



WIMPBASIC 2

BASIC Reference Manual

ProAction

3QD
DEVELOPMENTS LTD

APDL

WIMPBASIC 2

© 3QD Developments Ltd 2026
All rights reserved.

No part of this product may be reproduced in whole or part by any means without written permission of the publisher. Unauthorised hiring, renting, lending, public performance or broadcasting of this product or its parts is prohibited.

While every care is taken, the publisher cannot be held responsible for any errors in this product, or for loss of data or consequential effects from the use of this package.

Software and Documentation supplied by
APDL
a part of
3QD Developments Ltd
Single User version downloaded from the free of charge software section of
www.riscos.com

Please note that any free of charge downloads from RISCOS.com are supplied without any technical support package as per the licence agreement.

This document last edited on 29/09/26



If you have found this application and this manual useful please consider making a donation to help cover the costs of cataloguing, documenting, updating and hosting the APDL applications. Scanning the QR code will take you to our donations page.

Contents

General	5	FontStringWidth(	15
Graphics	5	Foreground	15
Omitted keywords	5	Format(	15
Formatting Strings	5	FormatFloat	15
Units	7	FormatInt	16
		FormatString	16
WimpBasic components	8	Free	16
Bar	8	GetDrawMemory(	16
: Colon	8	GetSpriteMemory(	16
		GetIconHandle(	16
WimpBasic Commands	9	GetSize(	16
Allocate(	9	GetWindowHandle(	16
AndNot	9	Goto	16
Ask(	9	Hex\$(	16
Background	9	Hide	16
CaretByHandle	9	HourGlass	16
CaretIcon	9	IconBackgroundColour	17
CaretIconHandle(	9	IconBColour	17
CaretWindow	10	IconForegroundColour	17
CaretWindowHandle(	10	IconFColour	17
Case	10	IconHide	17
Circle	10	IconIndex	17
Close	10	IconMove	17
CloseMenu	10	IconShow	17
Cls	10	If	17
Data	10	InputFocus	17
DefFn	11	Instr(	17
DefProc	11	Invert	17
Dim	11	IsFaded(	17
DirReadBoth(	11	IsOpen(	18
DirReadDir(	11	IsSelected	18
DirReadFile(	12	Line	18
DirSet(	12	LoadFailed	18
DragBoundary	12	LoadOK	18
DragBox	12	LoadType	18
DragPoint	12	Local	18
DragRubberBox	12	MenuChangeEntry	19
Draw	12	MenuChangeTitle	19
DrawAction	12	MenuEmpty	19
DrawLoad(	13	MenuExtend	19
DrawMode	13	MenuEmpty	19
DrawPlot	13	MenuIsFaded(	19
DrawRemove	13	MenuIsTicked(	19
Error	13	MenuTickEntry	19
FileCreateDirectory	13	Mode	19
FileGetFileType	14	Modified	19
FileLoad	14	Mouse	19
FileSave	14	MouseButton	20
FileSetFileType	14	MouseWindow(	20
FileTypeNumber	14	MouseX(	20
Fill	14	MouseY(	20
Filled	14	Move	20
Font	14	Next	20
Fonts(	14	None	20
FontColour	14	On Error Goto	20
FontList	15	Open	20
FontMenu	15	OpenAt	20
FontPresent(	15	OpenCentred	20
FontSelected(	15	OpenChild	20
FontStringHeight(	15	OpenFixedPane	20

OpenInsideBottom	21	Str\$(	27
OpenInsideLeft	21	Tab(	27
OpenInsideRight	21	Update	27
OpenInsideTop	21	When	27
OpenMenu	21	WindowIndex	27
OpenPane	21	WindowSetExtent	28
Origin	21	WindowSetTitle	28
OrNot	21	WorkAreaXToScreen(	28
Oscli	21	WorkAreaYToScreen(	28
OpenInsideRight	21		
OpenInsideTop	21	Appendix A	29
OpenMenu	21	WimpBasic Compiler Errors	29
OpenPane	21		
Origin	21	Appendix B	33
OrNot	21	WimpBasic RunTime Errors	33
Oscli	21		
Otherwise	22		
Outline	22		
Output	22		
PaneBottomLeft	22		
PaneBottomRight	22		
PaneTopLeft	22		
PaneTopRight	22		
Plot	22		
Point	22		
Pointer	22		
Print	22		
Print#	23		
Printer	23		
PrintPage	23		
Read	23		
ReadByte(	23		
ReadFloat(	23		
ReadInt(	23		
ReadString(	23		
Rectangle	24		
Refresh	24		
ReSize(	24		
Restore	24		
Resume	24		
Retry	24		
Return	24		
Rnd(	24		
Save	24		
SaveFailed	24		
SaveOK	24		
Screen	25		
ScreenXToWorkArea(	25		
ScreenYToWorkArea(	25		
SendMessage	25		
SetFade	25		
SetScrollX	25		
SetScrollY	25		
SetSelect	25		
SetSprite	26		
SizeOfFloat(	26		
SpritePutAt	26		
SpritePutAtScaled	26		
Store	26		
StoreByte	26		
StoreFloat	26		
StoreInt	27		
StoreString	27		

General

The WimpBasic language is similar to BBC Basic but there are a number of differences.

- Everything in WimpBasic is case insensitive ie

a\$ is the same as A\$

PROCDEMO is the same as PROCdemo or PROCDemo

- DefProcs and Procs must be immediately followed by the procedure name ie no spaces between the Proc and the name eg ProcXYZ
- WimpBasic does not accept multi line statements. Each statement must be on its own line.

Important: Windows and menus plotted in a procedure do not appear until the *EndProc* is reached. If you open something like a warning box and you want to close it after a period of time then the delay must be done in a separate procedure. See the *Warn* program in the tutorials for more information.

Graphics

There are differences between BBC Basic's graphics statements and the WimpBasic graphics statements. The graphics functions and Print are affected by the Output statement (see the *Keywords* section). Output simply controls to which 'device' the output goes. There are three possibilities.

- Screen: This is like the normal plot, print etc.
- Window: Plot and Print actions are 'cached' for the window and automatically redrawn when the window gets a redraw request or a Refresh Update command is issued.
- Printer: Output is cached until the PrintPage statement is used, then it is sent to the printer.

Coordinates are given in screen coordinates if the output is to Screen but in window coordinates if the output is to a window or the printer.

Omitted keywords

Although most BBC Basic keywords are mirrored in WimpBasic the following Keywords, Functions and Statements are not available.

!	?	I	AUTO	APPEND
CALL	CHAIN	CLEAR	CLG	COLOUR
COUNT	CRUNCH	COLOUR	DELETE	EDIT
END	ENVELOPE	EVAL	ERL	GET
GOSUB	GCOL	HIMEM	HELP	INPUT
INKEY\$	INKEY	INSTALL	LIST	LOAD
LOMEM	LET	LIBRARY	LVAR	NEW
OLD	ON	OFF	OVERLAY	PAGE
POINT(	POS	RUN	RENUMBER	SPC
STOP	TEXTLOAD	TEXTSAVE	TINT	TRACE
TWINO	TWIN	USR	VPOS	WIDTH

Formatting Strings

Description of the format string.

Format string:

[L<number>/R<number>/C<number>] [+/-]

L Means *Left justified*. The number is the field width.

R Means *Right justified*. The number is the field width.

C Means *Centre justified*. The number is the field width.

+/- stands for + or -. If you use + the relevant sign will always be shown. If you use - only negative signs will be shown. If negative is used a space will then be used if the number is positive. The sign generated by this is placed after the number and is additional to the sign placed before the number. In most cases it is not necessary to use this item.

A design file called **FormString** is available on the WimpBasic web site and this lets you play with format strings and see the results.

N<number>[dp<number2>]/E<number1>[dp][<number2>]/F<string> [+/-]

N stands for *Number*. The number following it is the number of digits to show before the decimal point. The decimal point can be a comma (,) or a full stop (.). The second number is the number of digits to display after the decimal point. If no value is given then no figures are given after the decimal point.

- E stands for *Exponent*. The number following it is the number of digits to show (4 = 1.000E). The second number is the number of digits to show in the exponent. The decimal point controls the number of digits to display after the decimal point.
- F stands for *Format*. In Format you give a Format picture. Two characters are used; # means always display character; @ means display the character unless it is 0, when a space is displayed.
- +/- stands for + or -. If you use + the relevant sign will always be shown. If you use - only negative signs will be shown. If negative is used a space will then be used if the number is positive. The sign generated by this is placed after the number and is additional to the sign placed before the number. In most cases it is not necessary to use this item.

