

Not supported yet. Will resize the display to <number> of lines.

QUIT Command

Syntax: QUIT

This command causes GamesBasic to finish executing the current program and return directly to the IDE. If issued in the immediate mode this will return the user to the IDE.

STOP Command

Syntax: STOP

Not Yet Supported. Will cause GamesBasic to exit immediate mode and restart the application where it was interrupted.

Error Messages

Run-time error messages

Overview

Run-time error messages, are errors which occur when the program is running. These sorts of errors can only occur while the program is running, for example a divide by zero error.

Error Listing

Error <0> : Failed To Start Tokenization ** Occurs when the tokenization routine fails to start even processing a source code line!

Error <1> : Testing Failed, there was not enough `Token Buffer` space to check your program for errors. Please check and increase the amount of `Token Buffer` space allocated.' ;

Error <2> : Testing Failed, could not test this program as there was not enough `Source Buffer` space to check your program for errors. Please check and increase of the amount of `Source Buffer` space allocated.' ;

Error <3> : Testing Failed, could not test this program because <Msg Generated By Validation Routines>

Error <4> : Testing Failed, the following problem raised during the first pass <Msg Generated by First Pass Routine>

Error <5> : Testing Failed, error is expression caused by <Msg>

Error <6> : Illegal screen number.

Error <7> : `TO` expected, <BLAH> found.'

Error <8> : Illegal APPEAR speed, value must be between 1 and 50.

Error <9> : Expected semi-colon `,` but found <BLAH>

Error <10> : Illegal <command> dimensions, please check all dimensions are present and valid integers.

Error <11> : Expected end of line.
Error <12> : Unexpected end of line found.

Error <13> : Expected `On` Or `Off`, but <BLAH> found

Error <14> : Run Time Error - Internal Error (RTIE), could not find a valid token in execution path.

Error <15> : Run Time Error - Screen <x> is not open.

Error <16> : Run Time Error - Loop stack overflow.

Error <17> : Run Time Error - Unterminated DO..LOOP structure, could not find LOOP command

Error <18> : Run Time Error - Invalid value within expression

Error <19> : Run Time Error - Invalid operator within expression

Error <20> : Run Time Error - Type mismatch within expression

Error <21> : Run Time Error - Value too big for variable

Error <22> : RTE (22) - Set Line command failed, line style value was out of bounds

Error <23> : RTE (23) - Circle command could not be performed, Windows '95 limits exceeded.

Error <24> : RTE (24) - Screen <x> is already open.

Error <25> : RTE (25) - Screen dimensions are too big or too small.

Error <26> : RTE (26) - ** NOT SUPPORTED ** Incorrect or unsupported screen resolution, must be "640x480", "320x200", "800x600", "1024x768", "640x400" or "320x240".' ; ;

Error <27> : RTE (27) - Screen number <x> is invalid!

Error <28> : RTE (28) - Screen x is not open.

Error <29> : RTE (29) - Could not clone current screen, clone destination screen already exists (x).

Error <30> : RTE (30) - Could not clone current screen, all available screens are used.

Error <31> : RTE (31) - Could not change ZOrder of screen (x). Screen is not open.

Error <32> : RTE (32) - Could not change the current screen to (x), this screen is not open.

Error <33> : RTE (33) - Could not change screen location, screen (x) is not open.

Error <34> : RTE (34) - Could not change screen dimensions, screen (x) is not open.

Error <35> : RTE (35) - COS, invalid value passed.

Error <36> : RTE (36) - SIN, invalid value passed.

Error <37> : RTE (37) - Invalid parameters passed to string function.

Error <38> : RTE (38) - Invalid parameters passed to maths function.

Error <39> : RTE (39) - ADD Command failed, could not find variable <x>.

Error <40> : RTE (40) - SUB Command failed, could not find variable <x>.

Error <41> : RTE (41) - MAX function failed, value passed were invalid and/or not of matching type

Error <42> : RTE (42) - MIN function failed, value passed were invalid and/or not of matching type

Error <43> : RTE (43) - SGN function failed, value passed was not valid and/or not of the correct type

Error <44> : RTE (44) - INT function failed, value passed was not valid and/or not of the correct type

Error <45> : RTE (45) - SQR function failed, value passed was not valid and/or not of the correct type

Error <46> : RTE (46) - EXP function failed, value passed was not valid and/or not of the correct type

Error <47> : RTE (47) - LOG function failed, value passed was not valid and/or not of the correct type

Error <48> : RTE (48) - LN function failed, value passed was not valid and/or not of the correct type

Error <49> : RTE (49) - ACOS, invalid value passed.