Examples:

The examples below assume that you have first set the Format string and then Printed the relevant string eg L20 and "abc" means

```
FormatString "L20"
Rem sets format to be 20 characters and left justified
Print "abc"
Rem prints the characters 'abc' followed by 17 spaces
```

with the result "abc"
The characters 'abc' are printed at the left of the string followed by 17 spaces.

Example 1

R20 and "abc" gives " abc"
The characters 'abc' are printed at the right of the string preceded by 17 spaces.

Example 2

C20 and "abc" gives " abc "
The characters 'abc' are printed in the centre of the field of 20 characters with relevant spaces either side.

Example 3

N8 and 31200.38 gives "31200"
The Format string says 'print up to 8 digits before the decimal point'. As there are only five digits this is all that is printed.

Example 4

N8,0 and 31200.38 gives "31200."
This time up to 8 characters are printed before the decimal point but no characters following it so just the decimal point is shown.

Example 5

N8,3 and 31200.38 gives "31200.380"
This time the format string asks for 3 decimal places so the string is padded out.

Example 6

E4.2 and 100 gives "1.000E02"
Where E means use exponents; 4 means use 4 digits; .2 means use two digits in the exponent.

Example 7

FABC###A and "yes" gives "ABCyesA"
The # characters are replaced in the Format picture by the string 'yes'.

Units

The units used in this manual for fonts are Points except for **FontStringHeight** and **FontStringWidth**, which are in OS coordinates.

The units used in windows are OS coordinates and the origin is top left. In this case bottom right is dependent upon the window size but the X value is positive and the Y value is negative eg 100,-100.

The units used for the screen are OS coordinates and the origin is bottom left. So bottom left is 0,0 and top right is *Screen:Width%*, *Screen:Height%*. These are system variables built into WimpBASIC.

WimpBasic components

This section lists all the keywords, statements, functions, and other components that are used in WimpBasic in alphabetical order.

Keywords that are identical to BBC BASIC are not highlighted. As the use and syntax of these keywords would be exactly as BBC Basic no further explanation is given and you should refer to the Basic User Guide.

All keywords which either do not appear in BBC Basic or which have a different syntax are in bold type and are followed by explanatory text with full syntax and examples where appropriats.

(Open bracket
) Close bracket
* Multiply
+ Plus
- Minus
/ Divide
^ Power of

| Bar Not used in WimpBasic Use ReadFloat and StoreFloat instead.

: Colon The colon is not used to separate statements in a line, as WimpBasic only allows one statement per line.

The colon has two functions. The first is to separate groups and variables. All global variables belong to a group and to use them you enter them as *Group:Variable*.

Example Screen:Height%

The second use for the colon is to define a label for a **Goto** or **Restore** statement. To do this the colon must be the first character on the line and followed by a valid label. Labels have the same rules as variable names except that (,), % and \$ are not allowed.

Example :LowMemory

```
REM Some code to handle low memory
On Error Goto LowMemory
```

< Less than
<< Shift left
= Equals
> Greater than
>> Arithmetic Shift right
>>> Logical Shift right

WimpBasic Commands

Abs(
Acs(
Adval(

Allocate(Used to allocate memory. Same as Dim n% 1000 in BBC Basic.

Syntax: n%=Allocate(size%)

And

AndNot See DrawAction for details.

Asc(

Ask(Opens a query window.

Syntax: answer%=Ask(OkButton%,CancelButton%,Message\$)

If *OkButton%* is True an OK button is displayed.

If *CancelButton%* is True a Cancel button is displayed. *Message\$* is displayed in the window.

Returns: 0 for Escape, 1 for OK and 2 for Cancel.

Asn(

Atn(

Background Background is used to set the background colour using RGB values. This has no immediate effect on the background, but when you next clear the background it is set to this colour.

The Output statement affects the behaviour of Background.

See the *Graphics* text for information.

Example: Background 255,0,0 Sets background to red

Beat

Beats

Bget#

Bput#

By

CaretByHandle Set location of the caret.

Syntax: CaretByHandle wimphandle%,iconnumber%,index%,height%

Where: *wimphandle%* is the wimp window handle, *iconnumber%* the icon number, *index%* the index into the string and *height%* the caret height (in OS units). The usual use is to use -1 for the height, which means it will be calculated to fit in the icon.

CaretIcon Put the caret in an icon.

Syntax: CaretIcon window:icon\$, index%, height%

Where: *window:icon\$* is the icon, *index%* the index into the string and *height%* the caret's height (in OS units). It is usual to use -1 for the height, which means it will be calculated to fit the icon.

CaretIconHandle(Reads the number of the icon containing the caret.

Syntax icon%=CaretIconHandle()

CaretWindow Put caret in window
Syntax: `CaretWindow window, x%, y%, height%`
Where: *window* is the window name, *x%,y%* are the location inside the window, *height%* is the caret's height.

CaretWindowHandle() Read the window handle of the window containing the caret.
Syntax: `windowhandle%=CaretWindowHandle()`

Case Different to Basic. Unlike BBC Basic, the action for each **When** must start on a new line eg

```
Case Day% of
  When 0
    day$="Sunday"
  When 1
    day$="Monday"
```

Chr\$(

Circle Same as Basic except for the effect of the **Output** keyword.

Close Use this to close and delete the window.
Syntax `Close WindowName`
All variables in the window are lost when the window is closed. If you want to close a window without losing the variables, use the **Hide** statement.

Close#

CloseMenu Closes the current menu.
Note that the menu will be closed even if your application doesn't 'own' it.

Cls Same as Basic except for the effect of the **Output** keyword. **Count** is not supported. See Graphics for details.

Cos(

Data Data behaves a bit different to BBC Basic. Expressions are not allowed (eg DATA 10*a) and each procedure/function has its own local data pointer. Data can only be read by the procedure/function that the Data is defined in. If you need global Data similar to BBC Basic here is how to emulate it:

```
DefFnReadGlobalData%(Reset%)
  Rem Called to read from 'global' Data.
  Rem If Reset% is TRUE a restore is performed to
  Rem move the data read pointer to the beginning.
  Rem If restore is performed value returned is
  Rem random otherwise returns next data value.
  Local Value%
  If 0 Reset% Then
    Restore DataStart
  Else
    Read Value%
  Endif
  :DataStart
  DATA "Fred", "Bloggs"
  =Value%
```

Instead of the normal:
`Read MyValue%`

you would then use
`MyValue%=FNReadGlobalData%(False)`

and instead of:

`Restore 0`

or similar you would Use

`MyValue%=FNReadGlobalData%(True)`

DefFn

DefFn is a single keyword and must have no spaces between itself and the name. In WimpBasic the function name should include a type character such as '%' or '\$'. This allows the compiler to check that you are using the right variable type and thus it is less likely you will get a run time error as in BBC Basic.

Note: Windows and menus plotted in a function do not appear until the end of the function. If you open something like a warning box and want to close it after a delay then the delay must be done in a separate procedure. See the **Warn** file in the tutorials for more information.

DefProc

DefProc is a single keyword and must have no spaces between itself and the name.

Note: Windows and menus plotted in a procedure do not appear until EndProc is reached. If you open a warning box and want to close it after a delay the delay must be done in a separate procedure. See the **Warn** file in Tutorials for more information.

Deg(

Dim

To allocate space i.e. *Dim n% 1000* you now use the **Allocate** function. Dim is only used for arrays in WimpBasic.

Dim(

DirReadBoth(

Read file and directory names from directory set by **DirSet**

Syntax: `file$=DirReadBoth()`
Example: `Dim objects$(100)
DirSet("ram:$")
object%=-1
Repeat
 object%+=1
 objects$(object%)=DirReadBoth()
Until objects$(object%)=""`

This will read the names of all files and directories in the '*Ram:\$*' directory into the array *objects\$*

DirReadDir(

Read directory names only from the directory set by **DirSet()**

Syntax: `dir$=DirReadFile()`
Example: `Dim dir$(100)
DirSet("ram:$")
dir%=-1
Repeat
 dir%+=1
 dir$(dir%)=DirReadDir()
Until dir$(dir%)=""`

Reads the names of all directories in '*Ram:\$*' into the array *dir\$*

DirReadFile() Read file names only, from the directory set by **DirSet()**

Syntax: `file$=DirReadFile()`

Example:

```
Dim files$(100)
DirSet("ram:$")
file%=-1
Repeat
  file%+=1
  files$(file%)=DirReadFiles()
Until files$(file%)=""
```

This will read the names of all files in the '*Ram:\$*' directory into the array *files\$*

DirSet() Sets the directory from which you want to read.

Syntax: `DirSet(Directory$)`

This also resets the 'pointer' used by **DirRead** so next **DirRead** will return first file in directory. Does not change working directory.

Div

DragBoundary Used to set the boundary for a drag action.

Syntax: `DragBoundary x0%,y0%,x1%,y1%`

Where: *x0%,y0%* is the lower left and *x1%,y1%* is top right. The coordinates given are screen coordinates.

DragBox Used to drag a box.

Syntax: `DragBox x0%,y0%,x1%,y1%`

Where: *x0%,y0%* is the lower left and *x1%,y1%* is top right. The coordinates given are screen coordinates.

DragPoint Used to drag a point when there is no box or any visible indication of the dragging. Normally, programs change the pointer during a drag like this.

Syntax: `DragPoint`

DragRubberBox Used to drag a size box.

Syntax: `DragRubberBox x0%,y0%,x1%,y1%`

Where: The *x0%,y0%* corner is fixed but the *x1%,y1%* corner will move with the mouse pointer.

Draw Same as Basic except for the **Output** statement. See **Graphics** text for details.

DrawAction Sets the current graphics action like GCOL in Basic.

Syntax: `DrawAction Action`

Where: *Action* can be any one of: *Store, And, AndNot, Eor, Invert, None, Or, OrNot*. See the BBC Basic manual for an explanation of these operators

DrawLoad(	Loads a Draw file into memory.
Syntax:	a%=DrawLoad(fileName\$)
Where:	a% is a handle to the Drawfile that you will use for DrawPlot , DrawRemove and GetDrawMemory .
DrawMode	Affects how some of the graphics statements work.
Syntax:	DrawMode Action
Where:	Action can be either Outline or Filled . This has effect on all graphics keywords where FILL was used in BBC Basic.
Example:	<pre> DrawMode Outline Circle 640,512,200 Rem This draws a circle outline DrawMode Filled Circle 640,512,300 Rem This draws a filled circle </pre>
DrawPlot	Plot a Draw file at a given coordinate. Output depends on the Output statement.
Syntax:	DrawPlot a%, x%, y%, scx%, scy%
Where:	a% is the the Drawfile handle obtained from DrawLoad(, x%, y% is the location of the :bottom left corner and scx%, scy% are the scale values.
DrawRemove	Removes a Draw file from memory.
Syntax:	DrawRemove a%
Where:	a% is the Drawfile handle as obtained from DrawLoad(
Note:	If you have used DrawPlot to plot a Drawfile in a window you must either do a Cls on that window or Close the window. You should do this after DrawRemove otherwise the program will attempt to redraw the file after it has been deleted and, because it is no longer there, a crash is the likely outcome.
Ellipse	
Else	
EndCase	
EndIf	
EndProc	
EndWhile	
Eof#	
Eor	
Err	
Error	There is no ERROR EXT in WimpBasic otherwise Error works as in BBC Basic.
Exp(	
Ext#	
False	
FileCreateDirectory	Creates a directory.
Syntax:	FileCreateDirectory "PathAndDirectoryName\$"
Where:	PathAndDirectoryName\$ is the full pathname of where you want the directory to be created.
Example:	<pre> DefProcMakeDirlocal a\$ Rem create a directory for the files FileCreateDirectory "RAM::RamDisc0.\$.TEST" EndProc </pre>

FileGetFileType	Reads the filetype of a given file.
Syntax:	<code>FileType%=FileGetFileType (FileName\$)</code>
FileLoad	Used to load a file into memory. Memory must have been allocated before using this statement.
Syntax:	<code>FileLoad FileName\$,MemoryLocation%</code>
Where:	<i>FileName\$</i> is the name of the file, including the full pathname, and <i>MemoryLocation%</i> is the address in memory to load the file to. You must make sure that the file isn't too long to be loaded at the address given without overwriting other data or program space. (For example, by using the Ext# function)
FileSave	Saves an area of memory as a file.
Syntax:	<code>FileSave FileName\$,MemoryLocation%,Size%</code>
Where:	<i>FileName\$</i> is the full pathname of the filename required, <i>MemoryLocation%</i> is the memory address for the start of the file and <i>Size%</i> is the length of the file in bytes.
FileSetFileType	Changes a file to a specific filetype.
Syntax:	<code>FileSetFileType (FileName\$,FileType%)</code>
Where:	<i>FileName\$</i> is the full pathname of the file and <i>FileType%</i> is the numeric filetype.
FileTypeNumber	Changes a textual filetype into a filetype number.
Syntax:	<code>DataFileType%=FileTypeNumber ("Data")</code>
Fill	Works like BBC Basic but not very useful in the Wimp environment as windows are usually partially drawn at each redraw event. The Output keyword also has an effect on the Fill statement. See Graphics for details. Fill should be avoided if at all possible.
Filled	See DrawMode for details.
Font	Select a specific font and its size.
Syntax:	<code>Font "Font name", Width%, Height%</code>
Where:	<i>"Font name"</i> is the font name, <i>Width%</i> is the character width% and <i>Height%</i> is the character height.
Fonts(	Read number of fonts available
Syntax:	<code>NumberOfFonts%=Fonts ()</code>
FontColour	Sets a font's colour.
Syntax:	<code>FontColour fr%, fg%, fb%, br%, bg%, bb%</code>
Where:	<i>fr%</i> =foreground red, <i>fg%</i> = foreground green, <i>fb%</i> =foreground blue, <i>br%</i> =background red, <i>bg%</i> =background green, <i>bb%</i> =background blue. The range is 0 to 255 with 0 no colour and 255 full colour. eg 0,0,0 = black, 255,255,255 = white, 255,0,0 = full intensity red

FontList	Reads the names of available fonts.
Syntax:	FontList Fonts\$()
Where:	<i>Fonts\$()</i> is an array previously dimensioned.
Example:	<pre>Rem We use Fonts() to read how many fonts Rem there are and create an array with an Rem entry for each font name. Dim Fonts\$ (Fonts()) Rem Read the font names into Fonts\$() FontList Fonts\$()</pre>
FontMenu	Creates a font menu with all available fonts.
Syntax:	FontMenu Menu, System%
Where:	<i>Menu</i> is the name of a menu created in the menu designer. If <i>System%</i> is <i>True</i> then the first entry in the menu will say 'System' for system font. The Menu should always be created with an entry for System font so that this command can toggle it on or off.
FontPresent(	Checks if the named font is installed.
Syntax:	is_it%=FontPresent("Font Name")
Where:	<i>"Font Name"</i> is the font you want to check for. If the font is present then <i>is_it%</i> will be <i>True</i> (-1).
FontSelected(	Returns the font name after the user has clicked on an entry in the font menu.
Syntax:	A%=FontSelected(MenuEntrys%, MenuLevel%)
Where:	<i>MenuEntrys%</i> is the parameter to the Menu selection function (see the Menu Designer). <i>MenuLevel%</i> is the submenu level where the font menu starts. If for example the font menu starts as submenu of the menu that the selection was made from, then <i>MenuLevel</i> is 2.
FontStringHeight(	Read height of a font string, in OS coordinates.
Syntax:	FontStringHeight("String")
Where:	<i>"String"</i> is the string you want to check.
FontStringWidth(	Read width of a font string, in OS coordinates.
Syntax:	FontStringWidth("String")
Where:	<i>"String"</i> is the string you want to check.
For	
Foreground	Used to set the foreground colour.
Syntax:	Foreground R%,G%,B%
Where:	<i>R%</i> is Red, <i>G%</i> Green and <i>B%</i> blue. The range is from 0 to 255. The Output keyword affects the Foreground statement.
Format(	Formats an expression.
Syntax:	Format(FormatString\$,Expression)
Where:	Expression can be integer, floating point or string. See Formatting Strings for full information.
FormatFloat	Sets the format string to use when formatting floating point numbers. This controls how floating point numbers are displayed in icons and by the Print statement.
Syntax:	FormatFloat <Format string> See Formatting Strings for full information.