Error <50> : RTE (50) - HCOS, invalid value passed.

Error <51> : RTE (51) - ASIN, invalid value passed.

Error <52> : RTE (52) - HSIN, invalid value passed.

Error <53> : RTE (53) - ATAN function failed, value passed was not valid and/or not of the correct type

Error <54> : RTE (54) - HTAN function failed, value passed was not valid and/or not of the correct type

Error <55> : RTE (55) - Invalid parameter passed to FONT command. Value passed was no valid and/or not of the correct type

Error <56> : RTE (56) - FONT DMODE command failed, invalid value passed

Error <57> : RTE (57) - Invalid parameter passed to command, excepted ON or OFF.

Error <58> : RTE (58) - SET CURS failed. One or more parameters are not simple integer values - Expressions are not allowed.

Error <59> : RTE (59) - Invalid integer value passed with preformatting commands. You have specified a invalid integer value, after a control character in a preformatted string.

Error <60> : RTE (60) - Invalid value passed to preformatting command. You have specified an invalid value, required by a preformatting command. Please check your expression and try again.

Error <61> : RTE (61) - SET TAB failed, invalid integer passed to command. Must be in range 0 - 80.

Error <62> : RTE (62) - Invalid variable name suggested to FOR Command, please verify the variable name used!

Error <63> : RTE (63) - Invalid operator in FOR command.

Error <64> : RTE (64) - Could find no terminating NEXT command for FOR command

Error <65> : RTE (65) - The Logical Operator requires it to have boolean (true/false) values on either side of it.

Error <66> : RTE (66) - Invalid Condition, Expression must return either boolean True or False.

Error <67> : RTE (67) - GOTO Failed, could not locate the label requested.

Error <68> : RTE (68) - DATETOSTR Failed, invalid number passed to function. Please verify a valid real number was passed to this function.

Error <69> : RTE (69) - TIMETOSTR Failed, invalid number passed to function. Please verify a valid real number was passed to this function.

Error <70> : RTE (70) - DATETOSTR Failed, invalid paramaters passed to function. Please verify you have passed a valid real number.

Error <71> : RTE (71) - TIMETOSTR Failed, invalid paramaters passed to function. Please verify you have passed a valid real number.

Error <72> : RTE (72) - FORMATDATETIME Failed, invalid parameters passed. Please verify parameters passed are a string and real number.

Error <73> : RTE (73) - FORMATDATETIME Failed, invalid Parameters passed, could not generate a valid string.

Error <74> : RTE (74) - STRTODATE Failed, invalid string passed to function.

Error <75> : RTE (75) - STRTODATE Failed, String contained no valid date.

Error <76> : RTE (76) - STRTOTIME Failed, invalid string passed to function.

Error <77> : RTE (77) - STRTOTIME Failed, String contained no valid time.

Error <78> : RTE (78) - CALL Command Failed, could not find procedure xxxx

Error <79> : RTE (79) - CALL Command Failed, out of return stack space.

Error <80> : RTE (80) - Cannot call Keyboard based command or function, Keyboard service is not running - Call "KEYBOARD ON" command.

Error <81> : RTE (81) - Expected single integer value passed to function.

Error <82> : RTE (82) - Expected single integer value/expression passed to function, parameter : <no>

Error <83> : RTE (83) - Expected single string value/expression passed to function, parameter : <no>

Error <84> : RTE (84) - Expected single simple integer value passed to function, parameter : <no>

Error <85> : RTE (85) - Expected single simple string value passed to function, parameter : <no>

Error <86> : RTE (86) - Expected a name of a valid <type> passed to function. Could not find file in parameter : <no>

Error <87> : RTE (87) - Out Of Sprite Space, too many sprite currently on screen.

Error <88> : RTE (88) - Out Of Sprite Image Space, too many sprite images cannot create another.

Error <89> : RTE (89) - Cannot create this sprite, sprite number is already created and in use.

Error <90> : RTE (90) - Call failed, sprite has not been created.

Error <91> : RTE (91) - Could not set sprite frame number, value is out of range.