FormatInt	Sets the format string to use when formatting integer values. This controls how integer values are displayed in icons and by the Print statement.
Syntax:	FormatInt <Format string> See Formatting Strings for full information.
FormatString	Sets the format string to use when formatting strings. Controls string display in icons and in the Print statement
Syntax:	FormatString <Format string> See Formatting Strings for full information.
Free	Used to free memory assigned by Allocate .
Syntax:	Free n%
Where:	n% has been previously allocated eg n%=Allocate(8192)
Note:	If you allocate memory in a local variable you must free it yourself. WimpBasic will not do it automatically.
Get\$	
GetDrawMemory(	Gets the memory location of a Drawfile.
Syntax:	address%=GetDrawMemory(DrawHandle%)
Where:	DrawHandle% is the handle returned by DrawLoad
GetSpriteMemory(	Gets the memory location of the sprite area.
Syntax:	address%=GetSpriteMemory()
Where:	address% is the address of the sprite area.
GetIconHandle(	Reads an icon number.
Syntax:	iconnumber%=GetIconHandle(Window:Icon\$)
Where:	Window:Icon\$ is the name of the icon whose handle you require. It's number is returned in iconnumber%.
GetSize(	Reads the size of an allocated block.
Syntax:	size%=GetSize(Block%)
Where:	Block% had previously been created using Allocate .
GetWindowHandle(	Used to read a window's wimp handle.
Syntax:	wimphandle%=GetWindowHandle(Window)
Where:	Window is the name of the window.
Goto	Goto is the same as in BBC Basic but uses labels instead of line numbers. Goto can only jump inside a procedure or function. See Colon (: for details on how to define a label.
Hex\$(	Formats an integer value into a hex number. Same as STR\$ in BBC Basic.
Example:	Hex\$(255){f0 } produces the result FF
Hide	Closes a window without losing the variables.
Syntax:	Hide WindowName
HourGlass	Displays the hourglass pointer during time consuming actions
Syntax:	HourGlass On HourGlass Off HourGlass n%
Where:	HourGlass On and HourGlass Off turn the hourglass pointer on/off. HourGlass n% will display a percentage under the hourglass pointer. n% should be in the range 0-100.

IconBackgroundColour	Change Icon background colour.
Syntax:	IconBackgroundColour Window:Icon\$, WimpColourNr%
Where:	<i>Window:Icon\$</i> is the icon and <i>WimpColourNumber%</i> is the wimp colour number (1 to 15).
Note:	IconBColour also works.
IconForegroundColour	Change Icon foreground colour.
Syntax:	IconForegroundColour Window:Icon\$, WimpColourNr%
Where:	<i>Window:Icon\$</i> is the icon and <i>WimpColourNumber%</i> is the wimp colour number (1 to 15).
Note:	IconFColour also works.
IconHide	Hide Icon. To show icon use IconShow .
Syntax:	IconHide Window:Icon\$
Where:	<i>Window:Icon\$</i> is the Icon to hide.
IconIndex	This is the index number of an icon in an array that has been selected. It is used when you have created an array of icons in a window, all having the same Click procedure. IconIndex tells you which of the icons in the array has been selected.
Syntax:	a\$=Window:Icon\$(IconIndex)
Where:	a\$ contains the contents of the icon that has been clicked on, <i>Window:Icon\$</i> is the name of an array of icons.
Example:	<pre>DefProcClickProcedure Window:Result\$=Window:Icon\$(IconIndex) EndProc</pre>
IconMove	Move Icon.
Syntax:	IconMove Window:Icon\$, x0%, y0%
Where:	<i>Window:Icon\$</i> is the icon to move, x0% and y0% are the x and y window co-ordinates.
IconShow	Show hidden icon. To hide icon use IconHide .
Syntax:	IconShow Window:Icon\$, x0%, y0%
Where:	<i>Window:Icon\$</i> is the icon to show, x0% and y0% are the x and y window co-ordinates.
If	Same as BBC Basic, except that the Then keyword is compulsory .
Input#	
InputFocus	Claims the input focus for a specified window and icon. This means that any input is directed to that window.
Syntax:	InputFocus WindowName,Icon name
Where:	<i>Window name</i> is the name of the window and <i>Icon name</i> is the full name of the icon.
Example:	InputFocus Main,Main:Guess\$
Instr(	Works like BBC Basic but third parameter (start position in the string) is compulsory.
Int(	
Invert	See DrawAction for details.
IsFaded(	Used to check if an icon is faded.
Syntax:	state%=IsFaded(Window:Icon\$)
Where:	<i>state%</i> is set to <i>True</i> if the icon is faded.

IsOpen(	Checks if a window is open ie visible.
Syntax:	<code>bool%=IsOpen(WindowName)</code>
Where:	<i>WindowName</i> is the name of the window and <i>bool%</i> is <i>True</i> for an open window and <i>False</i> for a hidden or closed window.
IsSelected(	Used to check if an icon is selected.
Syntax:	<code>state%=IsSelected(Window:Icon\$)</code>
Where:	<i>state%</i> is set to <i>True</i> if the icon is selected.
Left\$(	
Len(	
Let	
Line	Line does the same as in BBC Basic but is affected by the Output statement. See Graphics for details.
Ln(	
LoadFailed	Used in a load handler to tell WimpBasic that a load failed.
Syntax:	<code>LoadFailed</code>
Example:	<pre>DefProcLoadProcedure(Filename\$, FileType%) Rem This Procedure only loads text files. if FileType%\<\>Constant:TEXT then Loadfailed Else ProcDoLoad LoadOK Endif Endproc</pre>
LoadOK	Used to tell WimpBasic that a file was successfully loaded. You should check that the filename is not " <i><Wimp\$Scrap></i> ". If it is, you should remove it after loading. " <i><Wimp\$Scrap></i> " means the file has been loaded from an application rather than from disc. If your program displays the name of the file, or uses it as the name of the file to save later, don't use " <i><Wimp\$Scrap></i> ", use something like " <i>Unknown</i> " or " <i>Untitled</i> " instead.
	See the file FrameWork2 for an example of this.
Syntax:	<code>LoadOK</code>
Example:	<pre>DefProcLoadProcedure(Name\$, FileType%) Rem This Procedure only loads text files. If FileType%<>Constant:TEXT then Loadfailed Else ProcLoadIt LoadOK If Name\$="<Wimp\$Scrap>" then Oscli "remove <Wimp\$Scrap>" Endif Endproc</pre>
LoadType	Defines which data types the application will try to load.
Syntax:	<code>LoadType FileType%</code>
Example:	If for example <i>LoadType &FFF</i> is used in your Startup procedure then the Load procedure will be called whenever the user double clicks on a file with filetype &FFF. You can use LoadType more than once to handle more than one filetype. Normally you will use the FileTypes icon in the Main Window for selecting filetypes but this command may be useful if you want to alter the type of file accepted at different stages in a program.
Local	Works just like BBC Basic. However, in WimpBasic, all variables must be defined, either in

the variables section of the program, or as local, in the routine. When a routine exits, the local variables are lost.

Local variables cannot be accessed by sub procedures or functions.

Log(

MenuChangeEntry	Used to change a menu entry eg. <i>File''</i> is changed to <i>File '!WimpBasic'</i> .
Syntax:	<code>MenuChangeEntry MenuName,Entry%,"New text"</code>
Where:	<i>MenuName</i> is the name of the menu, <i>Entry%</i> the position of the entry in the menu (the first entry in a menu is number 1) and <i>"New text"</i> the new menu entry to replace the old one.
MenuChangeTitle	Used to change a menu title.
Syntax:	<code>MenuChangeTitle MenuName,"New title"</code>
Where:	<i>MenuName</i> is the name of the menu and <i>"New title"</i> is the new menu title to replace the old one.
MenuEmpty	Empties a menu. All entries in the menu are removed but the title is not modified.
Syntax:	<code>MenuEmpty MenuName</code>
MenuExtend	Extends a menu by adding an entry to it.
Syntax:	<code>MenuExtend MenuName,"Menu entry"</code>
Where:	<i>MenuName</i> is the name of the menu and <i>"Menu entry"</i> is the text for the new menu entry.
MenuEmpty	Empties a menu. All entries in the menu are removed but the title is not modified.
Syntax:	<code>MenuEmpty MenuName</code>
MenuIsFaded(	Checks if a menu entry is faded.
Syntax:	<code>bool%=MenuIsFaded(MenuName,Entry%)</code>
Where:	<i>bool%</i> is <i>True</i> if the entry is faded otherwise it is <i>False</i> .
MenuIsTicked(	Checks if a menu entry is ticked.
Syntax:	<code>bool%=MenuIsTicked(MenuName,Entry%)</code>
Where:	<i>bool%</i> is <i>True</i> if the entry is ticked otherwise it is <i>False</i> .
MenuTickEntry	Ticks an entry in a menu.
Syntax:	<code>MenuTickEntry MenuName,Entry%,bool%</code>
Where:	If <i>bool%</i> is <i>True</i> the entry is ticked otherwise it is unticked
Mid\$(	
Mod	
Mode	Same as in BBC Basic but uses WimpSetMode so all programs are informed about the mode change. programs should not use this. To read the Mode you are in use the global variable <i>Screen:Mode</i> .
Modified	Used to tell WimpBasic about an area modified in a window.
Syntax:	<code>Modified Window,x0%,y0%,x1%,y1%</code>
Where:	<i>Window</i> is the window name, <i>x%0,y%0</i> is the bottom left corner of the window and <i>x1%,y1%</i> is the top right corner.
Mouse	Mouse has only one form in WimpBasic.
Syntax:	<code>Mouse X%,Y%,Button%</code> This is identical to one of the mouse formats in Basic.

MouseButton	Used to read the mouse button state.
Syntax:	<code>Buttons%=MouseButton</code>
Where:	The values in <i>Buttons%</i> are as defined in your BASIC manual under the MOUSE function.
MouseWindow(	Checks if the mouse pointer is inside a window.
Syntax:	<code>bool%=MouseWindow(WindowName)</code>
Where:	<i>bool%</i> is <i>True</i> if the mouse pointer is inside the window otherwise it is <i>False</i> .
MouseX(	Reads the mouse X position in a window. Coordinate given is in window work coordinates.
Syntax:	<code>WorkX%=MouseX(WindowName)</code>
MouseY(	Reads the mouse Y position in a window. Coordinate given is in window work coordinates.
Syntax:	<code>WorkY%=MouseY(WindowName)</code>
Move	Same as Move in BBC Basic except that the Output statement has an affect on it. See Graphics for details.
Next	Works the same as BBC Basic but a variable name is required. A variable list eg (Next A,B,C) is not allowed.
None	See DrawAction for details.
Not Of	
On Error Goto	As in Basic except for the differences of WimpBasic Goto
Open	Opens a window. If it is already open it is moved to the front.
Syntax:	<code>Open WindowName</code>
Note:	Windows and menus do not appear until the <i>EndProc</i> is reached. If you open eg. a warning box, and want to close it after a delay the delay must be done in a separate procedure. See the Warn file in the Tutorials for more information.
OpenAt	Opens a window of the given size at the given position.
Syntax:	<code>OpenAt Window,x0%,y0%,x1%,y1%</code>
Where:	<i>Window</i> is the window name, <i>x0%.y0%</i> are the x and y coordinates of the bottom left of the window and <i>x1%,y1%</i> are the x and y coordinates of the top right.
Note:	Windows and menus do not appear until the <i>EndProc</i> is reached. If you open eg. a warning box, and want to close it after a delay the delay must be done in a separate procedure. See the <i>Warn</i> file in the Tutorials for more information.
OpenCentred	Opens the window at the centre of the screen.
Syntax:	<code>OpenCentred WindowName</code>
OpenChild	Opens a window as a child window of another window. This means that when the parent window is closed the child window is automatically closed also.
Syntax:	<code>OpenChild ParentWindow,WindowToOpen</code>
Where:	<i>ParentWindow</i> is the name of the Parent window and <i>WindowToOpen</i> the name of the window to open as a child.
OpenFixedPane	Opens a fixed pane window. OpenFixedPane opens pane windows that are always in a fixed position relative to their parent window. Pane windows opened with OpenPane can be removed from their parents by dragging them.
Syntax:	<code>OpenFixedPane ParentWin, PaneWin, x_offset, y_offset</code>
Where:	<i>ParentWin</i> is window to attach to and <i>PaneWin</i> is the window being opened and

x_offset, y_offset is the offset between the windows.

OpenIn(

OpenInsideBottom Opens a pane window attached to the inside bottom edge of a parent window. The pane is limited by the size of the parent window unlike **OpenPane** and **OpenFixedPane**, which extend beyond the parent window for use with things such as tool bars.

Syntax: `OpenInsideBottom ParentWin, PaneWin`

Where: *ParentWin* is the window to attach to and *PaneWin* is the window being opened.

OpenInsideLeft Opens a pane window attached to the inside left edge of a parent window. The pane is limited by the size of the parent unlike **OpenPane** and **OpenFixedPane**, which can extend beyond the parent for use with things such as tool bars.

Syntax: `OpenInsideLeft ParentWin, PaneWin`

Where: *ParentWin* is the window to attach to and *PaneWin* is the window being opened.

OpenInsideRight Opens a pane window attached to the inside right edge of a window. The pane is limited by the size of the parent window unlike **OpenPane** and **OpenFixedPane** which can extend beyond the parent window for use with things like tool bars.

Syntax: `OpenInsideRight ParentWin, PaneWin`

Where: *ParentWin* is the window to attach to and *PaneWin* is the window being opened.

OpenInsideTop Opens a pane window attached to the top inside of a parent window. The pane is limited by the size of the parent window unlike **OpenPane** and **OpenFixedPane**, which extend beyond the parent window for use with things such as tool bars

Syntax: `OpenInsideTop ParentWin, PaneWin`

Where: *ParentWin* is the window to attach to and *PaneWin* is the window being opened.

OpenMenu Opens a menu.

Syntax: `OpenMenu MenuName`

Note: Windows and menus do not appear until *EndProc* is reached. If you open something like a warning box and want to close it after a delay then the delay must be done in a separate procedure. See the **Warn** file in Tutorials for more information.

OpenOut(

OpenPane Opens a pane window. Pane windows opened with **OpenPane** can be removed from their parents by dragging them instead of the parent. **OpenFixedPane** opens pane windows that are always in a fixed position relative to their parent window.

Syntax: `OpenPane ParentWin, PaneWin, x_offset, y_offset`

Where: *ParentWin* is the window to attach to, *PaneWin* the pane to be opened, *x_offset*, *y_offset* the offset between windows.

OpenUp(
Or

Origin Works like in BBC Basic but is now affected by the **Output** statement. See **Graphics** for details.

OrNot See **DrawAction** for details.

Oscli The **OSCLI** statement is implemented with *WimpStartTask* instead of *OSCLI*; and registers are not set up as BBC Basic does.

Otherwise	The only difference between WimpBasic and Basic is that the statements to execute in Otherwise conditions can't follow the Otherwise statement as the colon (:) can't be used anymore to separate statements.
Example In Basic:	<pre>Case A% Of When 10: Print "10" Otherwise: Print "<>10" EndCase</pre> would have to be in WimpBasic: <pre>Case A% Of When 10 Print "10" Otherwise Print "<>10" EndCase</pre>
Outline	See DrawMode for details.
Output	Controls where the output for Print and Graphics statements goes to. See Graphics for details.
Syntax	Output WindowName
Where	WindowName is an existing window, the Screen or Printer
PaneBottomLeft	Lets you change the origin used when positioning panes opened with OpenPane and OpenFixedPane . After use of this command OpenPane and OpenFixedPane offsets are from the bottom left of the visible area of the parent window.
Syntax	PaneBottomLeft
PaneBottomRight	Lets you change the origin used when positioning panes opened with OpenPane and OpenFixedPane . After use of this command OpenPane and OpenFixedPane offsets are from the bottom right of the visible area of the parent window.
Syntax	PaneBottomRight
PaneTopLeft	Lets you change the origin used when positioning panes opened with OpenPane and OpenFixedPane . After use of this command OpenPane and OpenFixedPane offsets are from the top left of the visible area of the parent window.
Syntax	PaneTopLeft
PaneTopRight	Lets you change the origin used when positioning panes opened with OpenPane and OpenFixedPane . After use of this command OpenPane and OpenFixedPane offsets are from the top right of the visible area of the parent window.
Syntax	PaneTopRight
Pi	
Plot	Similar to BBC Basic but is affected by the Output statement. See Graphics for details.
Point	Similar to BBC Basic but is affected by the Output statement. See Graphics for details.
Pointer	Changes the mouse pointer.
Syntax:	Pointer "Pointer_Sprite_Name",xoffset%,yoffset%
Where:	<i>xoffset% and yoffset%</i> are the offsets in pixels of the active point. They are offsets from the top left. In most cases, they will be 0,0. Pointer sprites usually (and should) start with 'ptr' in their names. Some widely used sprites are 'ptrwrite','ptrmenu'. You can create new pointers or to overwrite existing pointers.
Print	Print has been modified quite a lot from BBC Basic. Output is controlled by the Output statement. See Graphics for details.

	The Output format is now controlled by the various Format statements instead of @% as in BBC Basic. Print in WimpBasic cannot be used for conversion to hex. Use the Hex\$ function instead. TAB cannot be used in the print line itself but can be used on its own to position the cursor prior to using Print. Example: To print 2 characters in and 3 lines down, you would use: <pre>TAB(2,3) Print"Hello"</pre> Parameters must be separated by commas.
Print#	Print# uses a different file format from BBC Basic for floating point numbers. Input# can read the old numbers but you cannot write them back using Print#. If you have data files using the BBC format and you want to use them with WimpBasic then you will have to write a conversion routine to use it with WimpBasic.
Printer	This is a modifier not a keyword. It directs output to the printer. See the Output statement and Graphics for details.
PrintPage	Prints a whole page
Syntax:	PrintPage SizeX%,SizeY%
Where:	SizeX%,SizeY% are the size you want the printed page to be This has no effect on printer page just on the scaling used.
Proc	
Ptr#	
Quit	
Rad(	
Read	The Read statement now only reads data from the procedure/function it is in. See Data for details.
ReadByte(	Reads a byte from memory.
Syntax:	value%=ReadByte(MemoryLocation%)
Where:	value% is the value of the byte read from the location given by <i>MemoryLocation%</i> .
Example:	value%=ReadByte(&8000) reads the first byte of application workspace, which starts at &8000 and places it in value%.
ReadFloat(	Reads the a floating point variable from memory.
Syntax:	value=ReadFloat(MemoryLocation%)
Where:	value is the value of the floating point variable read from the location given by <i>MemoryLocation%</i> .
	This should only be used when you know the memory location contains a floating point value.
ReadInt(	Reads the value of an integer variable from memory.
Syntax:	value%=ReadInt(MemoryLocation%)
Where:	value% is the value of the integer read from the location given by <i>MemoryLocation%</i> .
Example:	value%=ReadInt(&8000) reads the first byte of application workspace, which starts at &8000 and places it in value% as an integer.
ReadString(	Used to read a string from memory.
Syntax:	string\$=ReadString(MemoryLocation%)
Where:	string\$ contains the string read from the location given by <i>MemoryLocation%</i> .
Example:	string\$=ReadString(&8000) reads the string at &8000 and places it in string\$.

Rectangle	Works as in Basic but is now affected by the Output statement. See Graphics for details.
Refresh	Redraws a window and clears the background. Similar to Update , which does not clear the background. See further explanation in Graphics .
Syntax:	Refresh
Rem	
Repeat	
Report	
Report\$	
ReSize(	Used to change the size of an allocated block. Warning! The area can move when it is resized and so will cause problems if you are addressing the memory directly.
Syntax:	<code>newblock%=ReSize(oldblock%,size%)</code>
Where:	<i>newblock%</i> is the new address of the block, <i>oldblock%</i> is the old block address and <i>size%</i> is the new size for the block.
Example:	<code>oldblock%=Allocate(100)</code> <code>newblock%=ReSize(oldblock%,200)</code>
Restore	The Restore statement now only restores data read in the procedure/function it is in. See Data for details. It also uses labels instead of line numbers.
Syntax:	<code>Restore Label</code> If there is no label used Restore will restore to first Data statement in the procedure/function.
Resume	Starts execution again after an error. Execution is started in the line after the line that the error occurred in.
Syntax:	<code>Resume</code>
Retry	Same as Resume except the line with the error is executed again.
Syntax:	<code>Retry</code>
Return	As there is no Gosub statement in WimpBasic, Return only has an affect in procedure and function definitions to declare Return type parameters.
Right\$(	
Rnd(	A positive value returns a random integer value between 0 and the value given by <i>n%-1</i> . A negative value for <i>n%</i> sets the random seed.
Syntax:	<code>A%=Rnd(n%)</code>
Save	Used to save data.
Syntax:	<code>FileName\$=Save(Window:Icon\$,LeafName\$,FileSize%,FileType%)</code>
Where:	<i>Window:Icon\$</i> is the icon to drag. <i>LeafName\$</i> is the leafname (the writable icon in SaveAs window), <i>FileSize%</i> is size of the file and <i>FileType%</i> is the filetype to be used. You should attach a drag procedure to the file icon in your Saveas window. When the user drags and drops the icon, your procedure is called. In this procedure, you should call Save as shown above. See the tutorial file 'InOutDes' for an example.
SaveFailed	Used to tell WimpBasic that the program failed to save the data to the file whose name was returned by Save .
SaveOK	Tells WimpBasic that data has been saved to the file whose name is returned by Save . If this item is not used, WimpBasic will assume the save is OK unless SaveFailed is used. See the file Framework2 for an example.

Syntax:	SaveOK
Screen	This is a modifier not a keyword. It directs output to the screen. See the Output statement and Graphics for details.
ScreenXToWorkArea(	Converts X coordinate from screen coordinate to a window work coordinate (coordinate inside a window).
Syntax:	WorkX%=ScreenXToWorkArea(WindowName,ScreenX%)
Where:	WorkX% is the new window work coordinate, WindowName is the name of the window and ScreenX% is the original screen coordinate.
ScreenYToWorkArea(	Converts Y coordinate from screen coordinate to a window work coordinate (coordinate inside a window).
Syntax:	WorkY%=ScreenYToWorkArea(WindowName,ScreenY%)
Where:	WorkY% is the new window work coordinate, WindowName is the name of the window and ScreenY% is the original screen coordinate.
SendMessage	Calls a WimpSendMessage SWI. The Programmer's reference manual is likely to be needed for use of this statement. Those who know what it is, and need it, should use it. Others can cheerfully ignore it.
Syntax:	SendMessage EventNo%,DataBlock%,Handle%
SetFade	Fade or unfade an icon.
Syntax:	SetFade Window:Icon\$,Fade%
Where:	If Fade% is True the icon is faded otherwise it is unfaded.
SetScrollX	Sets position of X scroll bar in window.
Syntax:	SetScrollX Window,x%
Where:	Window is the name of the window and x% is the X scroll bar offset, relative to work space origin.
SetScrollY	Sets position of Y scroll bar in window.
Syntax:	SetScrollY Window,y%
Where:	Window is the name of the window and y% is the Y scroll bar offset, relative to work space origin.
SetSelect	Used to set or unset an icon.
Syntax:	SetSelect Window:Icon\$,NewState%
Where:	If NewState% is True the icon will be selected.

SetSprite Change a sprite in an icon.

Syntax: `SetSprite Window:Icon$,Sprite`

Where: *Window:Icon\$* is the name of the icon and *Sprite* is the name of the sprite.

Example: `SetSprite SaveAs:Sprite$, "file_FFF"`

Sgn(

Sin(

SizeOfFloat(

In WimpBasic the space needed to store a floating point value changes according to the precision chosen in the **Configure window**. Single precision (slightly less precision than in Basic) uses 4 bytes, Double precision (more precise than Basic and equal to Basic64) uses 8 bytes. If you store floating point values in memory using **StoreFloat** you can use the **SizeOfFloat()** function to read how many bytes are needed for the number.

Syntax: `BytesNeeded%=SizeOfFloat()`

Where: *BytesNeeded%* will be either 4 or 8.

Sound

SpritePutAt

Draw a sprite at a given coordinate. Output depends on the **Output** statement. See **Graphics** for details.

Syntax: `SpritePutAt SpriteName$,x%,y%`

Where: *SpriteName\$* is the name of the sprite and *x%,y%* are the bottom left coordinates for the sprite.

Note: With this command you can't use a sprite with the same name as your application. For example, if your application is called *!Test*, the sprite *!Test* is not available for use with **SpritePutAt** or **SpritePutAtScaled**. If you need to use the sprite you should copy it and rename it using !Paint. Changing the name in the filer is not sufficient.

SpritePutAtScaled

Draw a scaled sprite at a given coordinate. Output depends on the **Output** statement. See **Graphics** for details.

Syntax: `SpritePutAtScaled SpriteName$,x%,y%,Scalex%,Scaley%`

Where: *SpriteName\$* is the name of the sprite, *x%, y%* the coordinates for the bottom left and *Scalex%, Scaley%* the scale values described below. A scale value of 65536 (2^{16}) equals the sprite's original size. To halve the size use 32768 ($65536/2$) as the scale value, a higher value would enlarge it.

Note: With this command you can't use a sprite of the same name as your application. For example, if your application is called *!Test*, the sprite *!Test* is not available for use with **SpritePutAt** or **SpritePutAtScaled**. If you need to use the sprite you should copy it and rename it using !Paint. Changing the name in the filer is not sufficient.

Sqr(

Step

Stereo

Store

See **DrawAction** for details.

StoreByte

Stores a byte in memory.

Syntax: `StoreByte MemoryLocation%,value%`

Where: *MemoryLocation%* is the memory address and *value%* is the byte to be stored.

StoreFloat

Stores a floating point value in memory.

Syntax: `StoreFloat MemoryLocation%,value`

Where: *MemoryLocation%* is the memory address and *value* is the floating point value to be stored.

Warning: Float is not always the same size in WimpBasic. In single precision float, a variable takes 4 bytes but in double precision it takes 8 bytes. Use the **SizeOfFloat** function to read how many bytes are needed to store a floating point variable.

StoreInt	Stores an integer in memory.
Syntax:	StoreInt MemoryLocation%,value%
Where:	MemoryLocation% is the memory address and value% is the integer to be stored.
StoreString	Stores a string in memory.
Syntax:	StoreString MemoryLocation%,string\$
Where:	MemoryLocation% is the memory address and string\$ is the string to be stored.
Str\$(	There is no STR\$ in WimpBasic, use Hex\$ instead. The string returned is formatted according to the Format settings.
String\$(	
Sum(	
SumLen(	
Swap	
Sys	
Tab(	Same as in BBC Basic except that it is now affected by Output . See Graphics for details. Tab cannot be used in the print line but can be used on its own to position the cursor prior to using Print.
Tan(	
Tempo	
Then	
Time	
Time\$	
To	
True	
Until	
Update	Redraws a window without clearing the background. Similar to Refresh which does clear the background. See further explanation in Graphics .
Syntax:	Update
Val(	
Vdu	
Voice	
Voices	
Wait	
When	As the colon (:) cannot be used to separate keywords you must enter statements relating to When in the lines following the When statement. See Otherwise for details.
While	
WindowIndex	WindowIndex is used when you declare a window array using the form Window(n) when creating a window. It is used to find the index into the array for the window that should receive the event. For example, if you were opening windows yourself and had an array of windows this could be your open event handling procedure.
Example:	<pre> DefProcOpenWindows Rem Called when Wimp wants application to re-open Rem the window as a result of Rem Resize or moving of the window Rem MyWindow is an array of windows Open MyWindow(WindowIndex) EndProc </pre>

WindowSetExtent Changes window's size.

Syntax: `WindowSetExtent WindowName,x0%,y0%,x1%,y1%`

Where: *WindowName* is the name of the window;

x0% = new work area x minimum;

y0% = new work area y minimum;

x1% = new work area x maximum;

y1% = new work area y maximum;

WindowSetTitle Used to change the window title.

Syntax: `WindowSetTitle WindowName,NewTitle$`

Where: *WindowName* is the name of the window and *NewTitle\$* is the new window title.

WorkAreaXToScreen() Converts X coordinate from work area coordinate (coordinate inside a window) to screen coordinate.

Syntax: `ScreenX%=WorkAreaXToScreen(WindowName,WorkX%)`

Where: *WorkX%* is the work area coordinate; *WindowName* is the name of the window and *ScreenX%* is the screen coordinate.

WorkAreaYToScreen() Converts Y coordinate from work area coordinate (coordinate inside a window) to screen coordinate.

Syntax: `ScreenY%=WorkAreaYToScreen(WindowName,WorkY%)`

Where: *WorkY%* is the work area coordinate; *WindowName* is the name of the window and *ScreenY%* is the screen coordinate.

Appendix A

WimpBasic Compiler Errors

These errors may be encountered at Compile time ie when you compile your code and save the application. They should not be encountered when running your application.

Most errors are self explanatory and only the more obscure ones are explained in more detail. Explanations are shown in bold italic type.

Where the % character is shown followed by a letter this is replaced in the real error message by a more meaningful name. In general %s will be a string, %d will be a number and %f will be a floating point number.

%s is a function not a statement

%s is neither a function or statement

%s is not a function

%s is not valid procedure or function name

%s name should follow %s (ie. no space between)

%s should be a function not a procedure

%s should be a procedure not a function

. found without digits

The compiler found something like PRINT .

: is not valid in label

= is used to return values in function, this is a procedure

Binary number too long

A binary number can only have 32 digits in WimpBasic

Both parameters must be of same type ie. numeric or string.

Can't load menu file. Menu '%s' already exists

Can't store in array without dimension data

You have probably omitted a DIM

Can't swap an array with variable

SWAP requires either two variables or two arrays; not a mixture

Comma expected but found '%s' in proc/fn parameter list

Comma found when not expecting parameters

Compiler confused about data types. Try to rewrite line

Couldn't open file '

Couldn't open script file!

Couldn't write to file. Is there enough disk space?

E found when already found E in number

There can be only one E in a number. 1E3 is ok, but 1E45E4 is not

Else should be the only thing in the line when doing multi-line if's

Else without if

EndCase without Case

Endif should be the only thing in the line

Endif without if

EndProc found in function. Use = to exit functions.

EndProc or = should be last statements in procedure or function.

EndWhile without while

Exponent value out of range

You are using a number which is too big. Try double precision.

For,Next mismatch or incorrect variable name

Found ')' expected parameter.

Group '%s' doesn't exist

Using Fred:Bloggs% when group Fred doesn't exist will cause this.

Group '%s' is of unknown type

Group '%s' is read only

You can't modify the system, screen or printer group variables

Group name '%s' already exists, can't load windows

Hex number is too long, max 32 bits!

Illegal character %c

Illegal icon number

Illegal variable after Dim

Illegal variable name for Local '%s'.

Incorrect data type
Incorrect data type for Vdu
Incorrect local array definition
Incorrect number of parameters
Incorrect number of parameters to %s (%d found %d needed)
Incorrect number of parameters to %s (%d found %s needed)
Incorrect number of parameters to RECTANGLE
Incorrect parameter %d type, array variable needed
Incorrect parameter %d type, floating point array needed
Incorrect parameter %d type, floating point array variable needed
Incorrect parameter %d type, floating point variable needed
Incorrect parameter %d type, icon needed
Incorrect parameter %d type, integer array needed
Incorrect parameter %d type, integer array variable needed
Incorrect parameter %d type, integer variable needed
Incorrect parameter %d type, menu needed
Incorrect parameter %d type, numeric array variable needed
Incorrect parameter %d type, numeric expression or variable needed
Incorrect parameter %d type, numeric variable needed
Incorrect parameter %d type, numeric variable needed
Incorrect parameter %d type, string array needed
Incorrect parameter %d type, string array variable needed
Incorrect parameter %d type, string expression or variable needed
Incorrect parameter %d type, string or numeric needed
Incorrect parameter %d type, string variable needed
Incorrect parameter %d type, string variable needed
Incorrect parameter %d type, window name needed
Incorrect return type, numeric needed
Incorrect return type, string needed
Incorrect type for file handle

A file handle must be numeric

Incorrect type for parameter %d string needed
Incorrect type for parameter %d string variable needed
Incorrect type for parameter %d, numerical needed
Incorrect type for parameter 1
Incorrect type, numeric variable needed for parameter %d
Incorrect type, string needed for parameter %d
Invalid %s definition
Invalid array used as parameter
Invalid array variable in proc/fn definition
Invalid binary number

A binary number starts with % and may contain only 0 and or 1. %100400 is invalid

Invalid data following E

When specifying real numbers, E must be followed by a sensible numeric. a=1EXY is invalid

Invalid data for Data statement
Invalid data in file
Invalid expression for file handle numeric needed
Invalid hex number

Hex numbers start with & and may contain any of 0123456789ABCDefabcdef. A%=&AX56 is invalid

Invalid keyword %s in RECTANGLE
Invalid keyword found in proc/fn definition
Invalid label
Invalid label after %s
Invalid number of expressions after Case
Invalid number of expressions in Dim
Invalid number of expressions in window array
Invalid number of parameters after If
Invalid number of parameters after While only one needed
Invalid number of parameters for %s %s
Invalid parameter for Output must be a window or Screen
Invalid procedure/function read from library file
invalid return type for %s
Invalid stuff '%s' found after Sys

Invalid stuff after EndWhile
 Invalid stuff after Repeat
 Label '%s' defined but not used in line %d
 Label '%s' is already defined
 Label '%s' too long
 Line too long for compiler
 Load procedure '%s' doesn't exist.
 Local variable '%s' doesn't exist
 Menu %s doesn't exist
 Menu maker procedure '%s' used in menu '%s' doesn't exist.
 Menu selection procedure '%s' used in menu '%s' doesn't exist.
 Menu selection procedure '%s' used in menu '%s' for entry '%s' doesn't exist.
 Missing '=' after MID\$
 Missing () after variable name
 Missing) after Dim(function
 Missing) in proc/fn definition
 Missing ,
 Missing , after file handle
 Missing , and second parameter for Swap
 Missing =
 Missing = after %s
 Missing = in assignment
 Missing DefProc or DefFn in start of procedure %s
 Missing EndCase for Case statement in line %d
 Missing Endif for if at line %d
 Missing EndWhile for While statement in line %d
 Missing first parameter for Swap
 Missing keyword after %s
 Missing label
 Missing label '%s' for Goto in line %d
 Missing label after %s
 Missing Next for For statement in line %d
 Missing number after E
 For example D%=3E is invalid. The intention was probably either D%=&3E or D%=3E5
 Missing number after E in exponent format
 Missing Of after Case
 Missing operator. (For ex: %1009 instead of %100+9)
 Missing parenthesis after function %s
 Missing quote
 Missing second parameter for Swap
 Missing Then after If
 Missing To
 Missing Until for Repeat statement in line %d
 Missing variable after Dim(
 Missing variable name after group name
 Next without For
 Only DefProc and Rem are allowed before Local in code
 Only string parameters can use the ; in Bput#
 Otherwise already found for this Case statement
 Otherwise without Case
 parameter %d is of incorrect type for %s
 Parenthesis mismatch
 Parser error global variable without :
 Procedure name '%s%s' is too long.
 Procedure variable can't have : in it
 Procedure/function name too long in %s
 Quit function '%s' doesn't exist.
 Raw handler code '%s' doesn't exist.
 Return keyword is not allowed with array
 Select procedure '%s' doesn't exist.
 Sprite %s already exists
 Sprite %s doesn't exist

Startup procedure '%s' doesn't exist

String too long

String variables aren't allowed as control variables in For, Next

Stuff found after) in %s definition

Stuff found after SOUND ON/OFF

Swap, both arrays must be of same type

SWAP A%(),B%() is ok, but SWAP A%(),B\$() is not

SWI '%s' not found

If you refer to a SWI by name, the module providing that SWI must be present at compile time.

SYS number missing

There must be two parameters after TO

There should only be one parameter after =

Too many Case statements write code in more routines

Too many goto's and/or restore's

Too many label's

Too many Repeat/Until, While/Endwhile in one procedure

Type mismatch: Number needed

Type mismatch: String needed

Unrecognised stuff found after proc/fn definition

Until without Repeat

Value %f too high for selected precision.

Try changing to double precision.

Variable %s already exists

Variable %s doesn't exist

Variable %s:%s doesn't exist

When found after Otherwise, Code will never be reached

When without Case

Window %s doesn't exist

Window '%s' background button procedure '%s' doesn't exist.

Window '%s' event code '%s' doesn't exist.

Window '%s' icon %d button procedure '%s' doesn't exist.

Window can have either text or sprite in its title, not both

Window or menu %s doesn't exist

You can only negate (-value%) a valid numeric expression.

You can't Dim a icon array

You can't use -= with strings

You can't use an array as a parameter to a function or procedure

You can't use floating point number after

E in number

Only an integer can follow E. A=5E2 is ok, but A=5E2.5 is not

You can't use string variable as For control variable

You have reached maximum If level, divide routine into more procedures

You have reached maximum levels of Fors divide routine into subroutines

You have reached the line length limit. Please don't enter more characters in this line

Appendix B

WimpBasic RunTime Errors

These errors may be encountered when running your application.

Where the % character is shown followed by a letter this is replaced in the real error message by a more meaningful name. In general %s will be a string, %d will be a number and %f will be a floating point number.

Array hasn't been Dim'ed yet

You are trying to access an array before it is DIMed

Array subscript %d out of range

You are trying to access a non-existent array element. e.g

DIM A%(5)

A%(30)=5

Can't calculate power of strings

You can only take the power of a number

Can't Dim window variable. Window hasn't been opened yet

Variables in window groups don't exist until the window is opened.

Can't DIV strings

DIV only works on numbers

Can't divide strings

/ only works on numbers

Can't EOR strings

EOR only works on numbers

Can't multiply strings

** only works on numbers*

Can't pop data into string

You are attempting an invalid conversion

Can't pop string into float

You are attempting an invalid conversion

Can't pop string into float

You are attempting an invalid conversion

Can't pop string into integer

You are attempting an invalid conversion

Can't pop this data into string

You are attempting an invalid conversion

Can't pop window into float

You are attempting an invalid conversion

Can't pop window into float

You are attempting an invalid conversion

Can't pop window into integer

You are attempting an invalid conversion

Can't subtract strings

- only works on numbers

Cannot open file: %s

The specified file can't be found.

Compiler error: String given to Int

You are attempting an invalid conversion

Couldn't copy template

An error has occurred while loading. A file may be missing.

Couldn't find template '%s'

An error has occurred while loading. A file may be missing.

Couldn't load template

An error has occurred while loading. A file may be missing.

Data transfer failed: Receiver died

A failure has occurred while loading/saving from/to an application. It may not support inter application transfers

Floating point value out of range for integer

You are converting a real number which is too large. e.g.

A=1E23

A%=A

For control mismatch

A NEXT is missing. Usually this means the NEXT exists, but isn't reached. e.g.

FOR a%=1 to 5

IF a%>4 THEN b%=2 ELSE NEXT

Illegal value for Print

Incorrect data type found on stack

The wrong type of data has been found. Often caused by using READ to read a string into a numeric variable.

Incorrect type type found on stack

The wrong type of data has been found.

Incorrect number of parameters to procedure

The wrong number of parameters has been passed to a procedure

Incorrect number of subscripts

An array has been used with the wrong number of parameters. e.g.

DIM a%(4)

a%(4,5)=3

Incorrect number of subscripts to icon array

An icon array has been used with the wrong number of parameters.

Insufficient memory

You are out of memory

Insufficient memory for case control table

You are out of memory

Insufficient memory for display list

You are out of memory. Often caused by a large amount of data being printed to a window.

Insufficient memory for for control record

You are out of memory

Insufficient memory for loading file

You are out of memory

Insufficient memory for menu handler

You are out of memory

Insufficient memory for procedure control record

You are out of memory

Insufficient memory to create menu

You are out of memory

Internal error IP%x.%x

An internal error.

Internal error UP%x.%x

An internal error.

Invalid code in display list

This usually means an invalid VDU sequence has been used.

Invalid code in printer list

This usually means an invalid VDU sequence has been used.

Invalid data on stack

The wrong type of data has been found. Often caused by using READ to read a string into a numeric variable.

Invalid data type for BPut

Invalid data type for negate

The wrong type of variable has been used.

Invalid data type for operator 1

The wrong type of variable has been used.

Invalid data type for operator 2

The wrong type of variable has been used.

Invalid data type found in Next increment

The wrong type of variable has been used.

Invalid data type in When

The wrong type of variable has been used.

Invalid data type on stack

The wrong type of variable has been used.

Invalid icon array subscript

An array reference to an icon array is out of bounds.

Invalid immediate action in two_parameter_type

An internal error.

Invalid number of array dimensions

A silly number of dimensions has been requested.

Invalid number of parameters to code linker

An internal error.

Invalid number of subscripts to window array

You are using the wrong number of subscripts. e.g referring to a one dimensional array using 2 subscripts

Invalid size for array dimension

*You are using an invalid dimension. e.g.
DIM A%(-100)*

Invalid type for Next

NEXT must be followed by a number

Invalid type for step value

STEP must be followed by a number

Invalid type for Str\$

STR\$ only works on numbers

Invalid window array subscript

A subscript used is out of range

Invalid window number

A window is being referred to which doesn't exist

Invalid window number received

A window is being referred to which doesn't exist

Not enough memory for active windows -- increase wimplot

There is insufficient memory

Not enough memory for menu (1) -- increase wimplot

There is insufficient memory

Not enough memory for menu (2) -- increase wimplot

There is insufficient memory

Not enough memory for menu (3) -- increase wimplot

There is insufficient memory

Not enough memory for template load -- increase wimplot

There is insufficient memory

Not enough memory to change window title -- increase wimplot

There is insufficient memory

Not enough memory to load templates -- increase wimplot

There is insufficient memory

Parameters mismatch int or float and string given

The wrong sort of variable has been used

Parameters mismatch string and float given

The wrong sort of variable has been used

Parameters mismatch string and integer given

The wrong sort of variable has been used

Read data corrupt

A DATA statement has invalid data. The contents of DATA statements are only checked at run time

Read failed for %s

READ was unable to read the data.

Resume without error!

RESUME was found outside an error handler. It is only valid in an error routine.

ReTry without error!

RESUME was found outside an error handler. It is only valid in an error routine.

Run time library internal error 1, code is &%x
An internal error.

Stack underflow!
An internal error.

String too long
A string was generated which is > 255 characters

Template '%s' not found
Loading the application failed. Check all files are present.

Template file not found.
Loading the application failed. Check all files are present.

Template load failed
Loading the application failed. Check all files are present.

The window you are trying to access hasn't been created yet
A window does not exist until it is open.

The window you are trying to access hasn't been created yet, Use Open statement
A window does not exist until it is open.

This is not a arrayed window
You tried to use a subscript with a non-array window.

Unimplement c.mymath normal_action code
An internal error.

Unknown type found in DefProc
An invalid DEFPROC was found

Window title can't be set window hasn't been created
A window does not exist until it is open.

You can only compare string to string in When
An invalid comparison was found in a WHEN statement

You can't compare string to float in When
An invalid comparison was found in a WHEN statement

You can't compare string to integer in When
An invalid comparison was found in a WHEN statement

Here are just a few of the other products you will be able to
download from RISCOS.com free of charge
(You can make a donation to help support future product releases but this is entirely voluntary)



This Documentation is © 3QD Developments Ltd 2026
Some of the products shown above might not be available immediately.
Not all products will run on all versions of RISC OS, check the product information page for details.

Other Products list last updated 29/2/26